ECE656-Machine Learning and Adaptive Systems
Lectures 25 & 26

M.R. Azimi, Professor

Department of Electrical and Computer Engineering
Colorado State University

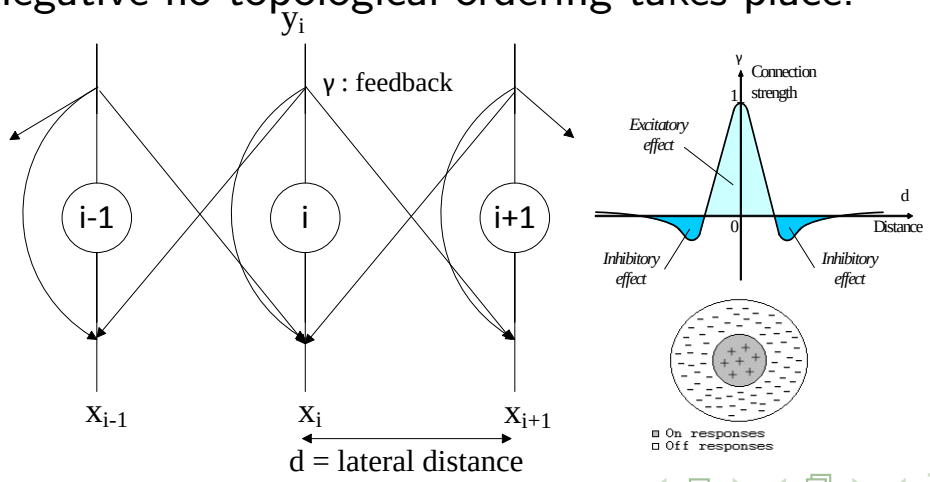
Fall 2015

Self-Organizing Maps (SOM)-Kohonen 1990

Internal representations of information in brain are organized spatially i.e. individual areas exhibit logical ordering of their functionalities, e.g. speech, vision, hearing, and motor functions.

Biological neural networks form of 2-D layers of neurons densely interconnected by lateral feedback synapses. The strength of these connections vary as a function of lateral distance. Neighborhood of 50-100 microns radius is characterized by short-range *excitatory* connections while 200-500 microns ring are longer-range (weaker) *inhibitory* connections. The feedback connection weights are shaped like *Mexican hat* as shown. The self-feedback is positive.

The response of planar array of neurons to excitation form an "activity bubble" with a center which is the center of excitation. Incoming positive feedback widens the radius of activity bubble whereas negative feedback narrows the bubble sharper. However, if the net feedback is too negative no topological ordering takes place.



SOM Lateral Interactions

Goal is to introduce machine learning algorithms that can:

- Learn mapping from high dimensional pattern space to low dimensional feature space through "self-organization" (unsupervised),
- Provide useful topological properties like many biological neural systems.

If we work with sampled Mexican Hat, an approximation for lateral connection function used in SOM is

$y_i(k+1) = f(x_i(k+1) + \sum_{m=-m_0}^{m_0} \gamma_m \; y_{i+m}(k))$

where $x_i(k)$ and $y_i(k)$ are the input and output of neuron i at time k and γ_m is the feedback coefficient for sampled Mexican hat. The second term show the effect of all the lateral connections within the excitatory region of $m \in [-m_0, m_0]$.

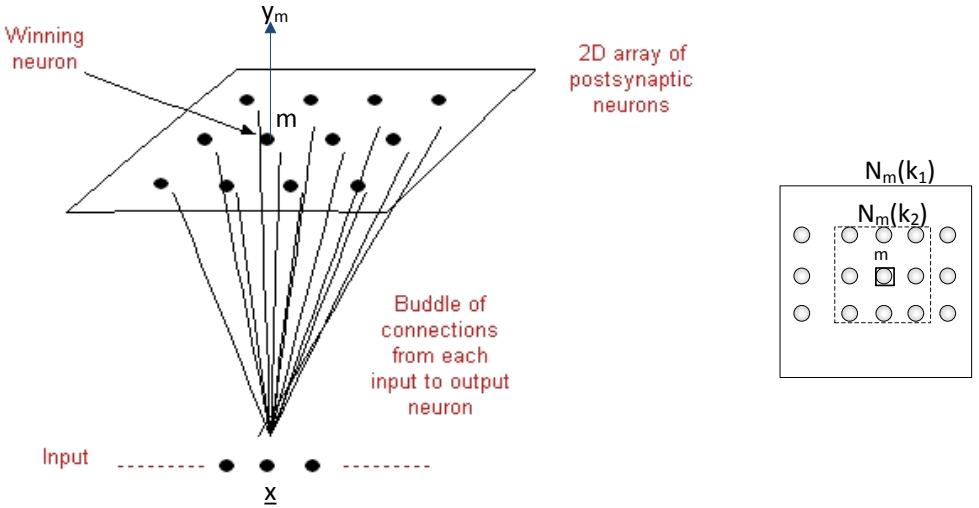
Consider a 2-D array of neurons connected to the input vector $\underline{x}$ through weight vector $\underline{w}_i$ for neuron i as shown. To build topologically ordered maps, correlated (cooperation among a topologically related subset) learning by spatially neighboring cells can be implemented by lateral feedback connections and interactions. Lateral interaction in artificial maps can be imposed by defining *Neighborhood Set* N_m centered around a cell m (winning cell) as shown.

