Learning to Shoot Goals Analysing the Learning Process and the Resulting Policies

Markus Geipel[1] and Michael Beetz[1]

Department of Computer Science, Technische Universität München

Abstract. Reinforcement learning is a very general unsupervised learning mechanism. Due to its generality reinforcement learning does not scale very well for tasks that involve inferring subtasks. In particular when the subtasks are dynamically changing and the environment is adversarial. One of the most challenging reinforcement learning tasks so far has been the 3 to 2 keepaway task in the RoboCup simulation league. In this paper we apply reinforcement learning to a even more challenging task: attacking the opponents goal. The main contribution of this paper is the empirical analysis of a portfolio of mechanisms for scaling reinforcement learning towards learning attack policies in simulated robot soccer.

1 Introduction

Reinforcement learning is a popular method for solving complex control tasks. It has been applied to a variety of problem domains in various forms. Our contribution will be to provide an in depth analysis of reinforcement learning for a problem of high complexity. For our work we focused on learning a complex control task in the context of simulated robot soccer: the *goalshooting* scenario. In our experiments we investigated the differences between hand-coded policies and learned ones. Our next focus was the process of learning. We monitored not only the success rate but also the action usage in the team as well as the change rate in the policy of each agent. Further more we developed a method to visualize learned policies and used it to compile a video clip of the changing policy during training. Finally we developed a concise method to bring a learned policy into human readable form.

We will now briefly introduce simulated robot soccer, and then specify the goalshooting task and its properties. The perceptions in robot soccer fall into three categories: visual, acoustic and self perception. Visual percepts include the relative distance and angle to all objects within the field of view. They are tainted by random noise according to the distance of the perceived object. The standard agent disposes of three physical actions: `turn`(*angle*), `dash`(*power*) and `kick`(*power*, *angle*) as well as an action for inter-agent communication. An in depth explanation of simulated robot soccer can be found in the server manual [1]. The *goalshooting* scenario is a sub problem of RoboCup simulated soccer. It is designed to mimic the last phase of an attack on the opponent's goal. A team of

three attackers tries to score a goal. The goal is guarded by two defenders and one goalie. In the initial formation the three attackers are positioned equidistantly on the 26 meter line. The ball is positioned one meter in front of a randomly chosen attacker. The two defenders are positioned on the 10 meter line, blocking the direct shooting lanes from the wing attackers to the goal. The shooting lane of the centre attacker is blocked by the goalie. An episode is counted as failure if a time limit is exceeded, the ball gets behind the 30 meter line or leaves the field. We will call the number of successfully completed episodes divided by the number of played episodes the success rate of the attackers. To maximize this success rate is the objective in the *goalshooting* scenario.

One of the most challenging reinforcement learning tasks so far has been the 3 to 2 keepaway task in the RoboCup simulation league, introduced by Stone and Sutton [2]. *Goalshooting* however, exhibits a higher complexity than *keepaway* for the following reasons: Shooting goals demands more coordination and rewards are more scarce. Reward is given only on completing the task either with failure or success. A series of right actions by more than one agent is needed to succeed. This gives also rise to the inter agent credit assignment problem. In *goalshooting* we have chosen to use complex opponents: the "UvA trilearn players 2003" from the binary distribution serve as defenders of the goal.

This paper will be structured as follows. We will describe how the reinforcement learning algorithm SARSA(λ) the *goalshooting* task. In the next section we will describe all the techniques we developed to comprehend and interpret the policies produced by learning agents as well as the process leading to them. Finally we will point out related work and last but not least draw a conclusion.

