ESANN'2008 proceedings, European Symposium on
Artificial Neural Networks - Advances in Computational Intelligence and Learning
Bruges (Belgium), 23-25 April 2008, d-side publi., ISBN 2-930307-08-0.

Self-Organizing Maps for cyclic and unbounded graphs

M. Hagenbuchner[1], A. Sperduti[2], A.C. Tsoi[3] *

1- University of Wollongong, Wollongong, Australia.
2- University of Padova, Padova, Italy.
3- Hong Kong Baptist University, Kowloon, Hong Kong.

Abstract. This paper introduces a new concept to the processing of graph structured information using self organising map framework. Previous approaches to this problem were limited to the processing of bounded graphs. The computational complexity of such methods grows rapidly with the level of connectivity, and are restricted to the processing of positional graphs. The concept proposed in this paper addresses these issues by reducing the computational demand, and by allowing the processing of non-positional graphs. This is achieved by utilising the state space of the self organising map instead of the states of the nodes in the graph for processing.

1 Introduction

Self-Organizing Maps (SOMs) are unsupervised learning models that allow the mapping of high-dimensional data onto a low-dimensional computational grid (display space) preserving the topology of the input space [1]. Early versions of SOMs included models capable of processing vectorial data as well as continuous signals [1]. The processing of more general types of data structures such as graphs, while considered in [1] was not fully developed until more recently [2, 3].

Graphs provide a generic means for data representation. With graphs it is possible to encode relationships between entities in a dataset. Graphs contain vectors and data sequences as special cases, and hence, graph structured data can be regarded as a superset for the most common methods of data representation.

Traditional data processing methods, e.g. SOMs, multilayer perceptrons, rely on a vectorial representation of data. Pre-processing was necessary in order to "squash" graph structured information into a vectorial form. Such pre-processing can result in the loss of important structural or relational information.

Machine learning methods can be particularly suitable for data processing tasks involving graphs due to their proven insensitivity to noise and the ability to generalize to unseen data. An example of such machine learning approaches is the Graph Neural Network [4] which is capable of processing generic graphs in a supervised learning paradigm. This paper proposes a method in the unsupervised learning paradigm for the processing of graph structured data. A first work in this area is described in [2] which proposes a method known as the SOM for structured data (SOM-SD), capable of processing trees with bounded outdegree. The idea was consequently extended in [3] to allow the processing of more general types of bounded graphs, e.g. which can encode contextual information. The approach in [3] has become known as the Contextual SOM-SD (CSOM-SD). A key issue in using the SOM-SD or CSOM-SD is the requirement of knowing the maximum outdegree of graphs a priori. In this paper, a

*The first and third author wish to acknowledge financial support from an Australian Discovery Project grant DP077148 (2007 - 2009) which has been received to support this research.

new concept of processing graph structured information is proposed to enable SOMs to encode much more generic types of graphs, e.g. unbounded graphs. Unbounded graphs may contain cyclic paths, or have an infinite number of branches. The existing methods, i.e. SOM-SD and CSOM-SD, cannot deal with this class of graphs, since they cannot handle undirected arcs and unbounded graphs. In addition, the method proposed in this paper has a lower computational complexity compared with SOM-SD and CSOM-SD.

This paper is organized as follows: The SOM-SD is briefly described in Section 2. A new way for processing and training a SOM on structured data is proposed in Section 3. Section 4 and Section 5 give experimental results and conclusions respectively.

2 SOMs for structured data

An approach in [2] enabled SOMs to map graph structured information by processing individual nodes of a graph, and by incorporating topological information about a node's offsprings in a directed acyclic graph. The introduction of CSOM-SD [3] extended such capabilities towards the processing of undirected and cyclic graphs. Both approaches include the standard SOM as a special case, and when applied to graphs, are restricted to learning domains for which the upper bound of any node's degree is known a priori (e.g. the maximum number of neighbors for any node in a graph is known a priori). In addition, the computational demand for such methods can be prohibitive for learning problems involving a high level of connectivity. This is best described by looking at how the SOM-SD processes data.

