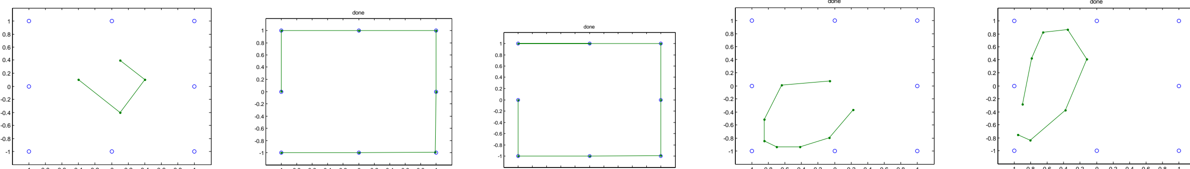
CS 391L HW6 Kohonen Self-organizing Map – Jay Liu – jayliu@cs
Traveling salesman
Initially, a starting path with proto-cities is created in the center of the expected path. The number of proto-cities is smaller than the number of real cities. The exactness of this initial path is not important. However, to reduce learning, it should be located approximately where the path would be. Next, we find the closest proto-city for each of the input city. Through experimentation, it was found that it is much better to choose the input city randomly. Going through input cities sequentially biases the output toward optimizing for the first city, and while the result is deterministic, it may be deterministically bad. The algorithm then performs online update, and the proto-city w as well as its neighbors are moved closer to the real city x, following the equation $w = w + \eta exp(-d^2/\alpha)*(x - w)$. d is the distance of the proto-city w being moved from the city's closest proto-city. η had to be initialized to be sufficiently large, and kept sufficiently large so that movements continue to be made after many iterations. It starts from 0.9, and decreases by 1/60th per iteration until 0.001. Similarly, α had to be relatively large to keep things from stopping to move, so it starts from 10, decreases by 1/30th till 0.01. A larger α means more neighbors will move more, and such movements help to introduce greater flexibility in the system.

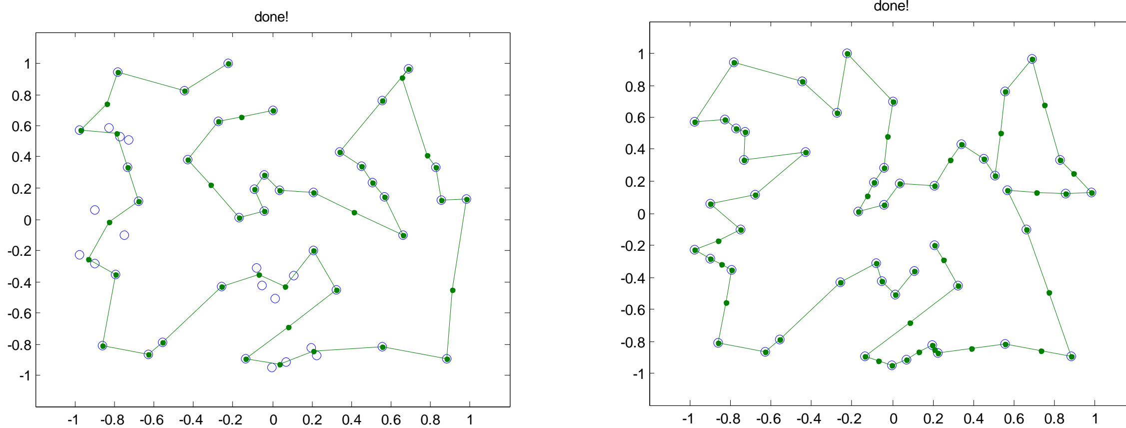
After an iteration of online updates, where an iteration means all cities have been picked, we check by how much each of the proto-cities has actually moved. There are two possibilities if a proto-city has not moved: it either has been uniquely attached to a real city, in which case the algorithm was a success, or it is stuck halfway between multiple real cities. To break such a tie, we create a new proto-city between the current proto-city and the next one. The exact distance between the two is random to avoid further deadlocks: $w_{new} = w_i + rand * (w_{i+1} - w_i)$.

Initially the number of proto-cities was not allowed to exceed the number of real cities. For the experiment, the simplest case is a square perimeter traversal. The blue dots are the input (the order is not important since the input cities are chosen in random order), and green dots are the proto-cities, where the line shows their order.



From left to right: (1) starting condition; (2) successful optimal path; (3) successful optimal path #2; (4) unsuccessful path; (5) unsuccessful path.

