CS407 Neural Computation

Lecture 7:
Neural Networks Based on Competition.

Lecturer: A/Prof. M. Bennamoun
Neural Networks Based on Competition.

- Introduction
- Fixed weight competitive nets
  - Maxnet
  - Mexican Hat
  - Hamming Net
- Kohonen Self-Organizing Maps (SOM)
- SOM in Matlab
- References and suggested reading
Introduction…Competitive nets

- The most extreme form of competition among a group of neurons is called “Winner Take All”.
- As the name suggests, only one neuron in the competing group will have a nonzero output signal when the competition is completed.
- A specific competitive net that performs Winner Take All (WTA) competition is the Maxnet.
- A more general form of competition, the “Mexican Hat” will also be described in this lecture (instead of a non-zero o/p for the winner and zeros for all other competing nodes, we have a bubble around the winner).
- All of the other nets we discuss in this lecture use WTA competition as part of their operation.
- With the exception of the fixed-weight competitive nets (namely Maxnet, Mexican Hat, and Hamming net) all of the other nets combine competition with some form of learning to adjust the weights of the net (i.e. the weights that are not part of any interconnections in the competitive layer).
Introduction…Competitive nets

- The form of learning depends on the purpose for which the net is being trained:
  - LVQ and counterpropagation net are trained to perform mappings. The learning in this case is supervised.
  - SOM (used for clustering of input data): a common use of unsupervised learning.
  - ART are also clustering nets: also unsupervised.
- Several of the nets discussed use the same learning algorithm known as “Kohonen learning”: where the units that update their weights do so by forming a new weight vector that is a linear combination of the old weight vector and the current input vector.
- Typically, the unit whose weight vector was closest to the input vector is allowed to learn.
Introduction...Competitive nets

- The weight update for output (or cluster) unit \( j \) is given as:

\[
\begin{align*}
\mathbf{w}_j(\text{new}) &= \mathbf{w}_j(\text{old}) + \alpha \left[ \mathbf{x} - \mathbf{w}_j(\text{old}) \right] \\
&= \alpha \mathbf{x} + (1 - \alpha) \mathbf{w}_j(\text{old})
\end{align*}
\]

- where \( \mathbf{x} \) is the input vector, \( \mathbf{w}_j \) is the weight vector for unit \( j \), and \( \alpha \) the learning rate, decreases as learning proceeds.

- Two methods of determining the closest weight vector to a pattern vector are commonly used for self-organizing nets.

- Both are based on the assumption that the weight vector for each cluster (output) unit serves as an exemplar for the input vectors that have been assigned to that unit during learning.

  - The first method of determining the winner uses the squared Euclidean distance between the I/P vector
Introduction...Competitive nets

and the weight vector and chooses the unit whose weight vector has the smallest Euclidean distance from the I/P vector.

– The second method uses the dot product of the I/P vector and the weight vector. The dot product can be interpreted as giving the correlation between the I/P and weight vector.

- In general and for consistency, we will use Euclidean distance squared.
- Many NNs use the idea of competition among neurons to enhance the contrast in activations of the neurons
- In the most extreme situation, the case of the Winner-Take-All, only the neuron with the largest activation is allowed to remain “on”.
Introduction...Competitive nets

1. Euclidean distance between two vectors

\[ \|x - x_i\| = \sqrt{(x - x_i)'(x - x_i)} \]

2. Cosine of the angle between two vectors

\[ \cos \psi = \frac{x^t x_i}{\|x\| \|x_i\|} \]

\[ \psi_1 < \psi_T < \psi_2 \]
Neural Networks Based on Competition.

- Introduction
- Fixed weight competitive nets
  - Maxnet
  - Mexican Hat
  - Hamming Net
- Kohonen Self-Organizing Maps (SOM)
- SOM in Matlab
- References and suggested reading
MAXNET

$$f(\text{net}) = \begin{cases} 0, & \text{net < 0} \\ \text{net}, & \text{net \geq 0} \end{cases}$$

$$W_M = \begin{bmatrix} 1 & -\varepsilon & \ddots & -\varepsilon \\ -\varepsilon & 1 & \ddots & -\varepsilon \\ \vdots & \ddots & \ddots & \ddots \\ -\varepsilon & -\varepsilon & \ddots & 1 \end{bmatrix}$$

$$0 < \varepsilon < \frac{1}{p}$$

$$y^{k+1} = \Gamma \left[ W_M \ y^k \right]$$
Hamming Network and MAXNET

- **MAXNET**
  - A recurrent network involving both excitatory and inhibitory connections
    - Positive self-feedbacks and negative cross-feedbacks
  - After a number of recurrences, the only non-zero node will be the one with the largest initializing entry from i/p vector
Maxnet  Fixed-weight competitive net

- **Maxnet** Lateral inhibition between competitors

![Diagram of Maxnet network]

