Residual Algorithms:
Reinforcement Learning with Function Approximation

Leemon Baird
Department of Computer Science
U.S. Air Force Academy, CO 80840-6234
baird@cs.usafa.af.mil
http://kirk.usafa.af.mil/~baird

ABSTRACT

A number of reinforcement learning algorithms have been developed that are guaranteed to converge to the optimal solution when used with lookup tables. It is shown, however, that these algorithms can easily become unstable when implemented directly with a general function-approximation system, such as a sigmoidal multilayer perceptron, a radial-basis-function system, a memory-based learning system, or even a linear function-approximation system. A new class of algorithms, *residual gradient* algorithms, is proposed, which perform gradient descent on the mean squared Bellman residual, guaranteeing convergence. I shown, however, that they may learn very slowly in some cases. A larger class of algorithms, *residual* algorithms, is proposed that has the guaranteed convergence of the residual gradient algorithms, yet can retain the fast learning speed of direct algorithms. In fact, both direct and residual gradient algorithms are shown to be special cases of residual algorithms, and it is shown that residual algorithms can combine the advantages of each approach. The direct, residual gradient, and residual forms of value iteration, Q-learning, and advantage learning are all presented. Theoretical analysis is given explaining the properties these algorithms have, and simulation results are given that demonstrate these properties.

1 INTRODUCTION

A wide range of optimal control problems can be viewed as *Markov Decision Problems* (MDPs), which are systems that change state stochastically, with the probability distribution for each transition determined by the current state, and the action chosen by a learning system. A number of reinforcement learning algorithms have been proposed that are guaranteed to learn a *policy*, a mapping from states to actions, such that performing those actions in those states maximizes the expected, total, discounted reinforcement received:

$$V = \left\langle \sum_t \gamma^t R_t \right\rangle \tag{1}$$

where R_t is the reinforcement received at time t, <> is the expected value over all stochastic state transitions, and γ is the discount factor, a constant between zero and one that gives more weight to near-term reinforcement, and that guarantees the sum will be finite for bounded reinforcement. In general, these reinforcement learning systems have been analyzed for the case of an MDP with a finite number of states and actions, and for a learning system containing a lookup table, with separate entries for each state or state-action pair. Lookup tables typically do not scale well for high-dimensional MDPs with a continuum of states and actions (the curse of dimensionality), so a general function-approximation system must typically be used, such as a sigmoidal, multi-layer perceptron, a radial-basis-function network, or a memory-based-learning system. In the following sections, various methods are analyzed that combine reinforcement learning algorithms with function approximation systems. Algorithms such as Q-learning or value iteration are guaranteed to converge to the optimal answer when used with a lookup table. An obvious method for combining them with function-approximation systems, which is called the *direct* algorithm here, can be implemented, but direct Q-learning or direct value iteration can fail to converge to an answer. A new class of algorithms, *residual gradient* algorithms, are shown to always converge, but residual gradient Q-learning and residual gradient value iteration may converge very slowly in some cases. Finally, a new class of algorithms, *residual* algorithms, are proposed. It will be shown that direct and residual gradient algorithms are actually special cases of residual algorithms, and that residual algorithms can be easily found such that residual Q-learning or residual value iteration have both guaranteed convergence, and converge quickly on problems for which residual gradient algorithms converge slowly.

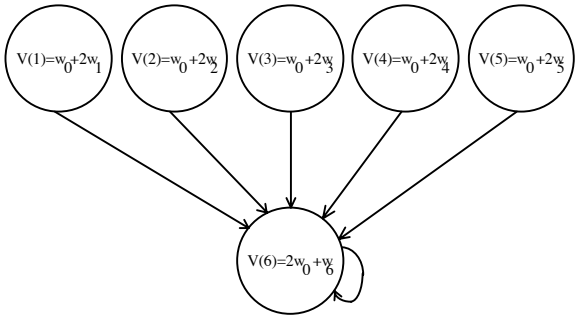


Figure 1. The star problem

2 ALGORITHMS FOR LOOKUP TABLES

Perhaps the simplest form of reinforcement learning problem is the task of learning the value function for a Markov chain, which is a degenerate MDP for which there is only one possible action to choose from in each state. Such problems are often solved using algorithms based upon dynamic programming (Bertsekas 87), which involves storing information associated with each state, then updating the information in one state based upon the information in subsequent states. For predicting the outcome of a Markov chain, an obvious learning algorithm is an incremental form of value iteration, which is defined as:

$$V(x) \xleftarrow{\alpha} R + \gamma V(x') \qquad (2)$$

Update (2) represents the learning that occurs after observing a transition from state x to state x' with immediate reinforcement of R. The value of the earlier state, $V(x)$, is modified to be closer to the value of the expression on the right side, $R+\gamma V(x')$, with the rate of learning controlled by a learning rate α. For this particular type of MDP, if each $V(x)$ is a separate entry i lookup table, then update (2) is also equivalent to three other reinforcement learning algorithms: *TD(0)* (Sutton 88), *Q-learning* (Watkins 89), and *advantage learning* (Baird 95). If an implementation of update (2) fails to converge in some cases, then all of these other algorithms also fail to converge in some cases, and so it is important to find an algorithm that can solve this simple MDP using general function-approximation systems.

3 DIRECT ALGORITHMS

If the MDP has a finite number of states, and each $V(x)$ is represented by a unique entry in a lookup table, and each possible transition is experienced an infinite number of times during learning, then update (2) is guaranteed to converge to the optimal value function as the learning rate α decays to zero at an appropriate rate. The various states can be visited in any order during learning, and some can be visited more often than others, yet the algorithm will still converge if the learning rates decay appropriately (Watkins, Dayan 92). If $V(x)$ was represented by a function-approximation system other than a lookup table, update (2) could 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. For any particular weight w in the function-approximation system, the weight change would be:

$$\Delta w = \alpha\Big(R + \gamma V(x') - V(x)\Big)\frac{\partial V(x)}{\partial w} \qquad (3)$$

Equation (3) is exactly the TD(0) algorithm, by definition. It could also be called the *direct* implementation of incremental value iteration, Q-learning, and advantage learning. The direct algorithm reduces to the original algorithm when used with a lookup table. Tesauro (90,92)has shown very good results by combining TD(0) with backpropagation (and also using the more general TD(λ)). Since it is guaranteed to converge for the lookup table, this approach might be expected to also converge for general function-approximation systems. Unfortunately, this is not the case, as is illustrated by the MDP shown in figure 1. In figure 1, there are six states, and the value of each state is given by the linear combination of two weights. Every transition yields a reinforcement of zero. During training, each possible transition is observed equally often. The function-approximation system is simply a lookup table, with one additional entry giving generalization. This is an

extremely benign form of function-approximation system. It is linear, it is general (can represent any value function over those states), the state vectors are linearly independent, and all have the same magnitude (1-norm, 2-norm, or infinity-norm). Furthermore, it has the desirable property that using backpropagation to change the value in one state will cause neighboring states to change by at most two-thirds as much. Therefore, this system exhibits only mild generalization. If one wished to extend the Watkins and Dayan proofs to function-approximation systems, this would appear to be an ideal system for which convergence to optimality could be guaranteed for the direct method. However, that is not the case.

If the weight w_0 were not being used, then the weights and values would all converge to zero, which is the correct answer. However, in this example, if all weights are initially positive, and $V(6)$ is initially much larger than all of the other values, then all of the values will grow without bound. This is due to the fact that when the first five values are lower than the value of their successor, $\gamma V(6)$, and $V(6)$ is higher than the value of its successor, $\gamma V(6)$, then w_0 is increased five times for every time that it is decreased, so it will rise rapidly. Of course, w_6 will fall, but more slowly, because it is updated less frequently. The net effect then is that all of the values and all of the weights go to positive infinity, except for w_6, which goes to negative infinity.

4 RESIDUAL GRADIENT ALGORITHMS

It is unfortunate that a reinforcement learning algorithm can be guaranteed to converge for lookup tables, yet be unstable for function-approximation systems that have even a small amount of generalization. Algorithms have been proved to converge for quadratic function-approximation systems (Bradtke 93), but it would be useful to find an algorithm that converges for any function-approximation system. 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:

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

For a system with a finite number of states, the optimal value function is the unique function that satisfies the Bellman equation. 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:

