Evolving Tree Algorithm Modifications

Vincenzo Cannella[1], Riccardo Rizzo[2], and Roberto Pirrone[1]

[1] DINFO - University of Palermo
Viale delle Scienze, 90128 Palermo, Italy
[2] ICAR - Italian National Research Council
Viale delle Scienze, 90128 Palermo, Italy
ricrizzo@pa.icar.cnr.it

Abstract. There are many variants of the original self-organizing neural map algorithm proposed by Kohonen. One of the most recent is the Evolving Tree, a tree-shaped self-organizing network which has many interesting characteristics. This network builds a tree structure splitting the input dataset during learning. This paper presents a speed-up modification of the original training algorithm useful when the Evolving Tree network is used with complex data as images or video. After a measurement of the effectiveness an application of the modified algorithm in image segmentation is presented.

1 Introduction

Growing neural networks are capable to adapt the number of neural units to the input patterns distribution. Many growing neural nets add new units at fixed pace during the learning procedure: new units are added near the unit that accumulates the greater approximation error (as in Growing Neural Gas [8]) or when there is not a neural unit that matches the input with a sufficient approximation (as in Growing When Required [2]). The evolving tree [6] is a hierarchical growing neural network with a tree structure and the neural units on each node of the tree. This neural network is attractive because it builds a gross clustering of the input patterns and then go on using a splitting procedure during the training phase. A neuron of the network is split if it is the best matching unit too often, meaning that its cluster is too populated. The network growing is not limited during learning.

This spitting and learning procedure builds a natural hierarchy of clusters that can be easily exploited. The training procedure of this network requires more time as the number of neurons increases, a drawback that can be serious with complex datasets (high dimensional patterns or huge amount of data).

In this paper we present an application of this network to the image segmentation using a faster learning procedure that can overcame the slow training process. Image segmentation is a natural testbed for the speed-up due to the large set of data and it highlights also the performances of the network in clustering. The images are segmented by splitting an initial gross segmentation in a "natural" region splitting procedure.

The paper is organized as follows: the next subsection explains the evolving tree algorithm, then the algorithm simplification is explained following with some results using some artificial two dimension input distributions, then the image segmentation results are shown and commented.

2 The Evolving Tree Algorithm

The neural units of the Evolving Tree network have two parameters: the weights vector (considered as one parameter) and a counter b_i that counts the times the unit is the winner unit. The learning algorithm is based on a top-down hierarchical process where the input pattern is passed from the root node of the tree to the leaf nodes. When an input vector is presented to an unit it first checks if it is a leaf node or not. If the unit is not a leaf then it calculates the distance of the input pattern from the weight vectors of its children, finds the winner unit, and passes the input to it, then the former steps are repeated. If the node is a leaf of the tree its weights are modified using the formula:

$$w_i(t+1) = w_i(t) + \alpha(t) h_{ci}(t) \left[x(t) - w_i(t)\right] \tag{1}$$

and the winner unit counter b_i is increased.

When the b_i parameter reaches a threshold θ the node is split into fixed number of children units. After the splitting the split neural unit will not learn anymore, because only the leaf nodes of the tree are corrected, this is a behavior that prevents a unit to learn too much [5] and resembles the "Conscience" mechanism [7]. This also means that a tree node is frozen after it was θ times the winning unit.

In the original algorithm the weight correction is propagated to the unit neighborhoods on the tree according to the winner take most principle. The neighborhood function is the usual gaussian neighborhood:

$$h_{ci}(t) = \exp\left(-\frac{\|r_c - r_i\|^2}{2 * \sigma^2}\right) \tag{2}$$

and the distance between unit c and unit i $\|r_c - r_i\|$ is calculated "on the tree" counting the number of "hops" from the winner leaf unit c to the other leaf units of the tree, as described in the original papers [3], [6], [5].

Each network unit is trained for θ steps at most, and considering the usual time decay of $\alpha(t)$ learning parameter reported in eq.3, units that are deep in the hierarchy receive a minor correction:

$$\alpha(t) = \alpha_{max}\left(\frac{\alpha_{min}}{\alpha_{max}}\right)^{\frac{t}{t_{max}}} \tag{3}$$

.

In eq.3 t_{max} is the total number of learning steps.

3 Considerations about the E-Tree algorithm

There are two considerations about the evolving tree that are worthwhile. The first one is related to the role of the neighborhood function, the second one is related to the learning parameter α and its time dependency.

