Supervised Reinforcement Learning in Discrete Environment Domains

Boris Jensen, Daniel Ortiz-Arroyo
Computational Intelligence and Security Lab
Department of Electronic Systems
Aalborg University, Denmark
Email: do@es.aau.dk

Nareli Cruz-Cortés
Center for Computing Research
National Politechnique Institute, Mexico
Email: nareli@cic.ipn.mx

Francisco Rodríguez-Henríquez
Computer Science Department
CINVESTAV-IPN, Mexico
Email: francisco@cs.cinvestav.mx

***Abstract*—This paper describes a supervised reinforcement learning-based model for discrete environment domains. The model was tested within the domain of backgammon game. Our results show that a supervised actor-critic based learning model is capable of improving the initial performance and then eventually reach similar performance levels as those obtained by TD-Gammon, an artificial neural network player (ANN) trained by temporal differences.**

***Keywords*-machine learning; reinforcement learning; actor-critic; automata player**

I. Introduction

In reinforcement learning an agent learns by interacting with its environment. The environment responds by changing its state and rewarding the agent with a scalar signal. The agent performs actions whose goal is to maximize the cumulative reward over time. This learning model has been applied successfully in a wide variety of different domains such as elevator dispatching [1] and backgammon [2] to name just a few. In the case of backgammon, Gerald Tesauro created TD-Gammon [2], a computer program built using a reinforcement learning technique called temporal-difference. TD-Gammon's learning mechanism updates a value function that represents the expected cumulative reward, according to the following equation:

$$V(s_t) = V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \quad (1)$$

where r_{t+1} is a reward signal at time $t+1$, and $V(s_t)$, $V(s_{t+1})$ are value functions of environmental state at time t and $t+1$, γ is the discounting rate and $\alpha \in]0,1]$ is the learning rate. This equation is called the TD(0) algorithm. The formula can be understood intuitively by realizing that $V(s_t)$ and $r_{t+1} + \gamma V(s_{t+1})$ are both estimates of the value function $V(s_t)$ at time t. The only difference is, that while $V(s_t)$ is based on an estimate of all the future rewards, $r_{t+1} + \gamma V(s_{t+1})$ includes knowledge only of the first of these rewards. The update of the value function by an amount (positively) proportional to the difference $(r_{t+1} + \gamma V(s_{t+1})) - V(s_t)$ is called the temporal difference error. The value of V at $s = s_t$ is adjusted in the direction of $r_{t+1} + \gamma V(s_{t+1})$, which is therefore called the target of the update.

The reward signal is a probabilistic function of the states s_t, s_{t+1} and action a_t. s_{t+1} itself is a probabilistic function of previous state s_t and action a_t, therefore rewards received could be very atypical. Updating the value function according to equation 1 will cause $V(s_t)$ to converge to V^π (value function of policy π) in the mean only if the learning rate α is sufficiently small, and with probability 1 only if α decreases with time [3].

Remarkably, TD-Gammon learned good game strategies after playing millions of training games against itself, eventually reaching a level comparable to human master players. Results of this approach were so successful that some of its tactics have been adopted by human players. Interestingly, recent research has shown that the human brain has a similar mechanism in which dopamine, a neurotransmitter, is used as the error signal [4].

In spite of its success reinforcement learning still has some drawbacks. One of its disadvantages is the number of episodes it takes to reach an acceptable level of performance. For instance, the version of TD-Gammon that reached master level was trained by playing 1.5 million games against itself [5]. Furthermore, applying reinforcement learning may not be feasible in some domains since the consequences of not performing initially at certain minimum level may be undesirable, for instance, when robots learn by interacting in real time with their environment. To overcome these limitations a new learning scheme called *supervised reinforcement learning* was proposed in [6]. In supervised reinforcement learning, simple heuristics act as a *supervisor* of a *learner*. The supervisor guides the learner until it reaches an acceptable level of performance. From there the learner takes control and continues learning on its own. This strategy retains the good features of reinforcement learning but prevents the learner from making many mistakes in the beginning when it has null knowledge about its environment. Supervised reinforcement learning techniques were proposed originally for continuous environment domains [7] [6].