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

If the Bellman residual is nonzero, then the resulting policy will be suboptimal, but for a given level of Bellman residual, the degree to which the policy yields suboptimal reinforcement can be bounded (Williams, Baird 93). This suggests it might be reasonable to change the weights in the function-approximation system by performing gradient descent on the mean squared Bellman residual, E. This could be called the *residual gradient* algorithm. To do this for a deterministic MDP, after a transition from a state x to a state x', with reinforcement R, a weight w would change according to:

$$\Delta w = -\alpha \left[R + \gamma V(x') - V(x) \right] \left[\frac{\partial}{\partial w} \gamma V(x') - \frac{\partial}{\partial w} V(x) \right] \tag{6}$$

For a system with a finite number of states, E is zero if and only if the value function is optimal. Therefore, performing gradient descent on E guarantees that E will eventually converge to a local minimum. This type of an algorithm could be called a residual gradient algorithm and, unlike the direct algorithms, residual gradient algorithms have guaranteed convergence of the mean squared Bellman residual. Furthermore if the function-approximation system is general enough to represent any value function, and there is a differentiable mapping from value functions to the corresponding weight vectors, then the residual gradient algorithm is guaranteed to converge to the optimal answer.

Although residual gradient algorithms have guaranteed convergence, that does not necessarily mean that they will always learn as quickly as direct algorithms. Applying the direct algorithm to the example in figure 2 causes state 5 to quickly converge to zero. State 4 then quickly converges to zero, then state 3, and so on. Information flows purely from later states to earlier states, so the initial value of w_4, its behavior over time, has no effect on the speed at which $V(5)$ converges to zero. Applying the residual gradient algorithm to figure 1 results in much slower learning. For example, if initially w_5=0 and w_4=10, then when learning from the transition from state 4 to state 5, the direct algorithm would simply decrease w_4, but the

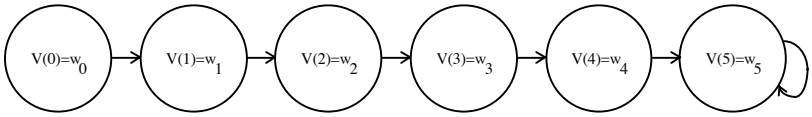


Figure 2. The hall problem.

residual gradient algorithm would both decrease w_4 and increase w_5. Thus the residual gradient algorithm causes information to flow both ways, with information flowing in the wrong direction moving slower than information flowing in the right direction by a factor of γ. If γ is close to 1.0, then it would be expected that residual gradient algorithms would learn very slowly on the problem in figure 2.

5 RESIDUAL ALGORITHMS

Direct algorithms can be fast but unstable, and residual gradient algorithms can be stable but slow. Direct algorithms attempt to make each state match its successors, 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 successors and its predecessors. These effects can be seen more easily by considering *epoch-wise* training, where a weight change is calculated for every transition once, then weight changes are summed and the weights are changed appropriately. In this case, the total weight change after one epoch for the direct method and the residual gradient method, respectively, are:

$$\Delta \mathbf{W}_d = -\alpha \sum_x \left[R + \gamma V(x') - V(x)\right]\left[-\nabla_{\mathbf{W}} V(x)\right] \quad (7)$$

$$\Delta \mathbf{W}_{rg} = -\alpha \sum_x \left[R + \gamma V(x') - V(x)\right] \left[\nabla_{\mathbf{W}} \gamma V(x') - \nabla_{\mathbf{W}} V(x)\right] \quad (8)$$

In these equations, $\mathbf{W}$, $\Delta\mathbf{W}$, and the gradients of $V(x)$ and $V(x')$ are all vectors, and the summation is over all states that are updated. If some states are updated more than once per epoch, then the summation should include those states more than once. The advantages of each algorithm can then be seen graphically.

Figure 3 shows a situation in which the direct method will cause the residual to decrease (left) and one in which it causes the residual to increase (right). The latter is a case in which the direct method may not converge. In figure 4, the residual gradient vector shows the direction of steepest descent on the mean squared Bellman residual. The dotted line represents the hyperplane that is perpendicular to the gradient. Any weight change vector that lies to the left of the dotted line will result in a decrease in the mean squared Bellman residual, E. Any vector lying along the dotted line results in no change, and any vector to the right of the dotted line results in an increase in E. If an algorithm always decreases E, then clearly E must converge. If an algorithm sometimes increases E, then it becomes more difficult to predict whether it will converge. A reasonable approach, therefore, might be to change the weights according to a weight-change vector that is as close as possible to $\Delta\mathbf{W}_d$, so as to learn quickly, while still remaining to the left of the dotted line, so as to remain stable.

