Table of Contents


```
% FML Lab_Q9: Multiple Encoding Strategy (BAM/Hetero-associative memory)
% Author: ARISH
% Notes:
% - Uses superposition (correlation matrix) W = sum_k (Y_k * X_k^T)
% - Retrieval: Y_hat = sgn(W * X_query), optional bidirectional refinement
% - Handles training retrieval and noised queries (bit flips)
% - Assumption: Y2 corrected to 9 elements by dropping its last element

clear; clc;
```

Define training patterns (±1)

X: 13-bit inputs (columns)

```
X = [ ...
    1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1;  % X1
    1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1;  % X2
    1  1  1 -1 -1 -1  1  1  1 -1 -1  1  1;  % X3
    1  1  1 -1 -1 -1  1  1  1 -1 -1 -1 -1   % X4
]';

% Y: make all 9-bit by correcting Y2 (drop last element)
Y1 = [ 1  1  1  1 -1 -1 -1  1  1 ]';
Y2 = [ 1  1  1 -1 -1 -1  1  1  1 ]';   % corrected: dropped final -1
Y3 = [ 1  1 -1 -1  1  1 -1 -1  1 ]';
Y4 = [ 1 -1  1 -1  1 -1  1 -1  1 ]';
Y  = [Y1 Y2 Y3 Y4];

assert(size(X,1)==13, 'X must be 13-bit.');
assert(size(Y,1)==9,   'Y must be 9-bit.');
assert(size(X,2)==size(Y,2), 'Same number of patterns in X and Y.');

P = size(X,2);    % number of pattern pairs
```

Multiple encoding (superposition)

Correlation matrix memory (hetero-associative)

```
W = zeros(size(Y,1), size(X,1));
for k = 1:P
```

```
    W = W + Y(:,k) * X(:,k)';  % outer product and sum
end
```

Helper functions

```
sgn = @(v) 2*(v>=0) - 1;    % maps v>=0 -> +1, v<0 -> -1
bitflip = @(vec, idx) flipBits(vec, idx);
hamming = @(a,b) sum(a~=b);
```

Retrieval on training patterns

```
fprintf('Retrieval on clean training X:\n');
for k = 1:P
    y_hat = sgn(W * X(:,k));
    fprintf('Pair %d: Hamming(Y_hat, Y_true) = %d\n', k, hamming(y_hat,
Y(:,k)));
end
```

```
Retrieval on clean training X:
Pair 1: Hamming(Y_hat, Y_true) = 0
Pair 2: Hamming(Y_hat, Y_true) = 0
Pair 3: Hamming(Y_hat, Y_true) = 0
Pair 4: Hamming(Y_hat, Y_true) = 0
```

Optional bidirectional refinement (BAM iteration)

Iterate a few steps: Y <- sgn(W*X), then X <- sgn(W"*Y)

```
fprintf('\nBidirectional refinement from clean X:\n');
for k = 1:P
    y_hat = sgn(W * X(:,k));
    x_hat = sgn(W' * y_hat);
    y_hat2 = sgn(W * x_hat);
    fprintf('Pair %d: Hamming after 1 iteration: Y=%d, X=%d\n', ...
        k, hamming(y_hat2, Y(:,k)), hamming(x_hat, X(:,k)));
end
```

```
Bidirectional refinement from clean X:
Pair 1: Hamming after 1 iteration: Y=0, X=0
Pair 2: Hamming after 1 iteration: Y=0, X=0
Pair 3: Hamming after 1 iteration: Y=0, X=0
Pair 4: Hamming after 1 iteration: Y=0, X=0
```

Noised retrieval experiments

Flip r random bits in X and (optionally) in Y, then retrieve

```
rng(7);  % reproducible
rX = 2;  % number of bits to flip in X query
```

```
rY = 0;  % number of bits to flip in Y (for reverse retrieval tests)

fprintf('\nNoised retrieval (flip %d bits in X):\n', rX);
for k = 1:P
    idxX = randperm(size(X,1), rX);
    x_noised = bitflip(X(:,k), idxX);
    y_hat = sgn(W * x_noised);
    fprintf('Pair %d: Hamming(Y_hat, Y_true) = %d\n', k, hamming(y_hat,
Y(:,k)));
end

% Optional reverse test: start from noised Y and recover X
fprintf('\nReverse noised retrieval (flip %d bits in Y):\n', rY);
for k = 1:P
    if rY>0
        idxY = randperm(size(Y,1), rY);
        y_noised = bitflip(Y(:,k), idxY);
    else
        y_noised = Y(:,k);
    end
    x_hat = sgn(W' * y_noised);
    fprintf('Pair %d: Hamming(X_hat, X_true) = %d\n', k, hamming(x_hat,
X(:,k)));
end
```

Noised retrieval (flip 2 bits in X):

Utility: flip specified bit indices

```
function out = flipBits(in, idx)
    out = in;
    out(idx) = -out(idx);   % flip ±1 by multiplying -1
end
```

```
Pair 1: Hamming(Y_hat, Y_true) = 0
Pair 2: Hamming(Y_hat, Y_true) = 0
Pair 3: Hamming(Y_hat, Y_true) = 2
Pair 4: Hamming(Y_hat, Y_true) = 0

Reverse noised retrieval (flip 0 bits in Y):
Pair 1: Hamming(X_hat, X_true) = 0
Pair 2: Hamming(X_hat, X_true) = 0
Pair 3: Hamming(X_hat, X_true) = 0
Pair 4: Hamming(X_hat, X_true) = 0
```

Published with MATLAB® R2025a