Algorithm

A self organizing map (SOM) was implemented in Matlab to organize a data set of iris flowers into three different classes. A SOM is a neural network that takes in high dimensional data and outputs a 2D representation of it, like a map. This type of network is trained using *Kohonen's learning rule* which is based on the euclidean distance between the actual output of the data and the closest output neuron to it. This means that the SOM is trained so that the output neurons are as closely representative of the data set as possible. For this program, the SOM was trained for 15 epochs over one batch. Learning rate and width of neighbourhood was set to 1 and 10 respectively, both decaying every epoch. The data set was of 40x40x4 dimensions so the output array was of 40x40.

Results

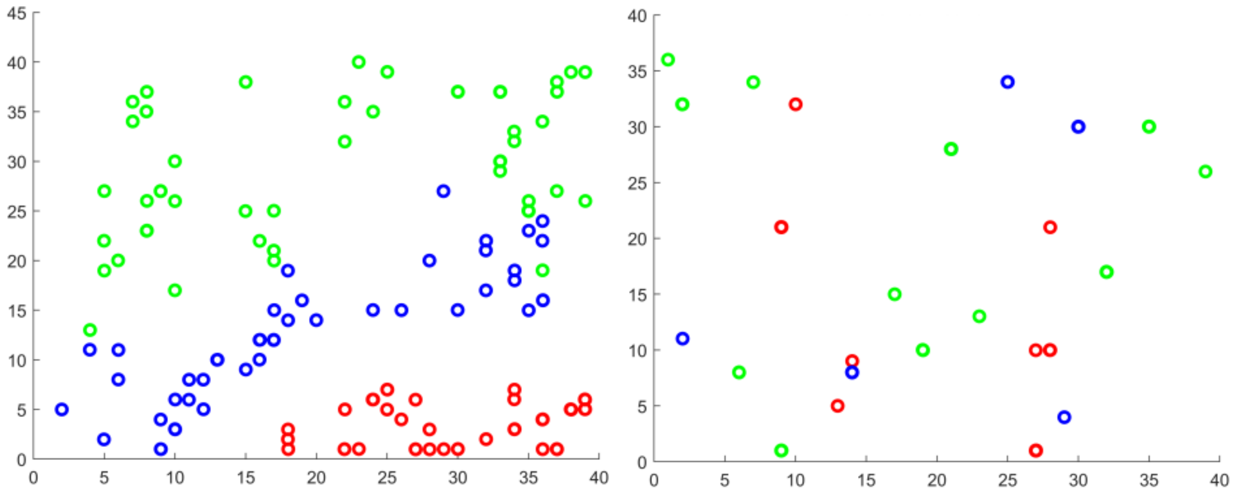


Figure 1: Self organised map to the left (post training). Unorganised map to the right (pre-training).

Discussion and conclusion

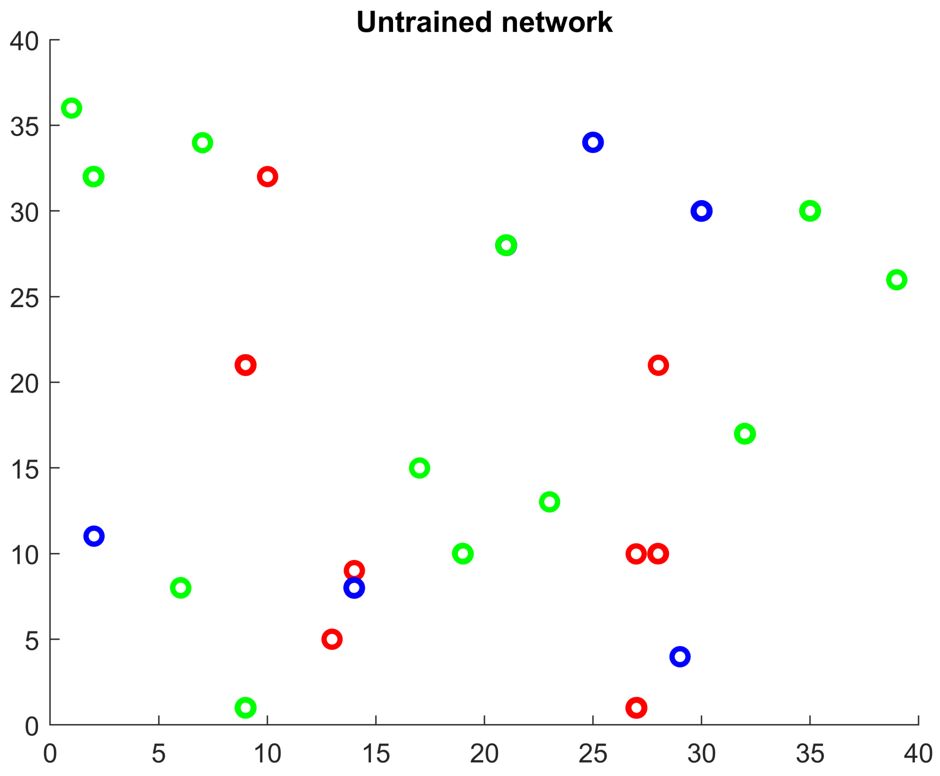
As one can see in figure 1, the SOM organizes the data well given how unorganised it was first. The points of the unorganised data set seem fewer than post training. This is because there are points overlapping each other pre-training. Noise was implemented when plotting the data to be able to see the points a little bit clearer. The parameter values for the learning rate and number of epochs that were supposed to work (0.1 and 10) didn't initially work, the network wouldn't learn to organize the data. 1 respectively 15 was used instead which was sufficient for the SOM to organize properly. A reason for this could be a programming error that went unnoticed, most likely in the data processing or in the weight updating. However, these parameters solve the problem. An increase of these parameters gave the SOM a chance to get somewhere and succeed the training. Every run the clusters would form in different locations. This is due to random initialization of the weights that affects the positioning of the clusters heavily.