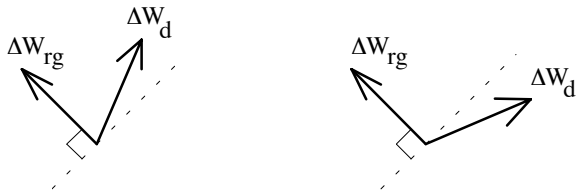


Figure 3. Epoch-wise weight-change vectors for direct and residual gradient algorithms

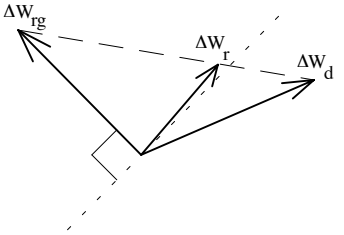


Figure 4. Weight-change vectors for direct, residual gradient, and residual algorithms.

This weighted average of a direct algorithm with a residual gradient algorithm could have guaranteed convergence, because $\Delta\mathbf{W}_r$ causes E to decrease, and might be expected to be fast, because $\Delta\mathbf{W}_r$ lies as close as possible to $\Delta\mathbf{W}_d$: Actually, the closest stable vector to $\Delta\mathbf{W}_d$ could be found by projecting $\Delta\mathbf{W}_d$ onto the plane perpendicular to $\Delta\mathbf{W}_{rg}$, which is represented by the dotted line. But the resulting vector would be collinear with $\Delta\mathbf{W}_r$, so $\Delta\mathbf{W}_r$ should learn just as quickly for appropriate choices of learning rate. $\Delta\mathbf{W}_r$ is simpler to calculate, and so appears to be the most useful algorithm to use. For a real number ϕ between 0 and 1, $\Delta\mathbf{W}_r$ is defined to be:

$$\Delta \mathbf{W}_r = (1-\phi)\Delta \mathbf{W}_d + \phi \Delta \mathbf{W}_{rg} \tag{9}$$

This algorithm is guaranteed to converge for an appropriate choice of ϕ. The algorithm causes the mean squared residual to decrease monotonically (for appropriate ϕ), but it does not follow the negative gradient, which would be the path of steepest descent. Therefore, it would be reasonable to refer to the algorithm as a *residual* algorithm, rather than as a *residual gradient* algorithm. A residual algorithm is defined to be any algorithm in the form of equation (9), where the weight change is the weighted average of a residual gradient weight change and a direct weight change. By this definition, both direct algorithms and residual gradient algorithms are special cases of residual algorithms.

An important question is 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 can learn 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. To accomplish this, simply use the lowest ϕ possible (between zero and one) such that:

$$\Delta \mathbf{W}_r \cdot \Delta \mathbf{W}_{rg} > 0 \tag{10}$$

As long as the dot product is positive, the angle between the vectors will be acute, and the weight change will result in a decrease in E. A ϕ that creates a stable system, in which E is monotonically decreasing, can be found by requiring that the two vectors be orthogonal, then adding any small, positive constant ε to ϕ to convert the right angle into an acute angle:

$$\begin{aligned} &\Delta \mathbf{W}_r \cdot \Delta \mathbf{W}_{rg} = 0 \\ &((1-\phi)\Delta \mathbf{W}_d + \phi \Delta \mathbf{W}_{rg}) \cdot \Delta \mathbf{W}_{rg} = 0 \\ &\phi = \frac{\Delta \mathbf{W}_d \cdot \Delta \mathbf{W}_{rg}}{\Delta \mathbf{W}_d \cdot \Delta \mathbf{W}_{rg} - \Delta \mathbf{W}_{rg} \cdot \Delta \mathbf{W}_{rg}} \end{aligned} \tag{11}$$

If this equation yields a ϕ outside the range [0,1], then the direct vector does make an acute angle with the residual gradient vector, so a ϕ of 0 should be used for maximum learning speed. If the denominator of ϕ is zero, this either means that E is at a local minimum, or else it means that the direct algorithm and the residual gradient algorithm yield weight-change vectors pointing in the same direction. In either case, a ϕ of 0 is acceptable. If the equation yields a ϕ between zero and one, then this is the ϕ that causes the mean squared Bellman residual to be constant. Theoretically, any ϕ above this value will ensure convergence. Therefore, a practical implementation of a residual algorithm should first calculate the numerator and denominator separately, then check whether the denominator is zero. If the denominator is zero, then ϕ=0. If it is not, then the algorithm should evaluate equation (11), including the addition of a small constant ε, then check whether the resulting ϕ lies in the range [0,1]. A ϕ outside this range should be clipped to lie on the boundary of this range.

