Synthesising Optimal Feedback Control with Neural Networks[1]

Nathan Edwards

[1]This Dissertation is submitted in partial fulfillment of the requirements for the degree of Bachelor of Science with Honours in Mathematics and Computer Science from the University of Western Australia.

Abstract

Despite recent advances in numerical methods for finding optimal open-loop control, the problem of finding optimal feedback control laws for general, non-linear problems with arbitrary initial conditions cannot easily be achieved except in rare cases where the HJB equation can be solved. Two methods of synthesising optimal feedback controls are discussed. The first, perturbation feedback control, adds robustness and error tolerance properties local to existing optimal open-loop trajectories. The second trains an artificial neural network to emulate the optimal feedback control law over an arbitrary set of initial conditions. Two training methodologies are presented for training this neural network controller. These synthesised feedback controllers are tested on a number of challenging problems which demonstrate that the perturbation feedback controller achieves excellent suboptimal results local to a previously computed open-loop trajectory, and that the artificial neural network controller can emulate the optimal feedback control arbitrarily closely for all trajectories with initial conditions from some non-trivial subset of the state space.

Acknowledgements

I would like to thank Dr. C. J. Goh for opening the field of neural networks to me, for providing me with a challenging and stimulating research problem for this dissertation, and for his patience, guidance and encouragement throughout the year. Thanks also are due to Dr. N. Spadaccini for his suggestions and encouragement.

Contents

1 Introduction

The numerical solution of optimal control problems is one of the largest present hurdles in the field of optimal control. The problem of finding optimal open-loop solutions numerically has been solved by a number of different methods, each of which give excellent results for a wide variety of non-trivial problems.

However, finding optimal feedback controls for general non-linear problems is an extremely difficult task. For the restricted class of problems with a linear dynamical system and quadratic performance index a genuinely optimal feedback control can be derived [1, 12], but this is not true for non-linear systems in general. This type of control is desirable because it produces the optimal trajectory from an arbitrary initial condition in some set, and is robust in the presence of system noise and parameter variation. However, despite the fact that the existence of such feedback controls can be proven for restricted cases, the only analytic method for finding optimal feedback control, the Hamilton-Jacobi-Bellman (HJB) equation, is analytically unsolvable for all but trivial problems. Unfortunately, while the HJB equation can be solved numerically, it suffers from the curse of dimensionality, which renders a numerical solution impractical for all but small problems.

This dissertation presents two methods for finding optimal feedback controls. The first, called perturbation feedback control, is a well-known practical alternative to optimal open-loop control. Perturbation feedback control is designed to exert local optimal feedback control on trajectories close to a previously computed optimal open-loop trajectory. The second is a new method presently being researched in the Mathematics Department at U.W.A. This method uses a backpropagation neural network as the optimal feedback control function. The resulting optimal feedback control is capable of supplying optimal control over an arbitrary set of initial conditions. Two methods for obtaining this artificial neural network optimal feedback control are presented.

Section 2 provides the necessary notation, problem formulations and optimal control theory results used throughout this dissertation. After perturbation feedback control and neural network feedback control are discussed in Sections 3 and 4, some highly non-linear optimal control problems are solved by these methods in Section 5. Finally, the Appendix provides a brief summary of the artificial neural network notation, gradient formulae and important theorems that will be used in subsequent sections of this dissertation.

2 Optimal Control: Notation and Important Results

In this section, the intention is not to furnish a brief exposition of the theory of optimal control (for this see [5] and [17]), but rather to lay down the necessary definitions, notation and problem formulations for ease of reference subsequently.

In general terms, the field of optimal control is concerned with optimising a performance index subject to or constrained by a dynamical system. Typically, some characteristics of the system are under direct control, while others change in response to these controlled characteristics according to the nature of the dynamical system. These are called respectively the controls, conventionally denoted by the vector $\mathbf{u}$, and the states, denoted by the vector $\mathbf{x}$, of the dynamical system.

Optimal control can be defined in two ways. The first, optimal open-loop control, is so called because the optimal control is a function of time only. This is essentially a predetermined control, which is not affected by changes in the state of the dynamical system. The second, optimal feedback control, is so called because the optimal control is a function of both the current state and the time. This means that the optimal control is being constantly re-evaluated as the states of the dynamical system change at each stage. In addition, two types of dynamical systems must be dealt with, those with discrete time steps and those in which time is considered as a continuous independent variable. The resulting optimal control problems are different, and must be treated separately. Each of these four types of optimal control problems will now be defined.

2.1 Optimal Open-Loop Control

2.1.1 Discrete Systems

For a discrete dynamical system, we define the problem $(\mathrm{P_D})$ as our standard discrete optimal control problem:

$$\min_{\mathbf{u}(\bullet)} \quad J \quad = \quad \Phi(\mathbf{x}(M)) + \sum_{i=0}^{M-1} L(\mathbf{x}(i), \mathbf{u}(i), i), \tag{2.1}$$

$$\text{such that} \quad \mathbf{x}(k+1) \quad = \quad f(\mathbf{x}(k), \mathbf{u}(k), k), \qquad k = 0, \ldots, M-1, \tag{2.2}$$

$$\mathbf{x}(0) \quad = \quad \mathbf{x}_0, \tag{2.3}$$

where $\mathbf{x}(k) \in \mathbb{R}^n, k = 0, \ldots, M$, $\mathbf{u}(k) \in \mathbb{R}^m, k = 0, \ldots, M-1$ and $\mathbf{x}_0 \in \Gamma$, where Γ is some bounded set of initial conditions. We require the function Φ to be continuously differentiable with respect to its argument, and f and L to be continuously differentiable with respect to $\mathbf{x}(k)$ and $\mathbf{u}(k)$.

This dissertation will not consider optimal control problems with arbitrary constraints. If necessary these may be included in the performance index as penalty functions. Nevertheless control bounds may be imposed. We now define the class of admissible controls $\mathcal{U}$. If no control bounds are imposed, $\mathcal{U}$ is the set of functions from $\mathbb{N}$ to $\mathbb{R}^m$. However, if the control is constrained to lie in some compact subset U of $\mathbb{R}^m$,

$$U = \{ \mathbf{v} \in \mathbb{R}^m : \alpha_i \leq \mathbf{v}_i \leq \beta_i, \, i = 1, \ldots, m \}, \tag{2.4}$$

then we define the class of admissible controls $\mathcal{U}$ by

$$\mathcal{U} = \{ \mathbf{u}(\bullet) : \mathbf{u}(k) \in U, \, k = 0, \ldots, M-1 \}. \tag{2.5}$$

Once a control $\mathbf{u}(\bullet) \in \mathcal{U}$ is decided upon, the states $\mathbf{x}(k)$ and hence the performance index J are completely determined. Thus we can optimise our choice of $\mathbf{u}(\bullet)$ such that J is minimised.

An important and frequently used result for discrete optimal control problems, stated here without proof, is the first order necessary conditions for a local minimum.

Theorem 2.1 (First Order Necessary Condition: Discrete System) *Let* $\mathbf{u}^* \in \mathcal{U}$ *be a local minimum control for the problem* $(\mathrm{P_D})$ *with the control constrained to lie in the set* U *and let* $\mathbf{x}^*$ *and* $\boldsymbol{\lambda}^*$ *be the corresponding optimal state and costate respectively. Define*

$$H(\mathbf{x}(k), \mathbf{u}(k), \boldsymbol{\lambda}(k+1), k) = L(\mathbf{x}(k), \mathbf{u}(k), k) + \boldsymbol{\lambda}^\top(k+1) f(\mathbf{x}(k), \mathbf{u}(k), k). \tag{2.6}$$

Then it is necessary that

$$\mathbf{x}^*(k+1) = f(\mathbf{x}^*(k), \mathbf{u}^*(k), k), \quad k = 0, \ldots, M, \tag{2.7}$$

$$\mathbf{x}^*(0) = \mathbf{x}_0, \tag{2.8}$$

$$\boldsymbol{\lambda}^*(k) = \left[\frac{\partial H(\mathbf{x}^*(k), \mathbf{u}^*(k), \boldsymbol{\lambda}^*(k+1), k)}{\partial \mathbf{x}(k)} \right]^\top, \quad k = 0, \ldots, M, \tag{2.9}$$

$$\boldsymbol{\lambda}^*(M) = \left[\frac{\partial \Phi(\mathbf{x}^*(M))}{\partial \mathbf{x}(M)} \right]^\top, \tag{2.10}$$

$$H(\mathbf{x}^*(k), \mathbf{u}^*(k), \boldsymbol{\lambda}^*(k+1), k) = \min_{\mathbf{v} \in U} H(\mathbf{x}^*(k), \mathbf{v}, \boldsymbol{\lambda}^*(k+1), k), \quad k = 0, \ldots, M-1. \tag{2.11}$$

This theorem is essentially a statement of the Pontryagin Minimum Principle for the problem ($\mathrm{P_D}$). If the control is unconstrained, then the necessary condition (2.11) becomes

$$\frac{\partial H(\mathbf{x}^*(k), \mathbf{u}^*(k), \boldsymbol{\lambda}^*(k+1), k)}{\partial \mathbf{u}(k)} = 0, \quad k = 0, \ldots, M-1. \tag{2.12}$$

2.1.2 Continuous Systems

For a continuous dynamical system, the standard optimal control problem ($\mathrm{P_C}$) is defined as follows:

$$\min_{\mathbf{u}(\bullet)} \quad J = \Phi(\mathbf{x}(T)) + \int_0^T L(\mathbf{x}(t), \mathbf{u}(t), t)dt, \tag{2.13}$$

$$\text{such that} \quad \dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t), t), \quad t \in [0, T], \tag{2.14}$$

$$\mathbf{x}(0) = \mathbf{x}_0, \tag{2.15}$$

where $\mathbf{x}(t) \in \mathbb{R}^n$, $\mathbf{u}(t) \in \mathbb{R}^m$, $t \in [0, T]$ and $\mathbf{x}_0 \in \Gamma$, where Γ is some bounded set of initial conditions. We require each of the functions Φ, L and f to be continuously differentiable in each of their respective arguments. The assumption that the process has initial time 0 and terminal time T may be made without loss of generality since if the process had initial time $t_s \neq 0$ then this could be transformed into the desired form by a suitable shift in the time axis.

In the absence of control bounds, we define the set of admissible controls $\mathcal{U}$ to be the set of piecewise continuous functions from $\mathbb{R}$ to $\mathbb{R}^m$. If the control is constrained to lie in the compact set $U \subseteq \mathbb{R}^m$ for all $t \in [0, T]$ defined by (2.4), then the class of admissible controls is defined by

$$\mathcal{U} = \{\, \mathbf{u}(\bullet) : \mathbf{u}(t) \in U, t \in [0, T] \,\}, \tag{2.16}$$

defining a more restricted class of admissible controls.

Once an admissible control $\mathbf{u}(\bullet)$ has been decided upon, the state $\mathbf{x}$ is completely determined. Thus the performance index J is known and we can choose the control $\mathbf{u} \in \mathcal{U}$ so that J is minimised.

An important result for continuous optimal control problems, analogous with discrete optimal control problems and again stated without proof, is the first order necessary conditions for a local minimum.

Theorem 2.2 (First Order Necessary Condition: Continuous System) *Let* $\mathbf{u}^* \in \mathcal{U}$ *be a local minimum control for the problem* ($\mathrm{P_C}$) *with the control constrained to lie in the set* U *and let* $\mathbf{x}^*$ *and* $\boldsymbol{\lambda}^*$ *be the corresponding optimal state and costate respectively. Define*

$$H(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, t) = L(\mathbf{x}, \mathbf{u}, t) + \boldsymbol{\lambda}^\top f(\mathbf{x}, \mathbf{u}, t). \tag{2.17}$$

Then it is necessary that

$$\dot{\mathbf{x}}^* = f(\mathbf{x}^*(t), \mathbf{u}^*(t), t), \quad t \in [0, T] \tag{2.18}$$

$$\mathbf{x}^*(0) = \mathbf{x}_0, \tag{2.19}$$

$$\dot{\boldsymbol{\lambda}}^* = \left[-\frac{\partial H(\mathbf{x}^*, \mathbf{u}^*, \boldsymbol{\lambda}^*, t)}{\partial \mathbf{x}}\right]^\top, \quad t \in [0, T] \tag{2.20}$$

$$\boldsymbol{\lambda}^*(T) = \left[\frac{\partial \Phi(\mathbf{x}^*(T))}{\partial \mathbf{x}(T)}\right]^\top, \tag{2.21}$$

$$H(\mathbf{x}^*(t), \mathbf{u}^*(t), \boldsymbol{\lambda}^*(t), t) = \min_{\mathbf{v} \in U} H(\mathbf{x}^*(t), \mathbf{v}, \boldsymbol{\lambda}^*(t), t), \quad t \in [0, T]. \tag{2.22}$$

Again, this is essentially a statement of the Pontryagin Minimum Principle for the problem ($\mathrm{P_C}$). If the control is unconstrained, then the necessary condition (2.22) becomes

$$\frac{\partial H(\mathbf{x}^*, \mathbf{u}^*, \boldsymbol{\lambda}^*, t)}{\partial \mathbf{u}} = 0, \quad t \in [0, T]. \tag{2.23}$$

2.2 Optimal Feedback Control

Section 2.1 gave necessary conditions for $\mathbf{u}^*$ to be a local optimal control for either discrete or continuous systems. This, however, only defines $\mathbf{u}^*$ as a function of t, written $\mathbf{u}^*(t)$. A control of this form is only optimal for the specific problem that was solved to compute it. If the initial condition in ($\mathrm{P_D}$) or ($\mathrm{P_C}$) changes, then $\mathbf{u}^*(t)$ is no longer optimal. Similarly, if at some time τ the state deviates from the optimal state trajectory $\mathbf{x}^*(t)$, then the control $\mathbf{u}^*(\tau)$ will no longer be optimal. Ideally, what is needed is a control that is dependent not only on time, but also on the state. A control such as this is called feedback control.

In realistic problems, there is often uncertainty in the exact initial condition of the system. Since it is dependent on the state of the system, optimal feedback control can provide the optimal control regardless of the initial condition. In addition, for realistic problems, there is often noise in the actuators that translate the theoretical control into actual motion. This causes the state to deviate from the theoretical optimal trajectory. Optimal feedback control can compensate and supply the new optimal control.

In addition to these properties, optimal feedback control inherits a number of useful properties from general feedback control, such as superior performance in the presence of parameter variation and inaccurate system dynamics. All of these properties make optimal feedback control a much sought-after type of control.

2.2.1 Discrete Systems

In order to formally define optimal feedback control, we make the following definition for $j = 0, \ldots, M-1$.

Problem $(\mathrm{P}_\mathrm{D}^j)$ Find the optimal control $\mathbf{u}(k)$, $k = j, \ldots, M-1$ for the optimal control problem

$$\min_{\mathbf{u}(k)} J(\mathbf{x}_j, \mathbf{u}) = \Phi(\mathbf{x}(M)) + \sum_{i=j}^{M-1} L(\mathbf{x}(i), \mathbf{u}(i), i), \tag{2.24}$$

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k), k), \qquad k = j, \ldots, M-1, \tag{2.25}$$

$$\mathbf{x}(j) = \mathbf{x}_j, \tag{2.26}$$

where the initial condition $\mathbf{x}_j \in \Xi_j$. The set Ξ_j is defined to be the set of states $\mathbf{x}(j)$ reachable at time j, by using a sequence of admissible controls $\mathbf{u}(k)$, $k = 0, \ldots, j-1$.

Note that the problem $(\mathrm{P}_\mathrm{D}^0)$ is equivalent to the problem (P_D). The optimal feedback controller $\psi : \mathbb{R}^n \times \mathbb{N} \to \mathbb{R}^m$ can now be defined. If $\mathbf{u}(j)$ is the optimal open-loop control at time-step j for the problem $(\mathrm{P}_\mathrm{D}^j)$ with initial condition $\mathbf{x}(j)$, then $\psi(\mathbf{x}(j), j) = \mathbf{u}(j)$. Note that a strictly state dependent optimal feedback control $\psi : \mathbb{R}^n \to \mathbb{R}^m$ can be similarly defined. A strictly state dependent optimal feedback controller is generally preferred, as this type of controller is unaffected by shifts and scalings of the time axis. For some simple problems, this appears to be possible. However, for many problems, it is not possible to find a strictly state dependent optimal feedback controller, and for these we allow time as a secondary argument. Both types of optimal feedback controller are used in this dissertation, so the more general notation of the state and time dependent controller will be used throughout. If a strictly state dependent feedback controller is required, then the time argument is considered to take no part in the calculation of the optimal feedback control.

Proposition 2.1 *If* $\mathbf{u}^*$ *and* $\mathbf{x}^*$ *are respectively the optimal open-loop control and state for the problem* (P_D) *with arbitrary initial condition in* Γ*, then*

$$\psi(\mathbf{x}^*(k), k) = \mathbf{u}^*(k), \quad k = 0, \ldots, M-1. \tag{2.27}$$

Proof Suppose at some time step $\hat{k}$,

$$\psi(\mathbf{x}^*(\hat{k}), \hat{k}) = \hat{\mathbf{u}} \neq \mathbf{u}^*(\hat{k}). \tag{2.28}$$

Then, by the definition of ψ, the problem $(\mathrm{P}_{\mathrm{D}}^{\hat{k}})$ with initial state $\mathbf{x}^*(\hat{k})$ will yield an optimal trajectory $\hat{\mathbf{x}}^*(k), k = \hat{k}, \ldots, M$ with performance index $J(\mathbf{x}^*(\hat{k}), \hat{\mathbf{u}}^*)$, where $\hat{\mathbf{u}}^*$ is defined by $\hat{\mathbf{u}}^*(k) = \hat{\mathbf{u}}$ for $k = \hat{k}$ and $\hat{\mathbf{u}}^*(k) = \mathbf{u}^*(k)$ for $\hat{k} < k \leq M - 1$. Then $J(\mathbf{x}^*(\hat{k}), \hat{\mathbf{u}}^*) < J(\mathbf{x}^*(\hat{k}), \mathbf{u}^*)$ by the definition of the optimal feedback control ψ. Hence the trajectory $\mathbf{x}^*(k)$ for $k = 0, \ldots, \hat{k} - 1$, $\hat{\mathbf{x}}^*(k)$ for $k = \hat{k}, \ldots, M$ yields a smaller objective function than the trajectory $\mathbf{x}^*(k)$ for $k = 0, \ldots, M$, resulting in a contradiction. ∎

By Proposition 2.1, if the optimal open-loop control $\mathbf{u}(k)$ for the problem $(\mathrm{P}_{\mathrm{D}})$ is replaced by the feedback control function $\psi(\mathbf{x}(k), k)$, the sequence of states $\mathbf{x}(k)$ generated from a fixed initial condition will be identical to those generated by the open-loop control $\mathbf{u}(k)$. As a consequence, the performance index J resulting from the optimal open-loop and feedback controls will be identical.

2.2.2 Continuous Systems

Here we formally define optimal feedback control for the problem $(\mathrm{P}_{\mathrm{C}})$. In a similar way to Section 2.2.1, we define a series of problems, for $\tau \in [0, T]$

Problem $(\mathrm{P}_{\mathrm{C}}^{\tau})$ Find the optimal control $\mathbf{u}(t), t \in [\tau, T)$ for the optimal control problem

$$\min_{\mathbf{u}(\bullet)} J(\mathbf{x}_\tau, \mathbf{u}) = \Phi(\mathbf{x}(T)) + \int_\tau^T L(\mathbf{x}(t), \mathbf{u}(t), t)dt, \tag{2.29}$$

$$\dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t), t), \quad t \in [\tau, T], \tag{2.30}$$

$$\mathbf{x}(\tau) = \mathbf{x}_\tau, \tag{2.31}$$

where the initial condition $\mathbf{x}_\tau \in \Xi_\tau$. The set Ξ_τ is defined, in a similar way to Section 2.2.1, as the set of states $\mathbf{x}(\tau)$ reachable at time τ by using an admissible control $\mathbf{u}(t)$ for $t \in [0, \tau)$.

As before, the problem $(\mathrm{P}_{\mathrm{C}}^{0})$ is equivalent to the problem $(\mathrm{P}_{\mathrm{C}})$. The optimal feedback controller $\psi : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^m$ can now be defined. If $\mathbf{u}(\tau)$ is the optimal open-loop control at time τ for the problem $(\mathrm{P}_{\mathrm{C}}^{\tau})$ with initial condition $\mathbf{x}(\tau)$, then $\psi(\mathbf{x}(\tau), \tau) = \mathbf{u}(\tau)$.

