Autonomous Agent Learning using an Actor-Critic Algorithm and Behavior Models

(Short Paper)

Victor Uc Cetina
Department of Computer Science
Humboldt University of Berlin
Unter den Linden 6, 10099 Berlin, Germany
cetina@informatik.hu-berlin.de

ABSTRACT

We introduce a Supervised Reinforcement Learning (SRL) algorithm for autonomous learning problems where an agent is required to deal with high dimensional spaces. In our learning algorithm, behavior models learned from a set of examples, are used to dynamically reduce the set of relevant actions at each state of the environment encountered by the agent. Such subsets of actions are used to guide the agent through promising parts of the action space, avoiding the selection of useless actions. The algorithm handles continuous states and actions. Our experimental work with a difficult robot learning task shows clearly how this approach can significantly speed up the learning process and improve the final performance.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*learning*

General Terms

Algorithms, Experimentation

Keywords

reinforcement learning, behavior model, actor-critic

1. INTRODUCTION

The idea of supervision or advice giving was first proposed in 1958 by McCarthy [9]. More recently, Clouse and Utgoff [5] presented an online method of SRL. With this method, a human teacher monitors the agent's progress. If the teacher determines that the agent is not performing well, the teacher takes control and offers advice in the form of an action that is executed at that time. Then, the agent learns from such advice by reinforcing the tendency to choose the action recommended and performed by the teacher. Another attempt to accelerate the learning process was the one proposed by Lin [6]. He introduced a very effective way to speed up the Reinforcement Learning (RL) process through the replaying of experiences. The advice-giver provides complete sequences of states and actions $s_t, a_{t+1}, s_{t+1}, a_{t+2}, \ldots$ that the agent replays internally many times. By doing so backwards, the learning process is further accelerated. Maclin and Shavlik [7, 8] approached the advice giving problem in a different way, by using connectionist Q-learning. The advice-giver watches the learner and occasionally makes suggestions, expressed as instructions in a simple programming language. Using knowledge-based neural networks, those programs are included into the agent's utility function. Later, Rosenstein and Barto [10] proposed a combination of supervised learning with and actor-critic architecture. They used a supervised learning method to include the knowledge provided by a human supervisor into the actor-critic learning process. A composite actor is formed with the actor, a supervisor and a gain scheduler. The action executed at each moment is the result of a linear combination of the actions provided by the actor and the supervisor. Abbeel and Ng [1, 2] studied the use of an expert in order to learn to perform a task in situations where the reward function is not provided or it is difficult to design. The main idea consists of using inverse reinforcement learning to try to obtain the unknown reward function which is supposed to be implicit in the expert's behavior. Their method manages to get performances close to that of the expert. Moreover, Atkeson and Schaal [3] used human demonstrations to have a robot learn to perform a task. First, they learn from the demonstrations a reward function and a task model. Then, based on the learned reward function and task model, they compute a policy. Finally, another work that is worth to mention is that by Carpenter *et al* [4], who approached the problem of how to handle advice from several sources and also how to solve conflicting advices. Due to space constraints we can only mention those approaches more similar to ours.

All of the works mentioned above have one thing in common, the fact that the supervisor provides one action or sequences of actions which should be directly performed by the agent in order to learn from the expert. One exception is the method proposed by Rosenstein and Barto[10], where the action executed is a combination of the action suggested by the supervisor and the action selected by the agent. We use an approach where the action suggested by the supervisor is not directly executed by the agent, not even a modification of it. Instead, the action is used to dynamically generate a reduced set of current relevant actions. We call relevant actions to those actions in the close neighborhood

Cite as: Autonomous Agent Learning using an Actor-Critic Algorithm and Behavior Models (Short Paper), Victor Uc Cetina, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 1353-1356.

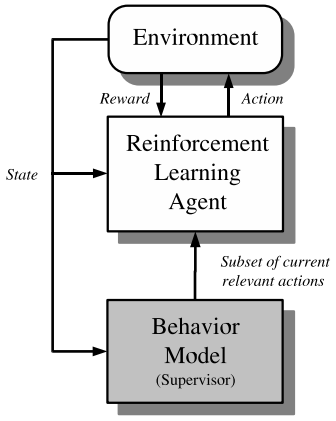

Figure 1: A supervised reinforcement learning architecture that uses a behavior model

of the action suggested by the supervisor. This subset of actions is then passed to the agent which uses it to select the next greedy action. The experiments performed show that our algorithm can significantly reduce the amount of training episodes required to learn a difficult task.

The rest of this paper is organized as follows. In Section 2 we explain the SRL architecture and introduce the algorithm. Section 3 contains the description of our testbed. In Section 4 we present the experimental results. Finally, in Section 5 we conclude the paper and mention our future work.

