Code for Self organising map task

```
% Self organising map. Written by: Axel Qvarnström
clear all
close all
clc

% Loading the data and labels
data = load('iris-data.csv');
labels = load('iris-labels.csv');

% Standardise the data
data = data ./ max(data);

% Initializing
initWeightArray = rand([40,40,4]);
weightArray = initWeightArray;
learningRate = 0.1;
learningDecay = 0.01;
width = 10;
widthDecay = 0.05;


nEpochs = 10;
nDataPoints = length(data);




for epoch = 1:nEpochs
    [learningRate, width] = Decay(learningRate, width, learningDecay, widthDecay, epoch);
    for i = 1:150
        randomDataPointIndex = randi(nDataPoints);
        input = data(randomDataPointIndex,:);
        term1Final = (weightArray(:,:,1) - input(1)).^2;
        term2Final = (weightArray(:,:,2) - input(2)).^2;
        term3Final = (weightArray(:,:,3) - input(3)).^2;
        term4Final = (weightArray(:,:,4) - input(4)).^2;
        distanceFinal = sqrt(term1Final + term2Final + term3Final + term4Final);

        % Updating
        minPos = WinningNeuronPos(distanceFinal);
        weights2Update = squeeze(weightArray(minPos(1),minPos(2),:))';
        neighbourhoodfun = NeighbourhoodFun(minPos, minPos, width);
        deltaWeights = DeltaWeights(learningRate, neighbourhoodfun, input, weights2Update);
        weights2Update = weights2Update + deltaWeights;
        weightArray(minPos(1),minPos(2),:) = weights2Update;

        % Updating Weights for the close neurons
        weightArray = UpdateWeightsForCloseNeurons(distanceFinal, learningRate, input,...
            weightArray, minPos,width);

    end

end

% Create empty lists to append the winning neuron positions using the
% final weight array. The different list correspond to the different
% labels
list1FinalWeight = zeros(50,2);
list2FinalWeight = zeros(50,2);
list3FinalWeight = zeros(50,2);

% Create empty lists to append the winning neuron positions using the
% initializing weight array. The different list correspond to the different
% labels
list1InitWeight = zeros(50,2);
list2InitWeight = zeros(50,2);
list3InitWeight = zeros(50,2);

% Creating indexes to append winning neurons to the right spot in the lists
list1Index = 1;
list2Index = 1;
list3Index = 1;

for j = 1:150
    input = data(j,:);
    label = labels(j);
    % Process for taking out the winning neuron position base on the final
    % updated weight array
    term1Final = (weightArray(:,:,1) - input(1)).^2;
    term2Final = (weightArray(:,:,2) - input(2)).^2;
    term3Final = (weightArray(:,:,3) - input(3)).^2;
    term4Final = (weightArray(:,:,4) - input(4)).^2;
    distanceFinal = sqrt(term1Final + term2Final + term3Final + term4Final);
    winningNeuronPositionFinal = WinningNeuronPos(distanceFinal);

    % Process for taking out the winning neuron position base on the
    % initial weight array
    term1Init = (initWeightArray(:,:,1) - input(1)).^2;
    term2Init = (initWeightArray(:,:,2) - input(2)).^2;
    term3Init = (initWeightArray(:,:,3) - input(3)).^2;
    term4Init = (initWeightArray(:,:,4) - input(4)).^2;
    distanceInit = sqrt(term1Init + term2Init + term3Init + term4Init);
    winningNeuronPositionInit = WinningNeuronPos(distanceInit) + normrnd(0,0.04,[1,2]);

    if label == 0
        list1FinalWeight(list1Index,:) = winningNeuronPositionFinal;
        list1InitWeight(list1Index,:) = winningNeuronPositionInit;
        list1Index = list1Index + 1;
    end
    if label == 1
        list2FinalWeight(list2Index,:) = winningNeuronPositionFinal;
        list2InitWeight(list2Index,:) = winningNeuronPositionInit;
        list2Index = list2Index + 1;
    end

    if label == 2
        list3FinalWeight(list3Index,:) = winningNeuronPositionFinal;
        list3InitWeight(list3Index,:) = winningNeuronPositionInit;
        list3Index = list3Index + 1;
    end

end

% Panel 1 based on the initial weight array
subplot(1,2,1);
scatter(list1InitWeight(:,1), list1InitWeight(:,2),'green','filled')
hold on
scatter(list2InitWeight(:,1), list2InitWeight(:,2),'red','filled')
scatter(list3InitWeight(:,1), list3InitWeight(:,2),'blue','filled')
legend('Iris Setosa', 'Iris Versicolour', 'Iris Virginica')
title('Panel 1, winning neuron positions using initial weights')

% Panel 2 based on the final weigth array
subplot(1,2,2);
scatter(list1FinalWeight(:,1), list1FinalWeight(:,2),'green','filled')
hold on
scatter(list2FinalWeight(:,1), list2FinalWeight(:,2),'red','filled')
scatter(list3FinalWeight(:,1), list3FinalWeight(:,2),'blue','filled')
legend('Iris Setosa', 'Iris Versicolour', 'Iris Virginica')
title('Panel 2, winning neuron positions using final weights')
```

Function to update learning rate and the neighbourhood width for each epoch:

```
    function [learningRate, width] = Decay(learningRate, width, learningDecay,...
    widthDecay, epoch)

    learningRate = learningRate * exp(-learningDecay * epoch);
    width = width * exp(-widthDecay * epoch);

end
```

Function for calculate the neighbourhood function value:

```
    function funValue = NeighbourhoodFun(pos, minPos, width)
    nominator = norm(pos - minPos)^2;
    denominator = 2 * width^2;
    funValue = exp(-nominator / denominator);

end
```

Function to calculate the delta weights:

```
    function deltaW = DeltaWeights(learningRate, neighbourhoodFun, inputs, Weight2Update)

    deltaW = learningRate * neighbourhoodFun * (inputs - Weight2Update);

end
```

Function that update the weights based on the neurons that are in a distance $< 3\sigma$:

```
function weightArray = UpdateWeightsForCloseNeurons(distance, learningRate, input,...
    weightArray, minPos,width)

    closeDistances = distance(distance < 3*width);
    nDistances = length(closeDistances);

    for i = 1:nDistances
        [x, y] = find(distance == closeDistances(i));
        closePos = [x, y];
        weights2Update = squeeze(weightArray(x,y,:))';
        neighbourhoodfun = NeighbourhoodFun(closePos, minPos, width);
        deltaWeights = DeltaWeights(learningRate, neighbourhoodfun, input, weights2Update);
        weights2Update = weights2Update + deltaWeights;
        weightArray(x,y,:) = weights2Update;
    end

end
```

Function to calculate the position for the winning neuron:

```
    function winningNeuronPos = WinningNeuronPos(distance)

    minDistance = min(distance, [],"all");
    [x,y] = find(distance == minDistance);
    winningNeuronPos = [x, y];

end
```