NEURAL NETWORKS
(ELEC 5240 and ELEC 6240)
special NN architectures

Bodgan M. Wilamowski

How to find proper architecture?

• Predict theoretically
  Reduce number of inputs using PCA
• Use evolutionary computation technique
• Try and error approach
• Start with large system and prune it.
  Eliminate (combine) neurons with the same similar (complement) responses for all patterns
• Start with smaller system and add neurons as needed
  Look for oscillatory weight changes during training process.
Principle Component Analysis

Hebb’s rule

\[ W_{ij} = cX_iY_j \]

Connection between two neurons become stronger if neurons are responding the same way.

By training a simple neuron where \( x \) are inputs and \( y \) are outputs will lead to a solution with weights specifying a most probable direction of patterns.

Oja’s modification:

\[ W_i = cX_iY - cW_iy^2 = cy\left(x_i - W_iy\right) \]

This leads to a solution where length of the weight vector is normalized to one and direction is set by the principle component.

In order to extract all \( M \) principle components then network with \( M \) neurons has to be used and

\[ W_{ij} = cyj \left(x_i - \sum_{k=j}^{M} y_k W_{ik}\right) \]

Deficiency of statistical approaches

(correlation or PC analysis)

![Graph showing correlation coefficients](image)
Deficiency of statistical approaches
(correlation or PC analysis)

- Correlation coefficient $\rho = 0.95988$
- Correlation coefficient $\rho = 0.99529$
- Correlation coefficient $\rho = 0.96314$
- Correlation coefficient $\rho = 0.995$
Functional Link Networks
Nilsen-Pao

Functional link networks for solution of the XOR problem: (a) using unipolar signals, (b) using bipolar signals.

Polynomial Networks

Fourier or other series? Nonlinear regression?
Functional Link Networks

Genetic algorithms

The counterpropagation networks

Kohonen layer

normalized inputs

Kohonen layer

unipolar neurons

summing circuits
The counterpropagation networks (ROM)

Hamming layer

unipolar neurons

summing circuits

inputs      outputs
1 -1 1 -1   -1 -1 1
-1 1 1 -1   -1 1 -1
-1 1 -1 -1   1 -1 1
-1 -1 -1 1   -1 -1 1
1 1 -1 -1   1 1 -1

11

The counterpropagation networks (ROM)

Hamming layer

unipolar neurons

summing circuits

inputs      outputs
1 -1 1 -1   -1 -1 1
-1 1 1 -1   -1 1 -1
-1 1 -1 -1   1 -1 1
-1 -1 -1 1   -1 -1 1
1 1 -1 -1   1 1 -1

12
The counterpropagation networks
(analog memory)

analog memory with analog address
analog memory with analog address

unipolar neurons
summing circuits

inputs          output
1 1             1.414
1 2             2.236
1 3             3.162
2 1             -2.236
2 2             2.828

\[ z = \sqrt{x^2 + y^2} \]

Normalization of outputs of unipolar neurons
analog memory with analog address

Normalization of outputs of unipolar neurons

\[
\sqrt{R^2 - \|X\|^2}
\]

unipolar neurons

summing circuits

RBF - Radial Basis Function networks

\[
\text{out} = \exp\left(-\frac{\|x - s\|^2}{\sigma}\right)
\]

hidden "neurons"

inputs

outputs

summing circuit

output normalization
RBF - Radial Basis Function networks

\[ \text{out} = \exp \left( - \frac{\| \mathbf{x} - \mathbf{s} \|^2}{\sigma} \right) \]

hidden "neurons"

inputs

x is close to \( s_2 \)

\( s_1 \) stored

\( s_2 \) stored

\( s_3 \) stored

\( s_4 \) stored

\( D \)

summing circuit

outputs

\( y_1 \)

\( y_2 \)

\( y_3 \)

output normalization

summing circuit
LVQ Learning Vector Quantization

Competitive Layer  LinearLayer

First layer detect subclasses
Second layer combines subclasses into a single class

First layer computes Euclidean distances between input pattern and stored patterns.
Winning “neuron” is with the minimum distance
LVQ Learning Vector Quantization

