CS407 Neural Computation

Lecture 6:
Associative Memories and Discrete Hopfield Network.

Lecturer: A/Prof. M. Bennamoun
AM and Discrete Hopfield Network.

- Introduction
- Basic Concepts
- Linear Associative Memory (Hetero-associative)
- Hopfield’s Autoassociative Memory
- Performance Analysis for Recurrent Autoassociation Memory
- References and suggested reading
Introduction...

- To a significant extent, learning is the process of forming associations between related patterns.
- Aristotle observed that human memory connects items (ideas, sensations, etc.) that are
  - Similar
  - Contrary
  - Occur in close proximity (spatial)
  - Occur in close succession (temporal)
- The patterns we associate together may be
  - of the same type or sensory modality (e.g. a visual image may be associated with another visual image)
  - or of different types (e.g. a fragrance may be associated with a visual image or a feeling).
- Memorization of a pattern (or a group of patterns) may be considered to be associating the pattern with itself.
Introduction…

- In this lecture, we will consider some relatively simple (single layer) NNs that can learn a set of pattern pairs (or associations).
- An associative memory (AM) net may serve as a highly simplified model of human memory. However, we shall not address the question whether they are all realistic models.
- AM provide an approach of storing and retrieving data based on content rather than storage address (info. storage in a NN is distributed throughout the system in the net’s weights, hence a pattern does not have a storage address).
Introduction...

- Each association is an I/P O/P vector pair, \( s:f \).
- Two types of associations. For two patterns \( s \) and \( f \):
  - If \( s = f \) the net is called auto-associative memory.
  - If \( s \neq f \) the net is called hetero-associative memory.

Association response: (a) autoassociation and (b) heteroassociation.
Introduction…

- In each of these cases, the net not only learns the specific patterns pairs that were used for training, but is also able to recall the desired response pattern when given an I/P stimulus that is similar, but not identical, to the training I/P.

- Associative recall
  - evoke associated patterns
  - recall a pattern by part of it
  - evoke/recall with incomplete/noisy patterns
Introduction…

- Before training an AM NN, the original patterns must be converted to an appropriate representation for computation.

- However, not all representations of the same pattern are equally powerful or efficient.

- In a simple example, the original pattern might consist of “on” and “off” signals, and the conversion could be “on” $\rightarrow +1$, “off” $\rightarrow 0$ (binary representation) or “on” $\rightarrow +1$, “off” $\rightarrow -1$ (bipolar representation).

- Two common training methods for single layer nets are usually considered:
  - Hebbian learning rule and its variations
  - gradient descent
Introduction...

- Architectures of NN associative memory may be
  - Feedforward or
  - Recurrent (iterative).

- On that basis we distinguish 4 types of AM:
  - Feedforward Hetero-associative
  - Feedforward Associative
  - Iterative Hetero-associative
  - Iterative associative

- A key question for all associative nets is
  - how many patterns can be stored before the net starts to “forget” patterns it has learned previously (storage capacity)
Introduction

- **Associative memories**: Systems for associating the input patterns with the stored patterns (prototypes)
  - **Dynamic systems** with feedback networks
  - **Static/Feedforward systems** without feedback networks

- **Information recording**: A large set of patterns (the priori information) are stored (memorized)

- **Information retrieval/recall**: Stored prototypes are excited according to the input key patterns
Introduction…

- No usable addressing scheme exists
  - Memory information spatially distributed and superimposed through the network
  - No memory locations have addresses

- Expectations regarding associative memories
  - As large of a capacity of $P$ stored patterns as possible
  - Data to be stored in a robust manner
  - Adaptability: Addition or elimination of associations
AM and Discrete Hopfield Network.

- Introduction
- Basic Concepts
- Linear Associative Memory (Hetero-associative)
- Hopfield’s Autoassociative Memory
- Performance Analysis for Recurrent Autoassociation Memory
- References and suggested reading
Basic Concepts

- Associative Mapping of Inputs to Outputs

\[ x = [x_1 \ x_2 \ \ldots \ x_n]' \]
\[ v = [v_1 \ v_2 \ \ldots \ v_m]' \]

Retreival: \[ v = M[x] \]

Figure 6.1  Block diagram of an associative memory.

