NASA Contractor Report 177599

Two Neural Network Algorithms for Designing Optimal Terminal Controllers with Open Final-Time

Edward S. Plumer

CONTRACT NGT-50642
October 1992

National Aeronautics and
Space Administration

NASA Contractor Report 177599

Two Neural Network Algorithms for Designing Optimal Terminal Controllers with Open Final-Time

Edward S. Plumer

Department of Electrical Engineering
Stanford University
Durand Building, Room 104
Stanford, CA 94035-4055

Prepared for
Ames Research Center
CONTRACT NGT-50642
October 1992

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

1 Abstract

Multilayer neural networks, trained by the backpropagation through time algorithm (BPTT), have been used successfully as state-feedback controllers for nonlinear terminal control problems. Current BPTT techniques, however, are not able to deal systematically with open final-time situations such as minimum-time problems. Two approaches which extend BPTT to open final-time problems are presented. In the first, a neural network learns a mapping from initial-state to time-to-go. In the second, the optimal number of steps for each trial run is found using a line-search. Both methods are derived using Lagrange multiplier techniques. This theoretical framework is used to demonstrate that the derived algorithms are direct extensions of forward/backward sweep methods used in N-stage optimal control. The two algorithms are tested on a Zermelo problem and the resulting trajectories compare favorably to optimal control results.

2 Introduction

The use of neural networks as controllers for dynamic systems is currently the subject of much interest. Controllers for linear systems can be designed using well-established techniques, but no general design approaches yet exist for the larger class of nonlinear systems. A neural network can be a useful alternative tool for the synthesis of nonlinear controllers. The feasibility of training neural controllers has, moreover, been demonstrated by numerous applications.

As distinguished by Bryson and Ho [1], there are two main classes of controllers: *regulators* and *terminal controllers*. A regulator maintains the state of the system about some known reference. It accomplishes this despite external disturbances and internal uncertainties. Narendra and Parthasarathy, among others, have analyzed neural network regulator structures [2,3], training their networks using Williams and Zipser's dynamic backpropagation algorithm [4]. Unlike a regulator, a terminal controller drives the plant to some final state while maintaining acceptable state and control values along the trajectory. Whereas a regulator continues its task indefinitely, a terminal controller stops when the desired final state is reached. One example of a terminal controller is the truck-backer of Nguyen and Widrow [5] in which a network was trained, using a variant of Werbos's backpropagation through time (BPTT) algorithm [6], to implement a state-feedback control law.

An important class of terminal controllers is that in which the controller does not know, and must optimize, the number of steps along the trajectory. However, current BPTT techniques for terminal controller design do not provide a systematic way of incorporating the time elapsed along a trajectory as part of the cost function. As a result, it is not practical to solve problems such as minimum-time control. On the other hand, such "open final-time" problems have been dealt with for several decades using classical optimal control methods [1]. However, as I point out in section 3, these optimal control techniques typically find a set of open-loop control vectors which are valid only along a single trajectory. This is a limitation, as often control over many trajectories is needed. Such control can be accomplished, if full state information is available, by capturing the optimal state-feedback control mapping over the desired range of state-space.

Dynamic programming [7,8] provides one way of computing this optimal feedback control; for problems with many dimensions, however, the computation and storage requirements of dynamic programming are prohibitive. A second, less problematic, technique involves the precomputation of a number of nominal optimal paths and the subsequent use of second variation methods to find optimal solutions near one of the precomputed trajectories [1]. However, this requires both deciding upon a good representative set of nominal trajectories as well as explicitly storing the set of computed control vectors.

Sigmoidal feedforward neural networks with 2 layers of neural elements are capable of approximating any sufficiently well-behaved function, given that they contain a sufficient number of nodes in the hidden layer [9,10]. In consequence, they can be used to approximate the control law without explicitly storing control vectors over the state-space. A way of combining the state-feedback structure of a neural network controller with the open final-time methods of optimal control would

permit the application of neural networks to the class of terminal control problems described above. This paper derives two such extensions to BPTT using well-established Lagrange multiplier methods. A similar theoretical framework for backpropagation has already been proposed by le Cun [11]. This paper extends the framework to describe a state-feedback control structure.

3 Review of N-Stage Optimal Control

Before considering the neural network problem, we first review N-stage optimal control methods for designing terminal controllers. A typical N-stage problem can be phrased as: choose the state-space trajectory, $\mathbf{x} = [x(0), \ldots, x(N)]$, and open-loop control sequence, $\mathbf{u} = [u(0), \ldots, u(N-1)]$, for the discrete-time plant, f, which minimize a cost function subject to the constraints of the system:

$$\begin{aligned} J^o &= \min_{\mathbf{x},\mathbf{u}} \left(\phi[x(N)] + \sum_{i=0}^{N-1} L[x(i), u(i)] \right) \\ x(0) &= x_0, \quad \text{known initial condition} \qquad (1) \\ x(i+1) &= f(x(i), u(i)), \quad i = 0, \ldots, N-1. \qquad (2) \end{aligned}$$

The key points to observe are 1) the solution is found for a single trajectory, 2) the problem requires explicitly finding the open-loop control vectors at each increment in time. This problem can be solved by converting it into a two point boundary value problem (TPBVP). To do this, the plant-update equation is first adjoined to the cost function using a Lagrange multiplier sequence or *adjoint vector* sequence, λ :

$$\bar{J} = \phi[x(N)] + \sum_{i=0}^{N-1} \Big(L[x(i), u(i)] + \lambda(i+1)^T \big(f_i - x(i+1) \big) \Big).$$

For notational convenience, a Hamiltonian sequence H_i is usually defined as

$$H_i \equiv L[x(i), u(i)] + \lambda(i+1)^T f(x(i), u(i)), \qquad i = 0, \ldots, N-1.$$

Substituting this H_i into $\bar{J}$, rearranging terms, and considering differential changes in $\bar{J}$ due to changes in $\mathbf{x}$ and $\mathbf{u}$ gives

$$\begin{aligned} d\bar{J} = &\left(\frac{\partial \phi}{\partial x(N)} - \lambda(N)^T \right) dx(N) + \frac{\partial H_0}{\partial x(0)} dx(0) + \frac{\partial H_0}{\partial u(0)} du(0) \\ &+ \sum_{i=1}^{N-1} \left(\left(\frac{\partial H_i}{\partial x(i)} - \lambda(i)^T \right) dx(i) + \frac{\partial H_i}{\partial u(i)} du(i) \right). \end{aligned}$$

Since $x(0)$ is fixed, $dx(0) = 0$. By the Kuhn-Tucker conditions [12], in order to have optimal $x(i)$ and $u(i)$, we must have the gradient vector $\nabla \bar{J} = 0$. Thus, we need $d\bar{J} = 0$ for all choices of $dx(1), \ldots, dx(N)$, and $du(0), \ldots, du(N-1)$. For this to hold, we must have

$$\lambda(N)^T = \frac{\partial \phi}{\partial x(N)} \qquad (3)$$

$$\lambda(i)^T = \frac{\partial H_i}{\partial x(i)} = \left(\frac{\partial L_i}{\partial x(i)} + \lambda(i+1)^T \frac{\partial f_i}{\partial x(i)}\right), \qquad i = 1, \ldots, N-1 \tag{4}$$

$$0 = \frac{\partial H_i}{\partial u(i)} = \left(\frac{\partial L_i}{\partial u(i)} + \lambda(i+1)^T \frac{\partial f_i}{\partial u(i)}\right), \qquad i = 0, \ldots, N-1. \tag{5}$$

The terminal value of the Lagrange multiplier sequence λ is given by equation 3. Earlier values of λ are then found using the iterative procedure given by equation 4. In order to have optimal $u(i)$ and $x(i)$, the *optimality condition* (eqn. 5) must be satisfied. These equations define a TPBVP which typically must be solved numerically. One numerical method [1] consists of guessing an initial control sequence, $u_0(i)$, and then making many trial runs of the system (eqns. 1-2) After each run, the terminal error is swept backward (eqns. 3-4) and the control vector at each time step is updated by gradient descent:

$$u_{k+1}(i) = u_k(i) - \mu\left(\frac{\partial L_i}{\partial u(i)} + \lambda(i+1)^T \frac{\partial f_i}{\partial u(i)}\right)_k, \qquad i = 0, \ldots, N-1. \tag{6}$$