Proposition 2.2 *If $\mathbf{u}^*$ and $\mathbf{x}^*$ are respectively the optimal open-loop control and state for the problem $(\mathrm{P}_{\mathrm{C}})$ with arbitrary initial condition in Γ, then*

$$\psi(\mathbf{x}^*(t), t) = \mathbf{u}^*(t), \quad t \in [0, T). \tag{2.32}$$

Proof Suppose at some intermediate time $\hat{t}$,

$$\psi(\mathbf{x}^*(\hat{t}), \hat{t}) = \hat{\mathbf{u}} \neq \mathbf{u}^*(\hat{t}). \tag{2.33}$$

Then, by the definition of ψ, the problem $(\mathrm{P}_\mathrm{C}^{\hat{t}})$ with initial state $\mathbf{x}^*(\hat{t})$ will yield an optimal trajectory $\hat{\mathbf{x}}^*(t), t \in [\hat{t}, T]$ with performance index $J(\mathbf{x}^*(\hat{t}), \hat{\mathbf{u}}^*)$, where $\hat{\mathbf{u}}^*$ is defined by $\hat{\mathbf{u}}^*(t) = \hat{\mathbf{u}}$, $t = \hat{t}$ and $\hat{\mathbf{u}}^*(t) = \mathbf{u}^*(t)$, $t \in (\hat{t}, T)$. Then $J(\mathbf{x}^*(\hat{t}), \hat{\mathbf{u}}^*) < J(\mathbf{x}^*(\hat{t}), \mathbf{u}^*)$ by the definition of the optimal feedback control ψ. Hence the trajectory $\mathbf{x}^*(t)$, $t \in [0, \hat{t})$, $\hat{\mathbf{x}}^*(t)$, $t \in [\hat{t}, T]$ yields a smaller objective function than the trajectory $\mathbf{x}^*(t)$, $t \in [0, T]$, resulting in a contradiction. ∎

Then substituting $\psi(\mathbf{x}(t), t)$ for $\mathbf{u}(t)$ in the problem (P_C) will generate the same state trajectory $\mathbf{x}(\mathrm{t})$, for a given initial condition, as the open-loop control. Thus the performance index J will also be the same.

For optimal control problems over continuous dynamical systems, the HJB equation provides an analytic method for finding optimal feedback control for the problem (P_C). It is presented here as a theorem.

Theorem 2.3 (Hamilton-Jacobi-Bellman Equation) *Define the value function V by*

$$V(\mathbf{x}_\tau, \tau) \quad = \quad \min_{\mathbf{u}(\bullet)}\{J(\mathbf{x}_\tau, \mathbf{u}) \textit{ for the problem } (\mathrm{P}_\mathrm{C}^{\tau})\}. \tag{2.34}$$

Then the HJB equation,

$$-\frac{\partial V(\mathbf{x}_\tau, \tau)}{\partial \tau} \quad = \quad \min_{\mathbf{v} \in U}\left\{\frac{\partial V(\mathbf{x}_\tau, \tau)}{\partial \mathbf{x}_\tau} f(\mathbf{x}_\tau, \mathbf{v}, \tau) + L(\mathbf{x}_\tau, \mathbf{v}, \tau)\right\}, \tag{2.35}$$

together with the boundary condition,

$$V(\mathbf{x}_T, T) \quad = \quad \Phi(\mathbf{x}_T), \tag{2.36}$$

govern the value function V for the problem (P_C). *The control* $\mathbf{v}$ *that supplies the minimum in* (2.35) *is dependent on the state, and is the optimal feedback control for the problem* (P_C).

In practice, as noted earlier, the HJB equation suffers from the curse of dimensionality. For most practical and even most contrived problems, the HJB equation is very difficult to solve. However, the HJB equation enables us to search for optimal feedback controls with confidence, as it demonstrates the existence and theoretical solution of this type of control.

Despite these theoretical results, the harsh reality is that for most practical problems the necessary conditions of Theorem 2.1 and Theorem 2.2 are difficult to solve analytically.

The open-loop first order necessary conditions for a continuous system in Theorem 2.2 require the simultaneous solution of $2n$ differential equations and m algebraic equations where $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{u} \in \mathbb{R}^m$. This is, in general, a non-trivial task. Nevertheless, the open-loop problem can now be solved numerically with recent advances in sophisticated computational algorithms. However, the situation for optimal feedback control is less promising. For continuous systems one can attempt to solve the HJB partial differential equation of Theorem 2.3. This is often a formidable exercise, even numerically. Similarly, for discrete systems the optimal control is no longer finite dimensional, as optimal feedback control is from an infinite dimensional function space. Two methods of synthesising optimal feedback control are now presented. The first, perturbation feedback control, is discussed in Section 3 and the second, neural network optimal feedback control, is discussed in Section 4.

3 Perturbation Feedback Control

Perturbation feedback control has found many applications in real world problems. A detailed exposition of perturbation feedback control and its applications can be found in [4, 5]. The basic idea is that perturbation feedback control "tracks" or "follows" a previously computed optimal open-loop trajectory. As can be seen from the following derivations, perturbation feedback control will produce excellent suboptimal trajectories as long as they do not deviate too far from the original optimal open-loop trajectory. Thus, perturbation feedback control cannot be used to provide optimal control over a large subset of the state space. Rather it is used to add robustness and error tolerance properties to existing optimal open-loop control.

3.1 Discrete Systems

Consider a control $\mathbf{u}$, and corresponding state $\mathbf{x}$ and costate $\boldsymbol{\lambda}$, which satisfy the conditions for a local minimum of Theorem 2.1 with respect to the standard problem formulation ($\mathrm{P_D}$) without control bound. Then, as before, the performance index is

$$J = \Phi(\mathbf{x}(M)) + \sum_{i=0}^{M-1} L(\mathbf{x}(i), \mathbf{u}(i), i). \tag{3.1}$$

The augmented performance index can now be defined:

$$\bar{J} = \Phi(\mathbf{x}(M)) + \sum_{i=0}^{M-1} \Big[H(\mathbf{x}(i), \mathbf{u}(i), i) - \boldsymbol{\lambda}^\top(i+1) f(\mathbf{x}(i), \mathbf{u}(i), i)\Big], \tag{3.2}$$

where H is defined as before:

$$H(\mathbf{x}(i), \mathbf{u}(i), \boldsymbol{\lambda}(i+1), i) = L(\mathbf{x}(i), \mathbf{u}(i), i) + \boldsymbol{\lambda}^\top(i+1) f(\mathbf{x}(i), \mathbf{u}(i), i). \tag{3.3}$$

Then $J = \bar{J}$ by the necessary condition (2.7) of Theorem 2.1.

Consider now the effect of a small perturbation $\Delta\mathbf{x}_0$ in the initial state. This will produce perturbations $\Delta\mathbf{x}(k)$ and $\Delta\boldsymbol{\lambda}(k)$ for $k = 0, \ldots, M$, and $\Delta\mathbf{u}(k)$ for $k = 0, \ldots, M-1$ in the state, costate and optimal control respectively. Next, consider the expansion of the augmented performance index $\bar{J}$ to second order (since the first order expansion disappears about a trajectory satisfying the first order necessary conditions) and the expansion of the system dynamics to first order, thus:

$$
\begin{aligned}
d^2\bar{J} &= \frac{1}{2}\left[(\Delta\mathbf{x}(M))^\top \Phi_{\mathbf{xx}}(\mathbf{x}(M))\Delta\mathbf{x}(M)\right] \\
&\qquad + \frac{1}{2}\sum_{i=0}^{M-1}\left[\begin{array}{cc}(\Delta\mathbf{x}(i))^\top & (\Delta\mathbf{u}(i))^\top\end{array}\right]\left[\begin{array}{cc}H^i_{\mathbf{xx}} & H^i_{\mathbf{xu}} \\ H^i_{\mathbf{ux}} & H^i_{\mathbf{uu}}\end{array}\right]\left[\begin{array}{c}\Delta\mathbf{x}(i) \\ \Delta\mathbf{u}(i)\end{array}\right], \quad (3.4)
\end{aligned}
$$

subject to

$$
\begin{aligned}
\Delta\mathbf{x}(i+1) &= f^i_{\mathbf{x}}\Delta\mathbf{x}(i) + f^i_{\mathbf{u}}\Delta\mathbf{u}(i), \qquad i = 0,\ldots,M-1, & (3.5)\\
\Delta\mathbf{x}(0) &= \Delta\mathbf{x}_0, & (3.6)
\end{aligned}
$$

where

$$
f^i_{\mathbf{x}} = \frac{\partial f(\mathbf{x}(i),\mathbf{u}(i),i)}{\partial\mathbf{x}(i)}, \qquad H^i_{\mathbf{xu}} = \frac{\partial H(\mathbf{x}(i),\mathbf{u}(i),\boldsymbol{\lambda}(i+1),i)}{\partial\mathbf{x}(i)\partial\mathbf{u}(i)}, \quad \text{etc.}
$$

Since the optimal solution for this neighboring trajectory is desired, the change in the optimal control $\Delta\mathbf{u}(i)$ for $i = 0,\ldots,M-1$ must be found so that $d^2\bar{J}$ is minimised. Note that this is a new optimal control problem in $\Delta\mathbf{u}(k)$ and $\Delta\mathbf{x}(k)$. The first order necessary conditions of Theorem 2.1 are applied, resulting in the equations

$$
\begin{aligned}
\Delta\boldsymbol{\lambda}(i) &= H^i_{\mathbf{xx}}\Delta\mathbf{x}(i) + H^i_{\mathbf{xu}}\Delta\mathbf{u}(i) + (f^i_{\mathbf{x}})^\top\Delta\boldsymbol{\lambda}(i+1), \qquad i = 0,\ldots,M-1, & (3.7)\\
\Delta\boldsymbol{\lambda}(M) &= \Phi_{\mathbf{xx}}\Delta\mathbf{x}(M), & (3.8)\\
0 &= H^i_{\mathbf{uu}}\Delta\mathbf{u}(i) + H^i_{\mathbf{ux}}\Delta\mathbf{x}(i) + (f^i_{\mathbf{u}})^\top\Delta\boldsymbol{\lambda}(i+1), \qquad i = 0,\ldots,M-1. & (3.9)
\end{aligned}
$$

Due to the linear-quadratic nature of the problem, the state and costate are assumed to have a linear relationship:

$$
\Delta\boldsymbol{\lambda}(i) = S(i)\Delta\mathbf{x}(i). \qquad (3.10)
$$

Note the similarity here with the derivation of the matrix Ricatti equation. The consequences of this linear relationship are now examined.

Firstly, $k+1$ is substituted for i in (3.10), and (3.5) is substituted for $\Delta\mathbf{x}(k+1)$ in the resulting expression:

$$
\begin{aligned}
\Delta\boldsymbol{\lambda}(k+1) &= S(k+1)\Delta\mathbf{x}(k+1) \\
&= S(k+1)\left(f^k_{\mathbf{x}}\Delta\mathbf{x}(k) + f^k_{\mathbf{u}}\Delta\mathbf{u}(k)\right). \qquad (3.11)
\end{aligned}
$$

Next, k is replaced by i in (3.11) and this expression for $\Delta\boldsymbol{\lambda}(i+1)$ is substituted into (3.7):

$$
\begin{aligned}
\Delta\boldsymbol{\lambda}(i) &= H^i_{\mathbf{xx}}\Delta\mathbf{x}(i) + H^i_{\mathbf{xu}}\Delta\mathbf{u}(i) + (f^i_{\mathbf{x}})^\top\left[S(i+1)\left(f^i_{\mathbf{x}}\Delta\mathbf{x}(i) + f^i_{\mathbf{u}}\Delta\mathbf{u}(i)\right)\right] \\
&= \left(H^i_{\mathbf{xx}} + (f^i_{\mathbf{x}})^\top S(i+1)f^i_{\mathbf{x}}\right)\Delta\mathbf{x}(i) + \left(H^i_{\mathbf{xu}} + (f^i_{\mathbf{x}})^\top S(i+1)f^i_{\mathbf{u}}\right)\Delta\mathbf{u}(i). \qquad (3.12)
\end{aligned}
$$

Similarly, this expression for $\Delta\boldsymbol{\lambda}(i+1)$ is substituted into (3.9):

$$\begin{aligned} 0 &= H^i_{\mathbf{uu}}\Delta\mathbf{u}(i) + H^i_{\mathbf{ux}}\Delta\mathbf{x}(i) + (f^i_{\mathbf{u}})^\top \left[S(i+1)\left(f^i_{\mathbf{x}}\Delta\mathbf{x}(i) + f^i_{\mathbf{u}}\Delta\mathbf{u}(i)\right)\right] \\ &= \left(H^i_{\mathbf{ux}} + (f^i_{\mathbf{u}})^\top S(i+1) f^i_{\mathbf{x}}\right)\Delta\mathbf{x}(i) + \left(H^i_{\mathbf{uu}} + (f^i_{\mathbf{u}})^\top S(i+1) f^i_{\mathbf{u}}\right)\Delta\mathbf{u}(i). \end{aligned} \tag{3.13}$$

Assuming the appropriate non-singularity properties for the coefficient matrix of $\Delta\mathbf{u}(i)$ are satisfied, an expression for $\Delta\mathbf{u}(i)$ in terms of $\Delta\mathbf{x}(i)$ can now be derived:

$$\Delta\mathbf{u}(i) = -\left(H^i_{\mathbf{uu}} + (f^i_{\mathbf{u}})^\top S(i+1) f^i_{\mathbf{u}}\right)^{-1}\left(H^i_{\mathbf{ux}} + (f^i_{\mathbf{u}})^\top S(i+1) f^i_{\mathbf{x}}\right)\Delta\mathbf{x}(i). \tag{3.14}$$

This is substituted back into (3.12) to derive an expression for $\Delta\boldsymbol{\lambda}(i)$ in terms of $\Delta\mathbf{x}(i)$:

$$\begin{aligned} \Delta\boldsymbol{\lambda}(i) &= \left(H^i_{\mathbf{xx}} + (f^i_{\mathbf{x}})^\top S(i+1) f^i_{\mathbf{x}}\right)\Delta\mathbf{x}(i) - \left(H^i_{\mathbf{xu}} + (f^i_{\mathbf{x}})^\top S(i+1) f^i_{\mathbf{u}}\right)\times \\ &\qquad \left(H^i_{\mathbf{uu}} + (f^i_{\mathbf{u}})^\top S(i+1) f^i_{\mathbf{u}}\right)^{-1}\left(H^i_{\mathbf{ux}} + (f^i_{\mathbf{u}})^\top S(i+1) f^i_{\mathbf{x}}\right)\Delta\mathbf{x}(i) \\ &= \Big[\left(H^i_{\mathbf{xx}} + (f^i_{\mathbf{x}})^\top S(i+1) f^i_{\mathbf{x}}\right) - \left(H^i_{\mathbf{xu}} + (f^i_{\mathbf{x}})^\top S(i+1) f^i_{\mathbf{u}}\right)\times \\ &\qquad \left(H^i_{\mathbf{uu}} + (f^i_{\mathbf{u}})^\top S(i+1) f^i_{\mathbf{u}}\right)^{-1}\left(H^i_{\mathbf{ux}} + (f^i_{\mathbf{u}})^\top S(i+1) f^i_{\mathbf{x}}\right)\Big]\Delta\mathbf{x}(i). \end{aligned} \tag{3.15}$$

Now, by the linearity assumption for $\Delta\boldsymbol{\lambda}(i)$ and $\Delta\mathbf{x}(i)$ in (3.10), the matrix coefficients in (3.10) and (3.15) must be equal:

$$\begin{aligned} S(i) &= H^i_{\mathbf{xx}} + (f^i_{\mathbf{x}})^\top S(i+1) f^i_{\mathbf{x}} - \left(H^i_{\mathbf{xu}} + (f^i_{\mathbf{x}})^\top S(i+1) f^i_{\mathbf{u}}\right)\times \\ &\qquad \left(H^i_{\mathbf{uu}} + (f^i_{\mathbf{u}})^\top S(i+1) f^i_{\mathbf{u}}\right)^{-1}\left(H^i_{\mathbf{ux}} + (f^i_{\mathbf{u}})^\top S(i+1) f^i_{\mathbf{x}}\right). \end{aligned} \tag{3.16}$$

This difference equation, and the boundary condition obtained from (3.8),

$$S(M) = \Phi_{\mathbf{xx}}(\mathbf{x}(M)), \tag{3.17}$$

define $S(i)$ for $i = 0, \ldots, M$.

Now, given a predetermined optimal open-loop control $\mathbf{u}(i)$ for $i = 0, \ldots, M-1$ and corresponding state $\mathbf{x}(i)$ for $i = 0, \ldots, M$, the change in control $\Delta\mathbf{u}(i)$ can be calculated for a given error in the state trajectory $\Delta\mathbf{x}(i)$ via (3.14),(3.16) and (3.17). Note that the change in the control $\Delta\mathbf{u}(i)$ in (3.14) is in terms of $\Delta\mathbf{x}(i)$, thus furnishing an optimal perturbation feedback control.

3.2 Continuous Systems

In a similar way to Section 3.1, consider a control $\mathbf{u}$, and corresponding state $\mathbf{x}$ and costate $\boldsymbol{\lambda}$ which satisfy the necessary conditions for a local minimum of Theorem 2.2 with respect to the standard problem formulation ($\mathrm{P_C}$) for continuous time without control bounds. Recall the performance index

$$J = \Phi(\mathbf{x}(T)) + \int_0^T L(\mathbf{x}(t), \mathbf{u}(t), t)dt. \tag{3.18}$$

As in Section 3.1, the augmented performance index is defined as

$$\bar{J} = \Phi(\mathbf{x}(T)) + \int_0^T \left[H(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, t) - \boldsymbol{\lambda}^\top f(\mathbf{x}, \mathbf{u}, t)\right] dt, \tag{3.19}$$

where

$$H(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, t) = L(\mathbf{x}, \mathbf{u}, t) + \boldsymbol{\lambda}^\top f(\mathbf{x}, \mathbf{u}, t). \tag{3.20}$$

The effect of a small perturbation $\Delta\mathbf{x}_0$ in the initial state is now considered. This will produce corresponding perturbations $\delta\mathbf{x}$, $\delta\boldsymbol{\lambda}$ and $\delta\mathbf{u}$ in the state, costate and optimal control respectively. As before, the expansion of the augmented performance index $\bar{J}$ is taken to second order and the system dynamics are expanded to first order, resulting in a new problem in $\delta\mathbf{x}$ and $\delta\mathbf{u}$:

$$\begin{aligned}\delta^2\bar{J} &= \frac{1}{2}\left[(\delta\mathbf{x}(T))^\top \Phi_{\mathbf{xx}}(\mathbf{x}(T))\delta\mathbf{x}(T)\right] \\ &\quad + \frac{1}{2}\int_0^T \begin{bmatrix} \delta\mathbf{x}^\top & \delta\mathbf{u}^\top \end{bmatrix} \begin{bmatrix} H_{\mathbf{xx}} & H_{\mathbf{xu}} \\ H_{\mathbf{ux}} & H_{\mathbf{uu}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix} dt,\end{aligned} \tag{3.21}$$

subject to

$$\delta\dot{\mathbf{x}} = f_{\mathbf{x}}\delta\mathbf{x} + f_{\mathbf{u}}\delta\mathbf{u}, \tag{3.22}$$

$$\delta\mathbf{x}(0) = \Delta\mathbf{x}_0. \tag{3.23}$$

The first order necessary conditions of Theorem 2.2 are applied, giving rise to the following equations:

$$\delta\dot{\boldsymbol{\lambda}} = -\left(H_{\mathbf{xx}}\delta\mathbf{x} + H_{\mathbf{xu}}\delta\mathbf{u} + f_{\mathbf{x}}^\top\delta\boldsymbol{\lambda}\right), \tag{3.24}$$

$$\delta\boldsymbol{\lambda}(T) = \Phi_{\mathbf{xx}}\delta\mathbf{x}(T), \tag{3.25}$$

$$0 = H_{\mathbf{uu}}\delta\mathbf{u} + H_{\mathbf{ux}}\delta\mathbf{x} + f_{\mathbf{u}}^\top\delta\boldsymbol{\lambda}. \tag{3.26}$$

From (3.26), assuming the appropriate non-singularity property for $H_{\mathbf{uu}}$, an expression for $\delta\mathbf{u}$ can now be derived:

$$\delta\mathbf{u} = -H_{\mathbf{uu}}^{-1}\left(H_{\mathbf{ux}}\delta\mathbf{x} + f_{\mathbf{u}}^\top\delta\boldsymbol{\lambda}\right). \tag{3.27}$$

This expression for $\delta\mathbf{u}$ is substituted into (3.22) to obtain

$$\begin{aligned}\delta\dot{\mathbf{x}} &= f_{\mathbf{x}}\delta\mathbf{x} - f_{\mathbf{u}}H_{\mathbf{uu}}^{-1}\left(H_{\mathbf{ux}}\delta\mathbf{x} + f_{\mathbf{u}}^{\top}\delta\boldsymbol{\lambda}\right) \\ &= \left(f_{\mathbf{x}} - f_{\mathbf{u}}H_{\mathbf{uu}}^{-1}H_{\mathbf{ux}}\right)\delta\mathbf{x} - \left(f_{\mathbf{u}}H_{\mathbf{uu}}^{-1}f_{\mathbf{u}}^{\top}\right)\delta\boldsymbol{\lambda}. \qquad (3.28)\end{aligned}$$

