CBR for State Value Function Approximation in Reinforcement Learning

Thomas Gabel and Martin Riedmiller

Neuroinformatics Group
Department of Mathematics and Computer Science
Institute of Cognitive Science
University of Osnabrück, 49069 Osnabrück, Germany
{thomas.gabel|martin.riedmiller}@uni-osnabrueck.de

Abstract. CBR is one of the techniques that can be applied to the task of approximating a function over high-dimensional, continuous spaces. In Reinforcement Learning systems a learning agent is faced with the problem of assessing the desirability of the state it finds itself in. If the state space is very large and/or continuous the availability of a suitable mechanism to approximate a value function – which estimates the value of single states – is of crucial importance. In this paper, we investigate the use of case-based methods to realise that task. The approach we take is evaluated in a case study in robotic soccer simulation.

1 Introduction

Case-based Reasoning (CBR) is based on the assumption that similar problems have similar solutions. Systems relying on that paradigm have been successfully used in several application domains, such as diagnosis, classification, prediction, control and action planning. Various reasons have contributed to the attractiveness of employing case-based methods: They are straightforward to implement, help in reducing the knowledge acquisition effort and they are noise-tolerant due to their approximate nature. In this work we will exploit these advantages in the context of Reinforcement Learning and thus, more specifically, in an application field that covers the tasks of prediction and action planning.

Reinforcement Learning (RL) follows the idea that an autonomously acting agent obtains its behaviour policy through repeated interaction with its environment on a trial-and-error basis. The experience the agent gathers that way is then processed and integrated into a mathematical function that tells how much it is worth aspiring to enter a specific state by performing a specific action. So, one central issue in RL represents the learning of that function, which reflects the value of a state and from which a good policy for action choice may be induced. That task is aggravated when the set of states in which the agent can find itself is infinite, i.e. when working with a large, continuous state space. Then, storing states' values explicitly is impossible and, hence, it becomes indispensable to make use of a suitable function approximation mechanism.

In this paper we investigate the use of CBR methods for that task. Their application seems promising insofar as they are considered suitable for handling noisy data and learning and generalising fast from few training examples. However, the approximation of a state value function in RL bears some inherent difficulties to be coped with: In particular, the function that we want to approximate with maximal accuracy is a moving target, i.e. changes over time, since at the beginning of the learning process only little is known about its shape, whereas at later stages of learning much more experience about its real shape will have been collected. Therefore, we present a systematic compilation of various CBR techniques to deal with this and other important problems and examine the capabilities of a CBR-based state value function approximation compared to a table-based and neural net-based function representation.

In Section 2 we introduce the necessary vocabulary, review some basics of the Reinforcement Learning paradigm and motivate the use of CBR technology to represent and approximate a state value function. Section 3 introduces our CBR-based approach to state value function approximation. We present a specialised RL algorithm, that employs a CBR-based function approximator, as well as necessary methods required for case base management. Furthermore, we discuss benefits and limitations of the ideas given. Section 4 reveals one of the underlying motivations of our work: Our research group participates in the RoboCup championship tournaments in robotic soccer simulation, where one of our main research goals is to realise an increasing part of our soccer-playing agents' behaviour by using machine learning techniques. So, we outline a specific sub-task – the intercept ball problem – of robotic soccer simulation and present results in solving that task with RL which we obtained using CBR methods for approximating the underlying state value function. Finally, Section 5 concludes.

2 The Reinforcement Learning Framework

One of the general aims of Machine Learning is to produce intelligent software systems, sometimes called agents, by a process of learning and evolving. Reinforcement Learning represents one approach that may be employed to reach that goal. In an RL learning scenario the agent interacts with its initially unknown environment, observes the results of its actions, and adapts its behaviour appropriately. To some extent, this imitates the way biological beings learn.

In each time step an RL agent observes the environmental state and makes a decision for a specific action, which, on the one hand, may incur some immediate reward (also called reinforcement) generated by the agent's environment and, on the other hand, transfers the agent into some successor state. The agent's goal is not to maximise the immediate reward, but its long-term, expected reward. To do so it must learn a decision policy π that is used to determine the best action for a given state. Such a policy is a function that maps the current state $s \in S$ to an action a from a set of viable actions A. This idea of learning through interaction with the environment can be rendered by the following steps that must be performed by an RL agent (illustrated and refined in Figure 1):

1. the agent perceives an input state s
2. the agent determines an action a using a decision making function (*policy*)
3. action a is performed
4. the agent obtains a scalar reward r from its environment (reinforcement)
5. information about the reward r that has been received for taking action a in state s is processed

The basic Reinforcement Learning paradigm is to learn the mapping $\pi : S \rightarrow A$ only on the basis of the rewards the agent gets from its environment. By repeatedly performing actions and observing resulting rewards, the agent tries to improve and fine-tune its policy. The respective Reinforcement Learning method (step 5) specifies how experience from past interaction is used to adapt the policy. Assuming that a sufficient number of states has been observed and rewards have been received, the optimal decision policy will have been found and the agent following that policy will behave perfectly in the particular environment.

