More on Training Strategies for Critic and Action Neural Networks in Dual Heuristic Programming Method

George G. Lendaris[1], Christian Paintz[2], Thaddeus Shannon[3]
1.Professor, Systems Science & Electrical Engineering, lendaris@sysc.pdx.edu
2. Graduate Student, Electrical Engineering, 3. Graduate Student, Systems Science Ph.D. Program
Portland State University
PO Box 751, Portland, OR 97207

ABSTRACT

This paper for the special session on Adaptive Critic Design Methods at the SMC '97 Conference describes a modification to the (to date) usual procedures reported for training the Critic and Action neural networks in the Dual Heuristic Programming (DHP) method [7]-[12]. This modification entails updating both the Critic and the Action networks each computational cycle, rather than only one at a time. The distinction lies in the introduction of a (real) second copy of the Critic network whose weights are adjusted less often (once per "epoch", where the epoch is defined to comprise some number N>1 computational cycles), and the "desired value" for training the other Critic is obtained from this Critic-Copy.

In a previous publication [4], the proposed modified training strategy was demonstrated on the well-known pole-cart controller problem. In that paper, the full 6 dimensional state vector was input to the Critic and Action NNs, however, the utility function only involved pole angle, not distance along the track (x). For the first set of results presented here, the 3 states associated with the x variable were eliminated from the inputs to the NNs, keeping the same utility function previously defined. This resulted in improved learning and controller performance. From this point, the method is applied to two additional problems, each of increasing complexity: for the first, an x-related term is added to the utility function for the pole-cart problem, and simultaneously, the x-related states were added back in to the NNs (i.e., increase number of state variables used from 3 to 6); the second relates to steering a vehicle with independent drive motors on each wheel. The problem contexts and experimental results are provided.

1. BACKGROUND

Dual Heuristic Programming (DHP) is a neural network approach to solving the Bellman equation [12]. The idea is to maximize a specified (*secondary*) utility function:

$$J(t) = \sum_{k=0}^{\infty} \gamma^k U(t+k) \qquad (1)$$

The term γ^k is a discount factor ($0 < \gamma < 1$) and $U(t)$ is the *primary* utility function, defined by the user for the specific application context. A useful identity:

$$J(t) = U(t) + \gamma J(t+1) \qquad (2)$$

In this paper, γ is assumed to be 1, and the usual methods of discretizing continuous models of plants is used. For DHP, at least two neural nets are needed, one for the ***action***NN functioning as the controller, and one for the ***critic***NN used to train the *action*NN. A third NN could be trained to copy the plant if an analytical description (model) of the plant is not available.

R(t) [dimension n] is the state of the plant at time t. The control signal **u**(t) [dimension a] is generated in response to the input **R**(t) by the *action*NN. The signal **u**(t) is then asserted to the plant. As a result of this, the plant changes its state to **R**(t+1). The *critic*NN's role is to assist in designing a controller (*action*NN) that is "good" relative to minimizing the specified cost function U(**R**(t),**u**(t)), which is designed to expresses the objective of the control application. In the DHP method, the *critic*NN estimates the gradient of J(t) with respect to **R**(t); the letter λ is used as a short-hand notation for this gradient, so the output of the *critic*NN is designated λ.

2. UPDATING PROCESS

We refer to Figure 1 to describe the computational steps used in the DHP methodology. First, we mention that the boxes with dark shading mean that we have an analytical expression for that item [note: if an analytical representation of the plant is not available, a NN may be trained up to emulate the plant and used in this role)]. The clear boxes are neural networks (NNs). The medium shaded boxes represent some critical equations that are to be solved in the training process. The dotted lines represent calculated values being fed into the respective boxes. The heavier dot-dash lines indicate where the learning/updating process occurs.

Reading Figure 1 from left to right, the current state **R**(t) is fed to the *action*NN, which then generates **u**(t). The model is informed of **R**(t) and **u**(t), and then generates **R**(t+1). **R**(t) is also fed to the critic#1 box and to the utility box, which generate, respectively, λ(**R**(t)) and U(**R**(t)). [Note, in the examples used herein, the Utility functions defined make use of **R**(t) but not **u**(t); if **u**(t) is included in the definition of the utility U, then we would show a connection from the output of the *action*NN to the utility box.] After the model generates **R**(t+1), this is fed to the critic#2 box, which then yields λ(**R**(t+1)) -- more simply denoted as λ(t+1). This value is a key component of the calculations in the medium-shaded boxes, which in turn

For IEEE SMC'97, Orlando, FL, Oct. 1997.

are needed to perform learning/adaptation of the *action*NN and the *critic*NN. The upper medium-shaded box calculates Δw_{ij}, and the lower medium-shaded box calculates $\lambda^{\circ}(t)$, the "desired" or "target" value for $\lambda(t)$ to enable backpropagation-type training of critic#1.

We now show the basic equations for the two medium-shaded boxes, and in particular, point out the role of $\lambda(t+1)$ in those computations.

2.1 The upper medium-shaded box.

In the present work, a basic Backpropagation algorithm is used (no embellishments) to adjust the weights in the action box. The weight-adjustment is calculated via:

$$\Delta w_{ij}(t) = lcoef \bullet \frac{\partial}{\partial w_{ij}(t)} J(t) \qquad (3)$$

where $\frac{\partial}{\partial w_{ij}(t)} J(t) = \sum_{k=1}^{u} \frac{\partial}{\partial u_k(t)} J(t) \bullet \frac{\partial}{\partial w_{ij}(t)} u_k(t)$

and $\frac{\partial}{\partial u_k(t)} J(t) = \frac{\partial}{\partial u_k(t)} U(t) + \frac{\partial}{\partial u_k(t)} J(t+1)$

and finally,

$$\frac{\partial}{\partial u_k(t)} J(t+1) = \sum_{s=1}^{n} \frac{\partial}{\partial R_s(t+1)} J(t+1) \bullet \frac{\partial}{\partial u_k(t)} R_s(t+1) \qquad (4)$$

Abbreviation: $\frac{\partial}{\partial R_s(t+1)} J(t+1) = \lambda(t+1) \qquad (5)$

$\lambda(t+1)$ is approximated by the critic, in response to the input **R**(t+1).

$\frac{\partial}{\partial u_k(t)} R_s(t+1)$ can be calculated from analytical equations of the plant, if they are available, or by backpropagation through a third neural net that has been previously trained to copy the plant.

2.2 The lower medium-shaded box.

In [4], the following equation is shown to hold:

$$\lambda_s^{\circ}(t) = \frac{\partial}{\partial R_s(t)} U(t) + \sum_{j=1}^{u} \left(\frac{\partial}{\partial u_j(t)} U(t) \bullet \frac{\partial}{\partial R_s(t)} u_j(t) \right) \qquad (6)$$
$$+ \sum_{k=1}^{n} \left(\frac{\partial}{\partial R_k(t+1)} J(t+1) \bullet \frac{\partial}{\partial R_s(t)} R_k(t+1) \right)$$
$$+ \sum_{k=1}^{n} \left\{ \sum_{j=1}^{a} \left(\frac{\partial}{\partial R_k(t+1)} J(t+1) \bullet \frac{\partial}{\partial u_j(t)} R_k(t+1) \bullet \frac{\partial}{\partial R_s(t)} u_j(t) \right) \right\}$$

which makes use of the identity of Equation 2. Note again the term containing the gradient of J(t+1) with respect to **R**(t+1): $\lambda(t+1)$. The subscript s indicates the components of the *vector* quantity $\lambda(t+1)$.

For convenience, we paraphrase Equation (6) in Equation (7). Using this version, we note that the *[~Action]* terms are calculated via the action box in Figure 1, and the *[~Critic(t+1)]* terms are calculated via the critic#2 box. Similarly, the *[~Plant]* and the *[~Utility]* terms are calculated via the respective dark-shaded box in Figure 1.

$$\lambda_s^{\circ}(t) = [\sim Utility] + \sum_{j=1}^{u} ([\sim Utility] \bullet [\sim Action])$$
$$+ \sum_{k=1}^{n} ([\sim Critic(t+1)] \bullet [\sim Plant])$$
$$+ \sum_{k=1}^{n} \left\{ \sum_{j=1}^{a} ([\sim Critic(t+1)] \bullet [\sim Plant] \bullet [\sim Action]) \right\} \qquad (7)$$

2.3 The training/update process.

Detailed descriptions of various strategies were given in [4] for "solving" (iterating) Equation 7. We describe some of them here (more loosely) using Figure 1. Keep in mind that the output of critic#2 is required for performing the calculations in the medium-shaded boxes, which in turn must be calculated to perform learning updates in the action and critic#1 boxes.

Strategy 1. Straight application of the equation.

In Figure 1, this means that after $\lambda(t+1)$ is calculated, **both** of the paths leaving critic#2 are traversed, so that the action box and the critic#1 box are updated in each iteration [in this strategy, the two boxes labeled critic#1 and critic#2 are always maintained identical -- i.e., could be the same physical box, just used for two different calculations].

Strategy 2a. Basic 2-stage process ["flip/flop"].[7]-[12]

During stage 1, train *critic*NN, not *action*NN;

In Figure 1, this means that after $\lambda(t+1)$ is calculated, only the path which adapts critic#1 is traversed, not the path which adapts the action box. This is repeated for a designated number of iterations, and then changed to stage 2 (from "flip" to "flop"). As in Strategy 1, critic#1 ≡ critic#2.

During stage 2, train *action*NN, not *critic*NN**.**

In Figure 1, this means that after $\lambda(t+1)$ is calculated, only the path which adapts the action box is traversed, not the path which adapts critic#1. This is repeated for a designated number of iterations, and then changed to stage 1 (from "flop" to "flip").

Strategy 3a. Modify 1st stage of the 2-stage process.

The modification here is to make critic#1 and critic#2 **two physically distinct objects**. Then, during Stage 1, after $\lambda(t+1)$ is calculated, adapt critic#1 as in Strategy 2, however, **leave critic#2 unchanged**. Repeat this for the designated number of iterations for Stage 1 ("flip" portion) of the process. Then, just before moving into Stage 2 (the "flop" portion), upload the weight values from critic#1 into critic#2, and then proceed to Stage 2.

Strategy 4. Single-stage process (as in Strategy 1),but use modifications introduced in Strategy 3.

Strategy 1 is modified to make critic#1 and critic#2 **two physically distinct objects** (as in Strategy 3).

4a. After $\lambda(t+1)$ is calculated, adapt both, the action box and the critic#1 box as in Strategy 1, however, **leave critic#2 unchanged**. Repeat this for a designated number of iterations (the familiar term '**epoch**' is used here for the designated number of iterations), and *at the end of each epoch, upload the weight values from critic#1 into critic#2*, and continue the process, epoch at a time.

4b. [Synopsis: while this version also adapts critic#1 via critic#2, in contrast to **4a** this strategy ***adapts the action box via critic #1.***] After $\lambda(t+1)$ is calculated via critic#2, do the bottom medium-shaded box [calculate $\lambda^{\circ}(t)$ for adapting critic#1] as in **4a**; however, now do something not shown in Figure 1: apply $\mathbf{R}(t+1)$ to critic#1, and use *the resulting* $\lambda(t+1)$ *value* to enter the upper medium-shaded box, and proceed to adapt the action box. Now apply $\mathbf{R}(t)$ to critic#1, and based on the $\lambda^{\circ}(t)$ already calculated, adapt critic#1. As in **4a**, at the end of the epoch, upload the weight values from critic#1 into critic#2.

2.4 Benefits of the modifications.

The modified strategies described above have demonstrated a benefit of allowing increased learning rates, thus reducing the convergence time for the learning process.