At each learning step k, all cells within N_m are updated, while those outside N_m are left unchanged. The winner cell m is selected using

$$||\underline{x} - \underline{w}_m|| = \min_i \; ||\underline{x} - \underline{w}_i||, \qquad \forall i \in N_m.$$

i.e. self-organization requires *Cooperation* and *Competition* at the same time.

SOM Lateral Interactions-Cont.



The radius of N_m in time-varying. Typically, $N_m(k_1) > N_m(k_2) > \cdots > N_m(k_l)$ for $k_1 < k_2 < \cdots < k_l$. Use very wide radius at the beginning and shrink monotonically with time. Toward the end of learning neighborhood consists of only center winning cell i.e. process reduces to simple competitive Kohonen learning for cluster learning. Note that the mechanism of lateral interaction in the Mexican Hat is embedded in the definition of localized neighborhood.

SOM Unsupervised Training

Involves the following steps:

1. Start with many neurons i.e. large number of neurons and large N_m. Note that large N_m means higher competition i.e. more positive feedback.
2. Choose the winner cell m (Competition) as before.
3. Apply the winner-take-all updating rule to all neurons (Cooperation) within N_m.

SOM Training

Then, update the weights using,

$$\underline{w}_i(k+1) = \underline{w}_i(k) + \alpha(k) h_{i,m}(k)[\underline{x}_k - \underline{w}_i(k)], \text{ for } i \in N_m(k)$$

$$\underline{w}_j(k+1) = \underline{w}_j(k), \text{ for } j \notin N_m(k)$$

where $0 < \alpha(k) < 1$ and $h_{i,m}(k)$ must be symmetric and decreasing function of lateral distance $d_{i,m}$ between the winner neuron m (center) and other neurons $i \in N_m$. A suitable choice is a Gaussian function which is translation invariant (i.e. independent of the location of the winning neuron),

$$h_{i,m}(k) = exp(-\frac{d_{i,m}^2}{2\sigma^2(k)})$$

where $d_{i,m}^2 = ||\underline{r}_i - \underline{r}_m||^2$. The width parameter $\sigma(k)$ measures the degree to which excited neurons in the vicinity of the winning neuron participate in the learning process. Since the neighborhood size N_m should shrink with time we use,

$$\sigma(k) = \sigma_0 exp(-k/\tau)$$

with τ being the time constant. Typically, we use $\tau = \frac{1000}{log\ \sigma_0}$.

Thus, this learning moves the weight vector of the winning neuron (and rest of neurons in $N_m(k)$) closer to the input vector $\underline{x}_k$. The rest of the neurons only get a fraction of the correction though.

SOM Training-Cont.

Conditions for Self-Organization of Feature Maps:

- Cells are exposed to a sufficient number of inputs.
- Only weights leading to an excited neighborhood of the map are affected.
- The size of the topological neighborhood should decreases monotonically with increasing number of iterations.
- Adjustment is propositional to activation received by each cell within neighborhood.

SOM leads to a planar neuron map with weights coding the PDF of the pattern vectors used for training. That is, the neurons transform inputs into an encoded distribution that represents the computed values of parameters by sites of maximum relative activity within the map.

Practical Considerations:

1. **Choice of neighborhood N_m:**
   - Neighborhood topology can be square, hexagonal, etc.
   - Size of N_m is very large (50% of diameter of the network) at the beginning and then gradually shrinks as learning progresses as mentioned before. Towards the end $N_m = 1$ i.e. only the winner neuron.
   - If N_m is too small to start with, the map won't be ordered globally. Rather various mosaic-like parcellations of the map are seen.
   - Clearly, $h_{i,m}(k)$ accomplishes both the shrinkage of neighborhood size as well as decrease in weight updates depending on lateral distance from the winner cell.

SOM Training-Cont.

2. **Number of Iterations:**
If number of neurons is M a good rule of thumb is to use at least $500M$ epochs to train. Note: this is not dependent on input dimensional N.

3. **Choice of $\alpha(k)$:**
For first 1000 steps choose $\alpha(k) \approx 1$ e.g. 0.995 thereafter use a monotonically decreasing function. Examples are:

Linear: $\alpha(k) = \alpha_0(1 - \frac{k-k_p}{k_q})$, $\alpha_0 = 0.9$

k_p is epoch at which decay starts and k_q is time constant.

Exponential: $\alpha(k) = \alpha_0 e^{(-\frac{k-k_p}{k_q})}$

Gaussian: $\alpha(k) = \alpha_0 e^{-\frac{\|\underline{r}_i - \underline{r}_m\|^2}{2\sigma^2(k)}}$

where $\underline{r}_i$, $\underline{r}_m$, and and $\sigma^2(k)$ were defined before.

4. **Variant of SOM Updating Rule:**
Choose the winner using the similarity matching as before. Then use *normalized* learning rule:

$$\underline{w}_i(k+1) = \frac{\underline{w}_i(k) + \alpha'(k) h_{i,m}(k) \underline{x}_k}{\|\underline{w}_i(k) + \alpha'(k) h_{i,m}(k) \underline{x}_k\|}, \text{ for } i \in N_m(k)$$

$$\underline{w}_j(k+1) = \underline{w}_j(k), \text{ for } j \notin N_m(k)$$

where $0 < \alpha'(k) < 1$ e.g. $\alpha'(k) = 100/k$.