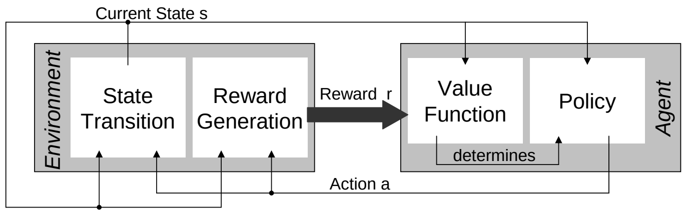

Fig. 1. Schematic View on RL Using State Value Functions

2.1 Learning Value Functions

The Reinforcement Learning problem is usually formalised as a Markov Decision Process (MDP) within the context of Dynamic Programming [3]. An MDP is a 4-tuple $M = [S, A, r, p]$ where S denotes the set of environmental states, A the set of actions the agent can perform, and $r : S \times A \rightarrow \mathbb{R}$ the function of immediate rewards $r(s, a)$ (sometimes called costs of actions) that arise when taking action a in state s. The function $p : S \times A \times S \rightarrow [0; 1]$ depicts a probability distribution $p(s, a, s')$ that tells how likely it is to end up in state s', when performing action a in state s.

Being in search of an optimal behaviour in an unknown environment, the agent needs a facility to differentiate between the desirability of possible successor states, in order to decide on a good action. A common way to rank states is by computing and using a so-called *state value function* $V^\pi : S \rightarrow \mathbb{R}$ which estimates the future rewards that can be expected when starting in a specific state s and taking actions determined by policy $\pi : S \rightarrow A$. Thus, $V^\pi(s) = E[\sum_{t=0}^{\infty} r(s_t, \pi(s_t)|s_0 = s)]$, where $E[\cdot]$ denotes the expected value. If we assume we are in possession of an "optimal" state value function $V^\star$, it is easy to infer the corresponding optimal behaviour policy[1] by exploiting that value function greedily according to $\pi^\star(s) := \arg\max_{a \in A}\{r(s,a) + \sum_{s \in S} p(s,a,s') \cdot V^\star(s')\}$.

So, the crucial question is, how to obtain the optimal state value function. To perform that task, Dynamic Programming methods, e.g. value iteration [2], may be employed which converges under certain assumptions to the optimal value function $V^\star$ of expected rewards. Value iteration is based on successive updates of the value function for all states $s \in S$ according to $V_{k+1}(s) := max_{a \in A}\{r(s,a) + \sum_{s' \in S} p(s,a,s') \cdot V_k(s')\}$, where index k denotes the sequence of approximated versions of V, until convergence to $V^\star$ is reached.

Research in Reinforcement Learning, however, has generated a variety of methods that extend those well-known optimisation techniques, aiming at applicability also in situations where large state spaces must be handled or the absence of a transition model p prevents the usage of simple value iteration. Although some details of the RL learning algorithm we use in the scope of this work are given in Section 2.2, a discussion on progress and state of the art in RL goes beyond the scope of this paper; the interested reader is referred to [21].

2.2 Temporal Difference Methods

Temporal difference (TD) methods comprise a set of RL algorithms that incrementally update state value functions $V(s)$ after each transition (from state s to s') the agent has gone through. This is particularly useful when learning along trajectories $(s_0, s_1, \dots, s_N)$ that start in some start state s_0 and end in some terminal state $s_N \in G$, where G is a set of goal state. This means learning can be performed online, i.e. the processes of collecting (simulated) experience and learning the value function run in parallel. In this work we update the value function's estimates according to the $TD(1)$ update rule [20], where the new estimate for $V(s_k)$ is calculated as $V(s_k) := (1-\alpha) \cdot V(s_k) + \alpha \cdot return(s_k)$ with $return(s_k) = \sum_{j=k}^{N} r(s_k, \pi(s_k))$ indicating the summed rewards following state s_k and α representing a decaying learning rate. The whole episode-based $TD(1)$ learning algorithm to be used in conjunction with a table-based function representation of V (i.e. the state value for each state is stored explicitly) proceeds as in Algorithm 1.

One inherent feature of this learning algorithm – as well as of any algorithm that optimises a state value function – is that at the beginning of learning the estimates for $V(s)$ are typically very coarse. To put it differently, initially V represents a rather noisy estimate of the true optimal value function $V^\star$, steadily converging towards $V^\star$ as long as all criteria for convergence are fulfilled. For the family of $TD(\lambda)$ algorithms convergence is guaranteed, if each state is visited by an infinite number of episodes and if the step size parameter α diminishes towards zero at a suitable rate.