The basic SOM consists of a q-dimensional lattice of *neurons* representing the display space. Every neuron i of the map is associated with an n-dimensional codebook vector $\mathbf{m}_i = (m_{i1}, \ldots, m_{in})^T$, where T transposes the vector. The neurons have a hexagonal neighborhood relationship; the most commonly used arrangement. The SOM is trained by updating the elements of $\mathbf{m}_i$ through a two step training procedure:

Step 1: One sample input vector $\mathbf{u}$ is randomly drawn from the input data set and its similarity to the codebook vectors is computed. When using the Euclidean distance measure, the winning neuron is obtained through:

$$r = \arg\min_i \|\mathbf{u} - \mathbf{m}_i\| \quad (1)$$

Step 2: $\mathbf{m}_r$ itself as well as its topological neighbours are moved closer to the input vector in the input space. The magnitude of the attraction is governed by the learning rate α and by a neighborhood function $f(\Delta_{ir})$, where Δ_{ir} is the topological distance between $\mathbf{m}_r$ and $\mathbf{m}_i$. The updating algorithm is given by:

$$\Delta\mathbf{m}_i = \alpha(t) f(\Delta_{ir})(\mathbf{m}_i - \mathbf{u}), \quad (2)$$

where α is a learning rate decreasing to 0 with time t, $f(.)$ is a neighborhood function. The most commonly used neighborhood function is the Gaussian function such as $f(\Delta_{ir}) = \exp\left(-\frac{\|\mathbf{l}_i - \mathbf{l}_r\|^2}{2\sigma(t)^2}\right)$, where the spread σ is called the neighborhood radius which decreases with time t, $\mathbf{l}_r$ and $\mathbf{l}_i$ are the coordinates of the winning neuron and the i-th neuron in the lattice respectively.

Steps 1 and 2 together constitute a single training step and they are repeated a given number of times. The number of training steps must be fixed prior to the commencement of the training process because the rate of convergence in the neighborhood function and hence the learning rate are calculated accordingly.

To allow for the processing of structured data this training algorithm is extended [2] by incorporating information about a node's neighbors. To achieve this, we assign the $\mathbf{u}_i$ to be the label of the i-th node in a graph, then form an input vector for the SOM by concatenating the label with the coordinates of the winning neuron of all neighbors (i.e., nodes reachable via a directed arc) of the node. These coordinates are referred to as the *state*s of neighbors. Formally, an input is defined as a vector $\mathbf{x}_i = (\mathbf{u}_i, \mathbf{y}_{ch[i]})$, where $\mathbf{y}_{ch[i]}$ is the concatenated list of states (coordinates of the winning neuron) of all the children of a node i. Since the size of vector $\mathbf{y}_{ch[i]}$ depends on the number of offsprings, and since the SOM training algorithm requires constant sized input vectors, padding with a default value is used for missing offsprings or for nodes with less than the maximum outdegree on a graph. The training algorithm of a SOM is altered to account for the fact that an input vector now contains hybrid information. Eq. 1 and Eq. 2 are respectively changed as follows:

$$r = \arg\min_i \|(\mathbf{x}_j - \mathbf{m}_i)^T \mathbf{\Lambda}\| \quad (3)$$

$$\Delta\mathbf{m}_i = \alpha(t) f(\Delta_{ir})(\mathbf{m}_i - \mathbf{u}) \quad (4)$$

where $\mathbf{\Lambda}$ is a $n \times n$ dimensional diagonal matrix; its diagonal elements $\lambda_{11} \cdots \lambda_{pp}$ are set to μ_1, all remaining diagonal elements are set to μ_2. The constant μ_1 influences the contribution of the data label component to the Euclidean distance, and μ_2 controls the influence of the states to the same distance measure.

A new step added to the training algorithm propagates structural information:

Step 3 The coordinates of the winning neuron are passed on to the parent node which in turn updates its vector $\mathbf{y}$ accordingly.

The SOM-SD requires the processing of data in a strict causal manner (i.e. from the leaf nodes towards the root). It is not possible to process nodes in any other order since otherwise in Step 3 it is possible that not all states of all neighbors are available. An extension which circumvents this problem has been proposed in the CSOM-SD [3]. The CSOM-SD builds on the SOM-SD by adding new steps to the training procedure to take both ancestors and descendants of a node into account. This is similar in nature to SOM-SD, and hence, the SOM-SD can be reduced to the CSOM-SD accordingly.

3 The GraphSOM

The computational demand of a SOM-SD grows with $Nk(p + qn)$, where N is the total number of nodes in the data set, k is the number of neurons on the map, p is the dimension of the data label attached to the nodes in a graph, q is the dimension of the map (typically $q = 2$), and n is the maximum number of neighbors of any node in a graph. Thus, the complexity of the training algorithm increases linearly. However, there is a dependence between N, k, and n in many practical learning problems. An increase in N, p, or n often requires a larger mapping space k. This is particularly the case for large scale learning problems that contain only few redundancies in the feature space covered by the data. Moreover, in many data mining applications, n can become very large (e.g. $n >> k$) which in turn can require large k. In other words, the computational complexity for large scale learning problems is close to a quadratic one.