This derivation assumed that the number of time-steps was known, an assumption which is also implicitly made in BPTT. In many terminal control problems, the final time, t_f, is not known *a priori* and an optimal selection of this final time must be made. To do this in optimal control, the problem is typically solved in continuous time so that the number of control parameters does not change as t_f is changed. Thus we have:

$$\begin{aligned} J^o &= \min_{\mathbf{x},\mathbf{u},t_f} \left(\phi[x(t_f), t_f] + \int_0^{t_f} L[x(t), u(t)]\, dt\right) \\ x(0) &= x_0 \\ \dot{x}(t) &= f(x(t), u(t)). \end{aligned}$$

This problem is similarly converted to a two-point boundary value problem with continuous forward and backward equations. In addition to requiring that an optimality condition similar to equation 5 holds, this procedure requires that the *transversality condition*

$$0 = \left(\frac{d\phi}{dt}\right)\Big|_{t=t_f} = \left(\frac{\partial \phi}{\partial t} + L[x(t), u(t)] + \lambda^T(t) f(x(t), u(t))\right)\Big|_{t=t_f} \tag{7}$$

be satisfied at the terminal point. This can be done by using gradient descent on t_f[1], an approach which motivates method 1 of section 4.

[1] One such routine, *fcnopt*, written by Bryson [13], numerically integrates the continuous time system using a fixed number of steps. The integration time step, Δt, is then varied by gradient descent in order to effect a change on t_f. This provides a way of varying t_f without changing the number of control vectors. The routine is used for comparison purposes in section 5.

4 Neural Network Terminal Controller

4.1 Optimal Control Formulation of Problem

Given this background, we can now describe the neural network control structure shown in figure 1. As before, the block f is a discrete-time model of the dynamic plant with sampling interval Δt. The model determines the next state of the plant given the current state $x(i) \in \Re^n$ and control $u(i) \in \Re^m$. The block g is a nonlinear state-feedback controller consisting of a multilayer feedforward neural network with weight vector $\theta \in \Re^w$. Of course, the state vector is assumed to be available to the controller at each iteration. Although this paper focuses on multilayer networks, the algorithms derived are equally applicable to any parameterized mapping which is differentiable with respect to the input and parameter vectors.

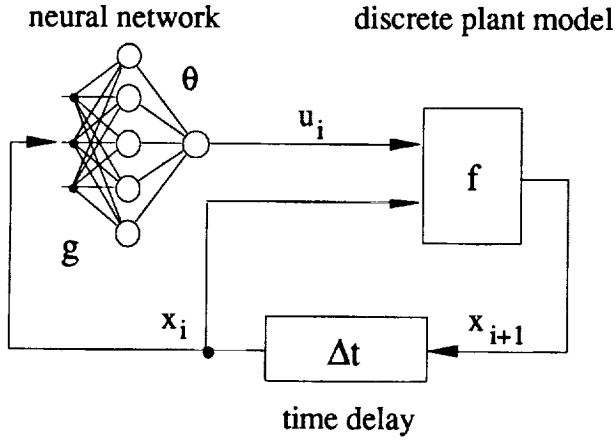

Figure 1: *Feedback control loop with neural network controller.*

So that we can deal systematically with having unknown trajectory lengths, we define a "time-to-go" function $N(x_0)$ which maps the initial state to the length of the associated trajectory. This is analogous to the parameter t_f described in section 3. Furthermore, in order to formulate the optimization problem over many trajectories simultaneously, we assume that the initial state is a discrete random vector[2] X_0 taking on values $x_o \in \left\{x_0^1, \ldots, x_0^P\right\}$ with some probability mass function $\mathsf{P}(X_0 = x_0^p)$.

The choice of $N(x_0)$, θ, and x_o determines a state trajectory $\mathbf{x}(x_0, \theta) = [x(0), \ldots, x(N(x_0))]$ and control sequence $\mathbf{u}(x_0, \theta) = [u(0), \ldots, u(N(x_0) - 1)]$. Given these sequences, we can define a trajectory cost $\tilde{J}(\mathbf{x}(x_0, \theta), \mathbf{u}(x_0, \theta), N(x_0))$. We could then attempt to find the θ and $N(x_0)$ which minimize the expected cost over all trajectories,

$$J^o = \min_{\theta, N(x_0)} \sum_{p=1}^{P} \tilde{J}(\mathbf{x}(x_0^p, \theta), \mathbf{u}(x_0^p, \theta), N(x_0^p))\ \mathsf{P}(X_0 = x_0^p), \tag{8}$$

[2]Justification of the derivations when X_0 is a continuous random vector is not presented due to the subtleties involved in applying the Kuhn-Tucker conditions when the parameter vector is not in $\Re^n$.

while implicitly incorporating the system constraints by judicious application of the chain-rule as the gradient $\nabla_\theta J$ is determined. If we disregard the minimization over $N(x_0)$ in equation 8 by assuming some value of N for each trajectory, then this unconstrained problem is equivalent to the usual formulation of BPTT. An alternative method is to rephrase equation 8 as a constrained optimization problem with the system constraints made explicit. To do this, we assume that the state and control sequences can be chosen independently of θ and then attempt to find the $\mathbf{x}(x_0)$, $\mathbf{u}(x_0)$, θ, and $N(x_0)$ which minimize the expected cost

$$J^o = \min_{\mathbf{x}(x_0),\mathbf{u}(x_0),\theta,N(x_0)} \sum_{p=1}^{P} \tilde{J}(\mathbf{x}(x_0^p), \mathbf{u}(x_0^p), N(x_0^p))\ \mathsf{P}(X_0 = x_0^p) \tag{9}$$

while also satisfying the system constraints,

$$x^p(0) = x_0^p\,, \qquad p = 1, \ldots, P \tag{10}$$

$$x^p(i+1) = f(x^p(i), u^p(i))\,, \qquad i = 0, \ldots, N(x_0^p) - 1, \quad p = 1, \ldots, P \tag{11}$$

$$u^p(i) = g(x^p(i), \theta)\,, \qquad i = 0, \ldots, N(x_0^p) - 1, \quad p = 1, \ldots, P. \tag{12}$$

along each trajectory. For notational convenience, we will use the expected value operator and drop the explicit dependence of $\mathbf{x}$ and $\mathbf{u}$ on x_0^p.

4.2 Choice of Cost Function

Equation 9 is a general form of the cost function. In this paper, a specific cost function is phrased as a Bolza problem:

$$J = \mathsf{E}\left[\phi[x(N(x_0)), N(x_0)\Delta t] + \sum_{i=0}^{N(x_0)-1} L[x(i), u(i)]\right]. \tag{13}$$

The term L is used to incorporate cost that is accumulated along the trajectory. For example, a quadratic function of the control vector will minimize control effort. Path state-constraints can be implemented as part of L using barrier-function methods [12]. The terminal cost ϕ can be used to implement soft terminal constraints. Note that hard terminal constraints which must be met *exactly* typically result in an ill-posed problem when this neural network structure is used. As an example cost function, consider

$$\phi[x(N^o), N^o\Delta t] = Q_t\, N^o\Delta t + (x_d - x(N^o))^T\, Q_x\, (x_d - x(N^o)). \tag{14}$$

This ϕ is linear in the final-time and is a quadratic function of the difference between the final state and a desired state. The matrix Q_x, assumed to be symmetric and positive semi-definite, weights the various state terms. Using this cost will minimize the trajectory time while approximately maintaining the desired end condition. More complicated functions of final state and time can be used as well. The soft terminal end-constraint on the state is adequately dealt with using standard BPTT. The extensions presented in this paper are designed to incorporate terms such as $Q_t\, N^o\Delta t$ which involve final time.