Similarly, this expression for $\delta\mathbf{u}$ is substituted into (3.24):

$$\begin{aligned}\delta\dot{\boldsymbol{\lambda}} &= -\left(H_{\mathbf{xx}}\delta\mathbf{x} - H_{\mathbf{xu}}H_{\mathbf{uu}}^{-1}\left(H_{\mathbf{ux}}\delta\mathbf{x} + f_{\mathbf{u}}^{\top}\delta\boldsymbol{\lambda}\right) + f_{\mathbf{x}}^{\top}\delta\boldsymbol{\lambda}\right) \\ &= -\left(H_{\mathbf{xx}} + H_{\mathbf{xu}}H_{\mathbf{uu}}^{-1}H_{\mathbf{ux}}\right)\delta\mathbf{x} - \left(f_{\mathbf{x}}^{\top} - H_{\mathbf{xu}}H_{\mathbf{uu}}^{-1}f_{\mathbf{u}}^{\top}\right)\delta\boldsymbol{\lambda}. \qquad (3.29)\end{aligned}$$

The equations (3.28) and (3.29) may now be written:

$$\delta\dot{\mathbf{x}} = A(t)\delta\mathbf{x} - B(t)\delta\boldsymbol{\lambda}, \qquad (3.30)$$

$$\delta\dot{\boldsymbol{\lambda}} = -C(t)\delta\mathbf{x} - A^{\top}(t)\delta\boldsymbol{\lambda}, \qquad (3.31)$$

where

$$A(t) = f_{\mathbf{x}} - f_{\mathbf{u}}H_{\mathbf{uu}}^{-1}H_{\mathbf{ux}}, \qquad (3.32)$$

$$B(t) = f_{\mathbf{u}}H_{\mathbf{uu}}^{-1}f_{\mathbf{u}}^{\top}, \qquad (3.33)$$

$$C(t) = H_{\mathbf{xx}} + H_{\mathbf{xu}}H_{\mathbf{uu}}^{-1}H_{\mathbf{ux}}. \qquad (3.34)$$

By noting the linear-quadratic nature of the problem, $\delta\mathbf{x}$ and $\delta\boldsymbol{\lambda}$ are assumed to have the standard linear relationship

$$\delta\boldsymbol{\lambda}(t) = S(t)\delta\mathbf{x}(t). \qquad (3.35)$$

Differentiating (3.35) with respect to t gives rise to

$$\delta\dot{\boldsymbol{\lambda}} = \dot{S}\delta\mathbf{x} + S\delta\dot{\mathbf{x}}. \qquad (3.36)$$

Next, the expression for $\delta\dot{\boldsymbol{\lambda}}$ in (3.31) is substituted into (3.36):

$$\dot{S}\delta\mathbf{x} + S\delta\dot{\mathbf{x}} = -C\delta\mathbf{x} - A^{\top}\delta\boldsymbol{\lambda}. \qquad (3.37)$$

Finally, the expression for $\delta\dot{\mathbf{x}}$ in (3.30) and the expression for $\delta\boldsymbol{\lambda}$ in (3.35) are substituted into (3.37) to obtain

$$\dot{S}\delta\mathbf{x} + S\left[A\delta\mathbf{x} - B\delta\boldsymbol{\lambda}\right] = -C\delta\mathbf{x} - A^{\top}\delta\boldsymbol{\lambda} \qquad (3.38)$$

$$\dot{S}\delta\mathbf{x} + S\left[A\delta\mathbf{x} - BS\delta\mathbf{x}\right] = -C\delta\mathbf{x} - A^{\top}S\delta\mathbf{x} \qquad (3.39)$$

$$\left[\dot{S} + SA + A^\top S - SBS + C\right] \delta\mathbf{x} \quad = \quad 0. \tag{3.40}$$

Since $\delta\mathbf{x}$ is arbitrary, we have the matrix Ricatti equation

$$\dot{S}(t) \quad = \quad -S(t)A(t) - A^\top(t)S(t) + S(t)B(t)S(t) - C(t). \tag{3.41}$$

The boundary condition for S is derived from (3.25) and (3.35):

$$S(T) \quad = \quad \Phi_{\mathbf{xx}}(\mathbf{x}(T)), \tag{3.42}$$

and the perturbation feedback control law can be derived from (3.27) and (3.35),

$$\delta\mathbf{u}(t) \quad = \quad -H_{\mathbf{uu}}^{-1}\left(H_{\mathbf{ux}} + f_{\mathbf{u}}^\top S(t)\right)\delta\mathbf{x}(t). \tag{3.43}$$

Now, given a predetermined optimal open-loop control $\mathbf{u}$ and its corresponding trajectory $\mathbf{x}$, the current state deviation $\delta\mathbf{x}$ is easily calculated and hence the corresponding change in the optimal control $\delta\mathbf{u}$ can be computed via (3.41), (3.42) and (3.43).

It is well known that expansions such as the second order expansion of $\bar{J}$ will only be valid as long as $\Delta\mathbf{x}(i)$ and $\delta\mathbf{x}$ are small. However, as long as this limitation is taken into account, perturbation feedback control will produce excellent suboptimal trajectories whilst adding robustness and error tolerance properties to existing optimal open-loop controls.

4 Optimal Feedback Control using Neural Networks

In view of the shortcomings of perturbation feedback control, other methods of obtaining optimal feedback control are needed. One approach is to assume a specific structure for the control, commonly linear, as in [3]. The parameters of these feedback control laws are then determined by solving an optimal parameter selection problem with a fixed initial condition. There are two main criticisms of this approach. First, this optimal feedback control is only optimal over all controllers with that specific structure. It is extremely unlikely that the structure of the globally optimal feedback control for a general problem takes on any of the convenient polynomial structures. For example, many optimal control problems admit bang-bang solutions which cannot be approximated by a polynomial of any degree. Secondly, this controller will not be optimal for any initial condition except the prescribed one.

The neural network approach is similar. The optimal feedback controller is chosen to take the form of a backpropagation neural network. This choice is primarily due to that property of backpropagation neural networks described in Theorem A.1 of the Appendix. Stated simply, given *any* function $f \in \mathrm{L}^2$, the set of square integrable functions, with domain a compact subset of $\mathbb{R}^n$, we can find a two layer backpropagation neural network which can emulate f to arbitrary accuracy. Thus, despite assuming a specific neural network structure for the optimal feedback control, the true optimal feedback control can be emulated, whatever structure it happens to be. Even bang-bang control, which cannot be emulated by any polynomial structure, can be emulated by a backpropagation neural network. In contrast to optimal feedback controllers with a polynomial structure, whose parameters can be found by conventional optimisation techniques, the feedforward neural network structure does not lend itself to optimisation algorithms of this sort, due to the highly irregular nature of the resulting optimal parameter selection problem and its multitude of local minima.

Two methods for training the neural network controller are proposed. The first, called the indirect method, is based on a supervised learning scheme. The second, called the direct method, is similar to a graded learning scheme [2]. These concepts and their application to a number of non-trivial problems can be found in [7, 8].

4.1 Indirect Method

For discrete systems, Section 2.2.1 establishes the idea that if at time step k with state $\mathbf{x}(k)$ the optimal open-loop control is $\mathbf{u}(k)$, then the output of the corresponding optimal feedback controller $\psi(\mathbf{x}(k), k)$ is $\mathbf{u}(k)$. Similarly for continuous systems, Section 2.2.2 establishes the idea that if at time t with state vector $\mathbf{x}(t)$ the optimal open-loop control is $\mathbf{u}(t)$, then the output of the corresponding optimal feedback controller $\psi(\mathbf{x}(t), t)$ is $\mathbf{u}(t)$. Thus the optimal feedback controller can be approximated by fitting a function through, in theory, all possible optimal open-loop state and control pairs arising from some compact set of initial conditions $\Gamma \in \mathbb{R}^n$. However, since this is computationally impossible, it is sufficient to generate a large but finite set of optimal state and control pairs by considering points along a finite number of optimal trajectories with initial conditions in Γ. These initial conditions are selected according to some density distribution ρ defined over the set Γ.

For a discrete system, the construction of the set of training points is now a simple matter. A finite set of initial conditions is selected from Γ according to ρ:

$$\Delta = \left\{ \boldsymbol{\xi}_i^0, \quad i = 1, 2, \ldots, N \right\} \subset \Gamma. \tag{4.1}$$

Then, for each $\boldsymbol{\xi}_i^0$, the optimal open-loop control problem is solved to obtain

$$\Upsilon_i = \left\{ (\boldsymbol{\xi}_i^k, k, \boldsymbol{\nu}_i^k), \quad k = 0, \ldots, M-1 \right\}, \tag{4.2}$$

where $\boldsymbol{\xi}_i^k$ and $\boldsymbol{\nu}_i^k$ are the state and corresponding optimal control at step k along the optimal trajectory resulting from the initial condition $\boldsymbol{\xi}_i^0$. Then the approximating function is required to interpolate the points of the set:

$$\Upsilon = \bigcup_{i=1,\ldots,N} \Upsilon_i = \left\{ (\boldsymbol{\xi}_i^k, k, \boldsymbol{\nu}_i^k), \quad k = 0, \ldots, M-1, \quad i = 1, 2, \ldots, N \right\}, \tag{4.3}$$

where $\boldsymbol{\xi}_i^k$ and k are the inputs for the function and $\boldsymbol{\nu}_i^k$ is the output from the function.

Continuous systems are dealt with similarly, and the definition of Δ is retained. However, the definition of Υ_i is not as clear cut. Along each optimal trajectory, the state/optimal control pairs are chosen according to some density distribution $\hat{\rho}(t)$ defined on [0,T), thus

$$\Upsilon_i = \left\{ (\boldsymbol{\xi}_i^j, t^j, \boldsymbol{\nu}_i^j), \quad j = 1, \ldots, \hat{N} \right\}, \tag{4.4}$$

where $t^j \in [0, T)$ for $j = 1, \ldots, \hat{N}$, and $\boldsymbol{\nu}_i^j$ and $\boldsymbol{\xi}_i^j$ are the optimal control and state respectively at time t^j resulting from the initial condition $\boldsymbol{\xi}_i^0$. Then as before, the training set becomes

$$\Upsilon = \bigcup_{i=1,\ldots,N} \Upsilon_i \quad = \quad \Big\{ (\boldsymbol{\xi}_i^j, t^j, \boldsymbol{\nu}_i^j), \quad j = 1, \ldots, \hat{N}, \quad i = 1, 2, \ldots, N \Big\}. \tag{4.5}$$

The construction of the training sets Υ_i can be accomplished relatively easily using the optimal control package MISER [15]. The choice of the density distributions ρ and $\hat{\rho}$ is completely arbitrary for the theoretical problems considered here, but for actual applications specific information might be available. For the problems in Section 5, the initial conditions are set up as a uniform grid of points "covering" the set of initial conditions Γ. In a similar way, for continuous systems state/optimal control pairs are chosen with respect to a uniform distribution along each optimal open-loop trajectory.

Once the set of training points Υ has been constructed, all that remains is to train the neural network to emulate the input/output relationship of these points. The required network structure is not known in advance and can only (at this stage) be determined experimentally. Once a structure has been decided on, the network must be trained to emulate the training set Υ. The network can be trained by the backpropagation algorithm (presented in Section A.1) or any other suitable method. For the simulations of Section 5, the backpropagation algorithm is used. The primary reasons for this are because of its track record of excellent achievements and the fact that the final trained network is more important here than the length of time the network training takes.

It should be noted that the training of a optimal neural network feedback controller by this method does not depend on any properties of the dynamical system or the absence of constraints in the problem formulation. This method depends only on the knowledge of the optimal control for many possible states. Thus the indirect method can construct an optimal feedback controller for any optimal control problem that can be solved by numerical techniques. MISER can handle many types of constraints, so training an optimal feedback controller for a constrained problem via the indirect method requires little additional work. However, the residual error due to the use of neural network training algorithms mean that constraints will rarely be satisfied exactly. For problems in which constraint satisfaction is paramount, the use of an optimal feedback controller trained by the indirect method is not advisable.

4.2 Direct Method

The direct method for training an optimal neural network feedback controller uses a different approach. As it has been defined, the indirect method uses a supervised learning methodology, attempting to minimise the error that the network produces with respect to previously determined optimal open-loop control information. In contrast, the direct method attempts to minimise the performance index directly. The neural network controller is incorporated into the discrete optimal control problem ($\mathrm{P_D}$) by replacing every occurrence of the control $\mathbf{u}(k)$ by the neural network feedback controller $\Psi(\mathbf{x}(k), k; \mathcal{W})$. Here, as defined in the Appendix, $\Psi(\mathbf{x}(k), k; \mathcal{W})$ represents the output of a backpropagation neural network with inputs $\mathbf{x}(k)$ and k and set of synaptic weights $\mathcal{W}$. Thus the problem ($\mathrm{P_D}$) becomes

$$\min_{\mathcal{W}} J = \Phi(\mathbf{x}(M)) + \sum_{i=0}^{M-1} L(\mathbf{x}(i), \Psi(\mathbf{x}(i), i; \mathcal{W}), i), \tag{4.6}$$

$$\text{such that} \quad \mathbf{x}(k+1) = f(\mathbf{x}(k), \Psi(\mathbf{x}(k), k; \mathcal{W}), k), \tag{4.7}$$

$$\mathbf{x}(0) = \boldsymbol{\xi}^0, \tag{4.8}$$

where the initial condition $\boldsymbol{\xi}^0$ is selected from the bounded set of initial conditions Γ according to a density distribution $\rho(\boldsymbol{\xi}^0)$. Then the performance index J is a function of $\boldsymbol{\xi}^0$ and $\mathcal{W}$ only. Thus the optimal neural network feedback controller has weights such that the expected performance index is minimised over the entire set Γ:

$$\min_{\mathcal{W}} E[J(\boldsymbol{\xi}^0, \mathcal{W})] = \int_{\boldsymbol{\xi}^0 \in \Gamma} J(\boldsymbol{\xi}^0, \mathcal{W}) \rho(\boldsymbol{\xi}^0) d\boldsymbol{\xi}^0. \tag{4.9}$$

This can be approximated by

$$\min_{\mathcal{W}} \quad E[J(\boldsymbol{\xi}^0, \mathcal{W})] \approx \min_{\mathcal{W}} \quad \frac{1}{N} \sum_{\boldsymbol{\xi}^0 \in \Delta} J(\boldsymbol{\xi}^0, \mathcal{W}), \tag{4.10}$$

where Δ is defined, as in the previous section, as the result of choosing N elements from the set Γ according to the density distribution ρ:

$$\Delta = \left\{ \boldsymbol{\xi}_i^0, \quad i = 1, 2, \ldots, N \right\} \subset \Gamma. \tag{4.11}$$

As N gets large, this approximation will get closer to the true minimum with respect to $\mathcal{W}$ of $E[J(\boldsymbol{\xi}^0, \mathcal{W})]$. We require N to be finite for computational purposes, but the set Δ must be sufficiently dense in Γ to represent the density distribution ρ. Once the elements of Δ have been selected, the resulting problem is an optimal parameter selection problem in which the optimal parameters are optimal not only for a single initial condition but for a non-trivial set of initial conditions. The optimal parameter selection problem can be stated as follows:

$$\min_{\mathcal{W}} \quad \frac{1}{N} \sum_{\boldsymbol{\xi}^0 \in \Delta} J(\boldsymbol{\xi}^0, \mathcal{W}), \tag{4.12}$$

where

$$\begin{aligned} J(\xi^0, \mathcal{W}) &= \Phi(\mathbf{x}(M)) + \sum_{i=0}^{M-1} L(\mathbf{x}(i), \Psi(\mathbf{x}(i), i; \mathcal{W}), i), && (4.13) \\ \mathbf{x}(k+1) &= f(\mathbf{x}(k), \Psi(\mathbf{x}(k), k; \mathcal{W}), k), && (4.14) \\ \mathbf{x}(0) &= \boldsymbol{\xi}^0. && (4.15) \end{aligned}$$

To determine the optimal parameters, in this case the synaptic weights, a learning scheme similar to the backpropagation algorithm is used. Whereas backpropagation uses the instantaneous gradient of the square error with respect to the synaptic weights, this learning scheme uses the instantaneous gradient of the performance index with respect to the weights. The update formula for a generic synaptic weight w is as follows:

$$w(k+1) \quad \leftarrow \quad \left(w(k) - \mu \left. \frac{\partial J(\boldsymbol{\xi}^0, \mathcal{W})}{\partial w(k)} \right|_{\boldsymbol{\xi}^0(k)} \right), \tag{4.16}$$

where μ is the learning rate as defined for the backpropagation algorithm in the Appendix and $w \in \mathcal{W}$. To use this weight update formula, the gradient $\partial J(\boldsymbol{\xi}^0, \mathcal{W})/\partial w$ must be calculated. For notational convenience $(\bullet)|_i$ will be used to represent the evaluation of $(\bullet)$ with $k = i$, $\mathbf{x} = \mathbf{x}(i)$ and $\mathbf{u} = \Psi(\mathbf{x}(i), i; \mathcal{W})$.

Two methods of calculating $\partial J/\partial w$ are available. The first method is called the forward gradient method. The computation of $\partial J/\partial w$ by the forward gradient method is essentially a repeated application of the chain rule for each time step in the trajectory. Firstly, $\partial \mathbf{x}(k)/\partial w$ is needed. The base cases $k = 0$ and $k = 1$ are considered first:

$$\begin{aligned} \frac{\partial \mathbf{x}(0)}{\partial w} &= 0, && (4.17) \\ \frac{\partial \mathbf{x}(1)}{\partial w} &= \left. \left(\frac{\partial f}{\partial \mathbf{u}} \frac{\partial \Psi}{\partial w} \right) \right|_0 + \left. \left(\frac{\partial f}{\partial \mathbf{x}} + \frac{\partial f}{\partial \mathbf{u}} \frac{\partial \Psi}{\partial \mathbf{x}} \right) \right|_0 \frac{\partial \mathbf{x}(0)}{\partial w} \\ &= \left. \left(\frac{\partial f}{\partial \mathbf{u}} \frac{\partial \Psi}{\partial w} \right) \right|_0 . && (4.18) \end{aligned}$$

Similarly for $k = 2, \ldots, M$,

$$\frac{\partial \mathbf{x}(k)}{\partial w} = \left(\frac{\partial f}{\partial \mathbf{u}}\frac{\partial \Psi}{\partial w}\right)\bigg|_{k-1} + \left(\frac{\partial f}{\partial \mathbf{x}} + \frac{\partial f}{\partial \mathbf{u}}\frac{\partial \Psi}{\partial \mathbf{x}}\right)\bigg|_{k-1} \frac{\partial \mathbf{x}(k-1)}{\partial w}. \tag{4.19}$$

With these gradients established, it is a simple matter to find the desired gradient:

$$\frac{\partial J}{\partial w} = \Phi'(\mathbf{x}(M))\frac{\partial \mathbf{x}(M)}{\partial w} + \sum_{k=0}^{M-1}\left(\frac{\partial L}{\partial \mathbf{x}} + \frac{\partial L}{\partial \mathbf{u}}\frac{\partial \Psi}{\partial \mathbf{x}}\right)\bigg|_{k} \frac{\partial \mathbf{x}(k)}{\partial w} + \left(\frac{\partial L}{\partial \mathbf{u}}\frac{\partial \Psi}{\partial w}\right)\bigg|_{k}, \tag{4.20}$$

where $\partial\Psi/\partial\mathbf{x}$ and $\partial\Psi/\partial w$ are computed via (A.4), (A.5), (A.6), (A.7) and (A.8) of the Appendix.

The forward gradient method is easy to understand, and relatively simple to implement. Thoughtful coding can ensure that the gradient computation above can be done in one forward pass, hence the name of the method. However, it is computationally slow, due the the nature of the equations (4.18)—(4.20). This is discussed in more detail in Section 5.