The cascade correlation architecture

hidden neurons

weights adjusted every step

once adjusted weights and then frozen
The cascade correlation architecture

Step 1 - output neuron is trained

weights adjusted every step

+1

input

output

Step 2 - hidden neuron is inserted and trained to the errors from the Step 1

hidden neuron #1

at this step hidden neuron #1 is not connected to the network and it is trained so its output values matches errors from the first step

Notice that this is one neuron training and an efficient algorithm can be used such as pseudo inversion

weights adjusted every step

once adjusted weights and then frozen
The cascade correlation architecture

Step 3 - output neuron is trained to the desired outputs

hidden neuron #1

at this step hidden neuron #1 is connected to the network and its weights are frozen

weights adjusted every step
once adjusted weights and then frozen

Step 4 - new hidden neuron is inserted and trained to the errors from the previous step

hidden neuron #1

at this step hidden neuron #1 is not connected to the network

weights adjusted every step
once adjusted weights and then frozen
The cascade correlation architecture

Step 5 - output neuron is trained to the desired outputs

at this step hidden neurons #1 and #2 are connected to the network and their weights are frozen

weights adjusted every step

once adjusted weights and then frozen

The cascade correlation architecture

Step 6 - new hidden neuron is inserted and trained to the errors from the previous step

weights adjusted every step

once adjusted weights and then frozen
The cascade correlation architecture

Step 7 - output neuron is trained to the desired outputs

Training to the maximum of correlation $C$

$$C = \sum_{j=1}^{M} \sum_{p=1}^{P} (e_{jp} - \bar{e}_j)(o_p - \bar{o})$$

$$\frac{\partial C}{\partial W_i} = \sum_{j=1}^{M} \sum_{p=1}^{P} s_j (e_{jp} - \bar{e}_j)f_p x_{ip}$$

where:
- $j=1$ to $M$ number of outputs
- $P=1$ to $P$ number of patterns
- $O$ output of hidden neuron
- $e$ errors on outputs of the network
- $s$ is sign of the correlation
**MAX net**

\[
W = \begin{bmatrix}
1 & -\varepsilon & -\varepsilon & -\varepsilon \\
-\varepsilon & 1 & -\varepsilon & -\varepsilon \\
-\varepsilon & -\varepsilon & 1 & -\varepsilon \\
-\varepsilon & -\varepsilon & -\varepsilon & 1
\end{bmatrix}
\]

**Activation function**

-0.0800 -0.4000 -0.3200 -0.2560 -0.2268
1.3600 1.2320 1.1584 1.1341 1.1341
-0.3200 -0.4000 -0.3200 -0.2560 -0.2268
0.6400 0.3680 0.1216 -0.1101 -0.2268
If inputs are binaries, for example \( \mathbf{X}=[1, -1, 1, -1, -1] \) then the maximum value of \( net \)
\[
net = \sum_{i=1}^{5} x_i w_i = \mathbf{XW}^T
\]
is when weights are identical to the input pattern \( \mathbf{W}=[1, -1, 1, -1, -1] \).
This concept can be extended to weights and patterns with analog values as long as both lengths of the weight vector and input pattern vectors are the same.

The Euclidean distance between weight vector $W$ and input vector $X$ is

$$
\|W - X\| = \sqrt{(w_1 - x_1)^2 + (w_2 - x_2)^2 + \cdots + (w_n - x_n)^2}
$$

$$
\|W - X\| = \sqrt{\sum_{i=1}^{n} (w_i - x_i)^2}
$$
**Kohonen Network**

\[ \| W - X \| = \sqrt{WW^T - 2WX^T + XX^T} \]

When the lengths of both the weight and input vectors are normalized to value of one

\[ \| X \| = 1 \quad \text{and} \quad \| W \| = 1 \]

Then the equation simplifies to

\[ \| W - X \| = \sqrt{2 - 2WX^T} \]

Please notice that the maximum value of net value \( net = 1 \) is when \( W \) and \( X \) are identical.