4.3 Solution of the Optimal Control Problem

We now derive two algorithms for solving the constrained optimization problem given by equations 10-12 and equation 13 using the optimal control methods introduced in section 3. Although somewhat complex, the derivations yield very simple, intuitive extensions to BPTT. Both algorithms use the same stochastic gradient descent structure as BPTT. The training procedures consist of a sequence of trial runs of the system with initial states drawn independently from the distribution of X_0. Each run is followed by a computation of the terminal error, a backward sweep of that error, and a weight update based on the instantaneous gradient from that one sweep. The new algorithms differ from BPTT in the selection of the stopping point for the forward sweep.

4.3.1 Method 1: Explicit Model of "Time-To-Go"

In the continuous open final-time problem of section 3, the final time t_f was treated as a parameter and gradient descent was used to update that parameter. Here we extend this idea to neural network controllers by attempting to explicitly determine the optimal trajectory lengths. In order to use gradient descent, however, we replace $N(x_o)$ with the continuous time-to-go function $t_f(x_0)$. To permit this, we first compute the state history $x(0), \ldots, x(N)$ along each trajectory by forward-iterating the system $N = \lfloor t_f(x_0)/\Delta t \rfloor$ steps using equations 10-12. We then simulate the effect of running the system for one partial step of length $\alpha\Delta t = t_f(x_0) - \Delta t\, N$ and find $x(t_f)$ by linear interpolation:

$$\begin{aligned} x(t_f) &\approx (1-\alpha)x(N) + \alpha x(N+1) \\ &= (1-\alpha)x(N) + \alpha f(x(N), u(N)). \end{aligned} \tag{15}$$

We also augment the cost function (eqn. 13) to reflect the cost incurred during this partial step:

$$J = \mathsf{E}\left[\phi[x(t_f), t_f] + \alpha L[x(N), u(N)] + \sum_{i=0}^{N-1} L[x(i), u(i)]\right]. \tag{16}$$

The $\mathbf{x}(x_0)$, $\mathbf{u}(x_0)$, θ, and $t_f(x_0)$ which minimize this cost function, subject to the system constraints (eqns. 10-12), are then sought. To do this, we first adjoin both the plant constraints (eqn. 11) and controller constraints (eqn. 12) to the cost function using two sets of Lagrange multiplier sequences, $\lambda_f(x_0) = [\lambda_f(0), \ldots, \lambda_f(N), \lambda_f(t_f(x_0))]$ and $\lambda_g(x_0) = [\lambda_g(0), \ldots, \lambda_g(N)]$. Like $\mathbf{x}$ and $\mathbf{u}$, these are ensembles of sequences indexed by x_0. The adjoined cost function becomes:

$$\begin{aligned} \bar{J} = \mathsf{E}\Big[&\phi[x(t_f), t_f] + \alpha L[x(N), u(N)] \\ &+ \lambda_f(t_f)^T\Big((1-\alpha)x(N) + \alpha f_N - x(t_f)\Big) + \lambda_g(N)^T\Big(g_N - u(N)\Big) \\ &+ \sum_{i=0}^{N-1}\Big\{L[x(i), u(i)] + \lambda_f(i+1)^T\Big(f_i - x(i+1)\Big) + \lambda_g(i)^T\Big(g_i - u(i)\Big)\Big\}\Big]. \end{aligned}$$

An ensemble of Hamiltonian sequences $H_i(x_0)$ is now defined as

$$\begin{aligned} H_i &\equiv L[x(i),u(i)] + \lambda_f(i+1)^T f(x(i),u(i)) + \lambda_g(i)^T g(x(i),\theta), \\ &\qquad\qquad i = 0,\ldots,N-1, \quad \forall x_0 \in \left\{x_0^1,\ldots,x_0^P\right\} \\ H_N &\equiv \alpha L[x(N),u(N)] + \lambda_f(t_f)^T\Big((1-\alpha)x(N) + \alpha f(x(N),u(N))\Big) + \lambda_g(N)^T g(x(N),\theta), \\ &\qquad\qquad \forall x_0 \in \left\{x_0^1,\ldots,x_0^P\right\}. \end{aligned}$$

Substituting the Hamiltonian sequences into $\bar{J}$, we simplify the expression to

$$\begin{aligned} \bar{J} &= \mathsf{E}\Bigg[\phi[x(t_f),t_f] + \Big(H_N - \lambda_f(t_f)^T x(t_f) - \lambda_g(N)^T u(N)\Big) \\ &\qquad + \sum_{i=0}^{N-1}\Big(H_i - \lambda_f(i+1)^T x(i+1) - \lambda_g(i)^T u(i)\Big)\Bigg]. \end{aligned}$$

Replacing t_f with $(N+\alpha)\Delta t$ and rearranging terms gives

$$\begin{aligned} \bar{J} &= \mathsf{E}\Bigg[\Big(\phi[x(t_f),(N+\alpha)\Delta t] - \lambda_f(t_f)^T x(t_f)\Big) + H_0 - \lambda_g(0)^T u(0) \\ &\qquad + \sum_{i=1}^{N}\Big(H_i - \lambda_f(i)^T x(i) - \lambda_g(i)^T u(i)\Big)\Bigg]. \end{aligned}$$

We now consider differential changes in $\bar{J}$ due to changes in θ, α, $x(1),\ldots,x(N),x(t_f)$, and $u(0),\ldots,u(N)$. We require that admissible changes in α be small enough that $\lfloor (t_f+dt_f)/\Delta t\rfloor = \lfloor t_f/\Delta t\rfloor$, thus allowing N to be treated as a constant. We then have

$$\begin{aligned} d\bar{J} &= \mathsf{E}\Bigg[\frac{\partial H_0}{\partial x(0)}\,dx(0) + \left(\frac{\partial \phi}{\partial x(t_f)} - \lambda_f(t_f)^T\right) dx(t_f) \\ &\qquad + \sum_{i=1}^{N}\left(\frac{\partial H_i}{\partial x_i} - \lambda_f(i)^T\right) dx(i) + \sum_{i=0}^{N}\left(\frac{\partial H_i}{\partial u_i} - \lambda_g(i)^T\right) du(i) \\ &\qquad + \sum_{i=0}^{N}\left(\frac{\partial H_i}{\partial \theta}\right) d\theta + \left(\frac{\partial \phi}{\partial t_f}\Delta t + \frac{\partial H_N}{\partial \alpha}\right) d\alpha\Bigg]. \end{aligned}$$

Since $x(0)$ is fixed for each trajectory, $dx(0) = 0$.

The function $t_f(x_0)$ can be implemented using a second neural network with weights ϑ,

$$t_f = \tau(x(0),\vartheta),$$

which will approximate the optimal time-to-go function to any required accuracy. The variation of the step ratio α can now be written as a function of the variation of the network weights ϑ:

$$\begin{aligned} \alpha &= \frac{1}{\Delta t}t_f - N \\ &= \frac{1}{\Delta t}\tau(x_0,\vartheta) - N \\ d\alpha &= \frac{1}{\Delta t}\frac{\partial \tau}{\partial \vartheta}\,d\vartheta. \end{aligned}$$

Substituting this expression for $d\alpha$ back into $d\bar{J}$ yields

$$\begin{aligned} d\bar{J} &= \mathsf{E}\Bigg[\left(\frac{\partial \phi}{\partial x(t_f)} - \lambda_f(t_f)^T\right) dx(t_f) \\ &\quad + \sum_{i=1}^{N}\left(\frac{\partial H_i}{\partial x_i} - \lambda_f(i)^T\right) dx(i) + \sum_{i=0}^{N}\left(\frac{\partial H_i}{\partial u_i} - \lambda_g(i)^T\right) du(i) \\ &\quad + \sum_{i=0}^{N}\left(\frac{\partial H_i}{\partial \theta}\right) d\theta + \left(\frac{\partial \phi}{\partial t_f} + \frac{1}{\Delta t}\frac{\partial H_N}{\partial \alpha}\right)\frac{\partial \tau}{\partial \vartheta} d\vartheta\Bigg]. \end{aligned}$$

