Multi-Player Residual Advantage Learning With General Function Approximation

Mance E. Harmon
Wright Laboratory
WL/AACF
2241 Avionics Circle
Wright-Patterson Air Force Base,
OH 45433-7308
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

A new algorithm, *advantage learning,* is presented that improves on *advantage updating* by requiring that a single function be learned rather than two. Furthermore, advantage learning requires only a single type of update, the learning update, while advantage updating requires two different types of updates, a learning update and a normilization update. The reinforcement learning system uses the residual form of advantage learning. An application of reinforcement learning to a Markov game is presented. The test-bed has 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, resulting in a 90 degree instantaneous change in the aircraft's heading. 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. Speed of learning is increased by a new algorithm, *Incremental Delta-Delta* (IDD), which extends Jacobs' (1988) *Delta-Delta* for use in incremental training, and differs from Sutton's *Incremental Delta-Bar-Delta* (1992) in that it does not require the use of a trace and is amenable for use with general function approximation systems. The advantage learning algorithm for optimal control is modified for games in order to find the minimax point, rather than the maximum. Empirical results gathered using the missile/aircraft test-bed validate theory that suggests residual forms of reinforcement learning algorithms converge to a local minimum of the mean squared Bellman residual when using general function approximation systems. Also, to our knowledge, this is the first time an approximate second order method has been used with residual algorithms. Empirical results are presented comparing convergence rates with and without the use of IDD for the reinforcement learning test-bed described above and for a supervised learning test-bed. The results of these experiments demonstrate IDD increased the rate of convergence and resulted in an order of magnitude lower total asymptotic error than when using backpropagation alone.

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 empirical results demonstrating the residual form of advantage learning solving a nonlinear game using a general neural network. 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, which results in a 90 degree instantaneous change in heading for the aircraft. 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. Rate of convergence is increased by a new algorithm we call Incremental Delta-Delta (IDD), which extends Jacobs' (1988) Delta-Delta for use in incremental training, as opposed to epoch-wise training. IDD differs from Sutton's Incremental Delta-Bar-Delta (1992) in that it does not require the use of a trace, averages of recent values, and is useful for general function approximation systems. The advantage learning algorithm for optimal control is modified for games in order to find the minimax point, rather than the maximum. Empirical results gathered using the missile/aircraft test-bed validate theory that suggests residual forms of reinforcement learning algorithms converge to a local minimum of the mean squared Bellman residual when using general function approximation systems. Also, to our knowledge, this is the first time an approximate second order method has been used with residual algorithms, and we present empirical results comparing convergence rates with and without the use of IDD for the reinforcement learning test-bed described above and for a supervised-learning test-bed.

In Section 2 we present advantage learning and describe its improvements over advantage updating. In Section 3 we review direct algorithms, residual gradient algorithms, and residual algorithms. In Section 4 we present a brief discussion of game theory and review research in which game theory has been applied to MDP-like environments. In Section 5 we present Incremental Delta-Delta (IDD), an incremental, nonlinear extension to Jacobs' (1988) Delta-Delta algorithm. Also presented in Section 5 are empirical results generated from an application of the IDD algorithm to a nonlinear supervised learning task. Section 6 explicitly describes the reinforcement learning testbed and presents the update equations for residual advantage learning. Simulation results generated using the missile/aircraft test-bed are presented and discussed in Section 7. These results include diagrams of learned behavior, a comparison of the system's ability to reduce the mean squared Bellman error for different values of ϕ (including an adaptive ϕ), and a comparison of the system's performance with and without the use of IDD.

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, & Klopf, 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 normalizing 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:

$$V^*(x) = \max_u A^*(x,u) \tag{1}$$

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

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

where $\gamma^{\Delta t}$ is the discount factor per time step, *K* is a time unit scaling factor, and <> 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, meaning the value of the action is also the value of the state; for suboptimal actions the second term is negative, representing the degree of suboptimality relative to the optimal action.

