Self Organizing Maps
Lab rapport IT3105 - Module 4

Sigve Skaugvoll & Thomas Wold

November 2017

1 Introduction

The assignments was to get a deep understanding of how Self organizing maps works, by implementing SOMs from scratch and solve the NP-hard Traveling Salesman Problem (TSP) and classify MNIST dataset.

2 Self orginizing maps (SOM)

2.1 What is SOM

Self organizing maps uses unsupervised training / learning, this means that the network learn to form their own classifications of the training data without external help. Thus giving that the input patterns share common features, and that the network will be able to identify these features across the range of input patterns.

SOM also uses competitive learning - the output neurons compete amongst themselves to be activated, wi the result that only one is activated at any one time. - This gives that the neurons are forced to organize themselves.

2.2 Structure

SOM usually just have 2 dimension, Input dimension and the output (neurons). The input should be maped into the neurons, and after training, into neighborhoods.

2.3 Topographic Maps

Principle of topographic map formation: “The spatial location of an output neuron in a topographic map corresponds to a particular domain or feature drawn from the input space”.

The layout of the neurons is in a ”order” the neuron has neighbors and the neighbors should be similar, thus the neighbors of a neuron can not change, but their weights/values can. That is what makes the

neighborhood neurons similar, when starting from random. Similar input goes into the same neighborhood of neurons.

2.4 Algorithm

1. Initialization:
   (a) Generate neurons and initialize them with small random weights
2. Sampling - Choose a random input case from all the inputs.
3. Matching - Find the Winning neuron. The Neuron that resembles the input the most by having smallest difference (distance) between neuron weights and input.
   (a) Euclidean distance works great.
   (b) Remember the smaller distance the closer / more similar to the input the neuron is.
4. Updating - Apply the weight update equation:
   (a) To do so we need to calculate the topological distance (distance in "graph", not weights) between the winning neuron and each neuron, the Neighborhood membership, the learning rate for this epoch and the new neighborhood size.
   (b) Learning Rate :
   (c) Neighborhood Membership :
   (d) Neighborhood Size :
   (e) Topological Distance : IF graph: — Winning neuron x coord - neuron x coord — + — Winning neuron y coord - neuron x coord — or if in a ring such as TSP, the distance in the ring between the winning neuron and the neurons.
5. Continuation / convergence - Keep returning to step 2 until the feature map stops changing.

2.5 Installation

We use Anaconda version of Python, which comes with many if not all required python modules to run the project.

If running the Annaconda version of Python, then skip this subsecetion, or if for some reason that conda misses numpy, replace ‘pip‘ with ‘conda‘.

2.5.1 Prerequisites

1. Python 3.5 or greater.
2. mathplotlib
3. numpy

2.5.2 Installing Numpy using pip

1. $ pip install mathplotlib
2. $ cd
3. $ cd .matplotlib/
4. $ vim matplotlibrc
5. Write: backend: TkAgg
6. Save and exit vim: hit ‘esc‘, then type ‘:wq‘

2.5.3 Installing Mathplotlib using pip

Optional: First activate your virtual environment, which has TensorFlow installed

1. $ pip install mathplotlib
2. $ cd
3. $ cd .matplotlib/
4. $ vim matplotlibrc
5. Write: backend: TkAgg
6. Save and exit vim: hit ‘esc‘, then type ‘:wq‘

What this means is : There's a directory located at " (tilde)/" root called matplotlib. so cd into (tilde)/.matplotlib. Create a file called matplotlibrc in (tilde)/.matplotlib/matplotlibrc there and add the following code: backend: TkAgg

2.6 TSP Dataset requirements

1. One city per row
2. Each city row, needs to start with an integer (be enumerated), then followed by x coordination then followed by the y coordination. Each seperated by a whitespace.
3. Coordination has to be x and y coordinates (euclidean), not latitude and longitude.

2.7 MNIST Dataset requirements

1. One case per row. FORMAT: F0,...,Fn,T
2. Supported separators [’,’;’]
3. Features has to be Integers / Real numbers
4. Labels has to be Integers / Real numbers

2.7.1 Solving TSP

When solving TSP, all the neurons starts in a circle in the middle of the input points. When a input point is chosen and a winner neuron is calculated, the winner neuron and it's neighbours is moved towards the chosen input in the coordinate-system. After a while all the inputs will have neurons moving towards them. After N number of epochs, all inputs will have neurons cloes to them, the learning rate is now very small and the neighbourhood size of each neuron is so small so that almost no changes are done in the upcoming epochs. When this happens a convergence has been reach and it is no point to continue the algorithm.

2.7.2 Classifying MNIST

The way SOM works to solve Mnist, Maps the input image to a neuron (a neuron represents a image, since one neuron has 784 pixels (weights)).