This paper describes a supervised reinforcement learning model that can be applied in discrete environment domains. Concretely, we have adapted the actor-critic model [7] to work within the domain of backgammon game. We have tested our method comparing its performance with a baseline

978-1-4244-7375-5/10/$26.00 2010

TD-Gammon player. The paper is organized as follows. Section II provides a brief overview of related work. Section III describes our approach with some detail. Section IV presents some preliminary results and section V presents our conclusions.

II. Related Work

The fusion between the fields of supervised and reinforcement learning was proposed by Utgoff and Clouse in [6][1]. The authors made the observation, that two fundamental sources of training information exist: *future payoff* achieved by taking actions according to a given policy from a given state, and the *advice* from an expert regarding which action to take, next. Training methods that rely on the future payoff are called *temporal difference* methods, while methods relying on expert advice are called *state preference* methods. In state preference methods the goal of the learner is to have the same preference as the expert when presented with a set of possible states. Thus, the only thing that matters, is that the sign of the slope of the learner's evaluation function between two states be the same as that of the expert. This means, that generally infinitely many functions exist that produce the same control decisions as the expert evaluation function, making state preference methods very flexible with regard to incorporating other types of learning information.

Utgoff and Clouse observed that temporal difference methods and state preference methods are orthogonal. Using the general model of an evaluation function as a parameterized function of the state, state preference methods attempt to change the parameters of an evaluation function of the possible next states to obtain the same slope as the expert. Temporal difference methods, on the other hand, are concerned with obtaining the correct value for the sequence of actions experienced by following the current policy. A state preference and a temporal difference method will be in conflict to the degree, that the expert is fallible. Therefore a rule is needed to determine how much to trust the expert. Another issue is that the expert may not be always available. Utgoff and Clouse implemented a heuristic where the learner only asks for supervisor input when the state is poorly modeled. This happens when most of the parameters of the utility function are updated as a result of the calculation of the temporal differences.

In 2004, Rosenstein and Barto [7] adapted the actor-critic model [3] for supervised reinforcement learning. The actor-critic model is a general model for reinforcement learning, where an actor makes decisions about which actions to take, while the critic learns a utility function of the states by reinforcement learning, and critizes the actor on the actions it chooses based on this utility function. This causes the actor to update its policy. One of the advantages of the actor-critic

[1]Benbrahim [8] also proposed combining the two ideas around the same time period

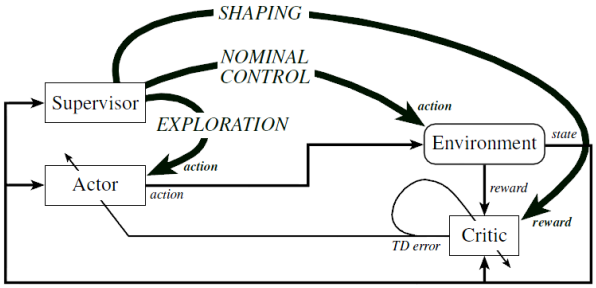

Figure 1. Supervised reinforcement learning using an actor-critic architecture [7]

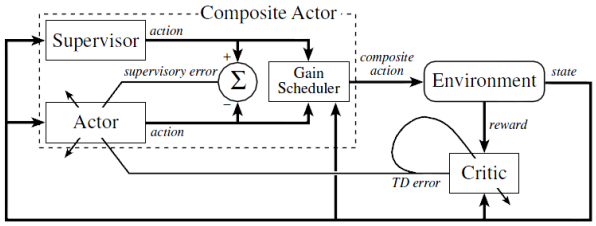

Figure 2. Action interpolation between actor and supervisor [7]

model, is that it separates decisions of which action to take, from the task of constructing the correct utility function. Based on this separation, Rosenstein and Barto identified 3 ways, in which a supervisor can influence the actions of a reinforcement learner. Figure 1 shows the 3 ways, which are: *value function shaping* for the critic, *exploratory advice* for the actor and *direct control*, in which the supervisor chooses the actions.

