BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Shreesh S Patil
UID no.	2023300170
Experiment No.	To implement unsupervised algorithm ● Kohonen Self Organizing Maps ● LVQ networks

Program	
THEORY :	# Unsupervised Learning Unsupervised learning is a machine-learning approach where algorithms identify hidden patterns and structures in unlabeled data. Unlike supervised learning, no target outputs are provided; the system independently discovers clusters, relationships, and anomalies based on data characteristics. ## Key Concepts ● **Clustering:** Groups similar data points based on feature similarity ● **Association:** Discovers dependency relationships among variables ● **Dimensionality Reduction:** Reduces the number of input variables while preserving essential information # Kohonen Self-Organizing Maps (KSOM / SOM) Kohonen Self-Organizing Maps are unsupervised neural networks designed for visualization and topological mapping of high-dimensional data into lower-dimensional (typically 2-D) grids, preserving neighborhood structure. Proposed by Teuvo Kohonen, they are inspired by biological sensory-mapping mechanisms. ## Working Principle 1. **Initialization:** Create a grid of nodes, each with a randomly initialized weight vector. 2. **Competition:** For each input vector, compute distances and identify

	the **Best Matching Unit (BMU)** (node closest to the input). 3. **Cooperation:** Neighboring nodes around the BMU are selected for simultaneous adjustment. 4. **Adaptation:** BMU and neighbors update their weights to move closer to the input vector (Repeated over iterations) ## Linear Vector Quantization (LVQ) Linear Vector Quantization is a supervised classification algorithm related to SOMs, but uses labeled training data. LVQ learns prototype vectors representing each class and refines them to improve class boundaries. Training Procedure 1. Initialization: Define prototype vectors, each associated with a class label. 2. Classification Step: For each labeled input, identify the nearest prototype. 3. Weight Update: ○ If class(label) = class(prototype): move prototype toward input ○ If class(label) ≠ class(prototype): move prototype away from input Result: Prototypes converge to represent class centers, enabling effective classification of future inputs.

PROGRAM:	import numpy as np
	
	def kohonen_som():
	# Step 0: Initialize weights, inputs, and parameters
	# The four input vectors to be clustered
	input_vectors = np.array([
	[0, 0, 1, 1],
	[1, 0, 0, 0],
	[0, 1, 1, 0],
	[0, 0, 0, 1]

```
    ])

    # Initial weight matrix
    weights = np.array([
        [0.2, 0.9],
        [0.4, 0.7],
        [0.6, 0.5],
        [0.8, 0.3]
    ], dtype=float)

    # Initial learning rate
    learning_rate = 0.5
    num_epochs = 10

    print("--- Initial State ---")
    print(f"Initial Learning Rate (alpha): {learning_rate}")
    print("Initial Weight Matrix:\n", weights)
    print("-" * 25, "\n")

    # --- Training Loop for 10 Epochs ---
    for epoch in range(1, num_epochs + 1):

        # Present each input vector to the network
        for x in input_vectors:

            # Step 2: Calculate squared Euclidean distance
            distances = [np.sum((x - weights[:, j])**2) for j in
range(weights.shape[1])]

            # Step 3: Find winner cluster (min distance)
            winning_cluster_index = np.argmin(distances)

            # Step 4: Update the weights of the winner unit
            winning_weights = weights[:, winning_cluster_index]
            weights[:, winning_cluster_index] += learning_rate * (x -
winning_weights)

        # Print matrix after each epoch
```

```
        print(f"--- Epoch {epoch} Complete ---")
        print("Updated Weight Matrix:\n", np.round(weights, 4))

        # Update the learning rate
        learning_rate *= 0.5
        print(f"New Learning Rate for Epoch {epoch + 1}: {learning_rate}")
        print("-" * 25, "\n")

    # --- Final Clustering ---
    print("--- Final Clustering Results ---")
    print("Assigning each input vector to the nearest cluster:\n")

    cluster1_vectors, cluster2_vectors = [], []

    for i, x in enumerate(input_vectors):
        distances = [np.sum((x - weights[:, j])**2) for j in
range(weights.shape[1])]
        winner = np.argmin(distances)
        print(f"Vector {i + 1} {x} is closest to Cluster {winner + 1}")

        if winner == 0:
            cluster1_vectors.append(str(x))
        else:
            cluster2_vectors.append(str(x))

    print(f"\nCluster 1 contains vectors: {', '.join(cluster1_vectors)}")
    print(f"Cluster 2 contains vectors: {', '.join(cluster2_vectors)}")

kohonen_som()
```