SOM- Key Properties

1. **Approximation of the Input Space:** The feature map, represented by the set of synaptic weight vectors, provides a good approximation to the input space.
2. **Topological Ordering:** The feature map generated by the SOM is topologically ordered in the sense that the spatial location of a neuron in the lattice corresponds to a particular domain or feature of the input patterns.
3. **Density Matching:** The feature map reflects variations in the statistics of the input distribution: regions in the input space from which sample vectors $\underline{x}$ are drawn with a high probability of occurrence are mapped onto larger domains of the output space, and therefore with better resolution than regions in input space from which sample vectors $\underline{x}$ are drawn with a low probability of occurrence.
4. **Feature Selection:** Given data from an input space with an arbitrary distribution, the self-organizing map is able to select a set of best features for approximating the underlying distribution.
5. **Typical Applications:** Include sensory mapping, vector quantization and data compression, speech recognition, robot control.

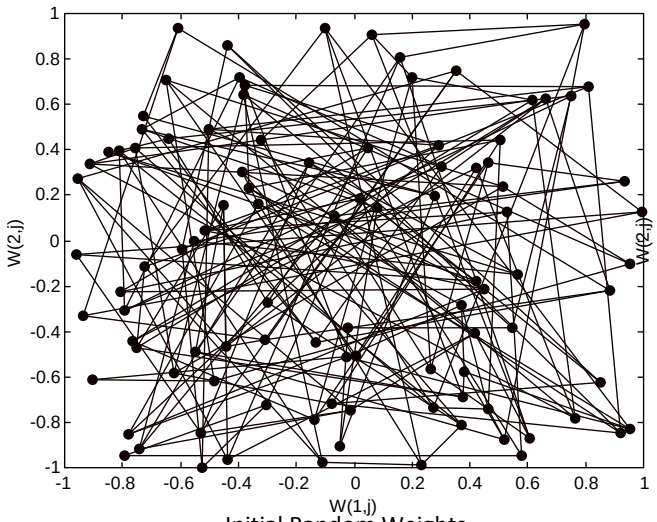
SOM-Example

To illustrate competitive learning, consider the Kohonen network with 100 neurons arranged in the form of a 2-D lattice with 10 rows and 10 columns. The network is required to classify 2-D input vectors: each neuron in the network should respond only to the input vectors occurring in its region.

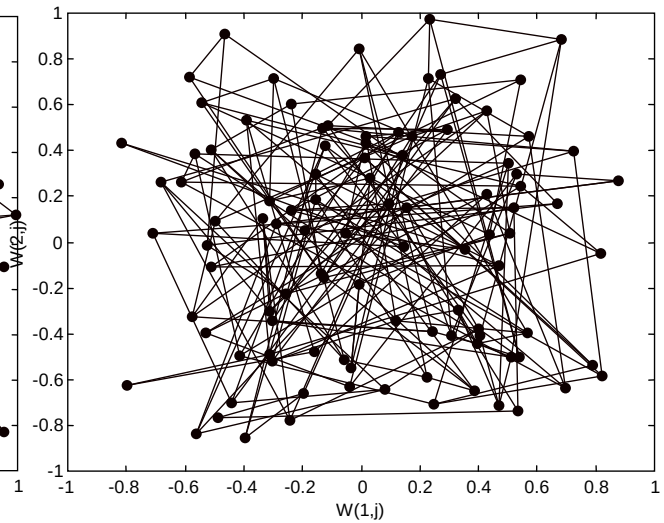
The network is trained with 1000 2-D input vectors, $\underline{x}_k = [x_{1k} x_{2k}]^t$, whose elements were generated randomly in a square region in the interval between -1 and +1, i.e. uniformly distributed between -1 and +1. The learning rate parameter was $\alpha = 0.1$.

The following figures show the stages of the learning until convergence when an ordered map is generated. At the end, the statistical distribution of the cells approaches that of the input vectors except for some distortion. Also, see examples on pages of 445-450 in your textbook.