A further observation is that a SOM-SD may present redundant information to the network because it concatenates state information of every neighbor to the network

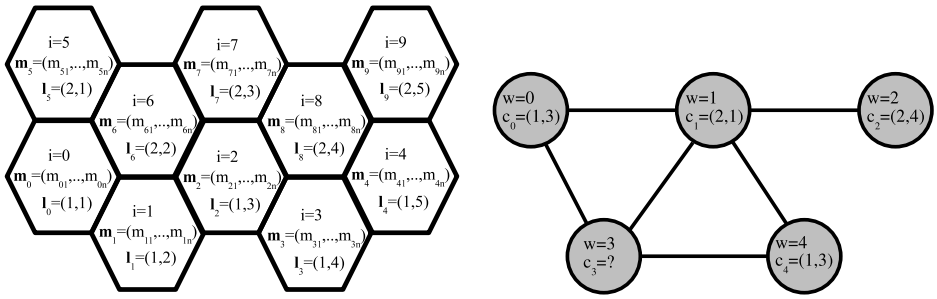

Fig. 1: A 2-dimensional map of size 2×5 (left), and an undirected graph (right). Each hexagon is a neuron. ID, codebook, and coordinate value for each neuron is shown. For each node, the node number, and coordinate of best matching codebook is shown.

input. Redundancies occur when several neighbors are mapped to the same location on the map. This likelihood increases with n, and becomes unavoidable when $n > k$. In addition, the SOM-SD can only process positional graphs since a change in the ordering of neighbors changes the input vector. There are many applications for which the order of neighbors is of no significance.

An approach that addresses these issues is by changing the way state information is added to an input vector. Instead of concatenating the mappings of all individual neighbors it is possible to simply concatenate the activation (state) of the map with respect to a node's neighbors. Formally, when processing a node i the input to the SOM is $\mathbf{x}_i = (\mathbf{u}_i, \mathbf{M}_{ne[i]})$, where $\mathbf{u}_i$ is defined as before, and $\mathbf{M}_{ne[i]}$ is a k dimensional vector containing the activation of the map $\mathbf{M}$ when presented with the neighbors of node i. An element M_j of the map is zero if none of the neighbors are mapped at the j-th neuron location, otherwise it is the number of neighbors that were mapped at that location. As an example, assume a SOM and graph as illustrated in Fig. 1. For simplicity we assume further that no data label is associated with any node in the graph. Then when processing node $w = 3$ the network input is the k dimensional vector $\mathbf{x}_3 = (0, 0, 2, 0, 0, 1, 0, 0, 0, 0)$. It can be observed that $\mathbf{x}_3$ summarizes the mappings of the neighbors of node 3 by listing the two activations at coordinate $(1, 3)$, the third neuron, and one activation at coordinate $(2, 1)$, the sixth neuron. Then the mapping of node 3 can be computed, and the information be passed onto its neighbors in the same way as is done for the SOM-SD. Note that the input remains unchanged regardless of the order of the neighbors. Note further that the dimension of $\mathbf{x}$ remains constant regardless of the maximum outdegree of any node in the dataset. We shall refer to this approach as the *GraphSOM*.

In comparison, the dimension of input vectors for the same graph when processed with SOM-SD is 6 (because $q = 2$ and $n = 3$). Thus, it can be observed that the number of computations with the GraphSOM is similar to CSOM-SD when processing small graphs. But since the input dimension is independent of the outdegree n, the GraphSOM becomes more and more efficient as the connectivity (and consequently the size) of a graph increases. When dealing with large maps then the approach can be approximated quite well by consolidating neighboring neurons into groups. Such approximation is valid since only those nodes are mapped in close proximity which

share the greatest similarities. This leaves the granularity of the mappings unchanged but allows a more compact representation of $\mathbf{M}_{ne[i]}$. For instance, when using a 2×5 grouping, the network input for node 3 becomes $\mathbf{x}_3 = (1, 2, 0, 0, 0)$. A more coarse grouping can be chosen to achieve greater speed gains at the cost of some accuracy in the representation of the state of the map.

Nodes can be processed in an arbitrary order by utilizing the mappings of neighbors from the previous iteration (if a mapping is not available at a current iteration). At the first iteration, any unknown mapping is assumed to be a random coordinate. The motivation for this approach is based on [3] which observes that the mappings between successive iterations can be assumed to become very similar as training proceeds. By using the mapping at a previous iteration, this is a valid approximation which allows the relaxation of the processing order of nodes.

Some advantages of the GraphSOM when compared to the (C)SOM-SD are:

- The computational demand is reduced when dealing with large graphs.
- The computational demand can be scaled arbitrarily through a grouping of neurons (at the cost of some inaccuracy in the representation of the graph).
- The input dimension is independent of the level of connectivity in a graph.
- The approach allows the processing of positional and non-positional graphs.
- It contains the CSOM-SD and the SOM-SD as special cases.