In [4], comparative training results were shown for all the strategies described. The well known pole-cart control problem [3], with the pole angle $\Theta(t)$ being controlled, was used as the test application. Strategy 4 was demonstrated to yield the fastest convergence, and among the best controller designs. In that paper, the full 6 dimensional state vector was input to the Critic and Action NNs, however, the utility function only involved pole angle, not distance along the track (x).

In the present paper, the 3 states associated with the x variable were first eliminated from the inputs to the NNs, keeping the same utility function previously defined. This resulted in improved learning and controller performance. Experimental results are given in Section 4.

Next, two applications of increased complexity are investigated, and comparative results are presented. This time, the exploration is confined to Strategies 1, 2a & 4. Strategy 1 is used as the baseline; next, Strategy 2a (cf. [7]-[12]) is shown with its relative performance; and finally, the new Strategies 4a & 4b are demonstrated with their relative improvements over both, Strategies 1 and 2a.

The two additional demonstration applications are:
a) the pole-cart problem again, except distance-along-the-track (x) is added in the utility function -- thus increasing the number of system states *needed* by the critic box from 3 to 6; b) steering control of a 4-wheel vehicle with independent electric motors on each wheel.

3. TEST-BED APPLICATIONS

For the pole-balancer test bed [3][4], the training procedure was to randomly initialize all the NNs (weight range: [-01, 01]), and to then provide a specified sequence of starting angles (with zero being the "desired" angle), allowing the system to train on each starting angle for a specified number of seconds. The measure used for comparing the various DHP strategies takes the values achieved by the primary utility function during training and accumulates these over the sequence of starting angles [this part is called C(j)], and add to this a penalty term according to the number of times the pole was dropped and had to be reset [this part is called D(j)], yielding a total measure M(j). The argument j labels a separate pass through the sequence of angles. In a sense, this measure incorporates the convergence speed of the strategy as well as the quality of the controller's actions along the way.

3.1 Speed of training convergence.

As is well known, convergence speed of the training process is directly influenced by the values of the learning coefficients [interchangeably called "gains" here]; if the gains are too high, the process does not converge; if set too low, while convergence may be achieved, the process takes a long time. There is an intermediate range of gain values for which the system may or may not converge; this is explored by re-initializing the system and re-starting the process. Depending on the test, we ran up to 100 trials to determine a kind of "probability of convergence".

We approximately determined for each strategy what range of gains resulted in convergence with probability very close to one. A set of gains was selected from these for each strategy, and a series of trials was run. As the gains were increased, at some point the convergence probability dropped significantly. The strategy being tested was determined to converge with probability close to 1 if the learning process converged to a controller capable of balancing the pole with zero drops through one complete pass through the training angles for at least 39 out of 40 trials. The gain values were then increased or decreased until the approximate edge of the "almost-sure" convergence region was determined (i.e., the highest values of the gains just before the probability started dropping away from one -- sort of an "edge" condition;e.g., $99\%^{+}$ for gains of .30, but much less for .35). We dubbed these the Edge Gain Values, recognizing of course that these were only approximately determinable.

The reason for going through this process is to provide a kind of "optimal" design for each of the training procedures, so they might be compared more fairly. A nice set of (approximate) Edge Gain Values were determined for the pole-cart "theta-only" and "theta-x" versions. Only partial results along this line were achieved with the steering control problem.

3.2 Pole-Cart test bed.

In [4], the actionNN and the criticNN for the pole-cart test bed were both provided the full state vector R(t) at

their input [as has been typical in the literature]. In our exploration for the present research to determine the Edge Gain Values, none could be discovered. This led to reducing the complexity of the action and critic NNs by eliminating from their inputs those states that are not used in the Utility function (i.e., not used in stipulating the objectives of the control task). Since $U(t) = (\Theta(t) - 0)^2$ was used [angle measured from vertical; target value of 0] only the $\Theta(t), \dot{\Theta}(t), \ddot{\Theta}(t)$ states are needed by the critic and action NNs. After removing the x(t)-related states for the present research, we were able to discover for each of the strategies a region of gains for which "probability one" convergence would occur. The Edge Gain Values of these regions for each strategy were determined, and used for a new set of comparative results, given in Section 4.

For the next level of complexity on which to explore the proposed new strategies, a utility function was defined for the same pole-cart problem which includes a distance-along-the-track term, x, with target position x=0:

$$U(t) = (\Theta(t) - 0)^2 + (x(t) - 0)^2 .$$

Since this equation includes both angle and distance terms, the critic and action NNs will profit from having both the 3 $\Theta(t)$-related and the 3 x(t)-related states fed into them. This was done, and again, a region of gains for which "probability one" convergence would occur was sought for each of the training strategies. Where successful, the Edge Gain Values for each strategy were determined, and used for the comparative results given in Section 4.

3.3 Steering control for 4-wheel vehicle test bed.

For a more complex plant, we turn to a 4-wheel vehicle with an independent electric drive motor on each wheel. The equations and documentation for the steering model were extracted from [1] which in turn were based on equations developed in [5].

A full nonlinear model for the vehicle is used along with the classical constant-velocity bicycle model [5] which was used in [2] for developing a front/rear wheel steering decoupling system. A front-wheel-only steering model is obtained via restricting rear-wheel steering angle to zero. The key unknown system parameters for which robustness is required are vehicle mass and tire/road side-slip coefficient of friction.

Figure 2 shows the assumed geometry. Reference coordinates: x_o-y_o; chassis coordinates: x-y, rotated by angle ψ. Mass is assumed concentrated at front and back wheels, with total mass m; wheelbase is $l = l_f + l_b$. Vehicle velocity at center of gravity has magnitude v, at angle β from x direction. Velocity v is assumed to be constant; β varies through a turn. At each wheel, steering angle is δ, subscripted by f & b for front and back. Velocity of chassis at the ith axle, i={f,b}, has assumed magnitude v_i, at angle β_i from the x axis. Thus, the wheel side-slip angle is $\alpha_i = \delta_i - \beta_i$, which represents the angular difference between the direction in which the wheel is pointed and the direction in which it is actually moving (due to slipping). This, in turn, produces a side force on the wheel with magnitude $f_i = f(\alpha_i)$ in the direction perpendicular to that in which the wheel is steered, where f_i is a monotonic function of α_i.

In [1], a set of kinematic, force and dynamic equations are developed, and these are combined to define a set of second order nonlinear state equations [bicycle steering model]. The state variables are β and ρ [where $\rho = \dot{\psi}$], and the inputs are front and back wheel steering commands, δ_f and δ_b. For the hierarchical control structure defined in [1], the steering control system is required to generate velocity commands for the wheel controllers. In Figure 3, these are shown as the outputs u_f and u_b. The symbols f_y and m_z represent, the tire side force in the y direction and the sum-of-the-moments generated by the two tire side forces around the center of gravity.

A Utility function was defined by the present authors for applying the DHP method as follows:

$$U(t) = -(ypos_{des} - ypos_{act})^2 - .5(yvel_{des} - yvel_{act})^2$$

As discussed earlier, since this utility function involves both position and velocity in the y direction, the associated state variables are fed to the action and critic NNs.

The preliminary experiments reported here are for performing a lane change, which entails a left-turn plus a right-turn type maneuver. The DHP method was used to design a controller for this task. It was decided to train up a separate NN for each of the left-turn and right-turn parts of the lane-change maneuver. These were then (easily) "pieced together" with some simple logic. After succeeding with this procedure using Strategy 4a, subsequent experiments performed for comparative purposes used only the left-turn portion of the lane-change task.

4. EXPERIMENTAL RESULTS

4.1 Pole-Cart test bed.

Results are tabulated in Tables I, II and III, for both the "Theta-Only" and the "Theta-X" versions of the Pole-Cart control test bed.

In both cases, we start with selecting the highest gains for which Strategy 1 has "probability one" convergence. This "optimal" performance for Strategy 1 provides a benchmark for comparisons. See Table III for gains. The value recorded in each row of columns 2 and 3 of Tables I & II is the M measure cited earlier. M is defined here as the sum C(1)+C(2)+C(3)+2*(D(1)+D(2)+D(3)), thus invoking a 2.0 utility penalty for a restart after a drop, a relatively inexpensive assumption.

Theta Only version.

When we use the Strategy 1 Edge Gains for Strategy 2a (column 2 in Table I) performance is worse than for Strategy 1, particularly relative to the number of "drops".

When the Strategy 1 Edge Gains are used for Strategies 4a and 4b, relative to each other, they perform approximately the same; however, relative to Strategy 1, we note the following: the standard error for the Strategy 1 mean cost is ±25%, whereas for Strategy 4 it is less than ±2%. The sample mean itself improves by approx. 40%, with the "drops" for the three methods being approx. the same (left number in column 4, Table I).

When the "optimal" Edge Gains for Strategies 4a and 4b are used (column 3 in Table I), there is another factor of 2 improvement in the total cost, with the number of "drops" going from approx. 17 down to 6.

These experimental results give more substantive empirical support for the (more qualitative) conclusions given in the previous paper [4].

Theta-X version.

When we used the Strategy 1 Edge Gains for Strategy 2a, a "stable region" of convergence could not be found -- the closest "probability of convergence" found was approximately .9.

When the Strategy 1 Edge Gains are used for Strategies 4a and 4b, relative to each other, they again perform approximately the same. Relative to Strategy 1, we note the following: the standard error is roughly the same for all three methods, but in this case, the total cost goes up slightly for Strategy 4 (using Strategy 1's gains).

When the "optimal" Edge Gains for Strategies 4a and 4b are used, however, Strategy 4 does provide improved results. As in the previous problem, the sample mean itself again improves approximately 40%, with the "drops" for Strategy 4a showing substantial improvement: from 58 --> 33, and for Strategy 4b from 58 -->41.

TABLE I: Theta-Only version of Pole-Cart problem.
TABLE II: Theta-X version of Pole-Cart problem.
Columns 2 & 3 are total cost [reported as sample mean ±standard error]. Column 4 gives average number of drops for the two gains.

Strategy	via Strategy 1 Edge Gains	via corresp. Edge Gains	$D(1)_{S1}/D(1)_{TEG}$
1	206 ± 53	206 ± 53	18 / 18
2a	239 ± 2.5	226 ± 3.5	45 / 42
4a	115 ± 2.0	**49** ± 1.0	16 / **6**
4b	128 ± 2.5	**48** ± 1.0	17 / **6**

Strategy	via Strategy 1 Edge Gains	via corresp. Edge Gains	$D(1)_{S1}/D(1)_{TEG}$
1	349 ± 11	349 ± 11	58 / 58
2a	not run	none found	-----
4a	394 ± 7.5	**207** ± 9.5	62 / **33**
4b	390 ± 10.5	**230** ± 9.5	67 / **41**

TABLE III: Edge Gains (learning coefficients) for Pole-Cart

Strategy	Theta-Only critic/action gains	Theta-X critic/action gains
1	.07/.25	.02/.2
2a	.07/1.0	none
4a	.3/.9	.04/.4
4b	.3/.9	.03/.4

These experiments again support the conclusion that Strategy 4, with its ability to utilize higher learning rates (gains), is able to provide better performance.

4.1 Four-wheel vehicle Steering test bed.

As with the pole-cart problem, experiments were run with strategies 1, 2a and 4. We were not able to find a region of "probability one" convergence for the car as we had been able to do for the pole-cart. In the quest to improve the situation, it was reasoned that since the utility function includes desired values for y and $\dot{y}$, and the fact that these change with time, it would be difficult for the critic to infer these changing targets based on indirect information. Therefore, it was decided to directly input target values for these two quantities to the critic (as well as to the action NN). The situation improved, such that for Strategy 4b, the "probability of convergence" reached .7-.8. For the other strategies, it came up to a range of .3-.5. We discovered again that it was better to leave out the bias terms in both NNs (we gave a similar observation in [4]).

