JG

UNIVERSITÄT LEIPZIG

Naturwissenschaftlich-Theoretisches Zentrum

Explorations in Q-learning
PART I
Improving speed, stability and
accuracy of Kohonen-based Q-learning

by

R. Der[1], M. Herrmann[2]

[1]Universität Leipzig
Fakultät für Mathematik und Informatik
Mathematisches Institut
Augustusplatz 10
04109 Leipzig
Germany

[2]NORDITA
Blegdamsvej 17
DK-2100 Copenhagen
Denmark

Preprint-Nr. 24/94

Abstract

The present paper is the first in a series of studies in Q-learning. We start by investigating the interaction of a look-up table implementation of Q-learning with the underlying adaptive discretization of the state space by a Kohonen feature map. Here, an interpolation scheme for the Q-function compensates for the drift of the state categories. On the other hand the resolution of the state space partition is guided both by an attentional mechanism for levelling the heterogeneous distribution of state vectors as well as the sequence of actions generated by the Q-learner. A combined algorithm based on these principles is applied to the standard pole balancing task resulting in a clear improvement of convergence time and stability. Further articles are devoted to the theoretical analysis of the Q-learning algorithm and to applications to chaos control.

1 Introduction

In recent years $\mathcal{Q}$-learning [1] has become one of the major implementations of the reinforcement learning paradigm. By the work of many investigators, experience has been gathered from studying toy models of increasing complexity so that by now the application to real world problems is at stake. From the point of view of the present paper there are essentially two different ways of implementing $\mathcal{Q}$-learning, either by means of look-up tables or by trying an ansatz for the $\mathcal{Q}$-function, e.g. using multi-layered neural nets. For continuous state spaces the look-up table is based on a discretization of the states of the system which is conveniently done by using a self-organizing feature map (SOFM). Our paper is concerned with this kind of approach and tries to reveal and to resolve several of the difficulties and pitfalls of this approach. Moreover, in an accompanying paper we try to contribute to an improved understanding of the properties of the $\mathcal{Q}$-learning algorithm by studying a simple model system which can be treated exactly. Explicit results for the convergence properties and the asymptotic behavior of the $\mathcal{Q}$-function obtained for the toy model provide (cf. discussion) cues for understanding the properties of more realistic situations.

The kind of problems with the SOFM-based $\mathcal{Q}$-learning we want to address firstly in this paper typically arise from the interference of the learning of the SOFM (Kohonen [2] or $\mathcal{K}$-learning) with that of the $\mathcal{Q}$-function. In fact, during $\mathcal{Q}$-learning the policy of the controller and hence the trajectories change which in turn influences the density of data points (states of the system) the SOFM receives as inputs. Since the Kohonen map adapts to this time dependent density it is dragged along by the changing policy. However, $\mathcal{Q}$-learning takes the nodes of the map as reference points, moving around these nodes changes the $\mathcal{Q}$-function as a function defined with respect to *physical* space. As a consequence we not only loose information on the best policy gathered so far but also introduce a further shift in the trajectories controlled by the $\mathcal{Q}$-function which again introduces a new drift into the Kohonen map. This feed back mechanism may lead to instabilities or to a decrease of learning speed and reliability of the learning procedure.

Clearly, the problem of instability can be avoided by choosing the time scales for the $\mathcal{K}$-learning much longer than that for the $\mathcal{Q}$-learning so that the $\mathcal{Q}$-learning can compensate for the drift of the nodes. This however is very time consuming and will break down in a dynamic environment where the map must be able to trace the changes imposed by the environment. We propose a more effective way of dealing with the problem of the interference of $\mathcal{K}$- and $\mathcal{Q}$-learning which relies on an interpolation scheme for the $\mathcal{Q}$-function based on the topology preservation in the SOFM, cf. Section 4.

Another set of problems arises from the fact that the Kohonen algorithm adapts the resolution of the state space according to the density of *data* points and not according to the needs of getting an optimal representation for the $\mathcal{Q}$-function which coarsely speaking means choosing the density of nodes proportional to the roughness of the $\mathcal{Q}$-function. A solution to this problem will be proposed in Sec. 3. Our proposals will be tested for the generic example of the pole balancer in Sec. 5 where the expected improvement over the usual implementation is all too clear.

