Neural Dynamic Optimization for Control Systems—Part II: Theory

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 theory of NDO, while the two other companion papers of this topic explain the background for the development of NDO and demonstrate the method with several applications including control of autonomous vehicles and of a robot arm, respectively.**

***Index Terms*—Dynamic programming, information time shift operator, learning operator, neural dynamic optimization, 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].

We present 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.

This paper mainly describes the theory of NDO, while the two other companion papers [8], [9] of this topic explain the background for the development 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 introduces NDO as an approximate technique for solving the dynamic programming (DP) problem based on neural network techniques. We formulate neural networks to approximate the optimal feedback solution. We introduce the learning operator and the information time shift operator, which characterize the learning process of the neural network in searching for the optimal feedback solution. The analysis of the learning operator provides not only a fundamental understanding of the learning process in neural networks but also some useful guidelines for selecting the number of weights of the neural network. Section III proves the equivalence of NDO to DP and feedforward optimal control (FOC) for linear quadratic problems, which demonstrates a connection between NDO and optimal control theory. Section IV has conclusions.

II. Neural Dynamic Optimization

This section presents 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 and real-time performance capabilities of neural networks. We formulate neural networks to approximate the optimal feedback solution. We call this method NDO.

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)05666-7.

A. Formulation

Let us consider the same discrete NTV-MIMO system, as in the case of DP and FOC

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

with

$$x[0] = x_0 \sim P(x_0) \tag{2}$$

where $x[k] \in \mathcal{R}^n$ is the state vector, $u[k] \in \mathcal{R}^m$ is the control input vector, x_0 is an initial state, and $P(x_0)$ denotes a probability distribution associated with x_0.

Here we formulate NDO so that it can find the optimal time-varying closed-loop solution and a family of trajectories over state space. In order to find the closed-loop solution, we may feed the state $x[k]$ to the multilayer feedforward neural network that is our controller. Since the multilayer feedforward network itself produces only a static mapping from the inputs of the input layer to the outputs of the final layer, the resulting feedback controller will be time-invariant. However, by feeding time information to the neural controller externally, we can allow the controller to have explicit time-dependency so that it can approximate the time-varying optimal feedback solution. In addition, the neural controller may receive some desired reference inputs to force the outputs of the system to follow them, depending on the goal of control. Fig. 1 depicts the overall configuration of the system and the neural controller. In order to find a family of optimal trajectories associated with different values of initial states, we treat the initial state x_0 as a random vector over state space with a probability $P(x_0)$, as shown in (2).

Then, we apply an optimal control framework so that the neural network can search the optimal feedback solution over state space. In other words, our objective is to find the optimal weight vector W of the multilayer feedforward neural network

$$u[k] = g(x[k], r[k+1], k; W) \tag{3}$$

minimizing the same form of the cost function as in the case of DP and FOC

$$J = E\left\{\phi(x[N], N) + \sum_{k=0}^{N-1} L(x[k], u[k], k)\right\}. \tag{4}$$

The expectation operator is employed in the cost function because we optimize the performance of the closed-loop system over an ensemble of trajectories associated with different values of initial states in the average sense. The neural network should be able to approximate the closed-loop optimal solution whose existence DP confirms. We note that this is a parameter optimization problem by treating W as a set of unknown parameters. Thus, we apply the calculus of variations to solve this parameter optimization problem.

B. Neural Euler–Lagrange (NEL) Equations

Let us find the optimal conditions for NDO. We treat both the system and control equations as constraints, and adjoin to the cost function (4) the system and the control equations, i.e.,

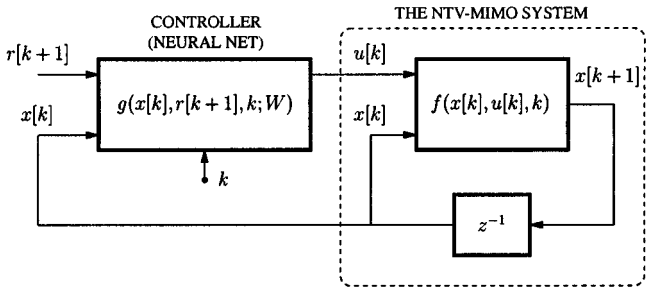

Fig. 1. Overall configuration of the system and the neural controller.

(1) and (3) with Lagrange multiplier vector sequences $\{\delta_x[k]\}$ and $\{\delta_u[k]\}$, respectively, as follows:

$$\begin{aligned}\bar{J} =& E\{\phi(N) + \delta_x^{\mathrm{T}}[0](x_0 - x[0]) \\ &+ \sum_{k=0}^{N-1}(L(k) + \delta_x^{\mathrm{T}}[k+1](f(k) - x[k+1]) \\ &+ \delta_u^{\mathrm{T}}[k](g(k;W) - u[k]))\}\end{aligned} \tag{5}$$

where we define

$$\begin{aligned} f(k) \triangleq& f(x[k], u[k], k) \\ L(k) \triangleq& L(x[k], u[k], k) \\ \phi(N) \triangleq& \phi(x[N], N) \\ g(k;W) \triangleq& g(x[k], r[k+1], k; W). \end{aligned}$$

Define the Hamiltonian as $H(k) \triangleq H(x[k], u[k], r[k+1], \delta_x[k+1], \delta_u[k], k, W)$

$$H(k) \triangleq L(k) + \delta_x^{\mathrm{T}}[k+1] f(k) + \delta_u^{\mathrm{T}}[k] g(k;W). \tag{6}$$

Substituting (6) into (5) and rearranging terms yields

$$\begin{aligned}\bar{J} =& E\Big\{\phi(N) - \delta_x^{\mathrm{T}}[N]x[N] + \delta_x^{\mathrm{T}}[0]x_0 \\ &+ \sum_{k=0}^{N-1}(H(k) - \delta_x^{\mathrm{T}}[k]x[k] - \delta_u^{\mathrm{T}}[k]u[k])\Big\}.\end{aligned} \tag{7}$$

Now consider differential changes in $\bar{J}$ caused by differential changes in x_0, $\{x[k]\}$, $\{u[k]\}$, $\{r[k+1]\}$, and W

$$\begin{aligned} d\bar{J} =& E\Big\{\left(\frac{\partial\phi(N)}{\partial x[N]} - \delta_x^{\mathrm{T}}[N]\right) dx[N] + \delta_x^{\mathrm{T}}[0] dx_0 \\ &+ \sum_{k=0}^{N-1}\left(\frac{\partial H(k)}{\partial x[k]} - \delta_x^{\mathrm{T}}[k]\right) dx[k] \\ &+ \sum_{k=0}^{N-1}\left(\frac{\partial H(k)}{\partial u[k]} - \delta_u^{\mathrm{T}}[k]\right) du[k] \\ &+ \sum_{k=0}^{N-1}\frac{\partial H(k)}{\partial r[k+1]} dr[k+1] \\ &+ \sum_{k=0}^{N-1}\frac{\partial H(k)}{\partial W} dW\Big\}. \end{aligned} \tag{8}$$

