Assignment 7

Description

The goal of this assignment was to design and organize the Self-Organizing Map (SOM) from the travelling salesman problem for neighborhood sizes 0-2, and with two different initial weight vectors for each neighborhood size.

Technical Details

The SOM is a tool that can be used to map higher dimensional data into a lower dimensional space, and provide a topographical representation of the data that can be used for tasks like clustering and sensitivity analysis. The SOM algorithm can be broken down into the following steps[1].

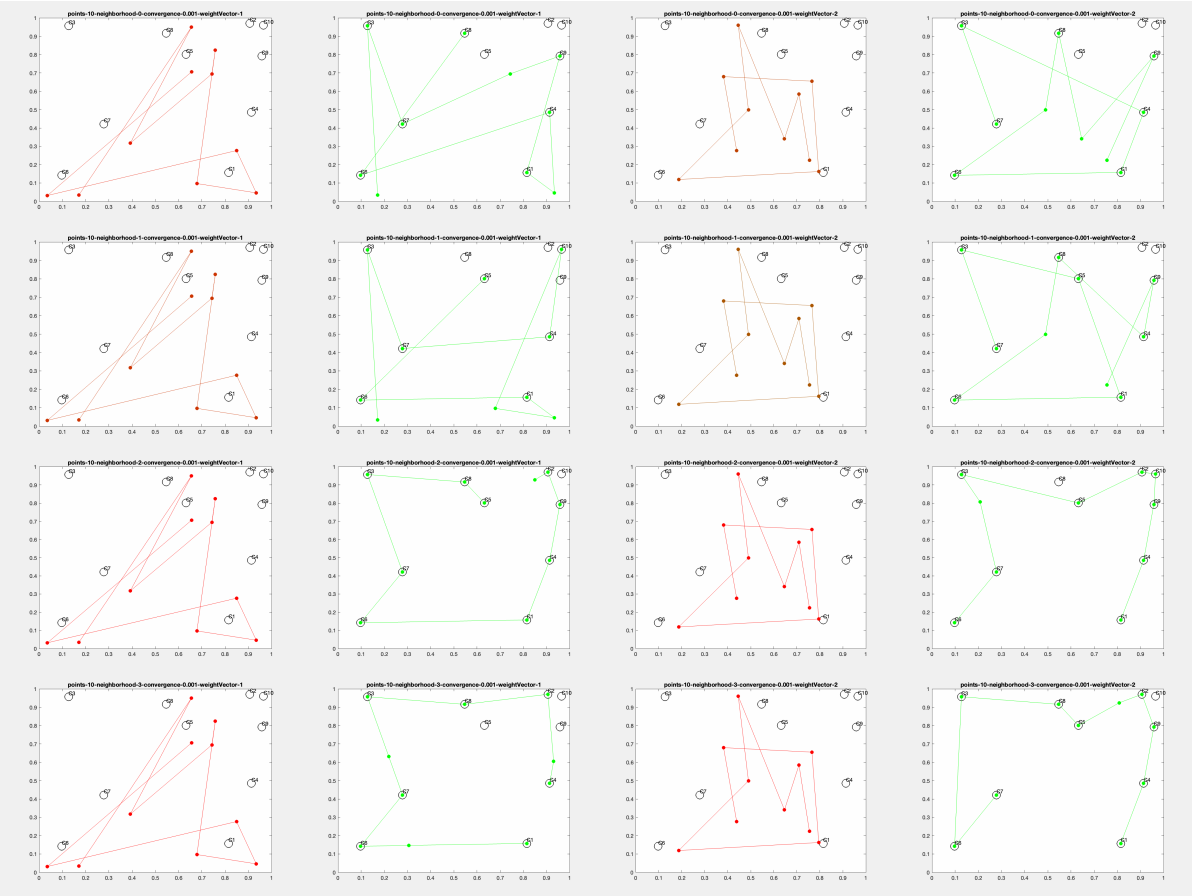
1. Each node's weights are initialized.
2. A vector is chosen at random from the set of training data.
3. Every node is examined to calculate which one's weights are most like the input vector. The winning node is commonly known as the Best Matching Unit (BMU).
4. Then the neighbourhood of the BMU is calculated. The amount of neighbors decreases over time.
5. The winning weight is rewarded with becoming more like the sample vector. The neighbors 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.

There are three main sections of the algorithm; matching, cooperation, and competition. In the matching stage (step 3), the weight unit that fits the matching criteria, which is often a minimum distance measure, is chosen as the best matching unit (BMU). In the cooperation stage (step 5) the BMU and the units within the neighborhood are updated to move towards (or potentially away from) the random vector. In the competition stage, the neighborhood has been updated to include only the BMU, and the BMU is the only unit that is updated, causing it to 'win' over the other units.

Results

The results of using SOM for the TSM problem are shown below. Each row has the start and end values of the weight vectors for a single neighborhood size and two different starting weights. The color of the lines and points change from red to green as the weight vector approaches the final solution. I also included a run for a neighborhood size of 3 for comparison. Parameter values are $sigma_0$="number of neighbors", tau=200.