[1] Note, that often – in particular, in cases where no model p of the environment is available – state-action value functions $Q : S \times A \rightarrow \mathbb{R}$ are learnt, which provide an estimation of how desirable it is to choose a specific action in a certain state. The paper at hand, however, deals with state-value functions only.

```
1. initialise state value function $V$ arbitrarily, let policy $\pi$ be given by $\varepsilon$-
   greedy exploitation of $V$
2. repeat
   (a) generate random start situation $s_0$ for current episode, set $k := 0$
   (b) while $s_k \notin G$ and $k < maxEpisodeLength$ do
       i. choose next action $a_k$ by exploiting $V$ $\varepsilon$-greedily according to
          $a_k := argmax_{a \in A}(r(s_k, a) + \sum_{s' \in S} p(s_k, a_k, s') \cdot V(s'))$
          or choose a random action with probability $\varepsilon$
       ii. perform $a_k$, entering state $s_{k+1}$ and perceiving immediate reward
          $r(s_k, a_k)$
   (c) for all steps $s_k$ in episode $(s_1, \dots, s_N)$
       i. $return(s_k) := \sum_{j=k}^{N-1} r(s_j, a_j) + r(s_N)$
       ii. $V(s_k) := (1 - \alpha) \cdot V(s_k) + \alpha \cdot return(s_k)$ with $\alpha$ as learning rate
   until stop criterion becomes true
```
Algorithm 1:
Episode-Based RL Algorithm Using Table-Based State Value Function

2.3 The Need for Function Approximation

As outlined in the previous sections, the determination of an optimal state value function is crucial to most RL methods. Intending to show the functioning of some new RL technique in principle, one usually chooses typical benchmark problems (grid worlds) that are very limited in terms of state and action space size. In those cases, having to deal with only a finite number of states, it is feasible to store $V(s)$ for each single state $s \in S$ explicitly using a tabular function representation with $|S|$ table entries. However, when aspiring to apply RL techniques to real world problems – as we do in this paper – and thus working with high-dimensional and probably continuous state spaces, computational and/or memory limitations inhibit the use of a tabular function representation. Instead, the employment of a function approximator becomes inevitable.

Thus, we deal with "suboptimal" methods that approximate the optimal state value function $V^\star(s)$: We replace the optimal value function by an appropriate approximation $\tilde{V}(s,t)$, where t determines the set of the approximator's parameters. In particular, we focus on the use of Case-Based Reasoning as a suitable technique to approximate $V^\star$ using k-nearest neighbour regression and compare it to other function approximation methods.

3 CBR-based Value Function Approximation

When approximating some target function $f(x) = y$, the system is usually provided with a set of training data tuples (x_i, y_i) of f's desired input-output behaviour and tries to reconstruct f so that these data pairs are explained best. So, for the case of approximating a state value function, an ideal training data set would be made up of tuples $(s, V^\star(s))$ with s covering some subset of S. Unfortunately, learning the optimal state value function $V^\star$ is the overall learning goal, which is why obtaining such a training set is impossible. In other words, the approximation of the value function must be conducted in parallel to computing

$V^\star$, which complicates the function approximator's adjustment heavily. As early estimates of $V(s)$ can be interpreted as noise-afflicted versions of the optimal values $V^\star(s)$, the application of CBR to approximate V appears promising in that respect. Moreover, CBR systems are straightforward to implement and comparatively easy to tune. This argument is striking when comparing CBR as function approximation scheme with the use of neural nets, which are notoriously hard to tune in the context of RL algorithms. The latter advantage of CBR is supported by Gordon [7]: A case-based function approximator can be characterised as a contraction mapping ("averager") whereas neural nets fall into the category of expansion mappings ("exaggerators"), that can exaggerate changes in their training values and cause instability in the respective learning algorithm.

3.1 Related Work

CBR-related (case-based, instance-based, and sometimes so-called memory-based) techniques have been used in the context of Reinforcement Learning at times.

The idea of using instances of stored experience to predict the value of some *solution attribute* of a new unseen example is the main feature of case-based regression algorithms. In [8] the idea of weighted k-nearest neighbour regression is introduced. Here, the numerical prediction of a query's solution attribute is determined as a weighted average of the solution attribute values of the query's nearest neighbours. Peng [13] was one of the first to use a nearest-neighbour approach in the context of value function approximation for RL. In that work, however, the important topic of case-base management is not addressed. Suitable techniques to limit the potentially rapid growth of the case base by remembering too many cases have been presented later on: For example, in [5] the authors apply instance-based regression in a relational RL context and develop strategies to confine the data inflow. Similar ideas are also part of the work of Ratitch [14], though here Sparse Distributed Memories, which are a specialised application of CBR using specific similarity measures, are used as the underlying prediction technique. In both [17] and [6], promising results of approximating value functions in continuous state spaces for dynamic control tasks are presented. Their special focus is, in the case of the former, to learn from a small amount of data, boosting the learning process with initial training examples from a human expert, and, in the case of the latter, relevant extensions that allow their algorithms' application also in more complex domains. A comprehensive article addressing the comparison of several memory-based approaches to function approximation is the one by Santamaria et. al [16]. Using their terminology the ideas we present in this paper ought to be classified as instance-based methods, as they reserve the term "case-based" explicitly for situations where the actions to be chosen represent the cases' solutions. Nevertheless, we proceed using the well-established CBR vocabulary in the following. The contribution of this work lies in a systematic compilation of various CBR techniques in an RL context and their application to tasks with real-time constraints. Moreover, we examine the performance of a CBR-based value function approximation in a case study in robotic soccer and compare it to other function approximation methods.