B. Chen & Zurada
Basic Concepts

- $M$ is a general matrix-type operator
  - Memory Paradigms (taxonomy)
    - Dynamic systems
    - Static/Feedforward systems

- Recording and Retrieval
  - **Recording**
    - $M$ is expressed as the prototype vectors stored
  - **Retrieval**
    - Mapping: $\mathbf{x} \rightarrow \mathbf{v}$
    - Linear or nonlinear
    - Input a key vector $\mathbf{x}$ and find a desired vector $\mathbf{v}$ previously stored in the network
Basic Concepts

– Input/Output Relation

• Heteroassociative memory

\[ x^{(i)} \rightarrow v^{(i)} \mid v^{(i)} \neq x^{(i)} \text{ for } i = 1,\ldots,p \]

– With same dimensionality or not

• Autoassociative Memory

\[ x^{(i)} \rightarrow v^{(i)} \mid v^{(i)} = x^{(i)} \text{ for } i = 1,\ldots,p \]

Two sets of prototype vectors

One set of prototype vectors

Most for recovery of undistorted prototypes
Basic Concepts

- The most important classes of associative memories are static and dynamic memories. The taxonomy is entirely based on their recall principles.

- Static Memory
  - Recall an I/P response after an input has been applied in one feedforward pass, and theoretically without delay.
  - Non-recurrent (no feedback, no delay)

\[ \mathbf{v}^k = M_1 \left[ \mathbf{x}^k \right] \]

- \( k \) denotes the index of recursion
- \( M_1 \) is an operator symbol
- This eq. Represents a system of non-linear algebraic eqs.
Basic Concepts

- Dynamic Memory
  - Produce recall as a result of output/input feedback interactions, which requires time
  - Recurrent, time-delayed
  - Dynamically evolved and finally converge to an equilibrium state according to the recursive formula

\[ v^{k+1} = M_2[x^k, v^k] \]

The operator \( M_2 \) operates at present instant \( k \) on the present input \( x^k \) and output \( v^k \) to produce the o/p in the next instant \( k+1 \)

\( \Delta \) is a unit delay needed for cyclic operation.

*The pattern is associated to itself (auto-)*
Basic Concepts

- The fig. below shows the block diagram of a recurrent heteroassociative memory net.

- It associates pairs of vectors \((x^{(i)}, v^{(i)})\).
Basic Concepts

- **Dynamic Memory**
  - **Hopfield model**: a recurrent autoassociative network for which the input $x^0$ is used to initialize $v^0$, i.e. $x^0 = v^0$, and the input is then removed for the following evolution

\[
M_2 [v^k] = v^{k+1}
\]

 nondinear mapping

\[
\Gamma[Wv^k] = v^{k+1}
\]

Operator $M_2$ consists of multiplication by a weight matrix followed by the ensemble of non-linear mapping operations $v_i = f(net_i)$ performed by the layer of neurons
Basic Concepts

\[ \Gamma = \begin{bmatrix} \text{sgn}(\cdot) & 0 & \ldots & 0 \\ 0 & \text{sgn}(\cdot) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & \ldots & \text{sgn}(\cdot) \end{bmatrix} \]

– Discrete Hopfield model:
  - Operated on bipolar binary values (±1)
  - Hard-limiting (binary) activation function

\[ v_i = f(net_i) = \text{sgn}(net_i) \]
Basic Concepts

Equilibrium states

Regarding the vector $v(k+1)$ as the state of the network at the (k+1)’th instant we can consider the recurrent equation $v^{k+1} = \Gamma [Wv^k]$ as defining a mapping of the vector $v$ into itself.

The example state transition map for a memory network can be represented as shown on the fig. below.

Each node of the graph is equivalent to a state and has one and only one edge leaving it.

1/ If the transitions terminate with a state mapping into itself, as in the case of node A, the the equilibrium A is the fixed point solution (equilibrium)

2/ If the transitions end in a cycle of states as in nodes B, then we have a limit cycle solution with a certain period. The period is defined as the length of the cycle. The fig. shows the limit cycle B of length 3.
AM and Discrete Hopfield Network.