In order to achieve convergence in the DHP process for this kind of problem, it was found useful to adapt a training method proposed in [6]. This method divides the state space into portions. The NNs start to train on the smallest fraction, here defined as a unit of time along the trajectory to be learned. Then, after successful learning to control the plant in that unit, the NNs are trained on two units of time, the fraction already learned and another one of the same size. After learning this bigger fraction of state space, a third one is added, etc., until the full expected range is successfully trained.

When Strategy 1 converges (relatively rare occurrence), it does so for only small gain values. Critic/Action gain values of .01/.03 gave reasonably good results (in terms of the total "cost" accumulated during training). On the other hand, gain values about a factor of 2 higher typically yielded divergent training. Also, for this application, using extremely low gain values also diverged (couldn't keep up with the changing desired trajectory?).

Strategy 2a appeared to have a (slightly) higher "probability of convergence" for the gains explored than did Strategy 1. Nevertheless, the accumulated cost was still about an order of magnitude higher than achieved with Strategies 4a and especially 4b.

Strategy 4a and 4b had substantially higher convergence probability, as indicated above.

We are not able at this time to give the kind of compar-

ative statistical data given for the pole-cart. The process of following a desired trajectory is substantially more complex for us to capture the kinds of measures that were done with the pole-cart.

A potentially important observation was that, using Strategy 4b (the fastest strategy), even after successful training, allowing the critic to continue to operate during a test run yielded lower total cost than was achieved with just the trained controller doing a "solo" run over the trajectory. This has good implications for the possibility of on-line training. A first hint in this direction was reported in [4] with the pole-cart in the context of increasing pole length from 1m to 2.4m, and the critic adapted the action NN, on-line, without dropping the pole.

5. CONCLUSION

The proposed modification to the training strategy for the DHP method continues to demonstrate improved learning performance over the more prevalent method, here called Strategy 2a. The proposed Strategies 4a & 4b were first evolved while working on the "theta-only" version of the Pole-Cart control problem [4], and now further demonstrated on the more complex "theta-x" version of the Pole-Cart problem, and thirdly, on an even more complex dynamical system, steering a 4-wheel vehicle.

These explorations also demonstrated to us that the specific information provided to the critic and action boxes can have a significant impact on performance. In particular, providing the full state vector **R**(t) to the critic may be detrimental, as demonstrated here for the theta-only test bed. For the steering control problem, it was found beneficial to additionally provide the critic with the *desired* (or *target)* values for y and $\dot{y}$. It appears that the choice of what inputs to provide the critic should be guided by the form of the specific utility function being used.

REFERENCES

[1] Accurate Automation Corp., "Advanced Intelligent Control of Next Generation Vehicles", NSF SBIR Phase I Final Report, Aug 1995.

[2] Ackermann, J., "Robust Decoupling, Ideal Steering Dynamics and Yaw Stabil. of 4WS Cars", *Automatica*, vol. 60, pp1761-1768, 1994.

[3] Barto, A., Sutton, R. & Anderson, C. " Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems" in *IEEE SMC Transactions*, Vol. SMC-13, No.5, Sep/Oct 1983.

[4] Lendaris,G. and Paintz, C. "Training Strategies for Critic and Action Neural Nets in Dual Heuristic Programming Method", in *PROCEEDINGS of ICNN'97, Houston*, IEEE, pp712-717, June, 1997.

[5] Mitschke, M., *Dynakik der Kraftfahrzeuge*, vol. C, Springer -Verlag, Berlin, 1990.

[6] Nguyen D. and Widrow, B., "The Truck Backer-Upper: an Example of Self Learning in Neural Networks", Ch 12 in *Neural Networks for Control*, Miller, Sutton & Werbos (eds), MIT Press, 1991.

[7] Prokhorov, D. and Wunsch, D. "Advanced Adaptive Critic Designs", *PROC WCNN'96*, pp. 83-87, San Diego, Erlbaum, Sept. 1996.

[8] Prokhorov, D., Santiago, R. & Wunsch, D., "Adaptive Critic Designs: A Case Study for Neurocontrol", in *Neural Networks*, vol. 8, no. 9, pp 1367-1372, 1995.

[9] Santiago, R., First Joint Mexico-US International Workshop on Neural Networks and Neurocontrol, Playacar, Mexico, Sept. 1995.

[10] Santiago, R. & Werbos, P. "New Progress Towards Truly Brain-Like Intelligent Control", *PROC WCNN '94*, pp. I-2toI-33, Erlbaum, 1994.

[11] Visnevski, N. & Prokhorov, D. "Control of a Nonlinear Multivariable System with Adaptive Critic Designs", in *Intelligent Engineering Systems through Artificial Neural Networks 6* (*PROC. ANNIE '96*), Dagli, et.al., Eds., ASME Press, pp. 559-565, 1996.

[12] Werbos, P. "Approximate Dynamic Programming for Real-Time Control and Neural Modeling", Ch. 13 in *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, (White, D.A. and Sofge, D.A., eds.), Van Nostrand Reinhold, New York, NY, 1994.

Figure 1: Computing Schema for Discussing Strategies.

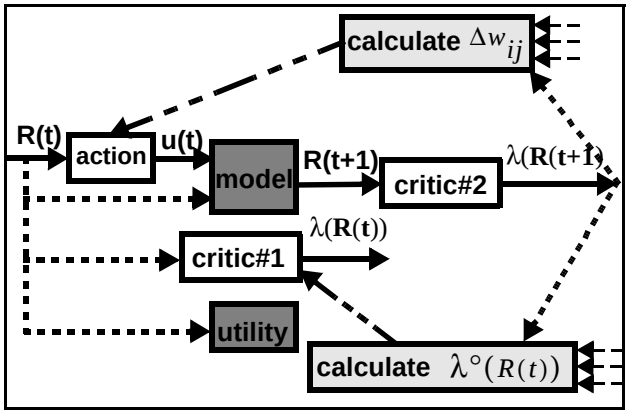


Figure 2. Assumed vehicle model geometry (after [1]).

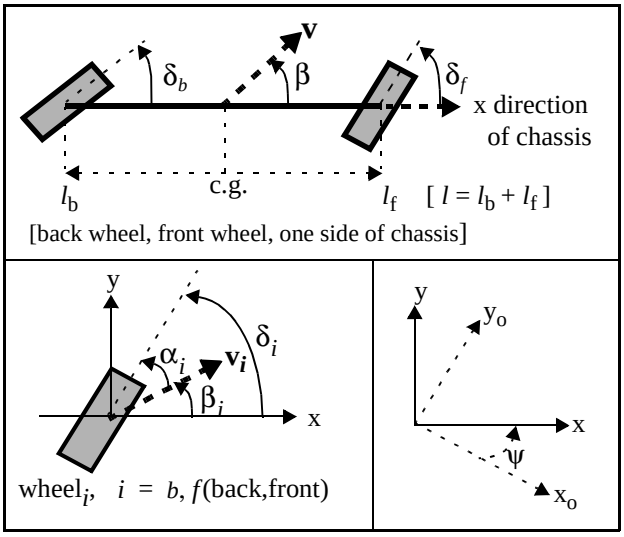
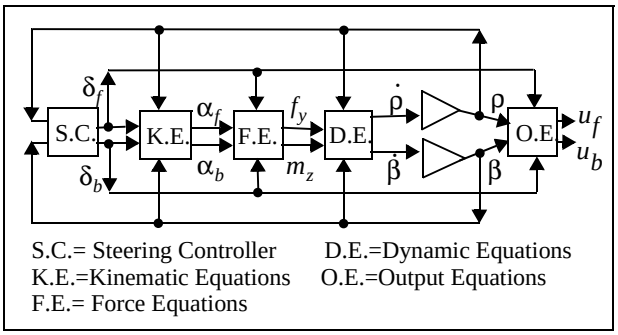


Figure 3. Block diagram for bicycle steering model. [1]



More on Training Strategies for Critic and Action Neural Networks in Dual Heuristic Programming Method

George G. Lendaris[1], Christian Paintz[2], Thaddeus Shannon[3]
1.Professor, Systems Science & Electrical Engineering, lendaris@sysc.pdx.edu
2. Graduate Student, Electrical Engineering, 3. Graduate Student, Systems Science Ph.D. Program
Portland State University
PO Box 751, Portland, OR 97207

ABSTRACT

This paper for the special session on Adaptive Critic Design Methods at the SMC '97 Conference describes a modification to the (to date) usual procedures reported for training the Critic and Action neural networks in the Dual Heuristic Programming (DHP) method [7]-[12]. This modification entails updating both the Critic and the Action networks each computational cycle, rather than only one at a time. The distinction lies in the introduction of a (real) second copy of the Critic network whose weights are adjusted less often (once per "epoch", where the epoch is defined to comprise some number N>1 computational cycles), and the "desired value" for training the other Critic is obtained from this Critic-Copy.

In a previous publication [4], the proposed modified training strategy was demonstrated on the well-known pole-cart controller problem. In that paper, the full 6 dimensional state vector was input to the Critic and Action NNs, however, the utility function only involved pole angle, not distance along the track (x). For the first set of results presented here, the 3 states associated with the x variable were eliminated from the inputs to the NNs, keeping the same utility function previously defined. This resulted in improved learning and controller performance. From this point, the method is applied to two additional problems, each of increasing complexity: for the first, an x-related term is added to the utility function for the pole-cart problem, and simultaneously, the x-related states were added back in to the NNs (i.e., increase number of state variables used from 3 to 6); the second relates to steering a vehicle with independent drive motors on each wheel. The problem contexts and experimental results are provided.

1. BACKGROUND

Dual Heuristic Programming (DHP) is a neural network approach to solving the Bellman equation [12]. The idea is to maximize a specified (*secondary*) utility function:

$$J(t) = \sum_{k=0}^{\infty} \gamma^k U(t+k) \qquad (1)$$

The term γ^k is a discount factor ($0 < \gamma < 1$) and $U(t)$ is the *primary* utility function, defined by the user for the specific application context. A useful identity:

$$J(t) = U(t) + \gamma J(t+1) \qquad (2)$$

In this paper, γ is assumed to be 1, and the usual methods of discretizing continuous models of plants is used. For DHP, at least two neural nets are needed, one for the ***action***NN functioning as the controller, and one for the ***critic***NN used to train the *action*NN. A third NN could be trained to copy the plant if an analytical description (model) of the plant is not available.

R(t) [dimension n] is the state of the plant at time t. The control signal **u**(t) [dimension a] is generated in response to the input **R**(t) by the *action*NN. The signal **u**(t) is then asserted to the plant. As a result of this, the plant changes its state to **R**(t+1). The *critic*NN's role is to assist in designing a controller (*action*NN) that is "good" relative to minimizing the specified cost function U(**R**(t),**u**(t)), which is designed to expresses the objective of the control application. In the DHP method, the *critic*NN estimates the gradient of J(t) with respect to **R**(t); the letter λ is used as a short-hand notation for this gradient, so the output of the *critic*NN is designated λ.

2. UPDATING PROCESS

We refer to Figure 1 to describe the computational steps used in the DHP methodology. First, we mention that the boxes with dark shading mean that we have an analytical expression for that item [note: if an analytical representation of the plant is not available, a NN may be trained up to emulate the plant and used in this role)]. The clear boxes are neural networks (NNs). The medium shaded boxes represent some critical equations that are to be solved in the training process. The dotted lines represent calculated values being fed into the respective boxes. The heavier dot-dash lines indicate where the learning/updating process occurs.