RESULT:

```
--- Initial State ---
Initial Learning Rate (alpha): 0.5
Initial Weight Matrix:
 [[0.2 0.9]
 [0.4 0.7]
 [0.6 0.5]
 [0.8 0.3]]
-------------------------

--- Epoch 1 Complete ---
Updated Weight Matrix:
 [[0.025 0.95 ]
 [0.3   0.35 ]
 [0.45  0.25 ]
 [0.725 0.15 ]]
New Learning Rate for Epoch 2: 0.25
-------------------------

--- Epoch 2 Complete ---
Updated Weight Matrix:
 [[0.0105 0.9625]
 [0.3141 0.2625]
 [0.518  0.1875]
 [0.6965 0.1125]]
New Learning Rate for Epoch 3: 0.125
-------------------------

--- Epoch 3 Complete ---
Updated Weight Matrix:
 [[0.0071 0.9672]
 [0.3198 0.2297]
 [0.5521 0.1641]
 [0.6873 0.0984]]
New Learning Rate for Epoch 4: 0.0625
```

```
--- Epoch 4 Complete ---
Updated Weight Matrix:
 [[0.0058 0.9692]
 [0.3221 0.2153]
 [0.5684 0.1538]
 [0.6837 0.0923]]
New Learning Rate for Epoch 5: 0.03125
------------------------

--- Epoch 5 Complete ---
Updated Weight Matrix:
 [[0.0053 0.9702]
 [0.3231 0.2086]
 [0.5764 0.149 ]
 [0.6822 0.0894]]
New Learning Rate for Epoch 6: 0.015625
------------------------

--- Epoch 6 Complete ---
Updated Weight Matrix:
 [[0.005  0.9707]
 [0.3236 0.2053]
 [0.5803 0.1467]
 [0.6815 0.088 ]]
New Learning Rate for Epoch 7: 0.0078125
------------------------

--- Epoch 7 Complete ---
Updated Weight Matrix:
 [[0.0049 0.9709]
 [0.3238 0.2037]
 [0.5823 0.1455]
 [0.6811 0.0873]]
New Learning Rate for Epoch 8: 0.00390625
```

```
--- Epoch 8 Complete ---
Updated Weight Matrix:
 [[0.0049 0.971 ]
 [0.3239 0.2029]
 [0.5832 0.145 ]
 [0.681  0.087 ]]
New Learning Rate for Epoch 9: 0.001953125
------------------------

--- Epoch 9 Complete ---
Updated Weight Matrix:
 [[0.0048 0.9711]
 [0.324  0.2025]
 [0.5837 0.1447]
 [0.6809 0.0868]]
New Learning Rate for Epoch 10: 0.0009765625
------------------------

--- Epoch 10 Complete ---
Updated Weight Matrix:
 [[0.0048 0.9711]
 [0.324  0.2023]
 [0.5839 0.1445]
 [0.6808 0.0867]]
New Learning Rate for Epoch 11: 0.00048828125
------------------------

--- Final Clustering Results ---
Assigning each input vector to the nearest cluster:

Vector 1 [0 0 1 1] is closest to Cluster 1
Vector 2 [1 0 0 0] is closest to Cluster 2
Vector 3 [0 1 1 0] is closest to Cluster 1
Vector 4 [0 0 0 1] is closest to Cluster 1

Cluster 1 contains vectors: [0 0 1 1], [0 1 1 0], [0 0 0 1]
Cluster 2 contains vectors: [1 0 0 0]
```