As in all self–organizing networks the neighborhood function in eq. 2 is used to train the neural units that are near the winner unit. Units that are near each other on the tree can be children of different nodes, for example, according to the original paper, if B_1 is the winner unit then also A_1 and A_2 are updated using the neighborhood function. This means that A_1 and A_2 are "moved" toward B_1 in the input space (see fig 3), and this movement can move A_1 and A_2 outside the Voronoi region V_A of unit A, and toward the Voronoi region V_B of B. Due to this mechanism the winner unit obtained using the top-down search on the tree structure can bring to a unit that is not the global best matching unit as defined in:

$$bmu = \arg\left\{\min_{i \in N}\{\|x - w_i\|\}\right\} \tag{4}$$

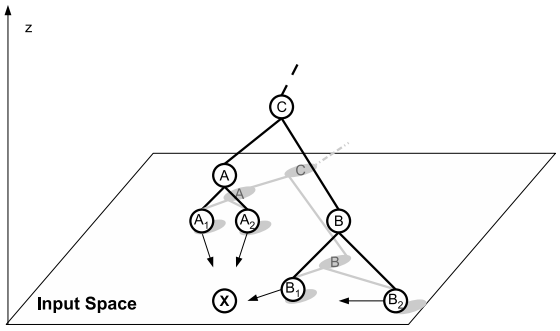


Fig. 1. Correction of the neighborhood units in Evolving Tree network: x is the input pattern. Weight vectors of the neural units are vectors in the input space but, in order to highlight the tree structure another dimension z, orthogonal to the input space, was added.

This problem was highlighted in [5] where the comparison between global best matching unit calculated with the formula above and the winner unit obtained using the top-down search on the Evolving Tree are compared.

In self–organizing networks the neighborhood function has many purposes, the main two of them are: allowing the spread of the lattice in fixed topology networks, as in the SOM network [1], and ease the unit distribution in a network with a fixed number of neural units, as in Neural Gas [9]. Neighborhood function allows to have more neural units where the input patterns are more frequent.

In Evolving Tree network there is not a structure to organize and, from this point of view, it resembles the Neural Gas.

Moreover the learning algorithm of the Evolving Tree is organized in a *"learn–then–split(if necessary)"* sequence so that all the units are trained before splitting. The splitting procedure generates new units where input patterns are dense. For this reason there are not "dead units" and correction propagation has a little influence on the efficiency of the network. According to this observation, neighborhood function can be neglected reducing the algorithm complexity.

The learning parameter α in eq. 1 decays during the training session as indicated in eq. 3. Due to the nature of the algorithm that creates new units where there is an high input density, we thought that there is not need to have a learning parameter variable with time. Adding more units in areas where there are many input patterns the unit correction will automatically decay because there will be more neurons in the same input space area.

4 Evolving Tree Algorithm Modifications

Due to the considerations of the preceding section, the neighborhood function $h_{ci}(t)$ seems not to be necessary, so that it can be reduced to the following:

$$h_{ci}(t) = \begin{cases} 1 \text{ if } i = c \text{ (i.e. unit i is the winning)} \\ 0 \text{ otherwise} \end{cases} \tag{5}$$

and the algorithm became similar to the hard competitive learning; moreover even the learning parameter α can be a fixed value, α_{const}.

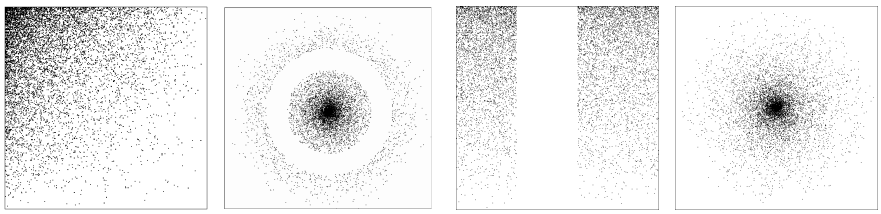
The proposed algorithm modifications, training with constant learning parameter α and /or without neighborhood function $h_{ci}(t)$, are tested using the four non uniform distributions in fig. 2, and the four combinations are summarized in table 1

Table 1. Algorithm modifications

Condition label	α	$h_{ci}(t)$
original algorithm	eq. 3	eq. 2
hard competitive learning	eq. 3	eq. 5
soft competitive learning, constant learning rate	α_{const}	eq. 2
hard competitive learning, constant learning rate	α_{const}	eq. 5

We implemented the learning algorithm in [6], not the version in [5]; for all the experiments the topology is a binary tree and the splitting threshold is $\theta = 400$. If α is variable the values are $\alpha_{max} = 0.5$, $\alpha_{min} = 0.05$, where α is constant is $\alpha_{const} = 0.3$.

