Kohonen Map On Hand Written Text

Project in Unsupervised Neural Network at EPFL

Hwang Chanhee
chanhee.hwang@epfl.ch
EPFL

Olagoke Lukman O.
lukman.olagoke@epfl.ch
EPFL

***Abstract*—Neural networks are systems composed of many simple processing elements (Neurons) operating in parallel whose function is determined by Network Strucuture , weights, and processing performed at the computing elements or nodes. We present here a type of Neural Nets called Kohonen self-organizing map and apply it in the recognition of hand written digits. We demonstrate how different parameters affects the results of such map and the criteria that can help us guarantee convergence.**

I. Introduction

A. *Self Organizing Map (SOM)*

The Self-Organizing map (SOM) is a computational principle that forms an ordered nonlinear projection of high dimensional input data items into a low-dimensional, usually 2-dimensional regular grid. The grid points are called *nodes*.The display produced onto the nodes can be regarded as similarity graph. Consider that some distance,

$$d(X_i, X_j)$$

between any kind of items X_i and X_j can be defined: pairs of items that have a small mutual distance will then be mapped onto the nearby nodes. On the other hand larger distances between items will in general not be preserved in the SOM projection, not even approximately.

The SOM algorithm involves an iterative procedure to generate a representation of the input space by a discrete set of weight vectors, which are associated with some neurons in the network(paper). The goal of the algorithm is to preserve and transform similarity relationships among input patterns into spatial relationship in the network. However this map formation process is affected by a number of challenging problems such as *type and number of optimal representations, the convergence to optimal representation, convergence speed as a function of algorithm parameters and avoidance of suboptimal representation.*

We will focus on some of these problems and shed light on some solutions. In particular it will be shown that convergence is affected by the so-called Neighborhood function-if the neighborhood function is monotonically decreasing and convex then there exist a no stable state other than the ordered ones(kohonen). Moreover in our implementations we have used Gaussain function as our neighborhood function so that these criteria can only be met if its "learning rate" and "width of kernel" are properly balanced. We will then show that that since the SOM represent a non linear projection of the probability density function p(x) of the high dimensional input onto a one- or two-dimension display, the convergence condition means that each best matching weingt (w*) must coincide with the centroid p(x) over the respective region of influence.

II. SOM MODEL

If X_t is an n-dimensional vectorial input item identified by the variable t (a running index of the samples and the iteration steps) and if $\{\mathrm{w}_i t\}$ is a spatially ordered set of vectorial models arranged on the grid, where i is the index of the node then the SOM algorithm can be summarised as:

1) Step1: Define the $\{\mathrm{w}_i t\}$ closest to X_t :

$$c = arg\min_i\{|X_t - w_i t|\}$$

The best matching node is called the *winner*. In this representation the winner is found by direct comparison of X_t with all $\mathrm{w}_i t$: the function that detrmines the winner is called the winner take all function.

2) Step 2: Correct the $\mathrm{w}_c t$ and $\mathrm{w}_i t$ int the neighborhood of $\mathrm{w}_c t$ on the grid towards $\mathrm{X}_i t$:

$$w_i(t+1) = w_i t + h_{ci}(t)[X(t) - w_i(t)] \qquad (1.1)$$

$\mathrm{h}_{ci}(t)$ is called the *neighborhood function*. It is similar to a smoothing kernel that has its maximum at the grid point i = c and its value decreases monotonically with increasing spatial difference between grid points i and c. Also $\mathrm{h}_{ci}(t)$ decreases with increasing sample index t and becomes narrower. It can be shown that as t tends to infinity.

The procedure is iterated many times. And we ask does $\mathrm{w}_c(t)$ finally coincides with X_t or put simply does the process converge: If the X_t and $\mathrm{w}_c t$ are Euclidean distance the convergence has been proven by Cheng[site paper]. We note that there are no fixed boundaries values for the model at the edges the grid: the values are determined freely in a regression process, when the neighborhood set N_i of the edge nodes is made to contain nodes from inside of the grid.