Reading Figure 1 from left to right, the current state **R**(t) is fed to the *action*NN, which then generates **u**(t). The model is informed of **R**(t) and **u**(t), and then generates **R**(t+1). **R**(t) is also fed to the critic#1 box and to the utility box, which generate, respectively, λ(**R**(t)) and U(**R**(t)). [Note, in the examples used herein, the Utility functions defined make use of **R**(t) but not **u**(t); if **u**(t) is included in the definition of the utility U, then we would show a connection from the output of the *action*NN to the utility box.] After the model generates **R**(t+1), this is fed to the critic#2 box, which then yields λ(**R**(t+1)) -- more simply denoted as λ(t+1). This value is a key component of the calculations in the medium-shaded boxes, which in turn

For IEEE SMC'97, Orlando, FL, Oct. 1997.

are needed to perform learning/adaptation of the *action*NN and the *critic*NN. The upper medium-shaded box calculates Δw_{ij}, and the lower medium-shaded box calculates $\lambda^{\circ}(t)$, the "desired" or "target" value for $\lambda(t)$ to enable backpropagation-type training of critic#1.

We now show the basic equations for the two medium-shaded boxes, and in particular, point out the role of $\lambda(t+1)$ in those computations.

2.1 The upper medium-shaded box.

In the present work, a basic Backpropagation algorithm is used (no embellishments) to adjust the weights in the action box. The weight-adjustment is calculated via:

$$\Delta w_{ij}(t) = lcoef \bullet \frac{\partial}{\partial w_{ij}(t)} J(t) \qquad (3)$$

where $\frac{\partial}{\partial w_{ij}(t)} J(t) = \sum_{k=1}^{u} \frac{\partial}{\partial u_k(t)} J(t) \bullet \frac{\partial}{\partial w_{ij}(t)} u_k(t)$

and $\frac{\partial}{\partial u_k(t)} J(t) = \frac{\partial}{\partial u_k(t)} U(t) + \frac{\partial}{\partial u_k(t)} J(t+1)$

and finally,

$$\frac{\partial}{\partial u_k(t)} J(t+1) = \sum_{s=1}^{n} \frac{\partial}{\partial R_s(t+1)} J(t+1) \bullet \frac{\partial}{\partial u_k(t)} R_s(t+1) \qquad (4)$$

Abbreviation: $\frac{\partial}{\partial R_s(t+1)} J(t+1) = \lambda(t+1) \qquad (5)$

$\lambda(t+1)$ is approximated by the critic, in response to the input **R**(t+1).

$\frac{\partial}{\partial u_k(t)} R_s(t+1)$ can be calculated from analytical equations of the plant, if they are available, or by backpropagation through a third neural net that has been previously trained to copy the plant.

2.2 The lower medium-shaded box.

In [4], the following equation is shown to hold:

$$\lambda_s^{\circ}(t) = \frac{\partial}{\partial R_s(t)} U(t) + \sum_{j=1}^{u} \left(\frac{\partial}{\partial u_j(t)} U(t) \bullet \frac{\partial}{\partial R_s(t)} u_j(t) \right) \qquad (6)$$
$$+ \sum_{k=1}^{n} \left(\frac{\partial}{\partial R_k(t+1)} J(t+1) \bullet \frac{\partial}{\partial R_s(t)} R_k(t+1) \right)$$
$$+ \sum_{k=1}^{n} \left\{ \sum_{j=1}^{a} \left(\frac{\partial}{\partial R_k(t+1)} J(t+1) \bullet \frac{\partial}{\partial u_j(t)} R_k(t+1) \bullet \frac{\partial}{\partial R_s(t)} u_j(t) \right) \right\}$$

which makes use of the identity of Equation 2. Note again the term containing the gradient of J(t+1) with respect to **R**(t+1): $\lambda(t+1)$. The subscript s indicates the components of the *vector* quantity $\lambda(t+1)$.

For convenience, we paraphrase Equation (6) in Equation (7). Using this version, we note that the *[~Action]* terms are calculated via the action box in Figure 1, and the *[~Critic(t+1)]* terms are calculated via the critic#2 box. Similarly, the *[~Plant]* and the *[~Utility]* terms are calculated via the respective dark-shaded box in Figure 1.

$$\lambda_s^{\circ}(t) = [\sim Utility] + \sum_{j=1}^{u} ([\sim Utility] \bullet [\sim Action])$$
$$+ \sum_{k=1}^{n} ([\sim Critic(t+1)] \bullet [\sim Plant])$$
$$+ \sum_{k=1}^{n} \left\{ \sum_{j=1}^{a} ([\sim Critic(t+1)] \bullet [\sim Plant] \bullet [\sim Action]) \right\} \qquad (7)$$

2.3 The training/update process.

Detailed descriptions of various strategies were given in [4] for "solving" (iterating) Equation 7. We describe some of them here (more loosely) using Figure 1. Keep in mind that the output of critic#2 is required for performing the calculations in the medium-shaded boxes, which in turn must be calculated to perform learning updates in the action and critic#1 boxes.

Strategy 1. Straight application of the equation.

In Figure 1, this means that after $\lambda(t+1)$ is calculated, **both** of the paths leaving critic#2 are traversed, so that the action box and the critic#1 box are updated in each iteration [in this strategy, the two boxes labeled critic#1 and critic#2 are always maintained identical -- i.e., could be the same physical box, just used for two different calculations].

Strategy 2a. Basic 2-stage process ["flip/flop"].[7]-[12]

During stage 1, train *critic*NN, not *action*NN;

In Figure 1, this means that after $\lambda(t+1)$ is calculated, only the path which adapts critic#1 is traversed, not the path which adapts the action box. This is repeated for a designated number of iterations, and then changed to stage 2 (from "flip" to "flop"). As in Strategy 1, critic#1 ≡ critic#2.

During stage 2, train *action*NN, not *critic*NN**.**

In Figure 1, this means that after $\lambda(t+1)$ is calculated, only the path which adapts the action box is traversed, not the path which adapts critic#1. This is repeated for a designated number of iterations, and then changed to stage 1 (from "flop" to "flip").

Strategy 3a. Modify 1st stage of the 2-stage process.

The modification here is to make critic#1 and critic#2 **two physically distinct objects**. Then, during Stage 1, after $\lambda(t+1)$ is calculated, adapt critic#1 as in Strategy 2, however, **leave critic#2 unchanged**. Repeat this for the designated number of iterations for Stage 1 ("flip" portion) of the process. Then, just before moving into Stage 2 (the "flop" portion), upload the weight values from critic#1 into critic#2, and then proceed to Stage 2.

Strategy 4. Single-stage process (as in Strategy 1),but use modifications introduced in Strategy 3.

Strategy 1 is modified to make critic#1 and critic#2 **two physically distinct objects** (as in Strategy 3).

4a. After $\lambda(t+1)$ is calculated, adapt both, the action box and the critic#1 box as in Strategy 1, however, **leave critic#2 unchanged**. Repeat this for a designated number of iterations (the familiar term '**epoch**' is used here for the designated number of iterations), and *at the end of each epoch, upload the weight values from critic#1 into critic#2*, and continue the process, epoch at a time.

4b. [Synopsis: while this version also adapts critic#1 via critic#2, in contrast to **4a** this strategy ***adapts the action box via critic #1.***] After $\lambda(t+1)$ is calculated via critic#2, do the bottom medium-shaded box [calculate $\lambda^{\circ}(t)$ for adapting critic#1] as in **4a**; however, now do something not shown in Figure 1: apply $\mathbf{R}(t+1)$ to critic#1, and use *the resulting* $\lambda(t+1)$ *value* to enter the upper medium-shaded box, and proceed to adapt the action box. Now apply $\mathbf{R}(t)$ to critic#1, and based on the $\lambda^{\circ}(t)$ already calculated, adapt critic#1. As in **4a**, at the end of the epoch, upload the weight values from critic#1 into critic#2.

2.4 Benefits of the modifications.

The modified strategies described above have demonstrated a benefit of allowing increased learning rates, thus reducing the convergence time for the learning process.

In [4], comparative training results were shown for all the strategies described. The well known pole-cart control problem [3], with the pole angle $\Theta(t)$ being controlled, was used as the test application. Strategy 4 was demonstrated to yield the fastest convergence, and among the best controller designs. In that paper, the full 6 dimensional state vector was input to the Critic and Action NNs, however, the utility function only involved pole angle, not distance along the track (x).

In the present paper, the 3 states associated with the x variable were first eliminated from the inputs to the NNs, keeping the same utility function previously defined. This resulted in improved learning and controller performance. Experimental results are given in Section 4.

Next, two applications of increased complexity are investigated, and comparative results are presented. This time, the exploration is confined to Strategies 1, 2a & 4. Strategy 1 is used as the baseline; next, Strategy 2a (cf. [7]-[12]) is shown with its relative performance; and finally, the new Strategies 4a & 4b are demonstrated with their relative improvements over both, Strategies 1 and 2a.

The two additional demonstration applications are:
a) the pole-cart problem again, except distance-along-the-track (x) is added in the utility function -- thus increasing the number of system states *needed* by the critic box from 3 to 6; b) steering control of a 4-wheel vehicle with independent electric motors on each wheel.

3. TEST-BED APPLICATIONS

For the pole-balancer test bed [3][4], the training procedure was to randomly initialize all the NNs (weight range: [-01, 01]), and to then provide a specified sequence of starting angles (with zero being the "desired" angle), allowing the system to train on each starting angle for a specified number of seconds. The measure used for comparing the various DHP strategies takes the values achieved by the primary utility function during training and accumulates these over the sequence of starting angles [this part is called C(j)], and add to this a penalty term according to the number of times the pole was dropped and had to be reset [this part is called D(j)], yielding a total measure M(j). The argument j labels a separate pass through the sequence of angles. In a sense, this measure incorporates the convergence speed of the strategy as well as the quality of the controller's actions along the way.

3.1 Speed of training convergence.

As is well known, convergence speed of the training process is directly influenced by the values of the learning coefficients [interchangeably called "gains" here]; if the gains are too high, the process does not converge; if set too low, while convergence may be achieved, the process takes a long time. There is an intermediate range of gain values for which the system may or may not converge; this is explored by re-initializing the system and re-starting the process. Depending on the test, we ran up to 100 trials to determine a kind of "probability of convergence".

We approximately determined for each strategy what range of gains resulted in convergence with probability very close to one. A set of gains was selected from these for each strategy, and a series of trials was run. As the gains were increased, at some point the convergence probability dropped significantly. The strategy being tested was determined to converge with probability close to 1 if the learning process converged to a controller capable of balancing the pole with zero drops through one complete pass through the training angles for at least 39 out of 40 trials. The gain values were then increased or decreased until the approximate edge of the "almost-sure" convergence region was determined (i.e., the highest values of the gains just before the probability started dropping away from one -- sort of an "edge" condition;e.g., $99\%^{+}$ for gains of .30, but much less for .35). We dubbed these the Edge Gain Values, recognizing of course that these were only approximately determinable.

The reason for going through this process is to provide a kind of "optimal" design for each of the training procedures, so they might be compared more fairly. A nice set of (approximate) Edge Gain Values were determined for the pole-cart "theta-only" and "theta-x" versions. Only partial results along this line were achieved with the steering control problem.