The second method for calculating the gradient $\partial J/\partial w$, called the Hamiltonian method, is motivated by the first order necessary conditions for optimal parameter selection problems. As before, the Hamiltonian is defined by

$$H(\mathbf{x}(k), \mathbf{u}(k), \boldsymbol{\lambda}(k+1), k) = L(\mathbf{x}(k), \mathbf{u}(k), k) + \boldsymbol{\lambda}^\top(k+1) f(\mathbf{x}(k), \mathbf{u}(k), k). \tag{4.21}$$

Then, to calculate $\partial J/\partial w$, the state equations

$$\begin{aligned} \mathbf{x}(k+1) &= f(\mathbf{x}(k), \Psi(\mathbf{x}(k), k; \mathcal{W}), k), & (4.22)\\ \mathbf{x}(0) &= \mathbf{x}_0, & (4.23) \end{aligned}$$

are solved forwards from $k = 0$ to $k = M$ to obtain $\mathbf{x}(k)$ for $k = 0, \ldots, M$. Next, the costate equations

$$\begin{aligned} \boldsymbol{\lambda}^\top(k) &= \frac{\partial H(\mathbf{x}(k), \mathbf{u}(k), \boldsymbol{\lambda}(k+1), k)}{\partial \mathbf{x}(k)} \\ &= \left(\frac{\partial L}{\partial \mathbf{x}(k)} + \boldsymbol{\lambda}^\top(k+1)\frac{\partial f}{\partial \mathbf{x}(k)}\right. \\ &\qquad\qquad \left. + \left(\frac{\partial L}{\partial \mathbf{u}(k)} + \boldsymbol{\lambda}^\top(k+1)\frac{\partial f}{\partial \mathbf{u}(k)}\right)\frac{\partial \Psi}{\partial \mathbf{x}}\right)\bigg|_{k}, & (4.24)\\ \boldsymbol{\lambda}^\top(M) &= \frac{\partial \Phi(\mathbf{x}(M))}{\partial \mathbf{x}(M)}, & (4.25) \end{aligned}$$

are solved backwards from $k = M$ to $k = 0$ to obtain $\boldsymbol{\lambda}(k)$ for $k = 0, \ldots, M$. With the states $\mathbf{x}(k)$ and the costates $\boldsymbol{\lambda}(k)$ defined, the gradient can now be calculated:

$$
\begin{aligned}
\frac{\partial J}{\partial w} &= \sum_{k=0}^{M-1} \frac{\partial H(\mathbf{x}(k), \mathbf{u}(k), \boldsymbol{\lambda}(k+1), k)}{\partial w} \\
&= \sum_{k=0}^{M-1} \left(\frac{\partial L}{\partial \mathbf{u}(k)} + \boldsymbol{\lambda}^{\top}(k+1) \frac{\partial f}{\partial \mathbf{u}(k)} \right) \frac{\partial \Psi}{\partial w}\bigg|_k , \qquad (4.26)
\end{aligned}
$$

where, as before, $\partial\Psi/\partial\mathbf{x}$ and $\partial\Psi/\partial w$ can be computed by (A.4), (A.5), (A.6), (A.7) and (A.8) from the Appendix.

In contrast to the forward gradient method, we cannot compute the gradient by the Hamiltonian method in one forward pass because of the forwards/backwards nature of these equations. Consequently, this method requires considerably more storage than the forward gradient method. Again, for a detailed analysis of the computational aspects involved the reader is referred to Section 5.

Thus far, the direct method has only been derived for a discrete dynamical system, but an extension of the method for continuous optimal control problems would be extremely useful. As for the discrete case, the extension of the forward gradient method for continuous systems is a direct application of the chain rule. First, we derive the following ordinary differential equation and boundary condition which defines $\partial\mathbf{x}/\partial w$ for all $t \in [0, T]$ and for all $w \in \mathcal{W}$:

$$
\begin{aligned}
\frac{\partial \dot{\mathbf{x}}}{\partial w} &= \left(\frac{\partial f}{\partial \mathbf{u}} \frac{\partial \Psi}{\partial w} \right)\bigg|_t + \left(\frac{\partial f}{\partial \mathbf{x}} + \frac{\partial f}{\partial \mathbf{u}} \frac{\partial \Psi}{\partial \mathbf{x}} \right)\bigg|_t \frac{\partial \mathbf{x}}{\partial w}, \qquad (4.27) \\
\frac{\partial \mathbf{x}(0)}{\partial w} &= 0. \qquad (4.28)
\end{aligned}
$$

After solving these differential equations, the desired gradient can then be computed from:

$$
\frac{\partial J}{\partial w} = \Phi'(\mathbf{x}(T)) \frac{\partial \mathbf{x}(T)}{\partial w} + \int_0^T \left(\frac{\partial L}{\partial \mathbf{x}} + \frac{\partial L}{\partial \mathbf{u}} \frac{\partial \Psi}{\partial \mathbf{x}} \right)\bigg|_t \frac{\partial \mathbf{x}}{\partial w} + \left(\frac{\partial L}{\partial \mathbf{u}} \frac{\partial \Psi}{\partial w} \right)\bigg|_t dt. \qquad (4.29)
$$

However, solving the ordinary differential equation (4.27) poses a large problem, as this represents a differential equation for each possible combination of the states with the weights of the network. After solving (4.27), the value of $\partial\mathbf{x}/\partial w$ as a function of t must be stored for many discrete values of t, until the integral in (4.29) can be computed. With the large number of weights in even a moderately sized network and the resulting number of equations that must be solved in this way, the memory requirements of the forward gradient method for continuous systems become restrictive.

The extension to the continuous case for the Hamiltonian method, however, results in more numerically realistic gradient formulae. In a similar way to the discrete case, the derivation is motivated by the first order necessary conditions for optimal parameter selection problems. The Hamiltonian is defined by

$$H(\mathbf{x},\mathbf{u},\boldsymbol{\lambda},t) \quad = \quad L(\mathbf{x},\mathbf{u},t)+\boldsymbol{\lambda}^{\top} f(\mathbf{x},\mathbf{u},t). \tag{4.30}$$

To compute the gradient $\partial J/\partial w$, the state equations

$$\dot{\mathbf{x}} \quad = \quad f(\mathbf{x}(t),\Psi(\mathbf{x}(t),t;\mathcal{W}),t), \tag{4.31}$$

$$\mathbf{x}(0) \quad = \quad \mathbf{x}_0, \tag{4.32}$$

are solved forwards in time from $t=0$ to $t=T$. Next, the costate equations

$$\begin{aligned}\dot{\boldsymbol{\lambda}}^{\top} &= -\frac{\partial H(\mathbf{x},\mathbf{u},\boldsymbol{\lambda},t)}{\partial \mathbf{x}} \\ &= -\left(\frac{\partial L}{\partial \mathbf{x}}+\boldsymbol{\lambda}^{\top}\frac{\partial f}{\partial \mathbf{x}}+\left(\frac{\partial L}{\partial \mathbf{u}}+\boldsymbol{\lambda}^{\top}\frac{\partial f}{\partial \mathbf{u}}\right)\frac{\partial \Psi}{\partial \mathbf{x}}\right)\bigg|_t, \qquad (4.33)\\ \boldsymbol{\lambda}^{\top}(T) &= \frac{\partial \Phi(\mathbf{x}(T))}{\partial \mathbf{x}(T)}, \qquad (4.34)\end{aligned}$$

are solved backwards from $t=T$ to $t=0$. With this accomplished, the required gradient can now be calculated

$$\begin{aligned}\frac{\partial J}{\partial w} &= \int_0^T \frac{\partial H(\mathbf{x},\mathbf{u},\boldsymbol{\lambda},t)}{\partial w}dt \\ &= \int_0^T \left(\frac{\partial L}{\partial \mathbf{u}}+\boldsymbol{\lambda}^{\top}\frac{\partial f}{\partial \mathbf{u}}\right)\frac{\partial \Psi}{\partial w}\bigg|_t\, dt. \qquad (4.35)\end{aligned}$$

Note that for the Hamiltonian method, the ordinary differential equation (4.33) for $\boldsymbol{\lambda}$ has the same dimension as the state vector. Thus the Hamiltonian method is considered to be the more practical method for computing the gradient $\partial J/\partial w$ for continuous dynamical systems.

5 Simulations and Results

This section demonstrates the superb ability of the neural network controllers trained by the direct and indirect methods to emulate optimal feedback control. These controllers are trained and compared with the optimal open-loop control for three non-trivial problems. These problems concern highly non-linear, discrete dynamical systems, except for the first part of Problem 3 which concerns a continuous dynamical system. The lack of examples from the continuous domain should not be interpreted as an indication of the failure of these methods for this case, rather it is due primarily to time constraints.

The first problem to be considered concerns a discretised version of the Duffing equation with negative damping; the second considers a discrete version of the problem of maximising rocket altitude; while the third deals with the complex dynamics of a two-link robot manipulator. For these robot dynamics, the problem of minimising the time taken to travel between two points in the joint space of the robot is considered for the continuous system dynamics using a neural network trained by the indirect method, whilst the problem of moving between two points in the joint space of the robot with minimum control effort is considered for the corresponding discrete system dynamics using a neural network trained by the direct method. However, before the results are reported, some of the implementation issues concerning the two neural network controller training methods are discussed.

5.1 Computational Considerations

For training the neural network controller by the indirect method, the tools necessary are already available. The optimal control package MISER [15] provides a method of computing the set of open-loop training points, and there are many neural network training packages available for training the network. In spite of this, a neural network training program was independently developed as part of this research to allow experimentation with training algorithms and to facilitate the incremental development of programs for the direct training method. Experimentation with conventional optimisation algorithms in comparison to backpropagation demonstrated the superiority of the backpropagation algorithm over a wide variety of training sets. The conventional algorithms implemented, steepest decent, three conjugate gradient variants and a memory-less quasi-Newton, converged to mean-square-error rates considerably higher than backpropagation for all the training sets considered. These results indicate that conventional optimisation algorithms, which use the gradient of the sum of the error over all the training points, do not perform as well as backpropagation, which uses the instantaneous gradient at each point of the training set. Thus, for training neural network controllers by the indirect method in the following problems, the backpropagation algorithm is used.

Training the neural network controller by the direct training method requires the development of new neural network training programs since the dynamics of the discrete time optimal control system must be included in the gradient computation, as demonstrated in Section 4.2. Note that the direct method has only been implemented for discrete dynamical systems. This is due primarily to time constraints. As a consequence, all of the following discussion relates to discrete systems only. Both the forward gradient and Hamiltonian methods of computing the required gradients were implemented to allow a comparison of their merits. The advantages and disadvantages of the two methods became evident as they were implemented.

The principal advantage of the forward gradient method is that it only requires the storage of important gradient information such as $\partial\Psi/\partial w$, $\partial\Psi/\partial\mathbf{x}$ and $\partial\mathbf{x}(k)/\partial w$ temporarily. This is important since it allows the storage to be reused for the same gradient information at the next time step along the computed trajectory. As a result, the memory required to train a neural network controller by the forward gradient method is not affected by the number of time steps M in the system. If the neural network being trained is large or the system has a large value of M then this can be significant. These savings in memory are achieved by calculating $\partial\mathbf{x}(k)/\partial w$ using $\partial\mathbf{x}(k-1)/\partial w$ and (4.18)—(4.19), and accumulating $\partial J/\partial w$ using (4.20) at each time step along the current trajectory. However, there is a price to pay for this. As with most computational algorithms, there is a trade off between storage requirements and speed. The forward gradient method suffers in speed due to the nature of (4.18)—(4.20). Equation (4.18) requires one matrix multiplication involving a matrix having the same row dimension as the weights "vector", whilst (4.19) requires two of these matrix multiplications and (4.20) needs three. These matrix multiplications are typically very expensive in time because the dimension of a weights "vector" for even a small network is relatively large compared to the dimension of the state or control vectors. For example, a network from $\mathcal{N}^3_{4,15,15,2}$ has a total of 349 weights. This notation for representing a class of networks is defined in the Appendix. Analysis of (4.18)—(4.20) shows that $4M$

Network	Dimension of $\mathcal{W}$	Forward Gradient		Hamiltonian	
		Time (s)	Memory (Kb)	Time (s)	Memory (Kb)
$\mathcal{N}^2_{2,15,1}$	61	960	24.3	537	33.2
$\mathcal{N}^3_{2,15,15,1}$	301	3849	32.4	1712	58.4
$\mathcal{N}^4_{2,15,15,15,1}$	541	7599	40.6	3220	83.4
$\mathcal{N}^5_{2,15,15,15,15,1}$	781	11 003	48.7	4543	108.5

Table 1: Execution time and memory requirements for the forward gradient and Hamiltonian implementations, 12100 trajectories, terminal time $M = 20$, 2 states, 1 control.

of these expensive matrix multiplications are required along the computed trajectory.

The Hamiltonian method is at the other end of the spectrum. In contrast to the forward gradient method, the gradient information $\partial\Psi/\partial w$ and $\partial\Psi/\partial\mathbf{x}$ must be stored at each point along the computed trajectory, as this information is required in reverse order to compute the costate in (4.24). Due to this, the Hamiltonian method requires extra storage proportional to M. As stated previously, this can be a serious limitation if the network size or the number of time steps are large. However, while the Hamiltonian method requires more memory than the forward gradient method, it is considerably faster. The Hamiltonian method only requires matrix multiplication of a matrix with row dimension the same as the weights "vector" in (4.26). Analysis of (4.26) shows that only M of these expensive matrix multiplications are necessary.

This analysis is reflected in Table 1, which shows the execution time in seconds and memory requirements in kilobytes of the forward gradient and Hamiltonian implementations for 12 100 trajectories (100 cycles through a training set of 121 initial conditions). Note that in this section, all timings are for a Sparc I workstation. This table clearly shows the superior speed and larger memory requirement of the Hamiltonian method in comparison to the forward gradient method, even though the terminal time M is not large. For the problems and networks considered here, the memory requirements of the Hamiltonian method were not prohibitive and hence it is used throughout, though previous calculations have shown that both methods yield consistent results.

5.2 Problem 1 — The Duffing Equation

Problem 1 concerns a discretised version of the Duffing equation with negative damping. The system dynamics are as follows:

$$\mathbf{x}_1(k+1) = \mathbf{x}_1(k) + h\mathbf{x}_2(k), \tag{5.1}$$

$$\mathbf{x}_2(k+1) = \mathbf{x}_2(k) + h\left[0.2\mathbf{x}_2(k) - \mathbf{x}_1(k) - 0.1\mathbf{x}_1^3(k) + \mathbf{u}(k)\right], \tag{5.2}$$

$$\mathbf{x}_1(0) = \boldsymbol{\xi}_1,\ \mathbf{x}_2(0) = \boldsymbol{\xi}_2, \quad \boldsymbol{\xi} \in \Gamma, \tag{5.3}$$

where $\mathbf{x} \in \mathbb{R}^2$ and $\mathbf{u}(k) \in \{\mathbf{v} \in \mathbb{R} : -0.05 \leq \mathbf{v} \leq 0.05\}$, for $k = 0, \ldots, M-1$. The terminal time and time step are chosen as $M = 10$ and $h = 0.1$ respectively in order to keep the problem tractable.

A quadratic performance index over the dynamical system defined by

$$J = \frac{1}{2}\left[\mathbf{x}_1^2(M) + \mathbf{x}_2^2(M)\right] + \frac{1}{2}\sum_{k=0}^{M-1}\left[\mathbf{x}_1^2(k) + \mathbf{x}_2^2(k) + \mathbf{u}^2(k)\right], \tag{5.4}$$

is chosen, and the set of initial conditions Γ over which the optimal control is required is defined by

$$\Gamma = \left\{\boldsymbol{\xi} \in \mathbb{R}^2 \ : \ \boldsymbol{\xi}_1 \in [-1, 1], \boldsymbol{\xi}_2 \in [-1, 1]\right\}. \tag{5.5}$$

The density distribution for the set Γ is assumed to be uniform, so Δ is chosen to be a discrete grid of points covering Γ by

$$\Delta = \left\{\boldsymbol{\xi} \in \mathbb{R}^2 \ : \ \boldsymbol{\xi}_1 = -1 + 0.2i, \boldsymbol{\xi}_2 = -1 + 0.2j, \, i, j = 0, \ldots, 10\right\}. \tag{5.6}$$

This results in a set Δ of 121 training initial conditions, and a set Υ of 1210 training points.

Once Δ and Υ have been defined the training can begin. We have no a priori information about the most suitable network structure, apart from the number of external inputs and outputs. The number of external inputs must be the number of states or the number of states plus one, depending on whether the controller is time dependent or not, and the number of external outputs must be the number of controls. Finding a suitable network is a painstaking trial and error process. First, a set of two layer networks with a range of sizes are chosen. Each of these is trained over about 1000 cycles of the training data or training initial conditions to provide an indication of its likely performance once the training algorithm converges. If none of these appear promising, then the number of layers are increased by one, a new set of networks is chosen, and the process is repeated. If a network appears capable of satisfactory performance, then it is trained until convergence. If the final network has a small enough error or performance index over the training set, then the search is finished. Otherwise, as before, the number of layers are increased by one, a new set of networks is chosen, and the process is repeated. This procedure is intended to ensure that the smallest possible network which achieves satisfactory performance is found, but this can in no way be guaranteed. This process is used to choose the networks in each problem that follows.

For the current problem, a network from $\mathcal{N}^4_{2,5,5,5,1}$, hence a strictly state dependent feedback controller, gives good results for the indirect method. On completion of 2000 cycles with a learning rate of 0.1, this controller exhibits a root-mean-square error around 0.5% over the training data Υ. This training represents at total of 2 420 000 changes to the weights of the network and requires about 90 minutes of cpu time. When tested on the actual problem, the sum of the performance index for this neural network controller over all the points in the training set Δ is within 0.1% of the sum of the performance index over all the points in the training set for the corresponding optimal open-loop solutions. For ease of reference this is referred to as 0.1% suboptimality.

For training by the direct method a $\mathcal{N}^4_{2,15,15,15,1}$ network, hence also a strictly state dependent feedback controller, gives good results. After just 20 cycles through the training set of initial conditions with a learning rate of 0.1, it exhibits 0.1% suboptimality. This represents only 2420 changes to the weights and requires just 5 minutes of cpu time. Interestingly, this network tested on the indirect method training data Υ shows a root-mean-square error of 0.8%.

Figure 5.2.1 demonstrates the ability of the neural network controllers to take on the structure of the optimal feedback control over the entire set Δ of training initial conditions. This figure shows the optimal open-loop performance index and the performance index produced by the neural network controllers for a subset of the training initial conditions Δ defined by $\{\boldsymbol{\xi}^i, i = 66, \ldots, 98 : \boldsymbol{\xi}^i_1 = -1 + 0.2(i \,|\, 11), \boldsymbol{\xi}^i_2 = -1 + 0.2(i \bmod 11)\}$, where " | " represents integer division. The results are so close as to be indistinguishable. Figures 5.2.2 to 5.2.4 show the states and control for a trajectory whose initial condition is part of the training set Δ. Figures 5.2.5 to 5.2.7 show the states and control for a trajectory

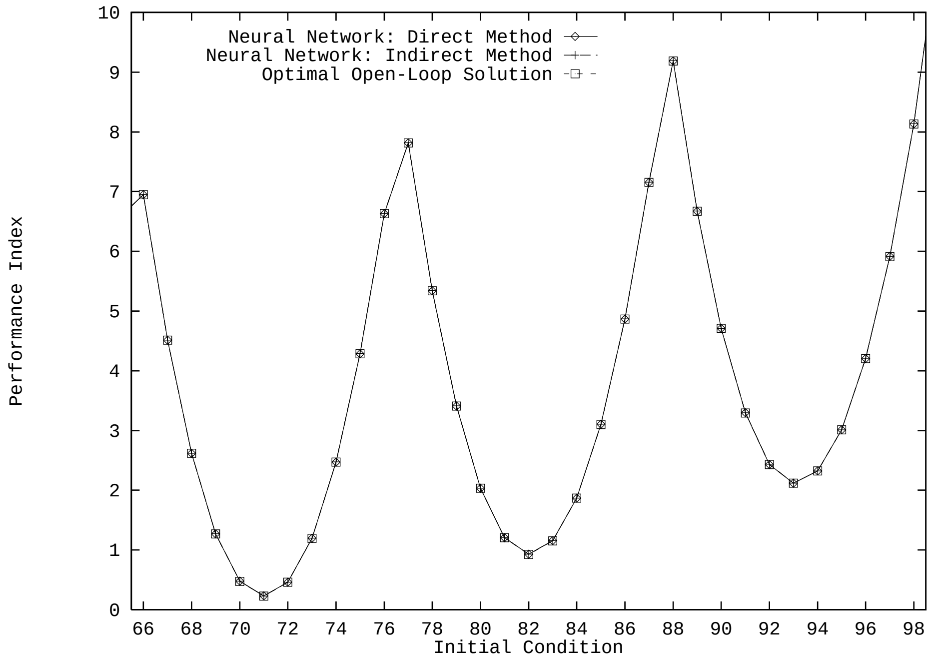

Figure 5.2.1: Neural Network v Optimal Open-Loop Controller: Problem 1.