### Competition:
- iterative process until the net stabilizes (at most one node with positive activation)

- **Step 0:**
  - Initialize activations and weight (set $0 < \varepsilon < 1 / m$, ) where $m$ is the # of nodes (competitors)

**Activation function for the Maxnet:**

\[
f(x) = \begin{cases} 
  x & \text{if } x > 0 \\ 
  0 & \text{otherwise}
\end{cases}
\]
Algorithm

Step 1: While stopping condition is false do steps 2-4

- **Step 2**: Update the activation of each node:

  \[ a_{j}(\text{new}) = f \left[ a_{j}(\text{old}) - \varepsilon \sum_{k \neq j} a_{k}(\text{old}) \right] \text{ for } j = 1, \ldots, m \]

- **Step 3**: Save activation for use in next iteration:

  \[ a_{j}(\text{old}) = a_{j}(\text{new}) \text{ for } j = 1, \ldots, m \]

- **Step 4**: Test stopping condition:

  If more than one node has a nonzero activation, continue; otherwise, stop.
Example

- **Note** that in step 2, the i/p to the function $f$ is the total i/p to node $A_j$ from all nodes, including itself.
- Some precautions should be incorporated to handle the situation in which two or more units have the same, maximal, input.
- Example:

$$\varepsilon = 0.2 \text{ and the initial activations (input signals) are:}$$

\[
\begin{align*}
    a_1(0) &= 0.2, \quad a_2(0) = 0.4 \quad a_3(0) = 0.4 \quad a_4(0) = 0.8 \\
\end{align*}
\]

As the net iterates, the activations are:

\[
\begin{align*}
    a_1(1) &= f(a_1(0) - 0.2 [a_2(0) + a_3(0) + a_4(0)]) = f(-0.12) = 0 \\
    a_1(1) &= 0.0, \quad a_2(1) = 0.08 \quad a_3(1) = 0.32 \quad a_4(1) = 0.56 \\
    a_1(2) &= 0.0, \quad a_2(2) = 0.0 \quad a_3(2) = 0.192 \quad a_4(2) = 0.48 \\
    a_1(3) &= 0.0, \quad a_2(3) = 0.0 \quad a_3(3) = 0.096 \quad a_4(3) = 0.442 \\
    a_1(4) &= 0.0, \quad a_2(4) = 0.0 \quad a_3(4) = 0.008 \quad a_4(4) = 0.422 \\
    a_1(5) &= 0.0, \quad a_2(5) = 0.0 \quad a_3(5) = 0.0 \quad a_4(5) = 0.421 \\
\end{align*}
\]
Neural Networks Based on Competition.

- Introduction
- Fixed weight competitive nets
  - Maxnet
  - Mexican Hat
  - Hamming Net
- Kohonen Self-Organizing Maps (SOM)
- SOM in Matlab
- References and suggested reading
Mexican Hat Fixed-weight competitive net

Architecture

- Each neuron is connected with excitatory links (positively weighted) to a number of “cooperative neighbors” neurons that are in close proximity.
- Each neuron is also connected with inhibitory links (with negative weights) to a number of “competitive neighbors” neurons that are somewhat further away.
- There may also be a number of neurons, further away still, to which the neurons is not connected.
- All of these connections are within a particular layer of a neural net.
- The neurons receive an external signal in addition to these interconnections signals (just like Maxnet).
- This pattern of interconnections is repeated for each neuron in the layer.
- The interconnection pattern for unit $X_i$ is as follows:
The size of the region of cooperation (positive connections) and the region of competition (negative connections) may vary, as may vary the relative magnitudes of the +ve and –ve weights and the topology of the regions (linear, rectangular, hexagonal, etc..)
Mexican Hat  Fixed-weight competitive net

- The **contrast enhancement** of the signal \( s_i \) received by unit \( X_i \) is accomplished by iteration for several time steps.
- The activation of unit \( X_i \) at time \( t \) is given by:

\[
x_i(t) = f \left[ s_i(t) + \sum_k w_k x_{i+k}(t-1) \right]
\]

- Where the terms in the summation are the **weighted signal** from other units cooperative and competitive neighbors) at the previous time step.
Mexican Hat  Fixed-weight competitive net

Nomenclature

\( R_2 = \) Radius of region of interconnections; \( X_i \) is connected to units \( X_{i+k} \) and \( X_{i-k} \) for \( k=1, \ldots R_2 \)

\( R_1 = \) Radius of region with +ve reinforcement; \( R_1 < R_2 \)

\( w_k = \) weights on interconnections between \( X_i \) and units \( X_{i+k} \) and \( X_{i-k} \):
- \( w_k \) is positive for \( 0 \leq k \leq R_1 \)
- \( w_k \) is negative for \( R_1 < k \leq R_2 \)

