Neural Dynamic Optimization for Control Systems—Part I: Background

Chang-Yun Seong, *Member, IEEE,* and Bernard Widrow, *Life Fellow, IEEE*

***Abstract*—The paper presents neural dynamic optimization (NDO) as a method of optimal feedback control for nonlinear multi-input-multi-output (MIMO) systems. The main feature of NDO is that it enables neural networks to approximate the optimal feedback solution whose existence dynamic programming (DP) justifies, thereby reducing the complexities of computation and storage problems of the classical methods such as DP. This paper mainly describes the background and motivations for the development of NDO, while the two other subsequent papers of this topic present the theory of NDO and demonstrate the method with several applications including control of autonomous vehicles and of a robot arm, respectively.**

***Index Terms*—Dynamic programming (DP), information time shift operator, learning operator, neural dynamic optimization (NDO), neural networks, nonlinear systems, optimal feedback control.**

I. Introduction

NONLINEAR control system design has been dominated by linear control techniques, which rely on the key assumption of a small range of operation for the linear model to be valid. This tradition has produced many reliable and effective control systems [1]–[4].

However, the demand for control methods of complex nonlinear multi-input-multi-output (MIMO) systems has recently been increasing for several reasons. First, most real-world dynamical systems are inherently nonlinear. Second, modern technology, such as high-performance aircraft and high-speed high-accuracy robots, demands control systems with much more stringent design specifications, which are able to handle nonlinearities of the controlled systems more accurately. Third, along with the demand for high performance, MIMO control systems often become preferred or required because of the availability of cheaper and more reliable sensors and actuators made possible by the advances in such technology. The challenge for control design is to fully utilize this additional information and degrees of freedom to achieve the best control system performance possible. Fourth, controlled systems must be able to reject disturbances and uncertainties confronted in real-world applications.

Unfortunately, there are few general practical feedback control methods for nonlinear MIMO systems [5], [6], although many methods exist for linear MIMO systems. The behavior of nonlinear systems is much more complicated and rich than that of linear systems because of both the lack of linearity and the associated superposition property [5], [7].

This paper presents neural dynamic optimization (NDO) as a practical method for nonlinear MIMO control systems. In order to handle the complexities of nonlinearity and accommodate the demand for high-performance MIMO control systems, NDO takes advantage of brain-like computational structures—neural networks—as well as optimal control theory. Our formulation allows neural networks to serve as nonlinear feedback controllers optimizing the performance of the resulting control systems. NDO is thus an offspring of both neural networks and optimal control theory.

In optimal control theory [8]–[10], the optimal solution to a nonlinear MIMO control problem may be obtained from the Hamilton–Jacobi–Bellman equation (HJB) or the Euler–Lagrange (EL) equations . The two sets of equations provide the same solution in different forms: EL leads to a sequence of optimal control vectors, called feedforward optimal control (FOC); HJB yields a nonlinear optimal feedback controller, called dynamic programming (DP). DP produces an optimal solution that is able to reject disturbances and uncertainties as a result of feedback. Unfortunately, computation and storage requirements associated with DP solutions can be problematic, especially for high-order nonlinear systems, a problem known as the curse of dimensionality [8], [10], [11]. The linear quadratic regulator (LQR) derived from DP has thus been applied to designing controllers for nonlinear MIMO systems through the linearization of the systems around an equilibrium point [9], [10]. However, such linearization imposes limitations on the stability and performance of closed-loop systems since it causes a loss of information about large motions and is valid only near the equilibrium point.

Neural networks [12]–[14], in contrast, have a massively parallel distributed structure, an ability to learn and generalize,[1] and a built-in capability to adapt their synaptic weights to changes in the surrounding environments. Neural networks are inherently nonlinear—a very important property, particularly if the underlying physical mechanisms for the systems are highly nonlinear. More importantly, they can approximate any nonlinear function to a desirable accuracy [15]–[17].

Manuscript received January 16, 2000; revised February 11, 2001. This work was supported in part by the National Aeronautics and Space Administration under Contract NCC2-1020, the Office of Naval Research under Contract N00014-97-1-0947, and the National Science Foundation under Contract ECS–9522085. This paper was recommended by Associate Editor L. O. Hall.

C.-Y. Seong was with the Information Systems Laboratory, Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA. He is now with Fineground Networks, Campbell, CA 95008 USA (e-mail: chang@fineground.com).

B. Widrow is with the Information Systems Laboratory, Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: widrow@stanford.edu)

Publisher Item Identifier S 1083-4419(01)05665-5.

[1]Generalization refers to the neural network producing reasonable outputs for inputs not encountered during training (or learning).

Thus, we propose an approximate technique for solving the DP problem based on neural network techniques that provides many of the performance benefits (e.g., optimality and feedback) of DP and benefits from the numerical properties of neural networks. We formulate neural networks to approximate optimal feedback solutions whose existence DP justifies. In other words, neural networks serve as building blocks for finding the optimal solution. As a result, NDO closely approximates—with a reasonable amount of computation and storage—optimal feedback solutions to nonlinear MIMO control problems that would be very difficult to implement in real-time with DP.

Incidentally, the method developed in the paper is not the same as neurodynamic programming [18], which involves approximating the optimal solutions for discrete-state systems where the number of states is finite and the number of available controls at each state is also finite. However, the two methods are related since they try to achieve the same goal: approximating the optimal feedback solution using neural networks.