3.2 Pole-Cart test bed.

In [4], the actionNN and the criticNN for the pole-cart test bed were both provided the full state vector R(t) at

their input [as has been typical in the literature]. In our exploration for the present research to determine the Edge Gain Values, none could be discovered. This led to reducing the complexity of the action and critic NNs by eliminating from their inputs those states that are not used in the Utility function (i.e., not used in stipulating the objectives of the control task). Since $U(t) = (\Theta(t) - 0)^2$ was used [angle measured from vertical; target value of 0] only the $\Theta(t), \dot{\Theta}(t), \ddot{\Theta}(t)$ states are needed by the critic and action NNs. After removing the x(t)-related states for the present research, we were able to discover for each of the strategies a region of gains for which "probability one" convergence would occur. The Edge Gain Values of these regions for each strategy were determined, and used for a new set of comparative results, given in Section 4.

For the next level of complexity on which to explore the proposed new strategies, a utility function was defined for the same pole-cart problem which includes a distance-along-the-track term, x, with target position x=0:

$$U(t) = (\Theta(t) - 0)^2 + (x(t) - 0)^2 .$$

Since this equation includes both angle and distance terms, the critic and action NNs will profit from having both the 3 $\Theta(t)$-related and the 3 x(t)-related states fed into them. This was done, and again, a region of gains for which "probability one" convergence would occur was sought for each of the training strategies. Where successful, the Edge Gain Values for each strategy were determined, and used for the comparative results given in Section 4.

3.3 Steering control for 4-wheel vehicle test bed.

For a more complex plant, we turn to a 4-wheel vehicle with an independent electric drive motor on each wheel. The equations and documentation for the steering model were extracted from [1] which in turn were based on equations developed in [5].

A full nonlinear model for the vehicle is used along with the classical constant-velocity bicycle model [5] which was used in [2] for developing a front/rear wheel steering decoupling system. A front-wheel-only steering model is obtained via restricting rear-wheel steering angle to zero. The key unknown system parameters for which robustness is required are vehicle mass and tire/road side-slip coefficient of friction.

Figure 2 shows the assumed geometry. Reference coordinates: x_o-y_o; chassis coordinates: x-y, rotated by angle ψ. Mass is assumed concentrated at front and back wheels, with total mass m; wheelbase is $l = l_f + l_b$. Vehicle velocity at center of gravity has magnitude v, at angle β from x direction. Velocity v is assumed to be constant; β varies through a turn. At each wheel, steering angle is δ, subscripted by f & b for front and back. Velocity of chassis at the ith axle, i={f,b}, has assumed magnitude v_i, at angle β_i from the x axis. Thus, the wheel side-slip angle is $\alpha_i = \delta_i - \beta_i$, which represents the angular difference between the direction in which the wheel is pointed and the direction in which it is actually moving (due to slipping). This, in turn, produces a side force on the wheel with magnitude $f_i = f(\alpha_i)$ in the direction perpendicular to that in which the wheel is steered, where f_i is a monotonic function of α_i.

In [1], a set of kinematic, force and dynamic equations are developed, and these are combined to define a set of second order nonlinear state equations [bicycle steering model]. The state variables are β and ρ [where $\rho = \dot{\psi}$], and the inputs are front and back wheel steering commands, δ_f and δ_b. For the hierarchical control structure defined in [1], the steering control system is required to generate velocity commands for the wheel controllers. In Figure 3, these are shown as the outputs u_f and u_b. The symbols f_y and m_z represent, the tire side force in the y direction and the sum-of-the-moments generated by the two tire side forces around the center of gravity.

A Utility function was defined by the present authors for applying the DHP method as follows:

$$U(t) = -(ypos_{des} - ypos_{act})^2 - .5(yvel_{des} - yvel_{act})^2$$

As discussed earlier, since this utility function involves both position and velocity in the y direction, the associated state variables are fed to the action and critic NNs.

The preliminary experiments reported here are for performing a lane change, which entails a left-turn plus a right-turn type maneuver. The DHP method was used to design a controller for this task. It was decided to train up a separate NN for each of the left-turn and right-turn parts of the lane-change maneuver. These were then (easily) "pieced together" with some simple logic. After succeeding with this procedure using Strategy 4a, subsequent experiments performed for comparative purposes used only the left-turn portion of the lane-change task.

4. EXPERIMENTAL RESULTS

4.1 Pole-Cart test bed.

Results are tabulated in Tables I, II and III, for both the "Theta-Only" and the "Theta-X" versions of the Pole-Cart control test bed.

In both cases, we start with selecting the highest gains for which Strategy 1 has "probability one" convergence. This "optimal" performance for Strategy 1 provides a benchmark for comparisons. See Table III for gains. The value recorded in each row of columns 2 and 3 of Tables I & II is the M measure cited earlier. M is defined here as the sum C(1)+C(2)+C(3)+2*(D(1)+D(2)+D(3)), thus invoking a 2.0 utility penalty for a restart after a drop, a relatively inexpensive assumption.

Theta Only version.

When we use the Strategy 1 Edge Gains for Strategy 2a (column 2 in Table I) performance is worse than for Strategy 1, particularly relative to the number of "drops".

When the Strategy 1 Edge Gains are used for Strategies 4a and 4b, relative to each other, they perform approximately the same; however, relative to Strategy 1, we note the following: the standard error for the Strategy 1 mean cost is ±25%, whereas for Strategy 4 it is less than ±2%. The sample mean itself improves by approx. 40%, with the "drops" for the three methods being approx. the same (left number in column 4, Table I).

When the "optimal" Edge Gains for Strategies 4a and 4b are used (column 3 in Table I), there is another factor of 2 improvement in the total cost, with the number of "drops" going from approx. 17 down to 6.

These experimental results give more substantive empirical support for the (more qualitative) conclusions given in the previous paper [4].

Theta-X version.

When we used the Strategy 1 Edge Gains for Strategy 2a, a "stable region" of convergence could not be found -- the closest "probability of convergence" found was approximately .9.

When the Strategy 1 Edge Gains are used for Strategies 4a and 4b, relative to each other, they again perform approximately the same. Relative to Strategy 1, we note the following: the standard error is roughly the same for all three methods, but in this case, the total cost goes up slightly for Strategy 4 (using Strategy 1's gains).

When the "optimal" Edge Gains for Strategies 4a and 4b are used, however, Strategy 4 does provide improved results. As in the previous problem, the sample mean itself again improves approximately 40%, with the "drops" for Strategy 4a showing substantial improvement: from 58 --> 33, and for Strategy 4b from 58 -->41.

TABLE I: Theta-Only version of Pole-Cart problem.
TABLE II: Theta-X version of Pole-Cart problem.
Columns 2 & 3 are total cost [reported as sample mean ±standard error]. Column 4 gives average number of drops for the two gains.

Strategy	via Strategy 1 Edge Gains	via corresp. Edge Gains	$D(1)_{S1}/D(1)_{TEG}$
1	206 ± 53	206 ± 53	18 / 18
2a	239 ± 2.5	226 ± 3.5	45 / 42
4a	115 ± 2.0	**49** ± 1.0	16 / **6**
4b	128 ± 2.5	**48** ± 1.0	17 / **6**

Strategy	via Strategy 1 Edge Gains	via corresp. Edge Gains	$D(1)_{S1}/D(1)_{TEG}$
1	349 ± 11	349 ± 11	58 / 58
2a	not run	none found	-----
4a	394 ± 7.5	**207** ± 9.5	62 / **33**
4b	390 ± 10.5	**230** ± 9.5	67 / **41**

TABLE III: Edge Gains (learning coefficients) for Pole-Cart

Strategy	Theta-Only critic/action gains	Theta-X critic/action gains
1	.07/.25	.02/.2
2a	.07/1.0	none
4a	.3/.9	.04/.4
4b	.3/.9	.03/.4

These experiments again support the conclusion that Strategy 4, with its ability to utilize higher learning rates (gains), is able to provide better performance.

4.1 Four-wheel vehicle Steering test bed.

As with the pole-cart problem, experiments were run with strategies 1, 2a and 4. We were not able to find a region of "probability one" convergence for the car as we had been able to do for the pole-cart. In the quest to improve the situation, it was reasoned that since the utility function includes desired values for y and $\dot{y}$, and the fact that these change with time, it would be difficult for the critic to infer these changing targets based on indirect information. Therefore, it was decided to directly input target values for these two quantities to the critic (as well as to the action NN). The situation improved, such that for Strategy 4b, the "probability of convergence" reached .7-.8. For the other strategies, it came up to a range of .3-.5. We discovered again that it was better to leave out the bias terms in both NNs (we gave a similar observation in [4]).

In order to achieve convergence in the DHP process for this kind of problem, it was found useful to adapt a training method proposed in [6]. This method divides the state space into portions. The NNs start to train on the smallest fraction, here defined as a unit of time along the trajectory to be learned. Then, after successful learning to control the plant in that unit, the NNs are trained on two units of time, the fraction already learned and another one of the same size. After learning this bigger fraction of state space, a third one is added, etc., until the full expected range is successfully trained.

When Strategy 1 converges (relatively rare occurrence), it does so for only small gain values. Critic/Action gain values of .01/.03 gave reasonably good results (in terms of the total "cost" accumulated during training). On the other hand, gain values about a factor of 2 higher typically yielded divergent training. Also, for this application, using extremely low gain values also diverged (couldn't keep up with the changing desired trajectory?).

Strategy 2a appeared to have a (slightly) higher "probability of convergence" for the gains explored than did Strategy 1. Nevertheless, the accumulated cost was still about an order of magnitude higher than achieved with Strategies 4a and especially 4b.

Strategy 4a and 4b had substantially higher convergence probability, as indicated above.

We are not able at this time to give the kind of compar-

ative statistical data given for the pole-cart. The process of following a desired trajectory is substantially more complex for us to capture the kinds of measures that were done with the pole-cart.

A potentially important observation was that, using Strategy 4b (the fastest strategy), even after successful training, allowing the critic to continue to operate during a test run yielded lower total cost than was achieved with just the trained controller doing a "solo" run over the trajectory. This has good implications for the possibility of on-line training. A first hint in this direction was reported in [4] with the pole-cart in the context of increasing pole length from 1m to 2.4m, and the critic adapted the action NN, on-line, without dropping the pole.

5. CONCLUSION

The proposed modification to the training strategy for the DHP method continues to demonstrate improved learning performance over the more prevalent method, here called Strategy 2a. The proposed Strategies 4a & 4b were first evolved while working on the "theta-only" version of the Pole-Cart control problem [4], and now further demonstrated on the more complex "theta-x" version of the Pole-Cart problem, and thirdly, on an even more complex dynamical system, steering a 4-wheel vehicle.

These explorations also demonstrated to us that the specific information provided to the critic and action boxes can have a significant impact on performance. In particular, providing the full state vector **R**(t) to the critic may be detrimental, as demonstrated here for the theta-only test bed. For the steering control problem, it was found beneficial to additionally provide the critic with the *desired* (or *target)* values for y and $\dot{y}$. It appears that the choice of what inputs to provide the critic should be guided by the form of the specific utility function being used.

REFERENCES

[1] Accurate Automation Corp., "Advanced Intelligent Control of Next Generation Vehicles", NSF SBIR Phase I Final Report, Aug 1995.

[2] Ackermann, J., "Robust Decoupling, Ideal Steering Dynamics and Yaw Stabil. of 4WS Cars", *Automatica*, vol. 60, pp1761-1768, 1994.

[3] Barto, A., Sutton, R. & Anderson, C. " Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems" in *IEEE SMC Transactions*, Vol. SMC-13, No.5, Sep/Oct 1983.

[4] Lendaris,G. and Paintz, C. "Training Strategies for Critic and Action Neural Nets in Dual Heuristic Programming Method", in *PROCEEDINGS of ICNN'97, Houston*, IEEE, pp712-717, June, 1997.

[5] Mitschke, M., *Dynakik der Kraftfahrzeuge*, vol. C, Springer -Verlag, Berlin, 1990.