In order to have optimal $x(i)$ and $u(i)$, we must have $\nabla \bar{J} = 0$, which requires that along every trajectory $d\bar{J} = 0$ for all choices of $d\theta$, $d\vartheta$, $dx(1), \ldots, dx(N), dx(t_f)$, and $du(0), \ldots, du(N)$. Therefore,

$$\lambda_f(t_f)^T = \frac{\partial \phi}{\partial x(t_f)}, \quad \forall x_0 \in \left\{x_0^1, \ldots, x_0^P\right\} \tag{17}$$

$$\lambda_f(i)^T = \frac{\partial H_i}{\partial x_i}, \quad i = 1, \ldots, N, \quad \forall x_0 \in \left\{x_0^1, \ldots, x_0^P\right\} \tag{18}$$

$$\lambda_g(i)^T = \frac{\partial H_i}{\partial u_i}, \quad i = 0, \ldots, N, \quad \forall x_0 \in \left\{x_0^1, \ldots, x_0^P\right\} \tag{19}$$

$$0 = \mathsf{E}\left[\sum_{i=0}^{N}\frac{\partial H_i}{\partial \theta}\right] = \mathsf{E}\left[\sum_{i=1}^{N}\left(\lambda_g(i)^T\frac{\partial g_i}{\partial \theta}\right)\right] \tag{20}$$

$$0 = \mathsf{E}\left[\left(\frac{\partial \phi}{\partial t_f} + \frac{1}{\Delta t}\frac{\partial H_N}{\partial \alpha}\right)\frac{\partial \tau}{\partial \vartheta}\right]. \tag{21}$$

The terminal condition on λ_f is given by equation 17. It is identical to the expression found for N-stage optimal control (eqn. 3). We can expand the equations for the adjoint vector (eqns. 18-19). For $i = N$ we have

$$\lambda_g(N)^T = \alpha\left(\frac{\partial L_N}{\partial u(N)} + \lambda_f(t_f)^T\frac{\partial f_N}{\partial u(N)}\right) \tag{22}$$

$$\lambda_f(N)^T = \alpha\left(\frac{\partial L_N}{\partial x(N)} + \lambda_f(t_f)^T\frac{\partial f_N}{\partial x(N)}\right) + \lambda_g(N)^T\frac{\partial g_N}{\partial x(N)} + (1-\alpha)\,\lambda_f(t_f)^T. \tag{23}$$

These expressions sweep the terminal error $\lambda_f(t_f)$ back through the simulated final step. For $i = 0, \ldots, N-1$ equations 18-19 become

$$\lambda_g(i)^T = \frac{\partial L_i}{\partial u(i)} + \lambda_f(i+1)^T\frac{\partial f_i}{\partial u(i)} \tag{24}$$

$$\lambda_f(i)^T = \left(\frac{\partial L_i}{\partial x(i)} + \lambda_f(i+1)^T\frac{\partial f_i}{\partial x(i)}\right) + \lambda_g(i)^T\frac{\partial g_i}{\partial x(i)}. \tag{25}$$

These expressions sweep the error at time step N back through the iterations of the feedback loop. Note that the backward sweep equation for the neural network structure (eqn. 25) and for optimal control (eqn. 4) differ only in the term $\partial g_i/\partial x(i)$. This term appears in equation 25 because

in a feedback control loop, the $u(i)$ are determined by $x(i)$. The optimality condition given by equation 20 is analogous to equation 5 derived for N-stage optimal control. In optimal control, however, the gradient is found with respect to each of the control vectors and gradient descent is directly employed on each of these vectors separately (eqn. 6). Here, the average weight gradient over the trajectory is used since the same control weights must be used at each stage. In practice, a weight update is made after each forward/backward sweep using the instantaneous gradient:

$$\theta_{k+1}(i) \;=\; \theta_k(i) - \mu_\theta \sum_{i=1}^{N} \left(\lambda_g(i)^T \frac{\partial g_i}{\partial \theta}\right)_k. \tag{26}$$

To analyze this stochastic gradient descent over time, we consider the total weight change after K random trajectories:

$$\Delta\theta \;=\; -\mu_\theta \sum_{k=1}^{K}\sum_{i=1}^{N} \left(\lambda_g^k(i)^T \frac{\partial g_i^k}{\partial \theta_k}\right).$$

If μ_θ is small, we can assume $\theta_1 \approx \theta_2 \approx \cdots \approx \theta_K$. Furthermore, the probability that x_o^k is chosen on the k^{th} iteration is $\mathsf{P}(X_0 = x_o^k)$. Thus the summation converges to

$$\Delta\theta \;\approx\; -\mu_\theta K\, \mathsf{E}\left[\sum_{i=1}^{N} \left(\lambda_g(i)^T \frac{\partial g_i}{\partial \theta}\right)\right].$$

We see that, as with the LMS algorithm, the slow adaptation process smooths the weight-gradient estimate so that the change in the weight vector follows the true gradient on average [14].

So far this derivation has produced an algorithm exactly like BPTT with an added partial final-step. The critical new term, however, is the transversality condition (eqn. 21) which can be expanded to give

$$0 \;=\; \mathsf{E}\left[\lambda_\tau^T \frac{\partial \tau}{\partial \vartheta}\right], \tag{27}$$

where

$$\lambda_\tau^T \;=\; \left(\frac{\partial \phi}{\partial t_f} + \frac{L_N}{\Delta t} + \lambda_f(t_f)^T \left(\frac{f[x(N), u(N)] - x(N)}{\Delta t}\right)\right) \tag{28}$$

will be called the *final-time error*. Equation 28 is a discrete-time approximation of equation 7. To improve the final-time, we can compute λ_τ after each sweep, backpropagate it through the network τ, and update the time-to-go network weights based on the resulting instantaneous gradient estimate:

$$\vartheta_{k+1}(i) \;=\; \vartheta_k(i) - \mu_\vartheta \left(\lambda_\tau^T \frac{\partial \tau}{\partial \vartheta}\right)_k. \tag{29}$$

To gain some insight into the transversality condition, consider the three terms of the final-time error (eqn. 28). The first term, $\partial\phi/\partial t_f$, is the direct effect on the terminal cost of varying t_f. The second term, $L_N/\Delta t$, is an incremental trajectory cost. The last term can be rewritten as

$$\lambda_f(t_f)^T \left(\frac{f[x(N), u(N)] - x(N)}{\Delta t}\right) \;\approx\; \left(\frac{\partial \phi}{\partial x}\right)^T \left(\frac{dx}{dt}\right)\Big|_{t_f}.$$

This is the indirect effect on the terminal error due to the terminal state changing. In summary, the algorithm uses the network τ to predict the stopping time. The final-state error (eqn. 17) for the given prediction of t_f is backpropagated through the stages of the feedback loop to update the controller network weights and the final-time error (eqn. 28) is backpropagated once through τ to update the time-to-go network weights.

4.3.2 Method 2: Implicit Determination of "Time-To-Go"

Although the optimization problem consists of finding both the optimal θ and $N(x_0)$, once training is complete it is often only necessary to store θ. The evolution of the actual physical control system will typically provide a natural stopping point. In training, however, some method must still be used to determine at what time-step to stop the forward run and compute the terminal error. In this second approach, we assume that once the controller is trained, the time-to-go mapping is no longer necessary and thus it is not explicitly stored. This, as we shall see from the experimental results, yields a simpler and more robust algorithm.