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:

$$V(x) \leftarrow (1-\alpha)V(x) + \alpha[R + \gamma V(x')] \tag{3}$$

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:

$$V(x) = \langle R + \gamma V(x') \rangle \tag{4}$$

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:

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

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 counterpart of the Bellman equation for advantage learning is:

$$A^*(x,u) = \left\langle R + \gamma^{\Delta t} A^*_{\max_u}(x',u) \right\rangle \frac{1}{\Delta t K} + \left(1 - \frac{1}{\Delta t K}\right) A^*_{\max_u}(x,u) \tag{6}$$

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

$$E = \left\langle \left(\left\langle R + \gamma^{\Delta t} A_{\max_u}(x',u) \right\rangle \frac{1}{\Delta t K} + \left(1 - \frac{1}{\Delta t K}\right) A_{\max_u}(x,u) - A(x,u) \right)^2 \right\rangle \tag{7}$$

where the inner <> is the expected value over all possible results of performing a given action *u* in a given state *x*, and the outer <> is the expected value over all possible states and actions.

3.3 Residual Algorithms

Direct algorithms can be fast but unstable, and residual gradient algorithms may be stable but slow. Direct algorithms attempt to make each state like its successor, but ignore the effects of generalization during learning. Residual gradient algorithms take into account the effects of generalization, but attempt to make each state match both its successor and its predecessors. A weighted average of a direct algorithm with a residual gradient algorithm could have guaranteed convergence if the weighting factor ϕ is were chosen correctly. Such an algorithm would cause the mean squared Bellman residual to decrease monotonically, but would not necessarily follow the negative gradient, which would be the path of steepest descent. Therefore, it would be reasonable to refer to such algorithms as *residual* algorithms (Baird, 1995).

There is the question of how to choose ϕ appropriately. One approach is to treat it as a constant, like the learning rate constant. Just as a learning rate constant can be chosen to be as high as possible without causing the weights to blow up, so ϕ can be chosen as close to 0 as possible without the weights blowing up. A ϕ of 1 is guaranteed to converge, and a ϕ of 0 might be expected to learn quickly if it is stable at all. However, this may not be the best approach. It requires an additional parameter to be chosen by trial and error, and it ignores the fact that the best ϕ to use initially might not be the best ϕ to use later, after the system has learned for some time. Fortunately, it is easy to calculate the ϕ that ensures a decreasing mean squared residual, while bringing the weight change vector as close to the direct algorithm as possible (described in Section 6).

4 MULTI-PLAYER GAMES

The theory of Markov decision processes (Barto *et al.*, 1989, Howard 1960) is the basis for most of the recent reinforcement learning theory. However, this body of theory assumes that the learning system's environment is stationary and, therefore, contains no other adaptive systems. Game theory (von Neumann and Morgenstern, 1947) is explicitly designed for reasoning about multi-player environments.

Differential games (Isaacs, 1965) are games 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:

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

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

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 INCREMENTAL DELTA-DELTA (IDD) FOR NONLINEAR FUNCTION APPROXIMATORS

5.1 IDD Derivation

Incremental Delta-Bar-Delta (IDBD) was proposed by Sutton (1992) as an extension to the Delta-Bar-Delta algorithm (Jacobs, 1988) that makes the algorithm amenable to incremental tasks (learning tasks in which examples are processed one by one and then discarded). The IDBD algorithm was described by Sutton as a *meta-learning* algorithm in the sense that it learns the learning-rate parameters of an underlying base learning system. In Sutton (1992), the base learning system was the Least-Mean-Square (LMS) rule, also known as the Widrow-Hoff rule (Widrow and Stearns, 1985), and the IDBD algorithm was derived for linear function approximation systems. Here, we present an extension to Jacobs (1988) Delta-Delta algorithm that is appropriate for incremental training when using nonlinear function approximation systems.

As in the IDBD algorithm, in IDD each parameter of the the neural network has an associated learning rate of the form

$$\alpha_i(t) = e^{\beta_i(t)} \tag{10}$$

(where *i* indicates the parameter of association) that is updated after each step of learning. There are two advantages resulting from the exponential relationship of the learning rate, α_i, and the memory parameter that is actually modified, β_i. First, this assures that α_i will always be positive. Second, it allows geometric steps in α_i. As β_i is incremented or decremented by a fixed step-size, then α_i will move up or down by a fraction of its current value, allowing α_i to become very small. IDD updates β_i by

$$\beta_i(t+1) = \beta_i(t) + \theta \frac{\Delta w_i(t+1)}{\alpha_i(t)} \Delta w_i(t) \tag{11}$$

