# Self-Organising Map

FFR135 Artificial Neural Networks

Axel Johansson
axejoh@student.chalmers.se

October 23, 2022

---

## *Solution proposal*

---

The task is to train a Self-Organising Map that maps a high dimensional input to a low dimensional output. The map learns to activate nearby output neurons for similar inputs. This is done by training the network weights with learning rule

$$\delta w = \eta h(i,i_o)(\vec{x} - \vec{w}) \tag{1}$$

where $h$ is the neighbourfunction defined as

$$h(i,i_o) = \exp\left\{-\frac{1}{2\sigma^2}|r_i - r_{i0}|^2\right\}. \tag{2}$$

First one initialize the weights randomly which will be saved for later to illustrate the difference. A map is fitted to the data by iterating the learning rule. I have chosen to update the position for *all* neurons instead of only updating when $\|r_i - r_{i0}\| < 3\sigma$. This will give almost the same answer since $h(i_0,i_{i0})$ is rapidly decreasing as you move away from the winning neuron (the norm increases). Especially since sigma decreases as the epochs increases according to

$$\sigma = \sigma_0 \exp\{-d_\sigma \cdot \text{epoch}\} \tag{3}$$

which makes the neighbourhood function even more pointier (lower standard deviation). This means that neurons far away doesn't move much anyways.

Once training is done the weights are used to get the position in the output space which is presented in figure 2. This can be compared to figure 1 where the weights are just randomly generated. The labels are used to color code the output and one can clearly see that the network groups similar data. There are multiple outputs on the same neuron for a random matrix making the outputs appear fewer. One could add noise to show them more clearly. *The Setosa seems to be linearly separable from figure 2.*

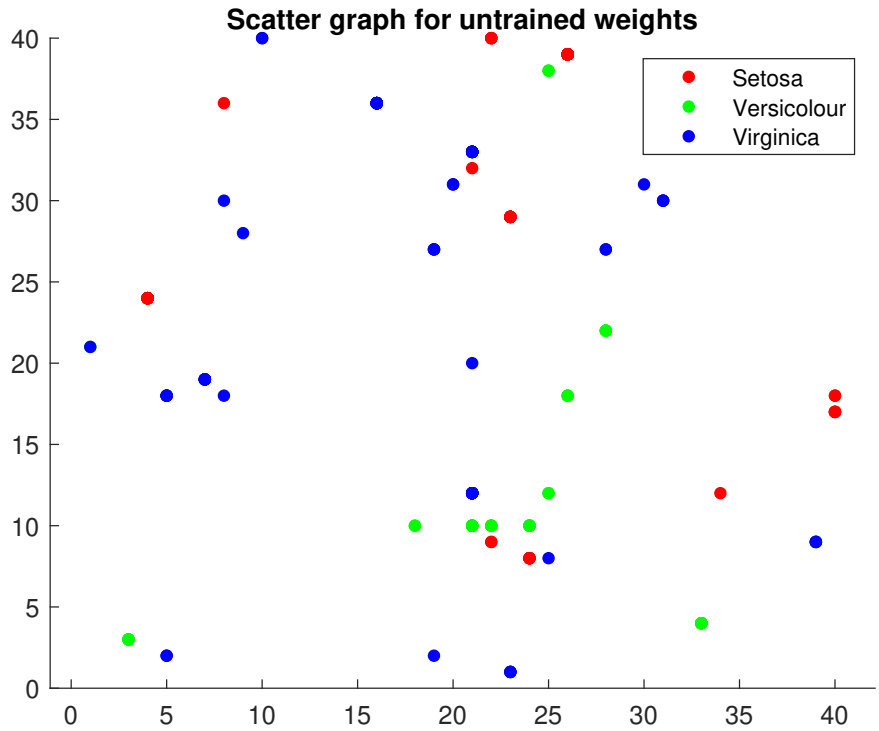


Figure 1: *This figure shows the output map for random weights where multiple outputs go on the same neuron appearing to be fewer. They could be done more visible by adding some noise.*