---

**Kohonen Network**

**The unsupervised training process**

(1) All patterns are normalized (the lengths of the pattern vectors are normalized to unity)

\[
Z_i = \frac{x_i}{\sqrt{\sum_{i=1}^{n} x_i^2}} \\
\cdots \\
Z_n = \frac{x_n}{\sqrt{\sum_{i=1}^{n} x_i^2}}
\]
(2) Weights are chosen randomly for all neurons

(3) Lengths of the weight vectors are normalized to unity

\[ v_i = \frac{w_i}{\sqrt{\sum_{i=1}^{n} w_i^2}} \]

\[ \ldots \]

\[ v_n = \frac{w_n}{\sqrt{\sum_{i=1}^{n} w_i^2}} \]

(4) A pattern is applied to input and net values are calculated for all neurons

\[ net = \sum_{i=1}^{5} z_i v_i = ZV^T \]

(5) A winning neuron is chosen (neuron with largest net value)
Kohonen Network
The unsupervised training process

(6) Weights for the winner $k$ are modified using a weighted average:

$$W_k = V_k + \alpha Z$$

$\alpha$ is the learning.
Weights of other neurons are not modified.

(7) Weights for the winning neuron are normalized.

$$v_i = \frac{W_i}{\sqrt{\sum_{i=1}^{n} W_i^2}}$$

$$v_n = \frac{W_n}{\sqrt{\sum_{i=1}^{n} W_i^2}}$$

(8) Another pattern is applied (go to step (4))
Kohonen Network
The unsupervised training process

During pattern applications some neurons are frequent winners and other never take part in the process. The latter ones are eliminated and the number of recognized clusters is equal to the number of surviving neurons.

Kohonen Network
The unsupervised training process

One disadvantage of this method is that the classification is strongly dependent on the initial set of randomly chosen weights.

Another disadvantage is that during the normalization process important information about the length of input patterns is lost.
Kohonen Network
The unsupervised training process example

patterns

6.1290  1.3876
4.1168  2.9694
6.2139  2.4288
5.9630  0.7258
1.0562  5.8288
1.8184  6.0148
2.6108  5.4870
1.5999  4.1317
1.1046  4.1969

normalized patterns

0.9753  0.2208
0.8110  0.5850
0.8759  0.4825
0.9314  0.3640
0.9927  0.1208
0.1783  0.9840
0.2894  0.9572
0.4296  0.9030
0.3611  0.9325
0.2545  0.9671

normalized initial weights

0.9459  0.3243
0.6690  0.7433
0.3714  0.9285
Kohonen Network
The unsupervised training process example

Neuron #1 is the winner. Weights for neuron #1 are updated and normalized

\[(0.9459 \ 0.3243) + \alpha (0.9927 \ 0.1208)\]

\[(0.9459 \ 0.3243) + 0.3 (0.9927 \ 0.1208) = (1.2437 \ 0.3606) \Rightarrow (0.9605 \ 0.2784)\]

Neuron #2 is the winner. Weights for neuron #2 are updated and normalized

\[(0.6690 \ 0.7433) + \alpha (0.8110 \ 0.5850)\]

\[(0.6690 \ 0.7433) + 0.3 (0.8110 \ 0.5850) = (0.9123 \ 0.9188) \Rightarrow (0.7046 \ 0.7096)\]
Kohonen Network

The unsupervised training process example

Neuron #3 is the winner. Weights for neuron #3 are updated and normalized

\[(0.3714 \ 0.9285) + \text{alpha} \ (0.2894 \ 0.9572)\]

\[(0.3714 \ 0.9285) + 0.3 \ (0.2894 \ 0.9572) = (0.4582 \ 1.2156) \Rightarrow (0.3527 \ 0.9357)\]

Kohonen Network

The unsupervised training process example

Neuron #1 is the winner. Weights for neuron #1 are updated and normalized

\[(0.9605 \ 0.2784) + \text{alpha} \ (0.9314 \ 0.3640)\]

\[(0.9605 \ 0.2784) + 0.3 \ (0.9314 \ 0.3640) = (1.2437 \ 0.3606) \Rightarrow (0.9605 \ 0.2784)\]
Kohonen Network
The unsupervised training process example