[6] Nguyen D. and Widrow, B., "The Truck Backer-Upper: an Example of Self Learning in Neural Networks", Ch 12 in *Neural Networks for Control*, Miller, Sutton & Werbos (eds), MIT Press, 1991.

[7] Prokhorov, D. and Wunsch, D. "Advanced Adaptive Critic Designs", *PROC WCNN'96*, pp. 83-87, San Diego, Erlbaum, Sept. 1996.

[8] Prokhorov, D., Santiago, R. & Wunsch, D., "Adaptive Critic Designs: A Case Study for Neurocontrol", in *Neural Networks*, vol. 8, no. 9, pp 1367-1372, 1995.

[9] Santiago, R., First Joint Mexico-US International Workshop on Neural Networks and Neurocontrol, Playacar, Mexico, Sept. 1995.

[10] Santiago, R. & Werbos, P. "New Progress Towards Truly Brain-Like Intelligent Control", *PROC WCNN '94*, pp. I-2toI-33, Erlbaum, 1994.

[11] Visnevski, N. & Prokhorov, D. "Control of a Nonlinear Multivariable System with Adaptive Critic Designs", in *Intelligent Engineering Systems through Artificial Neural Networks 6* (*PROC. ANNIE '96*), Dagli, et.al., Eds., ASME Press, pp. 559-565, 1996.

[12] Werbos, P. "Approximate Dynamic Programming for Real-Time Control and Neural Modeling", Ch. 13 in *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, (White, D.A. and Sofge, D.A., eds.), Van Nostrand Reinhold, New York, NY, 1994.

Figure 1: Computing Schema for Discussing Strategies.

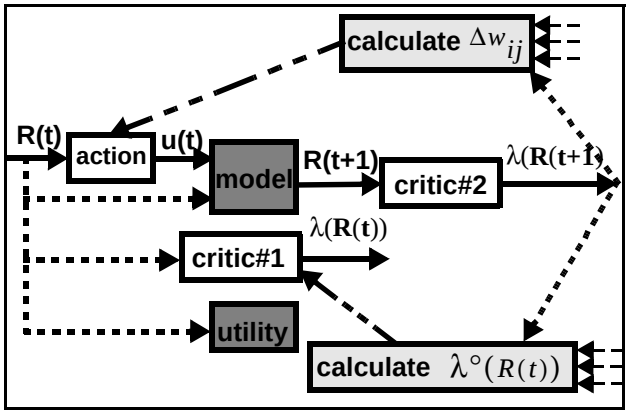


Figure 2. Assumed vehicle model geometry (after [1]).

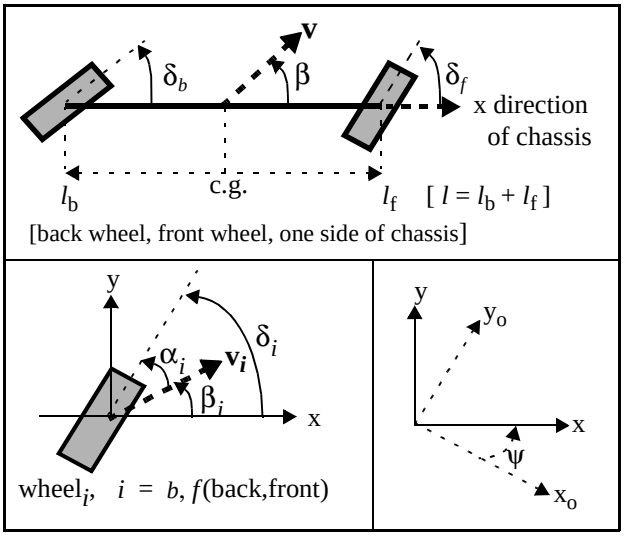
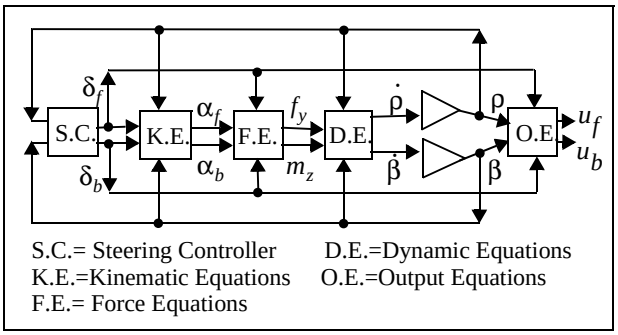


Figure 3. Block diagram for bicycle steering model. [1]



More on Training Strategies for Critic and Action Neural Networks in Dual Heuristic Programming Method

George G. Lendaris[1], Christian Paintz[2], Thaddeus Shannon[3]
1.Professor, Systems Science & Electrical Engineering, lendaris@sysc.pdx.edu
2. Graduate Student, Electrical Engineering, 3. Graduate Student, Systems Science Ph.D. Program
Portland State University
PO Box 751, Portland, OR 97207

ABSTRACT

This paper for the special session on Adaptive Critic Design Methods at the SMC '97 Conference describes a modification to the (to date) usual procedures reported for training the Critic and Action neural networks in the Dual Heuristic Programming (DHP) method [7]-[12]. This modification entails updating both the Critic and the Action networks each computational cycle, rather than only one at a time. The distinction lies in the introduction of a (real) second copy of the Critic network whose weights are adjusted less often (once per "epoch", where the epoch is defined to comprise some number N>1 computational cycles), and the "desired value" for training the other Critic is obtained from this Critic-Copy.

In a previous publication [4], the proposed modified training strategy was demonstrated on the well-known pole-cart controller problem. In that paper, the full 6 dimensional state vector was input to the Critic and Action NNs, however, the utility function only involved pole angle, not distance along the track (x). For the first set of results presented here, the 3 states associated with the x variable were eliminated from the inputs to the NNs, keeping the same utility function previously defined. This resulted in improved learning and controller performance. From this point, the method is applied to two additional problems, each of increasing complexity: for the first, an x-related term is added to the utility function for the pole-cart problem, and simultaneously, the x-related states were added back in to the NNs (i.e., increase number of state variables used from 3 to 6); the second relates to steering a vehicle with independent drive motors on each wheel. The problem contexts and experimental results are provided.

1. BACKGROUND

Dual Heuristic Programming (DHP) is a neural network approach to solving the Bellman equation [12]. The idea is to maximize a specified (*secondary*) utility function:

$$J(t) = \sum_{k=0}^{\infty} \gamma^k U(t+k) \qquad (1)$$

The term γ^k is a discount factor ($0 < \gamma < 1$) and $U(t)$ is the *primary* utility function, defined by the user for the specific application context. A useful identity:

$$J(t) = U(t) + \gamma J(t+1) \qquad (2)$$

In this paper, γ is assumed to be 1, and the usual methods of discretizing continuous models of plants is used. For DHP, at least two neural nets are needed, one for the ***action***NN functioning as the controller, and one for the ***critic***NN used to train the *action*NN. A third NN could be trained to copy the plant if an analytical description (model) of the plant is not available.

R(t) [dimension n] is the state of the plant at time t. The control signal **u**(t) [dimension a] is generated in response to the input **R**(t) by the *action*NN. The signal **u**(t) is then asserted to the plant. As a result of this, the plant changes its state to **R**(t+1). The *critic*NN's role is to assist in designing a controller (*action*NN) that is "good" relative to minimizing the specified cost function U(**R**(t),**u**(t)), which is designed to expresses the objective of the control application. In the DHP method, the *critic*NN estimates the gradient of J(t) with respect to **R**(t); the letter λ is used as a short-hand notation for this gradient, so the output of the *critic*NN is designated λ.

2. UPDATING PROCESS

We refer to Figure 1 to describe the computational steps used in the DHP methodology. First, we mention that the boxes with dark shading mean that we have an analytical expression for that item [note: if an analytical representation of the plant is not available, a NN may be trained up to emulate the plant and used in this role)]. The clear boxes are neural networks (NNs). The medium shaded boxes represent some critical equations that are to be solved in the training process. The dotted lines represent calculated values being fed into the respective boxes. The heavier dot-dash lines indicate where the learning/updating process occurs.

Reading Figure 1 from left to right, the current state **R**(t) is fed to the *action*NN, which then generates **u**(t). The model is informed of **R**(t) and **u**(t), and then generates **R**(t+1). **R**(t) is also fed to the critic#1 box and to the utility box, which generate, respectively, λ(**R**(t)) and U(**R**(t)). [Note, in the examples used herein, the Utility functions defined make use of **R**(t) but not **u**(t); if **u**(t) is included in the definition of the utility U, then we would show a connection from the output of the *action*NN to the utility box.] After the model generates **R**(t+1), this is fed to the critic#2 box, which then yields λ(**R**(t+1)) -- more simply denoted as λ(t+1). This value is a key component of the calculations in the medium-shaded boxes, which in turn

For IEEE SMC'97, Orlando, FL, Oct. 1997.

are needed to perform learning/adaptation of the *action*NN and the *critic*NN. The upper medium-shaded box calculates Δw_{ij}, and the lower medium-shaded box calculates $\lambda^{\circ}(t)$, the "desired" or "target" value for $\lambda(t)$ to enable backpropagation-type training of critic#1.

We now show the basic equations for the two medium-shaded boxes, and in particular, point out the role of $\lambda(t+1)$ in those computations.

2.1 The upper medium-shaded box.

In the present work, a basic Backpropagation algorithm is used (no embellishments) to adjust the weights in the action box. The weight-adjustment is calculated via:

$$\Delta w_{ij}(t) = lcoef \bullet \frac{\partial}{\partial w_{ij}(t)} J(t) \qquad (3)$$

where $\frac{\partial}{\partial w_{ij}(t)} J(t) = \sum_{k=1}^{u} \frac{\partial}{\partial u_k(t)} J(t) \bullet \frac{\partial}{\partial w_{ij}(t)} u_k(t)$

and $\frac{\partial}{\partial u_k(t)} J(t) = \frac{\partial}{\partial u_k(t)} U(t) + \frac{\partial}{\partial u_k(t)} J(t+1)$

and finally,

$$\frac{\partial}{\partial u_k(t)} J(t+1) = \sum_{s=1}^{n} \frac{\partial}{\partial R_s(t+1)} J(t+1) \bullet \frac{\partial}{\partial u_k(t)} R_s(t+1) \qquad (4)$$

Abbreviation: $\frac{\partial}{\partial R_s(t+1)} J(t+1) = \lambda(t+1) \qquad (5)$

$\lambda(t+1)$ is approximated by the critic, in response to the input **R**(t+1).

$\frac{\partial}{\partial u_k(t)} R_s(t+1)$ can be calculated from analytical equations of the plant, if they are available, or by backpropagation through a third neural net that has been previously trained to copy the plant.

2.2 The lower medium-shaded box.

In [4], the following equation is shown to hold:

$$\lambda_s^{\circ}(t) = \frac{\partial}{\partial R_s(t)} U(t) + \sum_{j=1}^{u} \left(\frac{\partial}{\partial u_j(t)} U(t) \bullet \frac{\partial}{\partial R_s(t)} u_j(t) \right) \qquad (6)$$
$$+ \sum_{k=1}^{n} \left(\frac{\partial}{\partial R_k(t+1)} J(t+1) \bullet \frac{\partial}{\partial R_s(t)} R_k(t+1) \right)$$
$$+ \sum_{k=1}^{n} \left\{ \sum_{j=1}^{a} \left(\frac{\partial}{\partial R_k(t+1)} J(t+1) \bullet \frac{\partial}{\partial u_j(t)} R_k(t+1) \bullet \frac{\partial}{\partial R_s(t)} u_j(t) \right) \right\}$$

which makes use of the identity of Equation 2. Note again the term containing the gradient of J(t+1) with respect to **R**(t+1): $\lambda(t+1)$. The subscript s indicates the components of the *vector* quantity $\lambda(t+1)$.