In order to optimize the choice of time-to-go function $N(x_0)$, we begin by reconsidering the general cost function (eqn. 9) and distributing the minimization over N across the expected value

$$\begin{aligned} J^o &= \min_{\mathbf{x}(x_0),\mathbf{u}(x_0),\theta,N(x_0)} \sum_{p=1}^{P} \tilde{J}\Big(\mathbf{x}(x_0^p),\mathbf{u}(x_0^p),N(x_0^p)\Big)\ \mathsf{P}(X_0 = x_0^p) \\ &= \min_{\mathbf{x}(x_0),\mathbf{u}(x_0),\theta} \sum_{p=1}^{P} \Big\{ \min_{N} \tilde{J}\Big(\mathbf{x}(x_0^p),\mathbf{u}(x_0^p),N\Big)\Big\}\ \mathsf{P}(X_0 = x_0^p). \end{aligned}$$

We are now minimizing the cost function over $N \in \mathcal{Z}^+$ for a specific trajectory and choice of θ. The distribution is allowed since the original minimization on $N(x_0)$ was taken over $\Re^n \times \mathcal{Z}^+$. The minimizing value of N within the expected value will depend on the choice of x_0 and θ and is defined as

$$N^o = N^o(x_0,\theta) \equiv \arg\min_{N} \tilde{J}\Big(\mathbf{x}(x_0),\mathbf{u}(x_0),N\Big).$$

Note that this is not the desired function $N^o(x_0)$ unless we have optimal θ. Assuming for the moment that N^o is known, we can substitute it back into the expression for J:

$$J^o = \min_{\mathbf{x}(x_0),\mathbf{u}(x_0),\theta} \sum_{p=1}^{P} \tilde{J}\Big(\mathbf{x}(x_0^p),\mathbf{u}(x_0^p),N^o\Big)\ \mathsf{P}(X_0 = x_0^p).$$

Effectively, we have projected the problem of optimizing over $\mathbf{x}$, $\mathbf{u}$, θ and $N(x_0)$ into a problem of only optimizing over $\mathbf{x}$, $\mathbf{u}$, and θ. We will come back to how to determine N^o.

First we consider how to find the optimal θ by again deriving a set of necessary first-order stationary conditions. As before, we seek the $\mathbf{x}$, $\mathbf{u}$, θ, and $N(x_0)$ which minimize

$$J = \mathsf{E}\left[\phi[x(N(x_0)),N(x_0)\Delta t] + \sum_{i=0}^{N(x_0)-1} L[x(i),u(i)]\right] \tag{30}$$

subject to the constraints given by equations 10-12. Substituting in, the as of yet unknown, N^o, we can write

$$J = \mathsf{E}\left[\phi[x(N^o), N^o\Delta t] + \sum_{i=0}^{N^o-1} L[x(i), u(i)]\right]. \tag{31}$$

We adjoin the system constraints to the cost function using two Lagrange multiplier sequences, λ_f and λ_g, as before:

$$\begin{aligned}\bar{J} = \mathsf{E}\Bigg[&\phi[x(N^o), N^o\Delta t] \\ &+ \sum_{i=0}^{N^o-1}\Big\{L[x(i), u(i)] + \lambda_f(i+1)^T\Big(f_i - x(i+1)\Big) + \lambda_g(i)^T\Big(g_i - u(i)\Big)\Big\}\Bigg].\end{aligned}$$

We substitute the Hamiltonian sequence,

$$H_i \equiv L[x(i), u(i)] + \lambda_f(i+1)^T f(x(i), u(i)) + \lambda_g(i)^T g(x(i), \theta), \quad i = 0, \ldots, N^o - 1,$$

into $\bar{J}$ and rearrange the terms to produce

$$\begin{aligned}\bar{J} = \mathsf{E}\Bigg[&\phi[x(N^o), N^o\Delta t] - \lambda_f(N^o)^T x(N^o) + H_0 - \lambda_g(0)^T u(0) \\ &+ \sum_{i=1}^{N^o-1}\Big(H_i - \lambda_f(i)^T x(i) - \lambda_g(i)^T x u(i)\Big)\Bigg].\end{aligned}$$

Consider differential changes in $\bar{J}$ due to changes in θ, $x(1), \ldots, x(N^o)$, and $u(0), \ldots, u(N^o-1)$. We require that admissible $\delta\theta$ are chosen small enough that $N^o(x_0, \theta) = N^o(x_0, \theta + \delta\theta)$, thus treating N^o as a constant:

$$\begin{aligned}d\bar{J} = \mathsf{E}\Bigg[&\frac{\partial H_0}{\partial x(0)}dx(0) + \left(\frac{\partial \phi}{\partial x(N^o)} - \lambda_f(N^o)^T\right)dx(N^o) \\ &+ \sum_{i=1}^{N^o-1}\left(\frac{\partial H_i}{\partial x(i)} - \lambda_f(i)^T\right)dx(i) + \sum_{i=0}^{N^o-1}\left(\frac{\partial H_i}{\partial u(i)} - \lambda_g(i)^T\right)du(i) \\ &+ \sum_{i=0}^{N^o-1}\left(\frac{\partial H_i}{\partial \theta}\right)d\theta\Bigg].\end{aligned}$$

Again, we know that $dx(0) = 0$. In order to have optimal θ, $x(i)$ and $u(i)$, we require that $d\bar{J} = 0$ for all choices of $d\theta$, $dx(1), \ldots, dx(N^o)$, and $du(0), \ldots, du(N^o - 1)$ along every trajectory, thus giving

$$\begin{aligned}\lambda_f(N^o)^T &= \frac{\partial \phi}{\partial x(N^o)}, && \forall x_0 \in \left\{x_0^1, \ldots, x_0^P\right\} \\ \lambda_f(i)^T &= \frac{\partial H_i}{\partial x(i)}, && i = 1, \ldots, N^o - 1, \quad \forall x_0 \in \left\{x_0^1, \ldots, x_0^P\right\}\end{aligned} \tag{32}$$

$$= \left(\frac{\partial L_i}{\partial x(i)} + \lambda_f(i+1)^T \frac{\partial f_i}{\partial x(i)}\right) + \lambda_g(i)^T \frac{\partial g_i}{\partial x(i)} \tag{33}$$

$$\lambda_g(i)^T = \frac{\partial H_i}{\partial u(i)}, \qquad i = 0, \ldots, N^o - 1, \quad \forall x_0 \in \left\{x_0^1, \ldots, x_0^P\right\}$$

$$= \left(\frac{\partial L_i}{\partial u(i)} + \lambda_f(i+1)^T \frac{\partial f_i}{\partial u(i)}\right), \tag{34}$$

$$0 = \mathsf{E}\left[\sum_{i=0}^{N^o-1} \frac{\partial H_i}{\partial \theta}\right]$$

$$= \mathsf{E}\left[\sum_{i=0}^{N^o-1} \lambda_g(i)^T \frac{\partial g_i}{\partial \theta}\right]. \tag{35}$$

Equations 32-34 are the same terminal condition and backward sweep equations as equation 17 and equations 24-25. The optimality condition (eqn. 35) is again satisfied using stochastic gradient descent:

$$\theta_{k+1}(i) = \theta_k(i) - \mu_\theta \sum_{i=0}^{N^o-1} \left(\lambda_g(i)^T \frac{\partial g_i}{\partial \theta}\right)_k. \tag{36}$$

The extension beyond BPTT in this algorithm is the new expression

$$N^o = \arg\min_N \left(\phi[x(N), N\Delta t] + \sum_{i=0}^{N-1} L[x(i), g(x(i), \theta)]\right) \tag{37}$$

which takes the place of the transversality condition of method 1 (eqn. 27). This expression is not in the form of a stationary condition but is rather in the form of an explicit expression for minimum value. Application of this second method is straight-forward: we simply stop the forward run at the time-step which minimizes the total cost function for the current value of θ. Note that this is not necessarily the time-step where the state is closest to the terminal position. We then use equations 33-34 to propagate the error at that time-step through the control feedback loop. In practice we assume that we can find some upper bound on N^o, N_{max}. The forward sweep is terminated after N_{max} iterations, $\tilde{J}\left(\mathbf{x}(x_0), \mathbf{u}(x_0), N\right)$ is computed for each $N = 0, \ldots, N_{max}$, and the time-step with minimum value of $\tilde{J}$ is chosen as N^o for that trajectory.

4.4 Comparison of Optimal Control Formulation to BPTT

It is instructive to compare the Lagrange multiplier equations just derived with the calculations carried out in BPTT. First consider equation 32. If we used a quadratic soft terminal-state constraint (eqn. 14), we can evaluate the terminal value of the Lagrange multiplier sequence as

$$\lambda_f(N)^T = \frac{\partial \phi}{\partial x(N)}$$

$$= \partial\Big((x_d - x(N))^T Q_x (x_d - x(N))\Big)/\partial x(N)$$

$$= 2(x_d - x(N))^T Q_x.$$

This is the same scaled state-error used in BPTT.

Next consider the backward equations (eqns. 33-34). Figure 2 shows a graphical representation of the various components of these equations for one time-step. The thin lines are the signal flow for the state and control vectors. The bold lines are the signal flow for the adjoint vectors. We can interpret $\lambda_f(i)$ to be the squared-error derivative at the plant output at the i^{th} stage and $\lambda_g(i)$ to be the squared-error derivative at the controller output. With these interpretations, we see that the adjoint vector equations exactly describe the BPTT process and that the λ are, in fact, the backpropagated quantities. The term $\lambda_g(i)^T \partial g_i/\partial x(i)$ is computed in BPTT by propagating the error $\lambda_g(i)$ through the controller network using the backpropagation algorithm described by Rumelhart [15] with internal activation values determined by the forward sweep of $x(i)$. This backpropagation also computes the weight gradient component $\lambda_g(i)^T \partial g_i/\partial \theta$. The error components $\lambda_f(i+1)^T \partial f_i/\partial u(i)$ and $\lambda_f(i+1)^T \partial f_i/\partial x(i)$ can be computed in several ways. If f is a neural network model of the plant then backpropagation can be used, as in the work of Nguyen and Widrow [5]. If the equations of the plant are known, then the Jacobian matrices $f_u(i) = \partial f_i/\partial u(i)$ and $f_x(i) = \partial f_i/\partial x(i)$ can be computed analytically. Alternatively, these matrices can be estimated numerically by perturbing the inputs to f and observing the output perturbations. The Jacobian matrices are then directly multiplied by $\lambda_f(i+1)$.

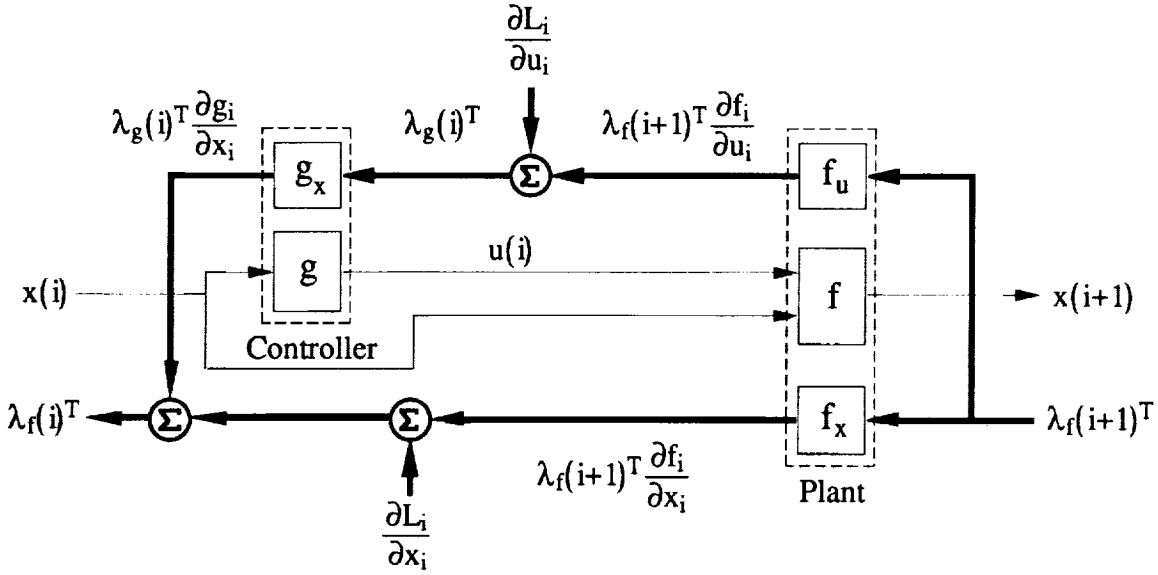

Figure 2: *Forward and backward signal paths through one stage.*

5 Experimental Results

This section presents the results of applying the time-optimal techniques to a simple optimal control problem. This Zermelo problem, proposed by Bryson [1], consists of a boat navigating in a river

with a linear current profile as shown in figure 3. The controller is required to steer the boat to the goal at the center of the river in minimum time from many initial positions. This system was chosen because it is simple enough to allow the family of optimal trajectories to be visualized and yet provides an interesting time-minimization problem.

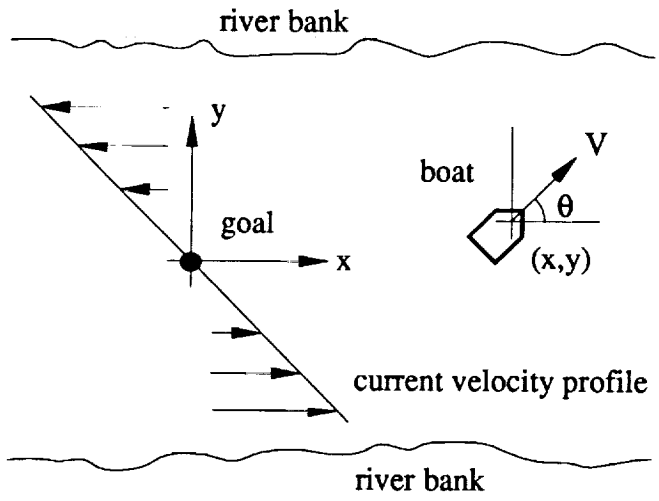

Figure 3: *Control of a boat navigating through a river with linear current profile.*

The boat state-vector was $[x, y, \theta_r]$, where (x, y) was the position of the bow of the boat with respect to the goal and $\theta_r = \theta - \tan^{-1}(y/x)$ was the direction of motion of the boat relative to the direction to the goal. This choice of relative angle instead of absolute angle prevented the desired direction from being a discontinuous function of position along the radial $\{x, y : \tan^{-1}(y/x) = \pi\}$. The control vector was the change in boat direction over a sampling interval, $\Delta\theta$. The velocity of the boat was a constant with respect to the water, $V = 1$, and the current profile was chosen to be $V_r = -y/25$. This gave the plant equations

$$\begin{aligned} x(i+1) &= x(i) + \cos\left(\theta_r(i) + \tan^{-1}\frac{y(i)}{x(i)}\right) - \frac{y(i)}{25} \\ y(i+1) &= y(i) + \sin\left(\theta_r(i) + \tan^{-1}\frac{y(i)}{x(i)}\right) \\ \theta_r(i+1) &= \theta_r(i) + u(i) \quad \text{normalized to } [-\pi, \pi] \quad . \end{aligned}$$

The initial state for each run, $(x_0, y_0, \theta_{r,0})$, was chosen from a uniform distribution over

$$\left\{x, y, \theta_r \; : \; x \in [-50, 50],\, y \in [-50, 50],\, \theta_r \in [-\pi/4, \pi/4]\right\}.$$

The cost function consisted of a soft quadratic end-constraint and a time minimization term:

$$J \;=\; \mathsf{E}\left[N^o \Delta t \;+\; x(N^o)^2 \;+\; y(N^o)^2\right].$$

The controller and time-to-go neural networks each had 3 input nodes, 10 hidden nodes, and 1 output node. The number of hidden nodes was chosen by evaluating performance with different network configurations and picking the smallest network with satisfactory performance.

Before training began, the controller weights were initialized to random values. A few sample trajectories for the untrained controller are shown in figure 4. These and all subsequent trajectories are plotted against paths derived by optimal control methods for each of the initial states separately[3]. The controller was then trained to emulate a coarse control law, thus providing an initial, rough guess of the weights. This coarse controller simply pointed the boat towards the goal at all times. Sample trajectories after this weight initialization training are shown in figure 5. Notice that the paths are not close to the optimal trajectories. This pre-training was done in order to keep the subsequent, on-line learning process stable. This weight initialization scheme was chosen as an alternative to the procedure used by Nguyen and Widrow [5] which required a scheduling of the initial states presented to the system. Their procedure adapted the controller based on many initial states whose $N^o(x_0)$ were small before initial states with larger $N^o(x_0)$ were attempted.

The two extended BPTT algorithms were then used to further train the control system to minimize the above cost function. Trajectories for method 1 after about 150,000 training cycles with $\mu_\theta = 10^{-3}$ and $\mu_\vartheta = 10^{-4}$ are shown in figure 6. Figure 7 shows the same trajectories for method 2 after about 30,000 training cycles with $\mu_\theta = 10^{-2}$. Larger values of μ caused learning instabilities. Although the trajectories were not identical to the optimal control solutions, they were very close. In figure 7, all of the trajectory times are within Δt of the optimal control solutions. In figure 6 the times differ by at most $3\Delta t$. Algorithm 1 was found empirically to require smaller μ than algorithm 1 to prevent the learning process from diverging. Also, the paths in algorithm 1 were not quite as close to the optimal continuous solutions. One possible reason for this was that errors in the time-prediction mapping may have caused artificial errors in the control mapping. Since the time-to-go is not explicitly represented in algorithm 2, it does not suffer from this problem.

6 Conclusions

This paper was written with two goals. The first was the presentation of two algorithms which extend backpropagation through time (BPTT) to terminal control problems with unknown final time. One algorithm that uses an auxiliary network to explicitly predict the optimal final-time is of theoretical interest because it is a direct extension of optimal control techniques, specifically, the optimal control method which satisfies a transversality condition through gradient descent on t_f. A second algorithm stops runs at the time-step which minimizes the total cost function, including the time-minimization term. This algorithm was found to be less sensitive to the learning rate, less likely to diverge, and easier to implement than was the first algorithm. Because of this, use of the second algorithm is recommended unless the time-to-go mapping is required. Even in situations which do not require final-time minimization, it is still necessary to decide upon a stopping point for the forward run. The choice of N should not be made indiscriminately, as this choice will have a direct effect on the controller weights. For example, a stopping heuristic which tends to pick small N might result in a controller which uses larger control effort than otherwise. Because of

[3]Recall from section 3 that the optimal control formulation is not valid for discrete time controllers with open final-time. Thus, comparisons were made using a continuous plant and Bryson's *fcnopt* routine [13].

this, algorithm 2 should be used to stop the run at the iteration which minimizes the required cost function, even if the cost does not involve trajectory-time.

The second goal of this paper was the demonstration of the relationship between classical optimal control methods and BPTT. BPTT, with the open final-time extensions presented here, can be directly derived using standard optimal control methods, and the propagated errors in BPTT are equivalent to the Lagrange multipliers in optimal control. A realization of the similarities between optimal control and BPTT will allow the application of optimal control techniques to neural networks, while, conversely, the ability of neural networks to realize a wide class of nonlinear functions will permit them to solve problems in classical optimal control that might otherwise have been difficult.

The use of a multi-layer neural network as a general tool for synthesizing an optimal state-feedback terminal controller depends on certain assumptions. First, by definition, the neural network realizes a continuous mapping from state to control. In terminal control problems, the desired mapping is not necessarily continuous. For example, in the boat problem above, if the direction had not been chosen as a relative angle, there would have been a discontinuity. Although a network can approximate a function with discontinuities, I have found that, in practice, this makes it difficult to obtain convergence for terminal controller problems. Second, due to the structure of the network, the range of state space over which the network attempts to learn the optimal control law must be restricted. We also assume that the problem is stationary, so the optimal cost function, J^o, does not depend explicitly on time. Thus the controller weights θ are not dependent on time either. Finally, and perhaps most importantly, the methods described assume full state feedback is possible.

There are a number of issues open to future research. Necessary first order stationary conditions have been examined, but an investigation of sufficient conditions for local minima with regard to conjugate and focal points [1] is still needed. Furthermore, the neural network control scheme presented here relies on full state information. In situations where the controller does not have this information, some form of state estimation must be used. This is an issue that needs to be addressed in the context of neural network terminal controllers.

7 Acknowledgements

I would like to thank the members of B. Widrow's research group, most especially M. Lehr and S. Piché, for their ideas and comments. Also, a special thanks to A. E. Bryson for discussions on optimal control.

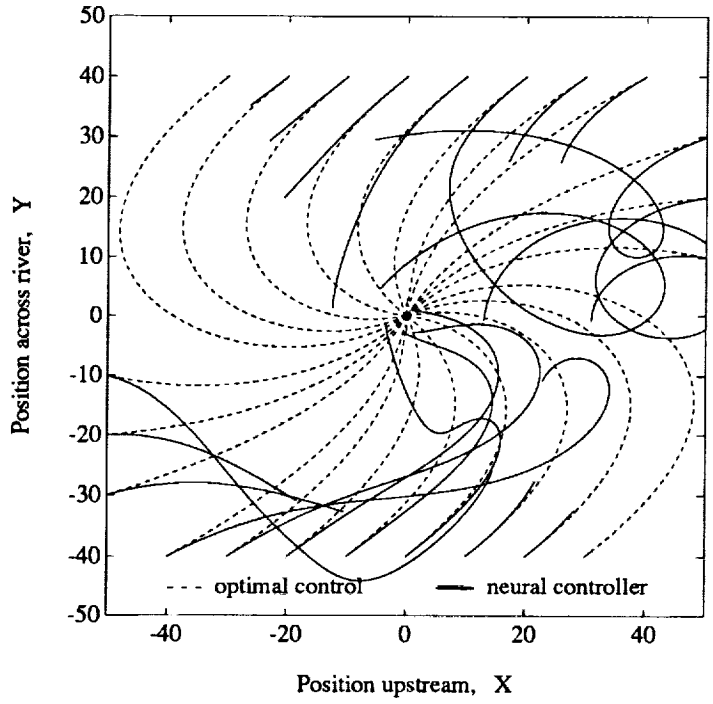

Figure 4: *Trajectories for 3x10x1 neural controller with random weights.*

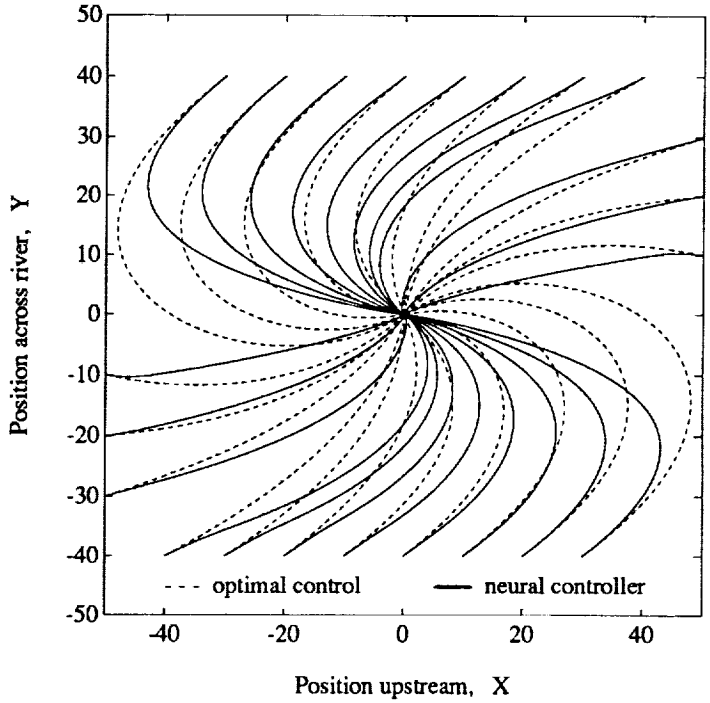

Figure 5: *Trajectories for 3x10x1 neural controller after rough weight initialization.*