Since x_0 and $\{r[k+1]\}$ are specified and fixed for each trajectory, dx_0 and $\{dr[k+1]\}$ are both equal to zero. Optimal solu-

tions require that $d\bar{J} = 0$ for all choices of $\{dx[k]\}$, $\{du[k]\}$, and dW. Thus, we must satisfy the following equations:

the adjoint system equations—

$$\begin{aligned}\delta_x^{\mathrm{T}}[k] =& \frac{\partial H(k)}{\partial x[k]} \\ =& \frac{\partial L(k)}{\partial x[k]} + \delta_x[k+1]^{\mathrm{T}} \frac{\partial f(k)}{\partial x[k]} \\ &+ \delta_u[k]^{\mathrm{T}} \frac{\partial g(k;W)}{\partial x[k]} \end{aligned} \tag{9}$$

$$\begin{aligned}\delta_u^{\mathrm{T}}[k] =& \frac{\partial H(k)}{\partial u[k]} \\ =& \frac{\partial L(k)}{\partial u[k]} + \delta_x[k+1]^{\mathrm{T}} \frac{\partial f(k)}{\partial u[k]} \end{aligned} \tag{10}$$

for $k = 0, \ldots, N-1$

with boundary condition—

$$\delta_x^{\mathrm{T}}[N] = \frac{\partial \phi(N)}{\partial x[N]} \tag{11}$$

the optimal condition—

$$\begin{aligned} 0 =& E\left\{ \sum_{k=0}^{N-1} \frac{\partial H(k)}{\partial W} \right\} \\ =& E\left\{ \sum_{k=0}^{N-1} \delta_u[k]^{\mathrm{T}} \frac{\partial g(k;W)}{\partial W} \right\}. \end{aligned} \tag{12}$$

Let us summarize the above derivations with the following theorem.

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

$$J = E\left\{ \phi(x[N], N) + \sum_{k=0}^{N-1} L(x[k], u[k], k) \right\} \tag{13}$$

subject to

$$x[k+1] = f(x[k], u[k], k) \tag{14}$$

with

$$x[0] = x_0 \sim P(x_0) \tag{15}$$

and

$$u[k] = g(x[k], r[k+1], k; W) \tag{16}$$

where N, $r[k]$, and the function f are specified, and x_0 is a random vector over state space with a probability $P(x_0)$, the optimal weight vector W of the neural network is obtained from the discrete *Neural Euler–Lagrange* (NEL) equations:

the system/control equations—

$$x[k+1] = f(x[k], u[k], k) \tag{17}$$

$$u[k] = g(x[k], r[k+1], k; W) \tag{18}$$

with $x[0] = x_0 \sim P(x_0),\ k = 0, 1, \ldots, N-1$

the adjoint system equations—

$$\begin{aligned}\delta_x[k] =& \frac{\partial f(k)}{\partial x[k]}^{\mathrm{T}} \delta_x[k+1] + \frac{\partial L(k)}{\partial x[k]}^{\mathrm{T}} \\ &+ \frac{\partial g(k;W)}{\partial x[k]}^{\mathrm{T}} \delta_u[k] \end{aligned} \tag{19}$$

$$\delta_u[k] = \frac{\partial f(k)}{\partial u[k]}^{\mathrm{T}} \delta_x[k+1] + \frac{\partial L(k)}{\partial u[k]}^{\mathrm{T}} \tag{20}$$

with $\delta_x[N] = (\partial \phi(N)/\partial x[N])^{\mathrm{T}},\ k = N-1, N-2, \ldots, 0$

the optimality condition—

$$E\left\{ \sum_{k=0}^{N-1} \frac{\partial g(k;W)}{\partial W}^{\mathrm{T}} \delta_u[k] \right\} = 0. \tag{21}$$

NEL consists of the system/control equations, the adjoint system equations, and the optimality condition. The system/control equations have an initial condition at the initial time $k = 0$. In contrast, the adjoint system equations have an initial condition at the end of the time horizon $k = N$. This is a two-point boundary value problem (TPBVP). The system/control equations, the adjoint system equations, and the optimality condition are all coupled to each other because all are involved with unknown parameters of W at the same time. Furthermore, since the optimality condition (21) is involved with the expectation operator associated with a probability distribution over different values of initial states, NEL cannot be solved as simultaneous nonlinear algebraic equations, in contrast with the case of EL. We must solve NEL numerically, exploiting the sequential nature of the problem.

C. Numerical Solution with Stochastic Steepest Descent Method

We may apply the stochastic steepest descent method, as does the LMS algorithm [10], [11]. The stochastic steepest descent algorithm for NDO consists of three steps:

1) forward sweep;
2) backward sweep;
3) update of the weight vector.

Step 1) Forward sweep

$$x[k+1] = f(x[k], u[k], k) \tag{22}$$

$$u[k] = g(x[k], r[k+1], k; W) \tag{23}$$

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

The forward sweep equations consist of the system and control equations, which are coupled to each other through the state $x[k]$ and the control input $u[k]$.

In order to initiate the forward sweep, suppose some desired reference inputs $\{r[k]\}$ are specified, depending on the goal of control. First of all, we initialize the weight vector of the neural network randomly. Then we pick initial states over state space with a probability $P(x_0)$, which is often recommended to have a uniform distribution. Given the chosen initial state x_0 and the specified reference inputs $r[k]$, we can compute the state $x[k]$ and the

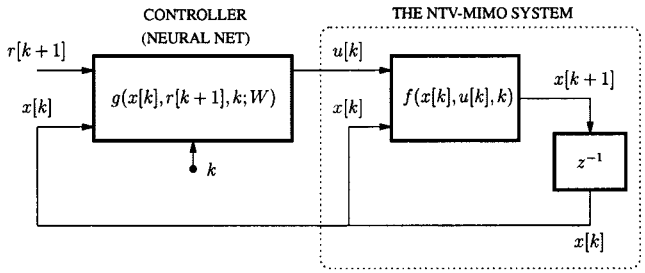

Fig. 2. Forward sweep of NDO.

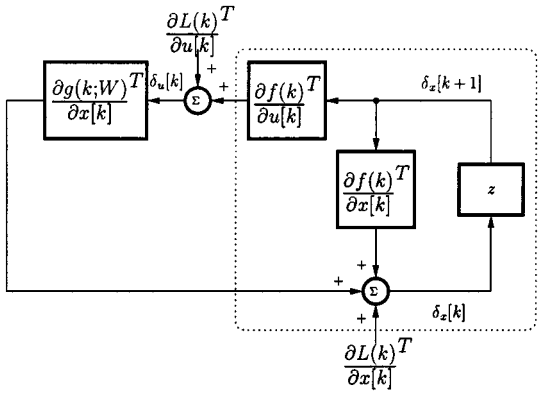

Fig. 3. Backward sweep of NDO.

control input $u[k]$ through $k = 0$ to N, using the system and control equations. Fig. 2 depicts the forward sweep.

