# ASSIGNMENT-1

## Bidirectional Associative Memory (BAM)

M. SATYA VARDHAN
VU22CSEN0200205

### Associative Memory

→ Associative memory refers to the ability to recall facts, information, (or) patterns by invoking chains of associations within a complex web of embedded knowledge.

→ It is inspired by how human memory operates:

a) Chains of thought are generated through conscious (or) unconscious reasoning.

b) A simple cue (eg: a fragrance, a name, (or) a partial input) can evoke an entire experience in complete richness and detail within fraction of a second.

→ Some key Characteristics

a) Associative Recall

→ Memories are retrieved based on similarity (or) association rather that explicit indexing.

→ Eg:- The name of a friend might trigger emotions (or) related experience.

b) Content-Addressable Memory (CAM)

→ Associative memory systems are often referred to as content-addressable memories because they retrieve stored

information based on partial (or) noisy input (content) rather than a specific address.

→ Eg:- Given a distorted (or) incomplete input vector, the system recalls the closest matching memory vector.

c) Error Correction

→ Associative memory systems inherently possess error correction capabilities.

→ They can recover complete patterns from noisy, incomplete (or) distorted inputs.

## Models of Associative Memory

1) Hopfield Network - A fully connected, symmetric network that uses an energy function to ensure convergence to stable states. It encodes memories using Hebbian learning and retrieves them through iterative updates.

2) Bidirectional Associative Memory (BAM) - It is a 2-layer heteroassociative memory system that performs bidirectional recall between input and output spaces. It ensures stability using an energy function derived from the weight matrix.

3) Boltzmann Machine - A stochastic extension of Hopfield networks that uses simulated annealing for relaxation and gradient descent based learning.

4) Brain-State-in-a-Box (BSB): It extends the linear associator

model and uses a piecewise linear signal function to constrain states within a hypercube.

## Bidirectional Associative Memory (BAM)

→ "Bidirectional Associative Memory (BAM)" is a two-layer neural network introduced by Bart Kosko. It is a heteroassociative memory system that maps input vector in one space to output vector in another space and vice-versa. BAM operates through bidirectional relaxation, where information flows back and forth between 2 layers until the system stabilizes.

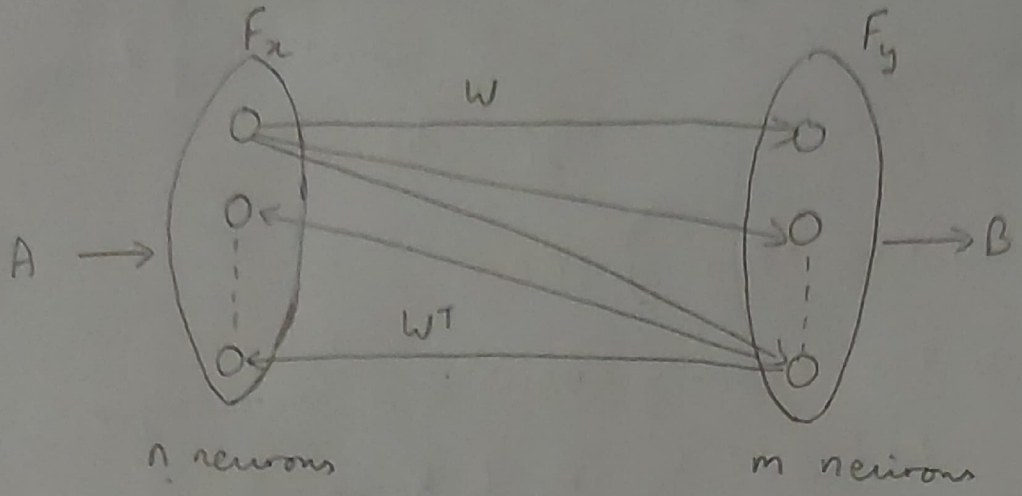
→ The "bidirectional" aspect refers to the ability to operate in both forward (layer A to layer B) and backward (layer B to layer A) directions, iteratively refining the output until convergence. This makes BAM particularly useful for tasks requiring paired associations, such as pattern recognition (or) memory retrieval across different domains.

## BAM Architecture

→ It is a 2-layer structure consisting of :-

a) Input layer ($F_x$) - Contains n neurons

b) Output layer ($F_y$) - Contains m neurons.