\( x = \) vector of activations

\( x_{\text{old}} = \) vector of activations at previous time step

\( t_{\text{max}} = \) total number of iterations of contrast enhancement

\( s = \) external signal
Mexican Hat  Fixed-weight competitive net

Algorithm

- As presented, the algorithm corresponds to the external signal being given only for the first iteration (step 1).
- Step 0 Initialize parameters $t_{\text{max}}, R_1, R_2$ as desired
  Initialize weights:
  \[
  w_{ij} = \begin{cases} 
  C_1 > 0 & \text{for } k = 0, \ldots, R_1 \\
  C_2 < 0 & \text{for } k = R_1 + 1, \ldots, R_2
  \end{cases}
  \]

  Initialize $x_{\text{old}}$ to 0

- Step 1 Present external signal $s$:
  \[
  x = s
  \]

  Save activation in array $x_{\text{old}}$ (for $i = 1, \ldots, n$):
  \[
  x_{\text{old}}[i] = x_i
  \]

  Set iteration counter: $t = 1$
Mexican Hat  Fixed-weight competitive net

Algorithm

Step 2  While $t$ is less than $t\_max$, do steps 3-7

Step 3  Compute net input ($i = 1, \ldots n$)

$$x_i = C_1 \sum_{k=-R_1}^{R_1} x\_old_{i+k} + C_2 \sum_{k=-R_2}^{-R_1+1} x\_old_{i+k} + C_2 \sum_{k=-R_1+1}^{R_2} x\_old_{i+k}$$

Step 4  Apply activation function $f$ (ramp from 0 to $x\_max$, slope 1):

$$f(x) = \begin{cases} 
0 & \text{if } x < 0 \\
x & \text{if } 0 \leq x \leq \text{max} \\
x\_\text{max} & \text{if } x > \text{max} 
\end{cases}$$

$$x_i = \min(x\_\text{max}, \max(0, x_i)) \quad i = 1, \ldots, n$$
Mexican Hat  Fixed-weight competitive net

Step 5  Save current activations in \( x_{\text{old}} \):
\[
x_{\text{old}}_i = x_i \quad i = 1, \ldots, n
\]

Step 6  Increment iteration counter:
\[
t = t + 1
\]

Step 7  Test stopping condition:
If \( t < t_{\text{max}} \), continue; otherwise, stop.

Note:
The +ve reinforcement from nearby units and –ve reinforcement from units that are further away have the effect of

– increasing the activation of units with larger initial activations,
– reducing the activations of those that had a smaller external signal.
Let’s consider a simple net with 7 units. The activation function for this net is:

\[
    f(x) = \begin{cases} 
        0 & \text{if } x < 0 \\
        x & \text{if } 0 \leq x \leq 2 \\
        2 & \text{if } x > 2 
    \end{cases}
\]

- **Step 0** Initialize parameters \( R_1 = 1, R_2 = 2, C_1 = .6, C_2 = -.4 \)
- **Step 1** \( t=0 \) The external signal \( s = (.0, .5, .8, 1, .8, .5, .0) \)
  \[x(0) = (0.0, 0.5, 0.8, 1.0, 0.8, 0.5, 0.0)\]
  Save in \( x_{\text{old}} \):
  \[x_{\text{old}}(1) = (0.0, 0.5, 0.8, 1.0, 0.8, 0.5, 0.0)\]
- **Step 2** \( t=1 \), the update formulas used in step 3, are listed as follows for reference:
Mexican Hat

Example

\[ x_1 = 0.6 \ _{old_1} + 0.6 \ _{old_2} - 0.4 \ _{old_3} \]
\[ x_2 = 0.6 \ _{old_1} + 0.6 \ _{old_2} + 0.6 \ _{old_3} - 0.4 \ _{old_4} \]
\[ x_3 = -0.4 \ _{old_1} + 0.6 \ _{old_2} + 0.6 \ _{old_3} + 0.6 \ _{old_4} - 0.4 \ _{old_5} \]
\[ x_4 = -0.4 \ _{old_2} + 0.6 \ _{old_3} + 0.6 \ _{old_4} + 0.6 \ _{old_5} - 0.4 \ _{old_6} \]
\[ x_5 = -0.4 \ _{old_3} + 0.6 \ _{old_4} + 0.6 \ _{old_5} + 0.6 \ _{old_6} - 0.4 \ _{old_7} \]
\[ x_6 = -0.4 \ _{old_4} + 0.6 \ _{old_5} + 0.6 \ _{old_6} + 0.6 \ _{old_7} \]
\[ x_7 = -0.4 \ _{old_5} + 0.6 \ _{old_6} + 0.6 \ _{old_7} \]