Process continues for all patterns

weights after the first iteration

weights after the second iteration

o - normalized patterns
x - initial weights
* - current weights
**Kohonen Network**

The unsupervised training process example

weights after the 30 iterations

0.9699 0.2435
0.8492 0.5281
0.3072 0.9516

- o - normalized patterns
- x - initial weights
- * - current weights

weights are representing center of clusters

**Kohonen Network**

discussion
Kohonen Network
the same example with two neurons

<table>
<thead>
<tr>
<th>Initial weights</th>
<th>after one iteration</th>
<th>after 30 iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9459 0.3243</td>
<td>0.3714 0.9285</td>
<td>0.9265 0.3764</td>
</tr>
<tr>
<td>0.3072 0.9516</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sarajedini and Hecht-Nielsen network

Let us consider stored vector \( w \) and input pattern \( x \). Both input and stored patterns have the same dimension \( n \). The square Euclidean distance between \( x \) and \( w \) is:

\[
|x - w|^2 = (x_1 - w_1)^2 + (x_2 - w_2)^2 + \cdots + (x_n - w_n)^2
\]

After defactorization

\[
|x - w|^2 = x_1^2 + x_2^2 + \cdots + x_n^2 + w_1^2 + w_2^2 + \cdots + w_n^2 - 2(x_1w_1 + x_2w_2 + \cdots + x_nw_n)
\]

finally

\[
|x - w|^2 = x^T x + w^T w - 2x^T w = \|x\|^2 + \|w\|^2 - 2 \text{net}
\]
Input pattern transformation on a sphere
fix to Kohonen network deficiency

Network with two neurons capable of separating crescent shape of patterns (a) input-output mapping, (b) network diagram
Spiral problem solved with sigmoidal type neurons  (a) network diagram, (b) input-output mapping.

ART
Adaptive Resonance Theory
ART
Adaptive Resonance Theory

Step 1: The vigilance threshold \( \rho \) is set, and for the \( n \)-tuple input vectors and \( M \) top-layer neurons the weights are initialized. The matrices \( W, V \) are \((M \times n)\) and each is initialized with identical entries:

\[
W = \begin{bmatrix} 1 \\ 1 + 1 \end{bmatrix} \quad (7.35a)
\]

\[
V = \begin{bmatrix} 1 \\ \end{bmatrix} \quad (7.35b)
\]

\[
0 < \rho < 1 \quad (7.35c)
\]

Step 2: Binary unipolar input vector \( x \) is presented at input nodes, \( x_i = 0, 1 \), for \( i = 1, 2, \ldots, n \).

Step 3: All matching scores are computed from (7.31) or (7.33) as follows:

\[
y^j_m = \sum x_i w_{jm}^x, \quad \text{for } m = 1, 2, \ldots, M \quad (7.36)
\]

In this step, selection of the best matching existing cluster, \( j \), is performed according to the maximum criterion (7.34) as follows:

\[
y^j = \max_{m=1,2,\ldots,M} y^j_m \quad (7.37)
\]

[Note: A practical note is that the recursences of (7.32) done by MAXNET may be skipped in a discrete-time learning simulation since their only consequence of importance is suppressing to zero all upper node outputs but the \( j \)th one.]

Step 4: The similarity test for the winning neuron \( j \) is performed as follows:

\[
\frac{1}{\|x\|} \sum x_i v_{jm} > \rho \quad (7.38)
\]

where \( \rho \) is the vigilance parameter and the norm \( \|x\| \) is defined for the purpose of this algorithm as follows:

\[
\|x\| = \sqrt{\sum x_i^2} \quad (7.39)
\]

If the test (7.38) is passed, the algorithm goes to Step 5.

If the test has failed, the algorithm goes to Step 6 only if the top layer has more than a single active node left. Otherwise, the algorithm goes to Step 5.