The approach is a generalization of the SOM-SD in that it can consider the order of nodes by simply concatenating the state of the map when processing each of the individual neighbors. The input vector in this case will be very sparse but the same behaviour as for the SOM-SD is obtained.

In the context of sequence processing, the idea to present the activation map as contextual information has been pursued by some researchers [6]. In particular, the Merge SOM model [6] stores compressed information of the winner in the previous time step, whereby the winner neuron is represented via its content rather than its location within the map. The content of a neuron i consists of the weight w_i and the context c_i. These two characteristics of neuron i are stored in a merged form, i.e. as linear combination of the two vectors. It should be stressed that this approach is different from the one proposed here.

4 Experiments

This section provides a preliminary assessment of the differences in performance between SOM-SD and the proposed GraphSOM when applied to a practical learning problem. The basis for the comparison is a recent application of the SOM-SD to the clustering of XML formatted scientific documents which was performed at an international competition (which consequently was won by the SOM-SD approach) on data mining in structured domains [5]. The data-mining competition defines Macro F1 and Micro F1 as the basis for the performance comparisons. The dataset consists of $12,107$ XML formatted documents, half of which are used for training, the remainder for testing purposes. XML is easily represented as a tree-like structure. The analysis of the dataset in [5] revealed that the corresponding graph structures in the training set feature a total of $3,966,123$ nodes, and a maximum outdegree of $1,023$. Training a SOM-SD on this

dataset was estimated to take $6,818$ days on 3GHz Pentium based hardware. As a consequence, pre-processing was applied in [5] by using some heuristics which reduced the number of nodes in the training set to $108,523$, and the maximum outdegree to 66. Training a SOM-SD consisting of $8,800$ neurons on these pre-processed data required 16 hours. The best performance obtained by the SOM-SD on this learning problem was: Macro F1 = 0.34, Micro F1 = 0.38.

The GraphSOM was applied to the same learning task but no pre-processing of the data was applied. The GraphSOM also used $8,800$ neurons though these were grouped into groups of 7 (a neuron and its six neighbors), and into groups of 18 (added 12 direct neighbors to the group of 7). This produced an input dimension of $1,258$ and 489 respectively. In comparison, the input dimension for the SOM-SD when using pre-processed data was $2 \times 66 = 132$. Despite a larger input dimension and a larger number of nodes in the training set for the GraphSOM, the training times were respectively 31 hours when using groups of 7, and 16 hours when using groups of 18. It turned out that most input vectors for the GraphSOM are sparse since the average outdegree for this dataset is just 48, and hence, Eq. 3 and Eq. 4 can be implemented much more efficiently. Additionally, the GraphSOM only requires to store the indexes of winning neurons rather than their coordinates which also contributed to a reduced computational demand. Using the groups of size 7, the performances were Macro F1=0.39 Micro F1=0.36. The group size 18 produced Macro F1=0.36, Micro F1=0.32. This shows that the learning task does not utilize positional information in the structure, that the pre-processing step for the SOM-SD may have removed some relevant features, and that the GraphSOM can process large datasets more efficiently and without requiring pre-processing.

5 Conclusions

This paper proposes a new way of processing structured information by a Self-Organizing Map. It was demonstrated that in comparison with preceding methods, the approach will reduce the computational demand when dealing with large graphs while increasing the types of graphs that can be encoded. The proposed method scales well with the size of the map by allowing the grouping of "topologically near" neurons into groups, and hence, allows the increase of speed at a minimal cost on accuracy. The SOM-SD and CSOM-SD methods are contained as special cases, and hence, can be assumed to exhibit the same behaviour when dealing with tree structured or positional graphs.

References

[1] T. Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, Berlin, Heidelberg, 1995.

[2] M. Hagenbuchner, A. Sperduti, and A.C. Tsoi. A self-organizing map for adaptive processing of structured data. *IEEE Transactions on Neural Networks*, 14(3):491–505, May 2003.

[3] M. Hagenbuchner, A. Sperduti, and A.C. Tsoi. Contextual processing of graphs using self-organizing maps. In *European symposium on Artificial Neural Networks*, 27 - 29 April 2005.

[4] F. Scarselli, S.L. Yong, M. Gori, M. Hagenbuchner, A.C. Tsoi, and M. Maggini. Graph neural networks for ranking web pages. In *Web Intelligence Conference*, 2005.

[5] M. Kc, M. Hagenbuchner, A.C. Tsoi, F. Scarselli, M. Gori, and S. Sperduti. Xml document mining using contextual self-organizing maps for structures. In *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 2007.

[6] M. Strickert and B. Hammer. Neural gas for sequences. In *WSOM'03*, pages pp 53–57, 2003.