For each distribution 10 training session are performed: the first training session is made of $t_{max} = 50.000$ learning steps, the second is made of $t_{max} = 100.000$ training steps and so on to $t_{max} = 500.000$. It should be noticed that due to the growing nature of the algorithm the number of neurons is slightly different from a training session to another but the differences can be neglected.



(a) (b) (c) (d)

Fig. 2. Input distributions used for the algorithm evaluation

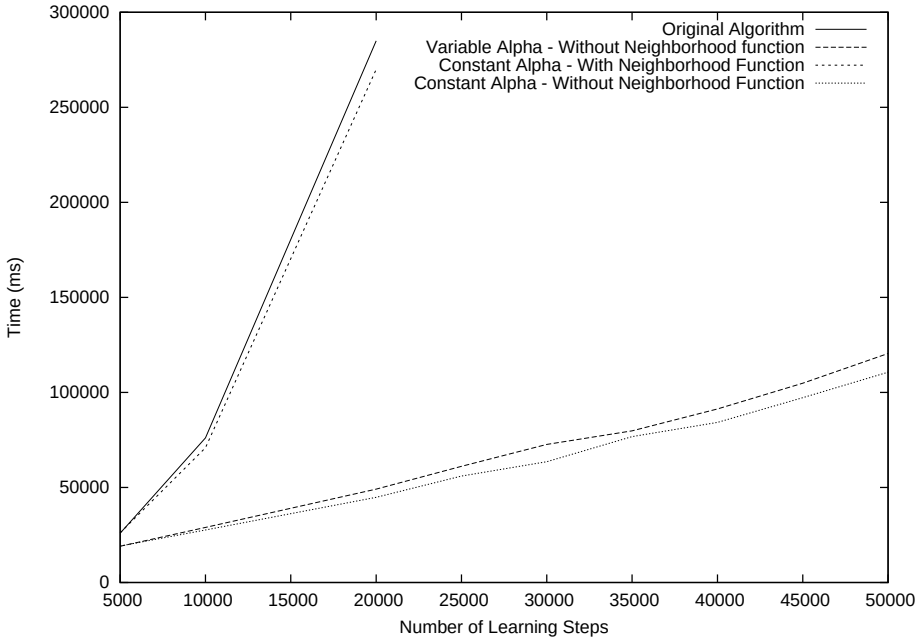


Fig. 3. Mean results for the distributions in fig. 2: the graphs show the processing time values obtained.

Fig. 3 shows that without the neighborhood function the network implementation is faster than the original one and the processing time is linear with the number of units.

In order to compare the results obtained from the algorithms two parameters were used: the entropy and the winning unit calculation.

If each unit is considered as a code vector of a codebook, the entropy maximization ensures that the quantization intervals are used with the same frequency during the quantification of the input signal. If the Evolving Tree network has N neural leaf units the input manifold is divided into V_i $i = 1, 2, \ldots N$ regions. After the learning phase, the probability that the input pattern v falls in the interval V_i should be $p(V_i) = \frac{1}{N}$, the entropy is calculated as

$$H = -\sum_{i=1}^{N} p\left(V_i\right) * \log\left[p\left(V_i\right)\right] \tag{6}$$

and the maximum theoretical value will be:

$$H_{max} = \log(N). \tag{7}$$

So using the entropy value, calculated at the end of the learning phase, it is possible to evaluate the distribution of the leaf node of the Evolving Tree in the input space.

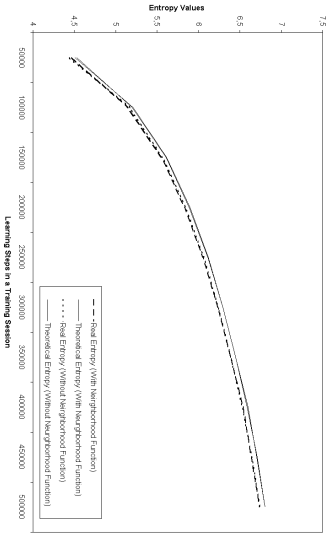


Fig. 4. Mean results for the distributions in fig. 2: the graphs show the comparison of the entropy values obtained.

