On-line Learning with Dynamic Cell Structures *

Ingo Ahrns, Jörg Bruske, Gerald Sommer
Christian Albrechts University of Kiel - Cognitive Systems Group
Preusserstr. 1-9 - 24118 Kiel - Germany
Phone: ++49 431 560480
Fax: ++49 431 560481
e-mail: jbr@informatik.uni-kiel.de

Abstract

Dynamic Cell Structures (DCS) belong to the class of *topology representing networks* recently introduced by [Ma94]. DCS networks rest upon radial basis function networks (RBF) and an additional lateral connection structure between the neural units. This connection structure reflects the topology of the input manifold.

Initial simulations on CMU benchmarks presented in [Br95] are very encouraging but were all involved with off-line learning tasks. In this article we argue that DCS networks are also capable of performing on-line learning tasks where the training examples are generated by a dynamic system (i.e. not a fixed training set). The proposed modifications to the original learning scheme in order to cope with the dynamics bear relevance not only for DCS but for Kohonen type networks and TRN in general. Our test environment is pole-balancing.

1 Introduction

Like Kohonen's famous SOM Dynamic Cell Structures (DCS) are capable of adaptive vector quantization, clustering, dimensionality reduction and have been extended for supervised learning. Moreover, DCS-networks form "topology preserving" feature maps [Ma94]. In this section we briefly summarize the foundations of DCS. Section 2 presents our extensions for better coping with dynamic systems and in section 3 we report simulation results.

Consider an input manifold $I \subset \mathbb{R}^n$ and an output manifold $O \subset \mathbb{R}^m$.

A *neural DCS-unit* is a point $u = (c_u, w_u, \alpha_u, r_u) \in I \times O \times [0,1]^I \times \mathbb{R}_{\geq 0}$, where c_u is called the *center*, w_u the *weight*, α_u the *activation function* and r_u the *resource* of the neural unit.

A *DCS-graph* is an undirected graph $N = (U, L)$ where $U \subset I \times O \times [0,1]^I \times \mathbb{R}_{\geq 0}$ is a finite set of neural units and $L \subset \{\{u, v\} | u, v \in U \wedge u \neq v\}$ is a set of *lateral connections* between different neural units.

*Published in: Proc. of ICANN'95, Vol.2 (1995) 141-146

The *lateral connection-strength* is a function $\rho : L \rightarrow \mathbb{R}_{\geq 0}$ which is usually given by a symmetrical adjacency matrix $C \in \mathbb{R}^{|U|\times|U|}$ for which $C_{ii} = 0$ for all $i \in \{1, \cdots, |U|\}$. Moreover, $C_{ij} = 0$ if unit i and unit j are not connected, and $0 < C_{ij} \leq 1$ if unit i and unit j are connected with strength C_{ij}. In general we have $\rho(\{u_i, u_j\}) = C_{ij}$.

A *DCS-network* is a triple (U, L, ρ) of a DCS-graph (U, L) and a lateral connection-strength ρ. In order to perform perfect topology learning, a simple competitive Hebbian learning rule is used ([Br95]). It was proven in [Ma94] that a lateral connection structure of neural units formed by the competitive Hebbian learning rule yields a perfect topology preserving map. In [Br95] and [Ma92] this learning rule was extended by introducing a forgetting constant in case of a non-stationary input distribution or changing neighborhood relations.

GDCS is a growing DCS network that employs ideas from Fritzke's Growing Cell Structures (GCS) [Fr95]. Like in GCS each neuron contains a local error variable which is called the resource of a neural unit. Such a resource variable accumulates error values if its neuron wins the competition for a given input stimulus and becomes the best matching unit. In case of supervised learning one can e.g. use the squared output error as an error measure. The reason for introducing resources is to guide the insertion of new neural units by the local error in order to get a uniform distribution of all resource values.

In GCS and GDCS centers are adapted in a Kohonen-like fashion, such that the centers of the best matching unit (bmu) and its neighbors are moved into the direction of the last input. In this article we argue that an improved adaptation rule can be obtained by taking into account the error variables – much like the classification performance is taken into account in LVQ-algorithms [Ko92].

2 DCS for dynamic system control

The GDCS-networks mentioned above has demonstrated effectiveness in supervised and unsupervised learning tasks [Br95]. Training sets were static and have been mixed to remove the influence of the order of sample presentation. This kind of neural net training is legitimate if all training data are directly attainable. The focus on this article is on demonstrating how GDCS are able to learn control functions on-line.