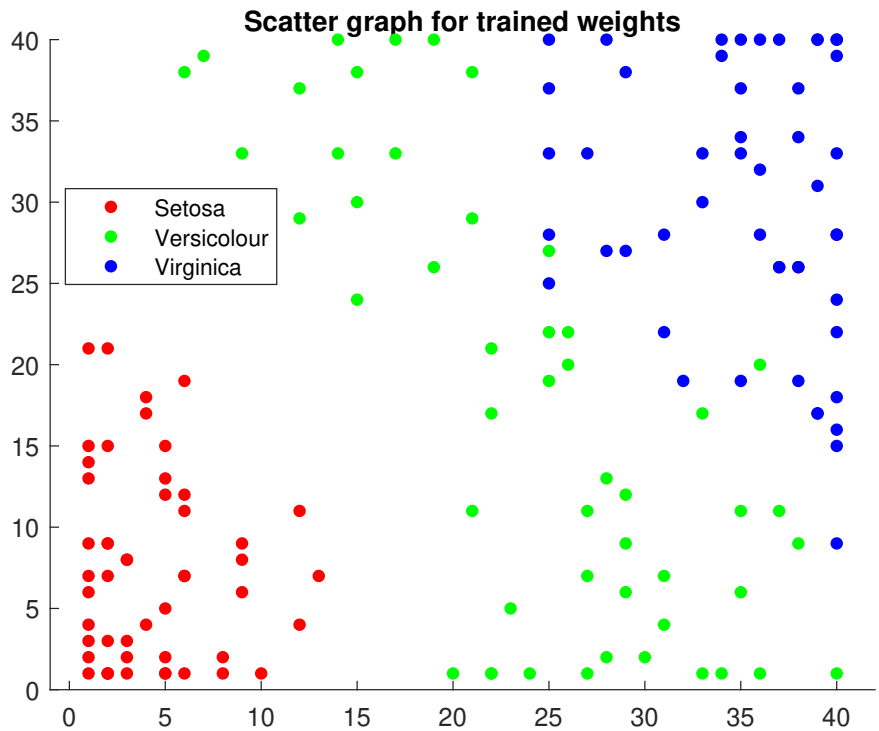


Figure 2: *This figure shows the output map for when the weights have been trained color coded with the labels. The training algorithm and parameters are shown in appendix 1. The Setosa seems to be linearly separable.*

# 1 MATLAB-kod

## 1.1 Main

```
%% Self organising map
clc, clf, clear
% Initialize
iris = csvread('iris_data.csv');
iris = iris / max(iris,[],'all');
iris_labels = csvread('iris_labels.csv');

w = rand([40,40,4]);
wUntrained = w;
nInputs = size(iris,1);
nEpochs = 100;
nNeurons = 40;

eta0 = 0.1;
dEta = 0.01;
sigma0 = 10;
dSigma = 0.05;

% Training
for epoch = 1:nEpochs
    sigma = sigma0*exp(-dSigma*epoch);
    eta   = eta0*exp(-dEta*epoch);
    for inputs = 1:nInputs
        x = iris(randi(nInputs),:);
        w = TrainMap(w, x, sigma, nNeurons, eta);
    end
end


% Get position with untrained and trained weights
unTrainedPposition = zeros(nInputs,2);
position = zeros(nInputs,2);
for k = 1:nInputs
    unTrainedPposition(k,:) = GetWinningNeuron(wUntrained, iris(k,:));
    position(k,:) = GetWinningNeuron(w, iris(k,:));
end

% Plot with untrained- and trained-weights
hold on
figure(1);
gscatter(unTrainedPposition(:,1),unTrainedPposition(:,2), iris_labels)
legend('Setosa','Versicolour','Virginica','location','northeast')
title('Scatter graph for untrained weights')

figure(2);
gscatter(position(:,1),position(:,2),iris_labels)
legend('Setosa','Versicolour','Virginica','location','northeast')
title('Scatter graph for trained weights')
hold off
```

## 1.2 Functions

### 1.2.1 TrainMap

```
function w = TrainMap(w, x, sigma, nNeurons, eta)
    i0 = GetWinningNeuron(w,x);
    dw = zeros(size(w));

    for i = 1:nNeurons
        for j = 1:nNeurons
            for k = 1:4
                dw(i,j,k) = eta * NeighbourhoodFunction(i0,i,j,sigma)*(x(k)-w(i,j,(k)));
            end
        end
    end
    w = w + dw;
end
```

### 1.2.2 NeighbourhoodFunction

```
function h = NeighbourhoodFunction(i0,i,j, sigma)
    ri0 = i0;
    ri = [i,j];
    h = exp(-norm(ri-ri0)/(2*sigma^2));
end
```

### 1.2.3 GetWinningNeuron

```
function i0 = GetWinningNeuron(w, x)
    d = zeros(40);
    for k = 1:4
        d = d + sqrt((w(:,:,k) - x(k)).^2);
    end
    minimum = min(min(d));
    [i,j]=find(d==minimum);
    i0 = [i,j];
end
```