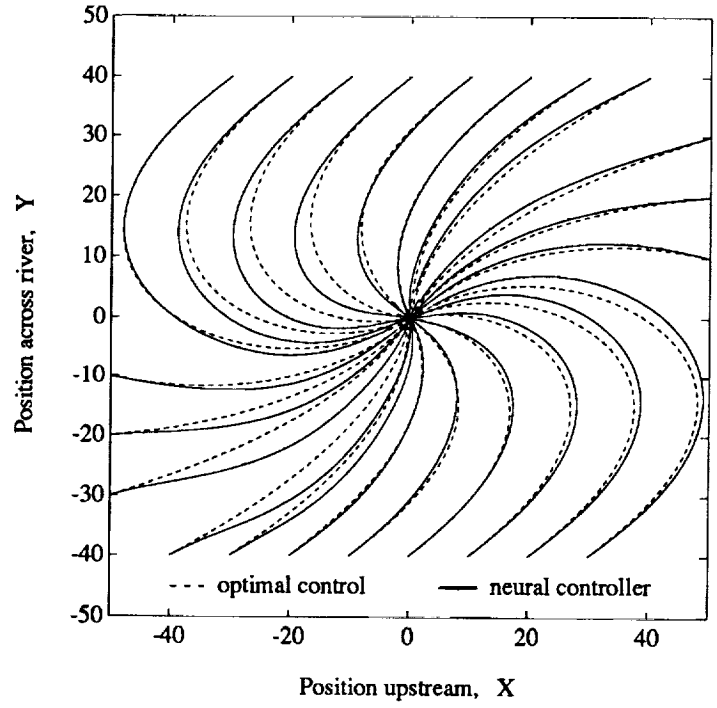

Figure 6: *Trajectories for 3x10x1 neural controller trained using method 1.*

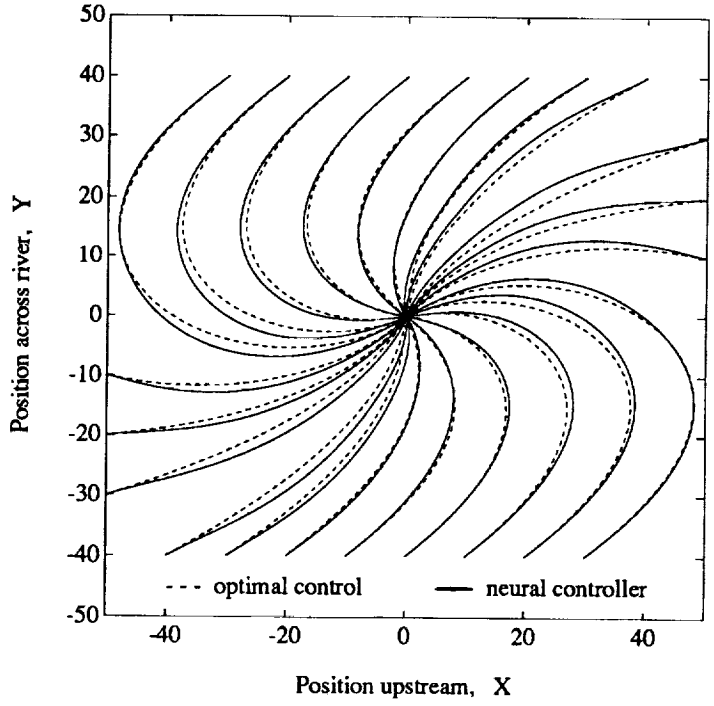

Figure 7: *Trajectories for 3x10x1 neural controller trained using method 2.*

REFERENCES

1. Bryson, A. E., Jr.; and Ho, Y.: Applied Optimal Control. Blaisdell Publishing Co., New York, 1969.

2. Narendra, K. S.; and Parthasarathy, K.: Gradient Methods for Optimization of Dynamic Systems Containing Neural Networks. IEEE Transactions on Neural Networks, vol. 2, March 1991, pp. 252–262.

3. Narendra, K. S.; and Parthasarathy, K.: Identification and Control of Dynamical Systems Using Neural Networks. IEEE Transactions on Neural Networks, vol. 1, March 1990, pp. 4–27.

4. Williams, R. J.; and Zipser, D.: A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. Neural Computation, vol. 1, no. 2, 1989, pp. 270–280.

5. Nguyen, D.; and Widrow, B.: The Truck Backer-Upper: An Example of Self-Learning in Neural Networks. Proceedings of the International Joint Conference on Neural Networks, vol. II, Washington, DC, June 1989, pp. 357–363.

6. Werbos, P.: Backpropagation through Time: What It Does and How to Do It. Proceedings of the IEEE, vol. 78, Oct. 1990, pp. 1550–1560.

7. Bellman, R.: Dynamic Programming, Princeton University Press, Princeton, NJ, 1957.

8. Dreyfus, S. E.: Dynamic Programming and the Calculus of Variations. Academic Press, New York, 1965.

9. Cybenko, G.: Approximation by Superpositions of a Sigmoidal Function. Mathematics of Control, Signals, and Systems, vol. 2, no. 4, 1989, pp. 303–314.

10. Irie, B.; and Miyake, S.: Capabilities of Three-Layered Perceptrons. Proceedings of the IEEE Second International Conference on Neural Networks, vol. I, San Diego, CA, July 1988, pp. 641–648.

11. le Cun, Y.: A Theoretical Framework for Back-Propagation. Proceedings of the 1988 Connectionist Models Summer School, San Mateo, CA, June 17–26, 1988, pp. 21–28. Morgan Kauffman.

12. Luenberger, D. G.: Linear and Nonlinear Programming. Addison-Wesley, Reading, MA, second edition, 1984.

13. Bryson, A. E., Jr.: Optimal Control of Dynamic Systems. Class notes AA-278A, Stanford University, 1990.

14. Widrow, B.; and Lehr, M.: 30 Years of Adaptive Neural Networks: Perceptron, Madaline and Backpropagation. Proceedings of the IEEE, Sept. 1990, pp. 1415–1442.

15. Rumelhart, D.; Hinton, G.; and Williams, R.: Learning Internal Representations by Error Propagation. Parallel Distributed Processing, vol. 1, eds. D. Rummelhart and J. McClelland, The MIT Press, Cambridge, MA, 1986.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1992	Contractor Report

4. TITLE AND SUBTITLE
Two Neural Network Algorithms for Designing Optimal Terminal Controllers with Open Final-Time

5. FUNDING NUMBERS
NGT-50642

6. AUTHOR(S)
Edward S. Plumer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Electrical Engineering
Stanford University
Durand Building, Room 104
Stanford, CA 94305-4055

8. PERFORMING ORGANIZATION REPORT NUMBER
A-92194

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Ames Research Center
Moffett Field, CA 94035-1000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CR-177599

11. SUPPLEMENTARY NOTES
Point of Contact: Charles Jorgensen, Ames Research Center, MS 269-3, Moffett Field, CA 94035-1000
(415) 604-6725

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified — Unlimited
Subject Category 63

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Multilayer neural networks, trained by the backpropagation through time algorithm (BPTT), have been used successfully as state-feedback controllers for nonlinear terminal control problems. Current BPTT techniques, however, are not able to deal systematically with open final-time situations such as minimum-time problems. Two approaches which extend BPTT to open final-time problems are presented. In the first, a neural network learns a mapping from initial-state to time-to-go. In the second, the optimal number of steps for each trial run is found using a line-search. Both methods are derived using Lagrange multiplier techniques. This theoretical framework is used to demonstrate that the derived algorithms are direct extensions of forward/backward sweep methods used in N-stage optimal control. The two algorithms are tested on a Zermelo problem and the resulting trajectories compare favorably to optimal control results.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Neural network, Terminal controller, Optimal control	23
	16. PRICE CODE A02

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		