2.1 The Task of Pole-Balancing

Pole-balancing is a famous example of a dynamic system which has often been reported to be a difficult control problem. In this article we consider the pole-balancing problem in its simplest version adopted from Ritter [Ri91]. System dynamics are then comprised by the following differential equation:

$$(m + \sin^2\theta)\ddot{\theta} + \frac{1}{2}\dot{\theta}^2 \sin(2\theta) - (m+1)\sin\theta = -f\cos\theta. \qquad (1)$$

Here, θ represents the angle of inclination of the pole, m is the mass of the base point of the pole, and the force applied to the base point is denoted as f. As an appropriate function for balancing the inverse pendulum, the force $f^L(\theta, \dot{\theta}) = 5\sin\theta + \dot{\theta}$ was taken

as a teacher signal for supervised learning. In our simulations we released the pole at a randomly chosen angle $-40° \leq \theta \leq 40°$ with zero velocity. The force for controlling the pole can be obtained by

$$f = \beta(t_k)y(\theta, \dot{\theta}) + (1 - \beta(t_k))f^L(\theta, \dot{\theta}), \tag{2}$$

where $\beta : [0, T] \rightarrow [0, 1]$ represents a monotonically increasing function which expresses the reliability of the control-function y of the network.

2.2 Improvements on DCS

Let us now discuss some modifications of DCS with respect to on-line learning in control problems.

2.2.1 Mixed insertion strategy

In the classical GDCS approach, a new neural unit is inserted after λ teaching steps between the units with the largest resource values. We call this method *error driven unit insertion*. It is useful, if one has a fixed training set T. In our control problem we don't have any fixed training sets. Rather training data are obtained according to the dynamics of the controlled system which can cause severe problems with unfolding in GCS as well as GDCS. The solution is to augment GDCS with a second insertion strategy operating in parallel to the error driven insertion strategy: We introduce a threshold $\delta \in \mathbb{R}_{\geq 0}$ which indicates the maximum distance of an input stimulus to the center of the best matching unit. If the distance exceeds δ and the number of neural units is below a given maximum unit number, we insert a new neural unit at the position of the given training pair in $I \times O$. That new unit can be connected to the last best matching unit. Such an additional *distance driven unit insertion* is quite common, see e.g. [Ha93],[Pl91], where insertion takes place if the stimulus distance and the approximation error both exceed a threshold. Besides coping with unfolding problems, the distance driven insertion yields a coarse scale approximation which will be refined by the error driven insertion strategy.

2.2.2 Non-Symmetric Lateral Connection Structure

As pointed out in [Ma92], it is possible to update the lateral connections of the best matching unit only. If we use a competitive Hebb rule as proposed in [Br95] for topology learning, including the deletion of lateral connections in the case of $C_{ij} < \theta$, unconnected neural units can result. Unconnected neural units become superfluous if they lie in regions of the input manifold where input stimuli have low probability to stem from. Such neural units are called *dead neurons*. In order to avoid such neurons we now give up the demand of symmetry and adapt the lateral connections of the bmu non-symmetrically. Consequently, lateral connections will only be removed, if the connection strengths of both connected

units fall below θ. We obtain:

$$C_{ij} := \begin{cases} 1 & : \{i,j\} = \{bmu, second\} \\ 0 & : i = bmu \wedge j \neq second \wedge (\alpha \cdot C_{ij} < \theta \wedge C_{ji} < \theta) \\ 0 & : j = bmu \wedge i \neq second \wedge (\alpha \cdot C_{ji} < \theta \wedge C_{ij} < \theta) \\ \alpha \cdot C_{ij} & : i = bmu \wedge j \neq second \wedge (\alpha \cdot C_{ij} \geq \theta \vee C_{ji} \geq \theta) \\ C_{ij} & : otherwise \end{cases}, \tag{3}$$

where the best matching unit and the second best matching unit for a given input stimulus $x \in I$ are defined by

$$\forall u \in U : \|c_{bmu} - x\| \leq \|c_u - x\|, \forall u \in U \setminus \{bmu\} : \|c_{second} - x\| \leq \|c_u - x\| \tag{4}$$

2.2.3 Error Modulated Kohonen Rule for Center Adaptation

The next modification of GDCS concerns the adaptation rule for center updating. If training data are generated by a dymamical system like the inverse pendulum tendencies will result which usually infers with traditional training methods. Another well known problem with Kohonen-type feature maps is to avoid that regions where inputs have a high temporary probability density attract all neural units. Instead of reflecting the input probability density the distribution of neural units should result in a uniform distribution of local approximation errors. This is the motivation for our *error modulated Kohonen-rule*: Let the resource values accumulate a kind of *mean value* of error signals (independent of frequency). We employ:

$$r_{bmu} := \gamma \cdot r_{bmu} + (1 - \gamma) \cdot err, \quad \gamma \in [0,1]. \tag{5}$$