Step 3 \( t=1 \)

\[ x_1 = 0.6 \ (0.0) + 0.6 \ (0.5) - 0.4 \ (0.8) = -0.2 \]
\[ x_2 = 0.6 \ (0.0) + 0.6 \ (0.5) + 0.6 \ (0.8) - 0.4 \ (1.0) = 0.38 \]
\[ x_3 = -0.4 \ (0.0) + 0.6 \ (0.5) + 0.6 \ (0.8) + 0.6 \ (1.0) - 0.4 \ (0.8) = 1.06 \]
\[ x_4 = -0.4 \ (0.5) + 0.6 \ (0.8) + 0.6 \ (1.0) + 0.6 \ (0.8) - 0.4 \ (0.5) = 1.16 \]
\[ x_5 = -0.4 \ (0.8) + 0.6 \ (1.0) + 0.6 \ (0.8) + 0.6 \ (0.5) - 0.4 \ (0.0) = 1.06 \]
\[ x_6 = -0.4 \ (1.0) + 0.6 \ (0.8) + 0.6 \ (0.5) + 0.6 \ (0.0) = 0.38 \]
\[ x_7 = -0.4 \ (0.8) + 0.6 \ (0.5) + 0.6 \ (0.0) = -0.2 \]
Mexican Hat

Example

Step 4: \( \mathbf{x} = (0.0, \ 0.38, \ 1.06, \ 1.16, \ 1.06, \ 0.38, \ 0.0) \)

Step 5-7 Bookkeeping for next iteration

Step 3: \( t=2 \)

\[
\begin{align*}
x_1 &= 0.6 (0.0) + 0.6 (0.38) - 0.4 (1.06) = -0.196 \\
x_2 &= 0.6 (0.0) + 0.6 (0.38) + 0.6 (1.06) - 0.4 (1.16) = 0.39 \\
x_3 &= - 0.4 (0.0) + 0.6 (0.38) + 0.6 (1.06) + 0.6 (1.16) - 0.4 (1.06) = 1.14 \\
x_4 &= - 0.4 (0.38) + 0.6 (1.06) + 0.6 (1.16) + 0.6 (1.06) - 0.4 (0.38) = 1.66 \\
x_5 &= - 0.4 (1.06) + 0.6 (1.16) + 0.6 (1.06) + 0.6 (0.38) - 0.4 (0.0) = 1.14 \\
x_6 &= - 0.4 (1.16) + 0.6 (1.06) + 0.6 (0.38) + 0.6 (0.0) = 0.39 \\
x_7 &= - 0.4 (1.06) + 0.6 (0.38) + 0.6 (0.0) = -0.196
\end{align*}
\]

Step 4: \( \mathbf{x} = (0.0, \ 0.39, \ 1.14, \ 1.66, \ 1.14, \ 0.39, \ 0.0) \)

Step 5-7 Bookkeeping for next iteration

The node Which has been enhanced
Neural Networks Based on Competition.

- Introduction
- Fixed weight competitive nets
  - Maxnet
  - Mexican Hat
  - Hamming Net
- Kohonen Self-Organizing Maps (SOM)
- SOM in Matlab
- References and suggested reading
Hamming Network  

- **Hamming distance** of two vectors is the number of components in which the vector differ.
- For 2 bipolar vectors, \( x \) and \( y \) of dimension \( n \),
  \[ x \cdot y = a - d \]
  where: \( a \) is number of bits in agreement in \( x \) and \( y \)
  \( d \) is number of bits different in \( x \) and \( y \)
  \[ d = n - a \]
  Hamming distance = \( HD(x, y) \)
  \[ x \cdot y = 2a - n \]
  \[ a = 0.5(x \cdot y + n) \]
  larger \( x \cdot y \) \( \Rightarrow \) larger \( a \) \( \Rightarrow \) shorter Hamming distance

- The Hamming net uses Maxnet as a subnet to find the unit with the largest input (see figure on next slide).

\[
\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}
\]
Hamming Network Fixed-weight competitive net
Architecture

The sample architecture shown assumes input vectors are 4-tuples, to be categorized as belonging to one of 2 classes.
Hamming Network  
Fixed-weight competitive net

Algorithm

Given a set of $m$ bipolar exemplar vectors $e(1), e(2), \ldots, e(m)$, the Hamming net can be used to find the exemplar that is closest to the bipolar i/p vector $x$.

The net i/p $y_{in}$ to unit $Y_j$ gives the number of components in which the i/p vector and the exemplar vector for unit $Y_j e(j)$ agree ($n$ minus the Hamming distance between these vectors).

