

Residual Advantage Learning Applied to a Differential Game

Mance E. Harmon
Wright Laboratory
WL/AAAT Bldg. 635 2185 Avionics
Circle
Wright-Patterson Air Force Base, OH
45433-7301
harmonme@aa.wpafb.mil

Leemon C. Baird III
U.S.A.F. Academy
2354 Fairchild Dr. Suite 6K41, USAFA,
CO 80840-6234
baird@cs.usafa.af.mil

Abstract

An application of reinforcement learning to a differential game is presented. The reinforcement learning system uses a recently developed algorithm, the residual form of advantage learning. The game is a Markov decision process (MDP) with continuous states and nonlinear dynamics. The game consists of two players, a missile and a plane; the missile pursues the plane and the plane evades the missile. On each time step each player chooses one of two possible actions; turn left or turn right 90 degrees. Reinforcement is given only when the missile hits the plane or the plane reaches an escape distance from the missile. The advantage function is stored in a single-hidden-layer sigmoidal network. The reinforcement learning algorithm for optimal control is modified for differential games in order to find the minimax point, rather than the maximum. As far as we know, this is the first time that a reinforcement learning algorithm with guaranteed convergence for general function approximation systems has been demonstrated to work with a general neural network.

1. INTRODUCTION

In Harmon, Baird, and Klopf (1995) it was demonstrated that the residual gradient form of the advantage updating algorithm could learn the optimal policy for a linear-quadratic differential game using a quadratic function approximation system. We propose a simpler algorithm, *advantage learning*, which retains the properties of advantage updating but requires only one function to be learned rather than two. A faster class of algorithms, residual algorithms, is proposed in (Baird, 95). We present here, for the first time, empirical results demonstrating the residual form of Advantage Learning solving a non-linear game using a general neural network. The game is a Markov decision process (MDP) with continuous states and non-linear dynamics. The game consists of two players, a missile and a plane; the missile pursues the plane and the plane evades the missile. On each time step each player chooses one of two possible actions; turn left or turn right 90 degrees. Reinforcement is given only when the missile either hits or misses the plane. The advantage function is stored in a single-hidden-layer sigmoidal network. The reinforcement learning algorithm for optimal control is modified for differential games in order to find the minimax point, rather than the maximum. This is the first time that a reinforcement learning algorithm with guaranteed convergence for general function approximation systems has been demonstrated to work with a general neural network.

2. BACKGROUND

2.1. Advantage Updating

The advantage updating algorithm (Baird, 1993) is a reinforcement learning algorithm in which two types of information are stored. For each state x , the value $V(x)$ is stored, representing an estimate of the total discounted return expected when starting in state x and performing optimal actions. For each state x and action u , the *advantage*, $A(x,u)$, is stored, representing an estimate of the degree to which the expected total discounted reinforcement is increased by performing action u rather than the action currently considered best. The optimal value function $V^*(x)$ represents the true value of each state. The optimal advantage function $A^*(x,u)$ will be zero if u is the optimal action (because u confers no advantage relative to itself) and $A^*(x,u)$ will be negative for any suboptimal u (because a suboptimal action has a negative advantage relative to the best action). Advantage updating has been shown to learn faster than Q-learning (Watkins, 1989), especially for continuous-time problems (Baird, 1993, Harmon, Baird, & Klopff, 1995).

2.2. Advantage Learning

Advantage learning improves on advantage updating by requiring only a single function to be stored, the advantage function $A(x,u)$. Furthermore, advantage updating requires two types of updates (learning and normalization updates), while advantage learning requires only a single type of update (the learning update). For each state-action pair (x,u) , the advantage $A(x,u)$ is stored, representing the utility (advantage) of performing action u rather than the action currently considered best. The optimal advantage function $A^*(x,u)$ represents the true advantage of each state-action pair. The value of a state is defined as:

$$V^*(x) = \max_u A^*(x,u) \quad (1)$$

The advantage $A^*(x,u)$ for state x and action u is defined to be:

$$A^*(x,u) = V^*(x) + \frac{\langle R + \gamma^{\Delta t} V^*(x') \rangle - V^*(x)}{\Delta t K} \quad (2)$$

where $\gamma^{\Delta t}$ is the discount factor per time step, K is a time unit scaling factor, and $\langle \rangle$ represents the expected value over all possible results of performing action u in state x to receive immediate reinforcement R and to transition to a new state x' . Under this definition, an advantage can be thought of as the sum of the value of the state plus the expected rate at which performing u increases the total discounted reinforcement. For optimal actions the second term is zero; for suboptimal actions the second term is negative.