where Δw_i is a change in the weight parameter w_i, and θ is the meta-learning rate. The derivation of IDD is similar in principle to that of IDBD and is presented below. We start with

$$\beta_i(t+1) = \beta_i(t) - \theta \frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial \beta_i(t)} \tag{12}$$

where $\left\langle E^2\right\rangle$ is the expected value of the mean squared error. Applying the chain rule we may write

$$\frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial \beta_i(t)} = \frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial w_i(t+1)} \frac{\partial w_i(t+1)}{\partial \alpha_i(t)} \frac{\partial \alpha_i(t)}{\partial \beta_i(t)} \tag{13}$$

By evaluating the last term of equation (13), shown in equation (14), and then substituting the results we arrive at equation (15).

$$\frac{\partial \alpha_i(t)}{\partial \beta_i(t)} = \frac{\partial e^{\beta_i}}{\partial \beta_i} = e^{\beta_i} = \alpha_i(t) \tag{14}$$

$$\frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial \beta_i(t)} = \frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial w_i(t+1)} \frac{\partial w_i(t+1)}{\partial \alpha_i(t)} \alpha_i(t) \tag{15}$$

By evaluating the next to last term of equation (15) and rearranging, we find the equality described by equation (16).

$$\frac{\partial w_i(t+1)}{\partial \alpha_i(t)} = \frac{\partial}{\partial \alpha_i(t)} \left(w_i(t) - \alpha_i(t) \frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial w_i(t)} \right)$$

$$\frac{\partial w_i(t+1)}{\partial \alpha_i(t)} = -\frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial w_i(t)} \tag{16}$$

Again, substituting the results of equation (16) into (15) produces

$$\frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial \beta_i(t)} = -\frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial w_i(t+1)} \frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial w_i(t)} \alpha_i(t) \tag{17}$$

Next, we define the change in the parameter w_i and rearrange for substitution.

$$\begin{aligned} \Delta w_i(t) &= -\alpha_i(t) \frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial w_i(t)} \\ -\frac{\Delta w_i(t)}{\alpha_i(t)} &= \frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial w_i(t)} \end{aligned} \tag{18}$$

By substituting the left-hand side of the second half of equation (18) into equation (17), and then deriving the equivalent of equation (18) for $\Delta w_i(t+1)$ and substituting into equation (17), we arrive at

$$\frac{\partial \frac{1}{2}\left\langle E^2(t+1)\right\rangle}{\partial \beta_i(t)} = -\frac{\Delta w_i(t+1)}{\alpha_i(t)} \Delta w_i(t) \tag{19}$$

Thus

$$\beta_i(t+1) = \beta_i(t) + \theta \frac{\Delta w_i(t+1)}{\alpha_i(t)} \Delta w_i(t) \tag{20}$$

The right-hand side of equation (19) provides a true unbiased estimate of the gradient of the error surface with respect to the memory parameter, β_i. An equivalent of IDBD for nonlinear systems can trivially be derived from IDD by replacing $\Delta w_i(t)$ with $\Delta \bar{w}_i(t)$, where $\Delta \bar{w}_i(t)$ is an exponentially weighted sum of the current and past changes to w_i. The trace is defined by equation (21).

$$\Delta \bar{w}_i(t) = (1-\varepsilon)\Delta w_i(t-1) + \varepsilon \Delta \bar{w}_i(t-1) \tag{21}$$

This form of IDBD for nonlinear systems includes another free parameter, ε, that determines the decay rate of the trace. If it were possible to choose ε perfectly for each training example, IDBD would, in the worst case, be equivalent to IDD, and would on average provide a better estimate of the gradient. However, the optimal value of ε is a function of the rate of change in the gradient of the error surface, and is therefore different for different regions of state space. Moreover, Jacobs' original motivations for using delta-bar-delta rather than delta-delta are no longer relevant when each learning rate is defined according to equation (10). For these reasons we used IDD to speed the convergence rate for our testbed.