2. SUPERVISED REINFORCEMENT LEARNING

2.1 Architecture

The SRL architecture, which we have tested previously with Sarsa and Q-learning in [12], focuses on reducing the amount of exploration of the action space, by giving advice to the agent about what actions would be good to try, given the current state of the environment. Figure 1 illustrates the main idea of having the standard RL agent interacting with the environment, meanwhile a behavior model is used to provide the agent with a subset of current relevant actions. Such subset of actions includes the action suggested by the behavior model, and its n closest neighbor actions. Notice that two different actions are considered to be neighbors if they are expected to produce similar results in the environment, when they are applied in similar states.

Two are the key features in this architecture: (1) it allows to considerably reduce the amount of exploration of the action space, and (2) the supervisor does not need to be a perfect teacher with the possession of the optimal policy. Instead, its expertise is used to guide the agent through relevant parts of the action space and not explicitly indicating which action should be performed at each moment. Given that the behavior model can be seen as a Multilayer Perceptron (MLP), or any other Supervised Learning (SL) method, trained with a set of collected examples, it will always provide an action, and therefore the subset of actions can be generated. If the suggested action is wrong, then the reduced action set would probably be also wrong. In those cases, the greedy action selected by the agent could be simply seen as an exploratory action. Of course, our behavior model is expected to be as accurate as possible. Under this condition, the agent will always have much to win and nothing to lose.

2.2 Algorithm

The whole learning process is divided in two phases. In the first phase, an expert is used to generate a set of examples of the form $s_t \rightarrow a_{t+1}$. That is, given the current state of the environment s_t, knowing which is the action a_{t+1} that our expert would perform in the next time step. Using such examples and a SL method we build the behavior model. Once we have built the behavior model, we proceed with the second learning phase.

The second learning phase is shown in Algorithm 1, which is a modified version of the typical actor-critic algorithm described by Sutton and Barto [11]. At each state s the behavior model is used to generate what we call the expert action a_e. Such action a_e is considered to be a near optimal action and we use it to create the set of current relevant actions A_s, where $A_s \subset A$. Such subset A_s is defined by the interval $(a_e - \hat{B}, a_e + \hat{B})$, where $\hat{B}$ specifies how far from the expert action a_e we are willing to explore the action space. By doing so, the agent will always have to select the greedy action from the set of most promising actions, which causes an improvement in the learning rate. The optimal size of $\hat{B}$ grows inversely proportional to the accuracy of our behavior model. In other words, with more accurate behavior models, we need a smaller $\hat{B}$. Notice that the set A_s is used only to choose the greedy action. Random actions are selected from the whole set A. By doing so, we let the agent exploit the knowledge provided by the supervisor as much as possible, at the same time that we allow it to explore the whole action space looking for better actions that are beyond the knowledge of the same supervisor.

The first learning phase can be seen as a straightforward application of SL. Meanwhile, the second learning phase could be implemented using modified versions of any RL algorithm.

3. ROBOT DRIBBLING TASK

In the RoboCup simulation league, one of the most difficult skills that the robots can perform is dribbling. Dribbling can be defined as the skill that allows a player to run on the field while keeping the ball always in its kick range. In order to accomplish this skill, the player must alternate run and kick actions. The run action is performed through the use of the command (**dash** *Power*), while the kick action is performed using the command (**kick** *Power Direction*), where $Power \in [-100, 100]$ and $Direction \in [-180, 180]$. Such commands, belong to the set of basic commands provided by the simulator.

There are three factors that make this skill a difficult one to accomplish. First, the simulator adds noise to the movement of objects, and to the parameters of commands. This is done to simulate a noisy environment and make the competition more challenging. Second, since the ball must remain close to the robot without collisioning with it, and at the same time it must be kept in the kick range, the margin for error is small. And third, the most challenging factor, the use of heterogeneous players during competitions. Using heterogeneous players means that for each game the simulator generates seven different player types at startup, and

Algorithm 1: Supervised Actor-Critic Algorithm

initialize the weights vectors of the *Actor* and *Critic* arbitrarily
foreach *training episode* **do**
  initialize s
  take suggested action a_e from *Behavior Model*
  generate set $(a_e - \hat{B}, a_e + \hat{B})$
  take greedy action $a \in (a_e - \hat{B}, a_e + \hat{B})$
  with probability ϵ choose random action $a \in A$
  repeat for each step of episode
    perform action a, observe r, s'
    $TDError \leftarrow r + \gamma Critic(s') - Critic(s)$
    $TargetValue \leftarrow Critic(s) + \alpha TDError$
    train *Critic* with example $(s, TargetValue)$
    if $TDError > 0$ **then**
      train *Actor* with example (s, a)
    end
    take suggested action a'_e from *Behavior Model*
    generate set $(a'_e - \hat{B}, a'_e + \hat{B})$
    take greedy action $a' \in (a'_e - \hat{B}, a'_e + \hat{B})$
    with probability ϵ choose random action $a' \in A$
    $s \leftarrow s'$, $a \leftarrow a'$
  until s *is terminal*
end

the eleven players of each team are selected from this set of seven types. Given that each player type has different "physical" capacities, an optimal policy learned with one type of player is simply suboptimal when followed by another player of different type. In theory, the number of player types is infinite.