- Introduction
- Basic Concepts
- Linear Associative Memory (Hetero-associative)
- Hopfield’s Autoassociative Memory
- Performance Analysis for Recurrent Autoassociation Memory
- References and suggested reading
Linear Associative Memory

- Belong to Static Networks
- No nonlinear or delay operations
- Dummy neurons with identity activation functions

\[ v_i = f\left(\text{net}_i\right) = \text{net}_i \]

\[ \mathbf{v} = \mathbf{Wx} \]
Linear Associative Memory

- Given \( p \) associations \( \{s^{(i)}, f^{(i)}\} \)

\[
\begin{align*}
    s^{(i)} &= \left[ s_1^{(i)}, s_2^{(i)}, \ldots, s_n^{(i)} \right]^t \\
    f^{(i)} &= \left[ f_1^{(i)}, f_2^{(i)}, \ldots, f_m^{(i)} \right]^t
\end{align*}
\]

- Apply Hebbian Learning Rule

\[
w'_{ij} = w_{ij} + f_i s_j
\]

\[W' = W + \left( fs \right)^t\] outer product (learning rule)

\[W_0 = 0\]

- \( p \) pairs learned (implies superposition of weights)

\[W' = \sum_{i=1}^{p} f^{(i)} s^{(i)} t\]

\[W' = FS^t\]

Recall that: 
- \( n \) = size of stimuli
- \( m \) = size of forced responses vects.

Stimuli (e.g. patterns)
Forced response (e.g. images of input patterns)
Hebbian Learning

- A purely feedforward, unsupervised learning
- The learning signal is equal to the neuron’s output
  \[ r = o_i = f(w_i^t x) \]
  \[ \Delta w_i = c \cdot o_i \cdot x = cf(w_i^t x) \cdot x \]
  or \[ \Delta w_{ij} = c \cdot f(w_i^t x) \cdot x_j \]
- The weight initialization at small random values around \( w_i = 0 \) prior to learning
- If the cross-product of output and input (or correlation) is positive, it results in an increase of the weight, otherwise the weight decreases
Linear Associative Memory

- Perform the associative recall using $s^{(j)}$

$$v = W' s^{(j)} = \left( \sum_{i=1}^{p} f^{(i)} s^{(i)t} \right) s^{(j)}$$

$$= f^{(1)} s^{(1)t} s^{(j)} + \ldots + f^{(j)} s^{(j)t} s^{(j)} + \ldots f^{(p)} s^{(p)t} s^{(j)}$$

- If the desired response is $v = f^{(j)}$

  - Necessary conditions: **orthonormal**

  $$s^{(i)t} s^{(j)} = 0, \quad \text{for} \ i \neq j$$

  $$s^{(j)t} s^{(j)} = 1$$

  - Rather strict and may not always hold
Linear Associative Memory

If a distorted input is presented

\[ s^{(j)\prime} = s^{(j)} + \Delta^{(j)} \]

\[ \mathbf{v} = \mathbf{f}^{(j)} + \mathbf{f}^{(j)} s^{(j)t} \Delta^{(j)} + \sum_{i \neq j}^{p} \mathbf{f}^{(i)} s^{(i)t} \Delta^{(j)} \]

- Cross-talk noise remains additive at the memory output to the originally stored association
- Linear associative memory perform rather poorly when with distorted stimuli vectors
  - Limited usage (not an accurate retrieval of the originally stored association).
Linear Associative Memory

- Example: three associations

\((s^{(1)}, f^{(1)}), (s^{(2)}, f^{(2)}), (s^{(3)}, f^{(3)})\)

\(s^{(1)} = (1,0,0)^T, s^{(2)} = (0,1,0)^T, s^{(3)} = (0,0,1)^T\)

\(f^{(1)} = (2,1,2)^T, f^{(2)} = (1,2,3)^T, f^{(3)} = (3,1,2)^T\)

\(M' = f^{(1)}s^{(1)t} + f^{(2)}s^{(2)t} + f^{(3)}s^{(3)t}\)

\[
\begin{bmatrix}
2 \\
1 \\
2
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 \\
2 & 0 & 1 & 0 \\
3 & 1 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
2 \\
1 \\
3
\end{bmatrix}
= \begin{bmatrix}
2 & 1 & 3 \\
1 & 2 & 1 \\
2 & 3 & 2
\end{bmatrix}
\]

\(v = M's^{(2)} = \begin{bmatrix}
2 & 1 & 3 \\
1 & 2 & 1 \\
2 & 3 & 2
\end{bmatrix} \begin{bmatrix}
0 \\
1 \\
0
\end{bmatrix} = \begin{bmatrix}
1 \\
2 \\
3
\end{bmatrix}\)
Alternative View

- **Goal:** Associate an input vector with a specific output vector in a neural net.