The above defines residual algorithms in general. For the specific example used in equations (7) and (8), the corresponding residual algorithm would be:

$$\begin{aligned} &\Delta \mathbf{W}_r = (1-\phi)\Delta \mathbf{W}_d + \phi \Delta \mathbf{W}_{rg} \\ &= -(1-\phi)\alpha \sum_x \left[R + \gamma V(x') - V(x)\right]\left[-\nabla_{\mathbf{w}} V(x)\right] \\ &\quad -\phi\alpha \sum_x \left[R + \gamma V(x') - V(x)\right]\left[\nabla_{\mathbf{w}} \gamma V(x') - \nabla_{\mathbf{w}} V(x)\right] \\ &= -\alpha \sum_x \left[R + \gamma V(x') - V(x)\right]\left[\phi \nabla_{\mathbf{w}} \gamma V(x') - \nabla_{\mathbf{w}} V(x)\right] \end{aligned} \tag{12}$$

To implement this in an incremental manner, rather than epoch-wise, the change in a particular weight w after observing a particular state transition would be:

$$\begin{aligned} \Delta w = -\alpha &\left[R + \gamma V(x') - V(x)\right] \\ &\left[\phi\gamma \tfrac{\partial}{\partial w} V(x') - \tfrac{\partial}{\partial w} V(x)\right] \end{aligned} \tag{13}$$

It is interesting that the residual algorithm turns out to be identical to the residual gradient algorithm in this case, except that one term is multiplied by ϕ.

To find the marginally-stable ϕ using equation (16), it is necessary to have 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 the function-approximation system. These will be *traces*, averages of recent values, used to approximate $\Delta \mathbf{W}_d$ and $\Delta \mathbf{W}_{rg}$, respectively. The traces are updated according to:

$$w_d \longleftarrow (1-\mu)w_d - \mu\left[R+\gamma V(x')-V(x)\right] \cdot\left[-\nabla_{\mathbf{w}} V(x)\right] \quad (14)$$

$$w_{rg} \longleftarrow (1-\mu)w_{rg} - \mu\left[R+\gamma V(x')-V(x)\right] \cdot\left[\nabla_{\mathbf{w}}\gamma V(x')-\nabla_{\mathbf{w}} V(x)\right] \quad (15)$$

where μ is a small, positive constant that governs how fast the system forgets. At any given time, a stable ϕ for use in equation (13) can be found that ensures convergence while perhaps maintaining fast learning, by using equation (16):

$$\phi = \frac{\sum_w w_d\, w_{rg}}{\sum_w (w_d - w_{rg})w_{rg}} + \mu \quad (16)$$

6 STOCHASTIC MDPS AND MODELS

The residual algorithm for incremental value iteration in equations (13) and (16) was derived assuming a deterministic MDP. This algorithm does not require that model of the MDP be known, and it has guaranteed convergence to a local minimum of the mean squared Bellman residual. If the MDP were nondeterministic, then the algorithm would still be guaranteed to converge, but it might not converge to a local minimum of the mean squared Bellman residual, as was noted by Werbos (1990). This might still be a useful algorithm, however, because the weights will still converge, and the error in the resulting policy may be small if the MDP is only slightly nondeterministic (deterministic with only a small amount of added randomness).

For a nondeterministic MDP, convergence to a local minimum of the Bellman residual is only guaranteed by using equation (17), which also reduces to (13) in the case of a deterministic MDP:

$$\Delta w = -\alpha\left[R+\gamma V(x_1')-V(x)\right] \left[\phi\gamma \tfrac{\partial}{\partial w} V(x_2') - \tfrac{\partial}{\partial w} V(x)\right] \quad (17)$$

Given a state x, it is necessary to generate two successor states, x_1' and x_2', each drawn independently from the distribution defined by the MDP. This is necessary because an unbiased estimator of the product of two random variables can be obtained by multiplying two independently-generated unbiased estimators. These two independently successor states are easily generated if a model of the MDP is known or is learned. It is also possible to do this without a model, by storing a number of state-successor pairs that are observed, and learning in a given state only after it has been visited twice. This might be particularly useful in a situation where the learning system controls the MDP during learning. If the learning system can intentionally perform actions to return to a given state, then this might be an effective learning method. In any case, it is never necessary to learn the type of detailed, mathematical model of the MDP that would be required by backprop through time, and it is never necessary to perform the types of integrals over successor states required by value iteration. It appears that residual algorithms often do not require models of any sort, and on occasion will require only a partial model, which is perhaps the best that can be done when working with completely-general function-approximation systems.