Nomenclature:

$n$  
number of input nodes, number of components of any i/p vector

$m$  
number of o/p nodes, number of exemplar vectors

$e(j)$  
the $j^{th}$ exemplar vector:

$$e(j) = (e_1(j), \ldots, e_i(j), \ldots, e_n(j))$$
Hamming Network   Fixed-weight competitive net
Algorithm

Step 0   To store the $m$ exemplar vectors, initialise the weights and biases:

$$w_{ij} = 0.5e_i(j), \quad b_j = 0.5n$$

Step 1   For each vector $x$, do steps 2-4

Step 2   Compute the net input to each unit $Y_j$:

$$y_{\_in\_j} = b_j + \sum_i x_i w_{ij} \quad \text{for } j = 1, \cdots, m$$

Step 3   Initialise activations for Maxnet

$$y_j(0) = y_{\_in\_j} \quad \text{for } j = 1, \cdots, m$$

Step 4   Maxnet iterates to find the best match exemplar.
Hamming Network  

- Upper layer: Maxnet
  - it takes the $y_{in}$ as its initial value, then $Y_j$ iterates toward stable state (the best match exemplar)
  - one output node with highest $y_{in}$ will be the winner because its weight vector is closest to the input vector
Hamming Network  

Fixed-weight competitive net

Example (see architecture n=4)

Given the exemplar vectors:

\[ e(1) = [1, -1, -1, -1] \]
\[ e(2) = [-1, -1, -1, -1] \]

\( m \) in this case = 2 (exemplar patterns)

The Hamming net can be used to find the exemplar that is closest to each of the bipolar i/p patterns, (1, 1, -1, -1), (1, -1, -1, -1), (-1, -1, -1, 1), and (-1, -1, 1, 1)

Step 1: Store the \( m \) exemplar vectors in the weights:

\[
W = \begin{bmatrix}
0.5 & -0.5 \\
-0.5 & -0.5 \\
-0.5 & -0.5 \\
-0.5 & -0.5
\end{bmatrix}
\]

where \( w_{ij} = \frac{e_i(j)}{2} \)

Initialize the biases:

\( b_1 = b_2 = 2 = n / 2 \)

\( n = \) number of input nodes = 4 (in this case)
Hamming Network  

Fixed-weight competitive net

Example

For the vector \( \mathbf{x} = (1, 1, -1, -1) \), let’s do the following steps:

\[
y_{in_1} = b_1 + \sum_i x_i w_{i1} = 2 + 1 = 3
\]

\[
y_{in_2} = b_2 + \sum_i x_i w_{i2} = 2 - 1 = 1
\]

These values represent the Hamming similarity because \((1, 1, -1, -1)\) agrees with \( \mathbf{e}(1) = (1, -1, -1, -1) \) in the 1\(^{st}\), 3\(^{rd}\), and 4\(^{th}\) components and because \((1,1, -1, -1)\) agrees with \( \mathbf{e}(2) = (-1, -1, -1, 1) \) in only the 3\(^{rd}\) component.

\[
y_1(0) = 3
\]

\[
y_2(0) = 1
\]

Since \( y_1(0) > y_2(0) \), Maxnet will find that unit \( Y_1 \) has the best match exemplar for input vector \( \mathbf{x} = (1, 1, -1, -1) \).
**Hamming Network**

Fixed-weight competitive net

**Example**

For the vector \( \mathbf{x} = (1, -1, -1, -1) \), let’s do the following steps:

\[
\begin{align*}
y_{\_ \_in1} &= b_1 + \sum_i x_i w_{i1} = 2 + 2 = 4 \\
y_{\_ \_in2} &= b_2 + \sum_i x_i w_{i2} = 2 + 0 = 2
\end{align*}
\]

Note that the input agrees with \( \mathbf{e}(1) \) in all 4 components and agrees with \( \mathbf{e}(2) \) in the 2\(^{nd}\) and 3\(^{rd}\) components.

\[
\begin{align*}
y_1(0) &= 4 \\
y_2(0) &= 2
\end{align*}
\]

Since \( y_1(0) > y_2(0) \), Maxnet will find that unit \( Y_1 \) has the best match exemplar for input vector \( \mathbf{x} = (1, -1, -1, -1) \).
Hamming Network

Fixed-weight competitive net

Example

For the vector $\mathbf{x} = (-1, -1, -1, 1)$, let’s do the following steps:

$$y_{\text{in}_1} = b_1 + \sum_i x_i w_{i1} = 2 + 0 = 2$$

$$y_{\text{in}_2} = b_2 + \sum_i x_i w_{i2} = 2 + 2 = 4$$

Note that the input agrees with $\mathbf{e}(1)$ in the $2^{\text{nd}}$ and $3^{\text{rd}}$ components and agrees with $\mathbf{e}(2)$ in all four components

$$y_1(0) = 2$$
$$y_2(0) = 4$$

Since $y_2(0) > y_1(0)$, Maxnet will find that unit $Y_2$ has the best match exemplar for input vector $\mathbf{x} = (-1, -1, -1, 1)$. 
Example

For the vector $\mathbf{x} = (-1, -1, 1, 1)$, let’s do the following steps:

\[
y_{in1} = b_1 + \sum_i x_i w_{i1} = 2 - 1 = 1
\]
\[
y_{in2} = b_2 + \sum_i x_i w_{i2} = 2 + 1 = 3
\]

Note that the input agrees with $\mathbf{e}(1)$ in the 2\textsuperscript{nd} component and agrees with $\mathbf{e}(2)$ in the 1\textsuperscript{st}, 2\textsuperscript{nd}, and 4\textsuperscript{th} components

\[
y_1(0) = 1
\]
\[
y_2(0) = 3
\]

Since $y_2(0) > y_1(0)$, Maxnet will find that unit $Y_2$ has the best match exemplar for input vector $\mathbf{x} = (-1, -1, 1, 1)$. 
Neural Networks Based on Competition.