- In this case, Hebb’s Rule is the same as taking the outer product of the two vectors:

\[
\mathbf{s} = (s_1, \ldots, s_i, \ldots s_n) \quad \text{and} \quad \mathbf{f} = (f_1, \ldots, f_i, \ldots f_m)
\]

\[
\mathbf{s}\mathbf{f} = \begin{bmatrix}
    s_1 \\
    \vdots \\
    s_i \\
    \vdots \\
    s_n
\end{bmatrix}
\begin{bmatrix}
    f_1 & \cdots & f_m
\end{bmatrix}
= \begin{bmatrix}
    s_1f_1 & \cdots & s_1f_m \\
    \vdots & \cdots & \vdots \\
    s_nf_1 & \cdots & s_nf_m
\end{bmatrix}
\]

← Weight matrix
Weight Matrix

- To store more than one association in a neural net using Hebb’s Rule
  - Add the individual weight matrices

- This method works only if the input vectors for each association are orthogonal (uncorrelated)
  - That is, if their dot product is 0

\[
\mathbf{s} = (s_1, \ldots, s_i, \ldots s_n) \text{ and } \mathbf{f} = (f_1, \ldots, f_i, \ldots f_m)
\]

\[
\mathbf{s} \cdot \mathbf{f} = [s_1 \ldots s_n] \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix} = 0
\]
Heteroassociative Architecture

There are $n$ input units and $m$ output units with each input connected to each output unit.

For $m = 3$

Activation Functions:

For bipolar targets:

$$y_i = \begin{cases} 
1 & \text{if } y_{\text{in}_i} > 0 \\
0 & \text{if } y_{\text{in}_i} = 0 \\
-1 & \text{if } y_{\text{in}_i} < 0
\end{cases}$$

For binary targets:

$$y_i = \begin{cases} 
1 & \text{if } y_{\text{in}_i} > 0 \\
0 & \text{if } y_{\text{in}_i} \leq 0
\end{cases}$$
Example

GOAL: build a neural network which will associate the following two sets of patterns using Hebb’s Rule:

\[
\begin{align*}
\mathbf{s}_1 &= (1, -1, -1, -1) & \mathbf{f}_1 &= (1, -1, -1) \\
\mathbf{s}_2 &= (-1, 1, -1, -1) & \mathbf{f}_2 &= (1, -1, 1) \\
\mathbf{s}_3 &= (-1, -1, 1, -1) & \mathbf{f}_3 &= (-1, 1, -1) \\
\mathbf{s}_4 &= (-1, -1, -1, 1) & \mathbf{f}_4 &= (-1, 1, 1)
\end{align*}
\]

The process will involve 4 input neurons and 3 output neurons.

The algorithm involves finding the four outer products and adding them.
Algorithm

Pattern pair 1:
\[
\begin{bmatrix}
1 \\
-1 \\
-1
\end{bmatrix}
\begin{bmatrix}
1 & -1 & -1
\end{bmatrix}
= 
\begin{bmatrix}
1 & -1 & -1 \\
-1 & 1 & 1 \\
-1 & 1 & 1
\end{bmatrix}
\]

Pattern pair 2:
\[
\begin{bmatrix}
-1 \\
1 \\
-1
\end{bmatrix}
\begin{bmatrix}
1 & -1 & 1
\end{bmatrix}
= 
\begin{bmatrix}
-1 & 1 & -1 \\
-1 & 1 & -1 \\
-1 & 1 & -1
\end{bmatrix}
\]

Pattern pair 3:
\[
\begin{bmatrix}
-1 \\
-1 \\
1
\end{bmatrix}
\begin{bmatrix}
-1 & 1 & -1
\end{bmatrix}
= 
\begin{bmatrix}
1 & -1 & 1 \\
1 & -1 & 1 \\
1 & -1 & 1 \\
-1 & 1 & -1 \\
-1 & 1 & -1 \\
-1 & 1 & -1
\end{bmatrix}
\]