7 MDPS WITH MULTIPLE ACTIONS

Residual algorithms can also be derived for reinforcement learning on MDPs that provide a choice of several actions in each state. The derivation process is the same. Start with a reinforcement learning algorithm that has been designed for use with a lookup table, such as Q-learning. Find the equation that is the counterpart of the Bellman equation. This should be an equation whose unique solution is the optimal function that is to be learned. For example, the counterpart of the Bellman equation for Q-learning is:

$$Q(x,u) = \left\langle R + \gamma \max_{u'} Q(x',u') \right\rangle \quad (17)$$

For a given MDP with a finite number of states and actions, there is a unique solution to equation (17), which is the optimal Q-function. The equation should be arranged such that the function to be learned appears on the left side, and everything else appears on the right side. The direct algorithm is just backpropagation, where the left side is the output of the network, and the right side is used as the "desired output" for learning. Given the counterpart of the Bellman equation, the mean squared Bellman residual is the average squared difference between the two sides of the equation. The residual gradient algorithm is simply gradient descent on E, and the residual algorithm is a weighted sum of the direct and residual gradient algorithms, as defined in equation (9).

8 RESIDUAL ALGORITHM SUMMARY

Most reinforcement learning algorithms that have been suggested for prediction or control have associated equations that are the counterparts of the Bellman

Table 1. Four reinforcement learning algorithms, the counterpart of the Bellman equation for each, and each of the corresponding residual algorithms..

Reinforcement Learning Algorithm	Counterpart of Bellman Equation (top) Weight Change for Residual Algorithm (bottom)
TD(0)	$V(x) = \langle R + \gamma V(x') \rangle$ $\Delta w_r = -\alpha\left(R + \gamma V(x'_1) - V(x)\right)\left(\phi\gamma \frac{\partial}{\partial w} V(x'_2) - \frac{\partial}{\partial w} V(x)\right)$
Value Iteration	$V(x) = \max_u \langle R + \gamma V(x') \rangle$ $\Delta w_r = -\alpha\left(\max_u \langle R + \gamma V(x') \rangle - V(x)\right)\left(\phi \frac{\partial}{\partial w} \max_u \langle R + \gamma V(x') \rangle - \frac{\partial}{\partial w} V(x)\right)$
Q-learning	$Q(x,u) = \left\langle R + \gamma \max_{u'} Q(x',u') \right\rangle$ $\Delta w_r = -\alpha\left(R + \gamma \max_{u'} Q(x'_1,u') - Q(x,u)\right)\left(\phi\gamma \frac{\partial}{\partial w} \max_{u'} Q(x'_2,u') - \frac{\partial}{\partial w} Q(x,u)\right)$
Advantage Learning	$A(x,u) = \left\langle R + \gamma^{\Delta t} \max_{u'} A(x',u') \right\rangle \frac{1}{\Delta t} + (1 - \frac{1}{\Delta t}) \max_{u'} A(x,u')$ $\Delta w_r = -\alpha\left(\left(R + \gamma^{\Delta t} \max_{u'} A(x'_1,u')\right) \frac{1}{\Delta t} + (1 - \frac{1}{\Delta t}) \max_{u'} A(x,u') - A(x,u)\right)$ $\cdot\left(\phi\gamma^{\Delta t} \frac{\partial}{\partial w} \max_{u'} A(x'_2,u') \frac{1}{\Delta t} + \phi(1 - \frac{1}{\Delta t}) \frac{\partial}{\partial w} \max_{u'} A(x,u') - \frac{\partial}{\partial w} A(x,u)\right)$