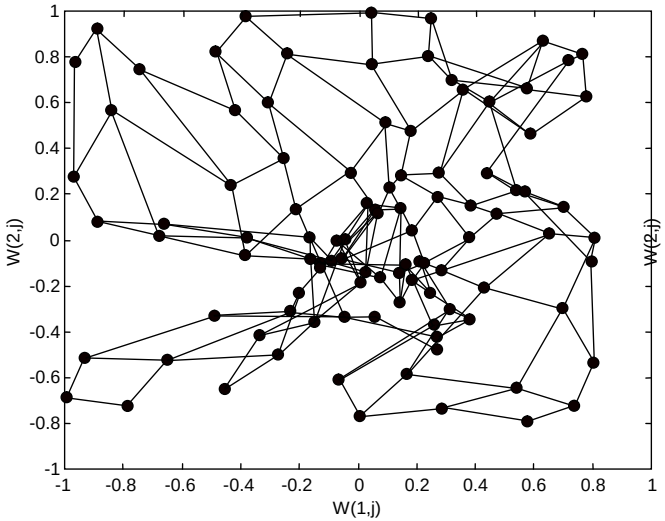
SOM-Example



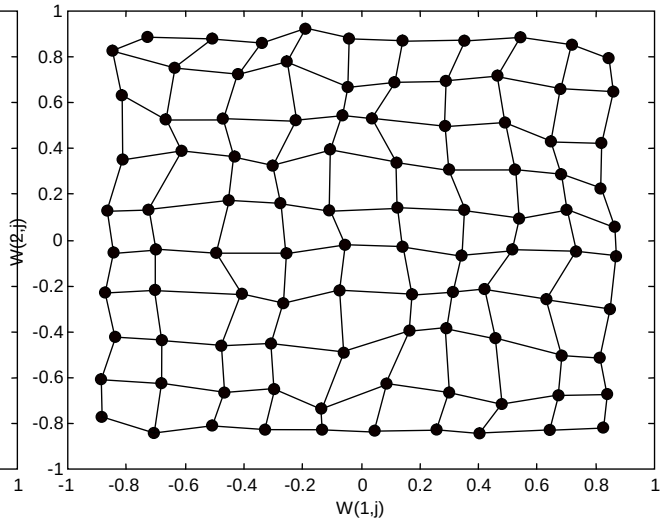
Initial Random Weights



Network After 100 Iterations



Network After 1000 Iterations



Network After 10000 Iterations

Fine-Tuning of SOM by LVQ Methods (Kohonen 1997)

To use SOM as a pattern classifier, weight vectors (clusters) must be fine-tuned using a supervised learning known as *Learning Vector Quantization (LVQ)*. Thus, LVQ is applied to find an optimal placement of $\underline{w}_i$ by an iterative learning process which starts by presenting patterns of known class labels (supervised) and assigning cells to different classes by majority voting.

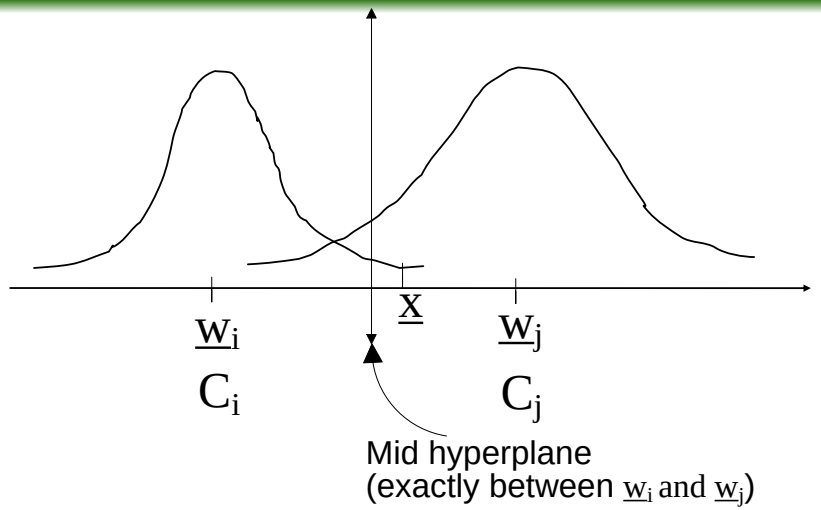
Kohonen proposed three different types of LVQ algorithms.

LVQ1 Algorithm:

It is assumed that several codebook vectors, $\underline{w}_i$s, have been established using unsupervised SOM (or classical VQ). The purpose of *LVQ1* is to pull the codebook (weight) vectors away (misclassifications) from the decision surface to make class boundaries more accurate for classification. The updating rule is based upon the principle of *Reward and Punishment*. Steps are:

- Apply SOM to the unlabeled data $\{\underline{x}_p\}_{p=1}^{P}$ to form the clusters and associated codebook vectors $\underline{w}_i,\ i \in [1, M]$.
- Use a limited labeled data set $\mathcal{X} = \{\underline{x}_p, d_p\}_{p=1}^{P'}$ with $P' \ll P$ to label and fine-tune the clusters in SOM. This is done by applying $\underline{x}_k \in C_i$ from set $\mathcal{X}$.

LVQ1 Algorithm



- If $\underline{x}_k$ is correctly classified i.e. $\underline{w}_i$ for cluster i is the nearest codebook vector (i.e. winner) then move them closer (attract or reward), i.e.

$$\underline{w}_i(k+1) = \underline{w}_i(k) + \beta(k)(\underline{x}_k - \underline{w}_i(k))$$
$$\underline{w}_j(k+1) = \underline{w}_j(k), \quad j \neq i$$

  If $\underline{x}_k$ is incorrectly classified i.e. $\underline{w}_l$ is the nearest codebook vector then move them apart (repel or punishment)

$$\underline{w}_l(k+1) = \underline{w}_l(k) - \beta(k)(\underline{x}_k - \underline{w}_l(k))$$
$$\underline{w}_j(k+1) = \underline{w}_j(k), \quad j \neq l$$

Start with small $\beta(0) \sim 0.01 - 0.03$ and slowly decay to zero over time (e.g., over 10000 iterations).

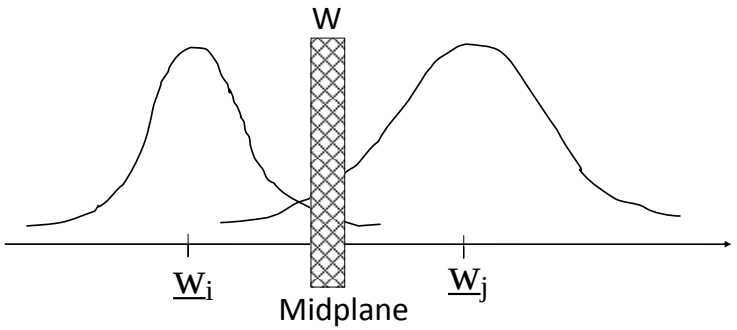
After training, the placement of $\underline{w}_i$s will be such that classification using nearest neighbor criterion closely resembles that of a Bayes classifier.