In the second part of our paper we try to extract some more fundamental results from a simple model which serves as a crude abstraction of the pole balancer. It turns

out, however, that the influence the choice of parameters has on the behavior of the algorithm is similar in both cases, cf. discussion.

2 $\mathcal{Q}$-Learning and $\mathcal{K}$-learning

2.1 $\mathcal{Q}$-learning

We consider a general control task of the following type. Suppose, a system may be characterized by the state vector $x \in \mathbf{X}$. Using an appropriate discretization of the generally continuous $\mathbf{X}$ the states can be represented by a finite number of categories $j(t)$. x is to be controlled towards and stabilized close to a target state x^* using a sequence of control actions $\{a(t)\}$, $t = 0, 1, \ldots$. The set of possible actions is assumed to be finite. The $\mathcal{Q}$-learning algorithms determines optimal action sequences $\{a(t)\}$ in correspondence to trajectories $\{j(t)\}$ based on reinforcement signals $r(t)$. Given $j(t)$ an action $a(t)$ to be applied is chosen according to

$$a(t) = \text{ argmax } \mathcal{Q}(j(t), a(t)) \tag{1}$$

$\mathcal{Q}(j(t), a(t))$ which measures the value of a state-action pair is adjusted according to

$$\Delta Q\,(j(t), a(t)) = \epsilon\,(r(t+1) + \gamma V(j(t+1)) - Q(j(t), a(t))) \tag{2}$$

where

$$V(j(t+1)) = \max_a Q(j(t+1), a) \tag{3}$$

The reinforcement signal $r(t)$ assumes an elevated value if the state has reached the target. It is possible to reward states that came close to the target or to use negative reinforcement if the system has reached a 'failure' state. However, as shown in the companion paper $\mathcal{Q}$-learning may fail in the case of negative reinforcement.

2.2 $\mathcal{K}$-learning

Both the control algorithm and the representation of the environment in the system should be adaptive by any of the following reasons: (α) The environment changes in time. (β) No sufficient prior knowledge is available. (γ) The representation of the environment in the system depends intrinsically on the behavior of the system itself.

This concerns in particular the discretization or *partition* of the state space. In the above situation it appears to be natural to base the $\mathcal{Q}$-learning algorithm on a self-organizing feature map. We were using the Kohonen or $\mathcal{K}$-learning algorithm for this purpose. The basic update rule for category vectors w_j representing a prototype state of category j reads

$$\Delta w_j = -\epsilon h_{js}(w_j - x). \tag{4}$$

Topology preservation between category space and state space is enforced by the Gaussian neighborhood function

$$h_{js} = \exp(-\frac{||j-s||^2}{2\sigma^2}). \tag{5}$$

which is centered at the best matching unit s

$$x \to s \quad \text{if} \quad ||w_s - x|| = \min_j ||w_j - x||. \tag{6}$$

Besides the topology of the categories the distribution of the w_j is guided by the frequency of input vectors x. In the present situation this causes the problem that if the controller has met the task and the state of system tends to be close to the target most of the time the rule (4) will contract all prototypes finally to this small region abandoning in this way the information on how to guide the system from a remote state to the target. For reasons mentioned above switching off the partitioner adaptivity is generally not desirable.

Further, the distribution of prototype vectors is not automatically the most optimal one with respect to the performance of the controller. These two issues are addressed by a modified version of the Kohonen algorithm.

3 State-space partitioner

3.1 Adjusting the frequency dependence of $\mathcal{K}$-learning

When rewriting Eq. (4) in terms of averages with respect to the input distribution $P(x)$. Note that s is dependent on x.

$$\overline{\Delta w_j} = -\int dx P(x) \epsilon h_{js}(w_j - x) \tag{7}$$

it obvious that by the choice

$$\epsilon \sim P(x)^{-1} \tag{8}$$

the algorithm would behave as if $P(x)$ were constant. In Eq. (7) the integral over the state space may splitted into regions in which s is constant, hence the learning rate according to (8) will be dependent only on the winning unit s. A feasible way of applying (8) reveals when expressing $P(x)$ in terms of the probability $P(s)$ of neuron s being the best-matching unit and the density $P(w)$ of prototype vectors $w_j \in \mathbf{X}$, i.e. by two quantities which are readily available from the feature map.

$$P(x) \sim P(w_s)P(s), \tag{9}$$

Eqs. (8) and (9) allow to determine a neuron-dependent learning rate which guarantees an even distribution of prototype vectors in those regions of the state space where $P(x)$ is greater than a certain minimal value. Formulated the other way round, as long as the neuron dependent learning rate ϵ_s does not exceed a certain maximal value the prototype distribution is constant.