Pattern pair 4:
\[
\begin{bmatrix}
-1 \\
-1 \\
1
\end{bmatrix}
\begin{bmatrix}
-1 & 1 & 1
\end{bmatrix}
= 
\begin{bmatrix}
1 & -1 & -1 \\
1 & -1 & -1 \\
1 & -1 & -1 \\
-1 & 1 & 1
\end{bmatrix}
\]
Weight Matrix

Add all four individual weight matrices to produce the final weight matrix:

\[
\begin{bmatrix}
1 & -1 & -1 \\
-1 & 1 & 1 \\
-1 & 1 & 1 \\
-1 & 1 & 1
\end{bmatrix}
+ \begin{bmatrix}
-1 & 1 & -1 \\
1 & -1 & 1 \\
-1 & 1 & -1 \\
-1 & 1 & -1
\end{bmatrix}
+ \begin{bmatrix}
1 & -1 & 1 \\
1 & -1 & 1 \\
-1 & 1 & -1 \\
-1 & 1 & -1
\end{bmatrix}
+ \begin{bmatrix}
1 & -1 & -1 \\
1 & -1 & -1 \\
1 & -1 & -1 \\
-1 & 1 & 1
\end{bmatrix}
\]

\[
= \begin{bmatrix}
2 & -2 & -2 \\
2 & -2 & 2 \\
-2 & 2 & -2 \\
-2 & 2 & 2
\end{bmatrix}
\]

Each column defines the weights for an output neuron
Example Architecture

- General Structure

\[
\begin{pmatrix}
2 & -2 & -2 \\
2 & -2 & 2 \\
-2 & 2 & -2 \\
-2 & 2 & 2
\end{pmatrix}
\]

\[
y_i = \begin{cases} 
1 & \text{if } y_{in_i} > 0 \\
0 & \text{if } y_{in_i} = 0 \\
-1 & \text{if } y_{in_i} < 0
\end{cases}
\]
Example Run 1

Try the first input pattern:

\[ s_1 = \begin{pmatrix} 1 & -1 & -1 & -1 \end{pmatrix} \quad f_1 = \begin{pmatrix} 1 & -1 & -1 \end{pmatrix} \]

\[ y_i = \begin{cases} 1 & \text{if } y_{\text{in}_i} > 0 \\ 0 & \text{if } y_{\text{in}_i} = 0 \\ -1 & \text{if } y_{\text{in}_i} < 0 \end{cases} \]

\[ y_{\text{in}_1} = 2 - 2 + 2 + 2 = 4 \text{ so } y_1 = 1 \]

\[ y_{\text{in}_2} = -2 + 2 - 2 - 2 = -4 \text{ so } y_2 = -1 \]

\[ y_{\text{in}_3} = -2 - 2 + 2 - 2 = -4 \text{ so } y_3 = -1 \]
Example Run II

- Try the second input pattern:

\[ s_2 = (-1 \ 1 \ -1 \ -1) \quad f_2 = (1 \ -1 \ 1) \]

\[ y_i = \begin{cases} 
1 & \text{if } y_{\text{in}_i} > 0 \\
0 & \text{if } y_{\text{in}_i} = 0 \\
-1 & \text{if } y_{\text{in}_i} < 0 
\end{cases} \]

\[ y_{\text{in}_1} = -2 + 2 + 2 + 2 = 4 \text{ so } y_1 = 1 \]

\[ y_{\text{in}_2} = 2 - 2 - 2 - 2 = -4 \text{ so } y_2 = -1 \]

\[ y_{\text{in}_3} = 2 + 2 + 2 - 2 = 4 \text{ so } y_3 = 1 \]
Example Run III

Try the Third input pattern:

\[ s_3 = (-1 \ -1 \ 1 \ -1) \quad f_3 = (-1 \ 1 \ -1) \]

\[
y_i = \begin{cases} 
1 & \text{if } y_{\text{in}_i} > 0 \\
0 & \text{if } y_{\text{in}_i} = 0 \\
-1 & \text{if } y_{\text{in}_i} < 0 
\end{cases}
\]

\[
y_{\text{in}_1} = -2 - 2 - 2 + 2 = -4 \quad \text{so } y_1 = -1
\]

\[
y_{\text{in}_2} = 2 + 2 + 2 - 2 = 4 \quad \text{so } y_2 = 1
\]

\[
y_{\text{in}_3} = 2 - 2 - 2 - 2 = -4 \quad \text{so } y_3 = -1
\]
Example Run IV