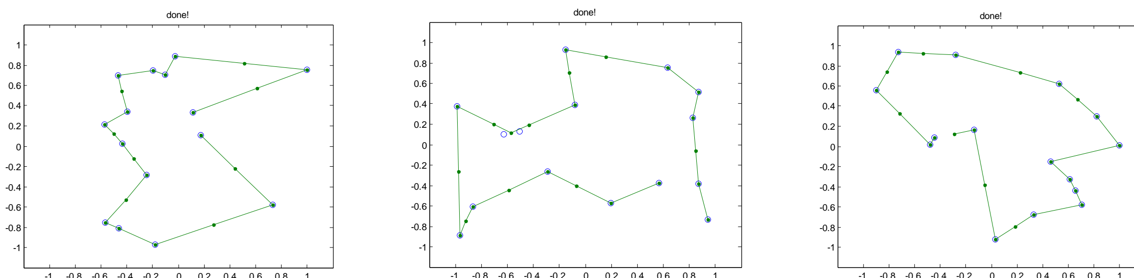
The success was mixed, even for such simple graph. One of the reasons the unsuccessful cases ceased to progress was because one or more of the proto-cities were no longer chosen as the nearest proto-city by any real cities, and hence become stale. Subsequently, there is not a sufficient number of proto-cities to trace the real cities. To resolve this problem, the maximum number of proto-cities was allowed to increase to 150% of the number of real cities. This alleviated much of the problem. For the simple case above, an optimal result was always found. The effect is also illustrated by the following 50-city example, where it can be observed that more proto-cities allowed more real-cities to match.



*From left to right: (1) # proto-cities <= # real cities; (2) # proto-cities <= 1.5 * # real-cities*

Since the new proto-cities are created along the path between other proto-cities, the increase in distance is negligible. Of course, the algorithm can be further improved to remove all proto-cities that do not correspond to real cities from the final output.

Earlier, it was noted that for a simple square path, the results were optimal. In general, if the path is a circular shape, the found path was almost always optimal. To observe optimality, we randomly generate a number of graphs with fifteen cities and visually inspect the outcome. It is concluded that the algorithm outputs solutions that look highly reasonable.



Randomly generated 15-city graphs.

Multidimensional data to 2-D torus mapping

In addition to the 13 given features, an additional 14th parameter was created to represent the known identity of the animal. This allows attributes for a particular animal to group as a cluster. Without this label, animals would intermingle more freely within a cluster. This label feature was a single input, and is assigned sequentially to the animals in 1/16 increments from 0 to 1.

First, an n x n network of neurons was created, and each neuron was initialized with 14 feature containers that were given random starting values. Randomly, an input was chosen, and the neuron with feature values closest to the input was found. The proximity algorithm uses the Euclidean distance. Then, this neuron w is moved closer to the input value x, again using the equation $w = w + \eta exp(-d^2/\alpha)*(x - w)$. In addition, the neighbors of the neuron were also moved. The network was considered to be a torus. The distance d to its neighbor is the sum of the shortest horizontal and vertical paths to that neighbor. Since there are two directions to travel on each axis to its neighbor, the shortest distance along each axis does not exceed $n/2$. The furthest neighbor is at most $n/2 + n/2 = n$ away.

There were three parameters that required adjustments. The two training parameters η and α affected the performance very much. A small η (< 0.5) was found to be ineffective as the closest neuron was not being pulled quickly enough before another input claims it. A large η = 0.95 was

used as starting value, decreasing slowly (by 0.133%) after each training iteration. In addition, α had to start large, and was picked as 20. When it was small (<1) clustering was not obvious even after many iterations. A larger α means more neurons further away from the peak were moved. Such large-scale movement was found to be very effective in creating clusters.

In addition, the size of the network was found to be important as well. Initially, a 10x10 network was created. This created smaller clustered results, and more importantly, at times an animal – the Zebra – was missing from the final result because the network was not large enough to differentiate between smaller variations in features. Making a larger 20x20 network made this problem disappear.

The result is flattened and shown below as a table. We see a concentration of Hunters in the middle and bottom right, and middle left. Sedate four legged animals on lower center. Small feathered animals are on top and bottom of left side.