Step 2) Backward sweep

$$\delta_x[k] = \frac{\partial f(k)}{\partial x[k]}^{\mathrm{T}} \delta_x[k+1] + \frac{\partial L(k)}{\partial x[k]}^{\mathrm{T}} + \frac{\partial g(k;W)}{\partial x[k]}^{\mathrm{T}} \delta_u[k] \tag{24}$$

$$\delta_u[k] = \frac{\partial f(k)}{\partial u[k]}^{\mathrm{T}} \delta_x[k+1] + \frac{\partial L(k)}{\partial u[k]}^{\mathrm{T}} \tag{25}$$

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

The backward sweep equations are the same as the adjoint system equations. Though these equations look complicated, they can be represented by the block diagram illustrated in Fig. 3. 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 $\delta_u[k]$. In this case, the co-output $\delta_u[k]$ is fed to the co-state $\delta_x[k]$ through $(\partial g(k;W)/\partial x[k])^{\mathrm{T}}$.

This is an anti-causal LTV system because the current co-state $\delta_x[k]$ depends on the future co-state $\delta_x[k+1]$ through (24) 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.

TABLE I
STOCHASTIC STEEPEST DESCENT ALGORITHM FOR NEURAL DYNAMIC OPTIMIZATION

- Enter a time horizon N, an update rate μ, and an initial guess of W.
- Repeat {
  1. Pick $x[0] = x_0 \sim P(x_0)$ over state space.
  2. Set the specified reference inputs $r[k]$, depending on the goal of control.
  3. Forward sweep.
  - For $k = 0, 1, \ldots, N-1$, compute

$$x[k+1] = f(x[k], u[k], k),$$

$$u[k] = g(x[k], r[k+1], k; W).$$

  - Store both $x[k]$ and $u[k]$, $k = 1, 2, \ldots, N$.
  4. Backward sweep.
  (a) Set $\delta_x[N] = \frac{\partial\phi(N)}{\partial x[N]}^{T}$.
  (b) For $k = N-1, N-2, \ldots, 1$, compute

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

$$\delta_x[k] = \frac{\partial f(k)}{\partial x[k]}^{T} \delta_x[k+1] + \frac{\partial L(k)}{\partial x[k]}^{T} + \frac{\partial g(k;W)}{\partial x[k]}^{T} \delta_u[k].$$

  5. Update W. For $k = N-1, N-2, \ldots, 0$, compute

$$\Delta W = -\mu \sum_{k=0}^{N-1} \frac{\partial g(k;W)}{\partial W}^{T} \delta_u[k],$$

$$W \longleftarrow W + \Delta W.$$

} until ΔW is small.

Step 3) Update of weight vector

$$\Delta W = -\mu \sum_{k=0}^{N-1} \frac{\partial g(k;W)}{\partial W}^{\mathrm{T}} \delta_u[k] \tag{26}$$

$$W \leftarrow W + \Delta W. \tag{27}$$

Then we update the weight vector W using the co-outputs $\delta_u[k]$, according to (26) and (27). The update of the weight vector can be viewed as a step to transform the co-outputs $\delta_u[k]$ into an incremental weight vector ΔW and then generate a new weight vector W. Obviously, $\delta_u[k] = 0$ for all k implies that $\Delta W = 0$. In other words, there is no change in the synaptic weight vector. We repeat the three steps until ΔW is small.

In short, the forward sweep defines feasible trajectories over which NDO searches for the optimal solution. The backward sweep conducts the optimization process over the feasible trajectories and generates the co-outputs $\delta_u[k]$. The update of the weight vector transforms the co-outputs $\delta_u[k]$ into the incremental synaptic weight vector ΔW. The stochastic steepest descent algorithm for NDO is summarized in Table I.

The computation requirement of the NDO algorithm for each iteration is composed of three components: forward sweep, backward sweep, update

of weight vector. The forward sweep requires approximately $(n(n+m)+n_w)\times N$ multiplications and additions because it evaluates the system/control equations from $k=0$ to $N-1$. The backward sweep needs roughly $(n\times n+m\times n+n_w)\times N$ operations because it computes not only Jacobians $(\partial f(k)/\partial x[k])^{\mathrm{T}}$ and $(\partial f(k)/\partial u[k])^{\mathrm{T}}$ but also calculates $(\partial g(k;W)/\partial x[k])^{\mathrm{T}}\delta_u[k]$ at each time step. The update of weight vector needs approximately $N\times m\times n_w$ operations. So the total number of multiplications and additions for each iteration using the NDO algorithm is approximately

$$(2n(n+m)+n_w(m+2))N \tag{28}$$

which means that the NDO algorithm requires the additional operations of $(n_w(m+2)-m)N$, compared to the FOC algorithm (see [8, eq. (36)]).

The memory requirement of the algorithm is approximately

$$(n+m)N+2\times n_w \tag{29}$$

since it stores not only the state $x[k]$ and the control input $u[k]$ from $k=0$ to $N-1$ but also the old and new weight vectors W's. Note that the memory requirement of NDO depends linearly on the dimension of the state n, the number of control inputs m, and the number of weights n_w. In contrast, the memory requirement of DP depends exponentially on the dimension of the state and the number of control inputs, which can be problematic for high-order nonlinear systems.

D. Learning Operator

In this section, we introduce the learning operator, which characterizes the learning process of the neural network in searching for the optimal feedback solution. The learning operator determines a mapping from information to knowledge and plays a central role in finding the optimal feedback solution in NDO. The analysis of the learning operator provides a fundamental understanding of the learning process in neural networks but also some useful guidelines for selecting the number of weights of the neural network.

Let us start with the update of the weight vector in the stochastic algorithm for NDO. The synaptic weight vector of the neural network is updated for each iteration:

$$\Delta W=-\mu\sum_{k=0}^{N-1}\frac{\partial g(k;W)}{\partial W}^{\mathrm{T}}\delta_u[k]. \tag{30}$$

Rewrite the above equation in the form of matrices as follows:

$$\begin{aligned}\Delta W&=-\mu G_w^{\mathrm{T}}\delta_u,\\&=-\mu\left[\begin{array}{ccc}\frac{\partial g(0;W)}{\partial W}^{\mathrm{T}} & \cdots & \frac{\partial g(N-1;W)}{\partial W}^{\mathrm{T}}\end{array}\right]\left[\begin{array}{c}\delta_u[0]\\ \vdots\\ \delta_u[N-1]\end{array}\right]\\&=-\mu\sum_{k=0}^{N-1}\frac{\partial g(k;W)}{\partial W}^{\mathrm{T}}\delta_u[k]\end{aligned} \tag{31}$$

where

$$G_w^{\mathrm{T}}\triangleq\left[\begin{array}{ccc}\frac{\partial g(0;W)}{\partial W}^{\mathrm{T}} & \cdots & \frac{\partial g(N-1;W)}{\partial W}^{\mathrm{T}}\end{array}\right]\in\mathcal{R}^{n_w\times n_m} \tag{32}$$

$$\delta_u\triangleq[\delta_u[0] \quad \cdots \quad \delta_u[N-1]]^{\mathrm{T}}\in\mathcal{R}^{n_m} \tag{33}$$

where n_w is the number of components of the synaptic weight vector, $n_m\triangleq N\times m$ is the time horizon, and m is the number of components of the control input vector to the system. The matrix G_w^{T}, the transpose of the Jacobian matrix G_w, determines a mapping from δ_u to ΔW. If we interpret δ_u as an information and ΔW as an incremental memory or knowledge stored in the neural network, then G_w^{T} determines the learning process of the neural network. We thus call G_w^{T} the learning operator, which makes sense because G_w^{T} depends on the current knowledge[1] W and stimuli such as $x[k]$, $r[k]$, and k of the system. The incremental knowledge ΔW is simply a linear combination of the columns of G_w^{T}. The columns of the learning operator G_w^{T} are the gradient vectors of the neural network $g(k;W)$ with respect to the synaptic weight vector W. Thus the incremental knowledge ΔW lies along the negative gradient of $g(k;W)$ with respect to W. We define additional terminology before we begin the analysis of the learning operator.