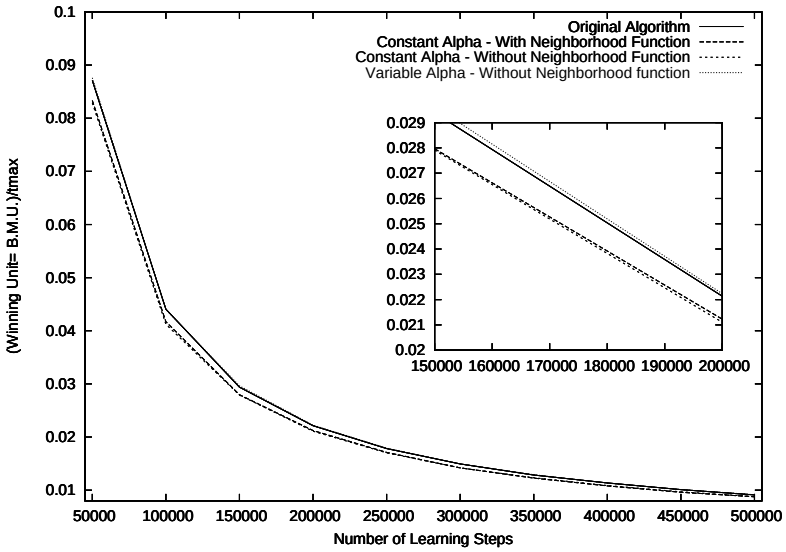


Fig. 5. Mean results for the distributions in fig. 2: the graphs show the number of times the winner unit obtained using the top-down research in Evolving Tree algorithm is the same of the global best matching unit.

Using the different learning algorithm reported in table 1 gives different number of neurons N and different values of theoretical entropy (e.g. eq. 7). The difference on number of neural units is below 4% and leads to a difference in entropy values of 0.8%. So that we consider the theoretical entropy value of the original algorithm as a reference for all the learning algorithm variations in table 1.

The comparison between the winning unit and the $b.m.u$ was used in [5] because the winning unit calculation using the top-down search on the Evolving Tree may not indicate the true best match unit obtained with the global search in the set of neural units. This method was already used in the same paper to evaluate the impact of some perturbations on the learning algorithm.

We report the number of times the global $b.m.u.$ obtained using the eq. 4 is coincident with the winning unit, compared to the total number of learning steps:

$$\frac{b.m.u. \equiv winning\ unit}{t_{max}} \tag{8}$$

the theoretical value should be 1 in all conditions.

The results obtained for different algorithms are compared in fig. 5 and in fig.4. Fig. 5 shows how many times the winning units is the global best matching unit. Fig. 4 shows the entropy values obtained for the original algorithm and the simplified algorithms. Differences due to the different neurons number can be neglected as said above.

Figures show that with a constant alpha the performances of the network are slightly worse.

5 Image Segmentation Application

In order to obtain the image segmentation the network is trained using a set of pattern X where each pattern $x \in X$is one for each pixel of the image. The input pattern x is obtained using the pixel position (i, j) and the color parameters H and S as shown below

$$x = [c_1 * i, c_1 * j,\ c_2 * H,\ c_2 * S]^T \tag{9}$$

where $c_1 = 0.1$ and $c_2 = 0.8$ are two constants used to weight the pixel position and the color information.