For the GIF version of these weight evolutions, please click here



We see that for a neighborhood size of 0 and 1, we pass through many of the points, but we definitely do not get the shortest route. This is because we do not take advantage of the cooperation stage of the algorithm. With a neighborhood size of 0, there is no cooperation, so the nodes are not able to re-order themselves in the way that is necessary. With one neighbor, there is still not enough cooperation to yield good results.

With a neighborhood size of 2 we get good results. The cooperation stage is able to re-order the units appropriately and we fit 80% of the points. The routes that are found are not identical, and the starting position of the weights clearly affect the final outcome. With a size of 3 we don't necessarily get better results. We still get 80% of the points, and don't get a shorter path. I haven't looked into it further, but I would think that there is a relationship between the desired solution space, the dimensions of the input vector, the total number of points, and the chosen neighborhood size.

I also ran the code with lower values of tau. I found that increasing the value gave better results. I believe this is because it extends the number of iterations where cooperation is in play. With a larger number of points it might be better to decrease this number for the sake of efficiency.

Conclusion

Based on my results I have drawn a few conclusions. Having a neighborhood size of less than 2 for the TSM problem yields sub-par results. Having a neighborhood of more than 2

isn't necessarily going to give better results for 10 points. Increasing tau with a neighborhood size of 2 increases the iteration duration of the cooperation phase and gives better results to a certain point.

Code

```
% Assignment 7
% Taten Knight

% Cleanup
clear all;
close all;
clc;

% Copied and updated from TSP_SOMS file
% Setup
numPoints = 10;
x = rand(numPoints, 2);
differentStarts = 2;
wStarts = cell(1, differentStarts);
for starts = 1:differentStarts
    wStarts{starts} = rand(numPoints,2);
end
convergence=.001;
sigmaMax = 3;
diffs = cell(1, sigmaMax + 1);
tau = 200;

if numPoints < 2 * sigmaMax + 1
    throw(MException('InputError:numPoints_nhMax',
'numpoints must be greater than or equal to 2 * nhMax + 1'))
end

% Run with different numbers of neighbors
for sigma0 = 0:sigmaMax
    for start = 1:differentStarts
        weights = [];
        diff=10;
        w = wStarts{start};
        %TSP using SOM
        titleString = ['points-' num2str(numPoints) '-
neighborhood-' num2str(sigma0) '-convergence-'
num2str(convergence) '-weightVector-' num2str(start)];
        fig = figure();

        % Iterative Loop
        its = 0;
        while diff>convergence
            its = its + 1;
            if size(weights) == 0
                weights(:, : , end) = w;
            else
```

```
                    weights(:, : , end + 1) = w;
                end
                % Save old weights
                oldw=w;

                % Change order that we check node with
                order=randperm(numPoints);

                % For each node
                for i=1:numPoints
                    % Distance vector = the distance between a
random x and the weight,
                    % w
                    d=ones(numPoints,1)*x(order(i),:)-w;
                    d=(d(:,1).^2+d(:,2).^2).^0.5;
                    % m2 is the index of the vector with the
minimum distance (aka the
                    % closest node to the datapoint
                    [m1 m2]=min(d);
                    % Update the weight vectors within the
neighborhood of the winning
                    % vector (+-2)
                    sigma = sigma0 * exp(-its / tau);

                    for d1=m2-floor(sigma):m2+floor(sigma)
                        d = abs(d1 - m2);
                        if d == 0
                            h = 1;
                        else
                            h = exp(- (d^2) / (2 * (sigma ^
2)));
                        end
                        modD1 = mod(d1, numPoints);
                        if d1 == 0 || d1 == numPoints
                            modD1 = numPoints;
                        end
                        w(modD1,:)=w(modD1,:)+h*(x(order(i),:)-
w(modD1,:));
                    end
                end
                % Calculate the current change in w
                diff=norm(oldw-w);
                diffs{sigma0 + 1} = [diffs{sigma0 + 1} diff];
            end
        %     plot(w(:,1),w(:,2),'r-o', 'LineWidth', 1,
'MarkerFaceColor', 'g', 'Color', 'g')
            weightsSize = size(weights);
            iters = weightsSize(3);
            for j = 1:iters
                roundedJ = round(j);
                color =  [1 - (j / iters), j / iters, 0];
                plot(x(:,1),x(:,2),'o', 'MarkerSize', 15,
'MarkerEdgeColor', 'k')
```

```
                title(titleString);
                axis([0 1 0 1])
                hold on
                for i = 1:numPoints
                    text(x(i,1),x(i,2)+0.02,['C' num2str(i)])
                end
                plot(weights(:,1, roundedJ),weights(:,2,
roundedJ),'r-o', 'LineWidth', .1, 'MarkerFaceColor', color,
'Color', color)
                frame = frame2im(getframe(fig));
                [A,map] = rgb2ind(frame,256);
                if j == 1
                    imwrite(A,map,[titleString
'.gif'],'gif','LoopCount', Inf ,'DelayTime', 3);
                elseif j == iters
                    imwrite(A,map,[titleString
'.gif'],'gif','WriteMode','append','DelayTime', 5);
                else
                    imwrite(A,map,[titleString
'.gif'],'gif','WriteMode','append','DelayTime', 10 / iters);
                end
                hold off
            end
        end
    end

    close all;
```

Sources

[1] Self-Organizing Maps