5.2 IDD Supervised Learning Results

The capabilities of IDD were initially assessed using a supervised-learning task. The intent of this experiment was to answer the question: Does the IDD algorithm perform better than the ordinary backpropogation algorithm? The task involved six real-valued inputs (including a bias) and one output. The inputs were chosen independently and randomly in the range [-1, 1]. The objective function was the square of the first input summed with the second input. The function approximator was a single-hidden-layer sigmoidal network with 5 hidden nodes. For each algorithm, we trained the network for 50,000 iterations and then measured the asymptotic error. This process was repeated 100 times using different initial random number seeds, and the results were averaged. The experiment was repeated for different values of α in the range [0.35, 0.9]. The equivalent was done for the IDD algorithm. The experiment was repeated for values of the meta-learning rate θ in the range [0.1, 1.0] in increments of 0.1. The results are presented in Figure 1, and show that the IDD algorithm finds a distribution of learning rates that is better than any single learning rate shared by all weights. The IDD algorithm consistently reduced the error by an order of magnitude more than backprop alone.

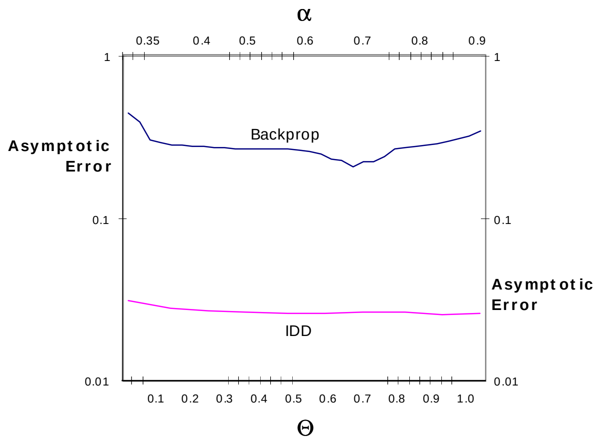

Figure 1: Comparison of IDD and the backpropogation algorithms

6 SIMULATION OF THE GAME

6.1 RESIDUAL ADVANTAGE LEARNING

During training, a state is chosen from a uniform random distribution on each learning cycle. The vector of weights in the function approximation system, **W,** is updated according to equation (22) on each learning cycle.

$$\begin{aligned}\Delta W = -\alpha\Bigg(& \left(R+\gamma^{\Delta t} A_{\min\max}(x',u)\right)\frac{1}{\Delta t K} + \left(1-\frac{1}{\Delta t K}\right) A_{\min\max}(x,u) - A(x,u)\Bigg) \\ & \bullet\left(\phi\gamma^{\Delta t}\frac{\partial}{\partial \mathbf{W}} A_{\min\max}(x',u)\frac{1}{\Delta t K} + \phi\left(1-\frac{1}{\Delta t K}\right)\frac{\partial}{\partial \mathbf{W}} A_{\min\max}(x,u) - \frac{\partial}{\partial \mathbf{W}} A(x,u)\right)\end{aligned} \tag{22}$$

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). A ϕ that ensures a decreasing mean squared residual, while bringing the weight change vector as close to the direct algorithm as possible can be calculated by maintaining an estimate of the epoch-wise weight change vectors. These can be approximated by maintaining two scalar values, w_d and w_{rg} associated with each weight w in the function approximation system. These are *traces*, averages of recent values, used to approximate $\Delta\mathbf{W}_d$ and $\Delta\mathbf{W}_{rg}$. The traces are updated on each learning cycle according to:

$$w_d \leftarrow (1-\mu)w_d - \mu\Big[\big(R+\gamma^{\Delta t}A_{\min\max}(x',u)\big)/\Delta tK + (1-1/\Delta tK)A_{\min\max}(x,u)\Big] \bullet \left[-\frac{\partial}{\partial w}A_{\min\max}(x,u)\right] \tag{23}$$

$$w_{rg} \leftarrow (1-\mu)w_{rg} - \mu\left[\begin{array}{l}\big(R+\gamma^{\Delta t}A_{\min\max}(x',u)\big)/\Delta tK + \\ (1-1/\Delta tK)A_{\min\max}(x,u) - A(x,u)\end{array}\right] \bullet \left[\begin{array}{l}\left(\gamma^{\Delta t}\frac{\partial}{\partial w}A_{\min\max}(x',u)\right)/(\Delta tK) + \\ (1-1/\Delta tK)\frac{\partial}{\partial w}A_{\min\max}(x,u) - \frac{\partial}{\partial w}A(x,u)\end{array}\right] \tag{24}$$