LVQ	
PROGRAM:	import numpy as np def lvq_network_auto_converge(tol=1e-4, max_epochs=500, learning_rate=0.1):

```
  weights = np.array([
     [0, 0, 1, 1],
     [1, 0, 0, 0]
  ], dtype=float)

  prototype_classes = [1, 2]

  training_vectors = np.array([
     [0, 0, 0, 1],
     [1, 1, 0, 0],
     [0, 1, 1, 0]
  ])
  target_classes = [2, 1, 1]

  print("--- Initial State ---")
  print(f"Learning Rate (alpha): {learning_rate}")
  print("Initial Weight Matrix:\n", weights)
  print(f"Classes: {prototype_classes}")
  print("-" * 40, "\n")

  # --- Training Loop ---
  for epoch in range(1, max_epochs + 1):
     print(f"Epoch {epoch} starting...")
     prev_weights = weights.copy()

     # Process each training vector
     for i, x in enumerate(training_vectors):
        target_class = target_classes[i]

        # Step 1: Compute distances
        distances = [np.sum((x - w)**2) for w in weights]
        winner_index = np.argmin(distances)
        winner_class = prototype_classes[winner_index]

        print(f" Training Vector {i+1}: {x}, Target Class={target_class}")
        print(f" Distances: {np.round(distances, 4)} → Winner =
{winner_index+1} (Class {winner_class})")
```

```
        # Step 2: LVQ update
        if target_class == winner_class:
            print(" Classes match : moving closer")
            weights[winner_index] += learning_rate * (x -
weights[winner_index])
        else:
            print(" Classes differ : moving away")
            weights[winner_index] -= learning_rate * (x -
weights[winner_index])

        print(f" Updated {winner_index+1}:
{np.round(weights[winner_index], 4)}")

    # --- Convergence Check ---
    max_change = np.max(np.abs(weights - prev_weights))
    print(f"--- Epoch {epoch} Complete ---")
    print("Updated Weight Matrix:\n", np.round(weights, 4))
    print(f"Max weight change this epoch = {max_change:.6f}")
    print("-" * 40, "\n")

    if max_change < tol:
        print(f"Convergence detected after {epoch} epochs (tolerance
{tol}).\n")
        break

# --- Final Classification ---
print("--- Final Classification Test ---")
all_vectors = np.array([
    [0, 0, 1, 1],
    [1, 0, 0, 0],
    [0, 0, 0, 1],
    [1, 1, 0, 0],
    [0, 1, 1, 0]
])

for i, x in enumerate(all_vectors):
    distances = [np.sum((x - w)**2) for w in weights]
```

```
        winner_index = np.argmin(distances)
        predicted_class = prototype_classes[winner_index]
        print(f"Vector {i+1} {x} Classified as Class {predicted_class}")

    print("\nFinal Weight Matrix:\n", np.round(weights, 4))

# Run LVQ network
lvq_network_auto_converge()
```