3.2 Function Approximation with *k*-Nearest Neighbour Regression

Our approach to CBR-based state value function approximation is based on the following main characteristics, that will be discussed in more detail subsequently:

- an attribute-value based state/case representation,
- the local-global principle for similarity assessment and retrieval, and
- k-nearest neighbour regression to predict the cases' solution attribute.

We assume a continuous, n-dimensional state space $S \subset \mathbb{R}^n$ where each $s = (s_1, s_2, \ldots, s_n) \in S$ is a vector of real numbers and each dimension has its individual domain $D_i \subset \mathbb{R}$. Accordingly, we define a *case* c^s for state s to be an $n+1$-dimensional real-valued vector $c^s = (s_1, \ldots, s_n, c^s_v)$, where the first n elements represent the case's problem part and correspond to state s. The last entry depicts the case's solution $c^s_v = V(s)$, i.e. the expected reward when the RL agent starts from s.

Using this notation the *global similarity* between two cases is defined as

$$sim(c^{s_1}, c^{s_2}) := \sum_{i=1}^{n} w_i \cdot sim_i(c_i^{s_1}, c_i^{s_2}) \tag{1}$$

The weights w_i, which are normalised so that $\sum_{i=1}^{n} w_i = 1$, are used to strengthen or weaken the relevance of individual dimensions. For all $i \in \{1, \ldots, n\}$ a *local similarity measure* $sim_i : D_i \times D_i \to [0,1]$ assesses the degree of similarity along a single dimension. Currently, we use the Euclidian distance for all sim_i. However, as previous research in learning similarity measure has shown [18], the adjustment of feature weights as well as of local measures may have a significant influence on the system's performance. Therefore, we currently plan to incorporate some of these ideas into our RL learning framework.

Case value (or state value, respectively) prediction according to k-nearest neighbour regression is defined as

$$\tilde{V}(s, CB) := \frac{\sum_{c^{s_i} \in NN_k(c^s)} sim(c^s, c^{s_i}) \cdot c_v^{s_i}}{\sum_{c^{s_i} \in NN_k(c^s)} sim(c^s, c^{s_i})} \tag{2}$$

so that $\tilde{V}(s, CB)$ stands for the currently predicted value of $V^\star(s)$ approximated with help of the CBR system's case base CB, where $NN_k(c^s)$ is the set of c^s's k nearest neighbours in CB. Other authors [12] use kernel functions to support the regression task: The weighted contribution of each neighbouring case's value c_v is then computed using the kernel being parameterised by the similarity function. Compared to that approach our regression scheme depicts a simplification, which we chose with regard to the learning of similarity measures we plan.

Working with a CBR-based value function approximation requires slight modifications to our episode-based RL algorithm given in Section 2.2. If that algorithm needs an estimated value for a specific state s it now computes $\tilde{V}(s, CB)$ instead of $V(s)$. However, the update of a state's value, i.e. the assignment of a new value to state s, cannot be realised in such a straightforward manner as in the case of the algorithm using a table-based representation of V. As can be seen in Algorithm 2 we add a new case containing the corresponding state's backed-up value to the case base, but also call appropriate case base management routines.

```
1. start with case base $CB = \varnothing$, let the approximated state value function
   be $\tilde{V}(s, CB)$, and policy $\pi$ be given by greedy $\varepsilon$-exploitation of $\tilde{V}$
2. repeat
   (a) generate random start situation $s_0$ for current episode, set $k := 0$
   (b) while $s_k \notin G$ and $k < maxEpisodeLength$ do
       i. choose next action $a_k$ by exploiting $\tilde{V}$ $\varepsilon$-greedily according to
          $a_k := \arg\max_{a \in A}(r(s_k, a) + \sum_{s' \in S} p(s_k, a_k, s') \cdot \tilde{V}(s', CB))$
          or choose a random action with probability $\varepsilon$
       ii. perform $a_k$, entering state $s_{k+1}$ and perceiving immediate reward
          $r(s_k, a_k)$
   (c) for all steps $s_k$ in episode $(s_1, \ldots, s_N)$
       i. $return(s_k) := \sum_{j=k}^{N-1} r(s_j, a_j) + r(s_N)$
       ii. create a new case $c^{new} := (s_k, return(s_k))$
       iii. $CB := CB \cup c^{new}$
       iv. call case base management routines
   until stop criterion becomes true
```

Algorithm 2:
Episode-Based RL Algorithm Using CBR-based Function Approximation

3.3 Case Base Management

Starting with an empty case base, the learning algorithm steadily increases its competence by storing new experiences. However, there are a number of reasons why the inflow of new cases ought to be limited.

– The more cases the case base contains, the longer the retrieval of the query's nearest neighbours takes. Although there exist techniques to reduce the computational effort during retrieval, e.g. *kd*-trees [6], it is advisable to limit the growth of the case base's size when intending to use the system for real-time control tasks.
– As already noted early estimates of the state value function's values represent rather noisy versions of the optimal values. Thus, it is indispensable to also discard some cases already stored. At this point, the difficulty arises to differentiate between important outliers and simply wrong estimates.
– Simple instance-based learning by just remembering all cases would not be applicable since the amount of data the agent collects would become unmanageable as the agent continues to learn.