2 Scaling Reinforcement Learning to *Goalshooting*

We did not implement our soccer agents from scratch but set the learning mechanism on top of the well known UvA Trilearn Open Source Distribution. It is available at [3] for free. Henceforth we will refer to the UvA Trilearn player of the Open Source Distribution as basic player. Figure 1 shows the architecture

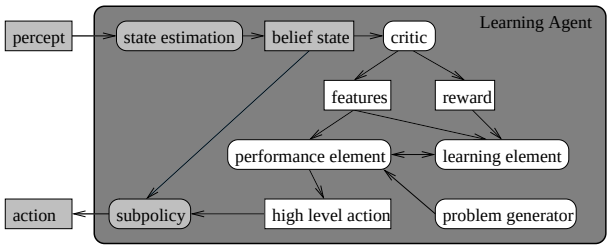


Fig. 1. The agent's architecture.

of the learning agents. The grey parts are already addressed by the basic player,

the white ones depict the learning layer that was added. The architecture takes its cues from the one proposed by Russell and Norvig [4] for learning agents. Rectangles represent data structures while boxes with round edges represent algorithmic parts. The critic calculates a feature set based on the belief state. It also estimates the utility of the current state and provides a reward signal. The performance element stores the current policy and is manipulated by the learning element, which implements the actual learning algorithm. The problem generator suggests actions for exploration.

The attackers are the ones to decide what to do when in possession of the ball. They choose an action from the set of parameterless high level actions. `shoot`: shoot the ball towards a randomly chosen corner of the goal. `dribble_goal`: dribble towards the goal. `dribble_free`: dribble towards an open spot. An open spot is chosen with the SPAR[1] algorithm. `pass_near`: pass to the nearest team mate. `pass_far`: pass to the second nearest team mate. `hold`: wait and see. They were built on top of the basic player's ready to use sub-policies or skills. Henceforth we will call these actions "high level actions" to distinguish them from the atomic actions understood by the soccer server.

The perceptions in simulated robot soccer can not directly be used as input for the learning algorithm. As can be seen in figure 1, state estimation brings the belief state in sync with the current perceptions. This is accomplished by the basic player. Still the state space is too big for learning to be feasible. The critic condenses it to a set of numerical values, called features. The feature set used by Stone and Sutton [2] to built soccer agents that master the *keepaway* task has been the role model for ours. The feature calculation is based on: the set of attackers A and the set of defenders D were the members are ordered by increasing distance from the agent, which is thus defined as a_1. g_r represents the center of the right goal. The distance between x and y is given by $\texttt{dist}(x, y)$ and the angle between x and y with vertex at z by $\texttt{ang}(x, y, z)$. The feature set consists of 7 numerical values. Four distances: $\texttt{dist}(a_1,g_r)$, $\texttt{dist}(a_1,d_1)$, $\texttt{dist}(a_1,a_2)$, $\texttt{dist}(a_1,a_3)$. Three angles: $\texttt{min}(\texttt{ang}(a_1,g_r, d \in D))$, $\texttt{min}(\texttt{ang}(a_1,a_2, d \in D))$, $\texttt{min}(\texttt{ang}(a_1,a_3, d \in D))$. The rationale is, that these angles are a good indicator for the width of an opponent free shooting lane to the goal or the team mates. Not only does the critic calculate an appropriate feature set but it also provides the reward signal.

The learning relies on the performance element, the learning element and the problem generator. The learning element depicted in figure 1 will tune the performance element based on the feature vector and the reward. It uses the SARSA(λ) algorithm as it is presented in "Reinforcement Learning: An Introduction" by Sutton and Barto [6]. The problem generators task is to suggest actions for exploration to the learning element. In our case the problem generator just returns a random action. The performance element holds the learned policy in form of two components: A value-function for each possible action and an arbitration mechanism that returns the action that has the highest value for the

[1] SPAR stands for Strategic Positioning with Attraction and Repulsion and was introduced by Veloso, Stone and Bowling [5]

current feature vector. The value-functions themselves are approximated by a CMAC-function-approximator[2].

3 Analyzing Learning and Learned Policies

Now that our test bed is properly defined we will ask the following questions: How well does a learned policy do compared to a hand-coded one and will it be more robust to changes in the environment? Will the policy itself be continuously changed or will there be bursts of changes? Is there a way to visualize the learned policies? Is there a way to bring a learned policy in to a human readable form?

