ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 389-394

A general framework for unsupervised processing of structured data

Barbara Hammer[1], Alessio Micheli[2], and Alessandro Sperduti[2]

(1) University of Osnabrück, Department of Mathematics/Computer Science, Albrechtstraße 28, 49069 Osnabrück, Germany, e-mail: hammer@informatik.uni-osnabrueck.de

(2) University of Pisa, Department of Computer Science, Corso Italia 40, 56125 Pisa, Italy, e-mail: {micheli,perso}@di.unipi.it

Abstract. We propose a general framework for unsupervised recurrent and recursive networks. This proposal covers various popular approaches like standard self organizing maps (SOM), temporal Kohonen maps, recursive SOM, and SOM for structured data. We define Hebbian learning within this general framework. We show how approaches based on an energy function, like neural gas, can be transferred to this abstract framework so that proposals for new learning algorithms emerge.

1. Introduction

Supervised recurrent neural networks constitute a well established approach for modeling sequential data e.g. in language processing or time series prediction. They can naturally be generalized to so-called recursive networks such that more complex data structures, such as tree structures, can be dealt with [4]. Since symbolic terms possess a representation as a tree, this generalization has successfully been applied in various areas where symbolic or hybrid data structures arise such as theorem proving, chemistry, image processing, or natural language processing [1, 3]. The training method for recursive networks is a straightforward generalization of standard backpropagation through time. Moreover, important theoretical investigations from the field of feedforward and recurrent neural networks have been transferred to recursive networks [3, 6].

Unsupervised learning constitutes an alternative important paradigm for neural networks which has been successfully applied in data mining and visualization (see [8]). Since additional structural information is often available, a transfer of standard unsupervised learning methods to sequences and more complex tree structures would be valuable such that unsupervised recurrent or recursive networks arise. Up to now, only few approaches enlarge SOM to sequences: the temporal Kohonen map (TKM) [2] and the recursive SOM (RSOM) [11] and variations thereof. We are aware of only one approach which processes tree structured data and thus also sequences in an unsupervised way: the SOM for structured data (SOMSD) [10]. These models are trained by Hebbian learning. They have been applied to monitor standard time series or image data, so far. Since a clear measure of the success of unsupervised processing of structured data and a theoretical formulation of the above dynamics and training method are lacking, their success is measured on supervised classification tasks or, alternatively, experts rate the visualization obtained from the neural maps. The reported results for the above models seem very promising, however, we need a general theoretical framework for a deeper insight.

Here we propose a general framework which transfers the idea of recursive processing of complex data to the unsupervised scenario and covers TKM, RSOM, SOMSD, and the standard SOM. We show how Hebbian learning can be formulated within this approach. We propose alternative training methods based on energy functions, such that popular methods in the field of unsupervised learning can be directly transferred to this general framework. This is a first step to a general theory of unsupervised recurrent and recursive networks and training algorithms in this framework with a clear mathematical objective.

2. The processing dynamics

We are interested in binary trees with labels in some set $\mathcal{W}$ equipped with a metric $d_{\mathcal{W}}$. ξ denotes the empty tree and $a(t_1,t_2)$ a tree with root label a and subtrees t_1 and t_2. The generalization to trees with fan-out k will be immediate. We start by briefly recalling the approaches for unsupervised processing structured data which should be covered by the general framework:

SOMSD: SOMSD, described in [10], performs unsupervised processing of tree structured data. It processes labeled trees in a recursive way by a neural map starting from the leaves up to the root. For this purpose, a lattice of neurons is defined. Each neuron in the map is equipped with weights (w, r_1, r_2). Hereby, $w \in \mathcal{W}$ is contained in the set of possible labels, r_1 and r_2 are contained in $\mathbb{R}^d$, d denoting the dimensionality of the lattice. They correspond to possible indices of neurons. Given a tree $a(t_1,t_2)$ as input, its distance from neuron n with weights (w, r_1, r_2) is recursively computed by $\alpha\|a - w\| + \beta\|\iota(t_1) - r_1\| + \beta\|\iota(t_2) - r_2\|$. Hereby, $\iota(t_1)$ and $\iota(t_2)$ denote the indices of the neurons with smallest distance from t_1 and t_2, respectively. α, $\beta > 0$ are constants. $\|\cdot\|$ is the Euclidian metric. Training is performed in a Hebb style making iteratively the weights of the winner and its neighborhood in the lattice more similar to the actual input, i.e. w is adapted into the direction a, r_1 and r_2 are adapted into the directions $\iota(t_1)$ and $\iota(t_2)$, respectively.