There exist a number of approaches to remove "useless" cases during training, e.g. the IBx algorithms by Aha [1]. For learning embedded in an RL context, however, more specialised techniques are necessary. In [6] it is pointed out that being selective in adding cases may slow down the learning rate. Furthermore, we need to stress that each new case $c^{new} = (s_k, return(s_k))$ composed by Algorithm 2 contains the currently most up-to-date estimate for the state value $V(s_k)$. these reasons we insist on explicitly storing this piece of brand-new information by adding it to the case base and removing its very nearest neighbour c^j for which it holds $sim(c^{new}, c^j) > 1 - \delta$ with some extremely small $\delta > 0$.

Anyway, when the number of cases stored in CB exceeds some critical value $|CB| > \mu$, so that the realisation of a retrieval/regression within a certain amount of time cannot be guaranteed, it is inevitable to also remove some cases. A first approach to tackle that problem would be to remove the oldest or least frequently used elements of CB. This idea seems intuitive, as old cases usually contain worse estimates of the corresponding state's value than newer ones, but this strategy might lead to a function approximator that easily "forgets" some of its valuable experience made in the past. This danger may become particularly problematic, when some regions of the state space are visited rather rarely during learning and hence eventually good estimates are erased due to infrequent occurrence.

More complex scoring measures calculating which cases are to be removed have been proposed by several authors. In [6] it is suggested to remove those cases that contribute least to the overall approximation. In [5] the authors pursue a more error-oriented view and propose the deletion of cases that contribute most to the prediction error of other examples. A considerable flaw of those more sophisticated measures is their complexity. The determination of the case(s) to be removed involves the computation of a score value for each $c^i \in CB$ which in turn requires at least one retrieval and regression, respectively, for each $c^j \in CB$ $(j \neq i)$. These repeated entire sweeps through the case base induce an enormous computational load, although optimisations may find a partial remedy. Consequently, these approaches are not best suited in systems which are learning with tight time requirements and handling a high-dimensional state space, which necessitates the use of larger case bases.

For these reasons, we employ a heuristic scoring measure that is made up of three components, computationally less demanding, and brought about good results during evaluation. As formalised in Algorithm 3 this measure's components reflect the distribution of cases throughout the state space, the correctness of predictions for values of the state value function as well as the case's age.

3.4 Benefits and Limitations

The main CBR principle, telling similar problems have similar solutions, can also be utilised when employing case-based methods for function approximation, provided that the target function to be approximated can be characterised as locally smooth. So, CBR's robustness against noisy data also applies when approximating state value functions. All experience is stored explicitly so that the negative influence of a wrong state value estimate is only local. In the RL context the function to be approximated is learnt concurrently with acting and thus is not static, but changes over time converging towards $V^\star$. Hence, early experience may be considered as "noise" at later stages of learning.

CBR is an approximate technique by nature. Accordingly, the quality of a case-based value function approximation depends strongly on the number of cases stored. Aiming to tackle high dimensional state spaces, the case base size that is needed to obtain high-quality approximations grows exponentially with the number of dimensions. Then, not only a memory shortage may arise, but also real-time usage of the system becomes impractical, as the time for case

```
1. if $|CB| \leq caseBaseMaxSize$ return
2. for all $c^i \in CB$
    (a) compute the set $NN_k(c^i)$ of the $k$ nearest neighbours around $c^i$
    (b) compute the similarity density around $c^i$ as
        $\varphi(c^i) := \frac{1}{k}\sum_{c^j \in NN_k(c^i)} sim(c^i, c^j)$
    (c) compute the standard deviation of stored state values within $c^i$'s nearest neighbours as $\sigma_v(c^j) := \sqrt{\frac{1}{k}\sum_{c^j \in NN_k(c^i)}(c_v^i - c_v^j)^2}$
    (d) compute the score components
        i. case neighbourhood score: $S_n(c^i) := \varphi(c^i) \cdot \sigma_v(c^i)$
        ii. regress error score: $S_e(c^i) := \sum_{c^j \in NN_k(c^i)} sim(c^i, c^j) \cdot |c_v^j - \hat{c}_v^j|$
            where $\hat{c}_v^j$ is the system's prediction for $c_v^j$ using $CB \setminus c^j$
        iii. age score: $S_a(c^i) := \frac{t(c^i)}{2|CB|}$ with $t(c^i)$ telling how many time steps
            ago $c^i$ has been added to $CB$
    (e) let the overall score $S(c^i)$ be the sum of its component
3. delete $\varrho$ cases with highest score values
```
Algorithm 3: Case Base Management: Deletion of Stored Cases

retrieval/regression grows at least logarithmically with the number of stored cases. Thus, a trade-off between approximation quality and real-time constraints has to be found. Another meaningful advantage of CBR systems is the speed at which they learn. As each piece of experience is remembered explicitly, the system is capable of representing a quite good, though far from perfect, function approximation with a rather small number of cases.