Rosenstein and Barto model is a flexible framework that interpolates between the actor and the supervisor. The general structure is shown in Figure 2. The output of both the actor and supervisor is a one-dimensional continuous scalar variable. These signals are fed into the gain scheduler unit, which computes the final output as a weighted average of the actor and supervisor's outputs. The weights can be obtained by an interpolation controlled by the actor that makes it seek explorative advice from the supervisor, or determined by the supervisor that uses them to ensure a minimum level of performance.

This model of learning is called *supervised actor-critic reinforcement learning*. The critic implements an ordinary state-value function, using TD(0), i.e. it computes the TD error $\delta = r_{t+1} + \gamma V(s_{t+1} - v(s_t))$ to update both its own state-value estimates, as well as the actor's policy.

The gain scheduler combines the actions of the actor and the supervisor according to an *interpolation parameter* k that determines the level of control (or autonomy) of the actor. Since actions are assumed to be scalars, the composite action is just a weighted sum of the two actions:

$$a = ka^E + (1-k)a^S \tag{2}$$

where a^E is the *actor's exploratory* action, and a^S is the *supervisor's* action, according to policies π^E and π^S, respectively. The actor has also a *greedy* policy π^A. The exploratory policy is just this greedy policy with an added gaussian noise with zero mean. The k value plays an important role in choosing the action and in adjusting the policy of the actor according to the reward received. Assuming that π^A is a parameterized function with parameter vector w, the equations for updating the actor's policy are:

$$w \leftarrow w + k\Delta w^{RL} + (1-k)\Delta w^{SL} \quad (3)$$

$$\Delta w^{RL} = \alpha\delta(a^E - a^A)\nabla_w \pi^A(s) \quad (4)$$

$$\Delta w^{SL} = \alpha(a^S - a^A)\nabla_w \pi^A(s) \quad (5)$$

where $\Delta w^{RL}, \Delta w^{SL}$ are the individual updates of reinforcement learning and supervised learning, respectively in Equation 3 [7]. k is used to interpolate between these two types of learning. Equation 4 is the actor's update function where α is the learning rate, and δ is the TD error from the critic. The effect of Equation 4 is either to move $\pi^A(s)$ closer to $\pi^E(s)$, when the reward for the exploratory action was higher than expected, leading to a positive TD error, or further away from it, when the TD error is negative. Equation 5 is a gradient decent rule for supervised learning. The effect is to move $\pi^A(s)$ closer to $\pi^S(s)$, regardless of the reward received.

III. Adapting the Actor-Critic Model to Discrete Environments

The supervised actor-critic model was designed for continuous domains and assumes that actions are continuous scalar values. Hence, to work in a discrete environment such as backgammon game it must be adapted. For instance, given that actions in backgammon are discrete moves, it is not possible to use a gain scheduler to produce a weighted sum of the supervisor and actor's actions, neither is possible to subtract actions nor creating an exploratory policy that is a noisy version of the greedy policy.

To adapt the model we need to find action interpolation and policy update rules that could emulate the way the original actor-critic model works. Emulating action interpolation in discrete actions was proposed in [7]. The idea is to interpret the k value as the probability for the gain scheduler to choose actor's action, instead of supervisor's action. This is a reasonable discrete approximation of the smooth mixing of the two actions given that the intent of equations 4 and 5 is to change the policy according to experience and according to the advice from the critic. Given that the policy of a reinforcement learning backgammon agent is implemented by the state-value function, the above intent translates into moving the value function in the direction of the received reward and also in the direction of the supervisor's value function.