LVQ2 Algorithm

LVQ2 Algorithm:

LVQ2 better complies with Bayes decision boundary than LVQ1. Instead of dealing with just the closest neighbor to an input vector $\underline{x}_k$, this algorithm modifies the two closest codebook vectors, $\underline{w}_i$ and $\underline{w}_j$.

Assume the two closest codebook vectors $\underline{w}_i$ and $\underline{w}_j$, to input pattern $\underline{x}_k$ belong to two different classes. The (incorrect) decision boundary between these two codebook vectors is always in the midplane of $\underline{w}_i$ and $\underline{w}_j$. Now, define a symmetric window of width W around the midplane and make corrections to both $\underline{w}_i$ and $\underline{w}_j$ when $\underline{x}_k$ falls into the window, on wrong side of the midplane. Steps are



- Apply $\underline{x}_k \in C_i$ from set $\mathcal{X}$ and choose two nearest codebook vectors $\underline{w}_i$ and $\underline{w}_j$.
- If $\underline{x}_k$ is closer to $\underline{w}_j$ and falls in W (on the wrong side) then,

$$\underline{w}_i(k+1) = \underline{w}_i(k) + \beta(k)(\underline{x}_k - \underline{w}_i(k))$$
$$\underline{w}_j(k+1) = \underline{w}_j(k) - \beta(k)(\underline{x}_k - \underline{w}_j(k))$$

For all other $l \neq i$ or j, we have

$$\underline{w}_l(k+1) = \underline{w}_l(k)$$

LVQ2 Algorithm-Cont.

- If the network correctly classifies $\underline{x}_k \in C_i$, only the weight vector $\underline{w}_i$ is moved toward the input vector.

Thus, the effect of learning is to move midplane moves toward intersection of the two distributions, hence minimum classification error.

Remarks:

1. Vector $\underline{x}_k$ is assumed to be in W if

$$min\left\{\frac{d_i}{d_j}, \frac{d_j}{d_i}\right\} > s$$

where $d_i = \|\underline{w}_i - \underline{x}_k\|$, $d_j = \|\underline{w}_j - \underline{x}_k\|$ and $s = \frac{1-W}{1+W}$.

2. In general, width of W depends on the number of training samples. For large number of samples, define a narrower window. For good statistical accuracy, a sufficient number of samples should fall into W.
3. Stopping criterion: Codebook vectors have stabilized or maximum number of epochs has been reached.

Background

Associative memory is one of the primary functions of the human brain, e.g., hearing only a few bars of music may recall a complete sensory experience including scenes and sounds. Unlike computer memory which is Location Addressable Memory (LAM) an associative memory is Content Addressable Memory (CAM) and has abilities:

- To store many associated (stimulus-response) pattern pairs. Storage is done through change in neurons connectivity (long-term memory).
- To accomplish this storage in a distributed (spatially), robust manner over a large densely interconnected network.
- To recall or retrieve the correct response pattern even though the input stimulus patterns may be distorted or incomplete (i.e. error correction). Recall or retrieval is done via state propagation through time (short-term memory).
- To add to existing memory.

During recall, system carries out a parallel search within the stored patterns to output the pattern which closely matches the stimulus pattern and retrieves it either fully or partially (no memory addressing). Deviation from this occurs when e.g., (a) overlapping of memory may bring about less faithful regeneration of the output patterns; or (b) too much distortion can cause incorrect recall, etc.

Basic Definitions

1. Hamming Space:

Space of N-D vectors $\underline{x} \in H^N$, with elements $x_i \pm 1$, i.e.

$H^N = \{\underline{x}, x_i \in (\pm 1)\}$

2. Hamming Distance:

Hamming distance (HD) which is measure of dissimilarity of vectors is defined as an integer equal to number of bits positions differing between two binary vectors of same size, i.e.

$HD(\underline{x}, \underline{y}) = \frac{1}{2} \sum_{i=1}^{N} |x_i - y_i|, \quad \underline{x}, \underline{y} \in H^N$

$HD_{max}(\underline{x}, \underline{y}) = N$ when $\underline{y} = \underline{x}^c$ i.e. complement of $\underline{x}$

It can be shown that, $ED(\underline{x}, \underline{y}) = \|\underline{x} - \underline{y}\| = 2\sqrt{HD(\underline{x}, \underline{y})}$

Types of Associative Memories

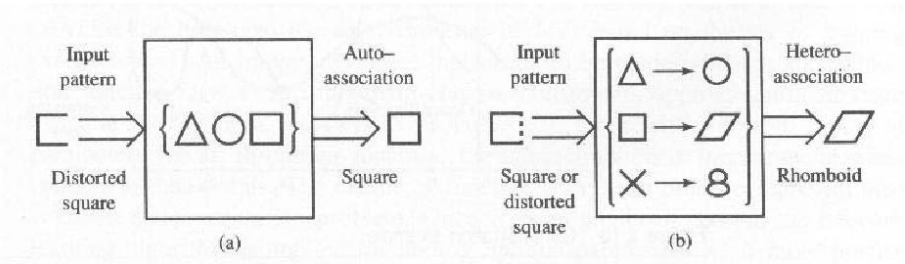
Given a set of K pairs of exemplar patterns $\{\underline{x}_i, \underline{y}_i\}_{i=1}^{K}$, $\underline{x}_i, \underline{y}_i \in H^N$, there can be three types of associative memories.