- Try the fourth input pattern:

\[ s_4 = (-1 \ -1 \ -1 \ 1) \ f_4 = (-1 \ 1 \ 1) \]

\[ y_i = \begin{cases} 
1 & \text{if } y_{\text{in}_i} > 0 \\
0 & \text{if } y_{\text{in}_i} = 0 \\
-1 & \text{if } y_{\text{in}_i} < 0 
\end{cases} \]

\[ y_{\text{in}_1} = -2 - 2 + 2 - 2 = -4 \text{ so } y_1 = -1 \]

\[ y_{\text{in}_2} = 2 + 2 - 2 + 2 = 4 \text{ so } y_2 = 1 \]

\[ y_{\text{in}_3} = 2 - 2 + 2 + 2 = 4 \text{ so } y_3 = 1 \]
Example Run V

- Try a non-training pattern

What do the 0’s mean?

\[ y_i = \begin{cases} 
1 & \text{if } y_{in_i} > 0 \\
0 & \text{if } y_{in_i} = 0 \\
-1 & \text{if } y_{in_i} < 0 
\end{cases} \]

\[ y_{in_1} = -2 - 2 + 2 + 2 = 0 \text{ so } y_1 = 0 \]

\[ y_{in_2} = 2 + 2 - 2 - 2 = 0 \text{ so } y_2 = 0 \]

\[ y_{in_3} = 2 - 2 + 2 - 2 = 0 \text{ so } y_3 = 0 \]
Example Run VI

- Try another non training pattern

What does this result mean?

\[
y_i = \begin{cases} 
1 & \text{if } y_{\text{in}_i} > 0 \\
0 & \text{if } y_{\text{in}_i} = 0 \\
-1 & \text{if } y_{\text{in}_i} < 0
\end{cases}
\]

\[
y_{\text{in}_1} = -2 + 2 - 2 - 2 = -4 \text{ so } y_1 = -1
\]

\[
y_{\text{in}_2} = 2 - 2 + 2 + 2 = 4 \text{ so } y_2 = 1
\]

\[
y_{\text{in}_3} = 2 + 2 - 2 + 2 = 4 \text{ so } y_3 = 1
\]
AM and Discrete Hopfield Network.

- Introduction
- Basic Concepts
- Linear Associative Memory (Hetero-associative)
- Hopfield’s Autoassociative Memory
- Performance Analysis for Recurrent Autoassociation Memory
- References and suggested reading
Hopfield’s Autoassociative Memory
(1982,1984)

- Recurrent/Dynamic Associative Memory
- Discrete-time and Asynchronous Update Mode
  (only one unit updates its activation at a time)

\[ w_{ij} = w_{ji} \]
(symmetric weights)

\[ w_{ii} = 0 \]
(no self connections)

Uses binary input vectors

Bipolar Binary Responses
Hopfield’s Autoassociative Memory

**Storage Algorithm**

To store a set of binary input vectors $s^{(m)}$

$$W = \left( \sum_{m=1}^{p} s^{(m)} s^{(m)\top} \right) - pI$$

$$w_{ij} = (1 - \delta_{ij}) \sum_{m=1}^{p} s_{i}^{(m)} s_{j}^{(m)}$$, where

$$\begin{align*}
\delta_{ij} & = 1 \quad \text{if } i = j \\
\delta_{ij} & = 0 \quad \text{if } i \neq j
\end{align*}$$

- In other words:

$$w_{ij} = \sum_{m=1}^{p} s_{i}^{(m)} s_{j}^{(m)}$$, for $i \neq j$$

$$w_{ii} = 0$$

- The diagonal elements set to be zero to avoid self-feedbacks

Only for bipolar binary response (Bipolar patterns $s = +1$ or $-1$)
Hopfield’s Autoassociative Memory

– The matrix only represents the correlation terms among the vector entries
– Invariant with respect to the sequence of storing patterns
– Additional autoassociations can be added at any time by superimposition
– Autoassociations can also be removed
Hopfield’s Autoassociative Memory

- Retrieval Algorithm
  - The output update rule can be expressed as:
  \[ v_i^{k+1} = \text{sgn} \left( \sum_{j=1}^{n} w_{ij} v_j^k \right) \]

  - Asynchronous and random update of neurons \( m, p, q \)