- Introduction
- Fixed weight competitive nets
  - Maxnet
  - Mexican Hat
  - Hamming Net
- Kohonen Self-Organizing Maps (SOM)
- SOM in Matlab
- References and suggested reading
Kohonen Self-Organizing Maps (SOM)

- Teuvo Kohonen, born: 11-07-34, Finland
- SOM = “Topology preserving maps”
- SOM assume a topological structure among the cluster units
- This property is observed in the brain, but not found in other ANNs
Kohonen Self-Organizing Maps (SOM)

Figure 7.10 Pattern structure: (a) natural similarity and (b) no natural similarity.

Figure 7.12 Mapping features of input $x$ into a rectangular array of neurons: (a) general diagram and (b) desirable response peaks for Figure 7.10(b).
Kohonen Self-Organizing Maps (SOM)

Architecture:

Figure 4.5 Kohonen self-organizing map.
Kohonen Self-Organizing Maps (SOM)

Architecture:

* * * {*} (* [*] *) *
{ } R=2 ( ) R=1 [ ] R=0

* * * * * * *

Linear array of cluster units

This figure represents the neighborhood of the unit designated by # in a 1D topology (with 10 cluster units).

Neighbourhoods for rectangular grid (Each unit has 8 neighbours)
Kohonen Self-Organizing Maps (SOM)

Architecture:

```
 R=2     -------
 R=1     ----
 R=0     ------------
```

**Neighbourhoods for hexagonal grid** (Each unit has 6 neighbours).

In each of these illustrations, the “winning unit” is indicated by the symbol # and the other units are denoted by *

**NOTE:** What if the winning unit is on the edge of the grid?

- Winning units that are close to the edge of the grid will have some neighbourhoods that have fewer units than those shown in the respective figures.
- Neighbourhoods do not “wrap around” from one side of the grid to the other; “missing” units are simply ignored.
Training SOM Networks

- Which nodes are trained?
- How are they trained?
  - Training functions
  - Are all nodes trained the same amount?
- Two basic types of training sessions?
  - Initial formation
  - Final convergence
- How do you know when to stop?
  - Small number of changes in input mappings
  - Max # epochs reached
Kohonen Self-Organizing Maps (SOM)

**Algorithm:**

**Step 0.** Initialise weights $w_{ij}$ (randomly)
- Set topological neighbourhood parameters
- Set learning rate parameters

**Step 1.** While stopping condition is false, do steps 2-8.

**Step 2.** For each i/p vector $x$, do steps 3-5

**Step 3.** For each $j$, compute:

$$D(J) = \sum_{i} (w_{ij} - x_i)^2$$

**Step 4.** Find index $J$ such that $D(J)$ is a minimum

**Step 5.** For all units $j$ within a specified neighbourhood of $J$, and for all $i$:

$$w_{ij}^{(new)} = w_{ij}^{(old)} + \alpha [x_i - w_{ij}^{(old)}]$$

**Step 6.** Update learning rate.

**Step 7.** Reduce radius of topological neighbourhood at specified time

**Step 8.** Test stopping condition
Kohonen Self-Organizing Maps (SOM)

Alternative structure for reducing R and $\alpha$:

- The learning rate $\alpha$ is a slowly decreasing function of time (or training epochs). Kohonen indicates that a \textit{linearly} decreasing function is satisfactory for practical computations; a \textit{geometric} decrease would produce similar results.

- The radius of the neighbourhood around a cluster unit also decreases as the clustering process progresses.

- The formation of a map occurs in 2 phases:
  - The initial formation of the correct order and
  - the final convergence

The second phase takes much longer than the first and requires a small value for the learning rate.

Many iterations through the training set may be necessary, at least in some applications.
Kohonen Self-Organizing Maps (SOM)

Applications and Examples:

- The SOM have had many applications:
  - Computer generated music (Kohonen 1989)
  - Traveling salesman problem

- Examples: A Kohonen SOM to cluster 4 vectors:
  - Let the 4 vectors to be clustered be:
    
    \[
    \begin{align*}
    s(1) &= (1, 1, 0, 0) & s(2) &= (0, 0, 0, 1) \\
    s(3) &= (1, 0, 0, 0) & s(4) &= (0, 0, 1, 1)
    \end{align*}
    \]
  
  - The maximum number of clusters to be formed is \(m=2\) (4 input nodes, 2 output nodes)
  