3. REINFORCEMENT LEARNING WITH CONTINUOUS STATES

3.1. Direct Algorithms

For predicting the outcome of a Markov chain (a degenerate MDP for which there is only one possible action), an obvious algorithm is an incremental form of value iteration, which is defined as:

$$V(x) \leftarrow (1 - \alpha)V(x) + \alpha[R + \mathcal{W}(x')] \quad (3)$$

If $V(x)$ is represented by a function-approximation system other than a look-up table, update (3) can be implemented directly by combining it with the backpropagation algorithm (Rumelhart, Hinton, & Williams, 86). For an input x , the actual output of the function-approximation system would be $V(x)$, the “desired output” used for training would be $R + \gamma V(x')$, and all of the weights would be adjusted through gradient descent to make the actual output closer to the desired output. Equation (3) is exactly the TD(0) algorithm, and could also be called the *direct* implementation of incremental value iteration, *Q*-learning, and advantage learning.

3.2. Residual Gradient Algorithms

Reinforcement learning algorithms can be guaranteed to converge for lookup tables, yet be unstable for function-approximation systems that have even a small amount of generalization when using the direct implementation (Boyan, 95). To find an algorithm that is more stable than the direct algorithm, it is useful to specify the exact goal for the learning system. For the problem of prediction on a deterministic Markov chain, the goal can be stated as finding a value function such that, for any state x and its successor state x' , with a transition yielding immediate reinforcement R , the value function will satisfy the Bellman equation:

$$V(x) = \langle R + \gamma V(x') \rangle \quad (4)$$

For a given value function V , and a given state x , the *Bellman residual* is defined to be the difference between the two sides of the Bellman equation. The *mean squared Bellman residual* for an MDP with n states is therefore defined to be:

$$E = \frac{1}{n} \sum_x [\langle R + \gamma V(x') \rangle - V(x)]^2 \quad (5)$$

Residual gradient algorithms change the weights in the function-approximation system by performing gradient descent on the mean squared Bellman residual, E . This is called the *residual gradient* algorithm. The residual gradient algorithm and a faster version called the *residual* algorithm are proposed in (Baird, 95).

The counterpart of the Bellman equation for advantage learning is:

$$A^*(x, u) = \left\langle R + \gamma \Delta_t A_{\max_u}^*(x', u) \right\rangle \frac{1}{\Delta_t K} + \left(1 - \frac{1}{\Delta_t K}\right) A_{\max_u}^*(x, u) \quad (6)$$

If $A(x, u)$ is an approximation of $A^*(x, u)$, then the mean squared Bellman residual, E , is:

$$E = \left\langle \left(\left\langle R + \gamma \Delta_t A_{\max_u}^*(x', u) \right\rangle \frac{1}{\Delta_t K} + \left(1 - \frac{1}{\Delta_t K}\right) A_{\max_u}^*(x, u) - A(x, u) \right)^2 \right\rangle \quad (7)$$

where the inner $\langle \rangle$ is the expected value over all possible results of performing a given action u in a given state x , and the outer $\langle \rangle$ is the expected value over all possible states and actions.

4. DIFFERENTIAL GAMES

Differential games (Isaacs, 1965) are played in continuous time, or use sufficiently small time steps to approximate continuous time. Both players evaluate the given state and simultaneously execute an action, with no knowledge of the other player's selected action.

The *value* of a game is the long-term, discounted reinforcement if both opponents play the game optimally in every state. Consider a game in which player A tries to minimize the total discounted reinforcement, while the opponent, player B, tries to maximize the total discounted reinforcement. Given the advantage $A(x, u_A, u_B)$ for each possible action in state x , it is useful to define the minimax and maximin values for state x as:

$$\text{minimax}(x) = \min_{u_A} \max_{u_B} A(x, u_A, u_B) \quad (8)$$

$$\text{maximin}(x) = \max_{u_B} \min_{u_A} A(x, u_A, u_B) \quad (9)$$

If the minimax equals the maximin, then the minimax is called a *saddlepoint* and the optimal policy for both players is to perform the actions associated with the saddlepoint. If a saddlepoint does not exist, then the optimal policy is stochastic if an optimal policy exists at all. If a saddlepoint does not exist, and a learning system treats the minimax as if it were a saddlepoint, then the system will behave as if player A must choose an action on each time step, and then player B chooses an action based upon the action chosen by A. For the algorithms described below, a saddlepoint is assumed to exist. If a saddlepoint does not exist, this assumption confers a slight advantage to player B.