  First Update \( \mathbf{v}^1 = \begin{bmatrix} v_1^0 & v_2^0 & \ldots & v_m^0 \end{bmatrix} \)
  
  Second Update \( \mathbf{v}^2 = \begin{bmatrix} v_1^0 & v_2^0 & \ldots & v_m^0 & v_p^0 \end{bmatrix} \)
  
  Third Update \( \mathbf{v}^3 = \begin{bmatrix} v_1^0 & v_2^0 & \ldots & v_m^0 & v_p^0 & v_q^0 \end{bmatrix} \)
Hopfield’s Autoassociative Memory

**Example:** two autoassociations

\[
\begin{align*}
\mathbf{s}^{(1)} &= \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \mathbf{s}^{(2)} = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \\
\mathbf{W} &= \begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix} - 2I = \begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}\\
\mathbf{v}_i^{k+1} &= \text{sgn} \left( \sum_{j=1}^{n} w_{ij} v_j^k \right)
\end{align*}
\]

**input:** \( \mathbf{v}^0 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{v}^1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{v}^2 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{v}^3 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{v}^4 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{v}^5 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{v}^6 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \)

Neuron 1    Neuron 2    Neuron 3    Neuron 1    Neuron 2    Neuron 3

convergence
Hopfield’s Autoassociative Memory

Example: Autoassociation for numbers (Lippmann 1987)

Stored Patterns

A Distorted Input Pattern “6”
Hopfield’s Autoassociative Memory

**Example:** Autoassociation for numbers

- A Distorted Input Pattern “2”
- A Distorted Input Pattern “9”
Hopfield’s Autoassociative Memory

- IF Unipolar Binary Patterns \((s = 0\ or\ 1)\) are used
  - Scale and shift the association patterns

\[
\omega_{ij} = \left(1 - \delta_{ij}\right) \sum_{m=1}^{p} \left(2s_i^{(m)} - 1\right)\left(2s_j^{(m)} - 1\right)
\]

- In other words:

\[
\omega_{ij} = \sum_{m=1}^{p} \left(2s_i^{(m)} - 1\right)\left(2s_j^{(m)} - 1\right) \quad \text{for} \ i \neq j
\]

\[
\omega_{ii} = 0
\]
Hopfield’s Autoassociative Memory

- **Performance Consideration**
  - Hopfield autoassociative memory also referred to as "Error Correcting Decoder"
  - Given an input equal to the stored memory with random error, it produces as output the original memory that is close to the input. Example:
Hopfield’s Autoassociative Memory

Performance Consideration

- Let us assume that \( s^{(m')} \) has been stored in the memory as one of \( p \) patterns.
- This pattern is now at the memory input. The activation value of the \( i \)-th neuron for retrieval of pattern \( s^{(m')} \) is:

\[
net_i = \sum_{j=1}^{n} w_{ij} s_j^{(m')}
\]

\( w_{ij} \)

If the effect of the diagonal is neglected

\[
\sum_{j=1}^{n} \sum_{m=1}^{p} s_i^{(m')} s_j^{(m')} s_j^{(m')}
\]

If \( s^{(m')} \) and \( s^{(m)} \) are statistically independent (also when orthogonal) the product vanishes (is zero) Otherwise, in the limit case, the product will reach \( n \) for both vectors being identical (note that it’s the dot product with entries of vectors = +1 or – 1)

\[
\approx s_i^{(m')} \tilde{n}, \text{where } \tilde{n} \leq n
\]

The major overlap case.

i.e. \( s^{(m')} \) does not produce any update and is therefore stable
Hopfield’s Autoassociative Memory

Performance Consideration

The equilibrium state for the retrieval pattern \(s^{(m')}\) is considered \(\text{net} = Ws^{(m')}\)

\[
= \left( \sum_{m=1}^{p} s^{(m)}s^{(m)t} - pI \right) s^{(m')}
\]

1. If the stored prototypes are statistically independent or orthogonal, and \(n>p\)

i.e.

\[
s^{(i)t}s^{(j)} = 0 \quad \text{for } i \neq j
\]

\[
s^{(i)t}s^{(j)} = n \quad \text{for } i = j
\]

then

\[
\text{net} = \left( s^{(1)t}s^{(1)} + s^{(2)t}s^{(2)} + \ldots + s^{(p)t}s^{(p)} - pI \right) s^{(m')}
\]