where μ is a small, positive constant that governs how fast the system forgets. On each time step a stable ϕ is calculated by using equation (25). This ensures convergence while maintaining fast learning:

$$\phi = \frac{\sum_w w_d w_{rg}}{\sum_w (w_d - w_{rg})w_{rg}} + \mu \tag{25}$$

It is important to note that this algorithm does not follow the negative gradient, which would be the steepest path of descent. However, the algorithm does cause the mean squared residual to decrease monotonically (for appropriate ϕ), thereby guaranteeing convergence to a local minimum of the mean squared Bellman residual.

6.2 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 Markov with continuous states and nonlinear dynamics. The state **x** is a vector ($\mathbf{x}_m$,$\mathbf{x}_p$) composed of two elements: the state of the missile, $\mathbf{x}_m$, and the state of the plane, $\mathbf{x}_p$, each of which are vectors composed of four elements containing the scalar values of the x and y coordinates and x and y velocities of the players in two-dimensional space. The action **u** is a vector (u_m,u_p) composed of two scalar elements. The element u_m represents the action performed by the missile, and the element u_p represents the action performed by the plane; an action value of 0.5 indicates an instantaneous 90-degree change of heading to the left of the current heading, and an action value of -0.5 indicates an instantaneous 90-degree change of heading to the right of the current heading. The next state $\mathbf{x}_{t+1}$ is a nonlinear 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. Therefore, the Euclidean distance measure of the change in position for one state transition is a fixed scalar value for each player, with the value for the missile being twice that of the value for 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. Pseudocode describing these dynamics follows:

```
deg2rad - converts degrees to radians; // trigonometric functions measured in radians
Plane_action - actions chosen by plane;
Plane_theta - current heading of plane in 2d space;
Missile_velocity_X - component vector velocity of missile in x dimension;
Missile_velocity_Y - component vector velocity of missile in y dimension;
1)  If (Plane_action = 0.5) then Plane_theta = Plane_theta + 90 * deg2rad;
    else Plane_theta = Plane_theta - 90 * deg2rad;
2)  Repeat step 1 for Missile.
3)  Normalize Plane_theta and Missile_theta.
4)  Missile_velocity_X = Missile_speed * cos(Missile_theta);
5)  Missile_velocity_Y = Missile_speed * sin(Missile_theta);
6)  Repeat steps 4 & 5 for Plane;
```

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 is greater than 2 units (plane escapes). A reinforcement of -1 is given when the distance is less 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, the output of which 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 the rate of convergence we used IDD as described in Section 5. 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 differences 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.

7 RESULTS

Experiments were formulated to accomplish three objectives. The first objective was to determine heuristically to what degree residual advantage learning could learn a reasonable policy for the missile/aircraft system. In Harmon, Baird, and Klopf (1995) it was possible to calculate the optimal weights for the quadratic function approximator used to represent the advantage function. This is not the case for the current system. The nonlinear dynamics of this system require more representational capacity than a simple quadratic network to store the advantage function. In using a single hidden-layer sigmoidal network we gain the representational capacity needed but lose the ability to calculate the optimal parameters for the network, which would have been useful as a metric. For this reason, our metric is reduced to simple observation of the system behavior, and is analogous to the metric used to evaluate Tesauro's TD-Gammon (Tesauro, 1990). Also, it is possible this game might be made less difficult to solve if expressed in an appropriate coordinate system, such as plane and missile centered polar coordinates. However, the motivation for this experiment is to demonstrate the ability of residual algorithms to solve difficult, nonlinear control problems using a general neural network. For this reason, the game was explicitly structured to be difficult to solve.

The second objective was to analyze the performance of three different forms of advantage learning: the residual gradient form, the direct form, and a weighted average of the two (values of ϕ in the range [0, 1]). The third and final objective was to evaluate the utility of IDD for this test-bed, and to address the following question. When using residual algorithms, which method increases the rate of convergence the most: 1) Using the residual gradient form of the algorithm with a second order method or, 2) Using the residual form of the algorithm with an adaptive ϕ or,
3) Some combination of the two?

Addressing the first objective, the reinforcement learning system implementing the residual form of advantage learning produced a reasonable 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 evade indefinitely 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.