The research presented in this paper was inspired by the works of Nguyen [19] and Plumer [20]. Nguyen showed the possibility of using neural networks in controlling a system with high nonlinearities by training them successfully to back up a trailer-truck to a loading dock. He primarily introduced a time-horizon-control concept to *static* neural network controllers, unraveling in time the feedback loop of the system-controller configuration to produce a large equivalent feedforward network. He applied the backpropagation algorithm [21], [22] to the large network to train the neural controller, which is called the backpropagation-through-time algorithm. He also pointed out the possibility of a connection between his work and optimal control theory. Plumer [20] extended Nguyen's work; he explored a connection between the neural network control and optimal feedback control. He illustrated their connection by applying the *static* neural network controller to some nonlinear SISO systems in finite horizon problems, such as terminal control. However, static controllers may have limitations in finite horizon problems because the optimal feedback solutions to such problems are inherently *time-varying* [8], [9].

NDO can produce the time-varying as well as time-invariant optimal feedback solutions to nonlinear MIMO control problems. NDO can handle not only finite-horizon problems (e.g., terminal control) but also infinite-horizon problems (e.g., regulation and tracking control). NDO can produce the neural controllers that are able to minimize the effect of disturbances and uncertainties that may be confronted in real-world applications.

This paper mainly describes the background and motivations for the development of NDO, while the two other subsequent papers [23], [24] of this topic present the theory of NDO and demonstrate the method with several applications including control of autonomous vehicles and of a robot arm, respectively.

Four sections comprise this paper. Section II is an overview of optimal control theory, including DP, the LQR, and FOC. We point out the problematic aspects of optimal control theory, which in turn provide the motivation for this research. In any case, optimal control theory remains useful because it provides theoretical limits to the performance of NDO, the method presented in the paper. Section III reviews neural networks and their properties, emphasizing their capability as general function approximators. Section IV provides conclusions.

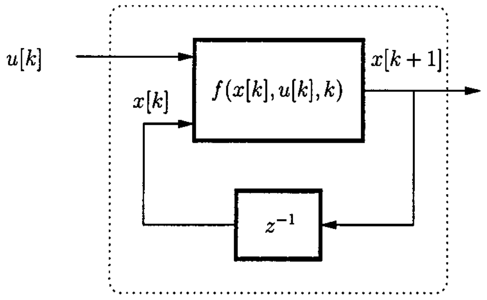


Fig. 1. NTV-MIMO system.

II. Optimal Control Theory

This section overviews optimal control theory, especially DP, the LQR, and FOC.

A. Dynamic Programming (DP)

DP finds optimal feedback solutions to nonlinear MIMO control problems. This approach fully exploits the state concept. Bellman derived the optimality condition for the optimal value of a cost function, and the control as a function of state using the principle of optimality [11], [25], [26]. A more descriptive name would be *nonlinear optimal feedback control*.

Suppose we have a discrete nonlinear-time-varying (NTV) MIMO system described by a state equation of the form

$$x[k+1] = f(x[k], u[k], k) \tag{1}$$

where $x[k] \in \mathcal{R}^n$ is the state vector, and $u[k] \in \mathcal{R}^m$ is the control input vector. The future state $x[k+1]$ depends on the current state $x[k]$ and the current input vector $u[k]$ through a nonlinear function $f(\cdot)$, which may have explicit time dependency. Fig. 1 represents the NTV-MIMO system.

A fairly general optimization approach finds the optimal feedback control to minimize a cost function of the form

$$J = \phi(x[N], N) + \sum_{k=0}^{N-1} L(x[k], u[k], k) \tag{2}$$

subject to (1) with the time horizon N and the function f specified. The function $\phi(\cdot)$ penalizes the state $x[N]$ at the time horizon $k = N$, and the function $L(\cdot)$ penalizes the states $x[k]$ and the control inputs $u[k]$ through $k = 0$ to $N - 1$. The functions $\phi(\cdot)$ and $L(\cdot)$ may have explicit time dependency, according to the goal of control.

The optimal solution to the problem can be obtained by solving the discrete HJB equation.

Theorem 1: For the given dynamic optimization problem of minimizing the cost function

$$J = \phi(x[N], N) + \sum_{k=0}^{N-1} L(x[k], u[k], k) \tag{3}$$

subject to

$$x[k+1] = f(x[k], u[k], k) \tag{4}$$

where N and the function f are specified, the optimal feedback solutions are obtained from the discrete HJB equation

$$J^o(x[k], k) = \min_{u[k]} L(x[k], u[k], k) + J^o(x[k+1], k+1) \quad (5)$$

with the initial condition

$$J^o(x[N], N) = \phi(x[N], N). \quad (6)$$

Proof: The proof of the theorem is given in the literature [9], [11], [27]. □

First, HJB justifies the existence of a nonlinear optimal feedback solution. Second, in order to obtain the optimal solution, we need to find the optimal cost function[2] $J^o(x[N], N)$ at the state $x[N]$ at the time horizon $k = N$ as an initial condition. Then, we solve HJB recursively backward in time over state space.

However, HJB can be very difficult to solve for high-order nonlinear systems. To make the computational procedure feasible it is necessary to quantize the admissible state and control values into a finite number of levels. The computational procedure does not yield an analytical expression, but the optimal feedback control law is implemented by extracting the control values from a storage device that contains the solution of HJB in tabular form. To calculate the optimal control sequence for a given initial condition, we enter the storage location corresponding to the specified initial condition and extract the control value $u[0]$ and the minimum cost. Next, by solving the state equation we determine the state of the system at $k = 1$, which results from applying $u[0]$ at $k = 0$. The resulting value of $x[1]$ is then used to reenter the table and extract $u[1]$, and so on. The optimal controller is physically realized by a table look-up device.

In addition, storage requirements of the HJP solutions increase very rapidly with the state dimension. Bellman referred to this drawback as *the curse of dimensionality*. To appreciate the nature of the problem, we may consider a fourth-order SISO system with 0 quantization levels in each state coordinate as well as 0 time steps. The solution of HJB to the control problem requires at least $10^2 \times 10^2 \times 10^2 \times 10^2 \times 10^2 = 10^{10}$ storage locations. Interpolation may also be required when one is using stored data to compute an optimal control sequence. For example, if the optimal control applied at some value of $x[k]$ drives the system to a state value $x[k+1]$ that is halfway between two points where the optimal controls are 1 and 0, then by linear interpolation the optimal control is 0.5. Naturally, the degree of approximation depends on the separation of the grid points in state coordinates, the interpolation scheme used, the system dynamics, and the cost function. A finer grid generally means greater accuracy, but also means increased storage requirements and computation time.

Linear Quadratic Problems: DP can be subject to intensive computation and extensive storage requirements, especially for high-order nonlinear systems. However, it produces important results for *linear quadratic problems* where systems are linear and cost functions are quadratic.

[2]The optimal cost function is also called the optimal return function or the optimal cost-to-go function.

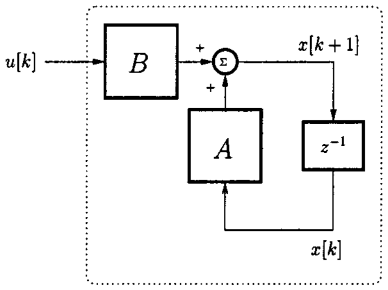


Fig. 2. LTI-MIMO system.

Consider a discrete linear-time-invariant (LTI) MIMO system described by a state equation of the form

$$x[k+1] = Ax[k] + Bu[k] \quad (7)$$

where $x[k] \in \mathcal{R}^n$ is the state vector and $u[k] \in \mathcal{R}^m$ is the control input vector. Fig. 2 depicts the discrete LTI-MIMO system. We want to find the optimal feedback control to minimize a constant-coefficient quadratic cost function

$$J = \frac{1}{2}x^{\mathrm{T}}[N]S[N]x[N] + \frac{1}{2}\sum_{k=0}^{N-1}(x^{\mathrm{T}}[k]Qx[k] + u^{\mathrm{T}}[k]Ru[k]) \quad (8)$$

with $S[N] \geq 0$, $Q \geq 0$, and $R > 0$.[3]

By applying the HJB (5) to the linear quadratic problem we may obtain the following result.

Theorem 2: For the linear quadratic dynamic optimization problem of minimizing the cost function

$$J = \frac{1}{2}x^{\mathrm{T}}[N]S[N]x[N] + \frac{1}{2}\sum_{k=0}^{N-1}(x^{\mathrm{T}}[k]Qx[k] + u^{\mathrm{T}}[k]Ru[k]) \quad (9)$$

subject to

$$x[k+1] = Ax[k] + Bu[k] \quad (10)$$

the optimal feedback controls are obtained from the following set of equations:

$$\begin{aligned} u^o[k] =& -K[k]x[k] && (11)\\ K[k] =& (B^{\mathrm{T}}S[k+1]B + R)^{-1}B^{\mathrm{T}}S[k+1]A && (12)\\ J^o[k] =& \frac{1}{2}x^{\mathrm{T}}[k]S[k]x[k] && (13)\\ S[k] =& Q + A^{\mathrm{T}}S[k+1]A \\ & - A^{\mathrm{T}}S[k+1]B(R + B^{\mathrm{T}}S[k+1]B)^{-1} \\ & \times B^{\mathrm{T}}S[k+1]A && (14) \end{aligned}$$

where the final condition $S[N]$ for (14) is given in (9).

Proof: The proof of the theorem is given in [9] and [10]. □

The HJB equation produces the set of solution equations for the linear quadratic problems. In order to obtain the optimal solution to the linear quadratic problems, we need to solve the Riccati (14) recursively backward in time. Then, we compute the

[3]If the system and weighting matrices vary according to time, the result to follow still holds.

time-varying Kalman gain $K[k]$, using (12). The Riccati equation is much easier to solve than the HJB equation.

We point out that the optimal feedback solution to the linear quadratic problem is a linear function of the state $x[k]$ rather than a nonlinear one. In other words, DP shows that linear state feedback is the best form of feedback for linear quadratic problems. Thus the resulting closed-loop system is also *linear* like the open-loop system, which is an important result for linear control systems. Note also that optimal feedback controls for LTI systems with *constant-coefficient* quadratic cost functions are *time-varying* because the Kalman gain $K[k]$ has explicit time-dependency. However, we may prefer a constant Kalman gain K, which enables the closed-loop system to be LTI like the open-loop system. The next subsection will discuss the conditions under which we can obtain a constant Kalman gain for the linear quadratic problem.

B. Linear Quadratic Regulator (LQR)

In Section II-A, where we considered the application of DP to linear quadratic problems, the closed-loop system is given by

$$x[k+1] = (A - BK[k])x[k]. \tag{15}$$

The resulting closed-loop system is *time-varying* since the optimal gain $K[k]$ is time-varying even though the open-loop system is *time invariant*.

However, this time-varying feedback is not always convenient to implement: it requires the storage of an entire sequence of $m \times n$ matrices. Accordingly, we are interested in a constant feedback gain. As one candidate for a constant feedback gain, we consider the limit of the optimal $K[k]$ as the time horizon N goes to infinity. We call this case the LQR [8]–[10].

If $S[k]$ does converge, then $S \triangleq S[k] = S[k+1]$ for a large N. Thus, in the limit, the Riccati (14) becomes the algebraic Riccati equation (ARE)

$$S = A^{\mathrm{T}}SA - A^{\mathrm{T}}SB(R + B^{\mathrm{T}}SB)^{-1}B^{\mathrm{T}}SA + Q \tag{16}$$

which does not depend on time. The limiting solution S to the Riccati (14) is clearly a solution of the ARE (16). The corresponding steady-state *Kalman gain* is

$$K = (B^{\mathrm{T}}SB + R)^{-1}B^{\mathrm{T}}SA \tag{17}$$

which is a constant feedback gain. The resulting closed-loop system is

$$x[k+1] = (A - BK)x[k] \tag{18}$$

which is also linear time invariant like the open-loop system.

The following theorem tells us when there exists a bounded limiting solution S.

Theorem 3: Let (A, B) be controllable, then there is a bounded limiting solution S to (14). Furthermore, S is a positive semidefinite solution to the ARE (16).

Proof: The proof of the theorem is given in [9] and [10]. □

The next theorem provides us with the conditions on the uniqueness of the bounded limiting solution S as well as the asymptotical stability of the closed-loop system in terms of the observability of the system through the fictitious output.

Theorem 4: Let C be a square root of the intermediate-state weighting matrix, so that $Q = C^{\mathrm{T}}C \geq 0$, and suppose $R > 0$. Suppose (A, C) is observable. Then (A, B) is controllable if and only if we have the following:

1) There is a unique positive definite limiting solution S to the Riccati (14). Furthermore, S is the unique positive definite solution to the ARE (16).
2) The closed-loop system

$$x[k+1] = (A - BK)x[k]$$

is asymptotically stable, where K is given by (17).

Proof: The proof of the theorem is given in [9] and [10]. □

The controllability of (A, B), which is a dynamical property of the open-loop system, is crucial to guarantee that there exists a unique stabilizing optimal feedback solution to the linear quadratic regulation problem. We can pick without any difficulties a proper $Q = C^{\mathrm{T}}C$ such that the observability of (A, C) is guaranteed. In other words, the open-loop system at least needs to be controllable to guarantee the unique optimal solution. We also point out that the ARE (16) is nonlinear with respect to unknown parameter S. However, we can solve it easily using the DLQR command in MATLAB.

The LQR has been applied to nonlinear MIMO feedback control design through linearization around an equilibrium point. However, such linearization may impose limitations on the stability and performance of closed-loop systems since it causes a loss of information about large motions and is valid only near the equilibrium point.

C. Feedforward Optimal Control (FOC)

FOC is an alternative method for nonlinear MIMO control problems. It treats the same dynamic optimization problem as DP and provides the same optimal solution in a different form: it finds the sequence of optimal control vectors with a specified initial state instead of the feedback solution for all possible initial states [8], [10]. It finds the optimal control solution relatively easily. However, its solution can be very sensitive to disturbances and uncertainties because of the lack of a feedback mechanism.

Let us consider the same form of the discrete NTV-MIMO system as in the case of DP

$$x[k+1] = f(x[k], u[k], k) \tag{19}$$

with

$$x[0] = x_0 \tag{20}$$

where $x[k] \in \mathcal{R}^n$ is the state vector and $u[k] \in \mathcal{R}^m$ is the control input vector. Note that an initial state x_0 is specified here and this was not done in the case of DP.

Here, we want to find the sequence of optimal control vectors $u[k]$ for $k = 0, \ldots, N-1$ to minimize the same form of the cost function as in the case of DP.

$$J = \phi(x[N], N) + \sum_{k=0}^{N-1} L(x[k], u[k], k) \tag{21}$$

subject to (19) and (20) with N, x_0, and the function f specified. Thus its optimal solution is not closed-loop but open-loop because $u[k]$ is no longer a function of the state $x[k]$. However, this control problem is relatively easy to solve because it becomes a parameter optimization problem by treating $\{u[k]\}$ as a set of unknown parameters.

The optimal control vector sequence $\{u[k]\}$ is obtained by solving the discrete EL equations.

Theorem 5: For the given dynamic optimization problem of minimizing the cost function

$$J = \phi(x[N], N) + \sum_{k=0}^{N-1} L(x[k], u[k], k) \tag{22}$$

subject to

$$x[k+1] = f(x[k], u[k], k) \tag{23}$$

with

$$x[0] = x_0 \tag{24}$$

where N, x_0, and the function f are specified, the sequence of the optimal control vectors $\{u[k]\}$ can be obtained from the discrete EL equations.

Proof: The proof of the theorem is given in [8]. □

Note that the EL equations consist of the system equation, the adjoint system equation, and the optimality condition:

the system equation—

$$x[k+1] = f(x[k], u[k], k), \; k = 0, \ldots, N-1 \tag{25}$$

with $x[0] = x_0$,

the adjoint system equation—

$$\delta_x[k] = \frac{\partial f(k)}{\partial x[k]}^{\mathrm{T}} \delta_x[k+1] + \frac{\partial L(k)}{\partial x[k]}^{\mathrm{T}}, \; k = N-1, \ldots, 0 \tag{26}$$

with $\delta_x[N] = (\partial\phi(N)/\partial x[N])^{\mathrm{T}}$

and the optimality condition—

$$\delta_u[k] = \frac{\partial f(k)}{\partial u[k]}^{\mathrm{T}} \delta_x[k+1] + \frac{\partial L(k)}{\partial u[k]}^{\mathrm{T}} = 0, \; k = N-1, \ldots, 0 \tag{27}$$

where we define

$$f(k) \triangleq f(x[k], u[k], k) \tag{28}$$

$$L(k) \triangleq L(x[k], u[k], k) \tag{29}$$

$$\phi(N) \triangleq \phi(x[N], N). \tag{30}$$

The system equation has an initial condition at the initial time $k = 0$, while the adjoint system equation has an initial condition at the end of the time horizon $k = N$. This is a two-point boundary value problem (TPBVP). Since both $x[k]$ and $\delta_x[k]$ have dimension n, $u[k]$ has dimension m, and $k = 0, \ldots, N-1$, this TPBVP has $N(2n+m)$ unknowns and an equal number of equations. It can be solved as a large set of simultaneous nonlinear algebraic equations (e.g., using the command FSOLVE in MATLAB). However, a more efficient method of solution, to be described in the next subsection, exploits the sequential nature of the problem.

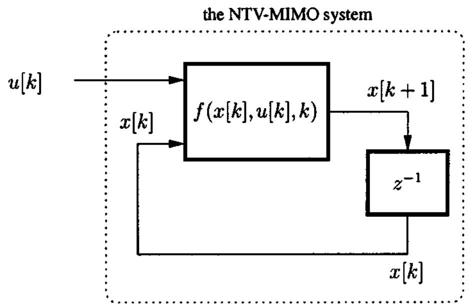


Fig. 3. Forward sweep of FOC.

1) Numerical Solution with Steepest Descent Method: We may apply steepest descent to obtain a numerical algorithm. The steepest descent algorithm for FOC, which exploits the sequential nature of EL, consists of three steps:

1) forward sweep;
2) backward sweep;
3) update of control vector sequence.

Step 1) Forward sweep

$$x[k+1] = f(x[k], u[k], k) \tag{31}$$

with $x[0] = x_0, \; k = 0, \ldots, N-1$.

The forward sweep equation is the same as the system equation. First of all, we need to make a reasonable guess about $u[k]$ from $k = 0$ through $N-1$. Then with the specified initial state x_0, we can compute $x[k]$ through $k = 1$ to N, using the system equation. Fig. 3 depicts the forward sweep.

Step 2) Backward sweep

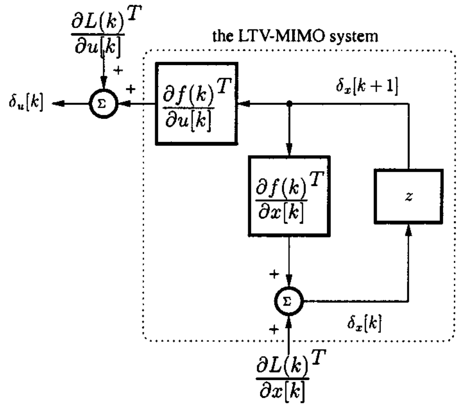
$$\delta_x[k] = \frac{\partial f(k)}{\partial x[k]}^{\mathrm{T}} \delta_x[k+1] + \frac{\partial L(k)}{\partial x[k]}^{\mathrm{T}} \tag{32}$$

$$\delta_u[k] = \frac{\partial f(k)}{\partial u[k]}^{\mathrm{T}} \delta_x[k+1] + \frac{\partial L(k)}{\partial u[k]}^{\mathrm{T}} \tag{33}$$

with $\delta_x[N] = (\partial\phi(N)/\partial x[N])^{\mathrm{T}} \; k = N-1, \ldots, 0$.

The backward sweep equations consist of the adjoint system equation and the optimality condition. These equations can be represented by a block diagram, as illustrated in Fig. 4. We have the co-state $\delta_x[k]$, two co-inputs $(\partial L(k)/\partial x[k])^{\mathrm{T}}$ and $(\partial L(k)/\partial u[k])^{\mathrm{T}}$ derived from the cost function, and the co-output $\delta_u[k]$.

This system is anti-causal and linear-time-varying (LTV) because the current co-state $\delta_x[k]$ depends on the future co-state $\delta_x[k+1]$ through (32) and the coefficients such as $(\partial f(k)/\partial x[k])^{\mathrm{T}}$ and $(\partial f(k)/\partial u[k])^{\mathrm{T}}$ are explicitly time-dependent. However, we can compute $\delta_x[k]$ and $\delta_u[k]$ backward in time because we have already obtained the sequences of $x[k]$ and $u[k]$ from the forward sweep.



$\frac{\partial L(k)}{\partial u[k]}^T$
the LTV-MIMO system
$\delta_u[k]$
$\frac{\partial f(k)}{\partial u[k]}^T$
$\delta_x[k+1]$
$\frac{\partial f(k)}{\partial x[k]}^T$
z
$\delta_x[k]$
$\frac{\partial L(k)}{\partial x[k]}^T$

Fig. 4. Backward sweep of FOC.

Step 3) Update of control vector sequence

$$\Delta u[k] = -\mu\delta_u[k] \quad (34)$$

$$u[k] \leftarrow u[k] + \Delta u[k]. \quad (35)$$

Then we update $u[k]$ using (34) and (35). We repeat the three steps until $\Delta u[k]$ is small.

FOC is an alternative method for nonlinear MIMO control problems because it is relatively easy and computationally tractable to find its solution. The computation requirement of the FOC algorithm for each iteration is composed of three components: forward sweep, backward sweep, update of control vector sequence. The forward sweep requires approximately $n(n+m) \times N$ multiplications and additions because it evaluates the system equation from $k=0$ to $N-1$. The backward sweep needs roughly $(n \times n + m \times n) \times N$ operations because it computes Jacobians $(\partial f(k)/\partial x[k])^{\mathrm{T}}$ and $(\partial f(k)/\partial u[k])^{\mathrm{T}}$ at each time step. The update of control vector sequence needs approximately $N \times m$ operations. So the total number of multiplications and additions for each iteration using the FOC algorithm is approximately

$$(2n(n+m)+m)N. \quad (36)$$

The memory requirement of the algorithm is approximately

$$(n+m)N \quad (37)$$

since it needs to store the state $x[k]$ and the control input $u[k]$ from $k=0$ to $N-1$. Note that the memory requirement of FOC depends linearly on the dimension of the state n and on the number of control inputs m, while that of DP increases exponentially with m and n.

However, FOC finds just a single trajectory for a given initial state. A severe problem for FOC is that its solutions are very sensitive to disturbances and uncertainties because of the lack of feedback.

2) Linear Quadratic Problems: The application of FOC to linear quadratic problems reveals—because FOC employs a simpler form of the equations—an important structural property of computation, the so-called *duality* of the forward and the backward sweeps.

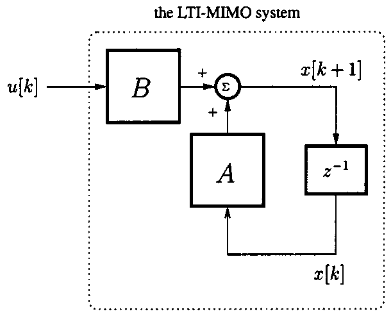


Fig. 5. Forward sweep of FOC for linear quadratic problems.

By applying EL and the steepest descent method to the linear quadratic problem, we obtain the following three steps.

Step 1) Forward sweep

$$x[k+1] = Ax[k] + Bu[k] \quad (38)$$

with $x[0] = x_0, k = 0, \ldots, N-1$.

The forward sweep equation is the same as the system equation. Fig. 5 depicts the forward sweep for linear quadratic problems.

Step 2) Backward sweep

$$\delta_x[k] = A^{\mathrm{T}}\delta_x[k+1] + Qx[k] \quad (39)$$

$$\delta_u[k] = B^{\mathrm{T}}\delta_x[k+1] + Ru[k] \quad (40)$$

with $\delta_x[N] = S[N]x[N],\ k = N-1, \ldots, 0$.

The backward sweep equations consist of the adjoint system equation and the optimality condition as in the general case of FOC. These equations can be represented by the block diagram illustrated in Fig. 6. We have the co-state $\delta_x[k]$, two co-inputs $Qx[k]$ and $Ru[k]$ derived from the cost function, and the co-output $\delta_u[k]$. Comparing Fig. 5 with Fig. 6 reveals the duality of the backward and the forward sweeps.

- The coefficient matrices of the backward sweep are the transpose of the coefficient matrices (A, B) of the forward sweep.
- The time advance operator in the backward sweep replaces the time delay operator in the forward sweep.
- The directions of signal flows in the backward sweep are reversed to those in the feedforward sweep.
- The summing junctions in the backward sweep replace the branching points in the forward sweep, and vice versa.

Step 3 Update of control vector sequence:

$$\Delta u[k] = -\mu\delta_u[k] \quad (41)$$

$$u[k] \leftarrow u[k] + \Delta u[k]. \quad (42)$$

The update equations are the same as in the general case of FOC.

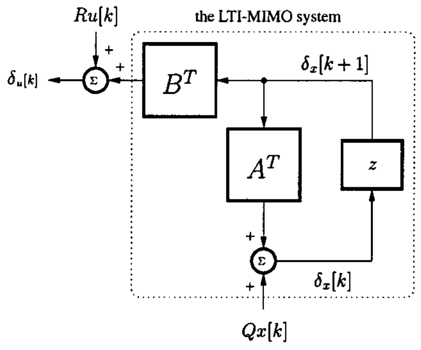


Fig. 6. Backward sweep of FOC for linear quadratic problems.

III. Neural Networks

This section reviews artificial neural networks, commonly referred to as *neural networks*. Neural networks are attractive in that they can approximate any nonlinear function to a desirable accuracy. In addition, neural networks possess other important properties. Neural networks have massively parallel distributed structures; implemented in hardware form, they can be inherently *fault tolerant* in the sense that their performance degrades gracefully under adverse operating conditions [28]. Neural networks can learn and generalize; generalization refers to the neural network producing reasonable interpolated outputs for inputs not encountered during learning (training) [29]. Neural networks are inherently nonlinear—a very important property, particularly if the underlying physical mechanism responsible for the generation of signals (e.g., a nonlinear MIMO system) is highly nonlinear. Neural networks have the built-in capability to *adapt* their synaptic weights to changes in the surrounding environment; a neural network trained to operate in a specific environment can be easily *retrained* to deal with minor changes in the operating environmental conditions [12].

An artificial neural network results from arranging a neuron model into various configurations. The most popular configuration is the multilayer feedforward network [12], [13], [29]. As illustrated in Fig. 7, it has layers of neurons, where the inputs to each neuron on a layer are identical, and equal to the collection of outputs from the previous layer (plus the augmented value "1"). The beginning layer of the network is called the *input layer*, its final layer is called the *output layer*, and all other layers of neurons are called the *hidden layers*. For multilayer neural networks, we use the following notation: $\mathcal{N}_{\alpha:\beta:\gamma\ldots}$. This means: *The neural network has α external inputs. There are β neurons in the network's first layer, γ neurons in the second layer, and so on.*

In addition, a single neuron extracted from the lth layer of an L-layer network is depicted in Fig. 7. The weight $w_{i:j}^{(l)}$ denotes connections between neuron i in layer $l-1$ and neuron j in layer l. The weights $w_{i:j}^{(l)}$ are arranged in the form of the column vector such that $W = [\ldots w_{i:j}^{(l)} \ldots]$. The weight vector W is an n_w-dimensional vector where n_w is the total number of weights in the network. The output (activation value) of the j-th neuron in layer l is represented by the variable $a_j^{(l)}$. Note that the jth neuron in layer l performs a weighted sum on its own input signals and passes the sum through an activation function $f(\,)$ to generate its own output $a_j^{(l)}$. The outputs $a_i^{(L)}$ in the final L-th layer corresponding to the overall outputs of the network. For

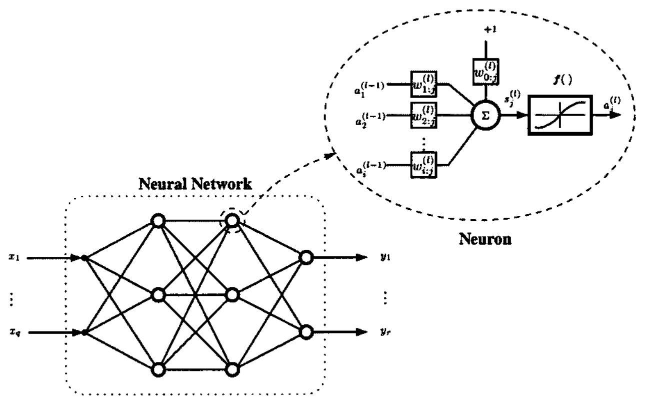


Fig. 7. Multilayer feedforward neural network consists of layers of neurons with each layer fully connected to the next layer. The network receives its q-dimensional input vector and generates its r-dimensional output vector.

TABLE I
Multilayer Neural Network Notation

$\mathcal{N}_{\alpha:\beta:\gamma\ldots}$	neural network has α external input vectors, β neurons in its first layer, γ neurons in its second layer, and so on
$w_{i:j}^{(l)}$	weight connecting neuron i in layer $l-1$ to neuron j in layer l
$w_{0:j}^{(l)}$	bias weight for neuron j in layer l
$s_j^{(l)} = \sum_i w_{i:j}^{(l)} a_i^{(l-1)} + w_{0:j}^{(l)}$	summing junction for neuron j in layer l
$a_j^{(l)} = f(s_j^{(l)})$	output (activation value) of neuron j in layer l
$x_i = a_i^{(0)}$	i-th external input to neural network
$y_i = a_i^{(L)}$	i-th overall output of neural network

convenience we define a variable y_i for the outputs. Thus $y_i \triangleq a_i^{(L)}$. We also define x_i as the external inputs to the network. The inputs may be viewed as a 0th layer that gives the relation $x_i \triangleq a_i^{(0)}$. This notation is summarized in Table I.

Now define an input vector $\boldsymbol{x}$ and output vector $\boldsymbol{y}$ as follows (see Fig. 7):

$$\boldsymbol{x} \triangleq [x_1\, x_2 \ldots x_q]^{\mathrm{T}}$$
$$\boldsymbol{y} \triangleq [y_1\, y_2 \ldots y_r]^{\mathrm{T}}.$$

Then, the feedforward network forms a mapping, $\boldsymbol{y} = \mathcal{N}(\boldsymbol{x}; W)$, from the inputs of the input layer to the outputs of the final layer parameterized by the weight vector W. Given the current model of the neuron, with fixed weights, this mapping is *static*; there are no internal dynamics or memory devices. Nevertheless, the following theorem makes the multilayer feedforward neural network a powerful tool for computation.

Theorem 6: (General function approximators):

Given any $\epsilon > 0$ and any L_2 function $\varphi : \mathcal{X} \subset \mathcal{R}^q \rightarrow \mathcal{R}^r$, there exists a two-layer feedforward neural network that can approximate $\varphi(\boldsymbol{x})$ to within ϵ mean squared error accuracy.

Proof: The proof of the theorem is given in [15]–[17]. $\square$

L_2 functions are square-integrable functions over a bounded set $\mathcal{X} \subset \mathcal{R}^n$. The function space L_2 includes every function that could ever arise in a practical problem. For example, it includes continuous functions and all discontinuous functions that are piecewise continuous on a finite number of subsets of a set $\mathcal{X}$. L_2 also includes more complicated functions that are only of mathematical interest. Thus, the neural network can approximate practically any nonlinear function to a desirable accuracy. Theoretically, two types of nonlinear activation function can be used: one is a signum (i.e., threshold) function and the other is a sigmoid function [12], [16], [22]. In practice, however, we desire activation functions that are sigmoidal. As a special case, linear neural networks—i.e., matrices that have linear activation functions—can approximate any linear function. However, this theorem does not explain how to find the weights of the neural network to approximate a nonlinear function to a desirable accuracy.

IV. Conclusion

As part of the background for the development of NDO, we overview optimal control theory, especially DP, the LQR, and FOC. We point out the problematic aspects as well as the performance benefits of each individual method. In particular, DP produces the optimal solution that is able to reject disturbances and uncertainties as a result of feedback. However, computation and storage requirement associated with DP solutions can be problematic, especially for high-order nonlinear systems. Thus, in the companion papers [23], [24], we propose NDO as an approximate technique for solving the DP problem based on neural network techniques that provides many of the performance benefits (e.g., optimality and feedback) of DP and benefits from the numerical properties of neural networks. Reference [23] presents the theory of NDO, and [24] demonstrates the method on several applications including control of autonomous vehicles and of a robot arm, respectively.

References

[1] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, Third ed. Reading, MA: Addison-Wesley, 1994.
[2] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*, Third ed. Reading, MA: Addison-Wesley, 1998.
[3] S. P. Boyd, *Linear Controller Design: Limits of Performance*. Englewood Cliffs, NJ: Prentice-Hall, 1991.
[4] T. Kailath, *Linear Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1980.
[5] J. E. Slotine and W. Li, *Applied Nonlinear Control*. Englewood Cliffs, NJ: Prentice-Hall, 1991.
[6] A. Isidori, *Nonlinear Control Systems: An Introduction*. New York: Springer-Verlag, 1989.
[7] H. K. Khalil, *Nonlinear Systems*. New York: Macmillan, 1992.
[8] A. E. Bryson, *Dynamic Optimization*. Menlo Park, CA: Addison-Wesley-Longman, 1999.
[9] F. L. Lewis, *Optimal Control*, 2nd ed. New York: Wiley, 1995.
[10] R. F. Stengel, *Optimal Control and Estimation*. New York: Dover, 1994.
[11] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.
[12] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1999.
[13] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: Perceptron, Madaline, and backpropagation," *Proc. IEEE*, vol. 78, pp. 1415–42, Sept. 1990.
[14] R. Hecht-Nielsen, *Neurocomputing*. Reading, MA: Addison-Wesley, 1990.
[15] A. N. Kolmogorov, "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition," *Doklady Akademii Nauk SSR*, vol. 114, pp. 953–56, 1957.
[16] R. Hecht-Nielsen, "Kolmogorov's mapping neural network existence theorem," in *Proc. IEEE 1st Int. Conf. Neural Networks*, vol. III, San Diego, CA, 1987, pp. 11–4.
[17] K. Hornik, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, pp. 359–66, 1989.
[18] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific, 1996.
[19] D. Nguyen, "Applications of neural networks in adaptive control," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1991.
[20] E. S. Plumer, "Optimal terminal control uising feedforward neural networks," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1993.
[21] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard Univ., Cambridge, MA, 1974.
[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, D. E. Rumelhart and J. L. McClell, Eds. Cambridge, MA: MIT Press, 1986, vol. 1, ch. 8.
[23] C. Seong and B. Widrow, "Neural dynamic optimization for control systems—Part II: Theory," *IEEE Trans. Syst., Man, Cybern. B*, vol. 31, pp. 490–501, Aug. 2001.
[24] ——, "Neural dynamic optimization for control systems—Part III: Applications," *IEEE Trans. Syst., Man, Cybern. B*, vol. 31, pp. 502–513, Aug. 2001.
[25] R. E. Bellman and S. E. Dreyfus, *Applied Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1962.
[26] R. E. Bellman and R. E. Kalaba, *Dynamic Programming and Modern Control Theory*. New York: Academic, 1965.
[27] D. G. Luenberger, *Introduction to Dynamic Systems: Theory, Models, & Applications*. New York: Wiley, 1979.
[28] G. R. Bolt, "Fault tolerance in artificial neural networks," Ph.D. dissertation, York University, York, ON, Canada, 1992.
[29] D. Rumelhart and J. McClell, Eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press, 1986, vol. 1.

Chang-Yun Seong (M'01) received the B.S. and M.S. degrees in aerospace engineering from Seoul National University, Seoul, Korea, in 1990 and 1992, respectively, and the M.S. degree in electrical engineering and the Ph.D. degree in aeronautics and astronautics from Stanford University, Stanford, CA, in 1998 and 2000, respectively.

He was a Research Engineer with the Systems Engineering Research Institute (SERI/KIST), Taejon, Korea, before he pursued his graduate studies at Stanford University. He is currently a Research Engineer with Fineground Networks, Campbell, CA. His research interests include neural networks, optimization, control systems, and digital signal processing.

Dr. Seong is a member of AIAA. He was a recipient of the Outstanding Teaching Assistant Award from the AIAA Stanford Chapter in 1997.

Bernard Widrow (M'58–SM'75–F'76–LF'95) received the S.B., S.M., and Sc.D. degrees in electrical engineering from the Massachusetts Institute of Technology (MIT), Cambridge, in 1951, 1953, and 1956, respectively.

He joined the MIT faculty and taught there from 1956 to 1959. In 1959, he joined the faculty of Stanford University, Stanford, CA, where he is currently Professor of Electrical Engineering. He is Associate Editor of several journals and is the author of about 100 technical papers and 15 patents. He is coauthor of *Adaptive Signal Processing* (Englewood Cliffs, NJ: Prentice-Hall, 1985) and *Adaptive Inverse Control* (Englewood Cliffs, NJ: Prentice-Hall, 1996). A new book, *Quantization Noise*, is in preparation.

Dr. Widrow is a Fellow of AAAS. He received the IEEE Centennial Medal in 1984, the IEEE Alexander Graham Bell Medal in 1986, the IEEE Neural Networks Pioneer Medal in 1991, the IEEE Signal Processing Society Medal in 1998, the IEEE Millennium Medal in 2000, and the Benjamin Franklin Medal of the Franklin Institute in 2001. He was inducted into the National Academy of Engineering in 1995 and into the Silicon Valley Engineering Council Hall of Fame in 1999. He is a Past President and currently a Member of the Governing Board of the International Neural Network Society.