equation. The optimal functions that the learning system should learn are also unique solutions to the Bellman equation counterparts. Given the Bellman equation counterpart for a reinforcement learning algorithm, it is straightforward to derive the associated direct, residual gradient, and residual algorithms. As can be seen from Table 1, all of the residual algorithms can be implemented incrementally except for residual value iteration. Value iteration requires that an expected value be calculated for each possible action, then the maximum to be found. For a system with a continuum of states and actions, a step of value iteration with continuous states would require finding the maximum of an uncountable set of integrals. This is clearly impractical, and appears to have been one of the motivations behind the development of Q-learning. Table 1 also shows that for a deterministic MDP, all of the algorithms can be implemented without a model, except for residual value iteration. This may simplify the design of a learning system, since there is no need to learn a model of the MDP. Even if the MDP is nondeterministic, the residual algorithms can still be used without a model, by observing x'_1, then using $x'_2 = x'_1$. That approximation still ensures convergence, but it may force convergence to an incorrect policy, even if the function-approximation system is initialized to the correct answer, and the initial mean squared Bellman residual is zero. If the nondeterminism is merely a small amount of noise in a control system, then this approximation may be useful in practice. For more accurate results, it is necessary that x'_1 and x'_2 be generated independently. This can be done if a model of the MDP is known or learned, or if the learning system stores tuples (x,u,x'), and then changes the weights only when two tuples are observed with the same x and u. Of course, even when a model of the MDP must be learned, only two successor states need to be generated; there is no need to calculate large sums or integrals as in value iteration.

9 SIMULATION RESULTS

Figure 5 shows simulation results. The solid line shows the learning time for the star problem in figure 1, and the dotted line show the learning time for the hall problem in figure 2. In the simulation, the direct method was unable to solve the star problem, and the learning time appears to approach infinity as ϕ approaches approximately 0.1034. The optimal constant ϕ appears to lie between 0.2 and 0.3. The adaptive ϕ was able to solve the problem in time close to the optimal time, while the final value of ϕ at the end was approximately the same as the optimal constant ϕ. For the hall problem from figure 2, the optimal algorithm is the direct method, $\phi = 0$. In this case, the adaptive ϕ was able to quickly reach $\phi = 0$, and therefore solved the problem in close to optimal time.

In each case, the learning rate was optimized to two significant digits, through exhaustive search. Each data point was calculated by averaging over 100 trials, each with different initial random weights. For the adaptive methods, the parameters μ=0.001 and ε=0.1 were used, but no attempt was made to optimize them. When adapting, ϕ initially started at 1.0, the safe value corresponding to the pure residual gradient method.

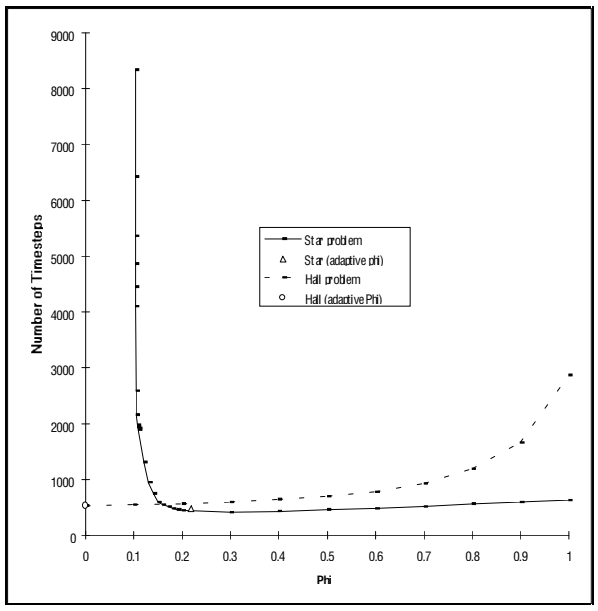


Figure 5. Simulation results for two MDPs

The lines in figure 5 clearly show that the direct method can be much faster than pure residual gradient algorithms in some cases, yet can be infinitely slower in others. The square and triangle, representing the residual gradient algorithm with adaptive ϕ, demonstrate that the algorithm is able to automatically adjust to the problem at hand and still achieve near-optimal results, at least for these two problems.

10 CONCLUSIONS

A new class of algorithms, *residual algorithms*, has been proposed for performing reinforcement learning with function approximation systems. The shortcomings of both direct and residual gradient algorithms have been shown. It has also been shown, both analytically and in simulation, that direct and residual gradient algorithms are special cases of residual algorithms, and that residual algorithms can be found that combine the beneficial properties of both. This allows reinforcement learning to be combined with general function-approximation systems, with fast learning, while retaining guaranteed convergence.

