Practical 5

Code : -

```
import numpy as np


def bipolarize(vector):
    """Convert binary (0,1) vectors to bipolar (-1,1)."""
    return np.where(vector == 0, -1, 1)


def train_bam(X, Y):
    """Train the Bidirectional Associative Memory (BAM)."""
    W = np.zeros((X.shape[1], Y.shape[1]))
    for x, y in zip(X, Y):
        W += np.outer(x, y)  # Hebbian learning rule
    return W


def recall_bam(W, input_vector, direction='X-to-Y'):
    """Recall function for BAM."""
    output = np.sign(np.dot(input_vector, W))
    return output


# Define training pairs
X_train = np.array([[1, 0, 1], [0, 1, 1]])  # Binary input vectors
Y_train = np.array([[0, 1, 1], [1, 0, 1]])  # Associated binary output vectors


# Convert to bipolar representation
X_train = np.array([bipolarize(x) for x in X_train])
Y_train = np.array([bipolarize(y) for y in Y_train])


# Train BAM
W = train_bam(X_train, Y_train)
```

```
# Test recall
X_test = X_train[0]
Y_test = Y_train[0]

# Recall Y from X
Y_recalled = recall_bam(W, X_test)

# Recall X from Y
X_recalled = recall_bam(W.T, Y_test)

print("Original X:", X_test)
print("Recalled Y:", Y_recalled)
print("Original Y:", Y_test)
print("Recalled X:", X_recalled)
```

Output : -

```
Original X: [ 1 -1  1]
Recalled Y: [-1.  1.  1.]
Original Y: [-1  1  1]
Recalled X: [ 1. -1.  1.]
```