To sum up, we can distinguish two main application fields for CBR-based function approximators: If maximal accuracy in approximating V and/or real-time application of the policy are not an issue, an RL agent using a case-based value function representation can become applicable within shortest time. Otherwise, a case-based function approximator might be used for the starting stage of the learning process: That way, average or even good approximation results may be obtained within a very short time and used to initialise and speed up the training of another approximator (e.g. a neural net) with which nearly maximal accuracy can be attained. As for the experiments presented in the following, we focus on the latter use of a case-based state value function approximator.

4 Experimental Evaluation

In the previous section we have introduced a number of methods to apply a CBR-based approach to state value function approximation within a Reinforcement Learning context. Now, we want to investigate the performance and usability of the ideas presented, comparing them to two different approaches to function approximation, viz a table-based representation and neural nets. The application scenario our evaluation is embedded in is robotic soccer simulation and thus, in particular, our research group's RoboCup competition team Brainstormers [10].

4.1 Robotic Soccer Simulation

RoboCup [22] is an international research initiative intending to expedite AI and intelligent robotics research by defining a set of standard problems where various technologies can and ought to be combined to solve them. Annually, there are championship tournaments in several leagues – ranging from rescue tasks to real soccer-playing robots and simulated ones. The focus of the evaluation at hand is laid upon RoboCup's 2D Simulation League, where two teams of simulated soccer-playing agents compete against one another using the Soccer Server [11], a real-time soccer simulation system.

Robotic Soccer represents an excellent testbed for Machine Learning and, particularly, for RL tasks. Several research groups have dealt with the task of learning parts of a soccer-playing agent's behaviour autonomously (e.g. [9]), also relying on case-based methods at times (e.g. [4]). From a learning point of view it is also our long-term goal to realise an agent that obtains its behaviour by entirely employing a Reinforcement Learning methodology: Although we made some progress towards tackling the more complex task of learning a cooperative team behaviour [10], the most convincing learning results have been obtained for smaller sub-problems so far, especially for the learning of basic behaviours, so-called *skills*.

Intercept Ball Task

One of the most important fundamental capabilities of a soccer player – whether simulated or real – is to intercept a running ball as quickly as possible. Since a match's course of action can only be influenced significantly, if a team is in ball possession, this skill is crucial for being competitive. In the scope of this experimental evaluation we focus on the intercept ball task.

The optimal behaviour for ball interception is of course to compute the best interception point and to move to that point along the shortest path. If the physical laws of the environment are known and the simulation is deterministic that calculation may be done exactly. An illustration of the intercept ball task is given in Figure 2. For more details on analytical solutions the reader is referred to [19]. However, as already mentioned, it is our aim to realise a growing part of our agents' behaviour as modules that were learnt using RL. Hence, we formalised the intercept task as an MDP, applied Algorithm 2, and learnt a state value function for this problem. The problem's state space is continuous and 6-dimensional, i.e. $S = \{s = (v_{b,x}, v_{b,y}, v_{p,x}, v_{p,y}, d_{bp}, \alpha_{bp})\}$ where $\boldsymbol{v_b}$ is the ball's and $\boldsymbol{v}_p$ the player's velocity, d_{bp} the distance and α_{bp} the relative angle between ball and player. Viable actions for the player are, as determined by the Soccer Server, turn (real-valued from $[-180°, 180°]$) and dash (with dash power parameter within $[0, 100]$). A ball is considered to have been intercepted successfully, when the player has gained "control" over it, which means the player has moved to the point where the ball is within the player's kickable area. We here only consider a deterministic soccer simulation environment where $\forall s \in S$ and $\forall a \in A$ there is a $s' \in S$ with $p(s, a, s') = 1$, although our algorithms and function approximation techniques work for stochastic environments as well.

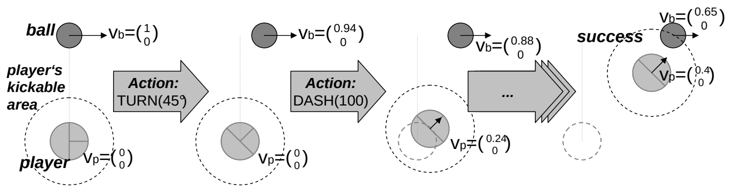

Fig. 2. Illustration of the Intercept Ball Task

4.2 Results

For the purpose of assessing the quality of learnt policies for ball interception, we focus on two evaluation measures: The average success rate measures the percentage of successful episodes, i.e. of those episodes in which the agent managed to intercept the ball in less than $maxEpisodeLength = 80$ simulation cycles. Speed brings about competitiveness. Thus, the more relevant measure are the *costs* (negative rewards) that are incurred during an episode. For the task at hand, those are best expressible in terms of the average episode length, i.e. the number of steps it took the agent to intercept the ball. All evaluation results presented in this section are based on episodes that we obtained using (a) the learnt state value function and the policy induced from it and (b) a fixed set of randomly created starting situations from which the agent had to intercept the ball.

In a first step we wanted to figure out which case base sizes are sufficient to gain satisfactory interception results. As Figure 3 shows surprisingly good results can be obtained with 1000 cases, reaching success rates of more than 90% and average sequence lengths of less than 25 steps. As a trade-off between accuracy and intended retrieval speed (being proportional to case base size) during real-time usage of the system we focus on $|CB| = 2000$ in the following.