A few remarks are in order. The maximal value of ϵ_s has to be chosen such that correlations in the input data which are caused by the fact that the system moves along continuous trajectories are averaged out. In regions of low $P(x)$ the system adapts at least with the maximum learning rate. Instead of '-1' other exponents can be used in Eq. (8) allowing e.g. for more faithful representations of the input distribution than the original Kohonen algorithm. Even 'inverse' prototype distributions where sparse regions of the input space are densely inhabited and vice versa are possible [3].

In practise, Eq. (9) would require to monitor sliding averages for both $P(w)$ and $P(s)$. More simply, it turned out to be sufficient to approximate $P(s)$ by the inverse lag between the current and the previous time of a unit to be the winner and $P(w)$ by a function of the current distance between x and w_s[1]

3.2 Action-dependent partitioning

A further improvement of the controller's performance is expected if the partition of the state space is taken to be action dependent on top of the modifications in section 3.1. The controller's decision which action to take is in some regions of the state space less critical than in others. Hence, critical regions should be resolved more accurately. In particular, we modify the $\mathcal{K}$-learning rule such that prototype vectors are attracted more strongly if the action proposed by the controller is different in subsequent time steps, whereas the learning rate may be low if one action is repeated. The neuron-dependent learning rate will, hence, include an action-dependent contribution in addition to the frequency-dependent part according to (8).

$$\epsilon_s = \begin{cases} \frac{\epsilon_0(t)}{P(w_s)P(s)}\epsilon_a(t) & \text{if } \frac{\epsilon_0(t)}{P(w_s)P(s)}\epsilon_a < \epsilon_{max} \\ \epsilon_{max} & \text{otherwise} \end{cases} \tag{10}$$

where

$$\epsilon_a(t) = \begin{cases} \epsilon_{a,1}\|a(t-1) - a(t)\| & \text{if } a(t+1) \neq a(t) \\ \epsilon_{a,0} & \text{otherwise} \end{cases} \tag{11}$$

Relation (11) tends to move more category prototypes towards boundaries between regions of different optimal actions while leaving the inside of these regions less densely inhabited.

It should be noted that various effects overlay to the ideal behavior stated so far. Correlations in the input data will corrupt partially the averaging assumed for the derivation of the neuron-dependent ϵ_s, while neighborhood collaboration essential to the $\mathcal{K}$-learning scheme effects the action-dependent modification of ϵ. The computer experiment in section 5 shows, however, a clear improvement when using the modified $\mathcal{K}$-learning rule, such that action dependent partitioning can be considered as definitely useful, whereas the neuron-dependent learning strengths are alienable.

Whereas action-dependent partitioning causes the representation of the state space to be more efficient, the $\mathcal{Q}$-function, in turn, is deteriorated by the drift of the category

[1]The relation between the intracell variance and the volume of a cell is dependent on the dimension of the input space. Assuming isotropy we may use the relation $volume = variance^{\frac{n}{n+2}}$.

vectors. The next section will consider the issue of overcoming such feedback loops by a compensation mechanism.

4 Compensating for drift of categories

The changes in the controller strategy being mediated by the $\mathcal{Q}$-learning process should naturally cause changes in the behavior of the controlled system. The partition of the state space underlying the $\mathcal{Q}$-learning algorithm therefore will become increasingly inappropriate to represent the actual distribution of the input data. An update of the categories of the partition, however, will influence also the representation of the $\mathcal{Q}$-function. $\mathcal{Q}$-values which relate to the position of the category prototypes in the state space generally should be different at the updated category positions. A simple way to resolve this problem is to separate the time scales of the partitioner learning and the $\mathcal{Q}$-learning, i.e. between subsequent partitioner updates must be enough time to readapt the $\mathcal{Q}$-function. This allows for a very slow adaptation of the categories only, in accordance to the time scales involved in most $\mathcal{Q}$-learning problems and, furthermore, results in temporary deteriorations of the controller's performance.

An algorithm like the following interpolation scheme enables the system to perform the partitioner update and the $\mathcal{Q}$-learning simultaneously while not abandoning the information already acquired in the category-based $\mathcal{Q}$-values.