Notation: We call the vector δ_u the*information vector*. We call the vector space $\mathcal{R}^{n_m}$ where δ_u belongs the *information space*. We call the vector ΔW the *incremental knowledge vector*. We call the vector space $\mathcal{R}^{n_w}$ where ΔW belongs the *knowledge space*. We call the vector W the *knowledge vector*.

Let us discuss the operations of G_w^{T} in the learning process of the neural network. Suppose that the matrix G_w^{T} has rank r. The learning operator G_w^{T} has four fundamental subspaces. The row space of G_w^{T}, which is spanned by the rows of G_w^{T}, we call the *information-receiving shell*. The information-receiving shell is a subspace of the information space $\mathcal{R}^{n_m}$. The information-receiving shell has dimension r, which is the same as the rank of G_w^{T}.

The nullspace of G_w^{T}, which consists of all vectors δ_u such that $G_w^{\mathrm{T}}\delta_u=0$, we call the *information-blocking shell* because any information in the nullspace has no effect on increasing the knowledge (i.e., $\Delta W=0$). The information-blocking shell has dimension (n_m-r) and is the orthogonal complement of the information-receiving shell in the information space $\mathcal{R}^{n_m}$. In other words, the information-blocking shell contains everything orthogonal to the information-receiving shell in the information space $\mathcal{R}^{n_m}$. The column space[2] of G_w^{T}, which is spanned by the columns of G_w^{T}, we call the *information-memorizing shell*. All the incremental knowledge ΔW stored in the neural network is simply a linear combination of the columns of G_w^{T} as shown in (31). That is why we call the column space of G_w^{T} the information-memorizing shell. The columns of G_w^{T} are the gradient vectors of the neural network $g(k;W)$ with respect to the synaptic weight vector W. The incremental knowledge ΔW is in the direction of the negative gradient of $g(k;W)$ with respect to W, as ΔW is simply a linear combination of the columns of G_w^{T}. The information-memorizing shell is a subspace of the

[1]We interpret the synaptic weight vector W as the knowledge stored in the neural network.

[2]The column space of a matrix A is often called the range of the matrix A in the literature of linear algebra.

memory space $\mathcal{R}^{n_w}$. The information-memorizing shell has the same dimension, r, as the information-receiving shell. In other words, the dimension (i.e., size) of the received information is the same as that of the stored information since the learning operator G_w^{T} is a matrix.

The left nullspace of G_w^{T}, which consists of all vectors ΔW such that $\Delta W^{\mathrm{T}} G_w^{\mathrm{T}} = 0$, we call the *uninfluenced-information shell*. The left nullspace cannot be reached by influence of the information δ_u because the left nullspace is orthogonal to the column space of G_w^{T} that carries the information δ_u. That is why we call the left nullspace of G_w^{T} the uninfluenced-information shell. The uninfluenced-information shell has the dimension of $(n_w - r)$, and is the orthogonal complement of the information-memorizing shell in the memory space $\mathcal{R}^{n_w}$. In other words, the uninfluenced-information shell contains everything orthogonal to the information-memorizing shell in the memory space $\mathcal{R}^{n_w}$.

The learning operator has four important properties. First of all, the learning operator is a matrix that determines a mapping from information vector δ_u to the incremental knowledge ΔW stored in the neural network. Second, the learning operator changes at each iteration, depending on the current knowledge W and the stimuli such as $x[k]$, $r[k]$, and k of the system. Thus its four fundamental subspaces change at each iteration. The information-receiving shell moves around in the information space, while the information-memorizing shell moves around in the knowledge space at each iteration, depending on the current knowledge W and the stimuli. The learning process in the neural network is dynamic in that sense. Third, the learning operator G_w^{T}, which determines a mapping from δ_u to ΔW, is a rectangular matrix. It may not be an invertible mapping from δ_u to ΔW. Nonetheless, the learning operator G_w^{T} is always an invertible mapping from its information-receiving shell to information-memorizing shell because every incremental knowledge vector ΔW in the information-memorizing shell comes from one and only one information vector δ_u in the information-receiving shell. In addition, there is no expansion or contraction of the dimension (size) of information processed (i.e., received and memorized) by the learning operator because the information-receiving and the information-memorizing shells of the learning operator always have the same dimension. Fourth, the rank of the learning operator determines the dimension (i.e., size) of information the learning operator can handle (or process) for each iteration of the training process because the information-receiving and the information-memorizing shells of the learning operator have the same dimension as the rank of the learning operator. The rank of the learning operator plays a key role in the processing of the learning operator. The rank of the learning operator G_w^{T} cannot exceed the number of its rows and columns, as shown in Theorem 2 .

Theorem 2: The learning operator G_w^{T} has a rank r bounded by the numbers of its rows and columns as follows:

$$r \leq n_w \text{ and } r \leq n_m \tag{34}$$

where n_w and n_m are the numbers of its rows and columns, respectively.

Proof: The proof results from the well-known properties of matrices [12], [13].

Let us analyze the operation of G_w^{T} in terms of its rank. We begin with the definition of full column rank of a matrix.

Definition 1: A matrix G has full column rank if its rank is the same as the number of its columns.

Conversely, we say that the matrix G has nonfull column rank if its rank is not the same as the number of its columns. We consider two cases: 1) nonfull column rank and 2) full column rank.

1) Nonfull Column Rank:

   Suppose that n_w, i.e., the number of rows of the learning operator, is smaller than n_m, i.e., the number of columns. In other words, the dimension (size) of the knowledge space is smaller than that of the information space. Then the learning operator is forced to have nonfull column rank for all training processes (or all training iterations) because $r \leq n_w < n_m$ according to Theorem 2. The information-receiving shell of the learning operator, which has dimension r, cannot cover the entire region but only a part of the information space. The information-blocking shell, which consists of all vectors δ_u such that $G_w^{\mathrm{T}} \delta_u = 0$, covers the rest of the information space and has $(n_m - r)$ dimension. Any information belonging to (or captured by) the information-blocking shell will not be stored in the synaptic weights of the neural network because $\Delta W = G_w^{\mathrm{T}} \delta_u = 0$. Thus the information-blocking shell may cause a loss of information during the learning process of G_w^{T} even though the net incremental-knowledge ΔW, which both the information-receiving and the information-memorizing shells produce, still lies along the negative gradient of $g(k, W)$ with respect to W.

2) Full Column Rank:

   Consider the alternative case, where the learning operator has full column rank. The rank r is equal to the number of columns n_m, which is equal to the dimension of the information space. Then the information-receiving shell of the learning operator covers the entire region of the information space because its dimension is the same as the rank of the learning operator, which turns out to be equal to the dimension of the information space n_m because the learning operator has full column rank. Then the information-blocking shell, which consists of all vectors δ_u such that $G_w^{\mathrm{T}} \delta_u = 0$, contains only the zero vector. In other words, any information vector except zero in the information space will contribute to increasing the knowledge (i.e., $\Delta W \neq 0$). Thus there is no loss of information in the learning processing of G_w^{T}, which is desirable. In addition, the learning operator, which has full column rank, has the following important structural property.