Comparison to Other Function Approximation Methods

The most straightforward way to represent a state value function in a continuous space is to discretise the state space along each of its dimensions and to use a table to explicitly store state values. Then, of course each real state has to be mapped into the grid induced and, the other way round, each table entry represents an entire subset of the state space. For the intercept ball problem and the comparison to a CBR-based approach we employed tables of different sizes ($5k$, $100k$, and $600k$ entries). Note, that with the exception of only the smallest table, these approaches exceed the CBR-based approaches' memory requirements by far (Figure 3). As to be expected, finer discretisations yield improved results. Interestingly, the difference between $T100k$ and $T600k$ is only marginal and even the latter does not supersede the results of $CB500/2000$. Secondly, we employed neural nets (feedforward with one hidden layer) to approximate V. After having experienced a certain number of episodes and states, respectively, the net was trained at a time with the collected data using the backpropagation variant $RPROP$ [15]. To generate efficient and stable learning results a considerable amount of work had to be invested into tuning relevant parameters. Anyway, as neural nets are capable of representing arbitrarily complex functions, this kind of function approximator reached the best overall results, at least in the long run of learning.

Having a look at the speed of the learning process, it becomes obvious that the CBR-based versions yielded their maximal accuracy after comparatively few training episodes. Thus, a good state value function approximation could be obtained very quickly, in general within less than 2000 training episodes (note the discontinuity in the chart's abscissa). During that time neither a table-based nor a neural net-based function approximation could reach comparable results.

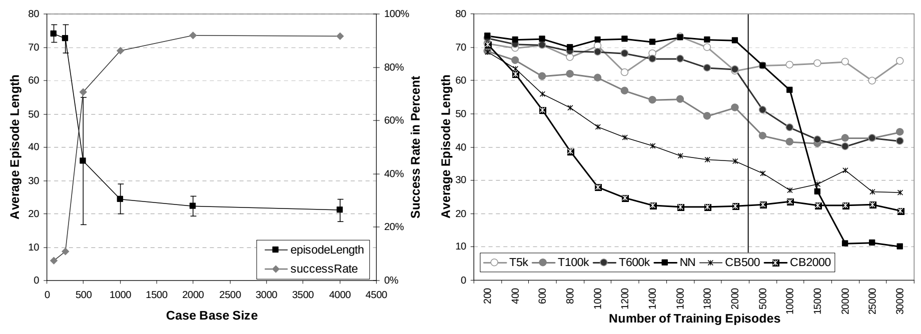

Fig. 3. Intercept Results for Varying Case Base Sizes (left) and for Different Function Approximators (right)

4.3 Discussion

The intercept case study has shown empirically that a CBR-based state value function representation can provide an approximation of good quality within very short learning time. However, it must be acknowledged that there are two important objections that prevent the employment of a completely CBR-based function approximator in a highly competitive domain like robotic soccer. First, the accuracy reached in approximating V is not sufficient when compared to the performance of the neural net as function approximator. Second, the time consumption for case retrieval grows with increasing case base size and, hence, with approximation accuracy. Consequently, it is unrealistic to perform an entire case retrieval and corresponding state value regression once per simulation cycle[2].

Nevertheless, we spot two main application scenarios for a function approximation using CBR in an RL context. On the one hand, its usage appears attractive when a new learning task is tackled: Then, it is usually difficult to figure out and settle upon relevant task-specific parameters appropriately (either when hand-coding or when trying to learn a solution for the task at hand). Using CBR might help to come to a good, though not optimal, behaviour policy,

[2] As far as a competition soccer team is considered, several agent behaviours will have to be executed (not only a behaviour for ball interception), so the 100ms a simulation cycle lasts in RoboCup cannot be reserved for the ball interception exclusively.

within little time, for example, when intending to learn more complex and less well-understood behaviours such as team-play.

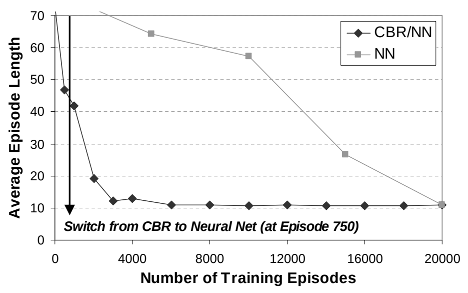

Fig. 4. Usage of CBR to Boost the Neural Net-Based Learning

On the other hand, an existing case base of state value pairs might be employed to boost the training of another function approximator. Investigating this idea, we first trained a CBR-based function approximator for a fixed number of training episodes (750) and then switched to using a neural net to represent the value function. We hereby used all the stored cases including their state values as training examples for the first training of the net and then switched to learning using that net. Figure 4 shows that the learning process could be decisively accelerated.

5 Conclusions

In this paper we applied case-based methods to approximate state value functions over high-dimensional, continuous state spaces, as required in the context of Reinforcement Learning. In so doing, we embedded a CBR-based function approximator into an episode-based $TD(1)$ learning algorithm, developed appropriate procedures to handle the growth of the case base and, for the purpose of evaluation, performed an empirical case study in the context of robotic soccer simulation, where we compared our approach to function approximation with two different ones. The results obtained showed that using a CBR-based state value function representation yields good behaviour policies for the RL agent within a very short time and with comparatively little case data. Almost optimal policies could, however, not be obtained – the quality of policies induced from neural nets representing the value function turned out to be superior, but here more tuning effort was needed to produce stable learning results.