This puts a restriction on the supervisor's model: instead of just emitting actions based on an unknown policy, implemented in an unknown way, the supervisor is now required to expose a state evaluation format, that should be the same as that of the actor - a vector of 4 doubles representing the probabilities of the different outcomes of the game. Given this fact, one possible implementation for the supervisor is to create a version of TD-Gammon that stops its training early. Originally the chosen supervisor was trained for 34500 games, after which it won approximately 34% of the games against Pubeval, a standard benchmark program.

In the following description we will refer to the combination of supervisor and actor-critic as an agent. The agent implements the state-value function as a neural network, so the update rule described by Eq. 3 is not in its most natural form for this kind of network. It is more natural to define a target output value for the state and then let the backpropagation mechanism take care of the rest. Interpreting the intent of the updates in the supervised actor-critic model in equation 3 we propose using a *combined target* (ct_t) output for the value function update of our agent calculated as:

$$ct_t = k(s_t)(r_{t+1} + V(s_{t+1})) + (1 - k(s_t))(o_{sup}(s_t)) \quad (6)$$

$$= k(s_t)V(s_{t+1}) + (1 - k(s_t))(o_{sup}(s_t)) \quad (7)$$

where o_{sup} is the output of the supervisor. In the backgammon domain, the reward is zero for all state transitions except the last one. This fact is modeled by removing in Equation 6 the reward term, and letting $V(s_{T+1})$ (where T is the time of the final state seen by the agent) be equal to one of four reward binary vectors, depending on the result of a game. An encoding of 1000 was used if the agent wins normally, 0100 if it wins by gammon, 0010 if lost normally or 0001 if lost a gammon[2]. For faster learning, training only started, once a game had finished, and updates to the estimated value of a state were done when the last state in the game was reached. In this way, some of the reward obtained for a game propagated down to the first state seen already during the training immediately after that game.

A. K-values

In [7], the supervised actor-critic model required implementing a function that would provide a state-dependent value of k, to be used for the actor policy update and the gain scheduler action interpolation. This function had two update requirements:

- Visiting a state should raise the value of k for that state.
- Not visiting a state for some time, should cause the k-value for that state to slowly drop.

The intuition behind the first requirement is that visits to a state increase the knowledge of the agent about that

[2]A gammon happens after finishing a game when a losing player was not able to remove any of its checkers from the board

state, so decisions made about that state can be trusted more in the future. The second requirement arises because the actor policy was implemented as a parameterized function. Given that in these functions the space of parameter values determines performance, not visiting a state for some time will generally have the effect of increasing the expected error of the policy over the observed samples, when compared to an optimal policy. Therefore, in these cases the supervisor should be trusted more.

The previous requirements for a state-dependent function of k were fulfilled in [7] using a variation of the tile encoding technique described in [3]. The test used 25 tilings over a 2-dimensional input space, but the weights associated with each tile were not updated according to any gradient descent method. Instead, the weights of visited tiles were increased by a small amount, and after each episode, all weights were multiplied by a factor of 0.999.

Unfortunately, tile coding schemes can not be used in our case given the large size of the state space of backgammon. The state representation for a backgammon board, used by both TD-Gammon and the agent is a 196-dimensional vector. Following sections describe some other methods we applied in our experiments.

B. Kanerva Coding

Tile coding of the k function worked well in [7] because there is a very simple relationship between the weights and the output that makes easy to implement a gradually decreasing k-value for states that had not been visited for some time. As discussed in [3], the same simple relationship among weights and output is also present in the Kanerva coding method. Hence, we implemented a k function interpolation based on Kanerva coding.

Designing the Kanerva-based k-function for our agent required determining the prototype space, how the prototypes should be generated, and how many prototypes are required. The prototype space is the 196-dimensional vector, corresponding to the input space for the agent. The Hamming distance could be used as metric. However, since the goal is simply to distinguish among the different states, then a more compact state representation that encodes every point in only 3 bits can be used. This encoding is shown in table I[3]. Since Kanerva coding scheme requires finding the distance between state representations, this compact encoding provided faster computations.