Theorem 3: If the learning operator has full column rank, then the number of rows n_w, i.e., the number of weights of the neural network, is greater than or equal to the number of columns n_m, i.e., the dimension of the information space.

Proof: Assume that the learning operator has full column rank. The rank r is equal to the number of columns n_m, which is equal to the dimension of the information space. Then according to Theorem 2 , the number of rows n_w must be greater than or equal to n_m.

However, the converse is not true. In other words, the condition that the number of the weights of the neural network n_w exceeds the dimension of the information space n_m does not guarantee that the learning operator maintains full column rank, because the rank r of the learning operator may depend on the current knowledge W, the topology of the neural network, and stimuli such as $x[k]$, $r[k]$, and k for each iteration. Nonetheless, the learning operator is likely to have full rank in practice as long as the number of weights of the neural network n_w exceeds the dimension of the information space n_m. Increasing the number of weights of the neural network n_w may help to improve the rank of the learning operator so that the operator can be more likely to keep full column rank. But if we too much increase the number of weights of the neural network n_w, then the knowledge space becomes extremely large and may exact an unnecessary storage and computation cost. Furthermore, the uninfluenced-information shell becomes extremely large because it has the dimension of $(n_w - r)$, which is effectively $(n_w - n_m)$. The learning process in the neural network becomes extremely localized because the dimension of the uninfluenced-information shell, i.e., $(n_w - r) \approx (n_w - n_m)$, becomes exceedingly larger than that of the information-memorizing shell, i.e., $r \approx n_m$. The weights of the neural network are highly likely to be stuck in an undesirable place and end up with a local minimum.

The above discussion leads to a guideline for selecting the number of weights of the neural network as follows.

Rule 1: Choose a number of weights of the network greater than the dimension of the information space n_m in order to avoid a loss of information caused by the information-blocking shell of the learning operator. However, picking too many weights for the network may result in an undesirable local minimum.

NDO can handle finite horizon problems such as terminal control since it runs finite time steps ahead. In addition, NDO can handle infinite horizon problems such as regulation and tracking by using a big enough time horizon N; the solution to a finite horizon problem converges to the solution to the corresponding infinite horizon problem if N is made sufficiently large.

For infinite horizon problems, however, Rule 1 apparently implies that NDO may require extensive memory because it suggests that we increase the number of weights of the network as N increases.

However, this is not necessarily true because Rule 1 is not valid for such problems. For instance, consider an infinite horizon problem such as regulation. Suppose that we have a time-invariant system and select a cost function that has no explicit time dependency. Then the optimal feedback solution converges to a time-invariant solution as N goes to infinity [14], [15]:

$$x[k] = f(x[k], u(k)) \tag{35}$$

$$u(k) = g^o(x[k]). \tag{36}$$

If the closed-loop system is stable, then it reaches its equilibrium point as $N \to \infty$. Once the system arrives in steady state, the system and control equations, (35) and (36) no longer provide new information about its dynamical behavior because the equilibrium is static. So when NDO trains neural networks for such problems, the learning operator does not need to have full-column rank because part of the information space R^{n_m} may contain no information. This observation leads to an additional guideline as follows.

Rule 2: For infinite-horizon problems (e.g., regulation and tracking control) where systems are time-invariant and cost functions have no explicit time dependency, do the following:

1) Exclude explicit time dependency from neural networks.
2) Do not increase as the number of weights of the neural network as time horizon N increases.

E. Information Time Shift Operator

Backward sweep effectively produces the information vector δ_u through minimization of the cost function. The information contained in the information vector is stored in the synaptic weights of the neural network by the learning operator G_w^{T}, as discussed in Section II-D. During the backward sweep the information sequence $\delta_u[k]$, which is also called the co-output vector, is generated backward in time because the current co-state $\delta_x[k]$ depends on the future co-state $\delta_x[k+1]$, and the initial condition is specified at the the end of the time horizon

$$\delta_x[k] = \frac{\partial f(k)}{\partial x[k]}^{\mathrm{T}} \delta_x[k+1] + \frac{\partial L(k)}{\partial x[k]}^{\mathrm{T}} + \frac{\partial g(k; W)}{\partial x[k]}^{\mathrm{T}} \delta_u[k] \tag{37}$$

$$\delta_u[k] = \frac{\partial f(k)}{\partial u[k]}^{\mathrm{T}} \delta_x[k+1] + \frac{\partial L(k)}{\partial u[k]}^{\mathrm{T}} \tag{38}$$

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

The current information sequence $\delta_u[k]$ is fed to the current co-state $\delta_x[k]$ through the matrix $(\partial g(k, W)/\partial x[k])^{\mathrm{T}}$, as shown in (37). The past information sequence $\delta_u[k-1]$ is computed from the current co-state $\delta_x[k]$ by the application of (38). So the current information sequence $\delta_u[k]$ makes contributions to creating the past information $\delta_u[k-1]$ through $(\partial g(k, W)/\partial x[k])^{\mathrm{T}}$. On the contrary, if $(\partial g(k, W)/\partial x[k])^{\mathrm{T}} = 0$, then the current information sequence $\delta_u[k]$ has no way to contribute to generating the past information sequence $\delta_u[k-1]$. Thus we call the matrix $(\partial g(k, W)/\partial x[k])^{\mathrm{T}}$ the information time shift operator, which determines a mapping from $\delta_u[k]$ to $\delta_x[k]$. The information time shift operator is a $n \times m$ matrix, where n is the number of components of the system state vector $x[k]$ and m is the number of components of the control input vector $u[k]$. This operator plays a central role in transferring the information contained in the information sequence $\delta_u[k]$ to the co-state $\delta_x[k]$, as illustrated as in Fig. 4.

F. Linear Quadratic Problems

The application of NDO to linear quadratic problems is important because it allows us to find easily a connection between NDO and DP as well as the duality of the forward and backward sweep of NDO. Specifically, we use a linear — rather than nonlinear — neural network as a general function approximator

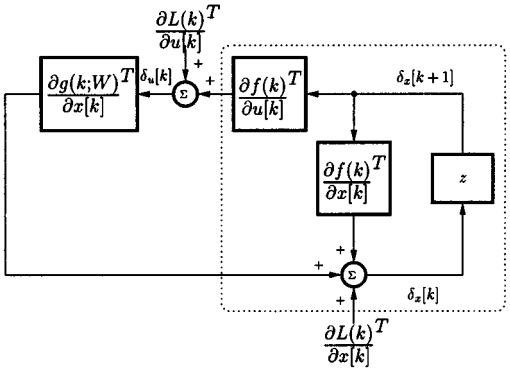

Fig. 4. Backward sweep of NDO.

since DP already proves that linear state feedback is the best form of feedback for those problems.

Let us consider a *controllable* LTI-MIMO system described by a state equation of the form

$$x[k+1] = Ax[k] + Bu[k] \tag{39}$$

with

$$x[0] = x_0 \sim P(x_0) \tag{40}$$

where

$x[k] \in \mathcal{R}^n$ state vector;
$u[k] \in \mathcal{R}^m$ control input vector;
x_0 initial state treated as a random vector over state space with a probability $P(x_0)$.

Here, we want to find the optimal weight matrix W of the linear neural network $u[k] = \mathbf{W}x[k]$, minimizing a constant-coefficient quadratic cost function of the form

$$J = E\left\{\frac{1}{2}x^{\mathrm{T}}[N]S[N]x[N] + \frac{1}{2}\sum_{k=0}^{N-1}(x^{\mathrm{T}}[k]Qx[k] + u^{\mathrm{T}}[k]Ru[k])\right\} \tag{41}$$

with

$$S[N] \geq 0,\ Q \geq 0,\ \text{and}\ R > 0 \tag{42}$$

where the weight matrix W is a $m \times n$ matrix. In the linear quadratic problem, we particularly exclude explicit time-dependency as well as reference inputs from the neural network. Thus the number of weights of the linear neural network[3] is automatically determined by the numbers of the states and the control inputs, in contrast with the general case of NDO. In addition, note that we should select the matrices Q and R so that they guarantee there exists a unique stabilizing optimal feedback solution according to Theorem 4 of [8, th. 4]. By applying NEL and the stochastic descent method to the linear quadratic problem, we obtain the following three steps:

[3]The linear neural network is completely expressed by a single matrix because the multiplication of any finite number of matrices produces a matrix.

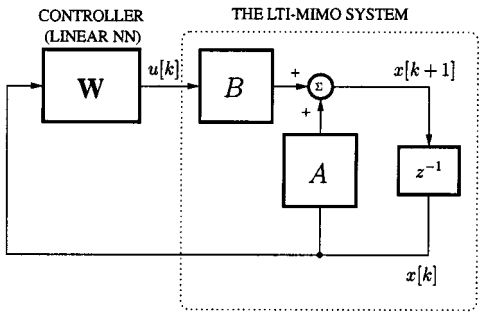

Fig. 5. Forward sweep of NDO for linear quadratic problems.

Step 1) Forward sweep:

$$x[k+1] = Ax[k] + Bu[k] \tag{43}$$
$$u[k] = \mathbf{W}x[k] \tag{44}$$

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

The forward sweep equations consist of the system and the control equations. Fig. 5 depicts the forward sweep of NDO for linear quadratic problems.

Step 2) Backward sweep:

$$\delta_x[k] = A^{\mathrm{T}}\delta_x[k+1] + Qx[k] + \mathbf{W}^{\mathrm{T}}\delta_u[k] \tag{45}$$
$$\delta_u[k] = B^{\mathrm{T}}\delta_x[k+1] + Ru[k] \tag{46}$$

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

The backward sweep equations are the same as the adjoint system equations, which can be represented by the block diagram illustrated in Fig. 6. We have the co-state $\delta_x[k]$, two inputs $Qx[k]$ and $Ru[k]$ derived from the cost function, and the output $\delta_u[k]$. In this case, the output $\delta_u[k]$ is fed to the co-state $\delta_x[k]$ through the matrix $\mathbf{W}^{\mathrm{T}}$ since $(\partial g(k;W)/\partial x[k])^{\mathrm{T}}$ simply reduces to $\mathbf{W}^{\mathrm{T}}$ for linear neural networks. This is an anti-causal LTI system because the coefficient matrices are constant, and the current co-state $\delta_x[k]$ depends on the future co-state $\delta_x[k+1]$ through (45), while the forward sweep is the causal LTI system. A comparison of Figs. 5 and 6 clearly reveals the *duality* of the backward and forward sweeps.

Step 3) Update of weight matrix:

$$\Delta\mathbf{W}^{\mathrm{T}} = -\mu\sum_{k=0}^{N-1} x[k]\delta_u^{\mathrm{T}}[k] \tag{47}$$
$$\mathbf{W}^{\mathrm{T}} \leftarrow \mathbf{W}^{\mathrm{T}} + \Delta\mathbf{W}^{\mathrm{T}}. \tag{48}$$

Then, we update the weight matrix $\mathbf{W}$ using (47) and (48). The update (47) has a much simpler form, compared with (26) in the general case of NDO. It says that the incremental knowledge $\Delta\mathbf{W}$ is simply a summation of the outer product of the state $x[k]$ and the co-output $\delta_u[k]$. The incremental knowledge

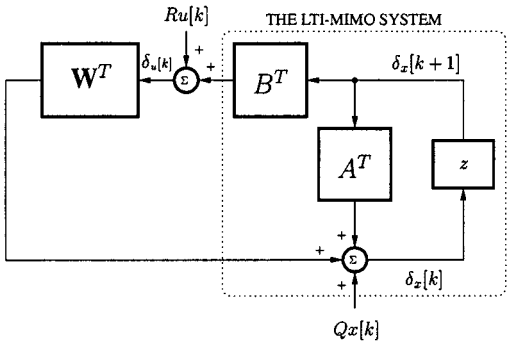

Fig. 6. Backward sweep of NDO for linear quadratic problems.

$\Delta\mathbf{W}$ is no longer dependent on the current knowledge $\mathbf{W}$ of the neural network, in contrast with the general case of NDO. We repeat the three steps until $\Delta\mathbf{W}$ is small.

NDO should enable the linear neural network to approximate the unique stabilizing optimal linear feedback solution to the linear quadratic problem to a desirable accuracy. Specifically, if we pick a big enough time horizon N, the values of the optimal weight matrix W that NDO produces should be very close to those of the constant Kalman gain K that the algebraic Riccati equation (ARE) produces for the linear quadratic regulator. The next section will prove that this is true. This fact demonstrates a connection between NDO and DP because NDO can approximate the optimal feedback solution whose existence DP guarantees.

III. Equivalence of NDO, DP, and FOC

This section shows the equivalence[4] of NDO, DP, and FOC. In particular, we consider the use of these methods to solve the linear quadratic problem. The mathematics of this problem is simple. See [16] for a description of a more general class of nonlinear problems.

A. Equivalence of NDO and FOC

NDO and FOC are different approaches and have different forms of optimality conditions. Nonetheless, NDO and FOC have a similar computation structure: forward sweep, backward sweep, and update, as follows:

- FOC

  Forward sweep—

$$x[k+1] = Ax[k] + Bu[k], \tag{49}$$

  Backward sweep —

$$\delta_x[k] = A^{\mathrm{T}}\delta_x[k+1] + Qx[k] \tag{50}$$
$$\delta_u[k] = B^{\mathrm{T}}\delta_x[k+1] + Ru[k] \tag{51}$$

[4]We call two methods equivalent if they produce the same solution.

