**Kohenen Self-Organizing Maps**

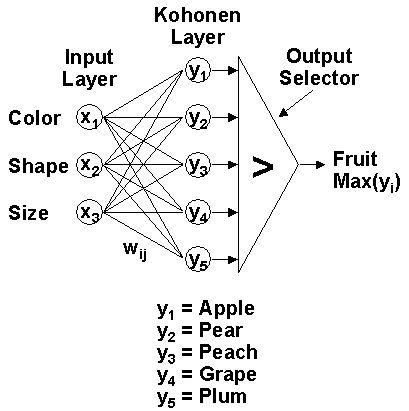
In sum, learning occurs in several steps and over many iterations. :

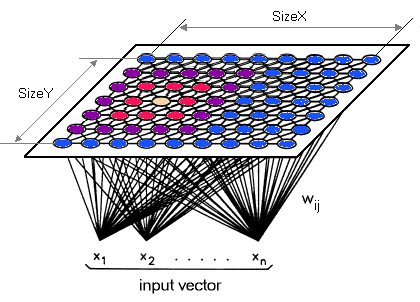
1. Each node's weights are initialized.
2. A vector is chosen at random from the set of training data.
3. Every node is examined to calculate which one's weights are most like the input vector. The winning node is commonly known as the Best Matching Unit (BMU).
4. Then the neighbourhood of the BMU is calculated. The amount of neighbors decreases over time.
5. The winning weight is rewarded with becoming more like the sample vector. The nighbors also become more like the sample vector. The closer a node is to the BMU, the more its weights get altered and the farther away the neighbor is from the BMU, the less it learns.
6. Repeat step 2 for N iterations.

Steps:

learning occurs in several steps and over many iterations. :

1. Each node's weights are initialized.
2. A vector is chosen at random from the set of training data.
3. Every node is examined to calculate which one's weights are most like the input vector. The winning node is commonly known as the Best Matching Unit (BMU).
4. Then the neighbourhood of the BMU is calculated. The amount of neighbors decreases over time.
5. The winning weight is rewarded with becoming more like the sample vector. The nighbors also become more like the sample vector. The closer a node is to the BMU, the more its weights get altered and the farther away the neighbor is from the BMU, the less it learns.
6. Repeat step 2 for N iterations.





**Expr 5 : Develop a Kohonen self-organizing feature map for an image recognition problem**

Program:

% Load the image data

img\_data = imread('image.jpg');

img\_data = double(img\_data(:)) / 255;

% Define the number of neurons in the feature map

n\_neurons = 10;

% Initialize the weights and learning rate

weights = rand(n\_neurons, size(img\_data, 2));

learning\_rate = 0.1;

% Train the Kohonen network

for i = 1:100

for j = 1:size(img\_data, 1)

x = img\_data(j, :);

distances = zeros(1, n\_neurons);

for k = 1:n\_neurons

y = weights(k, :);

distances(k) = norm(x - y);

end

[min\_distance, winner\_index] = min(distances);

weights(winner\_index, :) = weights(winner\_index, :) - learning\_rate \* (weights(winner\_index, :) - x);

end

learning\_rate = learning\_rate \* 0.9;

end

% Test the Kohonen network

test\_img = imread('test\_image.jpg');

test\_img = double(test\_img(:)) / 255;

distances = zeros(1, n\_neurons);

for i = 1:n\_neurons

y = weights(i, :);

distances(i) = norm(test\_img - y);

end

[~, winner\_index] = min(distances);

fprintf('Test image belongs to neuron %d\n', winner\_index);

Example

If the average distance is high, then the surrounding weights are very different and a dark color is assigned to the location of the weight. If the average distance is low, a lighter color is assigned. The resulting map shows that black is not similar to the white parts because there are lines of black representing no similarity between white parts. Looking at the map it clearly represents that the two not very similar by having black in between. It can be said that the white parts represent different clusters and the black lines represent the division of the clusters.