We will introduce the algorithm for a two-dimensional net and an n-dimensional state space. The case of a one-dimensional chain of neurons is similar, but much simpler. The Appendix formulates the algorithm for a net of arbitrary dimension d.

If a category prototype w_j, is updated

$$w_j \longrightarrow w'_j \tag{12}$$

the corresponding $\mathcal{Q}$-value has to be modified according to

$$\mathcal{Q}(w_j, a) \longrightarrow \mathcal{Q}(w'_j, a) \tag{13}$$

rather than leaving $\mathcal{Q}(w_j, a) = \mathcal{Q}(w'_j, a)$ unchanged. In order to determine $\mathcal{Q}(w'_j, a)$ the projection of the updated category vector w'_j onto the net subspace is represented as

$$w'_j = \beta_0 w_0 + \beta_1 w_1 + \beta_2 w_2, \quad \beta_0 + \beta_1 + \beta_2 = 1 \tag{14}$$

where w_0, w_1, w_2 are three suitably chosen prototype vectors from the two-dimensional net prior to the recent update, viz. w_{j_0} is the nearest neighbor of w'_j (the winning unit if w'_j were an stimulus vector to the old net). Apart from edge neurons any w_{j_0} has two pairs of neighbors. Out of each pair $w_{j_0 \pm e_m}$ (e_m, $m = 1, 2$ being the lattice vectors) that vector w_j is chosen which is the closer one w.r.t. w'_j. 'Closeness' is determined by maximizing the scalar product

$$E_m = \frac{\langle w'_j - w_{j_0}, w_{j_0 \pm e_m} - w_{j_0} \rangle}{\|w'_j - w_{j_0}\| \|w_{j_0 \pm e_m} - w_{j_0}\|} \tag{15}$$

$$= \frac{\sum_{k=1}^{n} (w'_{j,k} - w_{j_0,k})(w_{j_0 \pm e_m,k} - w_{j_0,k})}{\sqrt{\sum_{k=1}^{n} w'_{j,k} - w_{j_0,k}} \sqrt{\sum_{k=1}^{n} w_{j_0 \pm e_m,k} - w_{j_0,k}}} \tag{16}$$

In the case of edge neurons for at least one index only either $w_{j_0+e_m}$ or $w_{j_0-e_m}$ exists. If the existing neighbor is opposite (negative scalar product) to w'_j the interpolation rule (17) performs an extrapolation, i.e. some of the β_i are smaller than 0. The two chosen neighbors of w_{j_0} are denoted by w_{j_1} and w_{j_2}.

Supposed the number of main features of the data manifold is matched by the network dimension, a certain degree of topology conservation will have been achieved. Therefore, if j_0, j_1 and j_2 are not placed at a single line w_{j_0}, w_{j_1} and w_{j_2} will be far from being collinear. A linear approximation of the $\mathcal{Q}$-function can be obtained by

$$\mathcal{Q}(w'_j, a) = \beta_0 \mathcal{Q}(w_0, a) + \beta_1 \mathcal{Q}(w_1, a) + \beta_2 \mathcal{Q}(w_2, a) \tag{17}$$

where the coefficients β_i are given by Eq. (14). The determination of values of β_i involves elementary linear algebra, and is given in the Appendix. In the present case we find:

$$\begin{aligned} \beta_0 &= 1 - \beta_1 - \beta_2 \\ \beta_1 &= \frac{\langle v', v_1\rangle\langle v_2, v_2\rangle - \langle v', v_2\rangle\langle v_2, v_1\rangle}{\langle v_1, v_1\rangle\langle v_2, v_2\rangle - \langle v_1, v_2\rangle\langle v_2, v_1\rangle} \\ \beta_2 &= \frac{\langle v_1, v_1\rangle\langle v', v_2\rangle - \langle v_1, v_2\rangle\langle v', v_1\rangle}{\langle v_1, v_1\rangle\langle v_2, v_2\rangle - \langle v_1, v_2\rangle\langle v_2, v_1\rangle} \end{aligned} \tag{18}$$

where $v' = w'_j - w_{j_0}$, $v_1 = w_{j_1} - w_{j_0}$, $v_2 = w_{j_2} - w_{j_0}$, and $\langle\ ,\ \rangle$ denotes the scalar product (cf. (15)). Obviously, if $w'_j \equiv w_{j_i}$ for one i, i.e. $v' \equiv v_i$, then $\beta_i = 1$ and $\beta_j = 0$ for $j \neq i$.