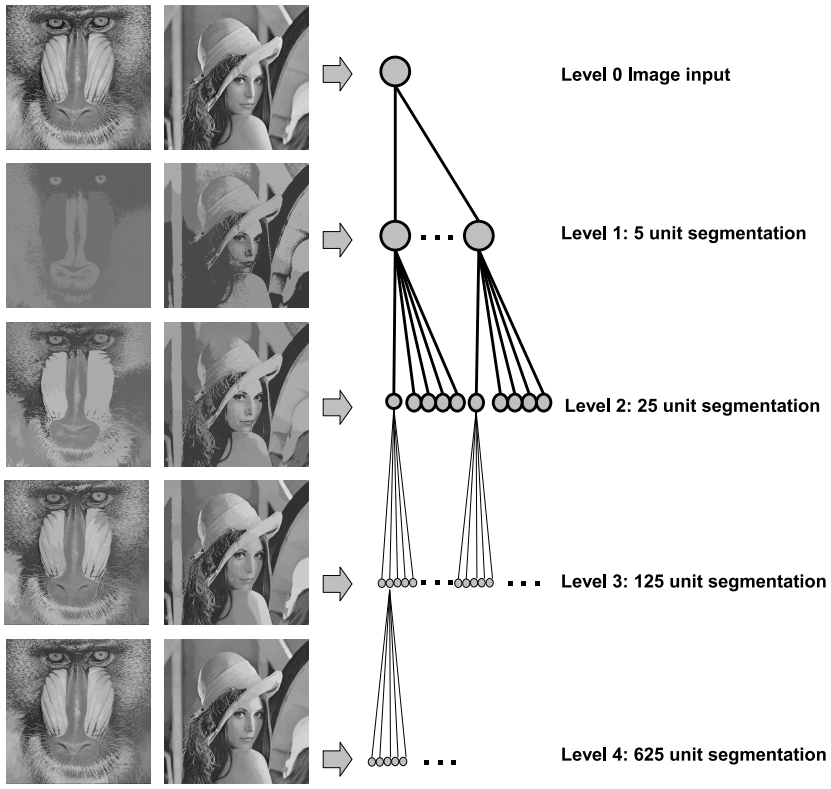


Fig. 6. The segmentation procedure.

After the learning on an image for segmentation the Evolving Tree builds a hierarchy as in fig. 6. The root of the network receives the image as input, the

unit is split in n children ($n = 5$ in fig. 6) and each children unit clusters the image pixels as shown in level 1 image. This is repeated for each unit at all the levels. In figs **??** and **??** on the left is shown the original image and on the right the segmentation results. The four segmented images are obtained on different levels of the network, as fig. 6 shows. The segmentation results in fig. 6 were obtained using the following parameters values : $\theta = 200$, $\alpha = 0.3$, 5 children for each node when splitting, and a learning time $t_{max} = 30000$. For the image Lena in fig. 6 where used the same parameters. The results of the image segmentation are on the left side of fig. 6.

6 Conclusions

The evolving Tree algorithm is an interesting and flexible growing algorithm inspired by the self–organizing map. The results presented demonstrates that it is possible to modify the algorithm with a small or null loss of performances. So the Evolving Tree can be adapted to the specific problem: for example neighborhood function can be neglected in order obtain a speed-up and the learning parameter can be a fixed value in order to use the network with non-stationary input distributions. The faster algorithm is useful when learning with complex data as images or videos. The image segmentation is a natural region splitting and hierarchy structure of the results obtained needs further investigations.

References

1. Kohonen, T. Self Organizing Maps. Springer Verlag (1997).
2. Marsland, S. and Shapiro, J. and Nehmzow, U. A Self-Organizing Network that Grows When Required. Neural Networks **15** (2002) 1041-1058.
3. Pakkanen, J. The Evolving Tree, a new kind of self-organizing neural network. Proceedings of the Workshop on Self-Organizing Maps '03 (2003) 311–316.
4. Pakkanen, J. and Iivarinen, J.: A Novel Self-Organizing Neural Network for Defect Image Classification. Proceedings of IJCNN (2004) 2553–2556.
5. Pakkanen, J. and Iivarinen, J. and Oja, E.: The Evolving Tree, a Hierarchical Tool for Unsupervised Data Analysis. Proceedings of IJCNN (2005) 1395–1399.
6. Pakkanen, J. and Iivarinen, J. and Oja, E. : The Evolving Tree — A Novel Self-Organizing Network for Data Analysis. Neural Processing Letters **20** (2004) 199-211.
7. DeSieno D.: Adding a conscience to competitive learning. Proc. ICNN'88, International Conference on Neural Networks, IEEE Service Center, Piscataway, N J, (1988) 117-124.
8. Fritzke B.: A growing neural gas network learns topologies. Advances in Neural Information Processing Systems, MIT Press, editors G. Tesauro and D. S. Touretzky and T. K. Leen, (1995) 625–632. .
9. T. M. Martinetz, S. G. Berkovich, K. J. Schulten: Neural Gas Network for Vector Quantization and its Application to Time-Series Prediction. IEEE Trans. on Neural Networks **4** (4) (1993) 558–569.