1. **Hetero-Associative Learning:** Implements a mapping $\Psi(.)$ of $\underline{y}_i = \Psi(\underline{x}_i)$ such that if $\underline{x}$ is any unknown vector close to $\underline{x}_k$, i.e.

   $HD(\underline{x}_k, \underline{x}) < HD(\underline{x}_i, \underline{x}), \forall i \in [1, K], i \neq k$

Basic Definitions-Cont.

then, $\Psi(\underline{x}) = \underline{y}_k$ i.e. it retrieves or recalls the corresponding pattern associated with $\underline{x}_k$.

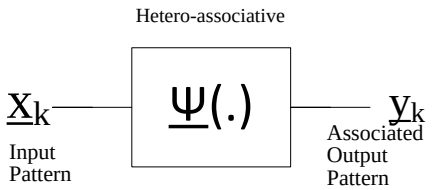


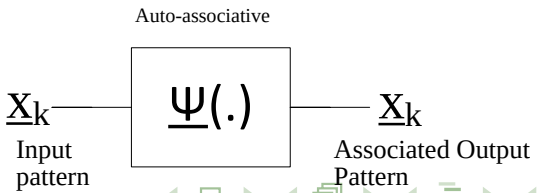
2. **Interpolative Associative Memory:** Implements a mapping $\Psi(.)$ of $\underline{y}_i = \Psi(\underline{x}_i)$ such that if $\underline{x}$ is a distorted version of $\underline{x}_k$, i.e. $\underline{x} = \underline{x}_k + \underline{d}_k$ then the output of the mapping also differs from the corresponding pattern by $\underline{e}_k$ i.e. ,

   If $\underline{x} = \underline{x}_k + \underline{d}_k \Rightarrow \underline{y} = \Psi(\underline{x}_k + \underline{d}_k) = \underline{y}_k + \underline{e}_k$.

3. **Auto-Associative Memory:** Here $\underline{y}_i = \underline{x}_i$ i.e. a mapping $\Psi(.)$ of $\underline{x}_i = \Psi(\underline{x}_i)$ such that if $\underline{x}$ is closer in HD to $\underline{x}_k$ than any other $\underline{x}_i$ i.e.
   $HD(\underline{x}_k, \underline{x}) < HD(\underline{x}_i, \underline{x}), \forall i \in [1, K], i \neq k$

   then $\Rightarrow \Psi(\underline{x}) = \underline{x}_k$





Linear Associator- Review

Recall from Lecture 6 that using the Hebbian learning rule for a network with M cells,

$$W(k+1) = W(k) + \mu \underline{y}_k \underline{x}_k^t$$

where $W(k)$ is the $M \times N$ weight matrix, a perfect linear associator can be built for data set $\{\underline{x}_k, \underline{y}_k\}_{k=0}^{K-1}$ as long as $\underline{x}_k$'s are orthonormal.

It was shown that if $W(0) = 0$ and $\mu = 1$, then

$W^* = \sum_{k=0}^{K-1} \underline{y}_k \underline{x}_k^t$: Storage Phase

Then, the recall phase involves applying $\underline{x}_l$ and retrieving corresponding pattern $\underline{y}_l$,

$$W^* \underline{x}_l = (\sum_{k=0}^{K-1} \underline{y}_k \underline{x}_k^t) \underline{x}_l = \underline{y}_l.$$

Now, if the input vector is different than one of the exemplar i.e. $\underline{x} = \underline{x}_l + \underline{d}_l$ then,

$W^* \underline{x} = (\sum_{k=0}^{K-1} \underline{y}_k \underline{x}_k^t (\underline{x}_l + \underline{d}_l)) = \underline{y}_l + \underline{e}_l$: Not perfect recall.

where $\underline{e}_l = (\sum_{k=0}^{K-1} \underline{y}_k \underline{x}_k^t) \underline{d}_l$.

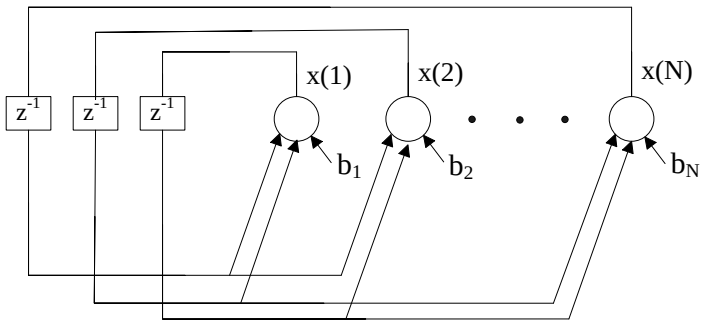
Remark

Note that when used for autoassociator $W^* = \sum_{k=0}^{K-1} \underline{x}_k \underline{x}_k^t$ is the correlation matrix. In this case Capacity of the system is limited to $K \leq N$ (maximum rank of correlation matrix).

Recurrent Associative Memory (Binary Hopfield Network)

The error in association can be suppressed by thresholding (nonlinear mapping) of the output and by feeding the output back to input. This can be performed by a recurrent network as shown. There are three basic goals:

- Given some initial state, the network should always converge to closest "stable state" or "attractor";
- Have as many stable states as possible $0.4N$, where N is the number of neurons.