In the one-dimensional case (18) simplifies to

$$\begin{aligned} \beta_0 &= 1 - \beta_1 \\ \beta_1 &= \frac{\langle w'_j - w_{j_0}, w_{j_1} - w_{j_0}\rangle}{\|w_{j_1} - w_{j_0}\|^2} \end{aligned} \tag{19}$$

It should be mentioned that due to linearity of the interpolation of the $\mathcal{Q}$-function the adjusted $\mathcal{Q}$-values are still subject to errors in addition to those due to the imperfectness of the current $\mathcal{Q}$-values at the basis points. This should be compared with the 'constant' approximation in case when the $\mathcal{Q}$-learning algorithm and the Kohonen algorithms are unrelated. For high net dimension, however, linear interpolation is preferable to non-linear schemes [4]. On the other hand a certain degree of readaptation of the $\mathcal{Q}$-values is certainly helpful in the present method in order to account for higher order errors.

A similar problem arises if the true $\mathcal{Q}$-function is discontinuous. Discontinuities relate to the discretization error in the $\mathcal{Q}$-function. There is no knowledge in the w_j where exactly the discontinuity is in between w_a and w_b, i.e. the mean value (17) is a good guess in both cases of w'_j being actually right or left to the discontinuity of the optimal $\mathcal{Q}$-function. Thus, an readaptation of the $\mathcal{Q}$-function is also needed, but this is less essential and can moreover be performed at a much faster time scale.

The efficiency needs to be proven in a typical problem to which the next section is devoted.

5 Example: Balancing a pole

The principles described above have been exemplified by the standard task of balancing a pole. The state of the pole is determined by its angular position θ and velocity $\dot{\theta}$. The vector $x = \binom{\theta}{\dot{\theta}}$ serves as input to the $\mathcal{K}$-learning partitioner. The learning rate of the partitioner is modified both by the frequency of inputs and the diversity of control actions as described above. The $\mathcal{Q}$-learning of control actions is subject to compensation of category drift and contains a simple form of explorational behavior by choosing actions according to (1) with probability c and randomly otherwise.[2] c is linearly increased reaching unity at the end of the learning period and kept at this value during the test phase. The length of the test phase is one third of the learning phase.

For the purpose of illustrating the principles described so far we have simulated the dynamics of the pole using the dynamical equations from Ref. [5].

$$(1 + \sin^2\theta)\ddot{\theta} + \frac{1}{2}\dot{\theta}^2 \sin(2\theta) - 2\sin\theta = -f\cos\theta \tag{20}$$

where θ is the angular deviation from the upright position. The pole (or inverted pendulum) consists of two point masses m and M which for convenience are set equal to 1. These are connected by a massless bar of length 1. The control force f is applied horizontally to the bottom mass.

Upon receiving an input the Kohonen partitioner returns the bestmatching unit as the category to which the input vector $(\theta, \dot{\theta})$ belongs. The category is submitted to the $\mathcal{Q}$-learning controller together with a reinforcement signal. Actions proposed by the controller are reinforced if a target region close to the vertical position is reached or they are penalized if the deviation exceeds a certain value.

The number of categories (Kohonen units) was 49 and 3 different actions have been allowed. More general cases, i.e. square nets of varying numbers of neurons $(4 \ldots 4096)$ or a larger number of actions which were symmetrical with respect to a zero action and equidistantly increasing from zero led for appropriate convergence times to similar results as reported below. The maximum allowed action was determined by comparison to a PD controller, i.e. $f = 5\sin\theta + \dot{\theta}$.

We compare the performance of the pole balancer trained by the full algorithm to algorithms lacking either compensation of category drift or action-dependent learning rate control in the $\mathcal{K}$-learning. As criterion of the performance we used the number of restarts that were necessary when the pole deviated more than one radiant form the upright position. Fig. 1 compares the algorithm with and without action-dependent learning, while Fig. 2 illustrates the improvement due to drift compensation. Displayed are the means over 100 runs of the moving-average failure frequencies.