RSOM: RSOM has originally been proposed for sequences in [11]. However, it can be generalized to tree structures. Even in this model a lattice of neurons is defined. Denote the number of neurons by N. The neurons are equipped with weights (w, r) where $w \in \mathcal{W}$. $r \in \mathbb{R}^N$ corresponds to the vector of activations of all neurons at the previous time step. The distance of a given sequence $[a_0, \ldots, a_l]$ over $\mathcal{W}$ from neuron n weighted with (w, r) is recursively computed by $\alpha\|a_0 - w\| + \beta\|E([a_1, \ldots, a_l]) - r\|$. Hereby, $E(s)$ for $s := [a_1, \ldots, a_l]$ denotes an exponentially transformed version of the vector of distances of s from all neurons; i.e. $E(s) = (\exp(-d_1(s)), \ldots, \exp(-d_N(s)))$ where $d_j(s)$ denotes the distance of s from neuron n_j. Again, Hebbian learning is used for training.

TKM: The temporal Kohonen map consists of a lattice of neurons where each neuron constitutes a leaky integrator [2]. The approach is defined for sequences over the set $\mathcal{W}$, only. Each neuron is equipped with a weight $w \in \mathcal{W}$. The distance of the sequence $[a_0, \ldots, a_l]$ from neuron n weighted with w is computed by $\sum_{i=1}^{l} \alpha \cdot \beta^i \cdot \|a_i - w\|$ where $\alpha \in (0, 1)$ and $\beta = 1 - \alpha$.

ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 389-394

The basic idea for a **general framework** for these models consists in the observation that all models share the basic recursive dynamics. They differ in the way how tree structures are internally represented with respect to the recursive processing and the weights of neurons. We need four ingredients for the definition of a general framework covering these approaches:

1. the set of labels of trees $\mathcal{W}$ with similarity measure $d_\mathcal{W} : \mathcal{W} \times \mathcal{W} \to \mathbb{R}$,
2. the set of formal representations of trees $\mathcal{R}$ with a similarity measure $d_\mathcal{R} : \mathcal{R} \times \mathcal{R} \to \mathbb{R}$, r_ξ denotes the representation of the empty tree ξ,
3. the set of N neurons $\mathcal{N}$ with weights given by $\mathcal{L} = (\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2) : \mathcal{N} \to \mathcal{W} \times \mathcal{R} \times \mathcal{R}$,
4. a representation function $rep : \mathbb{R}^N \to \mathcal{R}$ which maps activations of the map to a formal representation of the processed tree.

Then we can recursively define the activation of neuron n given the tree $a(t_1, t_2)$:

$$\tilde{d}(a(t_1,t_2),n) = \alpha \cdot d_\mathcal{W}(a, \mathcal{L}_0(n)) + \beta \cdot d_\mathcal{R}(R_1, \mathcal{L}_1(n)) + \beta \cdot d_\mathcal{R}(R_2, \mathcal{L}_2(n))$$

where

$$R_j = \begin{cases} r_\xi & \text{if } t_j = \xi \\ rep(\tilde{d}(t_j, n_1), \ldots, \tilde{d}(t_j, n_N)) & \text{otherwise} \end{cases}$$

for $j = 1, 2$ and $\alpha, \beta > 0$ are weighting factors. The choice of α and β determines the importance of the root label in comparison to the subtrees.