When the different dimensions are independent, prototypes can be generated by generating a random value (within the set of legal values) for each dimension. Unfortunately, this approach cannot be used in backgammon because the sets of legal values are not independent of each other. For instance, within the 24 dimensions corresponding to the points of the backgammon board, there cannot be 4 values

[3]Note that every player has 15 checkers in the beginning of the game

encoding	interpretation
000	0 checkers
001	1 opponent checker
010	2 opponent checkers
011	3 or more opponent checkers
100	1 agent checker
101	2 agent checkers
110	3 agent checkers
111	4 or more agent checkers

Table I
KANERVA PROTOTYPE ENCODING

of code 111 from table I, as that would indicate a board with 16 or more agent's checkers, which would be illegal. To overcome this problem, a series of games with 2 platers was conducted, where they were allowed to choose their moves randomly. Using this approach, any sample state from any one of these games is a legal board state that was translated into a prototype. The set of prototypes should cover many different states, i.e. the prototypes should not be too similar. This means that the samples should not come from early parts of the game, where the states will be likely similar for all games. Similarly, samples should not come from consecutive positions in the game. Since random games tend in the end toward the same type of positions (with most of the checkers in the opponent's home table), samples should not come from late stages of the game. Using these constraints, we chose every eighth of the first 64 board states in a game to produce prototypes.

To determine how many prototypes to generate, it is necessary to consider the tradeoff between an increased resolution achieved using a large number of prototypes and the reduced computational load from using a smaller numbers of prototypes. The time spent computing a k-value increases linearly with the number of prototypes, so a large number of prototypes would make playing a single game very slow. In the end, samples were taken from 1000 games, since that resulted in a reasonable game speed. This policy produced a rather small number of prototypes compared to the size of the full prototype space. However, each prototype must be able to generalize to many other states in order to be useful. To achieve a high degree of generalization, the set of prototypes activated by a state s is assumed to be the top 5% most similar prototypes, where similarity is the number of dimensions with the same encoding.

C. Artificial Neural Networks

The k function can be also implemented using ANNs. A simple way to increase the value of visited states could be to provide a target of 1, and then use backpropagation. However, decreasing the state value for states that have not been visited for a while may be difficult to implement with ANN. One naive way to implement this requirement would be to go over all non-visited states and train them with a target of 0. However, given that the number of unvisited

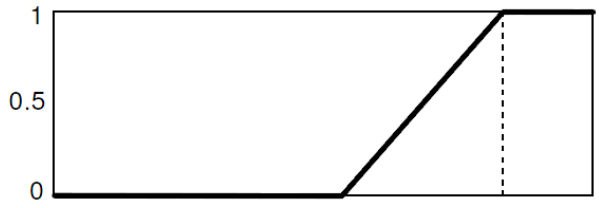

Figure 3. A trapezoid function

states in a simple game is extremely large, this is not a feasible solution.

If none of the infrequently visited states are encountered during a normal game (which by definition is unlikely to happen), we could implement the k-value by initializing a ANN to produce a value close to 0 for all inputs, and then training the ANN for visited states on a target of 1. The network can be initialized to produce (close to) 0 output for all states by setting all connection weights to 0 and all neuron biases to -20 for instance.

D. State Independent Interpolation

If we drop the requirement that the k-value should decrease for infrequently visited states, then the values for all states will rise toward 1 with varying speeds. For the set of states frequently visited during a normal game, those speeds may be roughly similar. This suggests a further simplification i.e. to simply disregard the specific state, and only vary the k-value as a non-decreasing function of the number of training episodes. A simple implementation could be a trapezoid function, that is zero for all training episodes below some threshold, rises linearly to 1 up to reaching some other threshold, and stays at 1 for the rest of the training episodes. This function is shown in Figure 3.