The setup of the experiment allowed to distinguish three learning phases: Initially, while the controller was essentially untrained the action-dependent $\mathcal{K}$-learning did not cause an improvement, whereas the compensation of category drifts showed a stronger effect, cf. Figs. 1, 2. While later the $\mathcal{Q}$-learning was still exploratory the number of failures tended to be fairly low and decreased with decreasing exploration rate. Differences among the variants of the algorithm are significant in this phase. During the test phase

[2]Strategies for exploring the combined state-action space are discussed in the companion paper [6].

no restarts were required except if the drift of categories was not compensated, where in a few runs the pole fell down.

6 Conclusion

We have proposed principles for fine-tuning the interaction of a Q-learning controller with the underlying state space partitioner. A speed-up of the learning is possible in the described algorithm due to the conservation of information stored in the Q-function against drift of category vectors. The skilled controller exhibits increased stability in a changing environment because of its ability of suppressing feed-back loops arising from the influence of the controllers activity on the state distribution. Some of the main handicaps in using a Kohonen partitioner in a general control task, hence, have been eliminated. Further we want to mention that the interpolation scheme of the Q-function relies essentially on the topology preservation of the self-organizing feature maps.

Acknowledgements

Part of this work has been done in the project "LADY: Lernverfahren zur Adaption Autonomer Einheiten in Dynamischer Fehlerintoleranter Umgebung" sponsored by the German Federal Ministry of Research and Technology (BMFT) under grant 01 IN 106B/3 and the project "Principles of Cortical Computation" of the EU-program "Human Capital and Mobility" under grant CHRX-CT93-0097.

References

[1] C. J. C. H. Watkins (1992) $\mathcal{Q}$-learning. *Machine Learning* 8, p. 279-292.

[2] T. Kohonen (1984) Self-organization and associative memory. Springer, Berlin, Heidelberg, New York.

[3] M. Herrmann, H.-U. Bauer, R. Der (1994) The 'Perceptual magnet effect': A model based on self-organizing feature maps. Submitted to *Biol. Cybern.*

[4] H. Ritter (1993) Parametrized self-organized maps. Proc. Int. Conf. on Artificial Neural Networks. Springer, London, p. 568–575.

[5] H. Ritter, T. Martinetz, K. Schulten (1992) Neural computation and self-organizing feature maps. Addison Wesley, Reading, Mass.

[6] R. Der, M. Herrmann, D. Smyth (1994) Explorations in $\mathcal{Q}$-learning. Part II. Explicit results for simple models. Submitted.

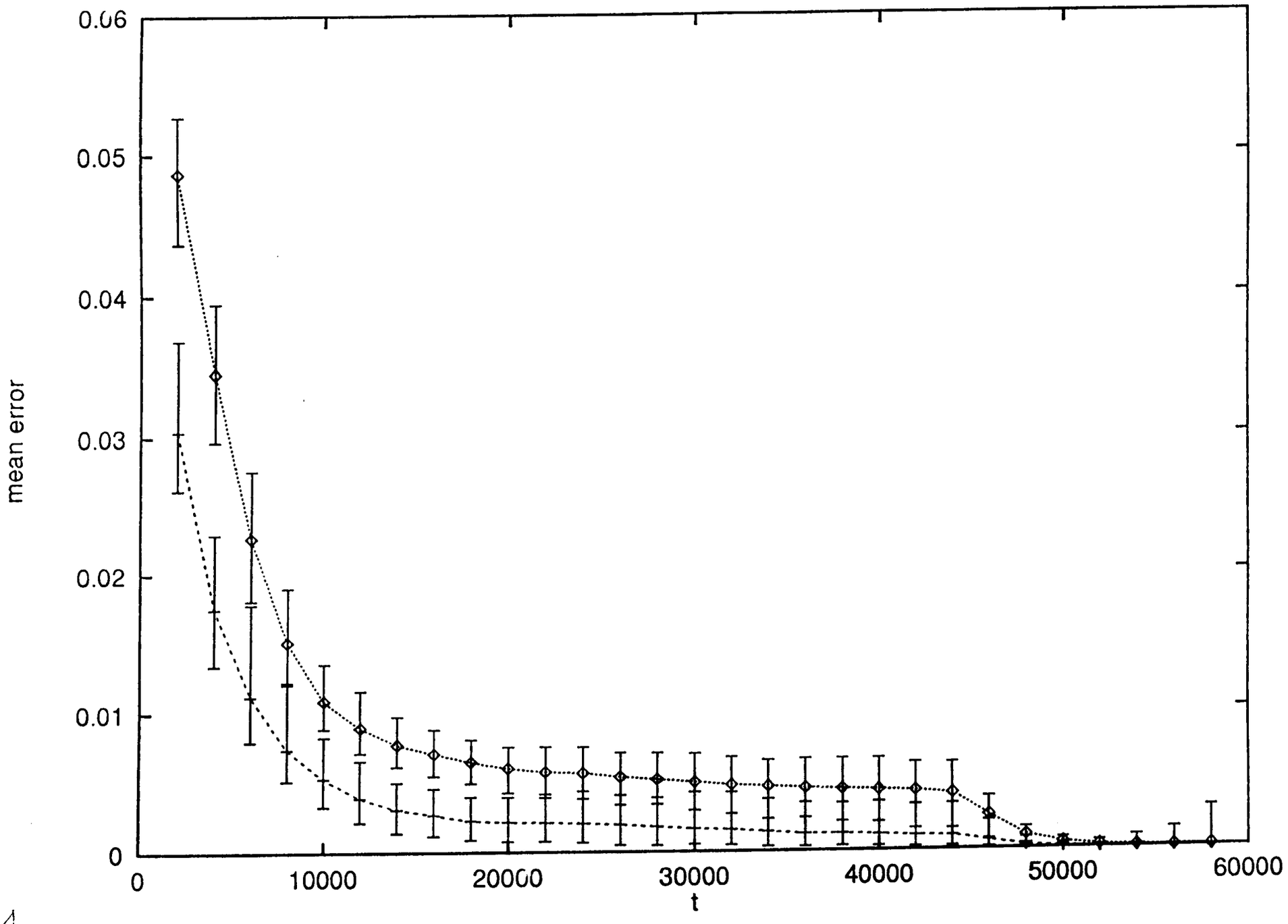
[7] R. Der, M. Herrmann (1994) Explorations in $\mathcal{Q}$-learning. Part III. $\mathcal{Q}$-learning chaos controller. Submitted.