Since it is assumed that \(n>p\), then the network will be in equilibrium state \(s^{(m')}\)
Hopfield’s Autoassociative Memory

2. If the stored prototypes are not statistically independent or orthogonal

$$\text{net} = n \ s^{(m')} - p \ s^{(m')} + \sum_{m \neq m'}^{p} (s^{(m)} s^{(m)t}) s^{(m')}$$

Equilibrium state term (as on the previous slide)

$$\eta = \sum_{m \neq m'}^{p} (s^{(m)} s^{(m)t}) s^{(m')} = (W - s^{(m')} s^{(m')} + I) s^{(m')}$$

- When the $i$-th element of the noise vector is larger than $(n - p) s^{(m')}_i$ and has opposite sign, then $s^{(m')}_i$ will not be the network’s equilibrium.

- The noise term obviously increases for an increased number of stored patterns, and also becomes relatively significant when the factor $(n-p)$ decreases.
AM and Discrete Hopfield Network.

- Introduction
- Basic Concepts
- Linear Associative Memory (Hetero-associative)
- Hopfield’s Autoassociative Memory
- Performance Analysis for Recurrent Autoassociation Memory
- References and suggested reading
Performance Analysis for Recurrent Autoassociation Memory

Example Let us look at the convergence properties of an autoassociative memory that stores the two vectors \( \mathbf{s}^{(1)} \) and \( \mathbf{s}^{(2)} \)

\[
\mathbf{s}^{(1)} = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{s}^{(2)} = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}
\]

\[
\mathbf{w} = \left( \sum_{n=1}^{p} \mathbf{s}^{(m)} \mathbf{s}^{(m)r} \right) - p \mathbf{I}
\]

\[
\mathbf{w} = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} - 2 \mathbf{I} = \begin{bmatrix} 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & -2 \\ -2 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \end{bmatrix}
\]

- When the net. is allowed to update, it evolves through certain states which are shown on the state transition diagram on the next slide.
- States are vertices of the 4D cube.
- Only transitions between states have been marked on the graph; the self-sustaining or stable state transitions have been omitted for clarity of the picture.
Performance Analysis for Recurrent Autoassociation Memory

- Since only asynchronous transitions are allowed for autoassociative recurrent memory, every vertex is connected by an edge only to the neighbouring vertex differing by a single bit.
- Thus there are only 4 transition paths connected to every state. The paths indicate directions of possible transitions.

\[
\begin{align*}
\begin{bmatrix}
-1 \\
1 \\
-1 \\
1 \\
\end{bmatrix}
\end{align*}
\]

Different update order will lead to different results.
Performance Analysis for Recurrent Autoassociation Memory

- Transitions are toward lower energy values. The energy values for each state have been marked in parentheses at each vertex.

Let's consider some sample transitions. Starting at $v^{(0)} = [-1 \ 1 \ -1 \ 1]^t$ and with nodes updating asynchronously in ascending order (highest entry of the vector first), we have the state sequence

$$v^1 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \quad v^2 = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \quad v^3 = v^4 = \ldots = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

As it has been marked on the state diagram.

- The actual convergence is toward a negative image of the stored pattern $s^{(2)}$.
- The initial pattern had no particular similarity to any of the stored patterns and apparently did not fall into any closeness category of patterns.

Let us look at a different starting vector $v^{(0)} = [-1 \ 1 \ 1 \ 1]^t$, the transition starting in ascending order leads to $s^{(1)}$ in single step.
Performance Analysis for Recurrent Autoassociation Memory

\[ v^1 = v^0 \quad v^2 = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \quad v^3 = v^2 \quad v^4 = v^3 \quad \text{etc.} \]

• You can also verify that by carrying out the transitions initialized at \( v^{(0)} \), but carried out in descending node order, pattern \( s^{(2)} \) can be recovered.
Advantages and Limits for Associative Recurrent Memories

- **Limits**
  - Limited capability
  - Converge to spurious memories (states).

- **Advantage**
  - The recurrences through the thresholding layer of processing neurons (threshold functions) tend to eliminate noise superimposed on the initializing input vector.
AM and Discrete Hopfield Network.

- Introduction
- Basic Concepts
- Linear Associative Memory (Hetero-associative)
- Hopfield’s Autoassociative Memory
- Performance Analysis for Recurrent Autoassociation Memory
- 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