→ The two layers are fully connected with bidirectional weights:-

a) Forward Weight (W): From $F_x$ to $F_y$

b) Reverse Weight ($W^T$): From $F_y$ to $F_x$

→ There are 3 main varieties of BAM

a) Binary (0 (or) 1)

b) Bipolar (-1 (or) +1)

c) Continuous

→ The weights are calculated using the Hebb rule and the formulae's differ when the neurons are in binary, bipolar, continuous state.

a) Binary State

→ For binary input vector the weight matrix $W = \{w_{ij}\}$ is given as

$$w_{ij} = \sum_p (2s_i(p) - 1)(2t_j(p) - 1)$$

here s - input t - target for the input.

b) Bipolar state

→ For bipolar input vector the weight matrix $W = \{w_{ij}\}$ is given as

$$w_{ij} = \sum_{p} (s_i(p))(t_j(p))$$

here s - input
t - target for the input.

Activation function

→ For binary input vector a step function is used:-

$$y_j = \begin{cases} 1 & \text{if } y_{in(j)} > 0 \\ y_j & \text{if } y_{in(j)} = 0 \\ 0 & \text{if } y_{in(j)} < 0 \end{cases} \rightarrow \text{For } F_y \text{ layer [output]}$$

$$x_i = \begin{cases} 1 & \text{if } x_{in(i)} > 0 \\ x_j & \text{if } x_{in(i)} = 0 \\ 0 & \text{if } x_{in(i)} < 0 \end{cases} \rightarrow \text{For } F_x \text{ layer [output]}$$

→ For bipolar input vector a step function is used:-

$$y_j = \begin{cases} 1 & \text{if } y_{in(j)} > \theta_j \\ y_j & \text{if } y_{in(j)} = \theta_j \\ -1 & \text{if } y_{in(j)} < \theta_j \end{cases} \rightarrow \text{For } F_y \text{ layer [output]}$$

$$x_i = \begin{cases} 1 & \text{if } x_{in(i)} > \theta_i \\ x_j & \text{if } x_{in(i)} = \theta_i \\ -1 & \text{if } x_{in(i)} < \theta_i \end{cases} \rightarrow \text{For } F_x \text{ layer [output]}$$

here $\theta$ can be any value between $-1, 0, +1$.

c) Continuous State

→ For a continuous input vector the weight matrix $W = \{w_{ij}\}$ is given as

$$w_{ij} = \sum_{p} (2s_i(p) - 1)(2t_j(p) - 1)$$

here $s \rightarrow$ input $t \rightarrow$ target for input.

→ Here we use sigmoid activation function

$$f(y_{in_j}) = \frac{1}{1 + e^{-y_{in_j}}}$$

Relaxation Procedure

→ Information flows back and forth between the 2 layers through alternating forward and reverse passes. The system iteratively updates neuron states in both layers until it reaches a stable equilibrium.

→ Stability is guaranteed by an energy function derived from the weight matrix:

$$E(A, B) = -A^T W B$$

here $A \rightarrow$ State vector of $F_x$

$B \rightarrow$ State vector of $F_y$

$W \rightarrow$ Weight matrix.

→ The energy function decreases monotonically over time, ensuring convergence to a stable state.

# BAM Algorithm

→ BAM operates as a two-layer neural network that associates pairs of patterns (X and Y) and retrieves one given the other, in either direction. The algorithm consists of :-

a) Training - A one-time computation of the weight matrix W to encode the pattern pairs.

b) Recall - An iterative process that uses W and its transpose $W^T$ to converge to a stored pair from a given input.

→ Phase 1 : Training Algorithm

a) The training phase constructs the weight matrix W to store the associations between X and Y patterns using Hebbian learning. It's a single-pass process without iterative optimization.

b) Inputs →

A set of P pattern pairs: $(X_1, Y_1), (X_2, Y_2), \ldots, (X_P, Y_P)$

here $X_i$ : Vector of length n (layer $F_x$ neurons)
$Y_i$ : Vector of length m (layer $F_y$ neurons)
} typically bipolar representation is taken.