To implement the trapezoid function we need to determine the left and right thresholds values. Looking at the initial performance of the baseline TD-Gammon player, it seems that the actor should be in full control only when its performance reaches that of the supervisor. Otherwise, the supervisor may actually hurt the combined performance if it keeps in control passing this threshold. The performance of the baseline TD-Gammon shows that to apply this strategy the right threshold should be roughly 5000 training episodes. For the left threshold, it is not completely clear, whether it is better to leave the supervisor in total control for a while, or whether control should start to transfer to the actor immediately.

In summary our modified supervised critic-actor learning mechanism for backgammon was implemented in the following way:

- The actor was implemented as a neural network.
- The combined action is either the actor or the supervisor's actions, which are partially dependent on k.
- The actor's neural network is trained on a weighted sum of the reinforcement target and the supervisor target, with the weights dependent on k.
- Three different interpolation functions for k were tested in our experiments: Kanerva, ANN and trapezoid function.

The next section presents some of the preliminary results of our approach.

IV. Preliminary Results

The performance results of our agent using 2 different interpolation functions (ANN and Kanerva) is shown in Figure 4. Figure 5 shows performance results when a trapezoid interpolation function is used. The percentage of winning games in each figure is obtained by making our agent (or the baseline TD-Gammon player) play against Pubeval, a standard benchmark program publicly available. The performance of the agent is measured every 1000 games. The figures show that, as expected, the agent starts winning around 34% of the games against pubeval as the supervisor is in full control. The figures also show that agents implemented using an interpolation function for k based on either Kanerva coding or ANN are not able to improve upon the performance of the baseline TD-Gammon. These results seem to indicate that the interpolation must be heavily favoring the lower performance supervisor throughout the training. There are a couple of reasons why this might be so. For the ANN based interpolation, during training the gradient for the sigmoid activation function evaluated at -20 is very small which causes that weight increments were also too small to have any effect during the course of the 100000 training episodes.

For the Kanerva-based interpolation one possible explanation for its low performance could be that the number of prototypes (approximately 8000), were not sufficient to capture the complexity of the state space in the backgammon game. Therefore, with the decision of always activating the closest 5% of the prototypes, such prototypes would be randomly activated. With the random activation, the weight increases due to state visits would spread evenly among all the protypes. Since none of the prototypes would consistently receive weight increases, the weight decreasing factor of 0.999, that was applied to all weights, might have been enough to keep all of the prototype weights at a low enough level, in such a way that the supervisor dominated the interpolation.

In contrast to both the Kanerva-based and neural network-based interpolation, the simple trapezoid interpolation worked reasonably well. Figure 5 shows the performance of our agent with trapezoid interpolations with left thresholds of 0 and right thresholds of 5000 and 50000. In both cases, our agent was able to avoid the initial bad performance of the baseline, although in the case of using a 5000 threshold the performance did actually fall a bit before the combined

978-1-4244-7375-5/10/$26.00 2010 ©IEEE

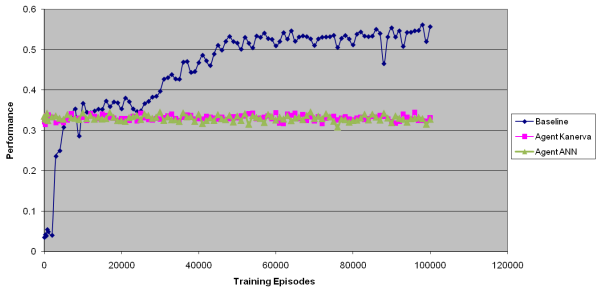

Figure 4. Performance of supervised reinforcement learning using Kanerva-based interpolation and ANN

learning mode within the agent really started to take off. Another problem shown in Figure 5 is that the baseline TD-gammon surpassed our agent's performance at around 45000 training episodes. The effect of choosing a different interpolation trapezoid function with a right threshold of 50000 is also shown on Figure 5. This new interpolation ensures, that the initial drop in performance seen when we use a shorter right threshold in the trapezoid interpolation does not occur. However, in this case the baseline surpassed our agent's performance much earlier and the agent stayed at the supervisor's performance level much longer.