  Suppose the learning rate is \(\alpha(0)=0.6\) and \(\alpha(t+1) = 0.5 \alpha(t)\)

- With only 2 clusters available, the neighborhood of node \(J\) (step 4) is set so that only one cluster updates its weights at each step (i.e. \(R=0\))
Kohonen Self-Organizing Maps (SOM)

Examples:

- **Step 0** Initial weight matrix

\[
W = \begin{bmatrix}
.2 & .8 \\
.6 & .4 \\
.5 & .7 \\
.9 & .3
\end{bmatrix}
\]

- Initial radius: \( R = 0 \)
- Initial learning rate: \( \alpha(0) = 0.6 \)

- **Step 1** Begin training

- **Step 2** For \( s(1) = (1, 1, 0, 0) \), do steps 3-5

- **Step 3**

\[
D(1) = (.2 - 1)^2 + (.6 - 1)^2 + (.5 - 0)^2 + (.9 - 0)^2 = 1.86
\]

\[
D(2) = (.8 - 1)^2 + (.4 - 1)^2 + (.7 - 0)^2 + (.3 - 0)^2 = 0.98
\]
Kohonen Self-Organizing Maps (SOM)

Examples:

- **Step 4** The i/p vector is closest to o/p node 2, so \( J=2 \)
- **Step 5** The weights on the winning unit are updated

\[
 w_{i2}(new) = w_{i2}(old) + 0.6[x_i - w_{i2}(old)] + 0.4w_{i2}(old) + 0.6x_i
\]

This gives the weight matrix:

\[
 W = \begin{bmatrix}
 0.2 & 0.92 \\
 0.6 & 0.76 \\
 0.5 & 0.28 \\
 0.9 & 0.12 \\
\end{bmatrix}
\]

This set of weights has not been modified.
Kohonen Self-Organizing Maps (SOM)

Examples:

- **Step 2** For \( s(2) = (0, 0, 0, 1) \), do steps 3-5

- **Step 3**

\[
D(1) = (0.2 - 0)^2 + (0.6 - 0)^2 + (0.5 - 0)^2 + (0.9 - 1)^2 = 0.66 \\
D(2) = (0.92 - 0)^2 + (0.76 - 0)^2 + (0.28 - 0)^2 + (0.12 - 1)^2 = 2.2768
\]

- **Step 4** The i/p vector is closest to o/p node 1, so \( J = 1 \)

- **Step 5** The weights on the winning unit are updated

This gives the weight matrix:

\[
W = \begin{bmatrix}
0.08 & 0.92 \\
0.24 & 0.76 \\
0.20 & 0.28 \\
0.96 & 0.12 \\
\end{bmatrix}
\]

This set of weights has not been modified.
Kohonen Self-Organizing Maps (SOM)

Examples:

- **Step 2** For \( s(3) = (1, 0, 0, 0) \), do steps 3-5

- **Step 3**

\[
D(1) = (.08 - 1)^2 + (.24 - 0)^2 + (.2 - 0)^2 + (.96 - 0)^2 = 1.8656 \\
D(2) = (.92 - 1)^2 + (.76 - 0)^2 + (.28 - 0)^2 + (.12 - 0)^2 = 0.6768
\]

- **Step 4** The i/p vector is closest to o/p node 2, so \( J = 2 \)

- **Step 5** The weights on the winning unit are updated

This gives the weight matrix:

\[
W = \begin{bmatrix}
.08 & .968 \\
.24 & .304 \\
.20 & .112 \\
.96 & .048
\end{bmatrix}
\]

This set of weights has not been modified.
Kohonen Self-Organizing Maps (SOM)

Examples:

- **Step 2**  
  For \( s(4) = (0, 0, 1, 1) \), do steps 3-5

- **Step 3**  
  \[
  D(1) = (0.08 - 0)^2 + (0.24 - 0)^2 + (0.2 - 1)^2 + (0.96 - 1)^2 = 0.7056
  
  D(2) = (0.968 - 0)^2 + (0.304 - 0)^2 + (0.112 - 1)^2 + (0.048 - 1)^2 = 2.724
  
- **Step 4**  
  The i/p vector is closest to o/p node 1, so \( J = 1 \)

- **Step 5**  
  The weights on the winning unit are updated
  
  This gives the weight matrix:

\[
W = \begin{bmatrix}
0.032 & 0.968 \\
0.096 & 0.304 \\
0.680 & 0.112 \\
0.984 & 0.048
\end{bmatrix}
\]

- **Step 6**  
  Reduce the learning rate \( \alpha = 0.5(0.6) = 0.3 \)

This set of weights has not been modified.
Kohonen Self-Organizing Maps (SOM)

Examples:

- The weights update equations are now
  \[ w_{ij}(new) = w_{ij}(old) + 0.3[x_i - w_{ij}(old)] + 0.7w_{ij}(old) + 0.3x_i \]

- The weight matrix after the 2\textsuperscript{nd} epoch of training is:
  \[
  W = \begin{bmatrix}
  0.016 & 0.980 \\
  0.047 & 0.360 \\
  0.630 & 0.055 \\
  0.999 & 0.024 \\
  \end{bmatrix}
  \]
  
  This set of weights has not been modified.