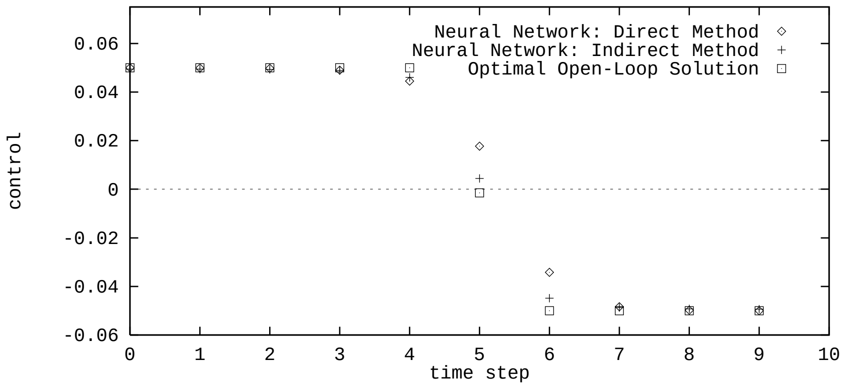

Figure 5.2.2: Neural Network v Optimal Open-Loop Controller: Problem 1, $\boldsymbol{\xi} \in \Delta$.

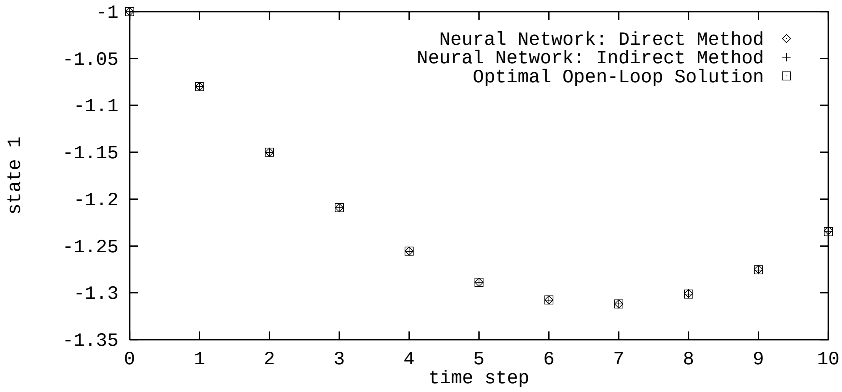

Figure 5.2.3: Neural Network v Optimal Open-Loop Controller: Problem 1, $\boldsymbol{\xi} \in \Delta$.

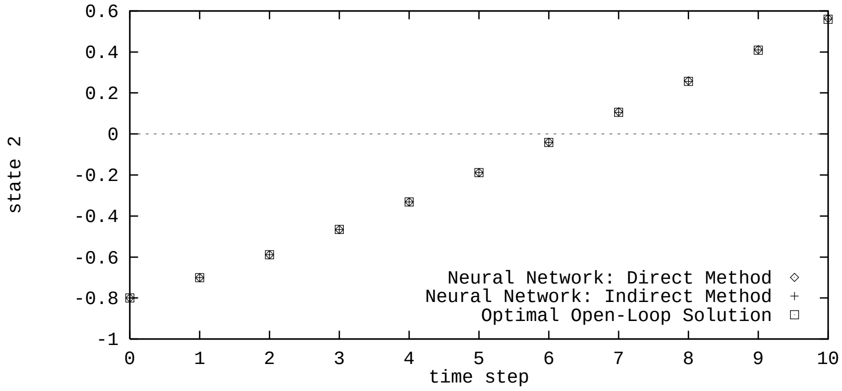

Figure 5.2.4: Neural Network v Optimal Open-Loop Controller: Problem 1, $\boldsymbol{\xi} \in \Delta$.

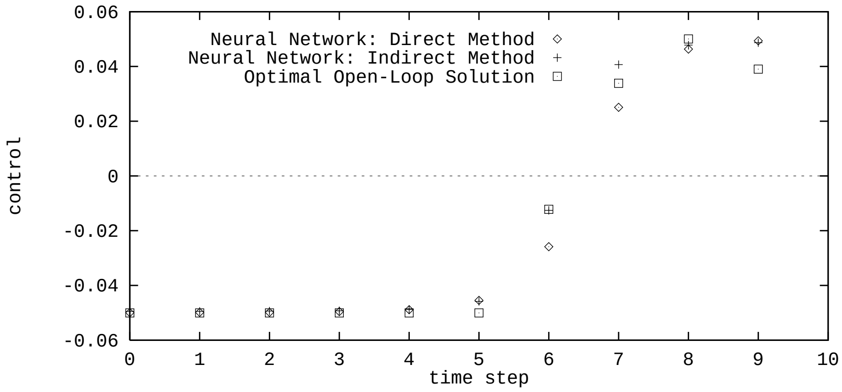

Figure 5.2.5: Neural Network v Optimal Open-Loop Controller: Problem 1, $\boldsymbol{\xi} \notin \Delta$.

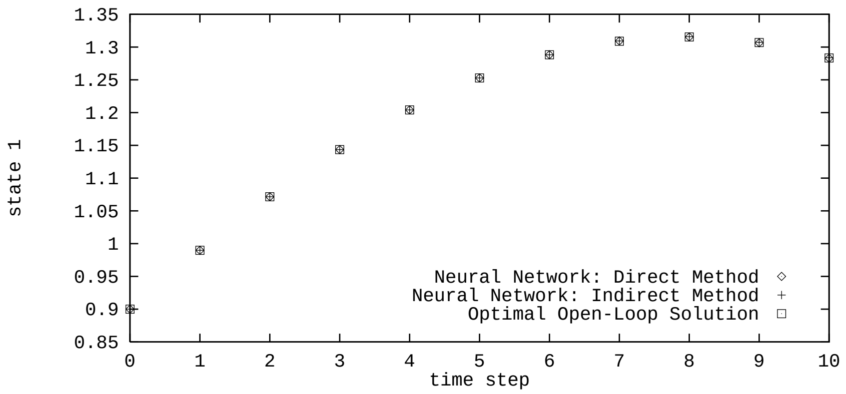

Figure 5.2.6: Neural Network v Optimal Open-Loop Controller: Problem 1, $\boldsymbol{\xi} \notin \Delta$.

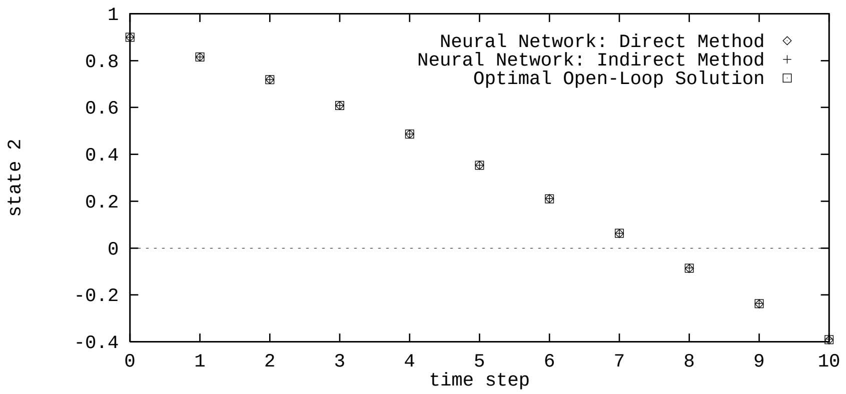

Figure 5.2.7: Neural Network v Optimal Open-Loop Controller: Problem 1, $\boldsymbol{\xi} \notin \Delta$.

whose initial condition is not in Δ. For each of these trajectories, both neural network controllers produce a performance index within 1% of the corresponding optimal open-loop performance index. This excellent result for the trajectory whose initial condition is not in Δ demonstrates that both controllers can generalise for initial conditions outside of the training set Δ. This, and the excellent correspondence between the performance indices demonstrated in Figure 5.2.1 suggests that the excellent results for these two trajectories will be reflected throughout the set Γ. Thus we conclude that these controllers emulate the optimal feedback control almost exactly over the entire set of initial conditions Γ.

5.3 Problem 2 — Maximum Rocket Height

Problem 2 concerns the vertical ascent of a rocket. The continuous system dynamics are taken from [6]. The discrete time version is obtained by the Euler scheme, where the time step is taken as one second. The system dynamics are governed by:

$$\mathbf{x}_1(k+1) = \mathbf{x}_1(k) - \mathbf{u}(k), \tag{5.7}$$

$$\mathbf{x}_2(k+1) = \mathbf{x}_2(k) + \mathbf{x}_3(k), \tag{5.8}$$

$$\mathbf{x}_3(k+1) = \mathbf{x}_3(k) + \frac{V\mathbf{u}(k) - Q(\mathbf{x}(k))}{\mathbf{x}_1(k)} - g, \tag{5.9}$$

$$\mathbf{x}_1(0) = \boldsymbol{\xi}_1,\ \mathbf{x}_2(0) = \boldsymbol{\xi}_2,\ \mathbf{x}_3(0) = \boldsymbol{\xi}_3, \quad \boldsymbol{\xi} \in \Gamma, \tag{5.10}$$

where $\mathbf{x}_1(k)$ is the mass of the rocket, $\mathbf{x}_2(k)$ is the rocket's altitude in km above the earth's surface, $\mathbf{x}_3(k)$ is the rocket's vertical velocity and $\mathbf{u}(k)$ is the mass flow rate of the rocket's fuel. The constant $V = 2$ ms^{-1} is the gas nozzle velocity and $g = 0.01$ kms^{-2} is the acceleration due to gravity (assumed constant). The function $Q(\mathbf{x}(k))$ is the aerodynamic drag of the rocket, defined by

$$Q(\mathbf{x}(k)) = 0.05 \exp\left[0.01\mathbf{x}_2(k)\right]\left[\mathbf{x}_3(k)\right]^2. \tag{5.11}$$

The terminal time for the system is $M = 50$, and the control is bounded by $0 \leq \mathbf{u}(k) \leq 0.04$ for $k = 0, 1, \ldots, M-1$.

In the original problem formulation, there is a terminal-state constraint:

$$\mathbf{x}_1(M) = 0.2. \tag{5.12}$$

Since constraints cannot be modelled explicitly by the direct training method, this is included in the performance index as a penalty function. The original control objective is to maximize the rocket's peak altitude by the choice of the control $\mathbf{u}(\mathrm{k})$. This suggests the performance index:

$$J = -k_1\mathbf{x}_2(M) + k_2(\mathbf{x}_1(M) - 0.2)^2, \tag{5.13}$$

where $k_1 = 0.01$ and $k_2 = 10$. For the original problem, the initial conditions are $\mathbf{x}_1(0) = 1$, $\mathbf{x}_2(0) = 0$ and $\mathbf{x}_3(0) = 0$. The set of initial conditions Γ over which the optimal feedback control is desired is obtained by suitable perturbations about these initial conditions:

$$\Gamma = \left\{ \boldsymbol{\xi} \in \mathbb{R}^3 \,:\, \boldsymbol{\xi}_1 \in [0.9, 1.1],\, \boldsymbol{\xi}_2 \in [-5, 5],\, \boldsymbol{\xi}_3 \in [0, 2] \right\}. \tag{5.14}$$

Again, we assume a uniform density distribution for the selection of training points from Γ and define Δ to be a grid of points covering Γ:

$$\Delta = \left\{ \boldsymbol{\xi} \in \mathbb{R}^3 \,:\, \boldsymbol{\xi}_1 = 0.9 + 0.05i,\, \boldsymbol{\xi}_2 = -5 + 2.5j,\, \boldsymbol{\xi}_3 = 0.5k,\, i, j, k = 0, \ldots, 4 \right\}. \tag{5.15}$$

This results in a set of 125 training initial conditions and a set of 6250 training points.

For the indirect method, the best results are obtained with a $\mathcal{N}^5_{3,20,20,20,20,1}$ network, a strictly state dependent feedback controller. Even this large network requires some 5000 cycles through the training data with a learning rate of 0.1 to achieve a root-mean-square error of 0.1%. This represents a total of 31 250 000 changes to the weights of the network, requiring approximately 9 days of cpu time. Tested on the actual problem, this network achieves 3.5% suboptimality.

Finding a suitable network by the direct method is considerably easier, as a $\mathcal{N}^3_{3,5,5,1}$ network, hence a strictly state dependent feedback controller, is able to achieve excellent results. This network is trained for 2500 cycles with a learning rate of 0.01 until 3% suboptimality is reached, and then for another 7500 cycles until 1% suboptimality is attained. These 10 000 training cycles represent 1 250 000 changes to the weights and require about $1\frac{1}{2}$ days of cpu time. A smaller initial learning rate than normal is needed here due to the large number of time steps M. As the number of time steps increases, the performance index becomes sensitive to small changes in the weights of the neural network. If too large a learning rate is used, the direct method will tend to settle in a local minimum a long way from the global minimum, or even become unstable. Interestingly, as with Problem 1, when this network is tested on the input/output training data Υ it fares worse than the network trained by the indirect method, attaining a mean-square-error of 0.4%.

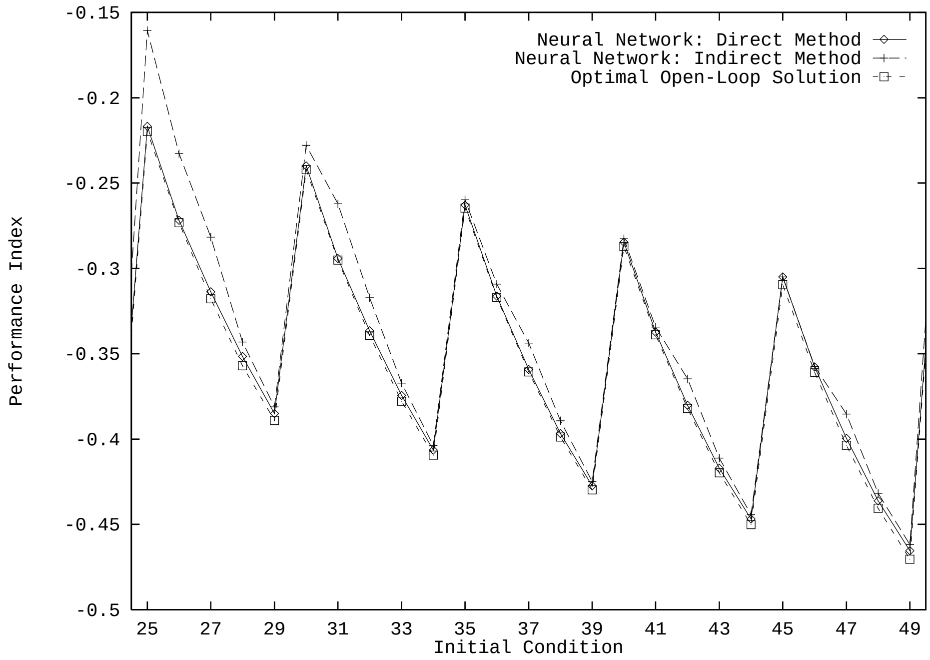

Figure 5.3.1: Neural Network v Open-Loop Controller: Problem 2.

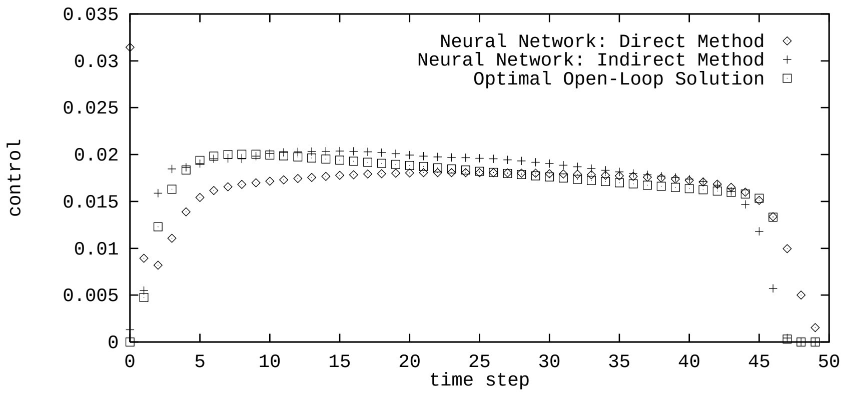

Figure 5.3.2: Neural Network v Open-Loop Controller: Problem 2, $\boldsymbol{\xi} \in \Delta$.

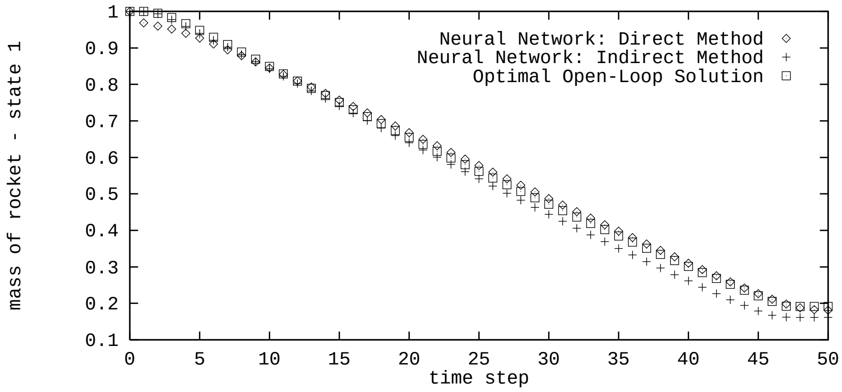

Figure 5.3.3: Neural Network v Open-Loop Controller: Problem 2, $\boldsymbol{\xi} \in \Delta$.

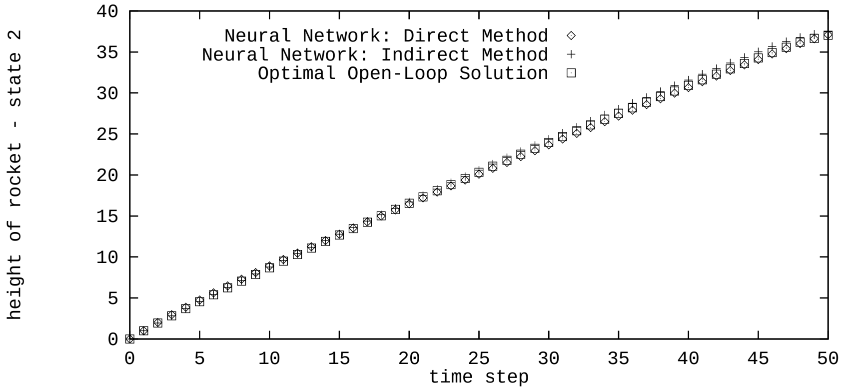

Figure 5.3.4: Neural Network v Open-Loop Controller: Problem 2, $\boldsymbol{\xi} \in \Delta$.

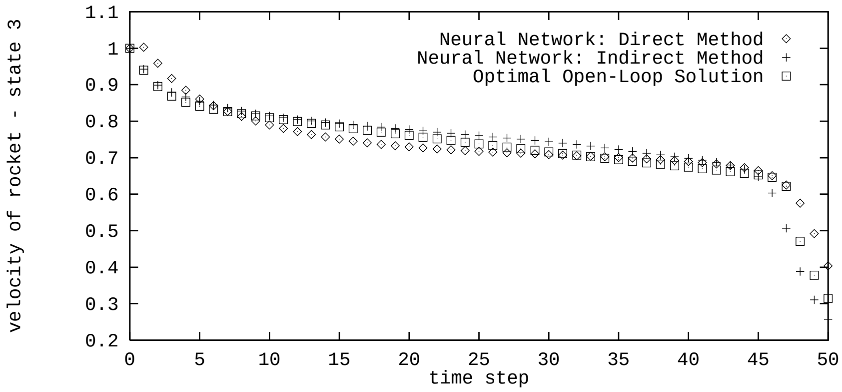

Figure 5.3.5: Neural Network v Open-Loop Controller: Problem 2, $\boldsymbol{\xi} \in \Delta$.

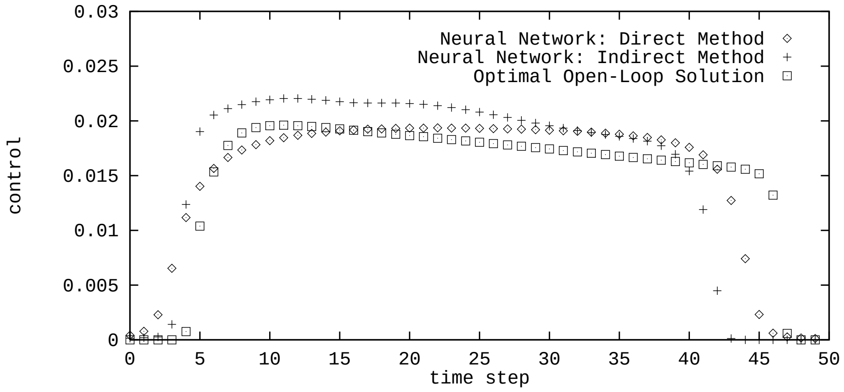

Figure 5.3.6: Neural Network v Open-Loop Controller: Problem 2, $\boldsymbol{\xi} \notin \Delta$.