Acknowledgments

This research was supported by the Department of Computer Science, U.S. Air Force Academy, and by Task 2312R102 of the Life and Environmental Sciences Directorate of the United States Office of Scientific Research. This work benefited greatly from many stimulating conversations with Rich Sutton, Mance Harmon, Harry Klopf, Gabor Bartha, Jim Morgan, and particularly Ron Williams, who first noted the slow convergence of pure residual gradient algorithms.

References

Baird, L. C. (1995). *Advantage Learning*. To be published as a U.S. Air Force technical report by the Department of Computer Science, U.S. Air Force Academy.

Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Englewood Cliffs, NJ: Prentice-Hall.

Bradtke, S. J (1993). Reinforcement learning applied to linear quadratic regulation. *Proceedings of the Fifth Conference on Neural Information Processing Systems* (pp. 295-302). Morgan Kaufmann.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature*. **323**, 9 October, 533-536.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, **3**, 9-44.

Tesauro, G. (1990). Neurogammon: A neural-network backgammon program. *Proceedings of the International Joint Conference on Neural Networks* **3** (pp. 33-40). San Diego, CA.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, **8**(3/4), 257-277.

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

Watkins, C. J. C. H., & Dayan, P. (1992). Technical note: Q-learning. *Machine Learning*, **8**(3/4), 279-292.

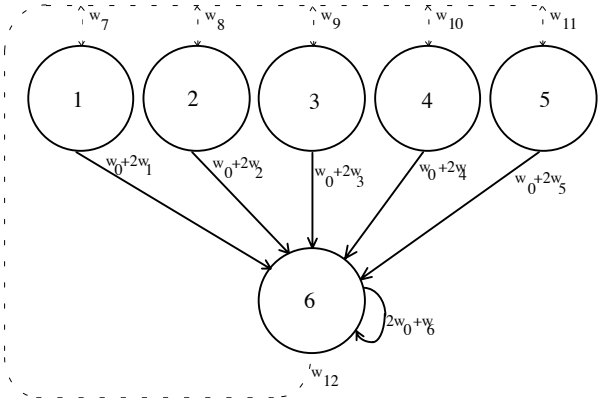
Werbos, P. J. (1990). Consistency of HDP Applied to a Simple Reinforcement Learning Problem. *Neural Networks* **3**, 179-189.

Williams, R. J., and Baird, L. C. (1993). *Tight Performance Bounds on Greedy Policies Based on Imperfect Value Functions*. Northeastern University Technical Report NU-CCS-93-14, November.

Errata

15 Jul 95

This paper contains 3 corrections of errors that appeared in the original version, as published in the proceedings of the Machine Learning Conference, 1995. I appreciate Mance Harmon pointing out the formatting problems that are fixed here, Justin Boyan pointing out that figure 1 was incorrect (fixed by adding the coefficients of 2), and Geoff Gordon pointing out that a sentence on page 3, column 2, paragraph 2 was incorrect (which was fixed by changing "continuous" to "differentiable"). Also, an additional example was added for the conference presentation. A modified star problem showed an MDP rather than a Markov chain:



In this MDP, every transition receives zero reinforcement, and each state has two actions, one represented by a solid line, and one represented by a dotted line. In all states, the solid action transitions to state 6, and the dotted action transitions to one of the states 1 through 5, chosen randomly with uniform probability. During training, a fixed stochastic policy is used to ensure sufficient exploration. In every state, the solid action is chosen with probability 1/6, and the dotted action is chosen with probability 5/6. This ensures that every state-action pair is explored infinitely often, and that each of the solid Q values is updated equally often. If the solid Q-values start larger than the dotted Q-values, and the trasition from state 6 to itself starts out as the largest of the solid Q-values, then all weights, Q-values, and values will diverge to infinity. This is true for both epoch-wise and incremental learning, and even for small learning rates or slowly decreasing learning rates. This example demonstrates that for a simple MDP with a linear function approximator able to represent all possible Q-functions, the Q-values can diverge, even when training on trajectories. None of the Q-learning or TD(0) theorems can guarantee convergence of Q-learning in this case, but residual Q-learning is guaranteed to converge with epoch-wise training.