For convenience, we paraphrase Equation (6) in Equation (7). Using this version, we note that the *[~Action]* terms are calculated via the action box in Figure 1, and the *[~Critic(t+1)]* terms are calculated via the critic#2 box. Similarly, the *[~Plant]* and the *[~Utility]* terms are calculated via the respective dark-shaded box in Figure 1.

$$\lambda_s^{\circ}(t) = [\sim Utility] + \sum_{j=1}^{u} ([\sim Utility] \bullet [\sim Action])$$
$$+ \sum_{k=1}^{n} ([\sim Critic(t+1)] \bullet [\sim Plant])$$
$$+ \sum_{k=1}^{n} \left\{ \sum_{j=1}^{a} ([\sim Critic(t+1)] \bullet [\sim Plant] \bullet [\sim Action]) \right\} \qquad (7)$$

2.3 The training/update process.

Detailed descriptions of various strategies were given in [4] for "solving" (iterating) Equation 7. We describe some of them here (more loosely) using Figure 1. Keep in mind that the output of critic#2 is required for performing the calculations in the medium-shaded boxes, which in turn must be calculated to perform learning updates in the action and critic#1 boxes.

Strategy 1. Straight application of the equation.

In Figure 1, this means that after $\lambda(t+1)$ is calculated, **both** of the paths leaving critic#2 are traversed, so that the action box and the critic#1 box are updated in each iteration [in this strategy, the two boxes labeled critic#1 and critic#2 are always maintained identical -- i.e., could be the same physical box, just used for two different calculations].

Strategy 2a. Basic 2-stage process ["flip/flop"].[7]-[12]

During stage 1, train *critic*NN, not *action*NN;

In Figure 1, this means that after $\lambda(t+1)$ is calculated, only the path which adapts critic#1 is traversed, not the path which adapts the action box. This is repeated for a designated number of iterations, and then changed to stage 2 (from "flip" to "flop"). As in Strategy 1, critic#1 ≡ critic#2.

During stage 2, train *action*NN, not *critic*NN**.**

In Figure 1, this means that after $\lambda(t+1)$ is calculated, only the path which adapts the action box is traversed, not the path which adapts critic#1. This is repeated for a designated number of iterations, and then changed to stage 1 (from "flop" to "flip").

Strategy 3a. Modify 1st stage of the 2-stage process.

The modification here is to make critic#1 and critic#2 **two physically distinct objects**. Then, during Stage 1, after $\lambda(t+1)$ is calculated, adapt critic#1 as in Strategy 2, however, **leave critic#2 unchanged**. Repeat this for the designated number of iterations for Stage 1 ("flip" portion) of the process. Then, just before moving into Stage 2 (the "flop" portion), upload the weight values from critic#1 into critic#2, and then proceed to Stage 2.

Strategy 4. Single-stage process (as in Strategy 1),but use modifications introduced in Strategy 3.

Strategy 1 is modified to make critic#1 and critic#2 **two physically distinct objects** (as in Strategy 3).

4a. After $\lambda(t+1)$ is calculated, adapt both, the action box and the critic#1 box as in Strategy 1, however, **leave critic#2 unchanged**. Repeat this for a designated number of iterations (the familiar term '**epoch**' is used here for the designated number of iterations), and *at the end of each epoch, upload the weight values from critic#1 into critic#2*, and continue the process, epoch at a time.

4b. [Synopsis: while this version also adapts critic#1 via critic#2, in contrast to **4a** this strategy ***adapts the action box via critic #1.***] After $\lambda(t+1)$ is calculated via critic#2, do the bottom medium-shaded box [calculate $\lambda^{\circ}(t)$ for adapting critic#1] as in **4a**; however, now do something not shown in Figure 1: apply $\mathbf{R}(t+1)$ to critic#1, and use *the resulting* $\lambda(t+1)$ *value* to enter the upper medium-shaded box, and proceed to adapt the action box. Now apply $\mathbf{R}(t)$ to critic#1, and based on the $\lambda^{\circ}(t)$ already calculated, adapt critic#1. As in **4a**, at the end of the epoch, upload the weight values from critic#1 into critic#2.

2.4 Benefits of the modifications.

The modified strategies described above have demonstrated a benefit of allowing increased learning rates, thus reducing the convergence time for the learning process.

In [4], comparative training results were shown for all the strategies described. The well known pole-cart control problem [3], with the pole angle $\Theta(t)$ being controlled, was used as the test application. Strategy 4 was demonstrated to yield the fastest convergence, and among the best controller designs. In that paper, the full 6 dimensional state vector was input to the Critic and Action NNs, however, the utility function only involved pole angle, not distance along the track (x).

In the present paper, the 3 states associated with the x variable were first eliminated from the inputs to the NNs, keeping the same utility function previously defined. This resulted in improved learning and controller performance. Experimental results are given in Section 4.

Next, two applications of increased complexity are investigated, and comparative results are presented. This time, the exploration is confined to Strategies 1, 2a & 4. Strategy 1 is used as the baseline; next, Strategy 2a (cf. [7]-[12]) is shown with its relative performance; and finally, the new Strategies 4a & 4b are demonstrated with their relative improvements over both, Strategies 1 and 2a.

The two additional demonstration applications are:
a) the pole-cart problem again, except distance-along-the-track (x) is added in the utility function -- thus increasing the number of system states *needed* by the critic box from 3 to 6; b) steering control of a 4-wheel vehicle with independent electric motors on each wheel.

3. TEST-BED APPLICATIONS

For the pole-balancer test bed [3][4], the training procedure was to randomly initialize all the NNs (weight range: [-01, 01]), and to then provide a specified sequence of starting angles (with zero being the "desired" angle), allowing the system to train on each starting angle for a specified number of seconds. The measure used for comparing the various DHP strategies takes the values achieved by the primary utility function during training and accumulates these over the sequence of starting angles [this part is called C(j)], and add to this a penalty term according to the number of times the pole was dropped and had to be reset [this part is called D(j)], yielding a total measure M(j). The argument j labels a separate pass through the sequence of angles. In a sense, this measure incorporates the convergence speed of the strategy as well as the quality of the controller's actions along the way.

3.1 Speed of training convergence.

As is well known, convergence speed of the training process is directly influenced by the values of the learning coefficients [interchangeably called "gains" here]; if the gains are too high, the process does not converge; if set too low, while convergence may be achieved, the process takes a long time. There is an intermediate range of gain values for which the system may or may not converge; this is explored by re-initializing the system and re-starting the process. Depending on the test, we ran up to 100 trials to determine a kind of "probability of convergence".

We approximately determined for each strategy what range of gains resulted in convergence with probability very close to one. A set of gains was selected from these for each strategy, and a series of trials was run. As the gains were increased, at some point the convergence probability dropped significantly. The strategy being tested was determined to converge with probability close to 1 if the learning process converged to a controller capable of balancing the pole with zero drops through one complete pass through the training angles for at least 39 out of 40 trials. The gain values were then increased or decreased until the approximate edge of the "almost-sure" convergence region was determined (i.e., the highest values of the gains just before the probability started dropping away from one -- sort of an "edge" condition;e.g., $99\%^{+}$ for gains of .30, but much less for .35). We dubbed these the Edge Gain Values, recognizing of course that these were only approximately determinable.

The reason for going through this process is to provide a kind of "optimal" design for each of the training procedures, so they might be compared more fairly. A nice set of (approximate) Edge Gain Values were determined for the pole-cart "theta-only" and "theta-x" versions. Only partial results along this line were achieved with the steering control problem.

3.2 Pole-Cart test bed.

In [4], the actionNN and the criticNN for the pole-cart test bed were both provided the full state vector R(t) at

their input [as has been typical in the literature]. In our exploration for the present research to determine the Edge Gain Values, none could be discovered. This led to reducing the complexity of the action and critic NNs by eliminating from their inputs those states that are not used in the Utility function (i.e., not used in stipulating the objectives of the control task). Since $U(t) = (\Theta(t) - 0)^2$ was used [angle measured from vertical; target value of 0] only the $\Theta(t), \dot{\Theta}(t), \ddot{\Theta}(t)$ states are needed by the critic and action NNs. After removing the x(t)-related states for the present research, we were able to discover for each of the strategies a region of gains for which "probability one" convergence would occur. The Edge Gain Values of these regions for each strategy were determined, and used for a new set of comparative results, given in Section 4.

For the next level of complexity on which to explore the proposed new strategies, a utility function was defined for the same pole-cart problem which includes a distance-along-the-track term, x, with target position x=0:

$$U(t) = (\Theta(t) - 0)^2 + (x(t) - 0)^2 .$$

Since this equation includes both angle and distance terms, the critic and action NNs will profit from having both the 3 $\Theta(t)$-related and the 3 x(t)-related states fed into them. This was done, and again, a region of gains for which "probability one" convergence would occur was sought for each of the training strategies. Where successful, the Edge Gain Values for each strategy were determined, and used for the comparative results given in Section 4.

3.3 Steering control for 4-wheel vehicle test bed.

For a more complex plant, we turn to a 4-wheel vehicle with an independent electric drive motor on each wheel. The equations and documentation for the steering model were extracted from [1] which in turn were based on equations developed in [5].

A full nonlinear model for the vehicle is used along with the classical constant-velocity bicycle model [5] which was used in [2] for developing a front/rear wheel steering decoupling system. A front-wheel-only steering model is obtained via restricting rear-wheel steering angle to zero. The key unknown system parameters for which robustness is required are vehicle mass and tire/road side-slip coefficient of friction.

Figure 2 shows the assumed geometry. Reference coordinates: x_o-y_o; chassis coordinates: x-y, rotated by angle ψ. Mass is assumed concentrated at front and back wheels, with total mass m; wheelbase is $l = l_f + l_b$. Vehicle velocity at center of gravity has magnitude v, at angle β from x direction. Velocity v is assumed to be constant; β varies through a turn. At each wheel, steering angle is δ, subscripted by f & b for front and back. Velocity of chassis at the ith axle, i={f,b}, has assumed magnitude v_i, at angle β_i from the x axis. Thus, the wheel side-slip angle is $\alpha_i = \delta_i - \beta_i$, which represents the angular difference between the direction in which the wheel is pointed and the direction in which it is actually moving (due to slipping). This, in turn, produces a side force on the wheel with magnitude $f_i = f(\alpha_i)$ in the direction perpendicular to that in which the wheel is steered, where f_i is a monotonic function of α_i.

In [1], a set of kinematic, force and dynamic equations are developed, and these are combined to define a set of second order nonlinear state equations [bicycle steering model]. The state variables are β and ρ [where $\rho = \dot{\psi}$], and the inputs are front and back wheel steering commands, δ_f and δ_b. For the hierarchical control structure defined in [1], the steering control system is required to generate velocity commands for the wheel controllers. In Figure 3, these are shown as the outputs u_f and u_b. The symbols f_y and m_z represent, the tire side force in the y direction and the sum-of-the-moments generated by the two tire side forces around the center of gravity.

A Utility function was defined by the present authors for applying the DHP method as follows:

$$U(t) = -(ypos_{des} - ypos_{act})^2 - .5(yvel_{des} - yvel_{act})^2$$

As discussed earlier, since this utility function involves both position and velocity in the y direction, the associated state variables are fed to the action and critic NNs.

The preliminary experiments reported here are for performing a lane change, which entails a left-turn plus a right-turn type maneuver. The DHP method was used to design a controller for this task. It was decided to train up a separate NN for each of the left-turn and right-turn parts of the lane-change maneuver. These were then (easily) "pieced together" with some simple logic. After succeeding with this procedure using Strategy 4a, subsequent experiments performed for comparative purposes used only the left-turn portion of the lane-change task.