3.1 Robustness of Hand-Coded Policies and Learned Ones

Question: "Are learned policies more robust in respect to changes in the environment than hand-coded ones?". In order to find an answer we used the setup just described to conduct the following experiment: First we will compare the performance of the learning algorithm with the performance of a hand-coded policy. In the next step we will slightly change the environment and look again. In order to make this comparison we need a hand-coded policy. It is essential to make a fair comparison. Thus the hand-coded policy should have exactly the same feature vector available as the learning algorithm. It took several days of testing to find and fine tune a hand-coded policy: The basic idea is to shoot if near enough to the goal. To pass as far as possible or shoot if the enemy is too near. Finally the agent will dribble if no other rule fits. The top plot (90 degrees field of view) in figure 2 shows the result: We plotted the success of the hand-coded policy as well a the success of the learned one. For the next experiment we slightly changed the environment. In the standard configuration, the agents have a view cone of 90^o. We changed it to 180^o. The bottom plot (180 degrees field of view) in figure 2 shows the result: While the learning from scratch is working just as fine as before (A), the hand-coded policy fails horribly. Furthermore the policy learned in the 90^o scenario seems to be quite robust. We see that learning agents starting with it, succeed with an average of 27 percent in the beginning and adapt pretty soon (B). The failure of the hand-coded policy is amazing because the hand-coded policy seemed to be well suited for both scenarios. This example shows that learning is the more flexible and more robust approach.

3.2 Visualization of the Learning Process

The agents basically learn a action-value-function which is represented by the function approximator. But this multidimensional function is not accessible for humans. The objective of this experiment is to visualize policies and the learning process. The key idea is to find a different policy representation. The policy can also be represented by a large number of typically visited points in the

[2] For a introduction to CMAC consult the work of J. S. Albus [7]

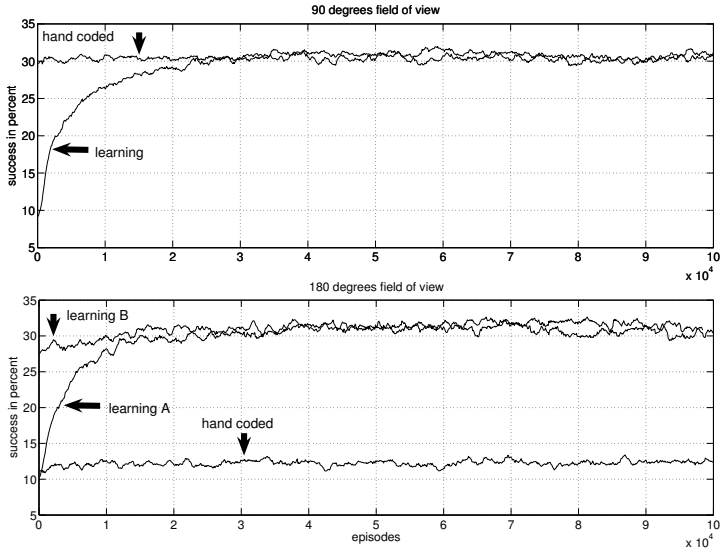


Fig. 2. Each curve is an average of 24 independent runs. See text for an explanation.

feature space. The original policy is used to tag each point with the action it would choose in that case. What we get is a set of classified samples. Principal Component Analysis was chosen to project the points in the seven dimensional space down to three dimensional one. The points are colored according to their classification. The policy after every episode was visualized as a picture. All the pictures were compiled to a video. So each frame corresponds to one episode and we can watch the learning taking place in fast motion. The video can be downloaded at `http://home.in.tum.de/geipel/da`.

3.3 The Learning Process

To see what happens to the policy during the learning process we will monitor the following aspects: success rate, action usage by the policies and change rate of the policy of each agent. The results for the training of one attacker team will be presented and interpreted. How do we assess these three aspects? Success rate is the number of successfull episodes divided by the overall number of episodes. The action usage can be extracted from the classified samples by counting all samples that were tagged with one specific action. Please note that not all samples are yet classified at the beginning of the training. For the change rate we count the samples for which the classification changed.