RESULT:
```
--- Initial State ---
Learning Rate (alpha): 0.1
Initial Weight Matrix:
 [[0. 0. 1. 1.]
 [1. 0. 0. 0.]]
Classes: [1, 2]
----------------------------------------

Epoch 1 starting...
 Training Vector 1: [0 0 0 1], Target Class=2
 Distances: [1. 2.] → Winner = 1 (Class 1)
 Classes differ : moving away
 Updated 1: [0.  0.  1.1 1. ]
 Training Vector 2: [1 1 0 0], Target Class=1
 Distances: [4.21 1.  ] → Winner = 2 (Class 2)
 Classes differ : moving away
 Updated 2: [ 1.  -0.1  0.   0. ]
 Training Vector 3: [0 1 1 0], Target Class=1
 Distances: [2.01 3.21] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.   0.1  1.09 0.9 ]
--- Epoch 1 Complete ---
Updated Weight Matrix:
 [[ 0.    0.1   1.09  0.9 ]
 [ 1.   -0.1   0.    0.  ]]
Max weight change this epoch = 0.100000
----------------------------------------

Epoch 2 starting...
 Training Vector 1: [0 0 0 1], Target Class=2
 Distances: [1.2081 2.01  ] → Winner = 1 (Class 1)
 Classes differ : moving away
 Updated 1: [0.    0.11  1.199 0.89 ]
 Training Vector 2: [1 1 0 0], Target Class=1
```

```
 Distances: [4.0218 1.21  ] → Winner = 2 (Class 2)
 Classes differ : moving away
 Updated 2: [ 1.   -0.21  0.    0.  ]
 Training Vector 3: [0 1 1 0], Target Class=1
 Distances: [1.6238 3.4641] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.     0.199  1.1791 0.801 ]
--- Epoch 2 Complete ---
Updated Weight Matrix:
 [[ 0.      0.199   1.1791  0.801 ]
 [ 1.     -0.21    0.      0.    ]]
Max weight change this epoch = 0.110000
----------------------------------------

Epoch 3 starting...
 Training Vector 1: [0 0 0 1], Target Class=2
 Distances: [1.4695 2.0441] → Winner = 1 (Class 1)
 Classes differ : moving away
 Updated 1: [0.     0.2189 1.297  0.7811]
 Training Vector 2: [1 1 0 0], Target Class=1
 Distances: [3.9025 1.4641] → Winner = 2 (Class 2)
 Classes differ : moving away
 Updated 2: [ 1.    -0.331  0.     0.   ]
 Training Vector 3: [0 1 1 0], Target Class=1
 Distances: [1.3084 3.7716] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.     0.297  1.2673 0.703 ]
--- Epoch 3 Complete ---
Updated Weight Matrix:
 [[ 0.      0.297   1.2673  0.703 ]
 [ 1.     -0.331   0.      0.    ]]
Max weight change this epoch = 0.121000
----------------------------------------

Epoch 69 starting...
 Training Vector 1: [0 0 0 1], Target Class=2
 Distances: [2.5014 0.    ] → Winner = 2 (Class 2)
 Classes match : moving closer
 Updated 2: [ 0.0011 -0.0014  0.      0.9989]
 Training Vector 2: [1 1 0 0], Target Class=1
 Distances: [0.554  2.9985] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.5263 1.     0.4737 0.    ]
 Training Vector 3: [0 1 1 0], Target Class=1
```

```
 Distances: [0.554  3.0007] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.4737 1.     0.5263 0.    ]
--- Epoch 69 Complete ---
Updated Weight Matrix:
 [[ 0.4737  1.      0.5263  0.    ]
 [ 0.0011 -0.0014  0.      0.9989]]
Max weight change this epoch = 0.000160
----------------------------------------

Epoch 70 starting...
 Training Vector 1: [0 0 0 1], Target Class=2
 Distances: [2.5014 0.    ] → Winner = 2 (Class 2)
 Classes match : moving closer
 Updated 2: [ 0.001  -0.0013  0.      0.999 ]
 Training Vector 2: [1 1 0 0], Target Class=1
 Distances: [0.554  2.9987] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.5263 1.     0.4737 0.    ]
 Training Vector 3: [0 1 1 0], Target Class=1
 Distances: [0.554  3.0006] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.4737 1.     0.5263 0.    ]
--- Epoch 70 Complete ---
Updated Weight Matrix:
 [[ 0.4737  1.      0.5263  0.    ]
 [ 0.001  -0.0013  0.      0.999 ]]
Max weight change this epoch = 0.000144
----------------------------------------

Epoch 71 starting...
 Training Vector 1: [0 0 0 1], Target Class=2
 Distances: [2.5014 0.    ] → Winner = 2 (Class 2)
 Classes match : moving closer
 Updated 2: [ 9.000e-04 -1.200e-03  0.000e+00  9.991e-01]
 Training Vector 2: [1 1 0 0], Target Class=1
 Distances: [0.554  2.9988] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.5263 1.     0.4737 0.    ]
 Training Vector 3: [0 1 1 0], Target Class=1
 Distances: [0.554  3.0006] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.4737 1.     0.5263 0.    ]
--- Epoch 71 Complete ---
```

```
Updated Weight Matrix:
 [[ 4.737e-01  1.000e+00  5.263e-01  0.000e+00]
 [ 9.000e-04 -1.200e-03  0.000e+00  9.991e-01]]
Max weight change this epoch = 0.000129
----------------------------------------

Epoch 72 starting...
 Training Vector 1: [0 0 0 1], Target Class=2
 Distances: [2.5014 0.    ] → Winner = 2 (Class 2)
 Classes match : moving closer
 Updated 2: [ 8.000e-04 -1.000e-03  0.000e+00  9.992e-01]
 Training Vector 2: [1 1 0 0], Target Class=1
 Distances: [0.554  2.9989] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.5263 1.     0.4737 0.    ]
 Training Vector 3: [0 1 1 0], Target Class=1
 Distances: [0.554  3.0005] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.4737 1.     0.5263 0.    ]
--- Epoch 72 Complete ---
Updated Weight Matrix:
 [[ 4.737e-01  1.000e+00  5.263e-01  0.000e+00]
 [ 8.000e-04 -1.000e-03  0.000e+00  9.992e-01]]
Max weight change this epoch = 0.000117
----------------------------------------

Epoch 73 starting...
 Training Vector 1: [0 0 0 1], Target Class=2
 Distances: [2.5014 0.    ] → Winner = 2 (Class 2)
 Classes match : moving closer
 Updated 2: [ 7.000e-04 -9.000e-04  0.000e+00  9.993e-01]
 Training Vector 2: [1 1 0 0], Target Class=1
 Distances: [0.554 2.999] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.5263 1.     0.4737 0.    ]
 Training Vector 3: [0 1 1 0], Target Class=1
 Distances: [0.554  3.0005] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.4737 1.     0.5263 0.    ]
--- Epoch 73 Complete ---
Updated Weight Matrix:
 [[ 4.737e-01  1.000e+00  5.263e-01  0.000e+00]
 [ 7.000e-04 -9.000e-04  0.000e+00  9.993e-01]]
Max weight change this epoch = 0.000105
```

```
--------------------------------------

Epoch 74 starting...
 Training Vector 1: [0 0 0 1], Target Class=2
 Distances: [2.5014 0.    ] → Winner = 2 (Class 2)
 Classes match : moving closer
 Updated 2: [ 6.000e-04 -8.000e-04  0.000e+00  9.994e-01]
 Training Vector 2: [1 1 0 0], Target Class=1
 Distances: [0.554  2.9991] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.5263 1.     0.4737 0.    ]
 Training Vector 3: [0 1 1 0], Target Class=1
 Distances: [0.554  3.0004] → Winner = 1 (Class 1)
 Classes match : moving closer
 Updated 1: [0.4737 1.     0.5263 0.    ]
--- Epoch 74 Complete ---
Updated Weight Matrix:
 [[ 4.737e-01  1.000e+00  5.263e-01  0.000e+00]
 [ 6.000e-04 -8.000e-04  0.000e+00  9.994e-01]]
Max weight change this epoch = 0.000094
--------------------------------------

Convergence detected after 74 epochs (tolerance 0.0001).

--- Final Classification Test ---
Vector 1 [0 0 1 1] Classified as Class 2
Vector 2 [1 0 0 0] Classified as Class 1
Vector 3 [0 0 0 1] Classified as Class 2
Vector 4 [1 1 0 0] Classified as Class 1
Vector 5 [0 1 1 0] Classified as Class 1

Final Weight Matrix:
 [[ 4.737e-01  1.000e+00  5.263e-01  0.000e+00]
 [ 6.000e-04 -8.000e-04  0.000e+00  9.994e-01]]
```