Definitions: Define the state of a SOM

$$\mathbf{w} : A \mapsto V$$

as a map from a discrete final set A of neurons to a complex subset V of R^d the input space, mapping each neuron j to its weight w_j. Let $C_k \in$ A be the adjacency structure of an undirected graph without weights with the set A of neurons as vertex set. This graph is called the *Kohonen graph*. The

Kohonen graph induces a metric d_A on A by defining $d_A(i,j)$ as the minimum distance between two neurons i,j ∈ A where $d_A(i,j)$ = 1 for two adjacent neuron. Unless otherwise stated we will use the Euclidean distance on V, d_V. Given an initial SOM state **w(0)** a SOM training sequrnce will be defined by applying learning rule of (1.1).

III. Organization Measure (Ordering) in SOM

Knowing what an ordered state is will help us in tackling convergence of the SOM map. We sate two key properties that will help qualify a function

$$\mu : W_{A,V} \mapsto R$$

to be called organization measure on a SOM. These are:

1) It should qualify the process of self-organization during training i.e its values should decrease on average
2) It should measure the quality of embedding of A into the data manifold.

The first condition is important for mathematical analysis and the second one for real applications of SOM. Several organization measures have been defined (soft) such as Inversion measure , Entropy, Energy function Measure , Measures of curvature etc.

But we will focus more on :

- similarity and metric measures (a)
- Data Oriented Measure: Intrinsic distance measure. (b)

In (a), we say that given a metric $\langle X, d_X \rangle$ and $\langle A, d_A \rangle$ a map $M : X \mapsto$ V is called similarity preserving if :

$$\forall x_1, x_2, x_3, x_4 \in X : d_X(x_1, x_2) < d_X(x_3, x_4)$$

$$\Rightarrow d_A(M(x_1), M(x_2)) \leq d_A(M(x_3), M(x_4))$$

Proposition a: if $\langle X, d_X \rangle$ and $\langle A, d_A \rangle$ be identical metric space with countable subsets. If M is a bijection from both M and M^{-1} then M is a (topographic) homeomorphism and X and Y are topologically equivalent through M [paper]. We then define a optimal map representation as a stationary state for which this proposition holds.[paper]. In (b) we introduce as measure the shortest mapped distance between the neurons whose weight are closest and seconclosest to a given data point in input space.

Formally, let x ∈ V be an input vector, i*(x) be the neuron closest, and j*(x) the neuron second closest to the input. The we define the intrinsic distance as:

$$d(x) = d_V(x, w_{i*(x)} + min_{S \in S_{i*j*}} \sum_{s_k, s_{k+1} \in S} d_V(w_{s_k}, w_{s_k+1})$$

where S_{i*j} represent the set of path starting at node i*(x) and ending at node j*(x) with edges from the Kohonen graph C_k and S ∈ S_{i*j} are set of edges $(s_k, s_{(k+1)})$. Thus i*(x) and j*(x) can be regarded as neurons that represent the data input "similarly well" in input space.

IV. Experimental Set Up And Results

A. Experimental set Up

In the set up we are given data set and labels to work with. The first step is to get the target digits to the labels(Boolean true if the digit is present and false if absent). This will be fed to the train set which we project the the digits in a nonlinear way onto the Kohonen map. We will test and tweak different parameters in other to observe when the algorithm actually converges. We will provide plots to show this.

B. Parameters

The list of our parameter is provided below:

- Neighborhood function : Gaussian function
- kernel
- learning rate
- Number of iterations
- Size of Kohonen map
-

Theses are the parameters that we fed into the algorithm. We observe that by changing these parameters we were able to gurantee convergence. In particular we will provide provides to justify some of the theoretical prepositions we made before.