Steps :-

(1) Initialize the weight matrix → create an n×m matrix W and set all elements to 0

$$W = 0_{n \times m}$$

ii) Compute the Weight Matrix: For each pattern pair $(X_i, Y_i)$ calculate the outer product and sum it into W:

$$W = \sum_{i=1}^{P} X_i^T Y_i$$

here $X_i^T$ : Transpose of $X_i$

$Y_i$ : A $m \times 1$ column vector

$X_i^T Y_i$ : An $n \times m$ matrix representing the contribution of pair i.

iii) Output the weight matrix : The final W is stored and used for recall. No further adjustments are made.

→ Phase 2 : Recall Algorithm

a) The recall phase retrieves a stored pattern pair by iteratively updating the states of layer A and layer B until convergence. It can start with an input to either layer.

b) Inputs →

→ Trained weight matrix W (size $n \times m$)

→ Initial pattern :

- Either X' (size n), a noisy (or) partial version of a stored $X_i$ (or)
- Y' (size m) a noisy (or) partial version of a stored $Y_i$

→ Threshold function $\phi$ (eg: $\phi(z) = 1$ if $z > 0$, $-1$ if $z < 0$, & unchanged (or) default if $z = 0$).

Steps →

(i) Initialize the layers:

→ If starting with $X'$:

Set layer A state to $X^{(0)} = X'$

set layer B state to $Y^{(0)} = 0$ (or any arbitary value)

→ If starting with $Y'$:

set layer B state to $Y^{(0)} = Y'$

set layer A state to $X^{(0)} = 0$

(ii) Iterate until convergence: For iteration $t = 1, 2, \ldots$

→ Forward Pass (A to B):

a) Compute the new state of layer B:

$$Y^{(t)} = \phi(X^{(t-1)} W)$$

b) Apply $\phi$ element-wise to the result.

→ Backward Pass (B to A):

a) Compute the new state of Layer A:

$$X^{(t)} = \phi(Y^{(t)} W^T)$$

b) Apply $\phi$ element-wise.

→ Check Convergence:

a) If $X^{(t)} = X^{(t-1)}$ & $Y^{(t)} = Y^{(t-1)}$ (no change), stop.

b) Otherwise, increment $t$ and repeat.

11) Updates can be synchronous (all neurons at once) (or) asynchronous (one neuron at a time), through synchronous is standard.

→ Output the result: The final $(x^{(t)}, y^{(t)})$ is the recalled pattern pair, ideally matching a stored $(x_i, y_i)$.

## Pseudocode of BAM

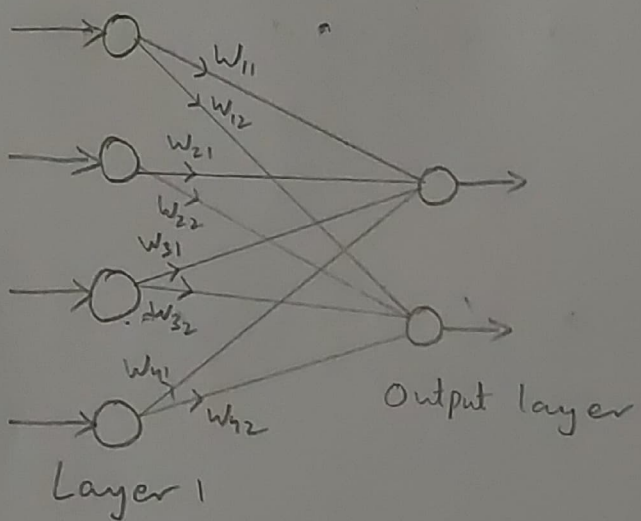
```
# Training Phase
Input : Pairs (X_1, Y_1), ...., (X_P, Y_P)
Initialize : W ← zeros(n, m)
For  i=1  to  P:
    W ← W + X_i^T * Y_i
Output : W

# Recall Phase
Input : W, Initial X' (or Y')
If X' is given:
    X ← X', Y ← zeros(m)
Else if Y' is given:
    Y ← Y', X ← zeros(n)
Repeat :
    Y_new ← sign (X * W)
    X_new ← sign (Y_new * W^T)
    If X_new = X and Y_new = Y
        Break
    X ← X_new, Y ← Y_new
```

Output : (X, Y)

## Example of Bidirectional Associative Memory

→ Train a Bidirectional Associative network to store the input vector $s = (s1, s2, s3, s4)$ to the output vector $t = (t1, t2)$.