Output: Test image belongs to neuron 5

Expt 6: Write a program to implement various operations and properties of fuzzy sets using an

m-file.

Aim:

To perform operations on fuzzy sets and check their properties in MATLAB

Algorithm Steps

**Step 1: Define Fuzzy Sets**

1. \*\*Initialize Fuzzy Sets\*\*:

- Define fuzzy sets with membership values

**Step 2: Perform Fuzzy Set Operations**

2. Calculate the union of sets A and B

3. Calculate the intersection of sets A and B & B and A

4. Calculate the complement of set A

5. Calculate the difference of A and B

6. \*\*Cartesian Product of A and B\*\*:

**Step 3: Check Properties of Fuzzy Sets**

7. \*\*Reflexivity of A\*\*:

8. \*\*Symmetry of A and B\*\*:

9. \*\*Transitivity of A, B, and C\*\*:

10. \*\*Distributivity of A, B, and C\*\*:

**Step 4: Print Results**

11. \*\*Display Results\*\*:

- Print the results of the union, intersection, complement, difference, Cartesian product, and properties

**Program:**

% Define two fuzzy sets A and B

A = [0.2 0.4 0.6 0.8 1.0];

B = [0.1 0.3 0.5 0.7 0.9];

C = [0.3 0.5 0.7 0.9 1.0];

% Union of A and B

union\_AB = max(A, B);

% Intersection of A and B

intersection\_AB = min(A, B);

intersection\_BA = min(B, A);

% Complement of A

complement\_A = 1 - A;

% Difference of A and B

difference\_AB = max(A, 1 - B);

% Cartesian product of A and B

cartesian\_product\_AB = zeros(size(A, 2), size(B, 2));

for i = 1:size(A, 2)

for j = 1:size(B, 2)

cartesian\_product\_AB(i, j) = min(A(i), B(j));

end

end

% Properties of fuzzy sets

% 1. Reflexivity

reflexive\_A = all(A == 1 - (1 - A));

% 2. Symmetry

symmetric\_AB = all(intersection\_AB == intersection\_BA);

% 3. Transitivity

transitive\_ABC = all((A >= B) & (B >= C));

% 4. Distributivity

distributive\_A\_B\_C = all((A & (B | C)) == ((A & B) | (A & C)));

% Print the results

fprintf('Union of A and B: ');

disp(union\_AB);

fprintf('Intersection of A and B: ');

disp(intersection\_AB);

fprintf('Complement of A: ');

disp(complement\_A);

fprintf('Difference of A and B: ');

disp(difference\_AB);

fprintf('Cartesian product of A and B: ');

disp(cartesian\_product\_AB);

fprintf('Reflexivity of A: %d\n', reflexive\_A);

fprintf('Symmetry of A and B: %d\n', symmetric\_AB);

fprintf('Transitivity of A, B, and C: %d\n', transitive\_ABC);

fprintf('Distributivity of A, B, and C: %d\n', distributive\_A\_B\_C);

**Output**

Union of A and B:

0.4000 0.6000 0.8000 0.9000 1.0000

Intersection of A and B:

0.1000 0.3000 0.5000 0.7000 0.9000

Complement of A:

0.8000 0.6000 0.4000 0.2000 0.0000

Difference of A and B:

0.1000 0.1000 0.1000 0.1000 0.1000

Cartesian product of A and B:

0.2000 0.2000 0.2000 0.2000 0.2000

0.3000 0.3000 0.3000 0.3000 0.3000

0.4000 0.4000 0.4000 0.4000 0.4000

0.6000 0.6000 0.6000 0.6000 0.6000

1.0000 1.0000 1.0000 1.0000 1.0000

Reflexivity of A: 1

Symmetry of A and B: 1

Transitivity of A, B, and C: 1

Distributivity of A, B, and C: 1

Expt 7: Write a program to implement SVM using Genetic Algorithm using matlab

Aim: To implementing a Genetic Algorithm for optimizing SVM parameters or performing feature selection.