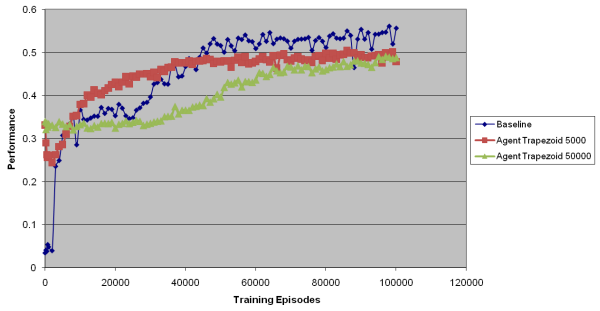

Figure 5. Performance of supervised reinforcement learning using trapezoid interpolation, left threshold 0, right threshold 5000 and 50000

We also tested our agent using a better supervisor, with more expertise. The new supervisor was trained by playing 45000 games. The results in Figure 6 shows that in this case the supervised actor-critic model was indeed capable of reaching similar performance levels as the baseline TD-Gammon and had better initial performance starting winning around 45% of the games.

V. Conclusions

This paper has presented a supervised reinforcement learning method for discrete environments that is capable of reaching similar levels of performance as a baseline TD-GAmmon learner, without having its initial bad performance. We tested our method within the backgammon domain. We experimented using different interpolation strategies from

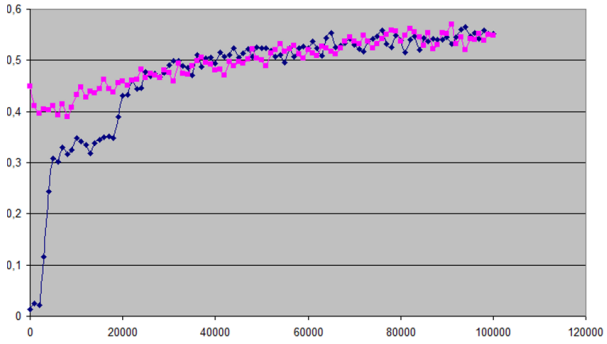

Figure 6. Performance of supervised reinforcement learning using a better supervisor and trapezoid interpolation, left threshold 0, right threshold 5000

which a simple state interpolation ramp function has shown to obtain the best performance. Our results also show the importance of two issues: having a supervisor with enough expertise and increasing learner autonomy at the right pace. If the learner has control too soon, the performance may drop initially since the supervisor has not had enough time to teach. If the supervisor remains in control for too long, performance will suffer in the longer term, since the learner is not allowed to learn from experience, and thereby will be incapable of surpassing the performance of its supervisor.

References

[1] R. Crites and A. Barto, "Improving elevator performance using reinforcement learning," in *Advances in Neural Information Processing Systems 8*. MIT Press, 1996, pp. 1017–1023.

[2] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998. [Online]. Available: http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html

[4] A. Smith, M. Li, S. Becker, and S. Kapur, "Dopamine, prediction error and associative learning: a model-based account." *Network*, vol. 17, no. 1, pp. 61–84, 2006. [Online]. Available: http://dx.doi.org/10.1080/09548980500361624

[5] G. Tesauro, "Programming backgammon using self-teaching neural nets," *Artif. Intell.*, vol. 134, no. 1-2, pp. 181–199, 2002.

[6] P. E. Utgoff and J. A. Clouse, "Two kinds of training information for evaluation function learning," in *In Proceedings of the Ninth Annual Conference on Artificial Intelligence*. Morgan Kaufmann, 1991, pp. 596–600.

[7] 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.

[8] H. Benbrahim, "Biped dynamic walking using reinforcement learning," Ph.D. dissertation, Durham, NH, USA, 1996, director-Miller,III, W. Thomas.