5. SIMULATION OF THE GAME

5.1. RESIDUAL ADVANTAGE LEARNING

During training, a state is chosen from a random distribution on each learning cycle. The vector of weights in the function approximation system, \mathbf{W} , is updated according to equation (10) on each time step.

$$\begin{aligned} \Delta W = & -\alpha \left((R + \gamma^{\Delta t} A_{\min \max}(x', u)) \frac{1}{\Delta t K} + \left(1 - \frac{1}{\Delta t K}\right) A_{\min \max}(x, u) - A(x, u) \right) \\ & \bullet \left(\phi \gamma^{\Delta t} \frac{\partial}{\partial \mathbf{W}} A_{\min \max}(x', u) \frac{1}{\Delta t K} + \phi \left(1 - \frac{1}{\Delta t K}\right) \frac{\partial}{\partial \mathbf{W}} A_{\min \max}(x, u) - \frac{\partial}{\partial \mathbf{W}} A(x, u) \right) \end{aligned} \quad (10)$$

The parameter ϕ is a constant that controls a trade-off between pure gradient descent (when ϕ equals 1) and a fast direct algorithm (when ϕ equals 0). ϕ can change adaptively by calculating two values, w_d and w_{rg} . These are *traces*, averages of recent values, updated according to:

$$\begin{aligned} w_d \leftarrow & (1 - \mu)w_d - \mu \left[(R + \gamma^{\Delta t} A_{\min \max}(x', u)) / \Delta t K + (1 - 1 / \Delta t K) A_{\min \max}(x, u) \right] \\ & \bullet \left[-\frac{\partial}{\partial w} A_{\min \max}(x, u) \right] \end{aligned} \quad (11)$$

$$\begin{aligned} w_{rg} \leftarrow & (1 - \mu)w_{rg} - \mu \left[(R + \gamma^{\Delta t} A_{\min \max}(x', u)) / \Delta t K + \right. \\ & \left. (1 - 1 / \Delta t K) A_{\min \max}(x, u) - A(x, u) \right] \\ & \bullet \left[\left(\gamma^{\Delta t} \frac{\partial}{\partial w} A_{\min \max}(x', u) \right) / (\Delta t K) + \right. \\ & \left. (1 - 1 / \Delta t K) \frac{\partial}{\partial w} A_{\min \max}(x, u) - \frac{\partial}{\partial w} A(x, u) \right] \end{aligned} \quad (12)$$

where μ was a small, positive constant that governed how fast the system forgets. On each time step a stable ϕ is calculated by using equation (13). This ensures convergence while maintaining fast learning:

$$\phi = \frac{\sum_w w_d w_{rg}}{\sum_w (w_d - w_{rg}) w_{rg}} + \mu \quad (13)$$

5.2. GAME DEFINITION

Our system is a differential game with a missile pursuing a plane, as in Rajan, Prasad, and Rao (1980) and Millington (1991). The action performed by the missile is a function of the state. The action performed by the plane is a function of the state and the action of the missile. The use of the minimax for determination of policy guarantees a solution to the game by ensuring a deterministic system.

The game is a Markov decision process with continuous states and non-linear dynamics. The state \mathbf{x} is a vector $(\mathbf{x}_m, \mathbf{x}_p)$ composed of the state of the missile and the state of the plane, each of which are composed of the position and velocity of the player in two-dimensional space. The action \mathbf{u} is a vector (u_m, u_p) composed of the action performed by the missile and the action performed by the plane, each of which is a scalar value; 0.5 indicates a 90 degree turn to the left, and -0.5 indicates a 90 degree turn to the right. The next state \mathbf{x}_{t+1} is a non-linear function of the current state \mathbf{x}_t and action \mathbf{u}_t . The speed of each player is fixed, with the speed of the missile twice that of the plane. On each time step the heading of each player is updated according to the action chosen, the velocity in both the x and y dimensions is computed for each player, and the positions of the players are updated.

The reinforcement function R is a function of the distance between the players. A reinforcement of 1 is given when the Euclidean distance between the players grows larger than 2 units (plane escapes). A reinforcement of -1 is

given when the distance grows smaller than 0.25 units (missile hits plane). No reinforcement is given when the distance is in the range [0.25,2]. The missile seeks to minimize reinforcement, while the plane seeks to maximize reinforcement.

The advantage function is approximated by a single-hidden-layer neural network with 50 hidden nodes. The hidden-layer nodes each have a sigmoidal activation function whose output lies in the range [-1,1]. The output of the network is a linear combination of the outputs of the hidden-layer nodes with their associated weights. To speed learning we use a separate adaptive learning rate for each weight in the network. There are 6 inputs to the network. The first 4 inputs describe the state and are normalized to the range [-1,1]. They consist of the difference in positions and velocities of the players in both the x and y dimensions. The remaining inputs describe the action to be taken by each player; 0.5 and -0.5 indicate left and right turns respectively.