Note that the set $\mathcal{R}$ enables a precise notation of how trees are internally stored in the map. The function *rep* constitutes the interface which maps activity profiles to internal representations of trees. The above approaches are contained in the general framework through appropriate choices of the internal representation of trees. The dynamics for sequences can be found if the term $d_\mathcal{R}(R_2, \mathcal{L}_2(n))$ is dropped. $\mathcal{W}$ constitutes the set of labels, commonly $\mathbb{R}^n$ with the Euclidian metric. SOMSD chooses $\mathcal{R}$ as vector space containing the indices of neurons in the lattice, e.g. $\mathbb{R}^2$ with the standard Euclidian metric for a two-dimensional lattice. $rep(x_1, \ldots, x_N)$ outputs the index corresponding to the minimum x_i. $\mathcal{L}$ emerges through training. RSOM uses the set of activations, i.e. $\mathbb{R}^N$, as formal representations $\mathcal{R}$, together with the Euclidian metric. $rep(x_1, \ldots, x_N) = (\exp(-x_1), \ldots, \exp(-x_N))$ computes an exponentially scaled version for the sake of robustness. $\mathcal{L}$ emerges through training again. TKM chooses $\mathcal{R}$ as the activation of all neurons, i.e. $\mathbb{R}^N$, together with the standard dot product. *rep* is the identity. Here only $\mathcal{L}_0$ is found by training whereas the formal representation attached to neuron n_i coincides with the ith unit vector. Hence the part $d_\mathcal{R}(R_1, \mathcal{L}_1(n))$ of the recursive computation coincides with the activation of neuron n in the previous recursive step. Standard SOM chooses $\mathcal{R}$ as the empty set, therefore no internal representation of trees and their respective distances is available in this map.

3. Hebbian learning

For simplicity, we assume that $\mathcal{W}$ and $\mathcal{R}$ are real-vector spaces. Assume that a set of trees is given for training. We are here only interested in learning the weights of the neurons, i.e. adaptation of $\mathcal{L}$. For learning, a neighborhood structure or lattice of neurons might be important in order to obtain a similarity preserving map. We assume that $nh : \mathcal{N} \times \mathcal{N} \to \mathbb{N}$ measures the degree up to which two neurons are neighbored. Hebbian learning has been used in all of the above models. It refers to the paradigm where the weights of the winner, i.e. the neuron with smallest distance from the given data point, is iteratively updated into the direction of the data point:

```
initialize the weights at random
repeat: choose some training pattern $t = a(t_1, t_2)$;
        compute $\tilde{d}(a(t_1, t_2), n_i)$ for all neurons $n_i$;
        compute the winner $n_{i_0}$;
        adapt the weights of all $n_i$ simultaneously:
            $\mathcal{L}_0(n_i) := \mathcal{L}_0(n_i) + \eta(nh(n_i, n_{i_0}))(a - \mathcal{L}_0(n_i))$;
            $\mathcal{L}_1(n_i) := \mathcal{L}_1(n_i) + \eta(nh(n_i, n_{i_0}))(R_1 - \mathcal{L}_1(n_i))$;
            $\mathcal{L}_2(n_i) := \mathcal{L}_2(n_i) + \eta(nh(n_i, n_{i_0}))(R_2 - \mathcal{L}_2(n_i))$;
```

where R_j $(j = 1, 2)$ constitutes the formal representation for t_j, i.e. r_ξ for $t_j = \xi$ or $rep(\tilde{d}(t_j, n_1), \ldots, \tilde{d}(t_j, n_N))$, respectively. $\eta : \mathbb{R} \to \mathbb{R}$ is a monotonically decreasing function which makes sure that the neighborhood of the winner is updated according to the distance from the winner. Since a proper representation of a tree can only be expected if all its subtrees are faithfully represented, Hebbian learning assumes that all subtrees of a tree are contained in the training set as well. In this case, a simultaneous adaptation of the weights with respect to a tree and all its subtrees is more efficient. Hebbian learning showed very promising results for SOMSD, RSOM, and TKM in [2, 10, 11].

4. Learning based on an energy function

Hebbian learning for standard SOM, i.e. in the case of simple vectors, can be derived alternatively as a stochastic gradient descent on an appropriate energy function. Vector quantization, i.e. only the winner is updated in each step, minimizes the quantization error [8]. An energy function for a variation of SOM has been proposed in [5]. This alternative formulation has the advantage that a clear objective of the dynamics is given and convergence is guaranteed for appropriate choices of the learning rates. Hence the question arises whether an energy function can be found for structure processing unsupervised networks, too. Commonly, the respective energy functions for variations of standard SOM decompose into a sum of terms which depend on the distances of a training pattern from the neurons. Adapted to structure processing networks, an energy function has the general form $E = \sum_t f(\tilde{d}(t, \mathcal{N}))$ where f depends on the choice of neighborhood cooperation. Here $\tilde{d}(t, \mathcal{N})$ is a shorthand notation for the vector of activations $(\tilde{d}(t, n_1), \ldots, \tilde{d}(t, n_N))$ of neurons given a tree t. If all involved functions are differentiable, we can easily compute the derivative with respect to some weight $\mathcal{L}_I(n_J)$, $I \in \{0, 1, 2\}$, of a neuron n_J:

$$\frac{\partial E}{\partial \mathcal{L}_I(n_J)} = \sum_t \sum_j \frac{\partial f(\tilde{d}(t, \mathcal{N}))}{\partial \tilde{d}(t, n_j)} \cdot \frac{\partial \tilde{d}(t, n_j)}{\partial \mathcal{L}_I(n_J)}$$

where the derivative of $\tilde{d}$ can recursively be computed starting from the leaves:

$$\partial\tilde{d}(a(t_1,t_2),n_j)/\partial\mathcal{L}_0(n_J) = \alpha\,\delta_j^J\partial d_{\mathcal{W}}(a,\mathcal{L}_0(n_j))/\partial\mathcal{L}_0(n_j)$$
$$+\,\beta\,\partial_1 d_{\mathcal{R}}(R_1,\mathcal{L}_1(n_j))\cdot\partial R_{1,\mathcal{L}_0(n_J)} + \beta\,\partial_1 d_{\mathcal{R}}(R_2,\mathcal{L}_2(n_j))\cdot\partial R_{2,\mathcal{L}_0(n_J)}$$

for $I=0$ and similar formulas if $I=1,2$. $\delta_j^J\in\{0,1\}$ is 1 iff $j=J$; $\partial_1 d_{\mathcal{R}}(r_1,r_2)$ denotes the vector of derivatives of $d_{\mathcal{R}}$ with respect to the first formal representation r_1; '$\cdot$' denotes the standard dot product; $\partial R_{k,\mathcal{L}_0(n_J)}$ for $k=1,2$ denotes the derivative of the formal representation of t_k, i.e. 0, if $t_k=\xi$, and $\sum_l \partial rep(\tilde{d}(t_k,\mathcal{N}))/\partial\tilde{d}(t_k,n_l)\cdot\partial\tilde{d}(t_k,n_l)/\partial\mathcal{L}_0(n_j)$ otherwise. Note that the first summand corresponds to the standard Hebb term whereas the remaining terms do not occur in Hebbian learning. Hebbian learning disregards the error over the subtrees. Unfortunately, the above updates have the complexity of N^2 in comparison to N for simple Hebbian learning. Since the additional terms in the above formulas are small due to the weighting β which is commonly smaller than 1, Hebbian learning can be regarded as an efficient approximation of the above stochastic gradient descent. Alternatively, a more efficient way of computing the gradients can be found if $\mathcal{R}$ is low dimensional. Then the derivatives can be computed similarly to backpropagation through time: first the gradients of the error function with respect to the internal representations are computed in linear time, using backpropagation from the root to the leaves. Afterwards, the derivatives with respect to weights can easily be derived from these intermediate values. We refer to [7] for formulas.

Various different energy functions, i.e. different choices of f, are of interest concerning unsupervised learning. Adapted to structures they look as follows: Simple vector quantization minimizes $\sum_t\sum_{n_j}\chi_{t,j}\tilde{d}(t,n_j)^2/2$ where $\chi_{t,j}\in\{0,1\}$ yields 1 iff n_j is the winner for t. SOM itself does not possess an energy function. The following objective has been proposed in [5]: $\sum_t\sum_j\chi_{t,j}\sum_k\eta(nh(j,k))\tilde{d}(t,n_k)^2/2$ where η is as above, nh refers to the neighborhood of a given lattice of neurons, and the winner in $\chi_{t,j}$ is defined in a slightly different way than in the standard SOM as the neuron with smallest averaged distance. Finally, the neural gas algorithm [9] constitutes a proposal where no neighborhood function has to be given a priori, nevertheless a neighborhood structure which is defined through the distance from the training patterns is taken into account. The corresponding energy function for structure processing networks is $\sum_t\sum_j\eta(rk(t,j))\tilde{d}(t,n_j)^2/2$ where $rk(t,j)$ denotes the number of neurons which are closer to t than n_j. All these energy functions can be used for unsupervised training of structure processing networks via a stochastic gradient descent using the above formulas for the derivatives.