2.7.3 Solving TSP vs MNIST

SOM does not know what the input should be when solving TSP, so neurons can not be associated with a target/class, which they can when classifying MNIST. Because when training MNIST we can count how many time a neuron wins, indicating that the neuron thinks it is for example a 3. Since the neuron thinks it is a 3, in the majority of the cases for that K-th traning epoc, the neuron gets labeled as a 3. This is not SUPERVISED learning, since we do not know if the neuron thinks it is the right class. And there is no backwards updating of the weights and neurons based on how wrong it thinks it is.

2.8 Performance Tests

2.8.1 Good configuration specifications

Abbreviation : Full name : Type

1. TR : Number of Training cases : Int
2. TE : Number of Testing cases : Int
3. N : Number of Neurons : Int
4. W : Number of Neuron weights AND input weights : Int
5. C : Number of Classes / targets : Int
6. C-Int. : Classification interval : Int
7. I-NBS : Initial Neighborhood Size : Int
8. NB Const. : Neighborhood Constant : Int
9. LR Const. : Learning Rate Constant : Int
10. T : Time to run : Seconds
11. TER : Result on Test cases : %
12. TRR : Result on Training cases : %

Configuration													
Dataset	TR	TE	N	W	C	Epoch	C-Int.	I-NBS	NB Const	LR	T	TER	TRR
TSP													
1	-	-	2	-	-	40	-	-	15	4	-	-	1.05%
2	-	-	2	-	-	40	-	-	15	4	-	-	1.05%
3	-	-	2	-	-	40	-	-	15	4	-	-	1.07%
4	-	-	2	-	-	40	-	-	15	4	-	-	1.04%
5	-	-	2	-	-	40	-	-	15	4	-	-	1.08%
6	-	-	2	-	-	40	-	-	15	2	-	-	1.05%
7	-	-	2	-	-	40	-	-	15	4	-	-	1.05%
8	-	-	3	-	-	40	-	-	15	2	-	-	1.08%
MNIST													
	1000	100	200	784	10	50	10	10	10	16	8 min	78%	87%
	500	100	115	784	10	40	10	12	16	20	3.4 min	76%	85.8%
	500	100	100	784	10	40	10	10	10	1	2.8 min	55%	62.8%
	500	100	100	784	10	40	10	1	10	10	2.55 min	75%	82.2%
	500	100	100	784	10	40	10	1	10	20	2.09 min	80%	85.8%
	750	200	100	784	10	40	10	15	15	20	6.84 min	68.5%	79.7%
	500	100	100	784	10	50	10	20	10	1	3.31 min	33.0%	55.2%
	2000	100	100	784	10	3	10	1	0	0	8.19 min	25.0%	28.5%
DEMO DAY													
MNIST	1000	100	200	784	10	50	10	10	10	16	8 min	78%	87%

3 Detailed Explanation of SOM

3.1 Important parameters

1. Number of neurons - Should be number GTE to class * 2, If there are many neurons there is a better chance to find a neuron that resembles the input.

2. Learning rate - Really important

3. Number of epochs - the more epochs, the more we can learn! Watch out so that you doent over train that ...

4. Neighborhood Size - Really important. If set to 0, means that no Neighbors should be updated, Thus no neighborhood and topological map.

5. Neighborhood Membership and function - The closer neurons are to winning neurons, the more there weights should be adjusted.

The relationship between neighborhood size and learning rate is important and difficult to find correct. As in all AI networks, the gradient decent on learning rate and in SOMs the neighborhood size, are really important. IF we have to low descending learning rate, we need more epochs, and if the neighborhood decreases to slow, it can make defining the neighborhood difficult, and needs more epochs to make the neighborhood size so low that only the correct neighborhood and neuron is fine tuned.

3.2 Input layer

3.2.1 TSP

TSP takes in 2 features, coordinates in a graph (or city locations). We want to map these to neurons, and make the neurons move to the input. The closer the better.

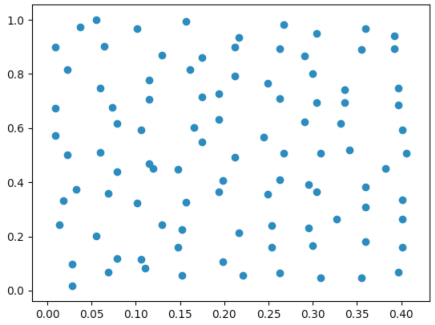

Figure 1: Input to TSP

3.2.2 MNIST

The input an 28x28pixel image represented as vector containing 784 pixels - because 28*28 = 784. So one image is an array of 784 pixels. The imput images are different handwritings of numbers between 0 and 9 - giving 10 classes.