Extra Problem	

PROGRAM:	import numpy as np
	
	# Given data
	w1 = np.array([1.0, 0.9, 0.7, 0.3, 0.2])
	w2 = np.array([0.6, 0.7, 0.5, 0.4, 1.0])

```
x  = np.array([0.0, 0.2, 0.1, 0.2, 0.0])

alpha = 0.2  # learning rate

# Compute squared Euclidean distance
D1 = np.sum((x - w1)**2)
D2 = np.sum((x - w2)**2)

print(f"Distance to Cluster 1 (w1): {D1:.4f}")
print(f"Distance to Cluster 2 (w2): {D2:.4f}")

# Determine winning cluster
if D1 < D2:
    winner = "Cluster 1"
    w_win = w1
else:
    winner = "Cluster 2"
    w_win = w2

print("\nWinning Cluster:", winner)

# Update weights of the winning unit
w_new = w_win + alpha * (x - w_win)

print("Old Weights:", w_win)
print("New Weights:", np.round(w_new, 4))
```

RESULT:
Distance to Cluster 1 (w1): 1.9000
Distance to Cluster 2 (w2): 1.8100

Winning Cluster: Cluster 2
Old Weights: [0.6 0.7 0.5 0.4 1.]
New Weights: [0.48 0.6 0.42 0.36 0.8]

CONCLUSION:	Kohonen SOM and LVQ efficiently classify and cluster data using competitive learning. SOM works unsupervised, forming topology-preserving maps, while LVQ is supervised and refines prototypes. Together, they demonstrate adaptive neural learning and pattern recognition.

BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering