INTERNATIONAL JOURNAL OF ROBUST AND NONLINEAR CONTROL
Int. J. Robust Nonlinear Control **9**, 1071–1096 (1999)

ON THE DESIGN OF A NEURAL NETWORK AUTOLANDER

C. COX*[1], S. STEPNIEWSKI[2], C. JORGENSEN[2], R. SAEKS[1] AND C. LEWIS[1]

[1] *Accurate Automation Corporation, 7001 Shallowford Road, Chattanooga, TN 37421, U.S.A*
[2] *NASA Ames Research Center, Moffett Field, CA 94035-1000, U.S.A*

SUMMARY

A research program directed at the development of an autolander for NASA's X-33 prototype reusable launch vehicle is described. The autolander is based on a new linear quadratic adaptive critic algorithm. It is implemented by an array of Functional Link neural networks and is trained by a modified Levenberg–Marquardt method. A full stability theory is developed for the new adaptive critic algorithm. Simulation results are presented for the linear–quadratic case. Copyright © 1999 John Wiley & Sons, Ltd.

Key words: adaptive critic; neural network; flight control

1. INTRODUCTION

An on-going research program directed at the development of a neural network autolander for a scale version of NASA's X-33 prototype reusable launch vehicle is described. See Figure 1. A new adaptive critic neural control algorithm[1,2] has been developed. This adaptive critic controller is implemented by four Functional Link neural networks and is trained by a modified Levenberg–Marquardt method. Unlike previous adaptive critic algorithms, where one intertwines three separate neural network training processes for the critic, the action, and the plant (if it is initially unknown), the present algorithm is characterized by a single training process. This greatly simplifies the algorithm, while simultaneously facilitating the development of a full stability theory.

This activity represents a continuation of a research program on real-time neural adaptive flight control for damaged aircraft initiated by the NASA Advanced Programs Office in 1994. In a joint NASA/Industry initiative, NASA Ames Research Center and the Boeing Company began a four-year research initiative with a goal of flight demonstrating a concept for identification of aircraft stability and control derivatives using neural networks. This program focused on optimization of aircraft performance under nominal, off-nominal, and simulated accident scenarios. It is also focused on methods to reduce the time necessary to develop control software for new designs.

In 1995, Levenberg–Marquardt perceptrons[3] were trained on large aerodynamic data sets covering the full flight envelope of an F-15. These neural networks were incorporated into the

* Correspondence to: C. Cox, Accurate Automation Corporation, 7001 Shallowford Road, Chattanooga, TN 37421, U.S.A.

Contract/grant sponsor: NASA Phase I & II SBIR contracts
Contract/grant number: NAS2-97039 and NAS2-98016

CCC 1049-8923/99/141071–26$17.50

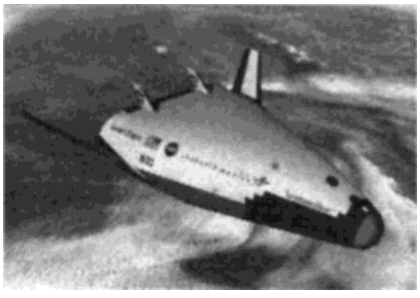

Figure 1. NASA X-33 Prototype Reusable Launch Vehicle

Figure 2. F-15 ACTIVE Aircraft Test Aircraft

third flight channel of a specially modified F-15 Advance Control Technology for Integrated Vehicles (ACTIV) test aircraft stationed at NASA Dryden Flight Research Facility, shown in Figure 2. They were used to provide the existing flight controller with the best available estimates of aircraft aerodynamics across the flight envelope and at the same time produce a dramatically more compact way to store stability and control derivatives. The second phase of this program began in 1996. It emphasized real-time on-line learning networks capable of handling significant changes in aircraft flight properties as a result of severe off nominal conditions such as loss of wing sections, ice accretion, mid-air collision, or actuator/control surface failures.[4] Meeting this goal required an integrated treatment of both system identification and control technology. Several advances in control technology have been incorporated since 1996, moving us closer to this goal.

In 1994, Kim and Calise[5] proposed a method for F-18 flight control where neural networks were employed to represent the nonlinear inverse transformations needed for feedback linearization. Totah[6] used this approach to control a simulated full degree of freedom nonlinear model of the F-15 ACTIV aircraft. This aircraft added canards and vectored thrust capability to permit simulation of abnormal flight conditions such as partial loss of wing surface or yaw moments caused by side panel damage. The aircraft was also modified to permit a third flight

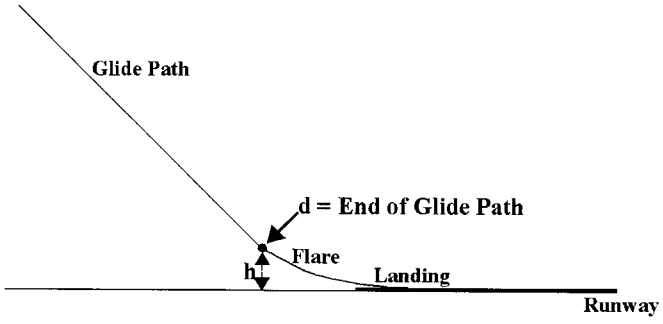

Figure 3. Typical landing trajectory

computer channel in which neural adaptive flight controllers could be embedded and an advanced onboard processor capable of accommodating neural designs.

In 1996, ACTIV demonstrated the ability of a Levenberg–Marquardt neural network to successfully learn aircraft aerodynamic coefficients with sufficient accuracy to drive the existing F-15 ACTIV controller. However, real-time, on-line learning of more dramatic changes to the aircraft plant or accidents exceeding design robustness limits required additional developments in neural learning algorithms. In 1997, a second generation on-line neural algorithm[4] was incorporated into a modified version of the NASA Langley SOFTT control architecture and was prepared for flight tests in the summer of 1998.

Third generation control architectures were designed from the beginning to incorporate neural networks. These are being considered for application to hypersonic vehicle control. The present paper discusses a linear adaptive critic architecture developed under an SBIR contract with Accurate Automation Corporation and part of the Ames in house Neuro Engineering Laboratory research program. This architecture appears fast, compact, and accurate enough to serve control functions for the problem of autolanding an unmanned hypersonic vehicle. Some recent theoretical results describing the adaptive critic design and an optimization methodology based on the Levenberg–Marquardt method supporting the neural network search requirements are presented.

In the following we describe:

- the new adaptive critic algorithm including the associated stability theory,
- its implementation as a set of Functional Link neural networks, and
- the modified Levenberg–Marquardt training method.

Unlike a traditional flight control system, an autolander is designed to implement a specific operation with high precision. A typical autolander trajectory is illustrated in Figure 3. The goal of the autolander is to precisely regulate the flight control system along a prescribed glide path, flaring at the end point of the glide path and landing, as is illustrated in Figure 3.

Since the dynamic range of the aircraft is limited during a landing, linear aircraft models will suffice for the autolander and, indeed, if accurate models of the aircraft and its environment were available 'any' standard controller would suffice. In practice, however, an aircraft model is, at best, an approximation which changes with flight condition, loading, engine characteristics, temperature, aging, etc., and, as such, some type of model adaptation process is required to precisely control the landing.

2. THE ADAPTIVE CRITIC ALGORITHM

Adaptive critic methods were introduced into neural control in the past decade[1,2,7] to circumvent the classical conundrum of optimal control theory where the optimal cost (or return) function is required to compute the optimal control law and vice versa. The Calculus of Variations attempts to circumvent this problem by directly evaluating the optimal control law, while Dynamic Programming takes the opposite approach by solving Bellman's equation for the optimal cost function.[8] Adaptive critic methods, on the other hand, are designed to learn the optimal cost function and the optimal control law on-line, converging asymptotically to the optimal solution. At each time step, the controller estimates the optimal cost of taking the system from the present state to the final state and uses that estimate (or 'critic') to specify a control law at that time step, while simultaneously updating the estimate of the optimal cost. (Hence it is called an 'adaptive critic').[1, 2, 7, 9–11]

2.1. Linear quadratic control and dynamic programming

Consider a linear system of the form

$$\dot{x} = Ax + Bu \quad x(0) = \hat{x} \tag{1}$$

with x in R^n, x in R^m. We desire to find a stabilizing feedback controller, K, such that $u = Kx$ minimizes the performance measure

$$J = \int_{t_0}^{\infty} [x^{\mathrm{T}}Qx + u^{\mathrm{T}}Ru]\, \mathrm{d}t \tag{2}$$

where Q is a positive matrix and R is a positive-definite matrix.

The well-known Dynamic Programming solution to this problem[12, 13] is obtained via the solution of the Hamilton–Jacobi–Bellman equation

$$\min_{K} \left[x^{\mathrm{T}}Qx + x^{\mathrm{T}}K^{\mathrm{T}}RKx + \frac{\partial V}{\partial x}(x)[Ax + BKx] \right] = 0, \quad x \in R^n \tag{3}$$

Here, $V : X \to R^+$ is the optimal cost function for the problem, i.e. $V(\hat{x})$ is the minimal achievable J for the optimal control problem with initial state $\hat{x}$, while the optimal feedback law, K, is the minimizing K in the Hamilton–Jacobi–Bellman equation. Moreover, the resultant closed-loop feedback system, $\dot{x} = [A + BK]x$, is asymptotically stable.[13]

In the linear quadratic case considered here, $V(x) = x^{\mathrm{T}}Px$, where P is the unique positive-definite solution of the algebraic matrix Riccati equation

$$A^{\mathrm{T}}P + PA - PBR^{-1}\, B^{\mathrm{T}}P = -Q \tag{4}$$

and the optimal control is given by

$$u = Kx = -R^{-1}B^{\mathrm{T}}Px \tag{5}$$

For the linear case, the Hamilton–Jacobi–Bellman equation takes the form

$$\begin{aligned} &x^{\mathrm{T}}Qx + x^{\mathrm{T}}PBR^{-1}B^{\mathrm{T}}Px + 2x^{\mathrm{T}}P[Ax - BR^{-1}B^{\mathrm{T}}Px] \\ &= x^{\mathrm{T}}Qx + x^{\mathrm{T}}PBR^{-1}B^{\mathrm{T}}Px + 2x^{\mathrm{T}}PAx - 2x^{\mathrm{T}}PBR^{-1}B^{\mathrm{T}}Px = 0, \quad x \text{ in } R^n \end{aligned} \tag{6}$$

where both $V(x)$ and K have been expressed in terms of P.

2.2. The adaptive critic algorithm

With the foundation of Section 2.1, we formulate the linear–quadratic adaptive critic algorithm as follows. Initially, we assume that the aircraft model is known, then we modify the algorithm for the case where A is unknown but B is known. We then modify it for the case where both A and B are unknown. To this end, we divide the time line into intervals $[t_i, t_{i+1}]$ with $t_0 = 0$. We then start with an initial positive-definite guess for the solution of the matrix Riccati equation, P^0, which is used in the time interval $[t_0, t_1]$. Equation (5) is then used to compute the control law, $u(s) = K^0 x(s)$, which is employed in this time interval. Clearly, if P^0 is the actual solution of the Riccati equation, K^0 is the optimal control law and the Hamilton–Jacobi–Bellman equation

$$x^{\mathrm{T}}(t_1)Qx(t_1) + x^{\mathrm{T}}(t_1)P^0BR^{-1}B^{\mathrm{T}}P^0x(t_1) + 2x^{\mathrm{T}}(t_1)P^0Ax(t_1) - 2x^{\mathrm{T}}(t_1)P^0BR^{-1}B^{\mathrm{T}}P^0x(t_1) = 0 \quad (7)$$

would be satisfied. In fact, there is no reason to expect that our initial guess, P^0 is the solution of the Riccati equation, in which case the error function in the Hamilton–Jacobi–Bellman equation will be non-zero:

$$e(x(t_1)) = x^{\mathrm{T}}(t_1)Qx(t_1) + x^{\mathrm{T}}(t_1)P^0BR^{-1}B^{\mathrm{T}}P^0x(t_1) + 2x^{\mathrm{T}}(t_1)P^0Ax(t_1) - 2x^{\mathrm{T}}(t_1)P^0BR^{-1}B^{\mathrm{T}}P^0x(t_1) \quad (8)$$

Therefore, we employ a learning law to choose a new estimate, $P^1 > 0$, for the solution of the Riccati equation designed to reduce $e(x)$. A modified Levenberg–Marquardt method for selecting P^1 is described in Section 3. Equation (5) is then used to compute the control law, $u(s) = K^1 x(s)$, which is employed in the time interval, $[t_1, t_2]$. This iterative process is then repeated in each time interval, selecting a new positive-definite P^i and K^i, driving the error function, $e(x)$, to zero as $i \to \infty$. The required adaptive critic algorithm is summarized as follows.

Adaptive critic algorithm with known A and B matrices

1. Set $i = 0$ and select an initial positive definite P^0.
2. Run the system with input $u(s) = -R^{-1}B^{\mathrm{T}}P^i x(s)$ in the time interval, $[t_i, t_{i+1}]$, and record the corresponding state trajectory, $x(t)$.
3. Evaluate the error function for the Hamilton–Jacobi–Bellman equation

$$e(x(t_i)) = x^{\mathrm{T}}(t_{i+1})Qx(t_{i+1}) + x^{\mathrm{T}}(t_{i+1})P^iBR^{-1}B^{\mathrm{T}}P^ix(t_{i+1}) + 2x^{\mathrm{T}}(t_{i+1})P^iAx(t_{i+1}) - 2x^{\mathrm{T}}(t_{i+1})P^iBR^{-1}B^{\mathrm{T}}P^ix(t_{i+1}) \quad (9)$$

   for $x(t_{i+1})$, and choose a new positive definite P^{i+1} to reduce the error function.
4. Set $i = i + 1$ and go to 2.

We now show that the above-described adaptive critic algorithm can be implemented for an aircraft with unknown dynamics. Substituting the equality

$$\dot{x} = Ax + Bu = Ax - BR^{-1}B^{\mathrm{T}}Px \quad (10)$$

into the error function for the Hamilton–Jacobi–Bellman equation (equation (9)) yields

$$e(x(t_{i+1})) = x^{\mathrm{T}}(t_{i+1})Qx(t_{i+1}) + x^{\mathrm{T}}(t_{i+1})P^iBR^{-1}B^{\mathrm{T}}P^ix(t_{i+1}) + 2x^{\mathrm{T}}P^i\dot{x}(t_{i+1}) \quad (11)$$

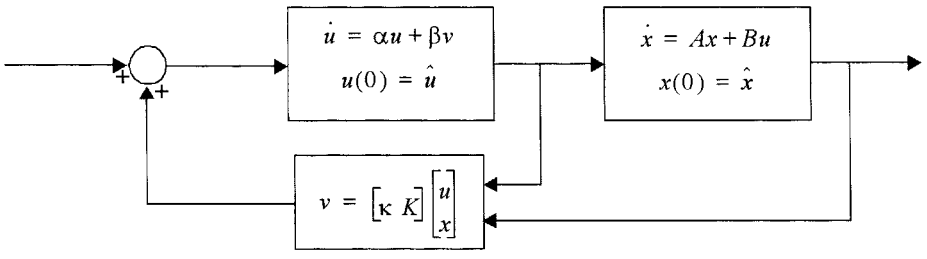

Figure 4. Autolander system with pre-compensator

which is independent of the state matrix, A. Since $\dot{x}$ can be estimated from the running system or computed numerically from x, the adaptive critic algorithm can be implemented without *a priori* knowledge of A. The modified algorithm is summarized as follows.

Adaptive critic algorithm with unknown A matrix

1. Set $i = 0$ and select an initial positive definite P^0.
2. Run the system with input $u(s) = -R^{-1}B^{\mathrm{T}}P^i x(s)$ in the time interval, $[t_i, t_{i+1}]$, and record the corresponding state trajectory, $x(t)$.
3. Evaluate the error function for the Hamilton–Jacobi–Bellman equation

$$e(x(t_{i+1})) = x^{\mathrm{T}}(t_{i+1})\,Qx(t_{i+1}) + x^{\mathrm{T}}(t_{i+1})\,P^i BR^{-1}B^{\mathrm{T}}P^i x(t_{i+1}) + 2x^{\mathrm{T}}P^i\dot{x}(t_{i+1}) \qquad (12)$$

   for $x(t_{i+1})$, and choose a new positive definite P^{i+1} to reduce the error function.
4. Set $i = i + 1$ and go to 2.

More generally, one can implement the adaptive critic algorithm with both A and B unknown by augmenting the aircraft dynamics with a prescribed pre-compensator as indicated in Figure 4. The pre-compensator is defined by any desired (controllable) linear system whose state vector is of the same dimension as the input vector for the aircraft. The dynamics of the augmented aircraft, obtained by combining the pre-compensator with the actual aircraft dynamics, take the form

$$\dot{\tilde{x}} = \begin{bmatrix} \dot{x} \\ u \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & \alpha \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + \begin{bmatrix} 0 \\ \beta \end{bmatrix} v = \tilde{A}\tilde{x} + \tilde{B}v \qquad (13)$$

where all of the (unknown) dynamics of the aircraft are now embedded in the augmented $\tilde{A}$ matrix while the augmented $\tilde{B}$ matrix is known. Furthermore, we may define an augmented performance measure by

$$\tilde{J} = \int_{t_0}^{\infty} [\tilde{x}^{\mathrm{T}}\tilde{Q}\tilde{x} + v^{\mathrm{T}}\tilde{R}v]\,\mathrm{d}t = \int_{t_0}^{\infty} \left[\begin{bmatrix} x \\ u \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + v^{\mathrm{T}}\tilde{R}v \right] \mathrm{d}t = J + \int_{t_0}^{\infty} [v^{\mathrm{T}}\tilde{R}v]\,\mathrm{d}t \qquad (14)$$

where $\tilde{R}$ is positive definite.

We now apply the adaptive critic algorithm for the case where A is unknown and B is known to the augmented system of equation (13) with the augmented performance measure of

equation (14), yielding a control law of the form

$$v = \tilde{K}\tilde{x} = [\kappa \; K] \begin{bmatrix} u \\ x \end{bmatrix} \tag{15}$$

By adding a pre-compensator to the unknown aircraft dynamics and augmenting the performance measure, it is possible to apply the linear–quadratic adaptive critic algorithm to an aircraft with completely unknown parameters.

Comments

1. *Convergence*: Nothing in the above derivation guarantees the convergence of P^i to P. Rather, we have simply shown that if the Levenberg–Marquardt (or other iterative) process converges then the limiting behaviour is optimal.
2. *Asymptotic optimality*: Since the control laws used at the beginning of the process are not optimal, the resultant control is not optimal, even if the Levenberg–Marquardt process converges. Rather, it is 'asymptotically optimal' in the sense that the control law approximates an optimal control law as $t \to \infty$.
3. *Global solution*: P can be completely determined by n independent state vectors. If the linear-quadratic adaptive critic algorithm converges for n independent state vectors, the control law it generates is valid globally, even if the state trajectory does not transverse the entire state space. Note that a global solution is *not* necessarily possible in the nonlinear case without traversing and entire state space.
4. *Implementation issues*: Real systems are plagued by problems such as actuator saturation, noise and structural mode interaction. Actuator saturation can result in a severe violation of a linearity assumption. It also can effectively cut the control loop. Noise can be a problem because it affects the estimation of $\dot{x}$. Perhaps careful filtering is a solution to the noise problem. Filters can also reduce the effects of structural mode interaction. We are currently investigating these issues.
5. *Nonlinearities*: Since an aircraft is a nonlinear system, if its state diverges sufficiently from the point of linearization, any fixed control solution will not necessarily be valid. But, the constant adaptation of our algorithm results in a time-varying control law that ameliorates this problem.
6. *Augmented performance measure*: Since $\tilde{J}$ differs from J only by the $v^{\mathrm{T}}\tilde{R}v$ term, if one makes $\tilde{R}$ small, the control law defined by $\tilde{K}$ approximates the optimal control for the given autolander. One cannot take $\tilde{R} = 0$, in which case the augmented optimal law would not be well defined. Moreover, if $\tilde{R}$ is taken to be 'too small' the resultant control inputs will be 'too large'.
7. *System identification*: With many neural control algorithms, an explicit system identification process is used to identify the unknown plant in parallel with the control process. In the above described linear adaptive critic algorithm, an explicit system identification process is not required, since the control law is constructed directly from observed data. Once the algorithm converges to P, however, one may solve the Riccati equation for A (or $\tilde{A}$), and, as such, the unknown aircraft dynamics are identified implicitly by the algorithm.
8. *Discrete-time case*: As is the case with most of control theory, the above-described adaptive critic algorithm can be readily transformed to the discrete time case by 'anyone skilled in the art'. This process, however, fails for the case where the aircraft dynamics are unknown since

both A and B are required by the discrete-time form of equation (5). Indeed, in the continuous-time case $K^i = -R^{-1}B^{\mathrm{T}}P^i$ can be evaluated without *a priori* knowledge of A, while the corresponding relationship, $K^i = -[R + B^{\mathrm{T}}P^iB]^{-1}B^{\mathrm{T}}P^iA$, in the discrete time case requires *a priori* knowledge of A. This is, however, not to imply that the discrete time-adaptive critic algorithm cannot be implemented when the aircraft dynamics are unknown, but simply that a more sophisticated transformation may be required. One approximation that seems to produce adequate results is $u(t+i) \approx -[R + B^{\mathrm{T}}P^iB]^{-1}B^{\mathrm{T}}P^i[x(t+i) - Bu(t+i-1)]$.

9. *High gain*: Unlike some adaptive controllers, our adaptive critic algorithm does not achieve performance increases by 'turning up the gain'. For the linear-quadratic case, the adaptive process approaches the optimal gains for the selected matrices of the performance metric. A term is included in the performance metric specifically to ensure that the controls do not get too large.
10. *Advantages over traditional LQR*. For a time-invariant plant and a time-invariant performance metric, this method is equivalent to traditional linear-quadratic control. One advantage of the adaptive critic over the traditional method is its ability to change gains when the performance metric varies or when the plant is unknown or varies. Also, the method has a straightforward and computationally tenable nonlinear generalization.

2.3. Stability and robustness

Unlike many neural control algorithms, the above-described linear adaptive critic algorithm is characterized by stability and robustness theorems.

Theorem 1

If the adaptive critic algorithm converges in the sense that P^i converges to P, the solution of the matrix Riccati equation, the resultant control law stabilizes the system.

Theorem 2

The system will be stable at every iteration of the learning process if $e(x(t_i)) < x^{\mathrm{T}}(t_i)\,Qx\,(t_i)$.

Proof of Theorem 1. Clearly

$$K^i = -R^{-1}B^{\mathrm{T}}P^i \rightarrow -R^{-1}B^{\mathrm{T}}P = K \tag{16}$$

converges to the optimal control, K, for the system of equations (1) and (2) when P^i converges to P, where K is a stabilizing control law via the classical linear quadratic theory.[13] Since stability is an asymptotic property, however, the fact that the limiting value of the control law stabilizes the system is sufficient to guarantee stability.

The proof of Theorem 2 requires the non-linear form of the adaptive critic algorithm[10, 14] which is summarized in Section 2.6 and, as such, the proof is deferred to Section 2.6.

Comments

1. *Guaranteed stability*: Theorem 1 guarantees the stability of the adaptive critic control law whenever the learning algorithm converges. This result is predicated on the convergence of the learning algorithm.

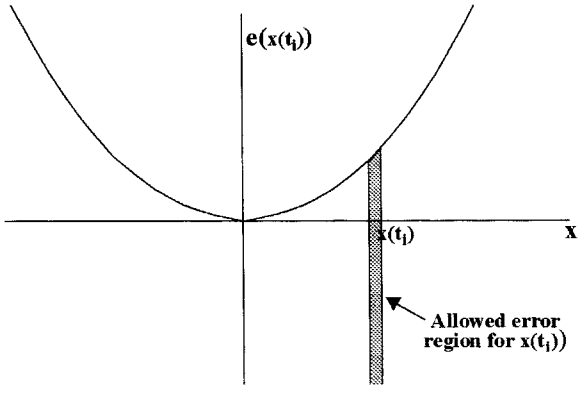

Figure 5. Error condition for stability at each iteration of the adaptive critic algorithm

Stability inequality: If one employs a learning algorithm which is designed to maintain the inequality of Theorem 2, stability will be guaranteed at every step of the learning process, and the *system will be stable even if the learning process fails to converge*. Indeed, this can be achieved by adding a Kuhn–Tucker-type inequality constraint to the Hamilton–Jacobi–Bellman error function or by modifying the Levenberg–Marquardt trust-region to guarantee that $e(x(t_i)) < x^{\mathrm{T}}(t_i)Qx(t_i)$ at every step of the training process described in Section 3.

Note that the inequality $e(x(t_i)) < x^{\mathrm{T}}(t_i)Qx(t_i)$ of Theorem 2 is not an error bound since $e(x(t_i))$ can have arbitrarily large negative values. This is illustrated for the case of a one-dimensional state space in Figure 5, where the allowed error for any given state vector, $x(t_i)$, is the semi-infinite vertical line through the state vector with upper bound $x^{\mathrm{T}}(t_i)Qx(t_i)$. Of course, the same concept applies with the x-axis replaced by an n-dimensional state space and the parabolic upper bound replaced by he appropriate n-dimensional quadratic 'bowl'. In particular, since the allowed error region is an open set in $(n+1)$-dimensional space one can readily incorporate this constraint into the Levenberg–Marquardt training algorithm.

2. *Robustness*: Theorem 2 also employs that the control algorithm is robust since stability is guaranteed in the face of computational errors and errors due to modelling, implementation, and environmental uncertainties. This is true so long as $e(x(t_i)) < x^{\mathrm{T}}(t_i)Qx(t_i)$ is satisfied.

2.4. Neural network realization

Although one can view the above-described linear adaptive critic as an abstract learning controller, the adaptive critic concept was originally developed for implementation via a neural network.[1,2,11] Here, one realizes the optimal cost function for the control problem, $V^i(x)$, as a neural network and trains it to minimize the error in the Hamilton–Jacobi–Bellman equation. The primary goal of this section is to develop a neural network realization of the linear quadratic adaptive critic algorithm described above as a system of functional links neural networks.[15]

By a functional link network, we refer to a neural network that is essentially a quadratic polynomial. As such, it can be viewed as a linear combination of nonlinear basis functions.

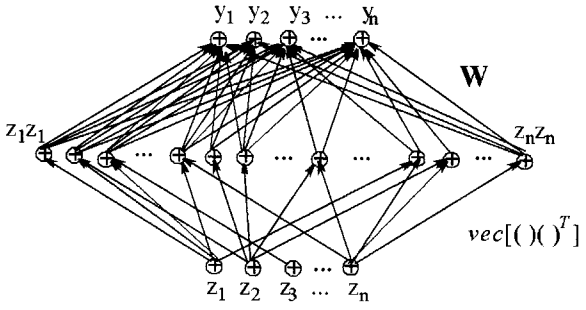

Figure 6. Outer product functional link

This casts a functional link in the same light as other popular neural networks, such as the Multiple Layer Perceptron (MLP) and the Radial Basis Function Network (RBFN).[16] The MLP and RBFN are generalized function approximators and, as such, are useful for the generalization of our algorithm to a non-linear plant. The functional link provides a convenient stepping stone to other neural networks.

A functional link consists of a 'hard-coded' nonlinear lower layer followed by a linear 'trainable' upper layer (or layers).[15] The concept was developed to circumvent the difficulty associated with 'backpropagating' through the upper layers to train the lower layers in a multilayer perceptron, while still providing a network which could be trained to approximate a nonlinear function. Although a variety of nonlinearities are possible, most functional link networks use the elements of the outer product of the input vector in the lower layer. Thus, consequently, if one inputs the n-vector, $z = [z_1, z_2, \ldots, z_n]^{\mathrm{T}}$, to a functional link, the first hidden layer will be made up of the n^2 neurons with values

$$[z_1 z_1, z_2 z_1, z_3 z_1, \ldots, z_n z_1, z_1 z_2, z_2 z_2, \ldots, z_n z_2, \ldots, z_1 z_n, z_2 z_n, \ldots, z_n z_n]^{\mathrm{T}} \tag{17}$$

To simplify this notion and facilitate the use of matrix tensor products in our analysis, we observe that the vector of equation (17) is given by $\mathrm{vec}[zz^{\mathrm{T}}]$, where 'vec[ ]' is the operation that transforms a $p \times q$ matrix to a pq-vector by stacking the columns of the matrix on top of each other. Using this notation, a typical outer product functional link is illustrated in Figure 6. Here, the vec[( ) ( )] operator takes the input n-vector to a hidden layer with neutron values $z_i z_j$; $i, j = 1, 2, \ldots, n$, while the hidden layer neurons are mapped to the output vector for a linear weight matrix, W.

The significance of the functional link network to our linear adaptive critic algorithm results from the observation that a quadratic form, $z^{\mathrm{T}} W z$, can be realized as a functional link network. To this end, we invoke the equality

$$z^{\mathrm{T}} W z = \mathrm{vec}[W]^{\mathrm{T}} \mathrm{vec}[zz]^{\mathrm{T}} \tag{18}$$

which realizes the quadratic form, $z^{\mathrm{T}} W z$, as the functional link network illustrated in Figure 7.

More generally, an arbitrary bilinear form, $z^{\mathrm{T}} W y$, can be identified with the quadratic form

$$z^{\mathrm{T}} W y = [z^{\mathrm{T}} \;\; y^{\mathrm{T}}] \begin{bmatrix} 0 & W \\ 0 & 0 \end{bmatrix} \begin{bmatrix} z \\ y \end{bmatrix} = \begin{bmatrix} z \\ y \end{bmatrix}^{\mathrm{T}} \tilde{W} \begin{bmatrix} z \\ y \end{bmatrix} \tag{19}$$

and realized as a functional link via Figure 7.

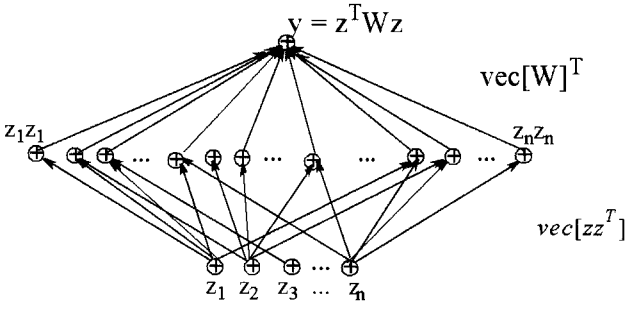

Figure 7. Realization of the quadratic form $z^{\mathrm{T}} Pz$ as a functional link

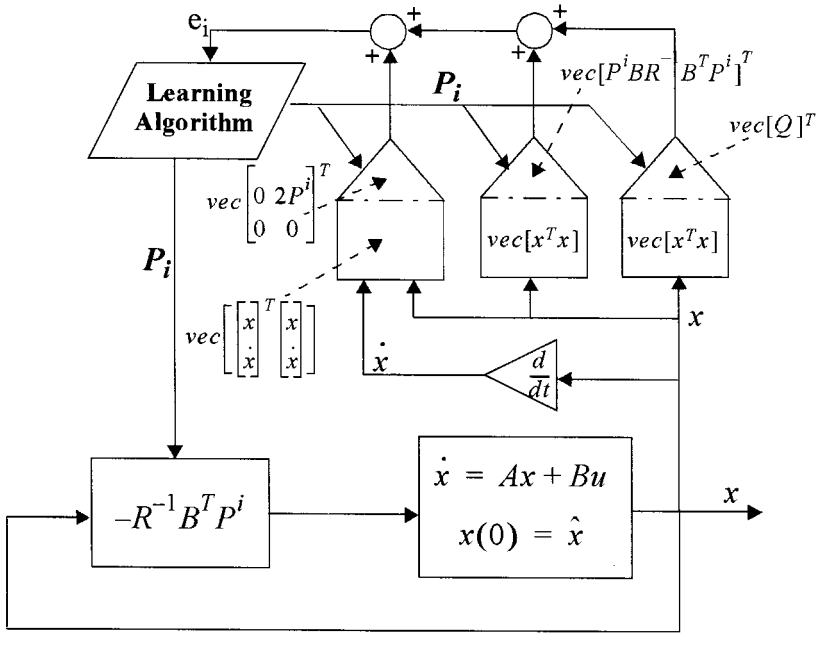

Figure 8. Realization of linear quadratic adaptive critics as a system of functional links

To implement the linear quadratic adaptive critic algorithm with a system of functional link networks, we observe that the error term for the Hamilton–Jacobi–Bellman equation for the linear quadratic case takes the form

$$e(x)(t_{i+1})) = x^{\mathrm{T}}(t_{i+1})\, Qx(t_{i+1}) + x^{\mathrm{T}}(t_{i+1})\, P^i BR^{-1} B^{\mathrm{T}} P^i x(t_{i+1}) + 2x^{\mathrm{T}} P^i \dot{x}(t_{i+1}) \tag{20}$$

which is the sum of two quadratic forms and a bilinear form, all of which may be realized by functional links as indicated in Figure 8.

Comments

1. *Coupled weights*: Although three separate neural networks are used in the functional link realization of the linear quadratic adaptive critic, the same trainable weight matrix (P)

appears in two of the three networks. As such, these networks must be updated simultaneously. This can, however, be achieved with little difficulty using the modified Levenberg–Marquardt training method described in Section 3.

2. *Positive-definite weight matrix*: To guarantee that trainable weight matrix, P^i, is positive definite, we represent it as the square of a symmetric invertible matrix, $P^i = [S^i]^2$ and train S^i rather than P^i. Since every positive-definite matrix has a square root, this transformation does not introduce any loss of generality or increase the number of degrees of freedom in the training process.
3. *Multilayer functional link*: Rather than using the single linear layer with weight matrix $\mathrm{vec}[P^i B R^{-1} B^{\mathrm{T}} P^i]^{\mathrm{T}}$ in the centre functional link of Figure 8, one can alternatively realize such a quadratic form with multiple linear layers, each realizing a single matrix factor. This is achieved by invoking the identity

$$\mathrm{vec}[M_1 M_2 M_3 \ldots M_n]^{\mathrm{T}} = \mathrm{vec}[M_1]^{\mathrm{T}} [M_2 \otimes I]^{\mathrm{T}} [M_3 \otimes I]^{\mathrm{T}} \ldots [M_n \otimes I]^{\mathrm{T}}. \tag{21}$$

As such, a quadratic form with k factors can be realized by a $k+1$ layer functional link, where the first layer is the usual outer product operator, and the upper layers are linear with weight matrices $[M_k \otimes I]^{\mathrm{T}}$, $[M_{k-1} \otimes I]^{\mathrm{T}}$, $\ldots$, $[M_2 \otimes I]^{\mathrm{T}}$, and $\mathrm{vec}[M_1]^{\mathrm{T}}$. Of course, this is simply an alternative representation of the same network and does not affect its performance or the training process.

2.5. Linear–quadratic examples

We simulated the regulation of an aircraft to a glide slope. We used two models. These are linear, longitudinal representations of the full-scale X-33 aircraft. The regulation was performed by a linear–quadratic adaptive critic algorithm. No actuator limits, sensor noises, or environmental disturbances were simulated. (These are issues of future research and development.)

2.5.1. First model. The state vector is $x^{\mathrm{T}} = [V, \alpha, q, \theta]$. This includes airspeed, angle-of-attack, pitch rate and pitch angle. All states are assumed to be directly observable. The control vector is $u^{\mathrm{T}} = [\delta_{\mathrm{e}}, \delta_{\mathrm{f}}]$, which includes elevator deflection and flap deflection. The first model was linearized about a state of $x_0^{\mathrm{T}} = [405\ \mathrm{ft/s}, 3{\cdot}35^\circ, 0^\circ\ \mathrm{s}^{-1}, -26{\cdot}65^\circ]$ and a control vector of $u_0^{\mathrm{T}} = [-9{\cdot}8^\circ, 0^\circ]$ at an altitude of 5000 ft. Maintenance of this state represents maintenance of the glide slope. All the plots shown have been translated so that a state of zero is actually this point. The A and B matrices of the first model follow:

$$A = \begin{bmatrix} -0{\cdot}0548 & 53{\cdot}83 & 0 & -27{\cdot}9 \\ -0{\cdot}000298 & -0{\cdot}8765 & 0{\cdot}9443 & -0{\cdot}03972 \\ 0 & -0{\cdot}0302 & -0{\cdot}2518 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{22}$$

$$B = \begin{bmatrix} 2{\cdot}734 & -14{\cdot}91 \\ -0{\cdot}03797 & -0{\cdot}03945 \\ -0{\cdot}0161 & -0{\cdot}0175 \\ 0 & 0 \end{bmatrix} \tag{23}$$

For this experiment, we did not use a prefilter so we assume that we know B.

The cost function matrices were chosen as follows:

$$Q = \begin{bmatrix} 25 & 0 & 0 & 0 \\ 0 & 25 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix} \tag{24}$$

$$R = \begin{bmatrix} 20 & 0 \\ 0 & 10 \end{bmatrix} \tag{25}$$

The initial P was chosen with no relevance to the selected model. We just chose a straightforward positive-definite matrix. Of course, this results in initial control gains that are no where near optimal. This causes a large jerk. But, it provides a good example of how the algorithm performs when required to learn a completely new set of gains.

$$P^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{26}$$

We ran a simulation of 10 s with an initial state of $x^{\mathrm{T}}(0) = [5\ \mathrm{ft/s},\ -2^\circ,\ -1^\circ\ \mathrm{s}^{-1},\ 2^\circ]$. (Remember, this is a deviation from x_0^{T}). This initial state was chosen to represent a perturbation off the glide slope. Figure 9 shows the cost-to-go as computed from the initial state, $x^{\mathrm{T}}(0) \cdot P^i \cdot x(0)$. This approaches the optimal values of 1256·4, which we verified by solving the Riccati equation. The step-like shape is because we sample $n = 4$ state vectors for each update of P at a rate of 10 samples per second. Figure 10 shows the deviation from the glide slope. (The glide path angle is the angle-of-attack minus the pitch angle.) Figure 11 shows the state values during this simulation. Figure 12 shows the control values. These values remain within reasonable bounds. Keep in mind that zero represents the point of linearization.

To verify the critic solution for P, we solved the Riccati equation. Both are almost the same:

$$P = \begin{bmatrix} 1{\cdot}0546 & 3{\cdot}0350 & -11{\cdot}9104 & -5{\cdot}9996 \\ 3{\cdot}0350 & 165{\cdot}5186 & -1123{\cdot}4 & -523{\cdot}5810 \\ -11{\cdot}9104 & -1123{\cdot}4 & 13864 & 5708{\cdot}2 \\ -5{\cdot}9996 & -523{\cdot}5810 & 5708{\cdot}2 & 2475{\cdot}9 \end{bmatrix} \tag{27}$$

$$P^{\mathrm{ric}} = \begin{bmatrix} 1{\cdot}0546 & 3{\cdot}0350 & -11{\cdot}9101 & -5{\cdot}9995 \\ 3{\cdot}0350 & 165{\cdot}5186 & -1123{\cdot}4 & -523{\cdot}5862 \\ -11{\cdot}9101 & -1123{\cdot}4 & 13864 & 5708{\cdot}1 \\ -5{\cdot}9995 & -523{\cdot}5682 & 5708{\cdot}1 & 2475{\cdot}8 \end{bmatrix} \tag{28}$$

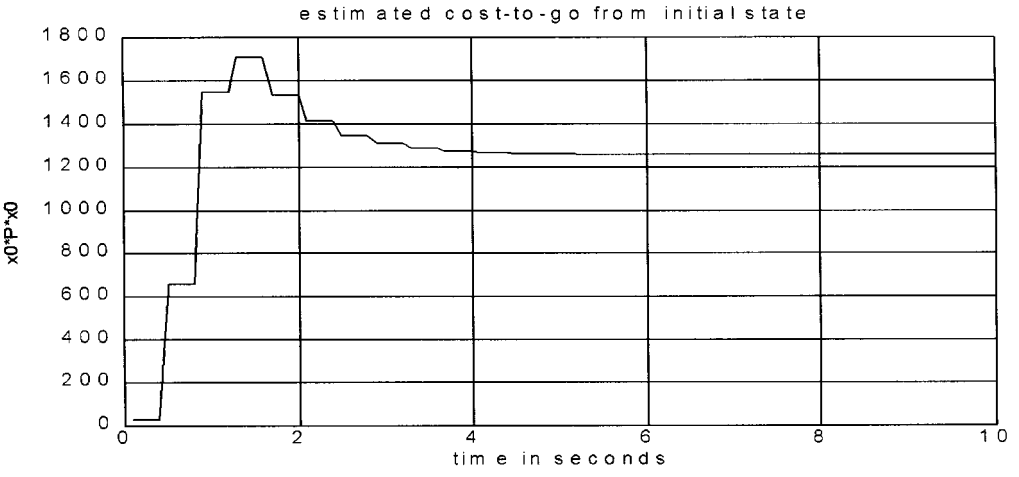

Figure 9. Cost-to-go as computed from the initial state (first model)

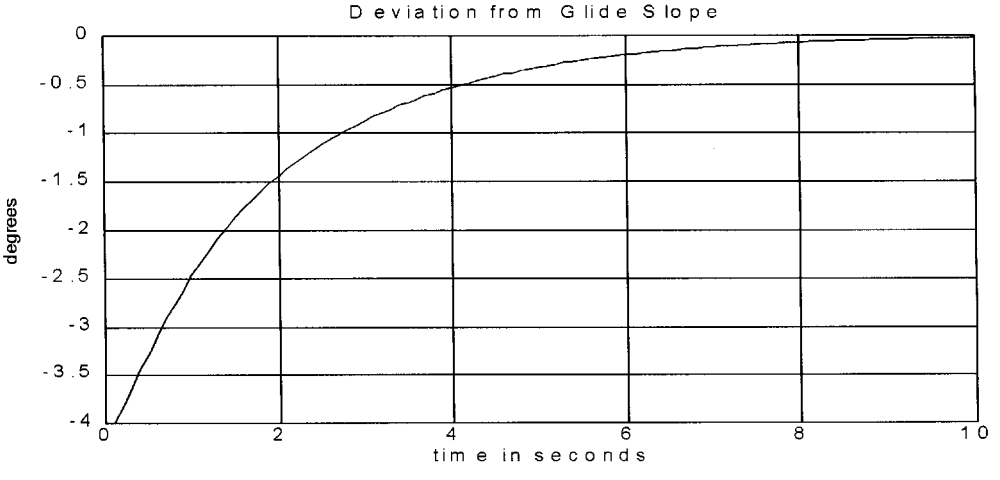

Figure 10. Deviation from glide slope (first model)

The final gain matrix is as follows:

$$K = \begin{bmatrix} -0{\cdot}1480 & -1{\cdot}0050 & 10{\cdot}6559 & 4{\cdot}4212 \\ 1{\cdot}5635 & 3{\cdot}2122 & 2{\cdot}0717 & -1{\cdot}0216 \end{bmatrix} \tag{29}$$

2.5.2. Second model. The second model was linearized about a state vector of $x_0^{\mathrm{T}} = [405\ \mathrm{ft/s}, 5{\cdot}7^\circ, 0^\circ, -24{\cdot}3^\circ]$ and a control vector of $u_0^{\mathrm{T}} = [-10{\cdot}6^\circ, 20^\circ]$. The matrices follow:

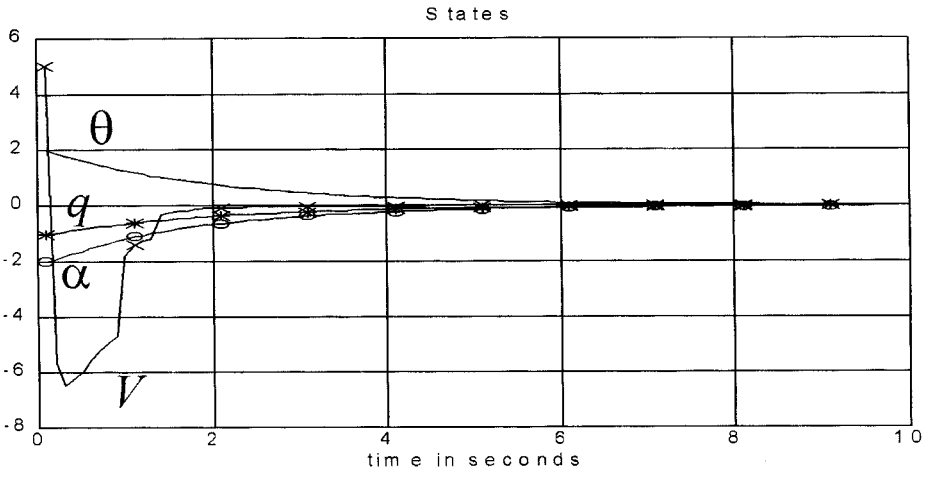

Figure 11. State values (first model)

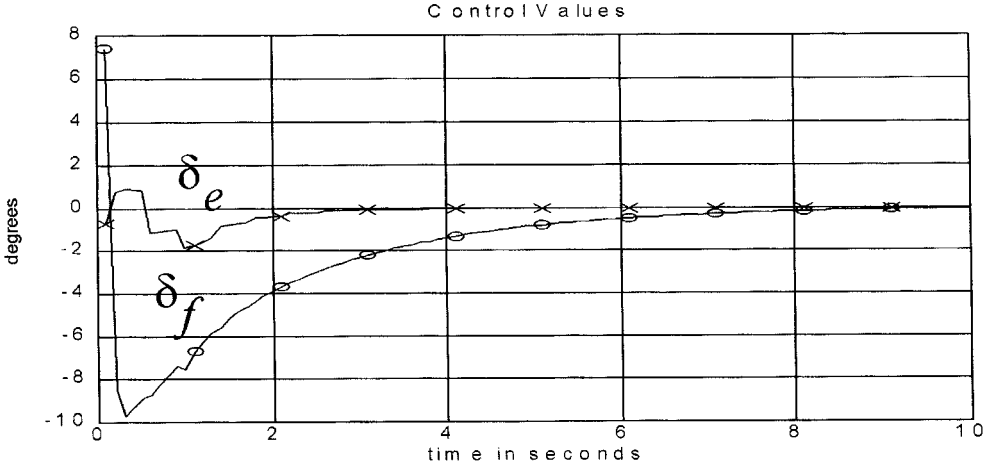

Figure 12. Control values (first model)

$$A = \begin{bmatrix} -0{\cdot}06512 & 61{\cdot}95 & 0 & -27{\cdot}9 \\ -0{\cdot}000298 & -0{\cdot}8765 & 0{\cdot}9443 & -0{\cdot}03972 \\ 0 & 0 & -0{\cdot}2518 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{30}$$

$$B = \begin{bmatrix} 2{\cdot}394 & -14{\cdot}91 \\ -0{\cdot}03797 & -0{\cdot}03945 \\ -0{\cdot}0161 & -0{\cdot}0175 \\ 0 & 0 \end{bmatrix} \tag{31}$$

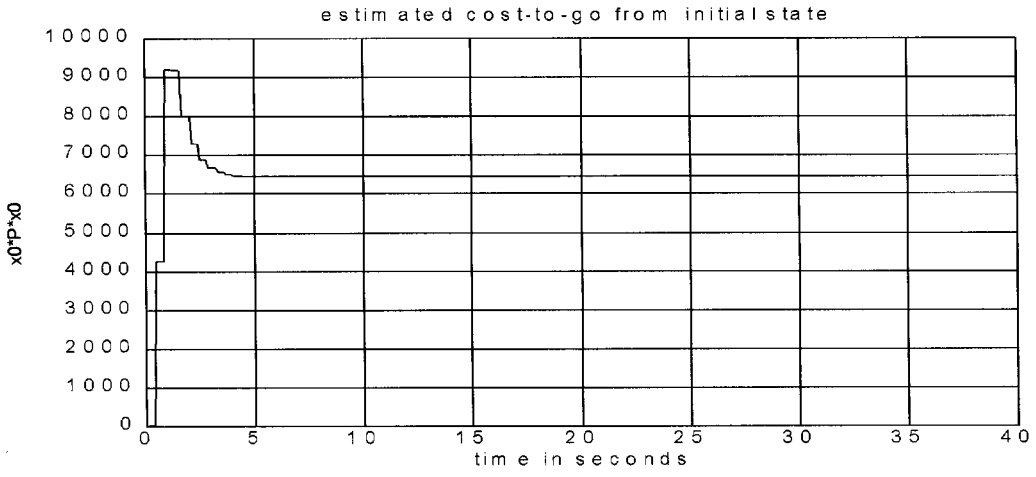

Figure 13. Cost-to-go as computed from the initial state (second model)

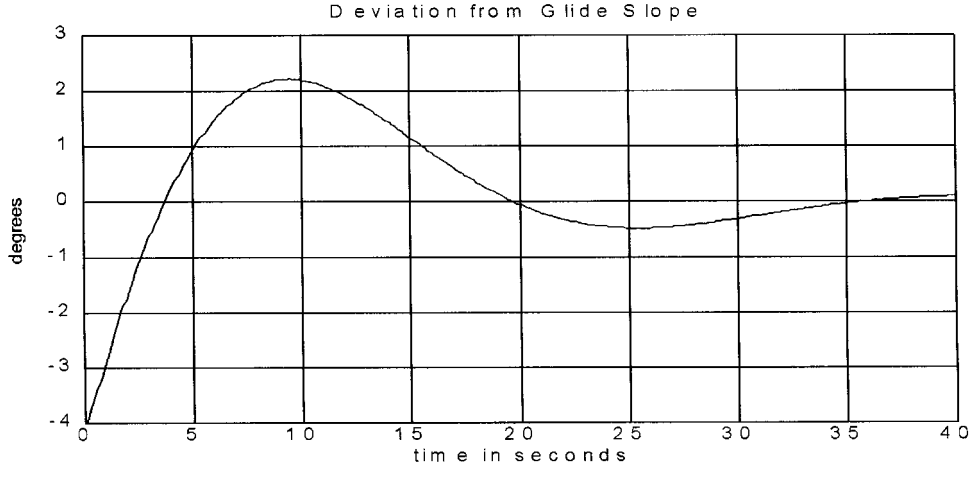

Figure 14. Deviation from glide slope (second model)

The same cost function matrices were used. We also used the same initial conditions for P^0 and for the state vector. The final estimate of cost-to-go from the initial state was 6451·4. We ran a simulation of 40 s to show the response setting, though this model will not be valid for 40 s during a real landing. Figure 13 shows the critical estimate of the cost-to-go. Figure 14 shows the deviation from the desired glide slope. Figure 15 shows the states. Figure 16 shows the control values.

The estimated P at the end of the simulation and the solution of the Riccati equation are the following:

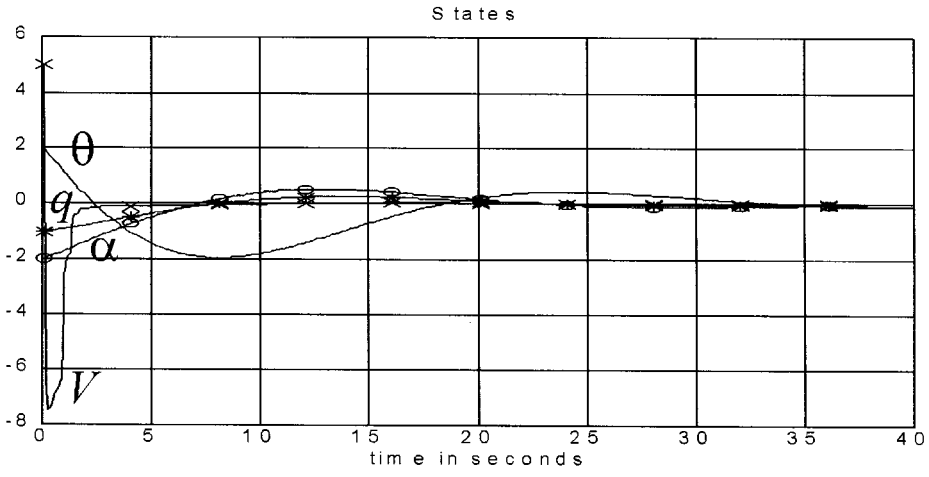

Figure 15. State values (second model)

$$P = \begin{bmatrix} 1{\cdot}0707 & 3{\cdot}7561 & 7{\cdot}6031 & -0{\cdot}2226 \\ 3{\cdot}7561 & 185{\cdot}6310 & 472{\cdot}0747 & -64{\cdot}7870 \\ 7{\cdot}6031 & 472{\cdot}4707 & 5046{\cdot}4 & 990{\cdot}3867 \\ -0{\cdot}2226 & -64{\cdot}7870 & 990{\cdot}3867 & 586{\cdot}5607 \end{bmatrix} \tag{32}$$

$$P^{\text{ric}} = \begin{bmatrix} 1{\cdot}0707 & 3{\cdot}7561 & 7{\cdot}6031 & -0{\cdot}2226 \\ 3{\cdot}7561 & 185{\cdot}6310 & 472{\cdot}0746 & -64{\cdot}7869 \\ 7{\cdot}6031 & 472{\cdot}0746 & 5046{\cdot}4 & 990{\cdot}3866 \\ -0{\cdot}2226 & -64{\cdot}7869 & 990{\cdot}3866 & 586{\cdot}5606 \end{bmatrix} \tag{33}$$

The gain matrix at the end of the simulation is

$$K = \begin{bmatrix} -0{\cdot}1149 & 0{\cdot}2828 & 4{\cdot}0485 & 0{\cdot}7009 \\ -1{\cdot}5683 & -4{\cdot}0418 & -0{\cdot}6426 & 1{\cdot}8094 \end{bmatrix} \tag{34}$$

2.6. The nonlinear case

The above-described linear adaptive critic algorithm can be extended, with little modification, to the nonlinear case by replacing the linear quadratic theory of Section 2·1 with the corresponding results from dynamic programming.[8,13,17] The purpose of this section is to summarize the nonlinear results developed in References 10 and 14 and to provide an outline of the proof of Theorem 2 from Section 2.3. We consider a non-linear aircraft model of the form

$$\dot{x} = f(x, u), \quad x(0) = \hat{x} \tag{35}$$

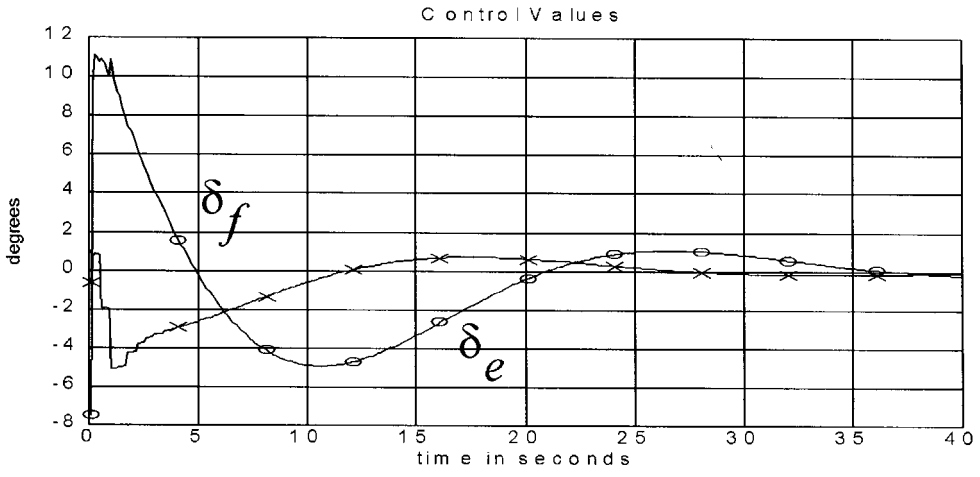

Figure 16. Control values (second model)

with x in R^n, x in R^m and with a singularity at $(\underline{x}, \underline{u})$, i.e. $f(\underline{x}, \underline{u}) = \underline{x}$. We desire to find a stabilizing feedback controller $k: R^n \to R^m$ such that, $u = k(x)$ minimizes the performance measure

$$J = \int_0^\infty l(x, u)\mathrm{d}t \tag{36}$$

where $l: R^n \times R^m \to R^+$ is greater than or equal to zero with equality if and only if $x = \underline{x}$ and $u = \underline{u}$.

The well-known Dynamic Programming solution to this problem[12, 13] is obtained via the solution of the Hamilton–Jacobi–Bellman equation

$$\min_k \left[l(x, k(x)) + \frac{\delta V}{\partial x}(x) f(x, k(x)) \right] = 0, \quad x \text{ in } R^n. \tag{37}$$

Here, $V: X \to R^+$ is the *optimal cost function* for the problem, i.e. $V(\hat{x})$ is the minimal achievable J for the optimal control problem with initial state $\hat{x}$, while the optimal feedback law, k, is the minimizing k in the Hamilton–Jacobi–Bellman equation. Moreover, the resultant feedback system, $\dot{x} = f(x, k(x))$ is asymptotically stable.[13] Unlike the linear quadratic case, however, the optimal control law, k, cannot be expressed explicitly in terms of V (or some intermediary such as the solution to the Riccati equation). This law can, however, be computed implicitly via the following set of m partial differential equations in m unknowns:

$$0 = \frac{\partial l(x(s),\ u(s))}{\partial u(s)} + \frac{\partial V(x(s))}{\partial x(s)} \frac{\partial [f(x(s),\ u(s))]}{\partial u(s)} \tag{38}$$

This can be solved for $u = k(x)$. Since m, is typically small even if n is large, the solution of equation (38) is typically straightforward.

Using the above notation, the linear adaptive critic algorithm of Section 2.2 can be extended to the nonlinear case by choosing V and the corresponding k to iteratively drive the error in the

Hamilton–Jacobi–Bellman equation to zero. The required adaptive critic algorithm for the case of a known aircraft model is summarized as follows.

Nonlinear Adaptive Critic Algorithm with known aircraft model

1. Set $i = 0$ and select an initial V^0.
2. Solve equation (38) for k^i.
3. Run the system with input $u(s) = k^i(x(s))$ in the time interval, $[t_i, t_{i+1}]$, and record the corresponding state trajectory, $\dot{x}(t) = f(x(t), k^i(x(s)))$
4. Evaluate the error function for the Hamilton–Jacobi–Bellman equation

$$e(x(t_{i+1})) = \left[l(x(t_{i+1}), k^i(x(t_{i+1}))) + \frac{\partial V^i}{\partial x}(x(t_{i+1}))\, f(x(t_{i+1}), k^i(x(t_{i+1}))) \right] \tag{39}$$

   for $x(t_{i+1})$, and choose a new V^{i+1} to reduce the error function.
5. Set $i = i + 1$ and go to 2.

Moreover, if one assumes that the aircraft dynamics are input affine then

$$\dot{x} = a(x) + b(x)u, \quad x(0) = \hat{x} \tag{40}$$

Equation (38) reduces to

$$0 = \frac{\partial l(x^o(s),\ u^o(s))}{\partial u^o(s)} + \frac{\partial V(x^o(s))}{\partial x^o(s)}\, b(x^o(s)), \tag{41}$$

which is not dependent on $a(x)$. By substituting the measured value of $\dot{x}(t_{i+1})$ for $f(x(t_{i+1}), k^i(x(t_{i+1})))$ in equation (39), the entire adaptive critic algorithm can be implemented without *a priori* knowledge of $a(x)$, while the same augmentation technique employed in Section 2.2 can be employed to circumvent the requirement that $b(x)$ be known *a priori*.

With this foundation, we sketch the proof of Theorem 2 from Section 2.3. The feedback law K^i obtained at some intermediate step of the linear–quadratic adaptive critic algorithm may not be the optimal control for the given aircraft and performance measure. But if $e_i(x) < x^T Q x$ then it is the optimal control for the given aircraft and feedback law with the modified (non-quadratic) performance measure

$$J = \int_0^\infty l^i(x, u)\mathrm{d}t = \int_0^\infty [x^T Q x + u^T R u - e_i(x)]\mathrm{d}t. \tag{42}$$

One can readily verify that the Hamilton–Jacobi–Bellman equation for the dynamic programming problem with the given aircraft model and the modified performance measure of equation (42) is satisfied using the original feedback law, K^i. Even though K^i is not the optimal control for the given linear-quadratic control problem, it is the optimal control for a non-quadratic dynamic programming problem for the given aircraft. Since the optimal feedback law for any dynamic programming problem stabilizes the feedback loop,[13] K^i is a stabilizing feedback law for the given aircraft. A detailed proof of this theorem applicable to both the linear and nonlinear cases appears in Reference 14.

Comments

1. Existence of the optimal control law: If one assumes input affine aircraft dynamics and that $l(x, u) = q(x) + r(u)$, equation (38) reduce to

$$0 = \frac{\partial r(u(s))}{\partial u(s)} + \frac{\partial V^i(x(s))}{\partial x(s)} b(x(s)), \tag{43}$$

   If $\partial r(u(s))/\partial u(s) \neq 0$, the inverse function theorem implies that equation (43) can be solved for $u(s)$ in terms of $x(s)$, guaranteeing the existence of the control law, k^i.

2. *Local optimality*: Since only those state vector alone the trajectory of the system are used in the iterative process to drive the error in Hamilton–Jacobi–Bellman equation to zero, the asymptotic optimality implied by the convergence of the error function is only valid in a neighbourhood of the state trajectory. In a nonlinear system, one would not expect to generate a globally optimally control without exploring the entire state space.

3. A MODIFIED LEVENBERG–MARQUARDT METHOD

The key to the successful implementation of the linear adaptive critic autolander is the development of an appropriate optimization technique for the algorithm of Section 2.2 or, equivalently, a neural network training method for its Functional Link implementation developed in Section 2.4. Although a variety of training algorithms are potentially applicable to the linear case (SVD, QR, etc.), the modified Levenberg–Marquardt method described in this section is ideally suited to the problem and is applicable to both linear and nonlinear cases. Moreover, different cost function and penalty terms for violating constraints can be incorporated into the modified Levenberg–Marquardt method, while the stability constraint of Theorem 2 can be incorporated into the Levenberg–Marquardt trust region.

3.1. Trust region and Levernberg–Marquardt training methods

Trust region or restricted step methods are known as very effective optimization schemes. In general, these types of training methods rely on a quadratic model of the error function around the current point, w_k,

$$\tilde{E}(w_k + \Delta w_k) = E(w_k) + \Delta w_k^{\mathrm{T}} g_k + \tfrac{1}{2} \Delta w_k^{\mathrm{T}} H_k \Delta w_k \tag{44}$$

In equation (44), g_k is the gradient of the error function, $g_k = \nabla E(w_k)$, and H_k is the $(p \times p)$ matrix of partial second derivatives or Hessian. Instead of the true Hessian $\nabla^2 E(w_k)$, its approximation may be used as well. When H_k is positive definite, the error model of equation (44) has a unique minimum that could be reached by taking a single step Δw_k that solves the set of linear equations

$$H_k \Delta w_k = -g_k \tag{45}$$

The trust-region methods realistically assume that equation (44) may be adequate only in a small neighbourhood around w_k which does not include the stationary point of $E(w)$. In such a case, a single correction Δw_k would lead to a partial training error reduction. The vector Δw_k must not only satisfy the condition $E(w_k + \Delta w_k) < E(w_k)$ but it also should not overshoot the area where

equation (44) is considered adequate. Hence, the name 'restricted step' or 'trust-region' methods. The accuracy of the approximation of equation (44) is verified through a comparison of the actual training error reduction $\Delta E(w_k) = E(w_k) - E(w_k + \Delta w_k)$ with the anticipated error decrease, $\Delta\tilde{E}(w_k) = E(w_k) - \tilde{E}(w_k + \Delta w_k)$. Based on this comparison, the trust region is either expanded or shrunk. The radius of the valid region or the maximum value of the chosen norm $\| \Delta w_k \|$ allowed may be estimated iteratively during the optimization process.[18] A penalty on violating the 'bounds' of the trust region may be also incorporated directly into equation (44). The latter approach is utilized in the neural network training algorithm described here.

The Levenberg–Marquardt technique is a special modification of the restricted step methods that takes advantage of a particular form of the error function

$$E(w_k) = \frac{1}{2q} \sum_{i=1}^{q} e_i(w_k)^2 \tag{46}$$

where q is the number of training vectors. In our linear adaptive critic application $q = 1$ and $e_1(w_k)$ is taken to be $e(x(t_{i+1}))$, while the weights, w_k, are the coefficients of the P^i matrix (or its square root S^i). Operating on equation (46) yields

$$\frac{\partial E(w_k)}{\partial w_j} = \frac{1}{q} \sum_{i=1}^{q} e_i(w_k) \frac{\partial e_i(w_k)}{\partial w_j} \tag{47}$$

or

$$g_k = \frac{1}{q} J_k^{\mathrm{T}} e_k \tag{48}$$

where J_k is the $(q \times p)$ Jacobian matrix

$$J_k = \left[\frac{\partial e_i(w_k)}{\partial w_j} \right] = \begin{bmatrix} \frac{\partial e_1(w_k)}{\partial w_1} & \frac{\partial e_1(w_k)}{\partial w_2} & \cdots & \frac{\partial e_1(w_k)}{\partial w_p} \\ \frac{\partial e_2(w_k)}{\partial w_1} & \frac{\partial e_2(w_k)}{\partial w_2} & \cdots & \frac{\partial e_2(w_k)}{\partial w_p} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_q(w_k)}{\partial w_1} & \frac{\partial e_q(w_k)}{\partial w_2} & \cdots & \frac{\partial e_q(w_k)}{\partial w_p} \end{bmatrix}, \quad e = \begin{bmatrix} e_1(w_k) \\ e_2(w_k) \\ \cdots \\ e_q(w_k) \end{bmatrix} \tag{49}$$

Evaluating second derivatives from equation (47), we obtain

$$\frac{\partial^2 E(w_k)}{\partial w_j \partial w_l} = \frac{1}{q} \sum_{i=1}^{q} \left(\frac{\partial e_i(w_k)}{\partial w_j} \frac{\partial e_i(w_k)}{\partial w_l} + e_i(w) \frac{\partial^2 e_i(w_k)}{\partial w_j \partial w_l} \right) \tag{50}$$

or

$$H_k = \frac{1}{q} \left(J_k^{\mathrm{T}} J_k + \sum_i e_i(w_k) \frac{\partial^2 e_i(w_k)}{\partial w_k^2} \right) \tag{51}$$

In the near neighbourhood of the minimum of the error function ($e(w_k) \approx 0$) or in sufficiently small area around the current point where $e(w_k)$ is close to being linear, the second term in equation (51) may be omitted. Then the Hessian matrix can be approximated by the product

$$H_k = \frac{1}{q} J_k^{\mathrm{T}} J_k \tag{52}$$

To ensure a positive-definite Hessian approximation that guarantees descent search directions, the Levenberg–Marquardt technique employs a simple modification of equation (52):

$$H_k = \frac{1}{q} (J_k^{\mathrm{T}} J_k + \lambda_k I) = \hat{H}_k + \frac{\lambda_k}{q} I \tag{53}$$

Here, λ_k ($\lambda_k > 0$) is a scalar which is regularly tuned during training. Indirectly, equation (53) controls the range of the 'trust-region'. Indeed, the new component $\lambda_k I$ effectively results in adding the term $\lambda_k \Delta w_k^{\mathrm{T}} \Delta w_k$ to the model of equation (44) which discourages the algorithm from taking large steps as λ_k increases. When the modified Hessian approximation is used, the correction vector Δw_k is determined by solving the set of equations

$$(H_k + \lambda_k I)\, \Delta w_k = -g_k \tag{54}$$

Following the general trust-region concept, the λ_k parameter is usually adjusted by comparing the actual and predicted error reduction.[18] When the actual error reduction is substantially better (larger) than the value predicted from the model of equation (44) incorporating approximation of equation (53), then λ_k is reduced. This operation shifts the search scheme toward a faster but riskier Newton-like optimization. When the error reduction is below the expected threshold, λ_k is increased and the algorithm tends to behave more like a sample gradient descent method. In the remaining cases, when the model of equation (44) roughly agrees with the actual minimization results, λ_k is left unchanged. In practical situations, this strategy of adjusting λ_k appears to be very efficient and robust.

3.2. Modified implementation of the Levenberg–Marquardt training algorithm

Although the Levenberg–Marquardt training algorithm uses the first derivatives only to build the Hessian approximation, this minimization scheme often exhibits a rapid convergence in comparison to other training methods. Unfortunately, the Levenberg–Marquardt technique is computationally expensive. It is the process of creating the Hessian approximation that requires the most processing time. To alleviate the problem of laborious calculations, we modified the original Levenberg–Marquardt training algorithm in such a way that the method does not automatically discard the Hessian approximation for every successful iteration but attempts to update and 'reuse' the current matrix when no significant changes to the error surface or the trust region are detected. Our approach was motivated by the fact that equation (44) approximates locally a small area of the error surface. This model is typically not flexible and accurate enough to include the minimizing point. However, it may be fairly adequate during several consecutive iterations, especially when the Hessian matrix is corrected, using information inferred from the changes of the first derivatives (gradients) along the search path. This kind of situation frequently occurs in the intermediate and final stage of the neural network training process when the error surface tends to change less rapidly.

Our modified Levenberg–Marquardt method employs a quasi-Newton update of the $\hat{H}_k$ matrix between evaluations the new Hessian approximations given by equation (53). A variety of quasi-Newton updating techniques has been proposed both to construct a direct Hessian approximation and its inverse. Most frequently used are the rank-2 corrections that preserve positive definiteness of the approximating matrix. These updating formulae are mainly utilized in the variable metrics optimization schemes. More general updates do not maintain a positive definite H. Other methods, such as the Huang family of formulae[19] even allow the matrix to be unsymmetric. In the modified Levenberg–Marquardt training method presented here, we employ the well-known DFP (Davidon–Fletcher–Powell) update

$$\hat{H}_{k+1} = \hat{H}_k + \Delta\hat{H}_k = \hat{H}_k + \frac{1}{b_k}\Delta g_k \Delta g_k^{\mathrm{T}} - \frac{1}{c_k}(\hat{H}_k \Delta w_k)(\hat{H}_k \Delta w_k)^{\mathrm{T}} + c_k t_k t_k^{\mathrm{T}}. \tag{55}$$

where $b_k = \Delta w_k^{\mathrm{T}} \Delta g_k$, $c_k = \Delta w_k^{\mathrm{T}} \hat{H}_k \Delta w_k$ and $t_k = \Delta g_k / b_k - \hat{H}_k \Delta w c_k$. The update formula of equation (55) is applied only to the first part of the Hessian approximation of equation (53), which is computed based on derivative information. In other words, the formula is initialized with the most recent matrix $J_k^{\mathrm{T}} J_k$.

A somewhat relaxed motivation for the choice of the DFP formula is that is computes the smallest correction $\Delta\hat{H}_k$ according to the weighted Frobenius norm

$$\Delta\hat{H}_k \underset{\in}{\min} R^{q\times q} \| G^{1/2}(\Delta\hat{H}_k)G^{1/2} \|_F \tag{56}$$

where G is any positive-definite matrix such that $G\Delta g_k = \Delta w_k$ holds. Since the Frobenius norm essentially calculates the value

$$\| A \|_F = \sqrt{\sum_{j=1}\sum_{i=1} a_{ij}^2} \tag{57}$$

it can be used to reflect the magnitude of the weighted update $\Delta\hat{H}_k$ in a manner resembling the L_2 norm of the vector.

A decision whether the matrix H_{k+1} can be obtained based on the result of equation (55) or a brand new approximation using equation (53) should be evaluated is based on two factors. Recall, that the configuration of the eigenvalues of H_k reflects how the algorithm 'sees' the error surface in the neighbourhood of the current point. When the matrix is positive definite with similar eigenvalues, the error function is approximated by a convex parabolic surface with almost circular contour lines. When the eigenvalues differ significantly but they are all positive, the surface looks more like a narrow valley. The model used by the Levenberg–Marquardt scheme does not consider negative curvatures.

Clearly, finding and inspecting all eigenvalues of the H_{k+1} matrix would make the training algorithm rather inefficient. Fortunately, certain quantities such as the sum of all eigenvalues can be computed with less overhead; this result is equal to the sum of its diagonal elements (i.e. its matrix trace). In our algorithm we compare the sum of eigenvalues of H_{k+1} with the initial matrix $H^* = J_k^{\mathrm{T}} J_k$ obtained during the most recent evaluation of the Hessian approximation of equation (53),

$$\sigma = \frac{|\operatorname{tr}(H_{k+1}) - \operatorname{tr}(H^*)|}{\operatorname{tr}(H^*)} \tag{58}$$

If σ is larger than a certain threshold, e.g. 0·03, then the new Hessian approximation is computed. Otherwise, the result of the quasi-Newton update of equation (55) is used to find the next weight correction.

Not all relevant changes to the Hessian matrix can be detected through the σ parameter and, as such, the algorithm also monoitors the value of λ_k which loosely reflects the extent of the trust region. A growing λ_k indicates a shrinking area where the model of equation (45) is adequate. The ratio λ_k/λ^* (where λ^* denotes a value adjusted for the most recent approximation, equation (53)) is used as a second parameter to diagnose whether the new Hessian approximation should be computed.

Comments

1. Obviously, the predicte error decrease $\Delta\tilde{\mathrm{E}}(\Delta w_k)$ can be computed using equation (44) directly.

$$\Delta\tilde{E}(\Delta w_k) = E(w_k) - \tilde{E}(w_k + \Delta w_k) = -\Delta w_k^{\mathrm{T}} g_k - \tfrac{1}{2}\Delta w_k^{\mathrm{T}} H_k \Delta w_k \tag{59}$$

   Vector $H_k\Delta w_k$ should be stored in computer memory since it can be used in the DFP update of equation (55). This, however, complicates slightly the implementation of the training algorithm. A computationally more efficient method to establish $\Delta\tilde{E}(\Delta w_k)$ may be obtained by invoking equation (54). Based on this expression we can write

$$H_k \Delta w_k = -g_k - \lambda_k I \Delta w_k \tag{60}$$

   By substituting equation (60) into equation (59) and performing some simple matrix algebra, we obtain[3]

$$\Delta\tilde{E}(\Delta w_k) = \tfrac{1}{2}(\Delta w_k^{\mathrm{T}} g_k + \lambda \Delta w_k^{\mathrm{T}} \Delta w_k) \tag{61}$$

2. The computational efficiency of the training algorithm could be further improved by revisiting the quasi-Newton update. Assuming that $b_k > 0$ and $c_k > 0$, equation (55) can be written in the clear symmetric, rank-2 form[20]

$$H_{k+1} = H_k + u_k u_k^{\mathrm{T}} - v_k v_k^{\mathrm{T}} \tag{62}$$

   where

$$v_k = \frac{1}{\sqrt{c_k + b_k}}(\hat{H}_k \Delta w_k), \qquad u_k = \frac{\sqrt{c_k + b_k}}{b_k}\Delta g_k - v_k \tag{63}$$

   Since in our training algorithm we initialize the quasi-Newton update with matrix $J_k^{\mathrm{T}} J_k$ which is only semi-positive definite, it is theoretically possible that $c_k = 0$. In this rather exceptional case or when $b_k \leqslant 0$ the Hessian update is abandoned and a new approximation using equation (53) is evaluated. When the satisfactory step Δw_k is taken, b_k should always be positive. In our case, this requirement comes mainly frm the Wolfe–Powell slope test for acceptable points[18]

$$g_{k+1}^{\mathrm{T}} \Delta w_k \geqslant \mu g_k^{\mathrm{T}} \Delta w_k \tag{64}$$

   where $\mu(0 \ll \mu < 1)$ is a fixed parameter. The condition of equation (64) states that the function $E(w_k)$ alone direction Δw_k should decrease faster locally at the point w_k than at w_{k+1}. A positive value of b_k also guarantees that the update of equation (55) produces a positive-definite matrix if the initial matrix is positive definite.

3.3. Functional link training algorithm

To apply the modified Levenberg–Marquardt training algorithm to the linear adaptive critic functional link array of Figure 8, we desire to minimize the error in the Hamilton–Jacobi–Bellman equation (12) over the set of all positive-definite weight matrices, P^i, at each point, $x(t_{i+1})$, alond the state trajectory. To alleviate the necessity of maintaining the positve-definite constraint on P^i we express P^i as the product of symmetric matrices $P^i = [S^i]^2$. This is done without loss of generality since every positive-definite matrix can be expressed as a product of symmetric matrices (its 'matrix square root') and conversely, while the number of independent weights in both case $(n(n+1)/2)$ is unchanged. As such, equation (12) takes the form

$$e(x(t_{i+1})) = x^{\mathrm{T}}(t_{i+1})Qx(t_{i+1}) + x^{\mathrm{T}}(t_{i+1})[S^i]^2 BR^{-1}B^{\mathrm{T}}[S^i]^2 x(t_{i+1}) + 2x^{\mathrm{T}}[S^i]^2 \dot{x}(t_{i+1}) \quad (65)$$

and we apply the modified Levenberg–Marquardt method to

$$E(S^i) = \tfrac{1}{2}[e(x(t_{i+1}))]^2 \quad (66)$$

while

$$\frac{\partial E(S^i)}{\partial s_{kl}} = e(x(t_{i+1}))\frac{\partial e(x(t_{i+1}))}{\partial s_{kl}} \quad (67)$$

where s_{kl} is an arbitrary entry in S^i.

Although somewhat complex the expression for $e(x(t_{i+1}))$ of equation (65) can be differentiated analytically yielding

$$\frac{\partial e(x(t_{i+1}))}{\partial s_{kl}} = 2x^{\mathrm{T}}(t_{i+1})[S^i E_{kl} + E_{kl}S^i]BR^{-1}B^{\mathrm{T}}[S^i]^2 x(t_{i+1}) + 2x^{\mathrm{T}}[S^i E_{kl} + E_{kl}S^i]\dot{x}(t_{i+1}) \quad (68)$$

where E_{kl} is a matrix with 1's in the k-l and l-k positions and 0's elsewhere. As such, the partial derivatives required for the Hessian approximations in the Levenberg–Marquardt method can be readily computed.

Comments

1. Note that the constraint of Theorem 2, to guarantee that the feedback law, K^i, produced by the linear adaptive critic algorithm be stabilizing at each iteration of the training process; can be readily incorprated into the modified Levenberg–Marquardt training method by limiting the size of the trust-region to maintain the inequality $e(x(t_i)) < x^{\mathrm{T}}(t_i)Qx(t_i)$ at each iteration.

4. CONCLUSIONS

Our goal in the above development has been to describe a new adaptive critic neural control algorithm for a scale version of the X-33 autolander. This algorithm is implemented by an array of four functional link neural networks with coupled weight matrices, which realize the four bilinear forms which appear in the Hamilton–Jacobi–Bellman equation for the autolander problem, while a modified Levenberg–Marquardt method is developed to simultaneously train these networks. The algorithm can be efficiently implemented and is characterized by a full stability theory. The linear quadratic version of the algorithm is presented in full along with a sketch of the nonlinear theory.

ACKNOWLEDGEMENTS

This research was supported in part by NASA Phase I & II SBIR Contracts NAS2-97039 and NAS2-98016. We would like to thank Lockheed Martin Skunkworks for their assistance.

REFERENCES

1. Werbos, P. J., 'Neurocontrol and supervised learning: an overview and evaluation', in *Handbook of Intelligent Control*. D. A. White and D. A. Sofge (Eds), Van Nostrand Reinhold, New York, 1994, pp. 65–89.
2. Werbos, P. J., 'Approximate dynamic programming for real time control and neural modeling', in *Handbook of Intelligent Control*, White and Sofge (Eds), Van Nostrand Reinhold, New York, 1994, pp. 493–525.
3. Norgaard P. M., C. C. Jorgensen and J. Ross, *Neural network prediction of new aircraft design coefficients*, *NASA AMES Research Center Technical Report*, NASA-TM-112197, 1997.
4. Jorgensen, C. C., 'Feedback linearized aircraft control using dynamic cell structures', *Proc. World Automation Congress*, *Anchorage*, Alaska, 1998.
5. Kim, B. S. and A. J. Calise, 'Nonlinear flight control using neural networks', *AIAA Paper 94-3646-CP*, 1994.
6. Totah, J., 'Simulation of a neural based flight controller', *AIAA Paper AIAA-96-3503*, 1996.
7. Barto, A., R. S. Sutton and C. W. Anderson, 'Neuronlike adaptive elements that can solve difficult learning problems', *IEEE Trans. Systems*, *Man and Cybernet*., **13**(5), 834–846 (1983).
8. Bellman, R. E., *Dynamic Programming*, Princeton University Press, Princeton, 1957.
9. Prokhorov, D. and L. Feldkamp, 'Primitive adaptive critics', *Proc. Int. Conf. on Neural Networks*, Vol. IV, 1997, pp. 2263–2267.
10. Cox, C. and R. Saeks, 'Adaptive critic control and functional link networks', *Proc. IEEE Conf. on Systems*, *Man*, *and Cybernetics*, Vol. 2, 1998, pp. 1652–1657.
11. Zaman, R., D. Prokhorov and D. Wunsch, 'Adaptive critic design in learning to play game of go', *Proc. Int. Conf. on Neural Networks*, Vol. I, 1997, pp. 1–4.
12. An, P., S. Aslam-Mir, M. Brown and C. J. Harris, 'A reinforcement approach to on-line optimal control', *Proc. World Congress on Neural Networks '94*, San Diego, Vol. 2, INNS, Earlbaum, 1994, pp. 66–72.
13. Luenberger, D. G., *Introduction to Dynamic Systems*: *Theory*, *Models*, *and Applications*, Wiley, New York, 1979.
14. Saeks, R. and C. Cox, 'On adaptive critics and neural networks', *Internal Report*, Accurate Automation Corp, Chattanooga, TN 37421, 1997.
15. Pao, Y.-H., *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, MA, 1989.
16. Haykin, S., *Neural Networks*: *A Comprehensive Foundation*, MacMillan, New York, 1994.
17. Bertsekas, D. P., *Dynamic Programming*: *Deterministic and Stochastic Models*, Prentice-Hall, Englewood Cliffs, NJ, 1987.
18. Fletcher, R., *Practical Methods of Optimization*, Wiley, New York, 1987.
19. Huang, H. Y., 'Unified approach to quadratically convergent algorithms for function minimization', *J. Optim. Theory Appl*., **5**, 405–423 (1970).
20. Stepniewski, S. W. and C. C. Jorgensen, Accelerated training for large feedforward neural networks, *Unpublished Notes*, NASA/TM-1998-112239, November 1998.