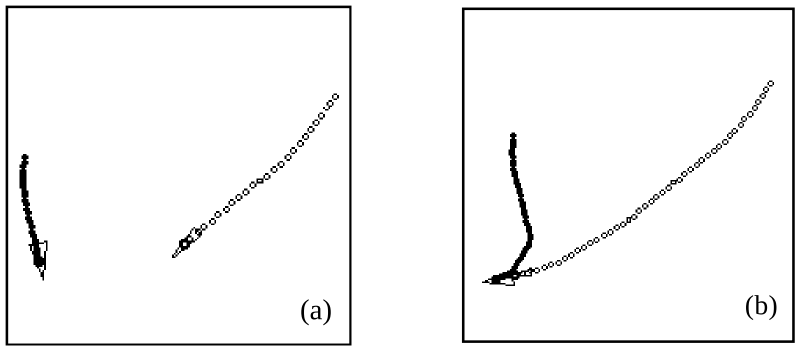

Figure 2 (a) Demonstration of missile leading plane after learning and (b) ultimately hitting the plane.

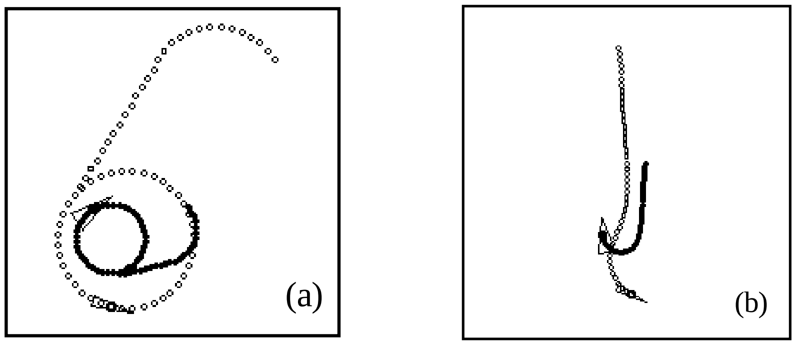

Figure 2 (a) Demonstration of the ability of the plane to survive indefinitely by flying in continuous circles within the missile's turn radius. (b) Demonstration of the learned behavior of the plane to turn toward the missile to increase the distance between the two in the long term.

In Experiment 2, the effects of different values of ϕ, the weighting factor used in the linear combination of the residual gradient update vector and the direct method update vector, and the use of IDD on the learning system's convergence rate were compared. For Δt values of 1.0 and 0.1, twelve different runs were accomplished, each using identical parameters with the exception of the weighting factor ϕ and the use or non-use of IDD. Figure 3 presents the results of these experiments. 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.

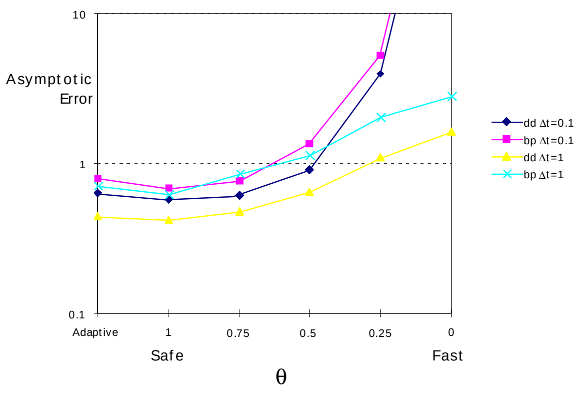

Figure 3: ϕ comparison

A ϕ of 1, which yields the residual gradient algorithm, minimized the mean squared Bellman residual more than the other values of ϕ test, including the adaptive ϕ. The use of IDD did result in a lower total error in all comparisons. Furthermore, the use of IDD was imperative for finding a control policy that generated reasonable trajectories for the aircraft. The use of IDD not only increased the rate of convergence, but resulted in a smaller asymptotic error level. When training with a value of 0 for the parameter ϕ, reducing the algorithm to the direct method of advantage learning, the weights grew without bound when using a time duration, Δt, of 0.1.