(brief version in Proc. WCNN'94, Orlando, Florida, p. 2472–2475.)

Figure captions

Figure 1: Improvement of performance due to action dependent learning rate.

Figure 2: Improvement of performance due to compensation of drift of category prototypes


0.06
0.05
0.04
0.03
0.02
0.01
0
mean error
0
10000
20000
30000
40000
50000
60000
t

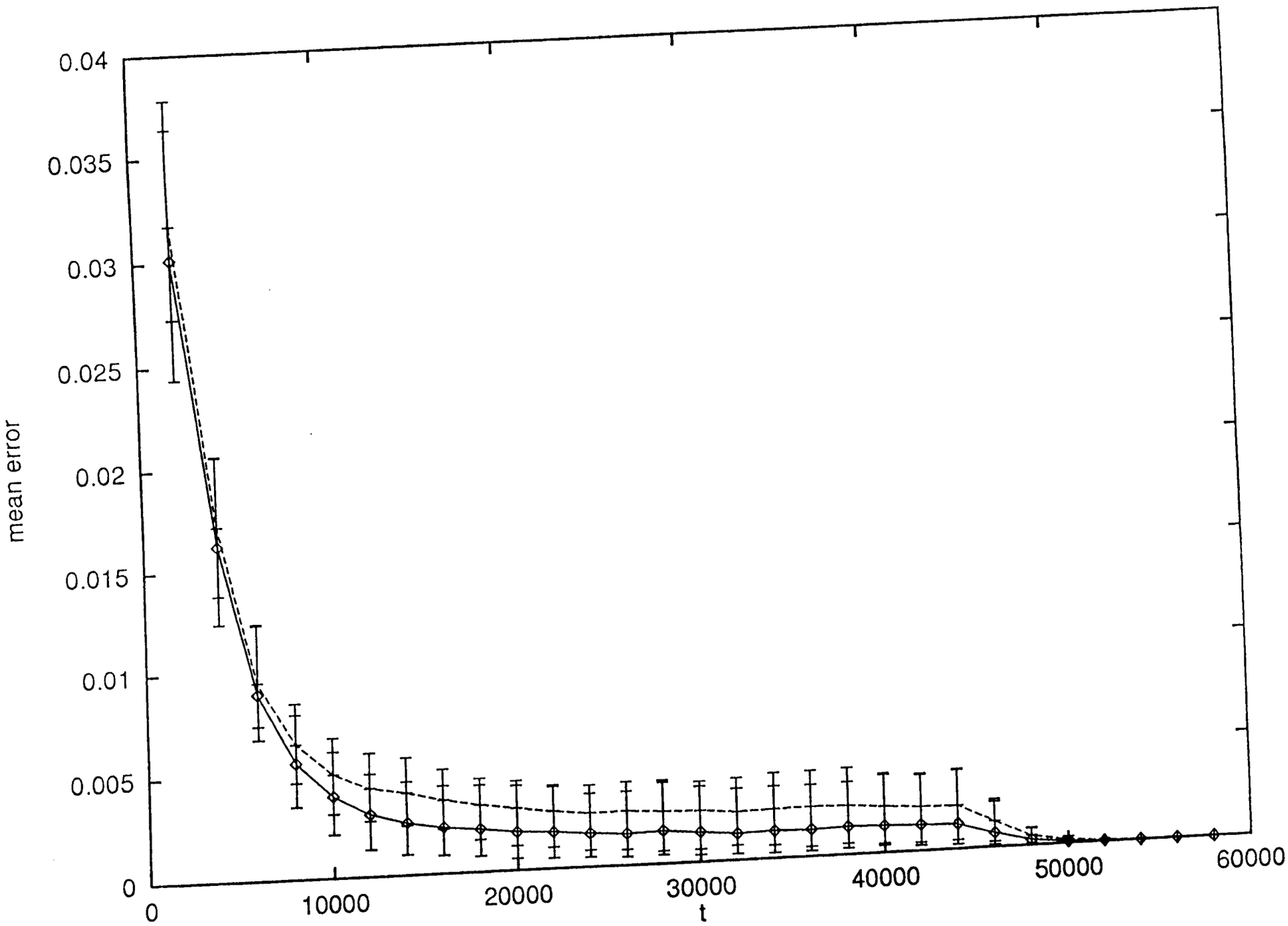

0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
mean error
0
10000
20000
30000
40000
50000
60000
t

fig 2.

Appendix

A Compensation of the drift of categories

Here we derive the transformation rule of the $\mathcal{Q}$-function in the case of arbitrary dimensions of both the net (d) and the state space (n). Consider a category j the vector w_j of which has moved to position w'_j. Suppressing the index j we write $w \longrightarrow w'$.

The values $\mathcal{Q}(w', a)$ are approximated by interpolation of $\mathcal{Q}$-values at positions w^i close to w', that not necessarily include $\mathcal{Q}(w, a)$.

$$\mathcal{Q}(w', a) = \sum_{i=0}^{d} \beta_i \mathcal{Q}(w^i, a), \quad \sum_{i=0}^{d} \beta_i = 1 \tag{21}$$