6. RESULTS

Experiments were formulated to accomplish two objectives. The first objective was to determine to what degree residual advantage learning could learn the optimal policy for the missile/aircraft system. The second objective was to compare the performances of advantage learning when implemented in the residual gradient form, in the direct form, and using weighted averages of the two by using values of ϕ in the range [0,1].

In Experiment 1, the residual form of advantage learning learned the correct policy after 800,000 training cycles. The missile learned to pursue the plane, and the plane learned to evade the missile. Interesting behavior was exhibited by both players under certain initial conditions. First, the plane learned that in some cases it is able to indefinitely evade the missile by continuously flying in circles within the missile's turn radius. Second, the missile learned to anticipate the position of the plane. Rather than heading directly toward the plane, the missile learned to lead the plane under appropriate circumstances.



Figure 1a - The first snapshot (pictures taken of the actual simulator) demonstrates the missile leading the plane and, in the second snapshot, ultimately hitting the plane.

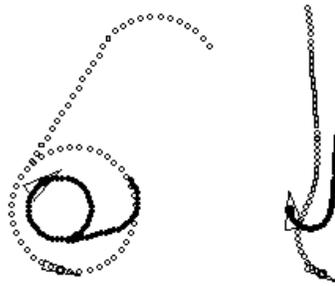


Figure 1b - The first snapshot demonstrates the ability of the plane to survive indefinitely by flying in continuous circles within the missiles turn radius. The second snapshot demonstrates the learned behavior of the plane to turn toward the missile to increase the distance between the two in the long term, a tactic used by pilots.

In Experiment 2, different values of ϕ were used for the weighting factor in residual advantage learning. Six different runs were accomplished, each using identical parameters with the exception of the weighting factor ϕ . Figure 2 presents the results of these experiments. In Figure 2, the dashed line is the error level after using an adaptive ϕ . A ϕ of 1 yields advantage learning in the residual gradient form, while a ϕ of 0 yields advantage learning implemented in the direct form.

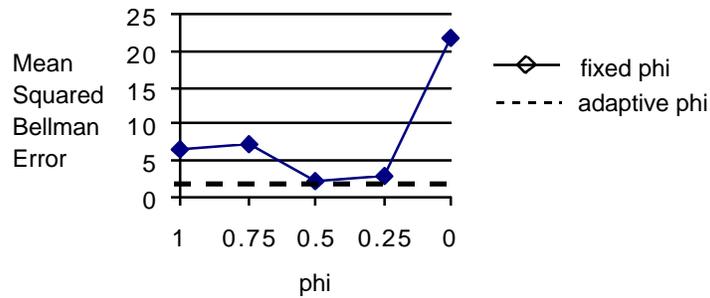


Figure 2: ϕ comparison

7. CONCLUSIONS

This is the first time that a reinforcement learning algorithm with guaranteed convergence for general function approximation systems has been demonstrated to work with a general neural network. This control problem is highly non-linear, with continuous states. In general, non-linear problems of this type are difficult to solve with classical game theory and control theory, and therefore appear to be good applications for reinforcement learning. It was shown that the residual algorithm with adaptive ϕ was able to perform as well as with the optimal ϕ . Furthermore, the policy learned by the system yielded behavior resembling the strategies used by pilots.

Acknowledgments

This research was supported under Task 2312R1 by the United States Air Force Office of Scientific Research.

References

- Baird, L.C. (1993). *Advantage updating*. Wright-Patterson Air Force Base, OH. (Wright Laboratory Technical Report WL-TR-93-1146, available from the Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145).
- Boyan, J.A., and A.W.Moore. (1995). Generalization in reinforcement learning: Safely approximating the value function. In Tesauro, G., Touretzky, D.S., and Leen, T.K. (eds.), *Advances in Neural Information Processing Systems 7*. MIT Press, Cambridge MA.
- Isaacs, Rufus (1965). *Differential games*. New York: John Wiley and Sons, Inc.
- Harmon, M.E., Baird, L.C., & Klopff, A.H. (1995). Advantage updating applied to a differential game. In Tesauro, G., Touretzky, D.S., and Leen, T.K. (eds.), *Advances in Neural Information Processing Systems 7*. MIT Press, Cambridge MA.
- Millington, P. J. (1991). *Associative reinforcement learning for optimal control*. Unpublished master's thesis, Massachusetts Institute of Technology, Cambridge, MA.
- Rajan, N., Prasad, U. R., and Rao, N. J. (1980). Pursuit-evasion of two aircraft in a horizontal plane. *Journal of Guidance and Control*. **3**(3), May-June, 261-267.
- Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by backpropagating errors. *Nature*. **323**, 9 October, 533-536.
- Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Doctoral thesis, Cambridge University, Cambridge, England.