Due to these three reasons, a good performance in the dribbling skill is very difficult to obtain. Up today, even the best teams perform only a reduced number of dribbling sequences during a game. Most of the time the ball is simply passed from one player to another.

4. EXPERIMENTS AND RESULTS

For the first learning phase, we constructed our dribbling behavior model based on the Wright Eagle team, which is a RoboCup team with highly developed skills. We collected the information of 500 games and with the help of some scripts, we extracted the sequences of the games where a player managed to dribble for at least 3 meters. Once that we gathered the examples, we filtered them using an application developed specifically to identify and eliminate incorrect examples. The final set of 18,000 examples were used to train 2 multilayer perceptrons. One MLP learned to predict the dash power and the other the kick power. The input of both MLPs is the current state, seen as a 12-dimensional vector. This vector consists of the following variables: *player decay, dash power rate, kickable margin, kick rand, ball position x - player position x, ball position y - player position y, ball velocity x - player velocity x, ball velocity y - player velocity y, ball velocity x, ball velocity y, player velocity x, player velocity y.* The first 4 variables are some of the parameters that define a type of player, and for this problem, they were the most useful during our experimentation. The other 8 variables are needed to specify the current physical state of the ball and player. The output of the MLPs are

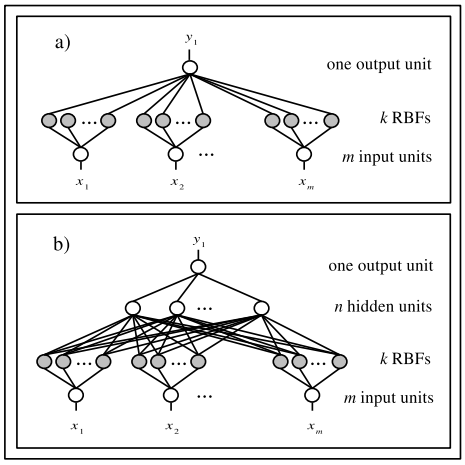

Figure 2: Different structures used to approximate the value function. (a) Radial basis functions. (b) Multilayer perceptron with one layer of radial basis functions

the dash power and kick power respectively, and together formed the behavior model. These MLPs predict the power of dashes and kicks with an error of 15 units. This error is big enough to prevent us from using those MLPs to directly control our agents. However, the knowledge encapsulated in them proved to be very useful when used as a supervisory source of information.

For the second learning phase, we implemented a RL agent that perceives the current state of the environment using the same input vector used by the behavior model. Each training episode was initiated placing the player in the center of the field with the ball besides it, at a distance of 0.5 meters, both with velocity zero. The training episodes were terminated either when the robot kicked the ball away from its kick margin, or when 35 actions were performed. The reward function gives always the scalar value resulting from the calculation of: 0.25(*player position x + ball position x + player velocity x + ball velocity x*). There is also a punishment of -100 everytime the player loses the ball or when there is a collision with it. The learning parameters were: $\epsilon = 0.3$, $\alpha = 0.01$ and $\gamma = 0.5$. A key design point when we work with reinforcement learning in continuous spaces is the structure used to approximate the value function. In our experimental work we employed two different structures: (1) an array of radial basis functions, and (2) a multilayer perceptron enhanced with one layer of radial basis functions. Such structures which are a linear and a non-linear function approximator respectively, are illustrated in Fig. 2.