In this network output of each neuron is connected to inputs of all other neurons (no self-feedback). Each neuron is a McCulloch-Pitts neuron. The weights $w_{i,j}$s (long-term memory) are established during the storage phase while x_is (short-term memory) or state variables evolve over time.

The input to neuron i comes from two sources: external input b_i and inputs from other neurons i.e. $net_i(t) = \sum_{j=1_{j\neq i}}^{N} w_{i,j}x_t(j) + b_i$ (t is discrete).

Recurrent Associative Memory (Binary Hopfield Network)

Output of the selected (currently undergoing update) neuron at iteration $t+1$ is

$$x_{t+1}(i) = \begin{cases} 1, & net_i(t) > T_i \\ x_t(i), & net_i(t) = T_i \\ -1, & net_i(t) < T_i \end{cases}$$

where T_i is the threshold for cell i.

Rules of Retrieval:

- Neurons are selected and updated asynchronously one at a time (only one cell updates at a time).
- Uniform random selection and updating is adopted.
- Neurons must be updated at same average rate.

Given the update rule, we would like to investigate the stability of the system. More specifically, we would like to know:

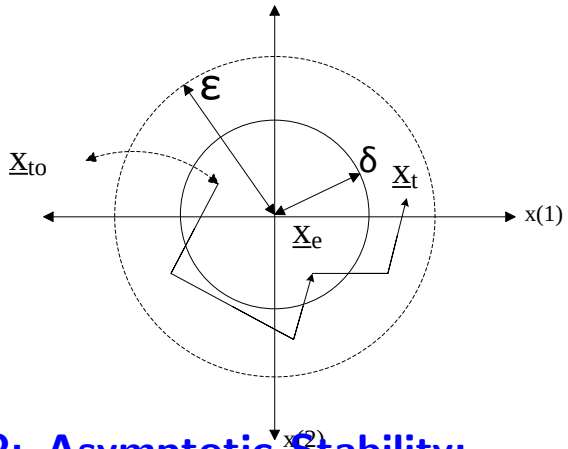
1. Given an unknown input pattern (initial condition) does the memory evolve toward an equilibrium state (attractor) corresponding to a stored pattern? (Global Stability)
2. How many equilibrium states (attractors) exist? How do we establish them ? How are they reached?
3. Does the system approach to a local attractor if it is displaced from it by small perturbation? (Local Stability)

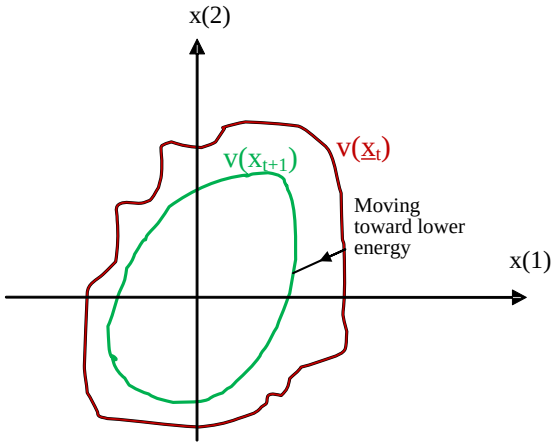
These questions relate to Lyapunov stability theory.

Lyapunov Stability of Nonlinear Systems- Review

Def1: Stability (zero-input):

Equilibrium state $\underline{x}_e$ is "stable" if for any given $\varepsilon > 0$, there exists $\delta(\varepsilon, t_o) > 0$ where t_o is initial time such that if $\|\underline{x}_{t_o} - \underline{x}_e\| < \delta$ then $\|\underline{x}_t - \underline{x}_e\| < \varepsilon$, $\forall t > t_o$.





Def2: Asymptotic Stability:

Equilibrium state $\underline{x}_e$ is "Asymptotically stable" iff

$$\|\underline{x}_{t_o} - \underline{x}_e\| < \delta \Longrightarrow \lim_{t \to \infty} \|\underline{x}_t - \underline{x}_e\| \to 0.$$

Def3: Lyapunov Function:

Scalar function, $v(\underline{x}_t)$ is a Lyapunov function for an autonomous nonlinear state equation $\underline{x}_{t+1} = \underline{f}(\underline{x}_t, 0, t)$ with $\underline{f}(0) = 0$ if,

(a) v is continuous in $\underline{x}$ and $v(\underline{0}) = 0$.
(b) v is positive definite, i.e. $v(\underline{x}) > 0$ for $\underline{x} \neq \underline{0}$.
(c) $\Delta v(\underline{x}) = v(\underline{x}_{t+1}) - v(\underline{x}_t) \leq 0$ or $-\Delta v(\underline{x})$ is PD.

Lyapunov Stability of Nonlinear Systems- Review

Theorem-Global Stability:

Consider nonlinear state equation $\underline{x}_{t+1} = \underline{f}(\underline{x}_t)$ with $\underline{f}(\underline{0}) = \underline{0}$. Then $\underline{x}_t = \underline{x}_e$ is stable, if there exists a Lyapunov function that satisfies (a)-(c) and further,

$$\lim_{\|\underline{x}\| \to \infty} v(\underline{x}) \to \infty.$$

This is sufficient condition for stability but not necessary.

Example:

Consider state equation
$x_{t+1}(1) = -1.5x_t(1)$
$x_{t+1}(2) = -0.5x_t(2)$

Choose $v(\underline{x}) = x_t^2(1) + x_t^2(2)$
$\Delta v(\underline{x}) = v(\underline{x}_{t+1}) - v(\underline{x}_t) = 1.25x_t^2(1) - 0.75x_t^2(2).$
Since $\Delta v(\underline{x})$ is indefinite in sign, the test fails.