  Update of control vector sequence —

$$\Delta u[k] = -\mu\delta_u[k] \tag{52}$$
$$u[k] \leftarrow u[k] + \Delta u[k] \tag{53}$$

- NDO

  Forward sweep —

$$x[k+1] = Ax[k] + Bu[k] \tag{54}$$
$$u[k] = \mathbf{W}x[k] \tag{55}$$

  Backward sweep —

$$\delta_x[k] = A^{\mathrm{T}}\delta_x[k+1] + Qx[k] + \mathbf{W}^{\mathrm{T}}\delta_u[k] \tag{56}$$
$$\delta_u[k] = B^{\mathrm{T}}\delta_x[k+1] + Ru[k] \tag{57}$$

  Update of weight matrix —

$$\Delta\mathbf{W}^{\mathrm{T}} = -\mu\sum_{k=0}^{N-1} x[k]\delta_u^{\mathrm{T}}[k] \tag{58}$$
$$\mathbf{W}^{\mathrm{T}} \leftarrow \mathbf{W}^{\mathrm{T}} + \Delta\mathbf{W}^{\mathrm{T}}. \tag{59}$$

In addition, according to Theorem 4 of [8, th. 4], there exists a unique optimal solution if and only if (A, B) is controllable, $(A, \sqrt{Q})$ is observable, $R > 0$, and time step N is sufficiently large. Before we show the equivalence of NDO and FOC explicitly, we point out that the backward sweep equations in NDO and FOC are identical except for the term $\mathbf{W}^{\mathrm{T}}\delta_u[k]$. So, if $\delta_u[k] = 0$, then the backward sweep equations are identical, regardless of the values of weight matrix W. In other words, the mechanics of backward sweep in both FOC and NDO become identical.

Theorem 4: For the linear quadratic problems of minimizing

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

subject to

$$x[k+1] = Ax[k] + Bu[k]$$

where (A, B) is controllable $(A, \sqrt{Q})$ is observable $R > 0$; supposing that the time horizon N is sufficiently large, then NDO and FOC are equivalent.

Proof: Suppose that FOC converges to the unique optimal solution. Then $\Delta u[k] = 0$, and $\delta_u[k]$ should be approximately zero, according to (52). So the mechanics of backward sweep in FOC and NDO become identical. In addition, $\delta_u[k] = 0$ leads to $\Delta\mathbf{W}^{\mathrm{T}} = 0$ in NDO. Thus NDO converges to the optimal solution. Conversely, suppose that NDO converges to the optimal solution. Then $\Delta\mathbf{W}^{\mathrm{T}} = 0$. Equation (58) can be expressed as follows:

$$\Delta\mathbf{W}^{\mathrm{T}} = -\mu\sum_{k=0}^{N-1} x[k]\delta_u^{\mathrm{T}}[k] = -\mu\left[\sum_{k=0}^{N-1} x_i[k]\delta_{u_j}[k]\right]$$

where $x_i[k]$ is the ith component of state vector $x[k]$, and $\delta_{u_j}[k]$ is the jth component of co-output vector $\delta_u[k]$. The (i, j)th element of the matrix $\Delta\mathbf{W}^{\mathrm{T}}$ is $\sum_{k=0}^{N-1} x_i[k]\delta_{u_j}[k]$, which is the

inner-product of $x_i[k]$ and $\delta_{u_j}[k]$ trajectories. We are searching for the optimal solution over state space, so $x_i[k]$ can be any arbitrary nonzero trajectory i.e., $\exists i$ and k such that $x_i[k] \neq 0$. So $\Delta \mathbf{W}^{\mathrm{T}} = 0$ leads to $\delta_{u_j}[k] = 0$ for $k = 0, \ldots, N-1$ since only zero vector is orthogonal to any arbitrary nonzero vector. So $\delta_u[k] = 0$ for all k. Thus, $\Delta u[k]$ should be zero in FOC, according to (52). FOC thus converges to the optimal solution. □

B. Equivalence of FOC and DP

This section reviews the equivalence of FOC and DP, which is known in [14], [15], and [17].

Let us begin with the application of FOC to linear quadratic problems. Then, the Euler–Lagrange (EL) equations reduce to a simpler form

the system equation—

$$x[k+1] = Ax[k] + Bu[k],\ k = 0, \ldots, N-1 \tag{60}$$

with $x[0] = x_0$

the adjoint system equation—

$$\delta_x[k] = A^{\mathrm{T}}\delta_x[k+1] + Qx[k],\ k = N-1, \ldots, 0 \tag{61}$$

with $\delta_x[N] = S[N]x[N]$

the optimality condition—

$$0 = B^{\mathrm{T}}\delta_x[k+1] + Ru[k],\ k = N-1, \ldots, 0. \tag{62}$$

From the optimality condition (62), the control input $u[k]$ is given by

$$u[k] = -R^{-1}B^{\mathrm{T}}\delta_x[k+1]. \tag{63}$$

The system and adjoint system equations with the control input $u[k]$ eliminated are

$$x[k+1] = Ax[k] - BR^{-1}B^{\mathrm{T}}\delta_x[k+1] \tag{64}$$

$$\delta_x[k] = A^{\mathrm{T}}\delta_x[k+1] + Qx[k]. \tag{65}$$

Now, assume that the co-state $\delta_x[k]$ has a linear relationship with the state $x[k]$

$$\delta_x[k] = S[k]x[k],\ k = N-1, \ldots, 0 \tag{66}$$

for some sequence of the $n \times n$ matrix $S[k]$, because $\delta_x[N] = S[N]x[N]$ holds at the end of the time horizon $k = N$, as shown in (61). If we can find a consistent formula for the postulated $S[k]$, then (66) is a valid assumption.

Let us find a formula for $S[k]$. We substitute (66) into (64) to get

$$x[k+1] = Ax[k] - BR^{-1}B^{\mathrm{T}}S[k+1]x[k+1]. \tag{67}$$

Solving for $x[k+1]$ yields

$$x[k+1] = (I + BR^{-1}B^{\mathrm{T}}S[k+1])^{-1}Ax[k] \tag{68}$$

which is a forward recursion for the state. Now substitute (66) into the adjoint system (65) to obtain

$$S[k]x[k] = A^{\mathrm{T}}S[k+1]x[k+1] + Qx[k] \tag{69}$$

and substitute (68) into (69) to get

$$\begin{aligned} S[k]x[k] =& A^{\mathrm{T}}S[k+1](I + BR^{-1}B^{\mathrm{T}}S[k+1])^{-1}Ax[k] \\ &+ Qx[k]. \end{aligned} \tag{70}$$

Since $x[k]$ is nonzero, and this equation holds for all state sequence given any $x[k]$, evidently

$$S[k] = A^{\mathrm{T}}S[k+1](I + BR^{-1}B^{\mathrm{T}}S[k+1])^{-1}A + Q \tag{71}$$

or using the matrix inversion lemma [15], [17]

$$\begin{aligned} S[k] =& A^{\mathrm{T}}(S[k+1] \\ &- S[k+1]B(B^{\mathrm{T}}S[k+1]B + R)^{-1}B^{\mathrm{T}}S[k+1])A \\ &+ Q. \end{aligned} \tag{72}$$

This is the Riccati equation, which is a backward recursion for the postulated $S[k]$, completely specifying $S[k]$ in terms of $S[k+1]$ and the known system and weighting matrices.