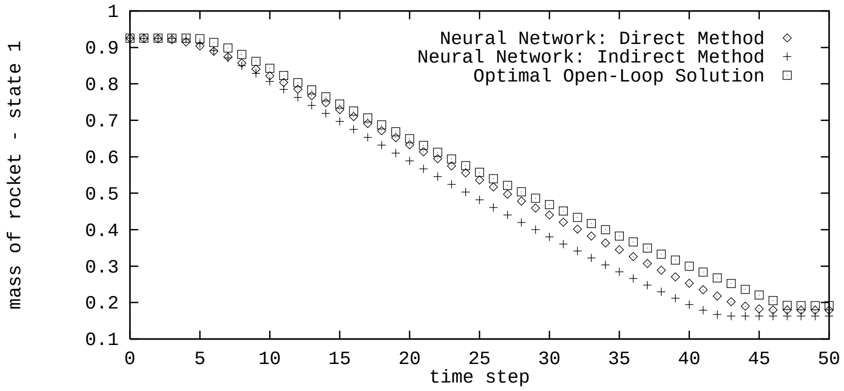

Figure 5.3.7: Neural Network v Open-Loop Controller: Problem 2, $\boldsymbol{\xi} \notin \Delta$.

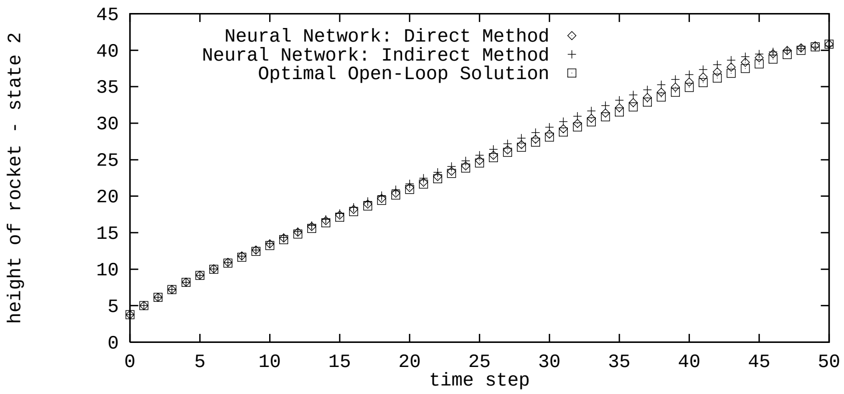

Figure 5.3.8: Neural Network v Open-Loop Controller: Problem 2, $\boldsymbol{\xi} \notin \Delta$.

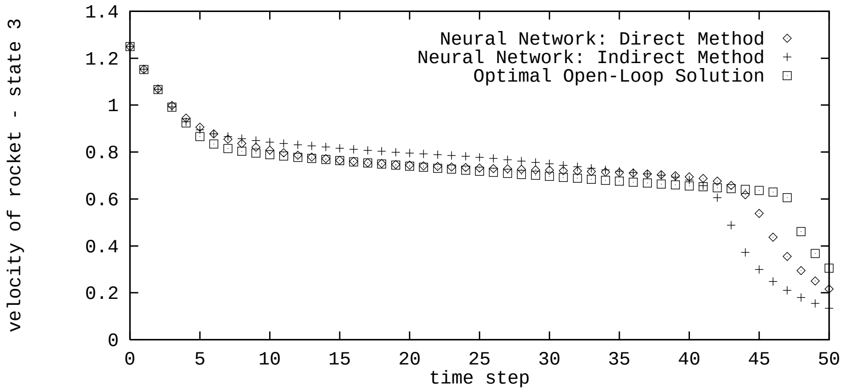

Figure 5.3.9: Neural Network v Open-Loop Controller: Problem 2, $\boldsymbol{\xi} \notin \Delta$.

Figure 5.3.1 shows the optimal open-loop performance index and the performance index produced by the neural networks over a subset of the training initial conditions Δ defined by $\{\boldsymbol{\xi}^i, i = 25, \ldots, 49 : \boldsymbol{\xi}_1^i = 0.9 + 0.05((i \,|\, 5) \,|\, 5), \boldsymbol{\xi}_2^i = -5 + 2.5((i \,|\, 5) \bmod 5), \boldsymbol{\xi}_3^i = 0.5(i \bmod 5)\}$. It is obvious from this figure that the neural network trained by the direct method does a better job of emulating the optimal feedback control than the neural network trained by the indirect method. However, both have successfully learnt the structure of the optimal feedback control over the entire training set of initial conditions Δ. Figures 5.3.2 to 5.3.5 show the control $\mathbf{u}$ and each of the states $\mathbf{x}_i$, $i = 1, 2, 3$, for a trajectory whose initial condition is in the training set of initial conditions Δ. Figures 5.3.6 to 5.3.9 show the control $\mathbf{u}$ and each of the states $\mathbf{x}_i$, $i = 1, 2, 3$, for a trajectory whose initial condition is not in Δ. For each of these trajectories, the solution produced by the neural network controller trained by the direct method is within 1% of the optimal open-loop performance index. For the indirect method, the solutions are within 4% of the optimal open-loop performance index. The consistency of the performance of the neural network controllers over these two trajectories, one whose initial condition is in Δ and one whose initial condition is not in Δ demonstrates that they are able to generalise for initial conditions not in the training set. This, and the correspondence between the performance indices in Figure 5.3.1 clearly demonstrate that both neural network controllers emulate the optimal feedback control over the entire set of initial conditions Γ.

5.4 Problem 3 — Two-Link Robot

In order to confirm these favourable results for neural network optimal feedback controllers, two problems concerning the highly non-linear dynamics of a two-dimensional two-link robot manipulator are posed for which a neural network feedback controller is constructed. The first problem is in the continuous dynamical system domain and seeks to find the necessary joint torques to control a two-link robot manipulator from an arbitrary point in the joint space of the robot to the origin in minimum time. For this problem, referred to as Problem 3(a), the optimal neural network feedback controller is trained by the indirect method. The second problem is in the discrete time domain and concerns finding the joint torques necessary to control a two-link robot manipulator from an arbitrary point in the joint space of the robot to the origin in a fixed number of steps with minimum control effort. For this problem, referred to as Problem 3(b), the optimal neural network feedback controller is trained by the direct method.

5.4.1 Problem 3(a)

The two-dimensional two-link robot manipulator continuous dynamics are as follows. For a more detailed description and further simulations see [7].

$$\dot{\mathbf{x}}_1 = \mathbf{x}_3, \tag{5.16}$$

$$\dot{\mathbf{x}}_2 = \mathbf{x}_4, \tag{5.17}$$

$$\dot{\mathbf{x}}_3 = \Lambda\left\{2hH_{22}\mathbf{x}_3\mathbf{x}_4 + hH_{22}\mathbf{x}_4^2 + hH_{12}\mathbf{x}_3^2 + H_{22}\mathbf{u}_1 - H_{12}\mathbf{u}_2\right\}, \tag{5.18}$$

$$\dot{\mathbf{x}}_4 = \Lambda\left\{-2hH_{12}\mathbf{x}_3\mathbf{x}_4 - hH_{11}\mathbf{x}_3^2 - hH_{12}\mathbf{x}_4^2 + H_{11}\mathbf{u}_2 - H_{12}\mathbf{u}_1\right\}, \tag{5.19}$$

$$\mathbf{x}_1(0) = \boldsymbol{\xi}_1,\ \mathbf{x}_2(0) = \boldsymbol{\xi}_2,\ \mathbf{x}_3(0) = \boldsymbol{\xi}_3,\ \mathbf{x}_4(0) = \boldsymbol{\xi}_4, \quad \boldsymbol{\xi} \in \Gamma, \tag{5.20}$$

where

$$\Lambda = \frac{1}{H_{11}H_{22} - H_{12}^2}, \tag{5.21}$$

$$H_{11} = I_1 + I_2 + m_1 l_1^2 + m_2\left(L_1^2 + l_2^2 + 2L_1 l_2 \cos\mathbf{x}_2\right), \tag{5.22}$$

$$H_{12} = I_2 + m_2 l_2^2 + m_2 L_1 l_2 \cos\mathbf{x}_2, \tag{5.23}$$

$$H_{22} = I_2 + m_2 l_2^2, \tag{5.24}$$

$$h = m_2 L_1 l_2 \sin\mathbf{x}_2, \tag{5.25}$$

Length of Link 1	L_1	0.4 m
Length of Link 2	L_2	0.4 m
Mass of Link 1	m_1	0.5 kg
Mass of Link 2	m_2	0.5 kg
Moment of Inertia about C.M. for Link 1	I_1	0.1 kgm^2
Moment of Inertia about C.M. for Link 2	I_2	0.1 kgm^2
Distance from Joint 1 to C.M. for Link 1	l_1	0.2 m
Distance from Joint 2 to C.M. for Link 2	l_2	0.2 m

Table 2: Numerical Constants for Equations (5.21)—(5.25)

$\mathbf{x}_1$ and $\mathbf{x}_2$ represent the joint angles and $\mathbf{x}_3$ and $\mathbf{x}_4$ represent the joint velocities. The values for each of the numerical constants are summarised in Table 2. The terminal state is constrained to be at the origin

$$\mathbf{x}_1(T) = 0, \mathbf{x}_2(T) = 0, \mathbf{x}_3(T) = 0, \mathbf{x}_4(T) = 0, \tag{5.26}$$

and the control vector $\mathbf{u}$ is constrained to lie in the compact set U defined by

$$U \quad = \quad \{\mathbf{v} : -10 \leq \mathbf{v}_i \leq 10, \, i = 1, 2\}. \tag{5.27}$$

The control objective is a minimum time trajectory which satisfies the constraints. Thus, the performance index is

$$J \quad = \quad T = \int_0^T dt. \tag{5.28}$$

The set of initial conditions Γ over which the optimal feedback controller is to be constructed is defined by

$$\Gamma \quad = \quad \left\{\boldsymbol{\xi} : \boldsymbol{\xi}_{1,2} \in [-\pi/8, \pi/8], \boldsymbol{\xi}_{3,4} = 0\right\}. \tag{5.29}$$

Then, as before, the training set of initial conditions Δ is chosen with respect to a uniform distribution over Γ:

$$\Delta \quad = \quad \left\{\boldsymbol{\xi} : \boldsymbol{\xi}_1 = -\frac{\pi}{8} + \frac{\pi}{16}i, \quad \boldsymbol{\xi}_2 = -\frac{\pi}{8} + \frac{\pi}{16}j, \quad \boldsymbol{\xi}_{3,4} = 0; \quad i, j = 0, \ldots, 4\right\}. \tag{5.30}$$

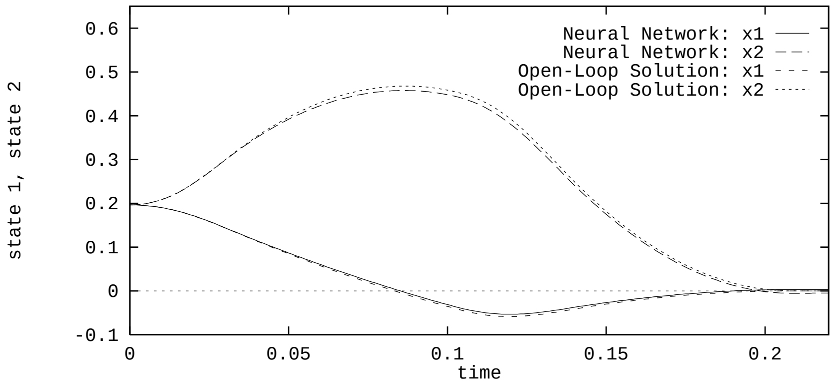

Figure 5.4.1: Neural Network v Open-Loop Controller: Problem 3(a), $\boldsymbol{\xi} \in \Delta$.

From each optimal open-loop trajectory resulting from the set Δ, 120 pairs of state and their corresponding optimal control are extracted. This defines a total of 3000 training points in the set Υ.

Some initial difficulties are experienced in training the neural network to emulate the set of training points Υ. To overcome this difficulty, a neural network is trained separately for each control. Note that this can be simulated with just one neural network by partitioning the network into two halves, and setting the weights on the connections between the two halves to zero. For each control, a $\mathcal{N}^4_{4,20,20,20,1}$ network, hence a strictly state dependent feedback control, is used, and trained until the average root-mean-square error is about 5%. This took approximately $1\frac{1}{2}$ days of cpu time.

Figures 5.4.1 to 5.4.8 demonstrate the neural network's effectiveness in emulating the optimal open-loop solution. Figures 5.4.1 to 5.4.4 show the states and controls for a trajectory whose initial condition is in the training set of initial conditions Δ. Figures 5.4.5 to 5.4.8 show the states and controls for a trajectory whose initial condition is not in Δ. In each case, the neural network optimal feedback controller does an excellent job of emulating the optimal open-loop control. It should be noted, however, that the neural controller does not satisfy the terminal state constraints exactly. This is to be expected due to the nature of the backpropagation algorithm, with which it is almost impossible to achieve zero error globally. However, in practice, this is not a serious shortcoming, as usually a kinematic controller is used over the final part of a trajectory to satisfy this type of constraint.

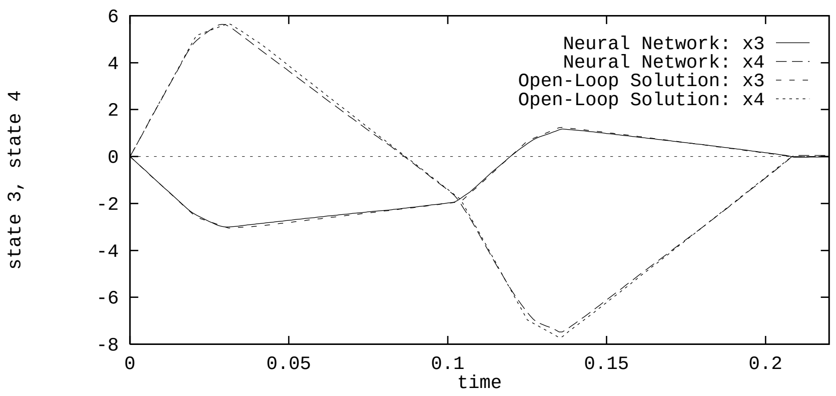

Figure 5.4.2: Neural Network v Open-Loop Controller: Problem 3(a), $\boldsymbol{\xi} \in \Delta$.

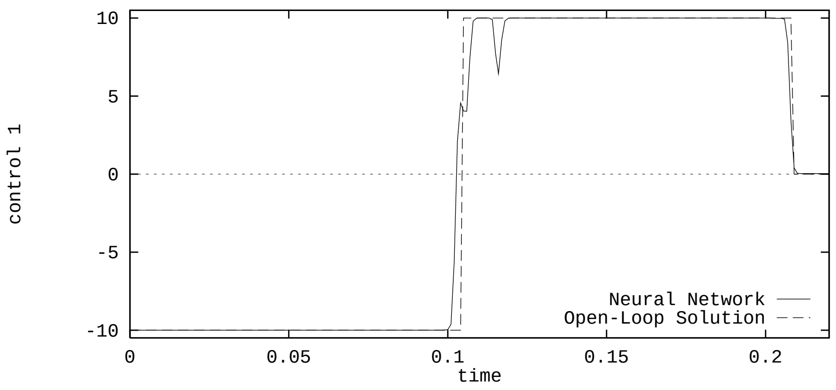

Figure 5.4.3: Neural Network v Open-Loop Controller: Problem 3(a), $\boldsymbol{\xi} \in \Delta$.

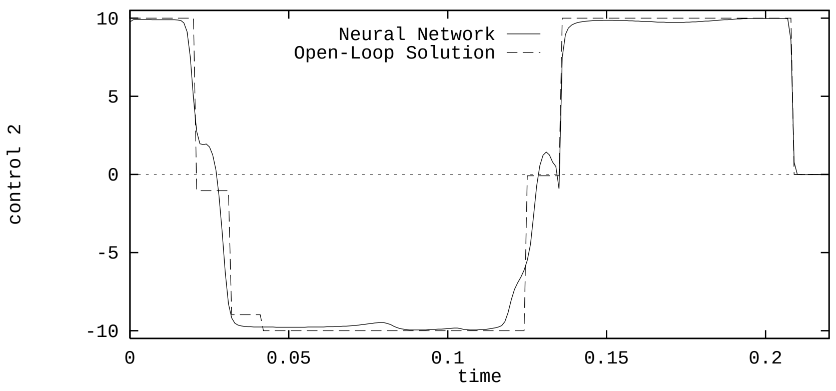

Figure 5.4.4: Neural Network v Open-Loop Controller: Problem 3(a), $\boldsymbol{\xi} \in \Delta$.

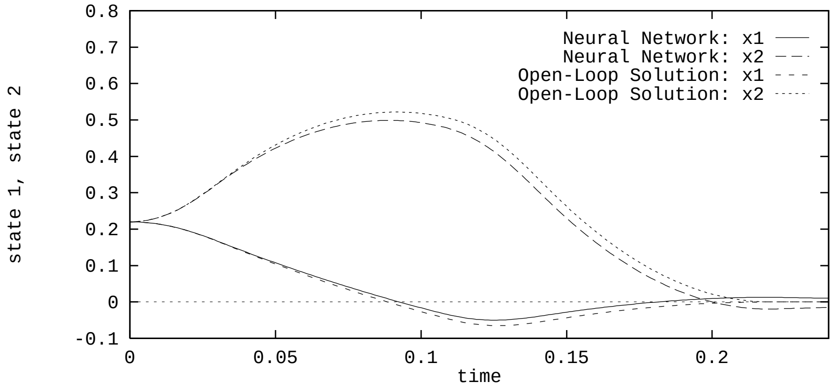

Figure 5.4.5: Neural Network v Open-Loop Controller: Problem 3(a), $\boldsymbol{\xi} \notin \Delta$.

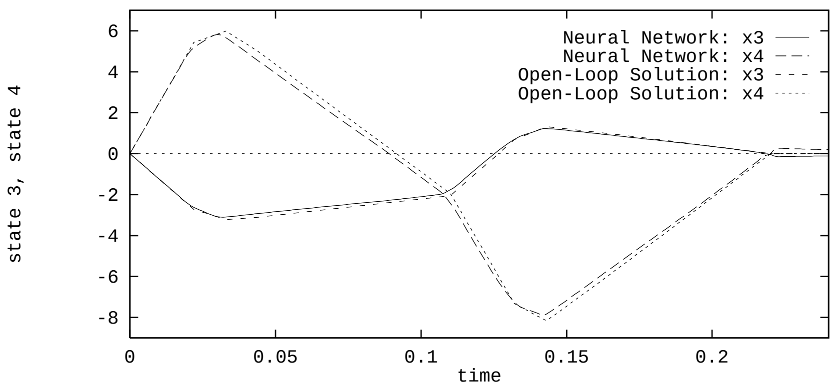

Figure 5.4.6: Neural Network v Open-Loop Controller: Problem 3(a), $\boldsymbol{\xi} \notin \Delta$.

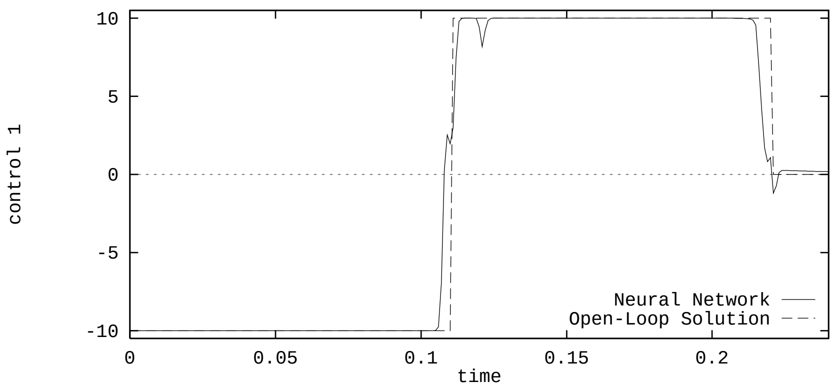

Figure 5.4.7: Neural Network v Open-Loop Controller: Problem 3(a), $\boldsymbol{\xi} \notin \Delta$.

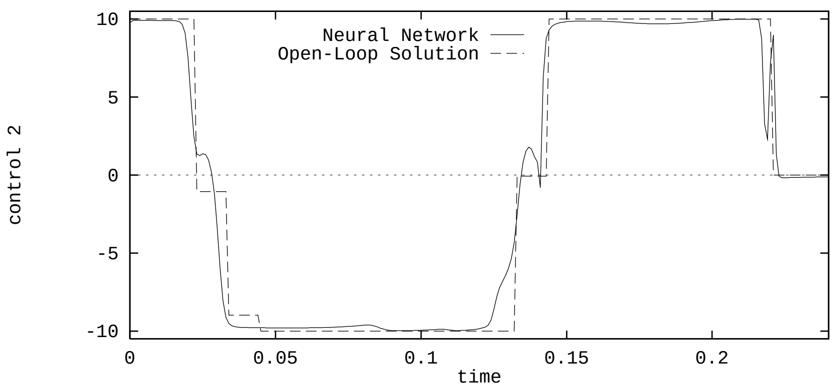

Figure 5.4.8: Neural Network v Open-Loop Controller: Problem 3(a), $\boldsymbol{\xi} \notin \Delta$.