C. Gaussian Function effect on algorithm

Thw Gaussian function which we used as our smoothing kernal carries the learning rate term and the satndard deviation. We define the Gaussian Function as :

$$\alpha(t).\frac{\exp(|r_c - r_i|^2)}{2\sigma^2(t)}$$

where r_c and r_i are the centre(winning) node and neighborhood node respectively. The choice of $\alpha(t)$ and $\sigma(t)$ is very crucial. If the $\sigma(t)$ is too small then the Kohonen map will not be ordered globally. Instead we will observe various kinds of parcellations of the map, between which various the ordering direction changes continuously. This is illustrated below setting $\sigma(t)$ to 3 and using a small $\alpha(t)$. The way to ensure convergence is to increase the number of iterations. Only then are we able to observe some ordering and convergence. In our implementation, we have used a threshold value to determine the minimum deviation or perturbation from the optimal state. It is then obvious that setting a very low value will make the algorithm run well above the point for which it has converged. However, it has been shown by [koh] [fuzzy] that once the weights are ordered , they cannot become disordered on further updating.

Fig1 shows the change in convergence threshold values with time. As we can see the zero level or below was not reached - it has not flatten out.

Fig2 shows the variation of the Gaussian function with time . We observe that it decreases with time which is actually the requirement for a neighborhood function. Convergence is attained by increasing the number of iterations. This gives the model more time to learn enough though the with of learning rate and kernel is small .

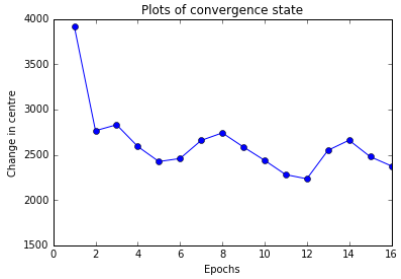

Fig. 1: Graph shows that map does not converge.

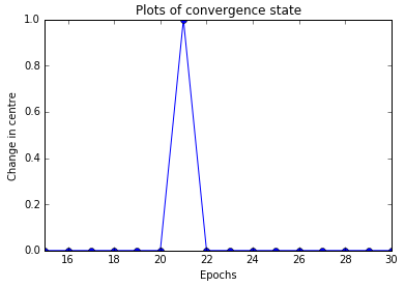

Fig. 2: in Gauss Function.

D. Ensuring Convergence and Accurate digit prototype representation

This can be achieved by achieved by starting with a fairly wide kernel width of the Gaussian function and letting this shrink with time. We will also let $\alpha(t)$ start large and shrink with time. If $\alpha(t)$ is constant as in our first run of the algorithm, then we choose $\alpha(t)$ of reasonable average size. We found values between 0.7 and 1.0 to be reasonable. In addition if the width is constant as in our first run of the algorithm then we have to choose width and learning rate that are within good range. If one is too large than the other then convergence wont be guaranteed.

Below we provide some plots of the run of the algorithm that satisfies the condition we have explained above.

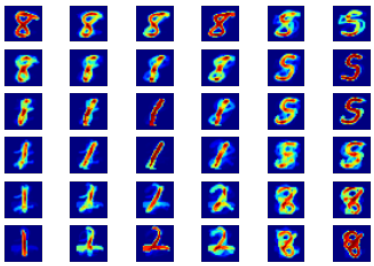

Fig. 3: Fair Representation of prototype

In the above we observe good representation of the digit by the prototype that best represent it. We also plot the convergence graph .

If we plot the convergence criteria graph we have.

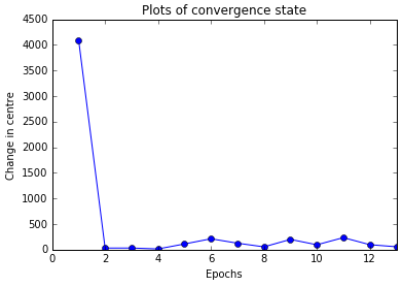

Fig. 4: Convergence observed Compare with Fig1.