Now let $c : I\!R_{\geq 0} \rightarrow [0,1]$ be a monotonically decreasing function with $c(0) = 1$ and $c(1) = 0$. Our modified update rule is

$$c_{bmu} := c_{bmu} + c\left(\frac{\bar{r}}{r_{bmu}}\right) \cdot \varepsilon_{bmu} \cdot (x - c_{bmu}), \tag{6}$$

$$\forall u \in Nh(bmu) : c_u := c_u + c\left(\frac{r_u}{r_{bmu}}\right) \cdot \varepsilon_{Nh} \cdot (x - c_u), \tag{7}$$

$$where \quad \bar{r} = \frac{1}{|Nh(bmu)|} \cdot \sum_{u \in Nh(bmu)} r_u \tag{8}$$

$$and \quad Nh(bmu) := \{v | \{bmu, v\} \in L\}. \tag{9}$$

Hence the bmu is only attracted by the input stimulus if $r_{bmu} > \bar{r}$ and neighbors are only attracted if their resource is below the bmu's resource, aiming at a uniform distribution of resource values.

3 Simulation results

We applied our improved GDCS-network to the task of pole balancing mentioned above. The GDCS-network was trained for a duration of k_{max} training steps. After a balancing

failure ($|\theta| > 45°$) or after k_{try} successful balancing steps the simulation was started again. For our experiments we used a DCS-network with the mixed unit insertion strategy, the non-symmetric lateral connection structure and the error modulated Kohonen rule for center adaptation. After 3000 learning steps our network, consisting of 70 neural units, was able to balance the pole, see Fig. 1.