In order to determine the optimal control, use (66) and (63) to get

$$u[k] = -R^{-1}B^{\mathrm{T}}\delta_x[k+1] \tag{73}$$

$$= -R^{-1}B^{\mathrm{T}}S[k+1]x[k+1] \tag{74}$$

$$= -R^{-1}B^{\mathrm{T}}S[k+1](Ax[k] + Bu[k]) \tag{75}$$

or

$$(I + R^{-1}B^{\mathrm{T}}S[k+1]B)u[k] = -R^{-1}B^{\mathrm{T}}S[k+1]Ax[k]. \tag{76}$$

Let us premultiply by R and then solve for the control $u[k]$ to obtain

$$u[k] = -K[k]x[k] \tag{77}$$

where $K[k] = (B^{\mathrm{T}}S[k+1]B + R)^{-1}B^{\mathrm{T}}S[k+1]A$, which is the Kalman gain sequence. For linear quadratic problems, thus, FOC yields the same form of the optimal solution that DP produces. Thus FOC is equivalent to DP. Let us summarize the above result with the following theorem.

Theorem 5: DP and FOC are one and the same thing for linear quadratic problems.

C. Equivalence of NDO and DP

DP shows that the optimal feedback control for the linear quadratic problem is obtained from the discrete Riccati equation, as shown in [8, th. 2]. As a result, DP proves that linear state feedback is the best form of feedback for linear quadratic problems. In addition, according to [8, th. 4], the time-varying Kalman gain converges to a unique constant stabilizing gain as time horizon N increases if and only if (A, B) is controllable, $(A, \sqrt{Q})$ is observable, and $R > 0$.

In contrast, NDO for the linear quadratic problem leads to the following set of equations: forward sweep, backward sweep, and update of the weight matrix:

- Forward sweep:

$$x[k+1] = Ax[k] + Bu[k] \tag{78}$$

$$u[k] = \mathbf{W}x[k] \tag{79}$$

- Backward sweep:

$$\delta_x[k] = A^{\mathrm{T}}\delta_x[k+1] + Qx[k] + \mathbf{W}^{\mathrm{T}}\delta_u[k] \tag{80}$$

$$\delta_u[k] = B^{\mathrm{T}}\delta_x[k+1] + Ru[k] \tag{81}$$

- Update of weight matrix:

$$\Delta\mathbf{W}^{\mathrm{T}} = -\mu\sum_{k=0}^{N-1} x[k]\delta_u^{\mathrm{T}}[k] \tag{82}$$

$$\mathbf{W}^{\mathrm{T}} \leftarrow \mathbf{W}^{\mathrm{T}} + \Delta\mathbf{W}^{\mathrm{T}}. \tag{83}$$

Recall that the linear—rather than nonlinear—neural network is employed as a general function approximator since DP proves that linear state feedback control is best for those problems.

The following theorem states that NDO produces the same optimal feedback solution that DP finds for linear quadratic problems under certain conditions.

Theorem 6: For the linear quadratic problems of minimizing

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

subject to

$$x[k+1] = Ax[k] + Bu[k]$$

where (A, B) is controllable, $(A, \sqrt{Q})$ is observable $R > 0$; supposing that the time horizon N is sufficiently large, then NDO and DP are equivalent.

Proof: The proof results from Theorems 4 and 5. □

Now, let us suppose that NDO solves a finite horizon problem using a large enough time horizon although the corresponding problem is itself an infinite horizon problem like a linear quadratic regulation. Then the following theorem states that as we increase the time horizon, the weight matrix of the linear neural network gets closer and closer to the constant Kalman gain, which is obtained from the following ARE:

$$\begin{aligned} u[k] &= -Kx[k] \\ K &= (B^{\mathrm{T}}SB + R)^{-1}B^{\mathrm{T}}SA \\ J^o &= \frac{1}{2}x^{\mathrm{T}}[0]Sx[0] \\ 0 &= S - A^{\mathrm{T}}SA + A^{\mathrm{T}}SB(R + B^{\mathrm{T}}SB)^{-1}BS^{\mathrm{T}}A - Q. \end{aligned}$$

Theorem 7: For the linear quadratic regulator minimizing

$$J = \lim_{N\to\infty}\sum_{k=0}^{N}\frac{1}{2}(x^{\mathrm{T}}[k]Qx[k] + u^{\mathrm{T}}[k]Ru[k])$$

subject to

$$x[k+1] = Ax[k] + Bu[k]$$

where (A, B) is controllable, $(A, \sqrt{Q})$ is observable, and $R > 0$; supposing that the time horizon N is sufficiently large, then NDO enables the linear neural network to approximate the unique optimal constant gain feedback solution of the linear quadratic regulation problem to any desirable accuracy.

Proof: The proof results from [8, ths. 4 and 6].

In a sense, Theorem 7 is consistent with the following observation: if a time horizon N is sufficiently large, the solution to a finite-horizon problem can approximate the solution to the corresponding infinite horizon problems, just as a finite impulse response (FIR) filter can approximate the corresponding infinite impulse response (IIR) filter if its tap-delay line is large enough.

IV. Conclusion

We present neural dynamic optimization (NDO) as an approximate technique for solving the DP problem based on neural network techniques. We formulate neural networks to approximate the optimal feedback solutions whose existences DP justifies. We introduce the learning operator and the information time shift operator, which characterize the learning process of the neural network in searching for the optimal solution. The analysis of the learning operator provides not only a fundamental understanding of the learning process in neural networks but also some useful guidelines for selecting the number of weights of the neural network. We prove the equivalence of NDO, DP, and FOC for linear quadratic problems, which demonstrates a connection between NDO and optimal control theory.

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. Combining the positive features of both methodologies, NDO inherits its practicality from neural networks and its generality from optimal control theory. NDO, however, has two potential drawbacks. First, the NDO solution is not a complete DP solution: it approximates the optimal solution. Local as well as global optima are possible. Its domain of attraction can be limited. Second, the stability of the weight update cannot be guaranteed because its analytical condition has not been developed. In practice, however, these two drawbacks can be overcome by retraining the neural network with different values of its update (i.e., learning) rate or initial weights, as illustrated in the subsequent paper [9] on this topic.

References

[1] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 3rd ed. Reading, MA: Addison-Wesley, 1994.

[2] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*, 3rd 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] C. Seong and B. Widrow, "Neural dynamic optimization for control systems — Part I, Background," *IEEE Trans. Syst., Man, Cybern.*, vol. 31, pp. 482–489, Aug. 2001.

[9] ——, "Neural dynamic optimization for control systems — Part III, Applications," *IEEE Trans. Syst., Man, Cybern.*, vol. 31, pp. 502–513, Aug. 2001.

[10] B. Widrow and M. Hoff, "Adaptive switching circuits," in *IRE Western Electric Show Conv. Rec.*, Aug 4, 1960, pp. 96–104.

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

[12] G. Strang, *Linear Algebra And Its Applications*, 3rd ed. Orlando, FL: Harcourt Brace Jovanovich, 1988.
[13] B. Noble and J. W. Daniel, *Applied Linear Algebra*. Englewood Cliffs, NJ: Prentice-Hall, 1988.
[14] A. E. Bryson, *Dynamic Optimization*. Menlo Park, CA: Addison-Wesley-Longman, 1999.
[15] F. L. Lewis, *Optimal Control*, 2nd ed. New York: Wiley, 1995.
[16] C. Seong, "Neural dynamic programming and its application to control systems," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1999.
[17] R. F. Stengel, *Optimal Control and Estimation*. New York: Dover, 1994.

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.