Self organising map

Erik Norlin

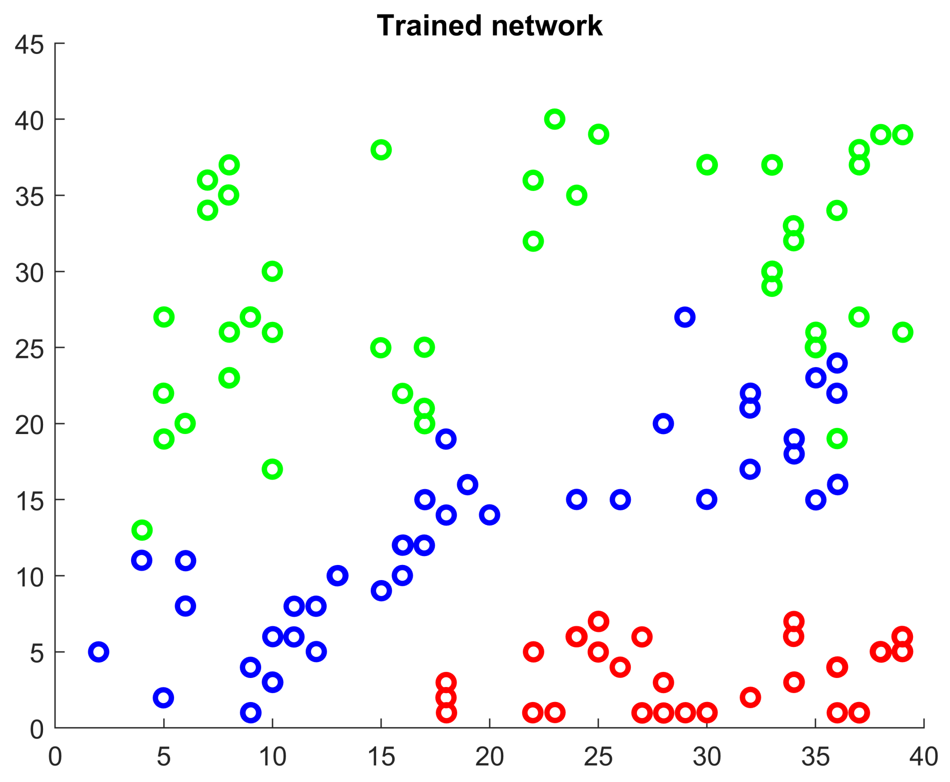
```
data = readmatrix("iris-data.csv");
targets = readmatrix("iris-labels.csv");
maxValue = max(max(data));
data = data / maxValue;
nInputs = height(data);
nEpochs = 15;

eta_0 = 1;
eta_decay = 0.001;

sigma_0 = 10;
sigma_decay = 0.05;

w_ij = rand(40,40,4);
nNeurons = 40;

% Plotting untrained network
r_0List = [];
class0 = [];
class1 = [];
class2 = [];
for iInput = 1:nInputs
    dataPoint = data(iInput,:);
    minDistance = Inf;

    % Computing the winning neuron
    for i = 1:nNeurons
        for j = 1:nNeurons
            distance = sqrt((w_ij(i,j,1) - dataPoint(1))^2 + ...
                            (w_ij(i,j,2) - dataPoint(2))^2 + ...
                            (w_ij(i,j,3) - dataPoint(3))^2 + ...
                            (w_ij(i,j,4) - dataPoint(4))^2);

            if distance < minDistance
                minDistance = distance;
                r_0 = [i+((2*rand)-1)*0.02,j+((2*rand)-1)*0.02];
            end
        end
    end

    r_0List = [r_0List ; r_0,targets(iInput)];

    if r_0List(iInput,3) == 0
        class0 = [class0 ; r_0List(iInput,:)];
    elseif r_0List(iInput,3) == 1
        class1 = [class1 ; r_0List(iInput,:)];
    elseif r_0List(iInput,3) == 2
        class2 = [class2 ; r_0List(iInput,:)];
    end
end
```

```
figure
hold on
plot(class0(:,1),class0(:,2),"or","Linewidth",2)
plot(class1(:,1),class1(:,2),"ob","Linewidth",2)
plot(class2(:,1),class2(:,2),"og","Linewidth",2)
title("Untrained network")
hold off
```



```
% Training
for iEpoch = 1:nEpochs

    % Updating learning parameters
    eta = eta_0*exp(-eta_decay*iEpoch);
    sigma = sigma_0*exp(-sigma_decay*iEpoch);

    for iInput = nInputs

        randPoint = randi(nInputs);
        dataPoint = data(randPoint,:);
        minDistance = Inf;
        r = [];

        % Computing the winning neuron
        for i = 1:nNeurons
            for j = 1:nNeurons
                distance = sqrt((w_ij(i,j,1) - dataPoint(1))^2 + ...
                                (w_ij(i,j,2) - dataPoint(2))^2 + ...
```

```
                                  (w_ij(i,j,3) - dataPoint(3))^2 + ...
                                  (w_ij(i,j,4) - dataPoint(4))^2);