In our view, the major strength of the CBR-enhanced learning approach is the speed with which good, though not optimal, learning results can be achieved. This refers to the fact that little time is needed to tweak a CBR system as well as to the little time needed for the learning process to run; after a few hundred training episodes already good policies are learnt. Furthermore, if one is interested in a near-optimal agent behaviour using, for example, a neural net-based state value function approximation, the learning process can be boosted using CBR as shown in Section 4.3.

An interesting issue for future research is the consideration of more sophisticated similarity measures on the basis of which to perform k-nearest neighbour retrieval and regression. This might increase the case-based function approximator's accuracy, as inherent similarities and dissimilarities of regions within the state space could be exploited better. Therefore, the incorporation of an approach to automatically optimise the CBR-based function approximator's local similarity measures and feature weights [18] seems promising.

References

1. D. Aha. Tolerating Noisy, Irrelevant and Novel Attributes in Instance-Based Learning Algorithms. *Journal of Man-Machine Studies*, 36(2):267–287, 1992.
2. R. E. Bellman. *Dynamic Programming.* Princeton University Press, USA, 1957.
3. D. P. Bertsekas and J. N. Tsitsiklis. *Neuro Dynamic Programming.* Athena Scientific, USA, 1996.
4. H.D. Burkhard, J. Wendler, T. Meinert, H. Myritz, and G. Sander. AT Humboldt in RoboCup-99. In *RoboCup*, pages 542–545, 1999.
5. K. Driessens and J. Ramon. Relational Instance Based Regression for Relational RL. In *Proceedings of ICML 2003*, pages 123–130, Washington, 2003. AAAI Press.
6. J. Forbes and D. Andre. Representations for Learning Control Policies. In *Proceedings of the ICML-2002 Workshop on Development of Representations*, pages 7–14. The University of New South Wales, 2002.
7. G. J. Gordon. Stable Function Approximation in Dynamic Programming. In *Proceedings of ICML 1995*, pages 261–268, San Francisco, 1995. Morgan Kaufmann.
8. J. D. Kelly and L. Davis. A Hybrid Genetic Algorithm for Classification. In *Proceedings of the Twefth International Joint Conference on Artificial Intelligence (IJCAI 1991)*, pages 645–650, Sydney, Australia, 1991. Morgan Kaufmann.
9. Gregory Kuhlmann and Peter Stone. Progress in Learning 3 vs. 2 Keepaway. In *RoboCup-2003: Robot Soccer World Cup VII*, Berlin, 2004. Springer Verlag.
10. A. Merke and M. Riedmiller. Karlsruhe Brainstromers – A Reinforcement Learning Way to Robotic Soccer II. In *RoboCup2001: Robot Soccer World Cup*, 2001.
11. I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer Server: A Tool for Research on Multi-Agent Systems. *Applied Artificial Intelligence*, 12(2-3):233–250, 1998.
12. D. Ormoneit and S. Sen. Kernel-Based Reinforcement Learning. Technical Report TR 1999-8, Statistics Institute, Stanford University, USA, 1999.
13. J. Peng. Efficient Memory-Based Dynamic Programming. In *12th International Conference on Machine Learning*, pages 438–446, USA, 1995. Morgan Kaufmann.
14. B. Ratitch and D. Precup. Sparse Distributed Memories for On-Line Value-Based Reinforcement Learning. In *Machine Learning: ECML 2004, 15th European Conference on Machine Learning*, pages 347–358, Pisa, Italy, 2004. Springer.
15. M. Riedmiller and H. Braun. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, pages 586–591, San Francisco, USA, 1993.
16. J. Santamaria, R. Sutton, and A. Ram. Experiments with RL in Problems with Continuous State and Action Spaces. *Adaptive Behavior*, 6(2):163–217, 1998.
17. William D. Smart and Leslie Pack Kaelbling. Practical Reinforcement Learning in Continuous Spaces. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, San Francisco, USA. Morgan Kaufmann.
18. A. Stahl and T. Gabel. Using Evolution Programs to Learn Local Similarity Measures. In *Proceedings of the 5th International Conference on Case-Based Reasoning (ICCBR 2003)*, pages 537–551, Trondheim, Norway, 2003. Springer.
19. F. Stolzenburg, O. Obst, and J. Murray. Qualitative Velocity and Ball Interception. In *Advances in AI, 25th German Conference on AI*, pages 283–298, Aachen, 2002.
20. R. S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3:9–44, 1988.
21. R. S. Sutton and A. G. Barto. *Reinforcement Learning. An Introduction.* MIT Press/A Bradford Book, Cambridge, USA, 1998.
22. M. Veloso, T. Balch, and P. Stone et al. RoboCup 2001: The Fifth Robotic Soccer World Championships. *AI Magazine*, 1(23):55–68, 2002.