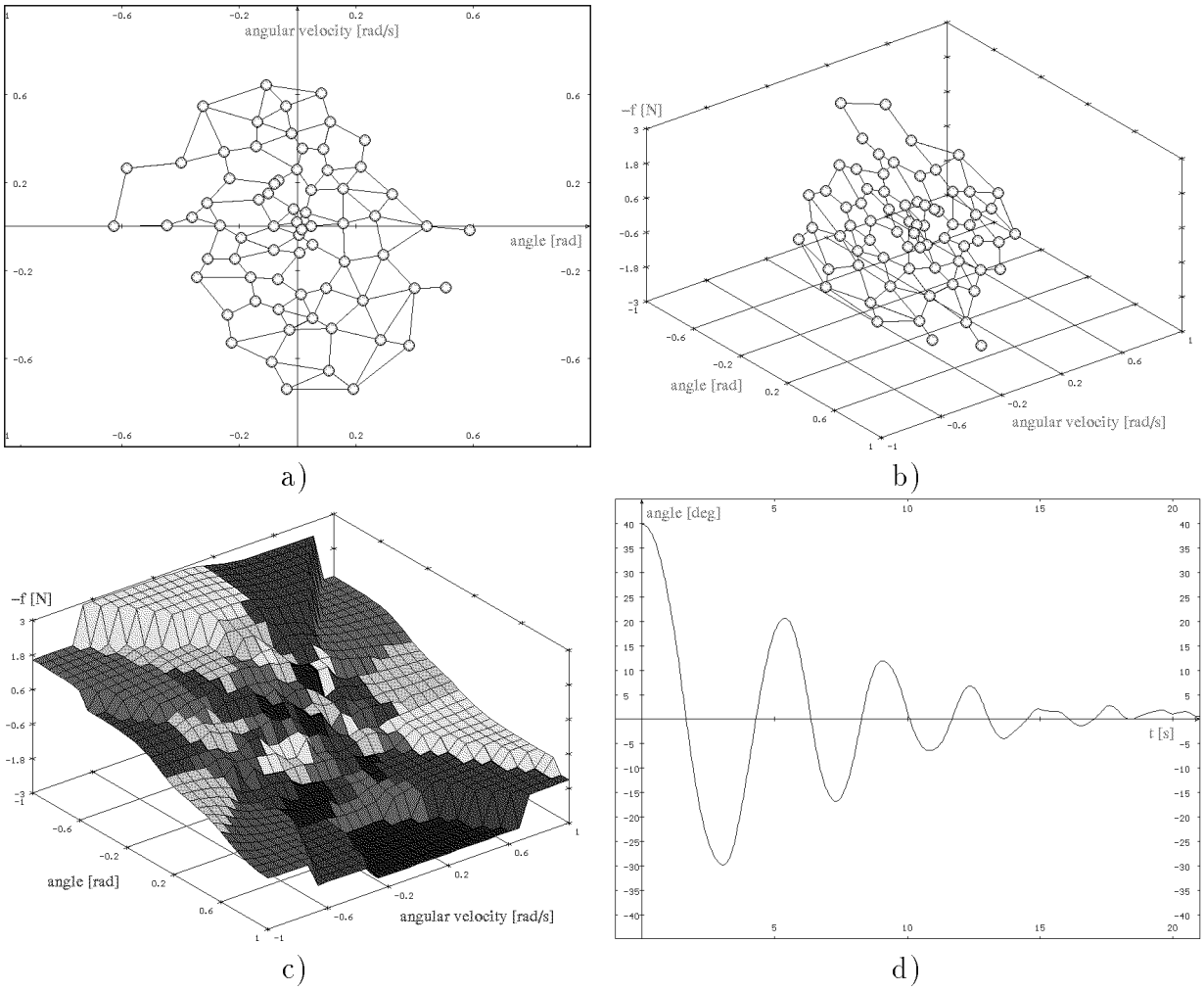

Figure 1: a), b) DCS-network. c) Approximate function $y(\theta, \dot{\theta})$. d) $\theta(t)$ during pole-balancing after training-phase.

The parameter set of our simulation was: $k_{max} = 3000$, $k_{try} = 100$, time step width $\Delta t = 0.3\ s$, pendulum mass $m = 1.0\ kg$, learning constants $\varepsilon_{bmu} = 0.1$, $\varepsilon_{Nh} = 0.001$, connection threshold $\theta = 0.01$, insertion constant $\delta = 0.075$, insertion step width $\lambda = 100$, resource averaging $\gamma = 0.8$, connection forgetting rate $\alpha = 0.88$, reliability function $\beta(t_k) = \frac{k}{k_{max}}$ and $c(x) = 2x^3 - 3x^2 + 1$ if $x \in [0, 1]$ and $c(x) = 0$ otherwise. Neural activations were computed by Gaussian basis-functions with a standard deviation of $\sigma = 0.2$. Output weights were adapted by using a simple delta-rule with a learning constant of $\eta = 0.001$.

4 Conclusion

On-line learning with respect to control of dynamic systems is difficult due to the non stationary probability density of input stimuli and resulting tendencies in the training data. In this article we proposed three modifications to the original GDCS learning scheme to cope with this problem without "freezing" of learning constants. First, our *error modulated Kohonen type learning rule* is no longer dominated by the input probability density but aims at a uniform approximation error (uniform distribution of resources) independent of the probability density. This idea applies to GCS and extensions of the Kohonen map as well. Second it was found that tendencies in the data can cause severe unfolding problems in GCS or GDCS if error driven insertion is employed only. We therefore decided to augment GDCS with a *distance driven insertion scheme* for approximation on a coarse scale. Error driven insertion remains responsible for error driven refinement. The idea applies to GCS as well. Finally, TRN may suffer from dead neurons if lateral connections are allowed to be forgotten. Our solution to this problem is to introduce *unsymmetric lateral connections*. Pole balancing in its simplest version was used for demonstration. Despite of its simplicity this simulation nevertheless had to cope with all the afore mentioned difficulties in on-line learning of dynamic systems.

Currently we work on using GDCS for obstacle avoidance on a TRC Labmate mobile robot.

References

[Br95] Jörg Bruske, Gerald Sommer: Dynamic Cell Structure learns Perfectly Topology Preserving Map, Neural Computation, Vol. 7, No. 4, 1995.

[Ma92] Thomas Martinez: Selbstorganisierende neuronale Netzwerke zur Bewegungssteuerung, Dissertation, DIFKI-Verlag, 1992.

[Ma94] Thomas Martinez: Topology Representing Networks, Neural Networks, Vol.7, pp. 505-522, 1994.

[Ri91] H. Ritter, T. Martinetz, K. Schulten: Neuronale Netze, Addison Wesley, 1991.

[Fr95] B. Fritzke: Growing Cell Structures - a self organizing network for unsupervised and supervised learning, Neural Networks, Vol.7, pp.1441-1460, 1995.

[Ha93] J. Hakala and R. Eckmiller: Node Allocation and Topographical Encoding (NATE) for Inverse Kinematics of a Redundant Robot Arm for Inverse Kinematics of a 6-DOF Robot Arm, ICANN 93, pp. 309-312, 1993.

[Ko88] Teuvo Kohonen: Adaptive, associative, and self-organizing functions in neural computing, Applied Optics, Vol. 26, pp.4910-4918, 1987.

[Ko92] T. Kohonen et. al.: LVQ-PAK, The Learning Vector Quantization Program Package, Version 2.0, Helsinki University of Technology, 1992.

[Pl91] J. Platt: A Resource-Allocating Network for Function Interpolation, Neural Computation, Vol.2, No. 3, pp.213-225, 1991.