Figure 3 shows the learning curves of the actor-critic algorithm using radial basis functions to approximate the value function, and 2 different implementations of the SRL algorithm, for different sizes of the relevant actions set A_s. The curves represent moving averages of size 1,000 that were averaged over 10 runs. From these results we can see that the supervised actor-critic method is clearly superior to the pure actor-critic version, when we use $\hat{B} = 15$. However when we use $\hat{B} = 10$, the resulting learning curve is worse than that obtained with the pure actor-critic algorithm. The reason for this is simple. We are reducing the exploration space much more than we should do, given the accuracy

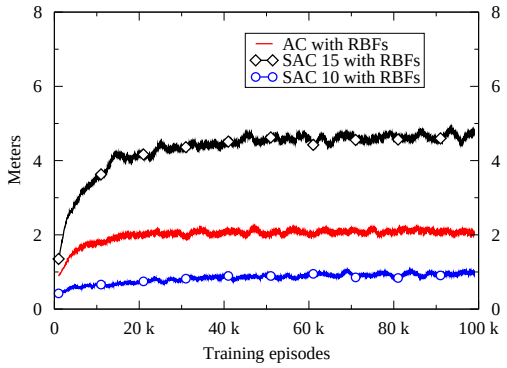

Figure 3: Learning curves of the Actor-Critic and the Supervised Actor-Critic algorithms using radial basis functions

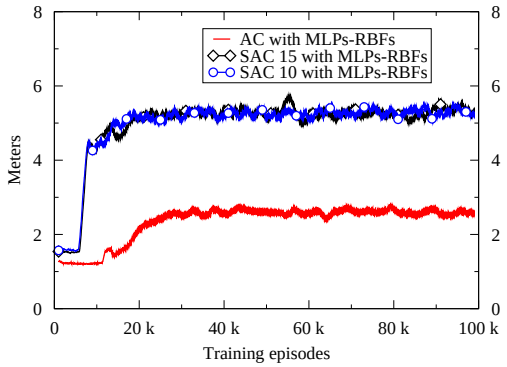

Figure 4: Learning curves of the Actor-Critic and the Supervised Actor-Critic algorithms using multilayer perceptrons with one layer of radial basis functions

of our behavior model. The typical actor-critic algorithm has a performance of 2 meters, after training for 100,000 episodes. Meanwhile, the best SRL algorithm, has a performance slightly under 5 meters.

In Fig. 4 we can see the learning curves of the same three algorithms, but using instead the non-linear value function. It is clear that the supervised actor-critic algorithm has also a much better performance than the typical actor-critic algorithm. Besides, we can see that the result with the simple actor-critic method is slightly better than that obtained with the linear value function in Fig. 3. We can also see that the performance of the supervised algorithm with $\hat{B} = 15$ is better than that obtained using a linear value function. Finally, when we check the performance of the supervised algorithm with $\hat{B} = 10$, something interesting occurs, the learning curve is identical to the curve obtained with the non-linear value function and $\hat{B} = 15$. In this case, the learning rate was not affected by the reduction of $\hat{B}$, as it happened when we used the linear function approximator. This difference is due to a better ability of the non-linear function approximator to generalize, which makes it more robust to changes in $\hat{B}$ than a linear function approximator.

5. CONCLUSION AND FUTURE WORK

We have presented one algorithm to implement supervised actor-critic learning. In our algorithm, behavior models previously learned from examples, are used to dynamically generate subsets of relevant actions at each moment. Using these subsets of actions, the agent can accelerate its learning rate. Performances obtained after 100,000 training episodes are better when we use the supervised version of the actor-critic algorithm, being more robust when the non-linear function approximator is used to represent the value function. We tested our algorithms with the robot dribbling problem, in the framework of the RoboCup simulation league. Such problem involves continuous state and action spaces with high dimensionality. Our future work will consider the use of eligibility traces and options, as a way to improve the final performances.

Acknowledgements This research work was supported by a PROMEP scholarship from the Education Secretariat of Mexico (SEP), and Universidad Autónoma de Yucatán.

6. REFERENCES

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, 2004.

[2] P. Abbeel and A. Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the 22nd International Conference on Machine Learning*, 2005.

[3] C. Atkeson and S. Schaal. Robot learning from demonstration. In *Proceedings of the Fourteenth International Conference on Machine Learning*, 1997.

[4] P. V. M. Carpenter P., Riley and G. Kaminka. Integration of advice in an action-selection architecture. *RoboCup 2002: Robot Soccer World Cup VI. Lecture Notes in Computer Science*, 2003.

[5] J. A. Clouse and P. E. Utgoff. A teaching method for reinforcement learning. In *Proceedings of the Ninth International Workshop on Machine Learning*, 1992.

[6] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, (8):293–321, 1992.

[7] R. Maclin and J. W. Shavlik. Incorporating advice into agents that learn from reinforcements. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994.

[8] R. Maclin and J. W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, (22):251–282, 1996.

[9] J. McCarthy. Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, 1958.

[10] M. T. Rosenstein and A. G. Barto. Supervised actor-critic reinforcement learning. In *Learning and Approximate Dynamic Programming: Scaling Up to the Real World*. John Wiley & Sons, 2004.

[11] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[12] V. Uc-Cetina. Supervised reinforcement learning using behavior models. In *Proceedings of the 6th International Conference on Machine Learning and Applications*, 2007.