E. Learning rate and Width changing with Time

We can use a linear, quadratic or exponential function for our learning rate. For the first few iterations $\alpha(t)$ should start with a large value and decrease with time. The ordering of the weights will generally occur at the initial period. After ordering phase learning rate will generally assume a constant value. In particular we observed that $\alpha(t)$ increases convergence rate when implemented as decreasing time function. In particular the text mappings becomes sharper and distinct in mapping regions for such implementations - this can be checked in the ipython notebook that comes with this paper. We have used the following rate decrease :

$$\alpha(t) = (0.9)(\frac{1-t}{1000})$$

as an implementation of changing learning rate.

The average kernel width defines in some way the stiffness of "elastic surface" to be fitted with data points. The optimal width value depends on size of map. This is because we generally have to start a large width as big as the diameter of the network in some cases and allow the width to decrease gradually until it attains a stable value. When decreasing $\alpha(t)$ and $\sigma(t)$ are combined the model becomes more powerful and attainment of convergence is obvious.

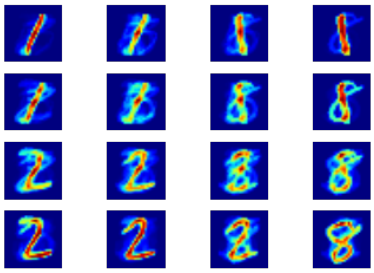

Fig. 5: Map with Reduce size with changing rate and kernel

F. Large size of map (K).

One obvious observation is that a large map gave blurred digits representation than a map small k. This can be seen clearly from the figure below comparing low and high size of map respectively:

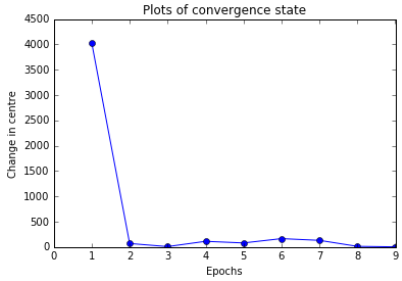

Fig. 6: Convergence curve for map with small learning rate and kernel width

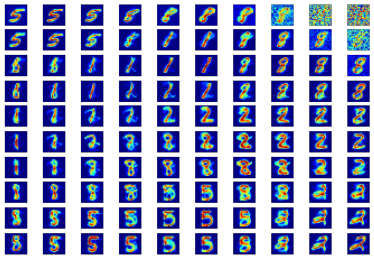

Fig. 7: Map with size of Map increased

This is true for maps sizes 3,20,8 which we experimented with. Large size of K will definitely affect the the choice of parameters - after all this is the space we want to project to. In general for very large maps for really large maps appropriate selection $\alpha(t)$ is needed and this involves more rigorous proof than would be required here.

V. Conclusion

We conclude that the convergence of SOM map is a very debated issue in the research community and there are no hard and fast rules to solve it. Convergence of SOM map has been proved in the following partial cases for 1D , 2D input and topology, discrete-valued input data and some variants of the SOM. But overall it still constitute a ill-posed problem.

If there is anything we would like to ascertain from this work it would be the fact that much is still left to be done. In particular we observed that the threshold value we set fro convergence affect the convergence rate - which is reasonable. But how large should this threshold value be remains unsolved. We observed that for lower threshold values the algorithm stops and also reports when this has occurred but for very small threshold values the loop almost always execute to the end.

However a good choice of learning rate, kernel and generally neighborhood function makes the running time of the algorithm bearable. Particularly choosing those parameters as functions that vary with time has shown to aid convergence of the set up.

A general rule of thumb which has been reported to experimentally to assist the statistical accuracy is that the number of steps must be at least 500 times the number of network nodes. On the other hand the number of components in input space has no effect on the number of iteration steps. This proposition works well for small maps - indeed our experimental set up of algorithm confirms this. But it fails when we have large size map. With large size maps we become strict on choice of parameters.

We have provided Ipython notebook to come with this work. More importantly we have shown the effects discussed here in a way more visual appealing way.