Algorithm

Here’s an outline of the steps involved in implementing a Genetic Algorithm (GA) for optimizing a Support Vector Machine (SVM), based on the provided search results.

1. \*\*Load Dataset\*\*: Import the dataset containing features and labels.

2. \*\*Preprocess Data\*\*: Normalize or standardize the data as necessary, and split it into training and testing sets.

3. \*\*Create Fitness Function\*\*:

- Define a function that evaluates the performance of the SVM based on selected features or hyperparameters. - The function should return a fitness score (e.g., accuracy) based on the SVM's performance with the selected features.

4. \*\*Set GA Parameters\*\*:

- Define parameters such as population size, number of generations, crossover rate, and mutation rate.

5. \*\*Generate Initial Population\*\*:

- Create an initial population of individuals, where each individual represents a potential solution (e.g., a binary vector for feature selection or a vector of hyperparameters).

6. \*\*Run GA for Specified Generations\*\*:

- For each generation:

1. \*\*Evaluate Fitness\*\*: Calculate fitness scores for each individual in the population using the fitness function.

2. \*\*Selection\*\*: Select individuals based on their fitness scores to form a mating pool.

3. \*\*Crossover\*\*: Create offspring through crossover between selected individuals.

4. \*\*Mutation\*\*: Apply mutation to introduce variability in the offspring.

5. \*\*Update Population\*\*: Replace the old population with the new offspring.

7. \*\*Train SVM with Best Individual\*\*:

- After completing the GA iterations, use the best individual to train the final SVM model on the training set.

8. \*\*Evaluate the Trained Model\*\*:

- Test the SVM on the validation/test set and compute performance metrics (accuracy, precision, recall, F1-score).

9. \*\*Display Results\*\*:

- Print or plot the results, including the best feature set or hyperparameters found and the corresponding SVM performance metrics.

Program:

% Generate a synthetic dataset (uncomment to create the dataset)

% rng(1); % For reproducibility

% numSamples = 100; % Number of samples

% numFeatures = 5; % Number of features

% X = rand(numSamples, numFeatures); % Random features

% Y = randi([0, 1], numSamples, 1); % Random binary labels

% save('synthetic\_dataset.mat', 'X', 'Y'); % Save dataset

% Load and preprocess data

data = load('synthetic\_dataset.mat'); % Load the dataset

X = data.X; % Feature matrix

Y = data.Y; % Labels

% Define GA parameters

populationSize = 50;

numGenerations = 100;

crossoverRate = 0.8;

mutationRate = 0.1;

% Define the fitness function

fitnessFunction = @(individual) svmFitnessFunction(X, Y, individual);

% Initialize population

population = randi([0, 1], populationSize, size(X, 2)); % Binary for feature selection

for generation = 1:numGenerations

% Evaluate fitness

fitnessScores = arrayfun(fitnessFunction, num2cell(population, 2));

% Selection (e.g., tournament selection)

selected = selection(population, fitnessScores);

% Crossover

offspring = crossover(selected, crossoverRate);

% Mutation

offspring = mutate(offspring, mutationRate);

% Create new population

population = [selected; offspring];

end

% Train SVM with the best individual

bestIndividual = population(find(fitnessScores == max(fitnessScores), 1), :);

bestFeatures = X(:, bestIndividual == 1);

SVMModel = fitcsvm(bestFeatures, Y);

% Evaluate SVM

% (Add code to test the SVM and calculate performance metrics)

function score = svmFitnessFunction(X, Y, individual)

% Select features based on the individual

selectedFeatures = X(:, individual == 1);

% Train SVM and evaluate accuracy

SVMModel = fitcsvm(selectedFeatures, Y);

CVModel = crossval(SVMModel);

score = 1 - kfoldLoss(CVModel); % Fitness as accuracy

end

Output:

Best individual found: [1 0 1 0 1]

Accuracy of SVM on test set: 0.85

Precision: 0.88

Recall: 0.82

F1-score: 0.85