4. EXPERIMENTAL RESULTS

4.1 Pole-Cart test bed.

Results are tabulated in Tables I, II and III, for both the "Theta-Only" and the "Theta-X" versions of the Pole-Cart control test bed.

In both cases, we start with selecting the highest gains for which Strategy 1 has "probability one" convergence. This "optimal" performance for Strategy 1 provides a benchmark for comparisons. See Table III for gains. The value recorded in each row of columns 2 and 3 of Tables I & II is the M measure cited earlier. M is defined here as the sum C(1)+C(2)+C(3)+2*(D(1)+D(2)+D(3)), thus invoking a 2.0 utility penalty for a restart after a drop, a relatively inexpensive assumption.

Theta Only version.

When we use the Strategy 1 Edge Gains for Strategy 2a (column 2 in Table I) performance is worse than for Strategy 1, particularly relative to the number of "drops".

When the Strategy 1 Edge Gains are used for Strategies 4a and 4b, relative to each other, they perform approximately the same; however, relative to Strategy 1, we note the following: the standard error for the Strategy 1 mean cost is ±25%, whereas for Strategy 4 it is less than ±2%. The sample mean itself improves by approx. 40%, with the "drops" for the three methods being approx. the same (left number in column 4, Table I).

When the "optimal" Edge Gains for Strategies 4a and 4b are used (column 3 in Table I), there is another factor of 2 improvement in the total cost, with the number of "drops" going from approx. 17 down to 6.

These experimental results give more substantive empirical support for the (more qualitative) conclusions given in the previous paper [4].

Theta-X version.

When we used the Strategy 1 Edge Gains for Strategy 2a, a "stable region" of convergence could not be found -- the closest "probability of convergence" found was approximately .9.

When the Strategy 1 Edge Gains are used for Strategies 4a and 4b, relative to each other, they again perform approximately the same. Relative to Strategy 1, we note the following: the standard error is roughly the same for all three methods, but in this case, the total cost goes up slightly for Strategy 4 (using Strategy 1's gains).

When the "optimal" Edge Gains for Strategies 4a and 4b are used, however, Strategy 4 does provide improved results. As in the previous problem, the sample mean itself again improves approximately 40%, with the "drops" for Strategy 4a showing substantial improvement: from 58 --> 33, and for Strategy 4b from 58 -->41.

TABLE I: Theta-Only version of Pole-Cart problem.
TABLE II: Theta-X version of Pole-Cart problem.
Columns 2 & 3 are total cost [reported as sample mean ±standard error]. Column 4 gives average number of drops for the two gains.

Strategy	via Strategy 1 Edge Gains	via corresp. Edge Gains	$D(1)_{S1}/D(1)_{TEG}$
1	206 ± 53	206 ± 53	18 / 18
2a	239 ± 2.5	226 ± 3.5	45 / 42
4a	115 ± 2.0	**49** ± 1.0	16 / **6**
4b	128 ± 2.5	**48** ± 1.0	17 / **6**

Strategy	via Strategy 1 Edge Gains	via corresp. Edge Gains	$D(1)_{S1}/D(1)_{TEG}$
1	349 ± 11	349 ± 11	58 / 58
2a	not run	none found	-----
4a	394 ± 7.5	**207** ± 9.5	62 / **33**
4b	390 ± 10.5	**230** ± 9.5	67 / **41**

TABLE III: Edge Gains (learning coefficients) for Pole-Cart

Strategy	Theta-Only critic/action gains	Theta-X critic/action gains
1	.07/.25	.02/.2
2a	.07/1.0	none
4a	.3/.9	.04/.4
4b	.3/.9	.03/.4

These experiments again support the conclusion that Strategy 4, with its ability to utilize higher learning rates (gains), is able to provide better performance.

4.1 Four-wheel vehicle Steering test bed.

As with the pole-cart problem, experiments were run with strategies 1, 2a and 4. We were not able to find a region of "probability one" convergence for the car as we had been able to do for the pole-cart. In the quest to improve the situation, it was reasoned that since the utility function includes desired values for y and $\dot{y}$, and the fact that these change with time, it would be difficult for the critic to infer these changing targets based on indirect information. Therefore, it was decided to directly input target values for these two quantities to the critic (as well as to the action NN). The situation improved, such that for Strategy 4b, the "probability of convergence" reached .7-.8. For the other strategies, it came up to a range of .3-.5. We discovered again that it was better to leave out the bias terms in both NNs (we gave a similar observation in [4]).

In order to achieve convergence in the DHP process for this kind of problem, it was found useful to adapt a training method proposed in [6]. This method divides the state space into portions. The NNs start to train on the smallest fraction, here defined as a unit of time along the trajectory to be learned. Then, after successful learning to control the plant in that unit, the NNs are trained on two units of time, the fraction already learned and another one of the same size. After learning this bigger fraction of state space, a third one is added, etc., until the full expected range is successfully trained.

When Strategy 1 converges (relatively rare occurrence), it does so for only small gain values. Critic/Action gain values of .01/.03 gave reasonably good results (in terms of the total "cost" accumulated during training). On the other hand, gain values about a factor of 2 higher typically yielded divergent training. Also, for this application, using extremely low gain values also diverged (couldn't keep up with the changing desired trajectory?).

Strategy 2a appeared to have a (slightly) higher "probability of convergence" for the gains explored than did Strategy 1. Nevertheless, the accumulated cost was still about an order of magnitude higher than achieved with Strategies 4a and especially 4b.

Strategy 4a and 4b had substantially higher convergence probability, as indicated above.

We are not able at this time to give the kind of compar-

ative statistical data given for the pole-cart. The process of following a desired trajectory is substantially more complex for us to capture the kinds of measures that were done with the pole-cart.

A potentially important observation was that, using Strategy 4b (the fastest strategy), even after successful training, allowing the critic to continue to operate during a test run yielded lower total cost than was achieved with just the trained controller doing a "solo" run over the trajectory. This has good implications for the possibility of on-line training. A first hint in this direction was reported in [4] with the pole-cart in the context of increasing pole length from 1m to 2.4m, and the critic adapted the action NN, on-line, without dropping the pole.

5. CONCLUSION

The proposed modification to the training strategy for the DHP method continues to demonstrate improved learning performance over the more prevalent method, here called Strategy 2a. The proposed Strategies 4a & 4b were first evolved while working on the "theta-only" version of the Pole-Cart control problem [4], and now further demonstrated on the more complex "theta-x" version of the Pole-Cart problem, and thirdly, on an even more complex dynamical system, steering a 4-wheel vehicle.

These explorations also demonstrated to us that the specific information provided to the critic and action boxes can have a significant impact on performance. In particular, providing the full state vector **R**(t) to the critic may be detrimental, as demonstrated here for the theta-only test bed. For the steering control problem, it was found beneficial to additionally provide the critic with the *desired* (or *target)* values for y and $\dot{y}$. It appears that the choice of what inputs to provide the critic should be guided by the form of the specific utility function being used.

REFERENCES

[1] Accurate Automation Corp., "Advanced Intelligent Control of Next Generation Vehicles", NSF SBIR Phase I Final Report, Aug 1995.

[2] Ackermann, J., "Robust Decoupling, Ideal Steering Dynamics and Yaw Stabil. of 4WS Cars", *Automatica*, vol. 60, pp1761-1768, 1994.

[3] Barto, A., Sutton, R. & Anderson, C. " Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems" in *IEEE SMC Transactions*, Vol. SMC-13, No.5, Sep/Oct 1983.

[4] Lendaris,G. and Paintz, C. "Training Strategies for Critic and Action Neural Nets in Dual Heuristic Programming Method", in *PROCEEDINGS of ICNN'97, Houston*, IEEE, pp712-717, June, 1997.

[5] Mitschke, M., *Dynakik der Kraftfahrzeuge*, vol. C, Springer -Verlag, Berlin, 1990.

[6] Nguyen D. and Widrow, B., "The Truck Backer-Upper: an Example of Self Learning in Neural Networks", Ch 12 in *Neural Networks for Control*, Miller, Sutton & Werbos (eds), MIT Press, 1991.

[7] Prokhorov, D. and Wunsch, D. "Advanced Adaptive Critic Designs", *PROC WCNN'96*, pp. 83-87, San Diego, Erlbaum, Sept. 1996.

[8] Prokhorov, D., Santiago, R. & Wunsch, D., "Adaptive Critic Designs: A Case Study for Neurocontrol", in *Neural Networks*, vol. 8, no. 9, pp 1367-1372, 1995.

[9] Santiago, R., First Joint Mexico-US International Workshop on Neural Networks and Neurocontrol, Playacar, Mexico, Sept. 1995.

[10] Santiago, R. & Werbos, P. "New Progress Towards Truly Brain-Like Intelligent Control", *PROC WCNN '94*, pp. I-2toI-33, Erlbaum, 1994.

[11] Visnevski, N. & Prokhorov, D. "Control of a Nonlinear Multivariable System with Adaptive Critic Designs", in *Intelligent Engineering Systems through Artificial Neural Networks 6* (*PROC. ANNIE '96*), Dagli, et.al., Eds., ASME Press, pp. 559-565, 1996.

[12] Werbos, P. "Approximate Dynamic Programming for Real-Time Control and Neural Modeling", Ch. 13 in *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, (White, D.A. and Sofge, D.A., eds.), Van Nostrand Reinhold, New York, NY, 1994.

Figure 1: Computing Schema for Discussing Strategies.

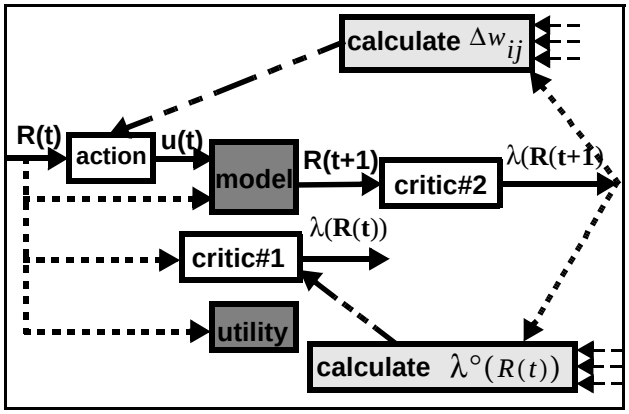


Figure 2. Assumed vehicle model geometry (after [1]).

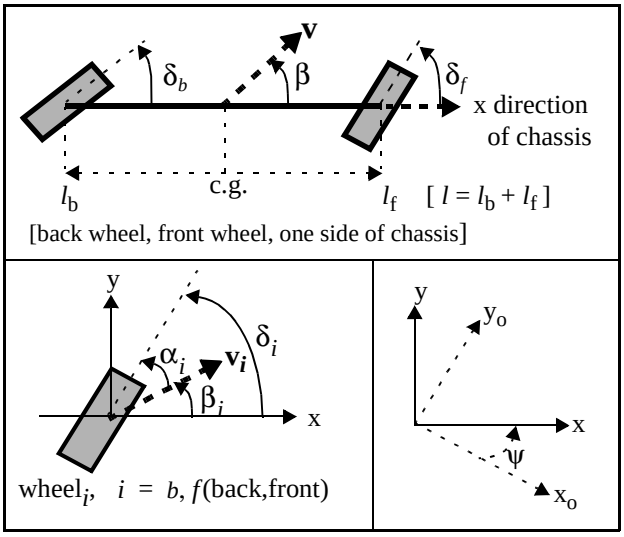


Figure 3. Block diagram for bicycle steering model. [1]