The w^i refer to the partitioner before the last update or the most recent series of updates. That category vector is chosen as w^0 which is closest to w'. Further, we choose out of each pair of neighbors of w^0 the one which is closest to w' as w^i, $i = 1, \ldots, d$. In the case of edge neurons there is sometimes no choice, of course. Introducing relative coordinates by

$$\begin{aligned} v_i &= w^i - w^0, \quad i = 1, \ldots, d \\ v' &= w' - w^0 \end{aligned} \tag{22}$$

the vectors v_i define a d-dimensional subspace $\mathcal{A} \subseteq R^n$, such that v' may be splitted into a vector in $\mathcal{A}$ and an orthogonal component.

$$v' = v'_{\mathcal{A}} + v'^{\perp} \tag{23}$$

$v'_{\mathcal{A}}$ is a linear combination of the v_i

$$v'_{\mathcal{A}} = \sum_{i=1}^{d} \beta_i v_i \tag{24}$$

On the other hand $v'^{\perp}$ is orthogonal to any v_j, i.e.

$$\begin{aligned} 0 &= \langle v'^{\perp}, v_j \rangle \qquad\qquad \forall j = 1, \ldots, d \\ &= \langle v' - \sum_{i=1}^{d} \beta_i v_i, v_j \rangle \\ \Leftrightarrow \langle v', v_j \rangle &= \sum_{i=1}^{d} \beta_i \langle v_i, v_j \rangle \end{aligned} \tag{25}$$

In order to determine the coefficients β_i, $i = 1, \ldots, d$ this system of d linear equations has to be solved. For small d Cramer's rule most conveniently yields the coefficients in (21). For $d \leq 2$ cf. Eqs. (18, 19). In the general case we have