Figure 3 shows the results for one team of attackers. The first plot shows the success of the team. The second the usage of the different actions used by all three players. Finally the third one shows the amount of change in each agent's

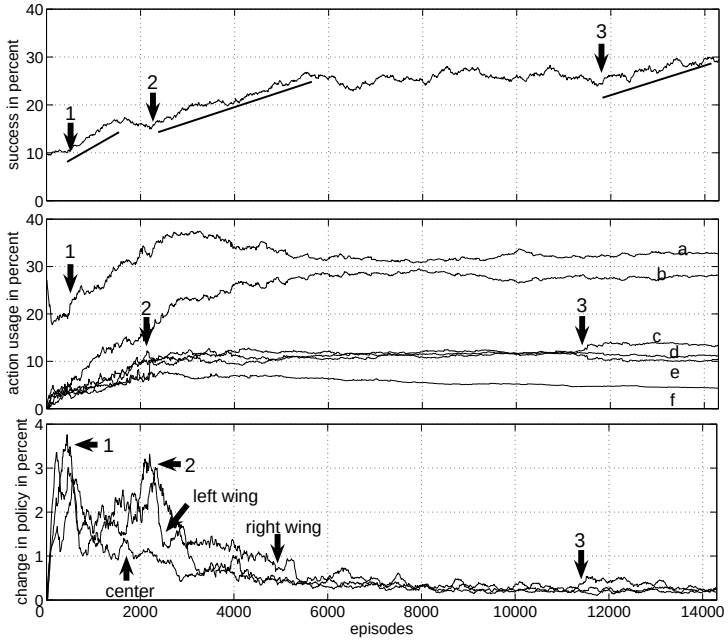


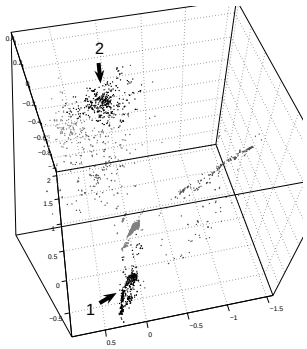
Fig. 3. Top: The success rate of the agents. Center: The usage of the actions. a a for shoot, b for dribble, c for dribble open, d for pass near, e for pass far and f for hold. Bottom: The changes in the policy for each agent.

policy. It can be seen that the success curve consists of three phases of increase with intermediate phases of stagnation. The beginning of each increase phase is marked with an arrow. The phase itself is underlined. Each phase of boosting success is launched by a sudden change in the policy of one or more agents. Phase one is most probably caused by steep increase in the usage of the action shoot. The policy of all agents undergoes heavy changes. Phase two goes along with a sudden steep increase of the usage of the action `pass far`. It is marked with an arrow and the number two. The "policy change" plot also shows that this change is only effecting agent one and three, the wing attackers. The third phase seems to be due to a change in the policy of agent one. The usage of hold is slightly increased while the usage of `dribble open` decreases.

3.4 Identifying Situations

In this subsection we describe a method to interpret the learned policy. We would like to express what the agent learned in simple rules like for example: "The agent shoots if the goal is near and free of opponents." We just saw that there are ways to visualize the policies. But this is not yet enough to properly interpret them. Again we will use the before mentioned policy representation based on a set of

classified samples. There are nearly single colored regions in the policy, as can be seen in the video. Such a region can be interpreted as a situation in which the policy chooses one action. First we will split the data by actions. This means all samples that were classified with action 0 form a new set, same is done for action 1, and so on. Now a clustering algorithm is applied to each. The clusters that are found are just these "situations". So we get a list of clusters for each action that describes in which situation these actions are applied. Figure 4 shows such a clustering for the shoot action. The Expectation-Maximization-Clusterer which



feature	mean	variance
DistanceGoal	**19.75**	7.78
DistanceNearOpponent	3.29	3.18
MinAngleTeammateNear	0.00	40.87
MinAngleTeammateFar	-0.02	0.48
MinAngleGoal	**73.81**	17.52
DistanceNearTeammate	-9.97	0.66
DistanceFarTeammate	-10.00	15.59