5.4.2 Problem 3(b)

To discretise the dynamics of the robotic manipulator, the continuous system dynamics in (5.16)—(5.20) are approximated by the Euler method, with discretisation interval ε.

$$\begin{aligned}
\mathbf{x}_1(k+1) &= \mathbf{x}_1(k) + \varepsilon \mathbf{x}_3(k), \qquad (5.31)\\
\mathbf{x}_2(k+1) &= \mathbf{x}_2(k) + \varepsilon \mathbf{x}_4(k), \qquad (5.32)\\
\mathbf{x}_3(k+1) &= \mathbf{x}_3(k) + \varepsilon\Lambda \Big\{ 2hH_{22}\mathbf{x}_3(k)\mathbf{x}_4(k) + hH_{22}[\mathbf{x}_4(k)]^2 \\
&\qquad + hH_{12}[\mathbf{x}_3(k)]^2 + H_{22}\mathbf{u}_1(k) - H_{12}\mathbf{u}_2(k) \Big\}, \qquad (5.33)\\
\mathbf{x}_4(k+1) &= \mathbf{x}_4(k) + \varepsilon\Lambda \Big\{ -2hH_{12}\mathbf{x}_3(k)\mathbf{x}_4(k) - hH_{11}[\mathbf{x}_3(k)]^2 \\
&\qquad - hH_{12}[\mathbf{x}_4(k)]^2 + H_{11}\mathbf{u}_2(k) - H_{12}\mathbf{u}_1(k) \Big\}, \qquad (5.34)
\end{aligned}$$

$$\mathbf{x}_1(0) = \boldsymbol{\xi}_1,\ \mathbf{x}_2(0) = \boldsymbol{\xi}_2,\ \mathbf{x}_3(0) = \boldsymbol{\xi}_3,\ \mathbf{x}_4(0) = \boldsymbol{\xi}_4,\ \boldsymbol{\xi} \in \Gamma, \qquad (5.35)$$

for $k = 0, \ldots, M$, where Λ, H_{11}, H_{12}, H_{22} and h are defined as in (5.21)—(5.25), with the numerical constants as in Table 2. The discretisation interval ε and the number of time steps M are set to 0.05 and 10 respectively. The control is unbounded, unlike Problem 3(a), but the terminal state constraint is retained in the form of a penalty function. Thus to represent the control objective of minimum control effort, we define the performance index to be

$$\begin{aligned}
J &= k_1 \Big([\mathbf{x}_1(M)]^2 + [\mathbf{x}_2(M)]^2 + [\mathbf{x}_3(M)]^2 + [\mathbf{x}_4(M)]^2 \Big) \\
&\qquad + k_2 \sum_{k=0}^{M-1} \Big([\mathbf{u}_1(k)]^2 + [\mathbf{u}_2(k)]^2 \Big), \qquad (5.36)
\end{aligned}$$

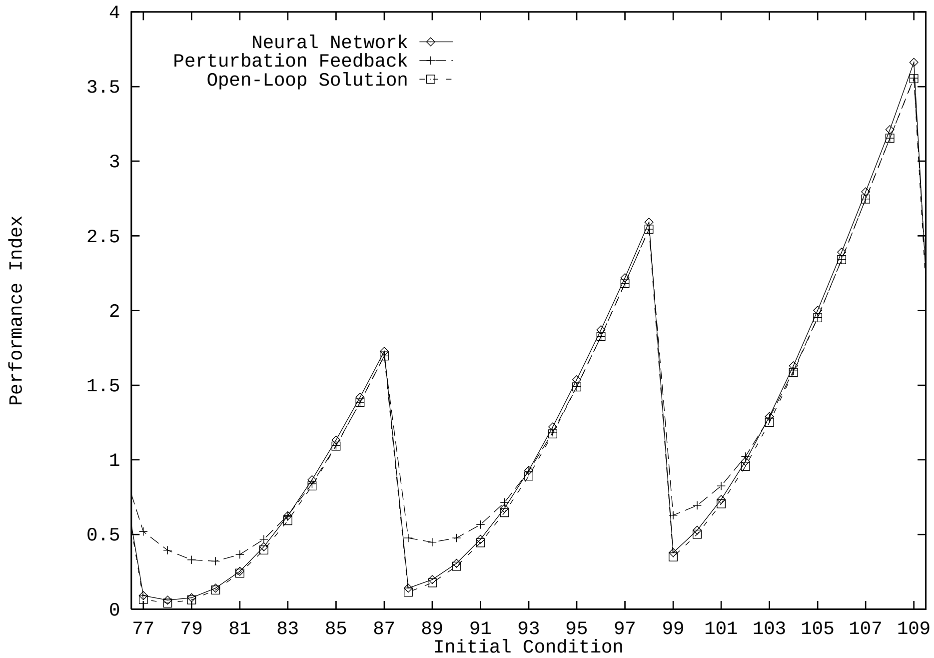

Figure 5.4.9: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b).

where the weighting factors k_1 and k_2 are set to 10 and 0.01 respectively to reflect the relative importance of the terminal state constraint and minimum control effort.

The set of initial conditions Γ is defined by

$$\Gamma \quad = \quad \left\{\boldsymbol{\xi} \, : \, \boldsymbol{\xi}_{1,2} \in [-1,1] \, , \boldsymbol{\xi}_{3,4} = 0\right\}. \tag{5.37}$$

Then the training set of initial conditions Δ is defined as a uniform grid covering Γ,

$$\Delta \quad = \quad \left\{\boldsymbol{\xi} \, : \, \boldsymbol{\xi}_1 = -1 + 0.2i, \quad \boldsymbol{\xi}_2 = -1 + 0.2j, \quad \boldsymbol{\xi}_{3,4} = 0; \quad i,j = 0, \ldots, 10\right\}. \tag{5.38}$$

A $\mathcal{N}^3_{5,50,50,2}$ network is trained by the direct method until it achieves about 3% suboptimality. Note that this neural network controller is time as well as state dependent. This network requires 12 000 cycles through the set of training initial conditions Δ. This takes about 13 days of cpu time. For this problem, the learning rate must initially be set to 10^{-5}, a very small value, due to the extreme sensitivity of the dynamical system to small changes in the neural network's weights. However, despite this extremely small learning rate, the direct training method is able to train the network to the required suboptimality. Since the control for this problem is unbounded, a perturbation feedback controller may also

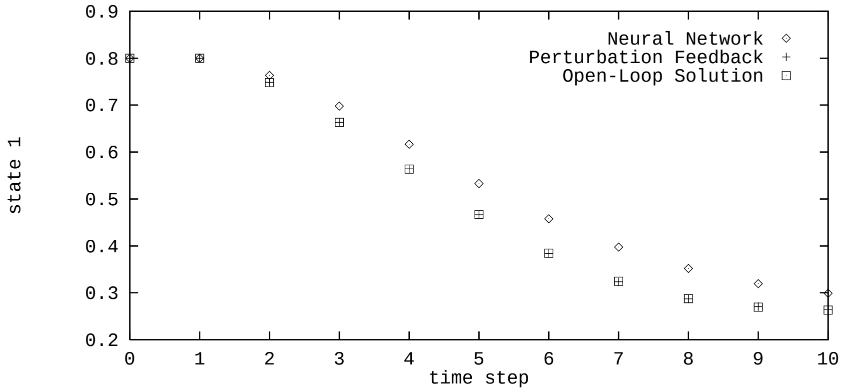

Figure 5.4.10: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b), $\boldsymbol{\xi} \in \Delta$, $\boldsymbol{\xi} \approx [0.7, 0.7, 0, 0]^\top$.

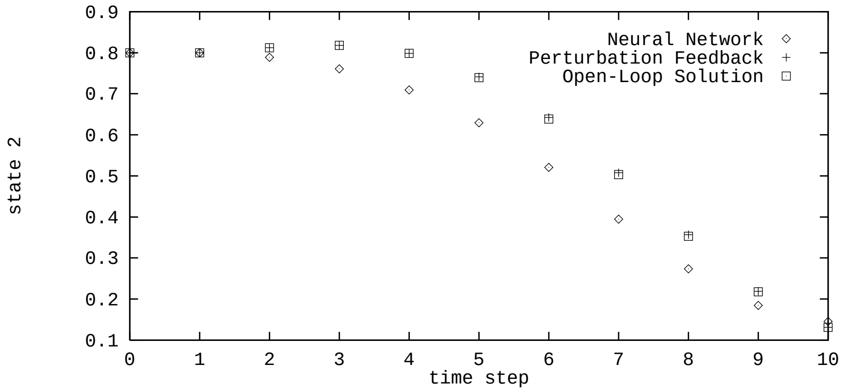

Figure 5.4.11: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b), $\boldsymbol{\xi} \in \Delta$, $\boldsymbol{\xi} \approx [0.7, 0.7, 0, 0]^\top$.

be constructed using a predetermined optimal open-loop trajectory with initial condition $\boldsymbol{\xi} = [0.7, 0.7, 0, 0]^\top$.

Figure 5.4.9 shows the optimal open-loop performance index and the performance index produced by the neural network and perturbation feedback controllers for a subset of the training set of initial conditions Δ defined by $\{\boldsymbol{\xi}^i, i = 77, \ldots, 109 : \boldsymbol{\xi}_1^i = -1 + 0.2(i \,|\, 11), \boldsymbol{\xi}_2^i = -1 + 0.2(i \bmod 11), \boldsymbol{\xi}_{3,4}^i = 0\}$. The closeness of the lines for the neural network controller and the optimal open-loop controller clearly demonstrate that the neural network controller can produce the optimal feedback control over the entire set of training initial conditions Δ. The perturbation feedback controller can also be seen to emulate the optimal open-loop control as long as the initial condition is not too far from

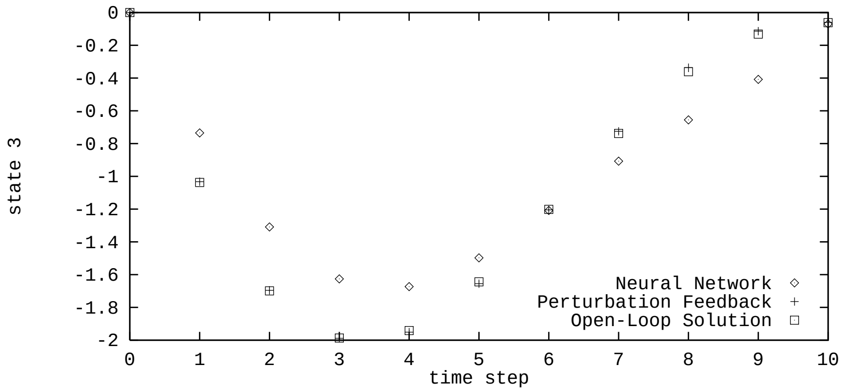

Figure 5.4.12: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b), $\boldsymbol{\xi} \in \Delta$, $\boldsymbol{\xi} \approx [0.7, 0.7, 0, 0]^\top$.

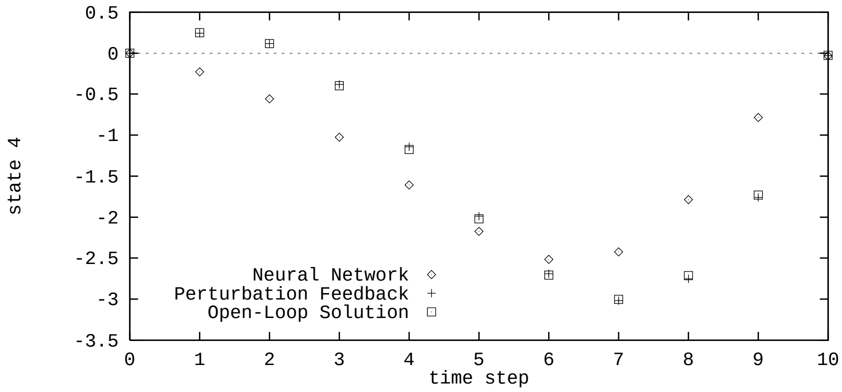

Figure 5.4.13: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b), $\boldsymbol{\xi} \in \Delta$, $\boldsymbol{\xi} \approx [0.7, 0.7, 0, 0]^\top$.

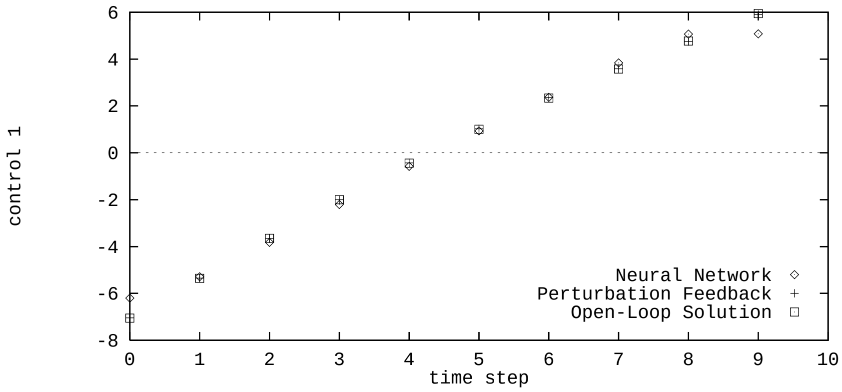

Figure 5.4.14: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b), $\boldsymbol{\xi} \in \Delta$, $\boldsymbol{\xi} \approx [0.7, 0.7, 0, 0]^\top$.

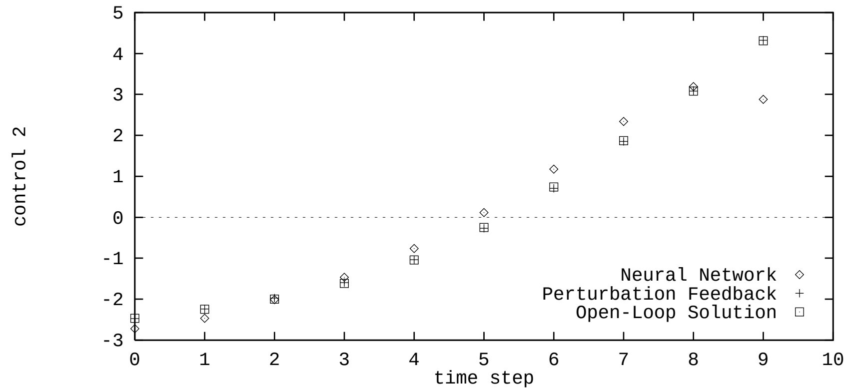

Figure 5.4.15: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b), $\boldsymbol{\xi} \in \Delta$, $\boldsymbol{\xi} \approx [0.7, 0.7, 0, 0]^\top$.

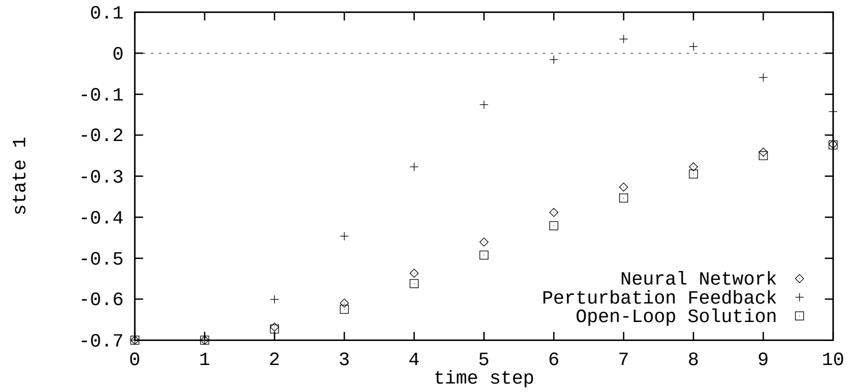

Figure 5.4.16: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b), $\boldsymbol{\xi} \notin \Delta$, $\boldsymbol{\xi} \not\approx [0.7, 0.7, 0, 0]^\top$.

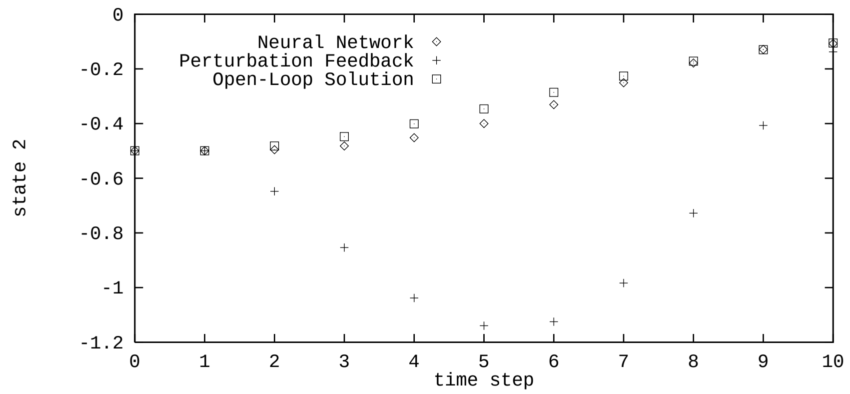

Figure 5.4.17: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b), $\boldsymbol{\xi} \notin \Delta$, $\boldsymbol{\xi} \not\approx [0.7, 0.7, 0, 0]^\top$.

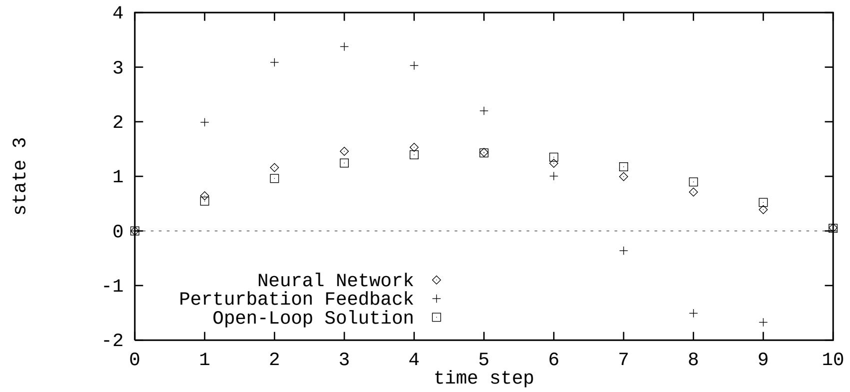

Figure 5.4.18: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b), $\boldsymbol{\xi} \notin \Delta$, $\boldsymbol{\xi} \not\approx [0.7, 0.7, 0, 0]^\top$.

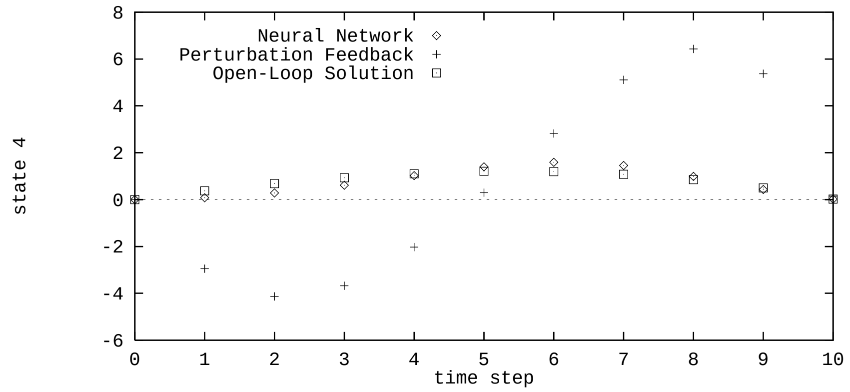

Figure 5.4.19: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b), $\boldsymbol{\xi} \notin \Delta$, $\boldsymbol{\xi} \not\approx [0.7, 0.7, 0, 0]^\top$.

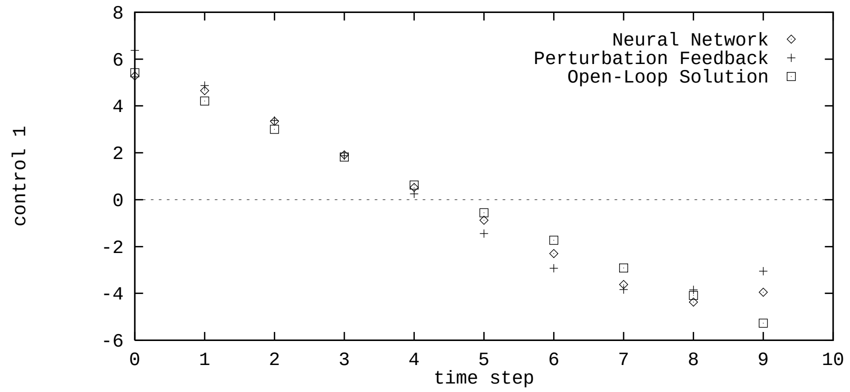

Figure 5.4.20: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b), $\boldsymbol{\xi} \notin \Delta$, $\boldsymbol{\xi} \not\approx [0.7, 0.7, 0, 0]^\top$.