Modifying the learning rate and after 100 iterations (epochs), the weight matrix appears to be converging to:

\[
W = \begin{bmatrix}
0.0 & 1.0 \\
0.0 & 0.5 \\
0.5 & 0.0 \\
1.0 & 0.0 \\
\end{bmatrix}
\]

The 1\textsuperscript{st} column is the average of the 2 vsects in cluster 1, and the 2\textsuperscript{nd} col is the average of the 2 vsects in cluster 2.
Example of a SOM That Use a 2-D Neighborhood

Character Recognition – Fausett, pp. 176-178
Traveling Salesman Problem (TSP) by SOM

- The aim of the TSP is to find a tour of a given set of cities that is of minimum length.
- A tour consists of visiting each city exactly once and returning to the starting city.
- The net uses the city coordinates as input \((n=2)\); there are as many cluster units as there are cities to be visited. The net has a linear topology (with the first and last unit also connected).

Fig. shows the Initial position of cluster units and location of cities.
Traveling Salesman Problem (TSP) by SOM

Fig. Shows the result after 100 epochs of training with R=1 (learning rate decreasing from 0.5 to 0.4).

Fig. Shows the final tour after 100 epochs of training with R=0

Two candidate solutions:

\[ ADEFGHIJBC \]
\[ ADEFGHIJCB \]
Neural Networks Based on Competition.

- Introduction
- Fixed weight competitive nets
  - Maxnet
  - Mexican Hat
  - Hamming Net
- Kohonen Self-Organizing Maps (SOM)
- SOM in Matlab
- References and suggested reading
Self-Organizing Maps

Self-organizing feature maps (SOFM) learn to classify input vectors according to how they are grouped in the input space. They differ from competitive layers in that neighboring neurons in the self-organizing map learn to recognize neighboring sections of the input space. Thus, self-organizing maps learn both the distribution (as do competitive layers) and topology of the input vectors they are trained on.

The neurons in the layer of an SOFM are arranged originally in physical positions according to a topology function. The functions gridtop, hextop, or randtop can arrange the neurons in a grid, hexagonal, or random topology. Distances between neurons are calculated from their positions with a distance function. There are four distance functions, dist, boxdist, linkdist, and mandist. Link distance is the most common. These topology and distance functions are described in detail later in this section.

Here a self-organizing feature map network identifies a winning neuron $j*$ using the same procedure as employed by a competitive layer. However, instead of updating only the winning neuron, all neurons within a certain neighborhood $N_{r*}(d)$ of the winning neuron are updated using the Kohonen rule. Specifically, we adjust all such neurons $i \in N_{r*}(d)$ as follows:

$$w(q) = w(q - 1) + \alpha(p(q) - w(q - 1)) \text{ or}$$

$$w(q) = (1 - \alpha)w(q - 1) + \alpha p(q)$$

Here the neighborhood $N_{r*}(d)$ contains the indices for all of the neurons that lie within a radius $d$ of the winning neuron $j*.$
Architecture

\[ n_i^1 = \| | W_{1,1} p - p \| \]
\[ a_i^1 = \text{compet}(n_i^1) \]
newsom -- Create a self-organizing map

Syntax

net = newsom

net = newsom(PR,[D1,D2,...],TFCN,DFCN,OLR,OSTEPS,TLR,TND)

Description

net = newsom creates a new network with a dialog box.

net = newsom (PR, [D1,D2,...], TFCN, DFCN, OLR, OSTEPS, TLR, TND) takes,

PR - R x 2 matrix of min and max values for R input elements.
Di - Size of ith layer dimension, defaults = [5 8].
TFCN - Topology function, default = 'hextop'.
DFCN - Distance function, default = 'linkdist'.
OLR - Ordering phase learning rate, default = 0.9.
OSTEPS - Ordering phase steps, default = 1000.
TLR - Tuning phase learning rate, default = 0.02;
TND - Tuning phase neighborhood distance, default = 1.

and returns a new self-organizing map.

The topology function TFCN can be hextop, gridtop, or randtop. The
distance function can be linkdist, dist, or mandist.
Neural Networks Based on Competition.

- Introduction
- Fixed weight competitive nets
  - Maxnet
  - Mexican Hat
  - Hamming Net
- Kohonen Self-Organizing Maps (SOM)
- SOM in Matlab
- References and suggested reading
Suggested Reading.


References:

These lecture notes were based on the references of the previous slide, and the following references

1. Berlin Chen Lecture notes: Normal University, Taipei, Taiwan, ROC. [http://140.122.185.120](http://140.122.185.120)