→ The training input-target output vector pairs are in Binary form.




| Input/Target | S1 | S2 | S3 | S4 | T1 | T2 |
|---|---|---|---|---|---|---|
| 1st | 1 | 0 | 0 | 0 | 0 | 1 |
| 2nd | 1 | 1 | 0 | 0 | 0 | 1 |
| 3rd | 0 | 0 | 0 | 1 | 1 | 0 |
| 4th | 0 | 0 | 1 | 1 | 1 | 0 |

→ Obtain the weight vector in bipolar form.

→ Equation to find weight matrix for training (input) and target (output) in bipolar form:

$$W_{ij} = \sum_{p=1}^{4} (2s_i(p) - 1)(2t_j(p) - 1)$$

$W_{11} = (2*1-1)*(2*0-1) + (2*1-1)(2*0-1) + (2*0-1)(2*1-1) + (2*0-1)(2*1-1)$

$= -1-1-1-1 = -4$

$W_{12} = (2*1-1)*(2*1-1) + (2*1-1)(2*1-1) + (2*0-1)(2*0-1) + (2*0-1)(2*0-1)$

$= 1+1+1+1 = 4$

$W_{21} = (2*0-1)*(2*0-1) + (2*1-1)(2*0-1) + (2*0-1)(2*1-1) + (2*0-1)(2*1-1)$

$= 1-1-1-1 = -2$

$W_{22} = (2*0-1)*(2*1-1) + (2*1-1)(2*1-1) + (2*0-1)(2*0-1) + (2*0-1)(2*0-1)$

$= -1+1+1+1 = 2$

$W_{31} = (2*0-1)*(2*0-1) + (2*0-1)(2*0-1) + (2*0-1)(2*1-1) + (2*1-1)(2*1-1)$

$= 1+1-1+1 = 2$

$W_{32} = (2*0-1)*(2*1-1) + (2*0-1)(2*1-1) + (2*0-1)(2*0-1) + (2*1-1)(2*0-1)$

$= -1-1+1-1 = -2$

$W_{41} = (2*0-1)*(2*0-1) + (2*0-1)(2*0-1) + (2*1-1)(2*1-1) + (2*1-1)(2*1-1)$

$= 1+1+1+1 = 4$

$$W_{42} = (2*0-1)*(2*1-1)+(2*0-1)(2*1-1)+(2*1-1)(2*0-1)+$$
$$(2*1-1)(2*0-1)$$

$$= -1-1-1-1 = -4$$

∴ Final weight matrix

$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} -4 & +4 \\ -2 & +2 \\ +2 & -2 \\ +4 & -4 \end{bmatrix}$$

Applications of BAM

1) Pattern Completion - BAM can complete (or) reconstruct patterns when given partial (or) noisy inputs. It retrieves the closest stored association even if the input is distorted (or) incomplete. Eg:- Consider a corrupted image (eg a pixelated (or) noisy version of an object). BAM can reconstruct the original image by associating the noisy input with the closest stored memory.

2) Speech Perception - BAM can associate sounds (input) with their corresponding meanings (or) words (output). This is particularly useful in speech recognition systems where ambiguous (or) noisy audio signals need to be mapped to specific linguistic interpretations. Eg:- A noisy (or) partially heard word can be associated with its correct meaning (or) transcription using BAM's error correction capabilities.

3) Multistable Perception - BAM can resolve ambiguous sensory inputs into stable interpretations. This is useful in scenarios where sensory data is inherently multistable. Eg:- Given an ambiguous visual stimulus (eg the Necker cube), BAM can stabilize the perception by associating it with one of the possible interpretations.

Limitations of Bidirectional Associative Memory (BAM)

1) Capacity Limitation → BAM has a limited capacity to store associations without interference. The maximum number of associations that can be stored reliably is approximately $0.15 \times \min(n, m)$ where $n, m$ are the dimensions of the input & output layers. Beyond this limit, the system becomes unstable, and the ability to recall associations deteriorates.

2) Spurious Memories → There are undesired stable states that do not correspond to any stored association. These states can emerge due to interference between stored associations (or) during the relaxation process. The system may converge to spurious memories instead of the desired associations, especially when the input probe is noisy (or) ambiguous.

3) Memory Annihilation → Memory annihilation occurs when encoding too many equidistant vectors (structured maps). Under certain conditions, all stored associations are lost and system fails to recall memory. This pathological case severely limits the usability of BAM in applications requiring large-scale data.