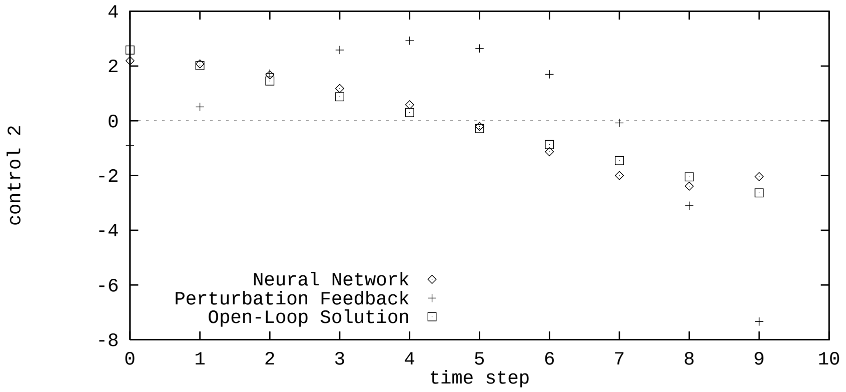

Figure 5.4.21: Neural Network and Perturbation Feedback Controllers v Open-Loop Controller: Problem 3(b), $\boldsymbol{\xi} \notin \Delta$, $\boldsymbol{\xi} \not\approx [0.7, 0.7, 0, 0]^{\top}$.

the nominal initial condition. Figure 5.4.10 to 5.4.15 show the states and controls for a trajectory whose initial condition is in the training set of initial conditions Δ and close to the nominal initial condition. Figure 5.4.16 to 5.4.21 show the states and controls for a trajectory whose initial condition is not in Δ and is far from the nominal initial condition. The performance index for each of the trajectories produced by the neural network controller is within 2% of the optimal open-loop solution. The perturbation feedback controller, on the other hand, achieves an identical performance index to the optimal open-loop solution for the first trajectory, whose initial condition is close to the nominal initial condition, but produces a performance index a large 36% from the optimal open-loop solution for the second trajectory whose initial condition is far from the nominal initial condition. This serves to demonstrate the limitations of perturbation feedback control and to confirm that the set of initial conditions Γ is indeed non-trivial. This clearly shows that even for a problem that is extremely sensitive to the controls, the direct method can train a neural network feedback controller to emulate the optimal feedback control for a non-trivial set of initial conditions Γ.

5.5 Conclusions

The problems presented here clearly demonstrate the effectiveness of both the direct and indirect methods for producing neural network feedback controllers which emulate the optimal feedback control law for a wide variety of non-trivial, non-linear optimal control problems. In the examples considered, the performance index for the neural network controller is only marginally inferior to the optimal open-loop solution over all initial conditions in a significantly large subset of the state space.

A number of important observations can be made from these simulations. Firstly, the neural network controllers trained by the direct method achieve better results in less time, with smaller networks and with less modifications to the weights than those trained by the indirect method. In one problem not included here, a network could not be found which the indirect method could train satisfactorily, whilst the direct method was able to train a relatively small network to 5% suboptimality. Thus, the direct method is preferred over the indirect method, except where constraints make the use of the indirect method necessary. Secondly, the controller trained by the direct method exhibits greater error on the input/output training data Υ, despite achieving superior suboptimality. This suggests that there is more to achieving good results than just learning the input/output nature of the optimal feedback controller. These observations raise some interesting questions about the properties of neural network controllers trained by these training methods. Thirdly, there does not appear to be any way to predict the size and structure of the neural network necessary to emulate the optimal feedback control. This is illustrated in part by the difference in size and structure of the networks that performed satisfactorily for the direct and indirect methods. Fourthly, it is not clear which problems need a time dependent optimal feedback controller. At this stage, only an ad hoc trial and error process determines whether a strictly state dependent optimal feedback control is used or not. Fifthly, for training the neural network controller by the direct method, a very small learning rate must sometimes be used, resulting in much slower convergence, in order to avoid causing the method to become unstable.

There is still a great deal of scope for research concerning these neural network controllers. Simulations for continuous dynamical systems using the direct training method have yet to be attempted and, in view of the benefits of this method, an extension of the direct method which can handle constraints would be extremely desirable. The general stability and robustness of the neural network controllers have yet to be studied in detail. This is necessary, in order to assess their suitability for systems susceptible to parameter variation and noise. In addition, there remains the question of just what size and structure of neural network should be used for a given problem. At present, the network's size and structure is chosen by a painstaking trial and error process. A set of heuristic rules to aid in deciding the neural network size and structure to choose for a given problem would streamline the process of training these neural network controllers.

Appendix: Artificial Neural Networks

The intention in this appendix is not to provide an introduction to artificial neural networks (for this see [9, 14] and many others), but rather to establish some necessary formulae and important theorems. Only the backpropagation neural network is discussed here as this is the neural network structure used throughout this dissertation.

The notation of [16] is adopted to provide a mechanism for describing backpropagation neural networks. $\mathcal{N}^K_{n_0,n_1,\ldots,n_K}$ represents the class of backpropagation neural networks with K layers, layer L containing n_L adalines; n_0 external inputs; and n_K external outputs. See [18] for a definition of adalines. A $\mathcal{N}^K_{n_0,n_1,\ldots,n_K}$ backpropagation neural network can be described mathematically as follows. For each layer of the network, define $\mathbf{z}^L \in \mathbb{R}^{n_L}$ to be the output of layer L. Then $\mathbf{z}^0$ and $\mathbf{z}^K$ represent the external input and output vectors respectively and $\mathbf{z}^L$ can be defined in terms of $\mathbf{z}^{L-1}$ by the recursive relationship

$$\begin{aligned} \mathbf{z}^L &= \mathbf{s}^L(W^L\mathbf{z}^{L-1} + V^L) & \text{(A.1)} \\ &= [s_L(W^L_1\mathbf{z}^{L-1} + V^L_1), s_L(W^{L-1}_2\mathbf{z}^L + V^L_2), \ldots, s_L(W^L_{n_L}\mathbf{z}^{L-1} + V^L_{n_L})]^\top, & \text{(A.2)} \end{aligned}$$

for $L = 1, \ldots, K$, where $W^L \in \mathbb{R}^{n_L \times n_{L-1}}$ and $V^L \in \mathbb{R}^{n_L}$. The output functions $s_L : \mathbb{R} \to \mathbb{R}$ for $L = 1, \ldots, K$ are defined to be monotonicly increasing continuously differentiable sigmoid functions. The sigmoid functions $s_L(\bullet) = \tanh(\bullet)$ or $s_L(\bullet) = [1 + exp(-(\bullet))]^{-1}$ are most commonly used for $L = 1, \ldots, K$, however a linear output function $s_K(\bullet) = (\bullet)$ is sometimes used for the Kth layer of the network. To represent the synaptic weights of the backpropagation neural network, the set $\mathcal{W} = \{W^L_{ij}, V^L_i,\ L = 1, \ldots, K, i = 1, \ldots, n_L, j = 1, \ldots, n_{L-1}\}$ is defined. Now, for an input vector $\mathbf{x}$, the output of the neural network $\Psi(\mathbf{x}; \mathcal{W})$ can be written

$$\begin{aligned} \Psi(\mathbf{x}; \mathcal{W}) &= \mathbf{s}^K(W^K\mathbf{s}^{K-1}(W^{K-1}\mathbf{s}^{K-2}(\cdots\mathbf{s}^2(W^2\mathbf{s}^1(W^1\mathbf{x} \\ &\qquad + V^1) + V^2)\cdots) + V^{K-1}) + V^K). \qquad \text{(A.3)} \end{aligned}$$

With the notation set, it is now possible to derive the gradient of Ψ with respect to both the synaptic weights $\mathcal{W}$ and the input vector $\mathbf{x}$, since this is needed in the main part of the text. By the chain rule, (A.1) and (A.2) we can write

$$\frac{\partial \Psi_l}{\partial W^L_{ij}} = \frac{\partial \Psi_l}{\partial \mathbf{z}^L_i}\frac{\partial \mathbf{z}^L_i}{\partial W^L_{ij}}$$

$$
\begin{aligned}
&= \frac{\partial \Psi_l}{\partial \mathbf{z}_i^L} \frac{\partial \mathbf{s}_i^L(W^L \mathbf{z}^{L-1} + V^L)}{\partial W_{ij}^L} \\
&= \frac{\partial \Psi_l}{\partial \mathbf{z}_i^L} \frac{\partial s_L(W_i^L \mathbf{z}^{L-1} + V_i^L)}{\partial W_{ij}^L} \\
&= \frac{\partial \Psi_l}{\partial \mathbf{z}_i^L} s'_L(W_i^L \mathbf{z}^{L-1} + V_i^L) \mathbf{z}_j^{L-1},
\end{aligned} \tag{A.4}
$$

for $L = 1, \ldots, K$, $i = 1, \ldots, n_L$, $j = 1, \ldots, n_{L-1}$ and $l = 1, \ldots, n_K$. Similarly,

$$
\frac{\partial \Psi_l}{\partial V_i^L} = \frac{\partial \Psi_l}{\partial \mathbf{z}_i^L} s'_L(W_i^L \mathbf{z}^{L-1} + V_i^L). \tag{A.5}
$$

We now require $\partial \Psi_l / \partial \mathbf{z}_i^L$. For the last layer (layer K), since $\Psi = \mathbf{z}^K$, we have

$$
\begin{aligned}
\frac{\partial \Psi_l}{\partial \mathbf{z}_i^K} &= \frac{\partial \Psi_l}{\partial \Psi_i} \\
&= \begin{cases} 1 & \text{if } i = l, \\ 0 & \text{if } i \neq l. \end{cases}
\end{aligned} \tag{A.6}
$$

Then, by a careful application of the chain rule, plus equations (A.1) and (A.2) we derive,

$$
\begin{aligned}
\frac{\partial \Psi_l}{\partial \mathbf{z}_i^L} &= \frac{\partial \Psi_l}{\partial \mathbf{z}_1^{L+1}} \frac{\partial \mathbf{z}_1^{L+1}}{\partial \mathbf{z}_1^L} + \cdots + \frac{\partial \Psi_l}{\partial \mathbf{z}_m^{L+1}} \frac{\partial \mathbf{z}_m^{L+1}}{\partial \mathbf{z}_m^L} \\
&= \sum_{k=1}^{m} \frac{\partial \Psi_l}{\partial \mathbf{z}_k^{L+1}} s'_L(W_k^{L+1} \mathbf{z}^L + V_k^{L+1}) W_{ki}^{L+1}.
\end{aligned} \tag{A.7}
$$

Note that this last equation is valid for $L = 0$, so, since $\mathbf{x} = \mathbf{z}^0$, $\partial \Psi_l / \partial \mathbf{x}_i$ can be derived for $i = 1, \ldots, n_0$:

$$
\frac{\partial \Psi_l}{\partial \mathbf{x}_i} = \sum_{k=1}^{m} \frac{\partial \Psi_l}{\partial \mathbf{z}_k^1} s'_L(W_k^1 \mathbf{z}^L + V_k^1) W_{ki}^1. \tag{A.8}
$$

The one of the primary reasons that backpropagation neural networks have found such popularity in recent years is due to the following theorem published in 1987/88 by a number of people working independently, see [10, 11, 13]. It says, in essence, that any square integrable function can be approximated to arbitrary accuracy by a two layer backpropagation neural network. Square integrable functions include continuous functions, functions with a finite number of discontinuities and many more. This is significant, in particular, for control problems, whose optimal controls often take on solutions of a bang-bang nature, containing, by definition, a finite number of discontinuities.

Theorem A.1 *Given any $\varepsilon > 0$ and $f \in \mathrm{L}^2([0,1]^n, \mathbb{R}^m)$, $\exists N^0 \in \mathbb{N}$ such that $\forall N > N^0$ there is a $\mathcal{N}^2_{n,N,m}$ backpropagation neural network such that*

$$\int_{[0,1]^n} \|f(\mathbf{x}) - \zeta(\mathbf{x}, N)\|^2 d\mathbf{x} \quad < \quad \varepsilon \tag{A.9}$$

where $\zeta(\mathbf{x}, N)$ is the output of the $\mathcal{N}^2_{n,N,m}$ network for the input vector $\mathbf{x}$.

The proof of this theorem is an existence proof only and there is no indication of how many processing elements are needed in the first layer to achieve arbitrary accuracy for a given f and ε. It is also important to note that the theorem does not give any indication of how the correct weights for the network might be derived once there are sufficient processing elements in the first layer. In fact, there is no guarantee that any of the existing learning algorithms will find the correct weights, even if the first layer has sufficient processing elements. However, this theorem says that given enough processing elements and the correct weights, we can approximate any L^2 function as closely as we desire.

Armed with this result, we can forge on with applications of feedforward neural networks, confident that as long as we are dealing with a square integrable function, it can be emulated by a two layer feedforward neural network. However, we do not limit ourselves to two layer networks. It is likely that the number of first layer processing elements necessary to emulate the function to desired accuracy will be prohibitive, and in these cases a network with more layers is generally used.

A.1 The Backpropagation Algorithm

Backpropagation neural networks are commonly trained by the the backpropagation algorithm. This algorithm trains the neural network to emulate input/output pairs. Its performance is measured for a single input/output pair by the square error

$$\varepsilon^2(\mathbf{x}, \mathbf{d}) \quad = \quad \|\Psi(\mathbf{x}; \mathcal{W}) - \mathbf{d}\|^2, \tag{A.10}$$

where $\mathbf{d}$ is the desired output for the input vector $\mathbf{x}$. For a set of input/output pairs the performance of the network is measured by the total square error

$$\begin{aligned} S &= \sum_{i=1}^{n_P} \varepsilon^2(\mathbf{x}^i, \mathbf{d}^i), \\ &= \sum_{i=1}^{n_P} \left\| \Psi(\mathbf{x}^i; \mathcal{W}) - \mathbf{d}^i \right\|^2, \end{aligned} \tag{A.11}$$

where the set contains n_P training points and $\mathbf{d}^i$ is the desired output for the input vector $\mathbf{x}^i$. The backpropagation algorithm modifies the synaptic weights of the backpropagation neural network so that the total square error S is minimised. The algorithm can be stated as follows:

The Backpropagation Algorithm

1. Set all synaptic weights to small random values. This defines $W_{ij}^L(0)$ and $V_i^L(0)$.

2. For each desired input/output training point $(\mathbf{x}^k, \mathbf{d}^k)$ do:

   (a) Present the input vector $\mathbf{x}^k$ to the network.

   (b) Calculate the output vector $\Psi(\mathbf{x}^k; \mathcal{W})$ by (A.3).

   (c) Modify the weights via

$$W_{ij}^L(k+1) \leftarrow \left(W_{ij}^L(k) - \mu \left. \frac{\partial \varepsilon^2}{\partial W_{ij}^L(k)} \right|_{\mathbf{x}^k \mathbf{d}^k} \right) \tag{A.12}$$

$$V_i^L(k+1) \leftarrow \left(V_i^L(k) - \mu \left. \frac{\partial \varepsilon^2}{\partial V_i^L(k)} \right|_{\mathbf{x}^k, \mathbf{d}^k} \right) \tag{A.13}$$

Thus, for the backpropagation algorithm, the gradients $\partial\varepsilon^2/\partial W_{ij}^L$ and $\partial\varepsilon^2/\partial V_i^L$ must be calculated at each training point. Since ε^2 is a function of Ψ, it is a simple matter to derive these. From (A.10) and using the chain rule we have

$$\frac{\partial \varepsilon^2}{\partial W_{ij}^L} = 2 \sum_{l=1}^{n_K} (\Psi_l - \mathbf{d}_l) \frac{\partial \Psi_l}{\partial W_{ij}^L}. \tag{A.14}$$

Similarly,

$$\frac{\partial \varepsilon^2}{\partial V_i^L} = 2 \sum_{l=1}^{n_K} (\Psi_l - \mathbf{d}_l) \frac{\partial \Psi_l}{\partial V_i^L}. \tag{A.15}$$

References

[1] Anderson, B.D.O. and Moore, J.B., *Linear Optimal Control*, Prentice Hall, 1971.

[2] Barto, A.G., Sutton, R.S. and Anderson, C.W., "Neuronlike Adaptive Elements That Can Solve Difficult Learning Problems", *IEEE Trans. on Systems, Man and Cybernetics*, SMC-13:834-846, 1983.

[3] Biswas, S.K. and Ahmed, N.U., "Optimal Feedback Control Of Power Systems Governed By Nonlinear Dynamics", *Optimal Control Applications and Methods*, Vol. 7, 1986, pp. 289-303.

[4] Breakwell, J.V., Speyer, J.L. and Bryson, A.E. Jr., "Optimisation And Control Of Nonlinear Systems Using The Second Variation", *SIAM Journal of Control*, Vol 1, 1963, p. 193.

[5] Bryson, A.E. and Ho, Y., *Applied Optimal Control*, Hemisphere Publishing Corporation, 1975.

[6] Evtushenko, Y.G., *Numerical Optimisation Techniques*, Optimization Softwares, New York, 1985.

[7] Goh, C.J., Edwards, N.J. and Zomaya, A.Y., "Feedback Control Of Minimum-Time Robot Trajectory Using Neural Networks", submitted to *Optimal Control Methods and Applications*, 1991.

[8] Goh, C.J. and Edwards, N.J., "Synthesis Of Discrete-Time Optimal Feedback Controller: The Neural Network Approach", submitted to *Automatica*, 1991.

[9] Hecht-Nielsen, R., *Neurocomputing*, Addison-Wesley Publishing, 1990.

[10] Hecht-Nielsen, R., "Theory Of The Backpropagation Neural Network", *Proc. of the Int. Joint Conf. on Neural Networks*, I, pp. 593-611, IEEE Press, New York, June, 1989.

[11] Irie, B. and Miyake, S., "Capabilities Of Three-Layered Perceptrons", *Proc. of the Int. Conf. on Neural Networks*, I, 641-648, IEEE Press, New York, July, 1988.

[12] Kwakanaak, H. and Sivan, R., *Linear Optimal Control Systems*, Wiley Interscience, 1972.

[13] Lapedes, A. and Farber, R., "How Neural Nets Work", in Anderson, D.Z. [Ed.], *Neural Information Processing Systems (Proc. of the IEEE NIPS Conf., Denver CO, 1987)*, 442-256, Am. Inst. of Physics, New York, 1988.

[14] Lippmann, R.P., "An Introduction To Computing With Neural Nets", *IEEE ASSP Magazine*, April, 1987, pp. 4-22.

[15] Jennings, L.S., Fisher, M.E., Teo, K.L. and Goh, C.J., *MISER3 Optimal Control Software : Theory And User Manual*, EMCOSS Pty Ltd, 1990.

[16] Narendra, K.S. and Parthasarathy, K., "Identification And Control Of Dynamical Systems Using Neural Networks", *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, 1990, pp. 4-27.

[17] Teo, K.L., Goh, C.J. and Wong, K.H., *Unified Computational Approach To Solving Optimal Control Problems*, Pitmann series in Pure and Applied Mathematics, Longman, 1991.

[18] Widrow, B., "30 Years Of Adaptive Neural Networks: Perceptron, Madaline, And Backpropagation", *Proceedings of the IEEE*, Vol 78, No. 9, 1990

Project Proposal - Nathan Edwards (joint Maths / Computer Science)

Supervisors - Dr C.J. Goh, Prof. A.I. Mees and Dr N. Spadaccini.

Title : Implementation of Neural Network Training Algorithms and their Application to Optimal Feedback Control Problems.

The problem of finding optimal controls for a system (eg. torques on a robot arm system to move from position to position in minimum time) can be solved in the open loop case by choosing the initial and terminal states (eg. start and goal configuration of the robot arm) and using numerical software to furnish the solution. However, open loop control is a function of the independent variable only (time or step number) and its performance in achieving the optimal solution and the stability of the system suffers if there is noise and/or parameter variation (eg. elasticity in joints affecting link lengths) in the system. In addition, to use optimal control for a general initial and terminal state (position and velocity etc) we must then generate a numerical solution for every possible initial and terminal state combination.

Closed loop control is dependent on both state and the independent variable (time or step number) which means that it can tolerate noise and parameter variation in the system to a much greater extent and that it can give the optimal control for an arbitrary state. However, closed loop control is impossible to calculate analytically for virtually all but trivial problems, and it cannot be calculated numerically.

This project looks at ways to use neural networks to simulate optimal feedback control and compares them to an existing analytic technique called perturbation feedback control. The neural network techniques will be applied to discrete (M discrete steps) rather than continuous (time as independent variable) systems. This will give an indication of how suitable these methods are and allow an insight into how the continuous case will behave.

Some original programs will be written for training the neural network which will allow a number of different training algorithms to be studied and implemented as well as the flexibility to provide training for our optimal feedback control problem. This will be done in a parallel language to take full advantage of the inherent parallel structure of neural networks.