Fig. 4. Clustering for the `shoot` action.

was used, describes clusters by the mean and variance of every dimension or respectively feature in our case. As an example we will pick one cluster and take a look at the cluster description. The description of the cluster marked 1 with the interesting facts highlighted can also be seen in figure 4. Informally speaking it shows a situation where the agent is relatively close to the goal. The goal itself is quite clear of opponents, as the minimal angle between agent, opponents and goal (`MinAngleGoal`) is comparatively wide. Further more, information about the teammates is inconsistent, which is expressed by negative values for the features `DistanceNearTeammate` and `DistanceFarTeammate`. The action shoot makes perfect sense. However not all clusters make sense. The cluster marked 2 is an example of a rather bad decision: The agent does know the position of its teammates and is far away from the goal. Passing would be the right action, not shooting. This fact that suboptimal clusters prevail, suggests that the learning algorithm does not find a global optimum but gets stuck in a local one.

4 Related Work

There is no directly related work, concerning the analysis tools we presented. There are however several interesting approaches to solve subproblems of simulated robot soccer with reinforcement learning. Riedmiller and Merke [8] employed

reinforcement learning to tune the positioning behavior of agents attacking the goal. The scenario included seven attackers and seven defenders. The rest of the attackers behavior is hard wired. Stone and Sutton [2] used reinforcement learning successfully to train soccer agents in the *keepaway* task. The keepers try to stay in possession of the ball as long as possible while hard wired takers aim to get the ball. The learning keepers where able to outperform hand-coded ones.

5 Conclusion

In this work we concentrated on analyzing the learning process in a challenging domain, namely the *goalshooting* task. Our key findings are the following: The SARSA(λ) algorithm in combination with a CMAC function approximator is able to achieve the same success rate as a tediously tuned hand-coded policy. Even more, for small changes in the environment, the learning shows stable results while the hand-coded policy may suddenly fail. Even though the policy learned by the agents is cryptic for humans there are ways to visualize and interpret it. The fact that we find suboptimal decisions in policies where the learning already leveled off suggests that the learning agent do not find a globally optimal policy. This is backed by the fact that the learned policies are sometimes very different although their success rate is the same. Learning in the *goalshooting* scenario with the agents described in this work, takes place in phases of increasing success with intermediate phases of stagnation. The boost in performance are due to sudden changes in the agent's policy.

References

1. Cheny, M., Foroughi, E., Heintz, F., Huangy, Z., Kapetanakis, S., Kostiadis, K., Kummeneje, J., Noda, I., Obst, O., Riley, P., ens, T.S., Wangy, Y., Yiny, X.: Users manual robocup soccer server for soccer server version 7.07 and later. `http://sserver.sourceforge.net/docs/manual.pdf` (2002)
2. Stone, P., Sutton, R.S.: Scaling reinforcement learning toward RoboCup soccer. In: Proceedings of the Eighteenth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA (2001) 537–544
3. Kok, J.: Uva trilearn website. `http://staff.science.uva.nl/~jellekok/robocup/2004/index_en.html` (2004)
4. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 1st edn. Prentice-Hall Inc., Upper Saddle River, New Jersey (1995)
5. Veloso, M., Stone, P., Bowling, M.: Anticipation as a key for collaboration in a team of agents: A case study in robotic soccer (1999)
6. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. 4th edn. MIT Press, Cambridge, MA (2002)
7. Albus, J.S.: A theory of cerebellar functions. In: Mathematical Biosciences. ??? (1971) 25–61
8. Riedmiller, M., Merke, A.: Using machine learning techniques in complex multi-agent domains. In Stamatescu, I., Menzel, W., Richter, M., Ratsch, U., eds.: Perspectives on Adaptivity and Learning, LNCS, Springer (2002)