Owl	Dove	Dove	Goose	Goose	Goose	Goose	Hawk	Lion	Horse	Horse	Tiger	Tiger	Tiger	Tiger	Hawk	Hawk	Hawk	Hawk	Owl
Dove	Dove	Goose	Goose	Goose	Goose	Goose	Goose	Hawk	Tiger	Horse	Tiger	Tiger	Tiger	Tiger	Cat	Hawk	Hawk	Owl	Owl
Hen	Dove	Goose	Duck	Goose	Goose	Goose	Dove	Dove	Tiger	Tiger	Tiger	Tiger	Tiger	Tiger	Tiger	Cat	Hawk	Owl	Owl
Hen	Hen	Duck	Duck	Duck	Goose	Goose	Tiger	Tiger	Tiger	Tiger	Tiger	Tiger	Tiger	Tiger	Cat	Cat	Cat	Hawk	Hen
Hen	Hen	Duck	Duck	Duck	Duck	Goose	Hen	Lion	Lion	Lion	Lion	Tiger	Tiger	Tiger	Cat	Cat	Cat	Hen	Hen
Hen	Hen	Duck	Duck	Duck	Duck	Dog	Tiger	Lion	Lion	Lion	Lion	Lion	Lion	Lion	Wolf	Cat	Cat	Dog	Hen
Hen	Hen	Duck	Duck	Hawk	Dog	Lion	Dog	Tiger	Lion	Lion	Lion	Lion	Lion	Wolf	Wolf	Wolf	Dog	Dog	Dog
Dog	Hen	Goose	Duck	Dove	Horse	Owl	Eagle	Horse	Lion	Lion	Lion	Wolf	Wolf	Wolf	Wolf	Dog	Dog	Dog	Dog
Fox	Wolf	Hawk	Duck	Fox	Dog	Wolf	Goose	Owl	Lion	Lion	Wolf	Wolf	Wolf	Wolf	Wolf	Wolf	Dog	Dog	Fox
Fox	Fox	Tiger	Fox	Eagle	Cat	Duck	Cat	Dog	Wolf	Tiger	Wolf	Wolf	Wolf	Wolf	Wolf	Wolf	Wolf	Fox	Fox
Fox	Cat	Hen	Hawk	Wolf	Tiger	Lion	Dove	Eagle	Fox	Horse	Lion	Wolf	Wolf	Wolf	Wolf	Wolf	Wolf	Fox	Fox
Tiger	Hawk	Dog	Fox	Fox	Cow	Wolf	Wolf	Wolf	Horse	Lion	Cow	Lion	Wolf	Wolf	Wolf	Wolf	Fox	Dove	Fox
Cow	Duck	Hen	Cat	Wolf	Goose	Lion	Cow	Cat	Tiger	Cow	Cow	Cow	Horse	Wolf	Wolf	Horse	Dog	Dog	Cow
Cat	Eagle	Fox	Lion	Fox	Horse	Dog	Dog	Cow	Cow	Cow	Cow	Cow	Cow	Wolf	Tiger	Wolf	Duck	Goose	Eagle
Goose	Goose	Eagle	Wolf	Tiger	Hen	Tiger	Cow	Cow	Cow	Cow	Cow	Cow	Cow	Cow	Tiger	Fox	Dove	Hawk	Dove
Hawk	Dove	Eagle	Hen	Cat	Horse	Hawk	Cow	Cow	Cow	Cow	Cow	Cow	Cow	Cow	Cow	Dog	Eagle	Eagle	Hen
Hawk	Dove	Dove	Hen	Duck	Horse	Horse	Horse	Horse	Zebra	Zebra	Cow	Cow	Cow	Cow	Fox	Fox	Hawk	Eagle	Eagle
Owl	Dove	Dove	Hawk	Owl	Hawk	Eagle	Horse	Horse	Horse	Horse	Zebra	Horse	Cow	Cow	Tiger	Eagle	Eagle	Eagle	Eagle
Dove	Dove	Dove	Goose	Goose	Hen	Horse	Zebra	Horse	Horse	Zebra	Horse	Horse	Horse	Horse	Cat	Hawk	Hawk	Eagle	Owl
Dove	Dove	Dove	Dove	Goose	Goose	Dove	Horse	Horse	Horse	Horse	Horse	Horse	Tiger	Tiger	Hawk	Hawk	Hawk	Owl	Owl

An interesting observation is that at first glance, Zebra is underrepresented, and seems to be sparse in the graph. However, upon reviewing the input data, Zebra has identical features as the Horse, except of course for their label. Therefore, Zebra and Horse are grouped closely together, with Zebra only showing when its label attribute was its distinguishing feature. Since they were labeled sequentially, Zebra and Horse – being adjacent neighbors in the input – only differ by 1/16. An improvement would be to maximize their label feature difference.

Submission

The submission package additionally contains *kmap.m* for the Traveling salesman problem and outputs a graph of the solution, and *animals.m* for the classification problem and outputs a table of animal labels. Both files are self-contained without input parameters.