This proposal yields immediate alternative training algorithms with clear objective for RSOM. Adaptations are to be performed for SOMSD since the involved function *rep* is not differentiable and the choice of the similarity $d_{\mathcal{R}}$ as distance in a lattice is not appropriate for VQ or NG, where no prior lattice is available. We can simply approximate the function *rep* using the soft-min function, for example, up to any desired degree. Furthermore, we can find alternative formal representations for VQ and NG. Assume N neurons are given. We define $\mathcal{R}=\mathbb{R}^N$. For VQ, we can choose $rep(x_1,\ldots,x_N)=e_i$ such that x_i is minimum, where e_i denotes the i th unit vector. For NG we can define

$rep(x_1,\ldots,x_N) = (rk(x_1),\ldots,rk(x_N))$ where $rk(x_i)$ denotes the number of x_j which are smaller than x_i. These alternative representations or differentiable approximations thereof take the decoupling of indices for VQ or data driven lattices for NG, respectively, into account.

5. Conclusions

We have proposed a general framework for unsupervised structure processing networks which covers various promising approaches from the literature. We have formulated simple Hebbian learning for this framework and proposed a general way of transferring algorithms based on an energy function. This yields new learning algorithms with a clear objective of the dynamics. Moreover, it makes the transfer of various approaches from the field of unsupervised networks such as the neural gas algorithm possible. A key ingredient of the general formulation is the notion of a formal representation of trees. This clear separation proposes the theoretical investigation of desirable properties of $\mathcal{R}$ and *rep* for adequate structure processing. Obvious demands are for example: *rep* should yield a compact image and *rep* should be noise tolerant. Other demands whose formalization is less obvious are: *rep* should preserve similarities of processed trees, *rep* should preserve information of the winner, *rep* should be a global mapping. First steps of a mathematical formalization of these demands have been performed in [7]: as an example, it is possible to formalize the degree of locality of the processing. One can show that a certain degree of globality of *rep* is necessary for proper processing independently of the choice of the other ingredients of this framework. Note that a precise abstract formalization of the framework is essential for these investigations and a deeper understanding of concrete realizations like SOMSD or RSOM.

References

[1] A.M.Bianucci, A.Micheli, A.Sperduti, and A.Starita, Application of cascade correlation networks for structures to chemistry, Journal of Applied Intelligence 12:117-146, 2000.

[2] G.Chappell and J.Taylor, The temporal Kohonen map, Neural Networks 6:441-445, 1993.

[3] P.Frasconi, M. Gori, A.Küchler, and A.Sperduti, A field guide to dynamical recurrent networks, in: J.F.Kolen, S.C.Kremer(eds.), From Sequences to Data Structures: Theory and Applications, pp.351-374, IEEE, 2001.

[4] P.Frasconi, M.Gori, and A.Sperduti, A general framework for processing of data structures, IEEE Transactions on Neural Networks 9(5):768-786, 1998.

[5] T.Heskes, Self-organizing maps, vector quantization, and mixture modeling, to appear in IEEE Transactions on Neural Networks.

[6] B.Hammer, Learning with Recurrent Neural Networks, LNCIS 254, Springer, 2000.

[7] B.Hammer, A.Micheli, and A.Sperduti, A general framework for self-organizing structure processing neural networks, manuscript available from the authors.

[8] T.Kohonen, Self-organizing maps, Springer, 1997.

[9] T.Martinetz and K.Schulten, Topology representing networks, Neural Networks 7(3):507-522, 1993.

[10] A.Sperduti, Neural networks for adaptive processing of structured data, in: G.Dorffner, H.Bischof, K.Hornik (eds.), ICANN'2001, pp. 5-12, Springer, 2001.

[11] T.Voegtlin, Context quantization and contextual self-organizing maps, in: Proc. Int. Joint Conf. on Neural Networks, vol.5, pp.20-25, 2000.