Step 5: Entries of the weight matrices are updated for index \( j \) passing the test of Step 4. The updates are only for entries \((i,j)\), where \( i = 1, 2, \ldots, M \), and are computed as follows:

\[
w_{ij}(t + 1) = \frac{v_{ij}(0)x_i}{0.5 + \sum_{j'=1}^M v_{ij'}(0)} \quad (7.40a)
\]

\[
v_{ij}(t + 1) = x_i v_{ij}(t) \quad (7.40b)
\]

This updates the weights of the \( j \)th cluster (newly created or the existing one). The algorithm returns to Step 2.

Step 6: The node \( j \) is deactivated by setting \( y_j \) to 0. Thus, this mode does not participate in the current cluster search. The algorithm goes back to Step 3 and it will attempt to establish a new cluster different than \( j \) for the pattern under test.

mess!

Copied form Jacek Zurada “Artificial Neural Systems” West Publishing Company 1992

Mountain clustering

Superposition of Gaussian hills

1. For each pattern a small hill of a bell (Gaussian) shape is formed.
2. These hills are forming mountains.
3. The highest mountains are the clusters

This is very computationally intensive process
Forming clusters as needed using minimum distance concept

much simpler and more efficient than ART

1. First pattern is applied and the cluster is formed
2. Next pattern is applied and then:
   a) If distance form all existing clusters is larger than threshold then a new cluster is formed
   b) Else weights of the closest cluster are updated

\[
W_k = \frac{mW_k + \alpha X}{m + 1}
\]

where \( m \) is the number of previous patterns of a given set which were used to update this particular neuron and \( \alpha \) is the learning constant

First pattern applied and the first cluster is formed

\[
\begin{array}{ll}
5.9630 & 0.7258 \\
4.1168 & 2.9694 \\
1.8184 & 6.0148 \\
6.2139 & 2.4288 \\
6.1290 & 1.3876 \\
1.0562 & 5.8288 \\
4.3185 & 2.3792 \\
2.6108 & 5.4870 \\
1.5999 & 4.1317 \\
1.1046 & 4.1969
\end{array}
\]
Forming clusters as needed using minimum distance concept

Second pattern applied and the first cluster is moved

Forming clusters as needed using minimum distance concept

Third pattern applied and a new cluster is formed
Forming clusters as needed using minimum distance concept

<table>
<thead>
<tr>
<th></th>
<th>5.9630</th>
<th>0.7258</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4.1168</td>
<td>2.9694</td>
</tr>
<tr>
<td></td>
<td>1.8184</td>
<td>6.0148</td>
</tr>
<tr>
<td></td>
<td>6.2139</td>
<td>2.4288</td>
</tr>
<tr>
<td></td>
<td>6.1290</td>
<td>1.3876</td>
</tr>
<tr>
<td></td>
<td>1.0562</td>
<td>5.8288</td>
</tr>
<tr>
<td></td>
<td>4.3185</td>
<td>2.3792</td>
</tr>
<tr>
<td></td>
<td>2.6108</td>
<td>5.4870</td>
</tr>
<tr>
<td></td>
<td>1.5999</td>
<td>4.1317</td>
</tr>
<tr>
<td></td>
<td>1.1046</td>
<td>4.1969</td>
</tr>
</tbody>
</table>

ED = 1.2895 5.6727

Fourth pattern applied and the first cluster is moved

<table>
<thead>
<tr>
<th></th>
<th>5.3476</th>
<th>1.4737</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6.2139</td>
<td>2.4288</td>
</tr>
<tr>
<td></td>
<td>1.8184</td>
<td>6.0148</td>
</tr>
<tr>
<td></td>
<td>1.5999</td>
<td>4.1317</td>
</tr>
<tr>
<td></td>
<td>1.1046</td>
<td>4.1969</td>
</tr>
</tbody>
</table>

After all 10 patterns

<table>
<thead>
<tr>
<th></th>
<th>5.4507</th>
<th>1.7694</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.6380</td>
<td>5.1318</td>
</tr>
</tbody>
</table>
**Parity problems**

**(FF networks)**

**Unipolar**

_weights = (-0.5, -1.5)_

**Bipolar**

_weights = (-0.5, -1.5, -2.5)_

**Parity problems**

**(FF fully connected)**

_weights = +1_
Parity problems
(cascade)

weights = +1

Parity problems
(pipeline architecture)

weights = +1