3.3 Output layer

3.3.1 TSP

We want to make the neurons snuggle up as tight as possible to the input, then finding out which neuron is closes to which input. Then we can find the shortest path, giving the best route. When we have found the closes neuron to each input, we can take the distance between all the closest neurons, sum it up (remember to take between first and last neuron).

3.3.2 Input vs Output

3.3.3 MNIST

We want to train the network to output the same number / classification as what is put in.

3.4 Proper vs Poorly trained

Here we show Picture of MNIST neurons during training!

The learning / training is done by adjusting neuron weights, so that the winning neuron and its neighbors gets more similar to the input, and creates a topological map.

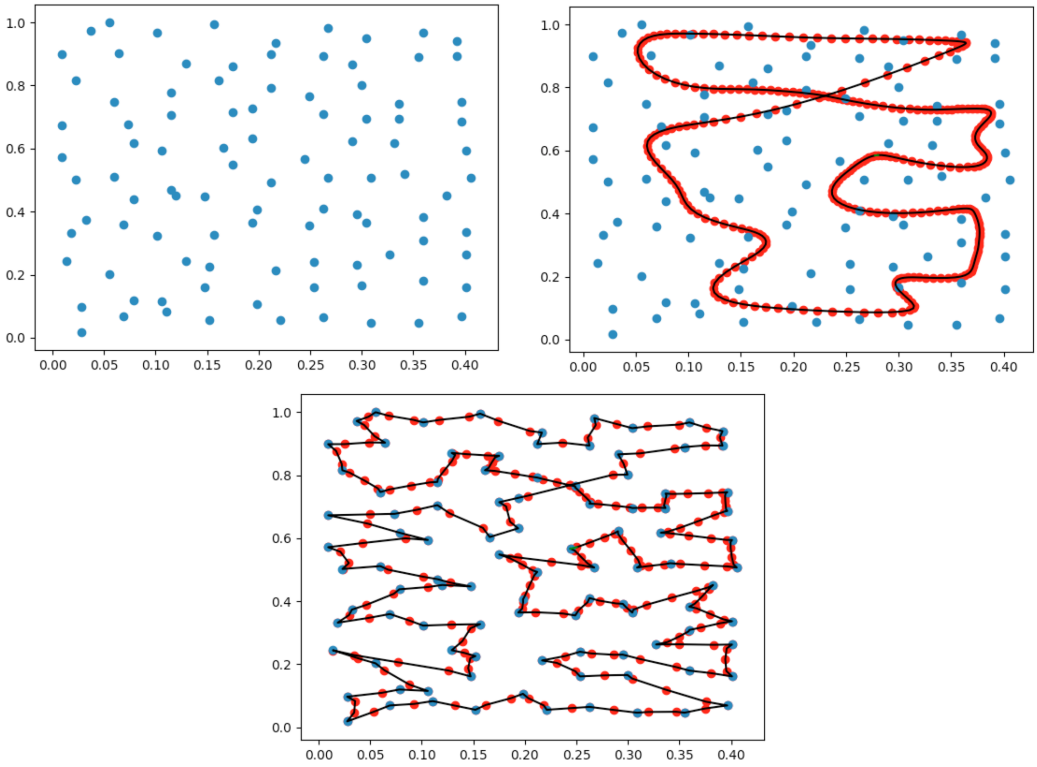

Figure 2: Input and wanted Output

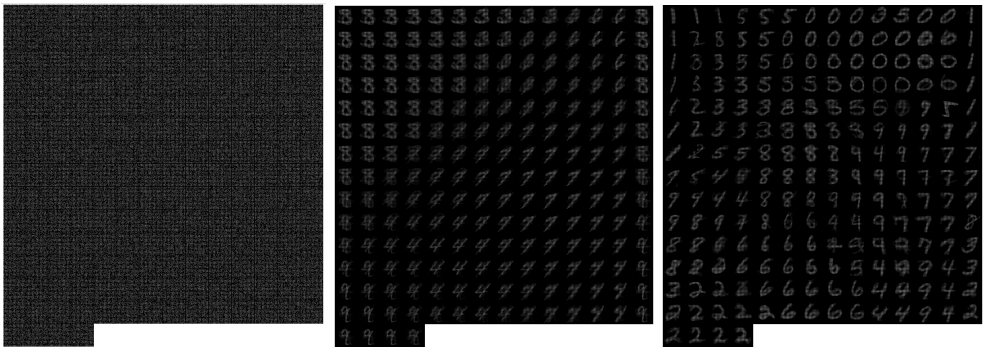

Figure 3: Comparison of neurons

4 Conclusion

This was an fun and educational project and a gentle introduction to unsupervised learning. "Also learned that it is hard to write performance worthy Python code...- even with numpy"