For our problem, we choose,

$$E = -\tfrac{1}{2}\textstyle\sum_i \sum_{j \neq j} \sum_j w_{i,j}x(i)x(j) - \sum_i b_i x(i) + \sum_i T_i x(i)$$

$$= -\tfrac{1}{2}\underline{x}^t W \underline{x} - \textstyle\sum_i b_i x(i) + \sum_i T_i x(i)$$

Note that E is PD since diagonal elements of W are zero $w_{i,i} = 0$ and $w_{i,j} = w_{j,i}$.

Global Stability of Binary Hopfield Network

This this function always decreases whenever the state of any neuron is updated. To see this form ΔE_{x_l} due to change in state of neuron l i.e. $\Delta x(l)$,

$$\Delta E_{x(l)} = (-\sum_{j_{j\neq i}} w_{l,j}x(j) - b_l + T_l)\Delta x(l) = -(net_l - T_l)\Delta x(l)$$

where $\Delta x_l = x^{new}(l) - x^{old}(l)$

Case 1: If there is no transition then $x^{old}(l) = x^{new}(l)$ and $\Delta x(l) = 0$, $\Delta E_{x(l)} = 0$.

Case 2: If there is a transition $x^{old}(l) = 1$ & $net_l < T_l$, $x^{new}(l) = -1$, $\Delta x(l) = -2$, and hence $\Delta E_{x(l)} < 0$.

Case 3: If $x^{old}(l) = -1$ & $net_l > T_l$, $x^{new}(l) = 1$, $\Delta x(l) = 2$, then $\Delta E_{x(l)} < 0$

i.e. always $\Delta E_{x(i)} \leq 0$ i.e. globally stable in the Lyapunov sense. Thus,

- State transitions lead to stable/equilibrium states.
- Every stable state lie on local minimum of E located at vertices of N-D [-1,1] cube.

To examine local stability, we need to know how information is stored. This is done next.

Local Stability of Binary Hopfield Network

Information Storage Process:

To store stable states (patterns) $\underline{x}_k, k \in [1, K]$ use (auto-association)

$$W = \sum_{k=1}^{K} \underline{x}_k \underline{x}_k^t - KI.$$

Note that the elimination of self-feedback makes $\underline{x}^t W \underline{x} < 0$ i.e. ND. For individual weights

$$w_{i,j} = (1 - \delta_{i,j}) \sum_{k=1}^{K} x_k(i) x_k(j)$$

where

$$\delta_{i,j} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

$$x_k(i) \in [-1, 1] : i^{th} \text{ element of } \underline{x}_k$$

For unipolar case i.e $x_k(i) \in [0, 1]$,

$$w_{i,j} = (1 - \delta_{i,j}) \sum_{k=1}^{K} (2x_k(i) - 1)(2x_k(j) - 1)$$

Local Stability:

Let us assume that the network at equilibrium state (one of stable states) $\underline{x}_m$ is perturbed, i.e. pattern $\tilde{\underline{x}}_m = \underline{x}_m + \underline{e}_m$ which is a slightly perturbed version of $\underline{x}_m$ is applied to the network.

If i^{th} neuron is selected for updating then,

$$net_i = \sum_{j_j \neq i} w_{i,j} \tilde{x}_m(j) = \sum_{k=1}^{K} x_k(i) \sum_{j_j \neq i} x_k(j) \tilde{x}_m(j)$$

Local Stability of Binary Hopfield Network-Cont.

Now, we use the important property that for $\underline{x}, \underline{y} \in H^N$,

$\sum_j x(j)y(j) = \underline{x}^t \underline{y} = N - 2HD(\underline{x}, \underline{y})$

Proof:

$HD(\underline{x}, \underline{y}) = \frac{1}{4}[ED(\underline{x}, \underline{y})]^2 = \frac{1}{4}\sum_{j=1}^{N}(x(j) - y(j))^2 =$
$\frac{1}{4}\sum_{j=1}^{N} x^2(j) + y^2(j) - 2x(j)y(j)$

$= \frac{1}{4}[2N - 2\sum_{j=1}^{N} x(j)y(j)]$

Applying this property yields,

$$net_i = \sum_{k=1}^{K} x_k(i)(N - 2HD(\underline{x}_k, \tilde{x}_m))$$

Two Scenarios can happen:

1. If $\tilde{\underline{x}}_m$ is different than $\underline{x}_k$ (i.e. $m \neq k$) then $\underline{x}_k^t \tilde{\underline{x}}_m = (N - 2HD(\underline{x}_k, \tilde{\underline{x}}_m)) \approx 0$.
2. If $\underline{x}_k \approx \tilde{\underline{x}}_m$ (i.e. $m = k$) then $(N - 2HD(\underline{x}_k, \tilde{\underline{x}}_m)) \approx N$.

   Thus, we can replace $(N - 2HD(\underline{x}_k, \tilde{\underline{x}}_m)) = N\ \delta_{k,m}$, which gives

   $$net_i \approx N x_m(i)$$

   If $x_m^{old}(i) = \pm 1 \Rightarrow net_i \approx \pm N \Rightarrow x_m^{new}(i) = \pm 1$

   i.e. no change in state occurs, hence locally stable.