                if distance < minDistance
                    minDistance = distance;
                    iMin = i;
                    jMin = j;
                end
            end
        end

        % Computing nearest neighbours to the winning neuron
        for i = 1:nNeurons
            for j = 1:nNeurons
                distance = sqrt((i - iMin)^2 + (j - jMin)^2);
                if distance < 3*sigma
                    r = [r ; i,j,distance];
                end
            end
        end

        % Updating weights
        nIncrements = height(r);
        deltaWeight = zeros(40,40,4);
        for iIncrement = 1:nIncrements
            i = r(iIncrement,1);
            j = r(iIncrement,2);
            rDistance = r(iIncrement,3);

            h = exp(-1/(2*sigma^2) * rDistance^2);
            for k = 1:length(dataPoint)
                deltaWeight(i,j,k) = deltaWeight(i,j,k) + eta*h*(dataPoint(k) - w_ij(i,j,k));
            end
        end
        w_ij = w_ij + deltaWeight;
    end
end

% Plotting trained network
r_0List = [];
class0 = [];
class1 = [];
class2 = [];
for iInput = 1:nInputs
    dataPoint = data(iInput,:);
    minDistance = Inf;

    % Computing the winning neuron
    for i = 1:nNeurons
        for j = 1:nNeurons
            distance = sqrt((w_ij(i,j,1) - dataPoint(1))^2 + ...
                            (w_ij(i,j,2) - dataPoint(2))^2 + ...
                            (w_ij(i,j,3) - dataPoint(3))^2 + ...
                            (w_ij(i,j,4) - dataPoint(4))^2);
            if distance < minDistance
                minDistance = distance;
                r_0 = [i+((2*rand)-1)*0.02  ,j+((2*rand)-1)*0.02];
            end
        end
    end

    r_0List = [r_0List ; r_0,targets(iInput)];

    if r_0List(iInput,3) == 0
        class0 = [class0 ; r_0List(iInput,:)];
    elseif r_0List(iInput,3) == 1
        class1 = [class1 ; r_0List(iInput,:)];
    elseif r_0List(iInput,3) == 2
        class2 = [class2 ; r_0List(iInput,:)];
    end
end

figure
hold on
plot(class0(:,1),class0(:,2),"or","Linewidth",2)
plot(class1(:,1),class1(:,2),"ob","Linewidth",2)
plot(class2(:,1),class2(:,2),"og","Linewidth",2)
title("Trained network")
hold off
```


Trained network