Residual algorithms increase the speed of convergence by following the gradient of the direct method as close as possible, while still guaranteeing convergence to a local minimum by ensuring the weight change vectors monotonically reduce the mean squared Bellman residual error. By combining residual algorithms with IDD, we have two separate mechanisms pursuing the same goal: a fast rate of convergence. How these mechanisms interact, complement, or inhibit one another is not fully understood. It was necessary to put a ceiling of -4 on the growth of β to ensure system stability. One might think that by simply decreasing the meta-learning rate θ, one would not need to add heuristically a ceiling to the parameter β. This turned out not to be the case. Even for a very small θ, there did exist β that grew to the point of causing the system to be unstable (the weights grew to infinity).

Further research is needed to determine if, in general, using second-order methods with residual gradient algorithms is desirable over using residual algorithms with an adaptive ϕ. Although this was the case for the experiments described above, the following results were also generated using the missile/aircraft test-bed using slightly different parameter settings (e.g., k, Δt, μ). These results reflect the use of IDD.

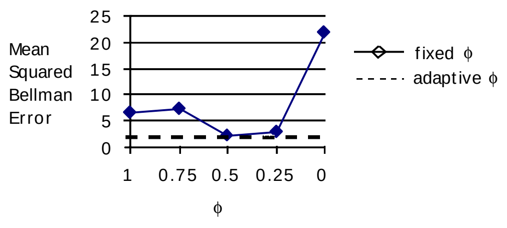

Figure 4: The use of an adaptive ϕ reduced the mean squared Bellman residual further than any of the tested fixed ϕ's.

For this set of experiments the Bellman error was minimized the most by combining the use of an adaptive ϕ with the use of IDD. The resultant control policy from these experiments also produced aircraft trajectories that looked reasonable. It is the case that using IDD resulted in a lower mean squared Bellman error for all values of ϕ, including the adaptive ϕ. Why this is the case and how these mechanisms interact will be explored in future research.

8 CONCLUSIONS

The results gathered using the missile/aircraft test-bed provide evidence that residual forms of reinforcement learning algorithms produce reinforcement learning systems that are stable and converge to a local minimum of the mean squared Bellman residual when using general function approximation systems. 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.

The data also suggests that the use of second-order methods may be desirable or even necessary, as was the case for this test-bed, to generate the desired control policy. Although much research has been accomplished investigating approaches for speeding rates of convergence, the results gathered from applying these methods in supervised learning tasks may not necessarily hold true for reinforcement learning tasks. This stems from fundamental differences in the nature of supervised and reinforcement learning. For this reason, we feel that a rigorous comparison of these methods implemented in a reinforcement learning system is an appropriate topic for future research.

Acknowledgments

This research was supported under Task 2312R1 by the United States Air Force Office of Scientific Research. Thanks to Michael Littman for a useful discussion of Markov games. We also wish to thank Matt Rizki and Lou Tambourino of the Adaptive Vision Laboratory at Wright State University for making available the use of their equipment.

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).

Baird, L. C. (1995). Residual Algorithms: Reinforcement Learning with Function Approximation. In Armand Prieditis & Stuart Russell, eds. *Machine Learning: Proceedings of the Twelfth International Conference*, 9-12 July, Morgan Kaufman Publishers, San Francisco, CA.

Barto, A. G., Sutton, R. S., & Watkins, C. J. C. H. (1989). Learning and sequential decision making. Technical Report 89-95, Department of Computer and Information Science, Univeristy of Massachusetts, Amherst, Massachusetts. Also published in *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, Michael Gabriel and John Moore, editors. MIT Press, Cambridge MA (1991).

Boyan, J.A., and Moore, A.W. (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.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks 1*, pp. 295-307.

Harmon, M.E., Baird, L.C, & Klopf, 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.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. 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.

Sutton, R. S. (1992). Adapting bias by gradient descent: an incremental version of delta-bar-delta. In *Proceedings of the Tenth National Conference on Machine Learning*. MIT Press, pp. 171-176, Cambridge MA.

Tesauro, G. (1990). Neurogammon: A neural-network backgammon program. *Proceedings of the International Joint Conference on Neural Networks* (pp. 33-40). San Diego, CA.

Von Neumann, J., and Morgenstern, O. (1947). *Theory of Games and Economic Behavior*. Princeton University Press, Princeton NJ.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Doctoral thesis, Cambridge University, Cambridge, England.

Widrow, B., and Stearns, S. D. (1985). *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall.