Least Squares Solutions of the HJB Equation With Neural Network Value-Function Approximators

Yuval Tassa and Tom Erez

***Abstract*—In this paper, we present an empirical study of iterative least squares minimization of the Hamilton–Jacobi–Bellman (HJB) residual with a neural network (NN) approximation of the value function. Although the nonlinearities in the optimal control problem and NN approximator preclude theoretical guarantees and raise concerns of numerical instabilities, we present two simple methods for promoting convergence, the effectiveness of which is presented in a series of experiments. The first method involves the gradual increase of the horizon time scale, with a corresponding gradual increase in value function complexity. The second method involves the assumption of stochastic dynamics which introduces a regularizing second derivative term to the HJB equation. A gradual reduction of this term provides further stabilization of the convergence. We demonstrate the solution of several problems, including the 4-D inverted-pendulum system with bounded control. Our approach requires no initial stabilizing policy or any restrictive assumptions on the plant or cost function, only knowledge of the plant dynamics. In the Appendix, we provide the equations for first- and second-order differential backpropagation.**

***Index Terms*—Differential neural networks (NNs), dynamic programming, feedforward neural networks, Hamilton–Jacoby–Bellman (HJB) equation, optimal control, viscosity solution.**

I. Introduction

THE field of optimal control is concerned with finding the control law which, applied to a given dynamical system, will minimize some performance index, usually the temporal integral of an incurred cost. One way of solving this problem involves the computation of the value function, a measurement of the performance index as a function of space and time. In the continuous case considered here, the value function satisfies a nonlinear partial differential equation called the Hamilton–Jacobi–Bellman (HJB) equation. In the simple case of linear dynamics and quadratic costs, this equation famously reduces to the matrix Riccati equation, which can be accurately solved by analytical or numerical methods.

In the general case, the HJB equation has proven difficult to solve. One reason for this is that value functions are frequently differentiable only almost everywhere, requiring the framework of nonsmooth analysis and viscosity solutions [1], so that a weak sense of solution can first be defined and then shown to exist. Even when a solution is guaranteed to exist, the task of finding it is not made any more tractable. When using approximation schemes, artifacts or hidden extrapolation introduced by an imperfect approximator can couple with the nonlinearity inherent in the minimization operator [see (3)] and produce divergent feedback phenomena [2]. Finally, the computational complexity required to describe the value function grows exponentially with the dimension of its domain ("curse of dimensionality"), limiting the feasibility of an effective approximation. These difficulties in solving the general case have led many research efforts towards finding those special cases for which a suitably designed technique can be proven to always work.

Feedforward neural networks (NNs) provide a generic framework for smooth function approximation, with a sound theoretical foundation [3] and a vast knowledge based on their implementation in terms of architectures and learning algorithms. Previous studies (e.g., [4] and [5]; see Section II) have shown the potential of NN approximation of value functions for special cases.

In this paper, we study a straightforward approach of performing least squares minimization of the HJB residual with an NN approximation of the value function. This approach is attractive because it is simple to implement, requires few *a priori* assumptions and can be applied to almost any problem, including nonlinear problems which frustrate solution by classical methods of control theory. Deemed prone to numerical instabilities and divergent phenomena, this course is rarely taken in practice. Instead of retreating to a limited set of cases where convergence can be guaranteed due to some restrictive property, we wish to propose here two methods to overcome the drawbacks of the unconstrained problem. Intended as a proof-of-concept, this paper provides no formal convergence analysis, but rather presents a series of successful experiments and argues that these methods are likely to promote convergence in most cases. In the rest of this section, we review related work and the required theoretical background. In Section II, we present the details of the framework and our proposed methods for promoting convergence and discuss their motivation. Section III describes the results of numerical experiments with one linear quadratic and two nonlinear optimal control problems, and finally, Section IV contains a conclusion of the presented work and some possible ramifications. The Appendix provides the equations for differential backpropagation in feedforward NNs.

A. Related Work

Research regarding value function approximation has been done in several academic disciplines. When the dynamics are

Manuscript received December 1, 2005; revised September 20, 2006; accepted February 5, 2007.

Y. Tassa is with The Interdisciplinary Center for Neural Computation, The Hebrew University, Jerusalem 91904, Israel (e-mail: tassa@alice.nc.huji.ac.il; yuval.tassa@gmail.com; tassa@pob.huji.ac.il).

T. Erez is with the Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO 63130 USA (e-mail: etom@cse.wustl.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2007.899249

a discrete Markov decision process, the dynamic programming [6] framework provides a rigorous environment for the description and analysis of algorithms. Though extensions to continuous time and space have been made [7], these involve approximating the value function around a single optimal trajectory rather than in a volume of the state space.

In the reinforcement learning (RL) branch of computational learning, the optimal control problem is recast as interaction between an agent (the controller) and an environment (the plant), with the negative cost (usually denoted as $\mathcal{L}$) considered as a reward (denoted r) which the agent strives to maximize. This perspective induces several departures from the control theoretic approach, notably, a bias towards online methods and minimal assumed prior knowledge of the environmental dynamics, e.g., in temporal-difference (TD) methods [8]. This focus on online methods, which might be attributed to the biological plausibility of similar processes taking place in nervous systems [9], biases the application of gradient-based methods towards stochastic gradient descent. In this paper, we show how high condition numbers of the error Hessian give second-order batch methods a distinct computational advantage. Least squares temporal difference (LSTD) and related methods [10] comprise a class of RL algorithms which do use batch learning. The work presented in this paper can be considered an extension of these methods to the continuous case.

When solving continuous problems, most work in RL has concentrated on discretization of the state space and subsequent solution by dynamic programming or related methods [11], probably due to the availability of powerful analytical tools for the discrete case. An exception is Doya's generalization of TD-λ to continuous problems [12] using radial-basis functions. This approach was subsequently applied by Coulom [5] using sigmoidal NNs. In Coulom's work, though arguably a most impressive use of NNs for value function estimation, the learning is computationally intensive and rather unstable, with nonmonotonic convergence. Besides the work of Coulom, mixed results have been reported regarding the application of NNs to value function approximation. After the initial success of Tesauro [13], most papers reported unsatisfactory results [2], [14], [15].

In the control community, the general control problem is usually defined to be that of forcing the output of some dynamical system to follow a given desired signal. By considering the difference between the current and desired states, the problem is recast as bringing this difference to the origin. This assumption, which in optimal control manifests as constraints on the cost function to be zero at the origin and positive elsewhere, conceals the possibilities afforded by other types of problems. For example, Coulom [5] uses a cost function which is linear in the velocity to design "swimmers" whose controllers give rise to limit cycle, rather than stabilizing, dynamics. Another meta-constraint on control theoretic research, due to the applied nature of the field, is the legitimate emphasis on provably convergent algorithms.

Value function approximation has not been a popular method for nonlinear control design, perhaps due to the availability of other powerful methods. An important exception is [16], which uses a Galerkin expansion to construct the value function, but requires computationally intensive integrations. Other current state-of-the-art approximation approaches include adaptive meshing [11] and level sets [17].

Though extensive use of NNs has been performed in the control theory literature [18], they have been used rarely for value function approximation. In cases where they have been used, only special cases of the problem had been considered. Goh [19] used an NN to solve a nonlinear quadratic optimal regulator problem. Several assumptions were made, including a quadratic cost function and the origin being a fixed point of the dynamics, which allow the author to assume a sum-of-squares structure for the value function. Abu-Khalaf and Lewis [4] assumed a convex cost and approximated the value function using a linear function approximator with polynomial basis functions. By accepting these restrictions, they enjoyed the guarantees conferred by a unique minimum. This paper effectively demonstrates how the approaches of [19] and [4] can be extended to general feedforward NNs and general nonlinear optimal control problems. Although provable convergence is lost, general applicability and *de facto* stability are gained.

A necessary step in the use of NNs as approximate solutions to differential equations is the computation of the derivatives of the output of the network with respect to (w.r.t.) its input variables, and the derivatives of these values w.r.t. the weights of the network. Although the equations for computing these quantities have appeared before, either implicitly [20] or explicitly [5], [21], [22], we feel that a clearer derivation is required to do justice to their great usefulness, and include them in the Appendix.

B. HJB Equation

Consider a state vector $\mathbf{x} \in X \subseteq \mathbb{R}^d$ of some dynamical system which evolves according to $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$, with $\mathbf{u} \in U \subseteq \mathbb{R}^m$ being a control signal. Both X and U are assumed to be bounded compact sets, with U convex. Given some scalar cost $\mathcal{L}(\mathbf{x}, \mathbf{u})$, our goal is to find the control law or policy $\mathbf{u} = \mathbf{u}(\mathbf{x}, t)$, which will minimize the cost[1] incurred along the trajectory $\mathbf{x}(t)$ as measured by an integral performance index called the value function

$$V^u(\mathbf{x}) = \int_t^\infty \frac{1}{\tau} e^{-\frac{t'-t}{\tau}} \mathcal{L}\left(\mathbf{x}(t'), \mathbf{u}\left(\mathbf{x}(t')\right)\right) dt'. \tag{1}$$

Putting ∞ as the limit of the integral makes the value and policy independent of t. The exponential discounting function $e^{-(t'-t/\tau)}$ ensures convergence and determines the horizon time scale. The coefficient $1/\tau$ is not essential to the exposition but was added because it normalizes the exponential integrand and fixes V to be in the same units as $\mathcal{L}$. Specifically, for $\tau \to 0$, the normalized exponent shrinks to a δ-function and $V \to \mathcal{L}$. Our goal is to find a policy for which the value function is minimized. This value function, denoted V^*, is called the optimal value function

$$V^*(\mathbf{x}) = \min_{\mathbf{u}(\mathbf{x})} \left[V^u(\mathbf{x})\right].$$

[1]In the formally equivalent continuous reinforcement learning framework, a corresponding reward is maximized.

By direct differentiation w.r.t. t, any function which satisfies (1), including V^*, satisfies the linear differential relation

$$\tau \frac{d}{dt} V(\mathbf{x}) = V(\mathbf{x}) - \mathcal{L}(\mathbf{x}, \mathbf{u})$$

or

$$V(\mathbf{x}) = \tau \dot{V}(\mathbf{x}) + \mathcal{L}(\mathbf{x}, \mathbf{u}) = \tau \frac{\partial V(\mathbf{x})}{\partial \mathbf{x}} \cdot \mathbf{f}(\mathbf{x}, \mathbf{u}) + \mathcal{L}(\mathbf{x}, \mathbf{u}). \quad (2)$$

Now, assume we have found V^* and invoke the minimum principle: Since V^* is already optimal w.r.t. $\mathbf{u}$, any perturbation to $\mathbf{u}$ must necessarily increase the $\mathbf{u}$-dependent right-hand side

$$V^*(\mathbf{x}) = \min_{\mathbf{u}} \left[\tau \frac{\partial V^*(\mathbf{x})}{\partial \mathbf{x}} \cdot \mathbf{f}(\mathbf{x}, \mathbf{u}) + \mathcal{L}(\mathbf{x}, \mathbf{u}) \right]. \quad (3)$$

This nonlinear partial differential equation (PDE) is called the HJB equation.[2]

C. Solutions of the HJB Equation

For linear dynamics and quadratic costs, the HJB equation famously reduces to the Riccati matrix equation of the linear quadratic regulator. For general problems, however, V^* is not differentiable everywhere and the equation does not hold in the classical sense. The usual framework for analyzing nonsmooth solutions to HJB equations is the formalism of viscosity solutions. A way to avoid this is to introduce some stochasticity in the state dynamics $d\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u})dt + \mathbf{n}dt$ with $\mathbf{n}$ denoting a Brownian motion term of covariance $\mathbf{N}$. In this case, the modified HJB equation becomes

$$V^*(\mathbf{x}) - \frac{\tau}{2} \mathrm{tr} \left(\mathbf{N} \cdot \frac{\partial^2 V^*(\mathbf{x})}{\partial \mathbf{x}^2} \right) = \min_{\mathbf{u}} \left[\tau \frac{\partial V^*(\mathbf{x})}{\partial \mathbf{x}} \cdot \mathbf{f}(\mathbf{x}, \mathbf{u}) + \mathcal{L}(\mathbf{x}, \mathbf{u}) \right] \quad (4)$$

which provides a regularizing effect that guarantees that the value function is differentiable everywhere [23].

When attempting to solve the HJB equation in some finite compact volume of space $X \subseteq \mathbb{R}^d$, boundary conditions on ∂X must be considered. For the integral (1) to be defined, the trajectory $\mathbf{x}(t)$ must either remain in X or, upon reaching the boundary, incur some terminal cost. When stabilizing controllers are sought in the control theoretic context, the concept of a Lyapunov function is used to ensure that trajectories are restricted to a domain. In the general nonlinear context, where such techniques are unavailable, boundary conditions must be dealt with explicitly.

D. Value Iteration

Given any value function $V(\mathbf{x})$, a policy which minimizes the RHS of (3)

$$\mathbf{u}^{\mathrm{G}}(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{u}} \left[\tau \frac{\partial V(\mathbf{x})}{\partial \mathbf{x}} \cdot \mathbf{f}(\mathbf{x}, \mathbf{u}) + \mathcal{L}(\mathbf{x}, \mathbf{u}) \right] \quad (5)$$

is called a greedy policy. Assuming $\mathcal{L}(\mathbf{x}, \mathbf{u}) = \mathcal{L}_x(\mathbf{x}) + \mathcal{L}_u(\mathbf{u})$, if $\mathcal{L}_u(\mathbf{u})$ and $\mathbf{f}(\mathbf{u})$ are differentiable, and $\partial \mathcal{L}_u / \partial \mathbf{u}$ invertible, then the minimization in (3) can be reduced to differentiating w.r.t. $\mathbf{u}$, equating to zero and solving for $\mathbf{u}$. If the dynamics are affine in the control $\dot{\mathbf{x}} = \mathbf{f}_x(\mathbf{x}) + \mathbf{f}_u(\mathbf{x}) \cdot \mathbf{u}$,[3] then a sufficient condition for the invertibility of $\partial \mathcal{L}_u / \partial \mathbf{u}$ is the Legendre–Clebsch condition (convexity of $\mathcal{L}_u$)

$$\frac{\partial^2}{\partial \mathbf{u}^2} \left(\frac{\partial V(\mathbf{x})}{\partial \mathbf{x}} \cdot \mathbf{f}(\mathbf{x}, \mathbf{u}) + \mathcal{L}(\mathbf{x}, \mathbf{u}) \right) = \frac{\partial^2}{\partial \mathbf{u}^2} \mathcal{L}_u(\mathbf{u}) > 0.$$

The greedy policy is then given in closed form by

$$\mathbf{u}^{\mathrm{G}} = -\frac{\partial \mathcal{L}_u}{\partial \mathbf{u}}^{-1} \left(\tau \frac{\partial V(\mathbf{x})}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{u}} \right)^{\mathrm{T}}. \quad (6)$$

The process of iteratively forming a value function for a given policy, and then, deriving a new greedy policy with respect to the new value function, is called value iteration or method of approximations, and has been shown to converge when exact measurements are possible [24]. When the object of the iterative improvement is the policy rather than the value function, the process is called policy iteration. In cases such as this one, where the greedy policy is a deterministic closed-form function of the value, the value- and policy-iteration algorithms become nearly indistinguishable.

E. Differential NNs

The Pineda architecture [25] is a generalized topology for feedforward NNs which can generate layered and other topologies as special cases. Given a feedforward network V, an input vector $\mathbf{x}$, and a weight vector $\mathbf{w}$, the Appendix shows how to calculate the following quantities in the Pineda formalism:

$$\begin{array}{ll} V(\mathbf{x}, \mathbf{w}) & \dfrac{\partial}{\partial \mathbf{w}} V(\mathbf{x}, \mathbf{w}) \\ \dfrac{\partial}{\partial \mathbf{x}} V(\mathbf{x}, \mathbf{w}) & \dfrac{\partial}{\partial \mathbf{w}} \dfrac{\partial}{\partial \mathbf{x}} V(\mathbf{x}, \mathbf{w}) \\ \dfrac{\partial^2}{\partial \mathbf{x}^2} V(\mathbf{x}, \mathbf{w}) & \dfrac{\partial}{\partial \mathbf{w}} \dfrac{\partial^2}{\partial \mathbf{x}^2} V(\mathbf{x}, \mathbf{w}). \end{array}$$

The quantities on the left are computed by "forward" propagation while the values on the right are computed by "back" propagation, using intermediate values obtained in the forward pass. We used standard $\tanh$ sigmoids but the nonlinearity can belong to any class of smooth functions [radial basis function (RBF), polynomial, trigonometric, etc.].

F. Least Squares

The approach taken here to approximating a solution to (3) is to minimize the square of the left-hand side (LHS) minus the RHS, called the residual or error. More generally, given an error vector $\mathbf{e} \in \mathbb{R}^n$, which is a function of a weight vector $\mathbf{w} \in \mathbb{R}^q$, the object of least squares minimization is to find $\mathbf{w}_{\mathrm{LS}} = \arg\min_{\mathbf{w}} [\mathbf{e}^{\mathrm{T}} \mathbf{e}]$. The Gauss–Newton approximation takes the first-order expansion of $\mathbf{e}$ in $\mathbf{w}$

$$\mathbf{e}_{\mathbf{w}} = \mathbf{e} + \mathbf{J}\mathbf{w} + O(\mathbf{w}^2)$$

[2] This semiformal derivation is intended to provide an intuition of the problem. For rigorous results, see, e.g., [23].

[3] As is usually the case in mechanical systems.

where $\mathbf{J} = \partial\mathbf{e}/\partial\mathbf{w}$ is the $n \times q$ Jacobian matrix. The square error is then given by the quadratic

$$\mathbf{e}_{\mathbf{w}}^{\mathrm{T}}\mathbf{e}_{\mathbf{w}} = \mathbf{e}^{\mathrm{T}}\mathbf{e} + 2\mathbf{w}^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}\mathbf{e} + \mathbf{w}^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{w} + O(\mathbf{w}^3)$$

which is minimized by $\Delta\mathbf{w}_{\mathrm{GN}} = -(\mathbf{J}^{\mathrm{T}}\mathbf{J})^{-1}\mathbf{J}^{\mathrm{T}}\mathbf{e}$. In the Levenberg–Marquardt variation, the approximated Hessian $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ is conditioned by a scaled identity matrix[4] with a positive factor μ

$$\Delta\mathbf{w}_{\mathrm{LM}} = -(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \mu\mathbf{I})^{-1}\mathbf{J}^{\mathrm{T}}\mathbf{e}. \tag{7}$$

Of the various heuristics for controlling μ during minimization, we use the one suggested in [26].

II. Proposed Method

In this section, we first present a straightforward implementation of the techniques presented previously. Then, we describe some common causes for approximation failure, and afterwards, suggest two methods that circumvent many of these causes.

A. Naive Implementation

The naive approach to least squares minimization of the HJB residual is the following. We take as an input n points $\mathbf{x}_{i=1\ldots n} \in X$, and an NN approximation of the value function $V(\mathbf{x}, \mathbf{w})$ with a weight vector $\mathbf{w}$ initialized using any standard weight initialization scheme (e.g., [27]). Then, the algorithm repeatedly performs Levenberg–Marquardt steps on the squared HJB residual for all points simultaneously until a local minimum is reached.

Algorithm: *Naive* $(\mathbf{x}_{i=1\ldots n} \in X, \mathbf{w}_{\mathrm{initial}})$

1) **repeat**
2) **for** $i \leftarrow 1$ to n
3) **do** $\mathbf{u}_i^{\mathrm{G}} \leftarrow -(\partial\mathcal{L}_u/\partial\mathbf{u})^{-1}(\tau(\partial V(\mathbf{x}_i, \mathbf{w})/\partial\mathbf{x}) \cdot(\partial\mathbf{f}(\mathbf{x}_i)/\partial\mathbf{u}))^{\mathrm{T}}$
4) $\dot{\mathbf{x}}_i \leftarrow \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i^{\mathrm{G}})$
5) $e_i \leftarrow V(\mathbf{x}_i, \mathbf{w}) - \tau(\partial V(\mathbf{x}_i, \mathbf{w})/\partial\mathbf{x})\cdot\dot{\mathbf{x}}_i - \mathcal{L}(\mathbf{x}_i, \mathbf{u}_i^{\mathrm{G}})$
6) $(\partial e_i/\partial\mathbf{w}) \leftarrow (\partial/\partial\mathbf{w})V(\mathbf{x}_i, \mathbf{w}) - \tau(\partial/\partial\mathbf{w})(\partial/\partial\mathbf{x}) \times V(\mathbf{x}_i, \mathbf{w})\cdot\dot{\mathbf{x}}_i$
7) $\mathbf{e} \leftarrow (e_1, e_2, \ldots e_n)^{\mathrm{T}}$
8) $\mathbf{J} \leftarrow ((\partial e_1/\partial\mathbf{w}), (\partial e_2/\partial\mathbf{w}), \ldots (\partial e_n/\partial\mathbf{w}))^{\mathrm{T}}$
9) $\mathbf{w} \leftarrow \mathbf{w} - (\mathbf{J}^{\mathrm{T}}\mathbf{J} + \mu\mathbf{I})^{-1}\mathbf{J}^{\mathrm{T}}\mathbf{e}$
10) $\mu \leftarrow \mu_{\mathrm{new}}$ as in [26]
11) **until** $\Delta(\mathbf{e}^{\mathrm{T}}\mathbf{e}) < \epsilon$
12) **return** the weight vector $\mathbf{w}$

[4]The variation which takes the diagonal of $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ (à la Marquardt) rather than the identity (à la Levenberg) was tried and found to be less stable.

Note that we assume we can compute $(\partial\mathbf{f}(\mathbf{x}, \mathbf{u})/\partial\mathbf{u})$ and $(\partial\mathcal{L}_u/\partial\mathbf{u})^{-1}$ [step 3)], $\mathbf{f}(\mathbf{x}, \mathbf{u})$ [step 4)], and $\mathcal{L}(\mathbf{x}, \mathbf{u})$ [step 5)]. Additionally, apart from the obvious requirement to calculate $V(\mathbf{x}, \mathbf{w})$ and $\partial V(\mathbf{x}, \mathbf{w})/\partial\mathbf{x}$ [steps 3) and 5)], we need the derivatives of these quantities w.r.t. $\mathbf{w}$ [step 6)]. The Appendix explains in detail how these quantities are calculated. Note that since we never actually simulate the plant dynamics, we do not have to determine a time step Δt, nor do we have to deal with the accuracy issues which arise in numerical integration.

B. Distribution of Points and Conditioning

One of our main results is the extremely wide range of sensitivities of the squared error $\mathbf{e}^{\mathrm{T}}\mathbf{e}$ to changes in $\mathbf{w}$. In all our experiments, the condition number of the approximated Hessian $\mathbf{J}^{\mathrm{T}}\mathbf{J} + \mu\mathbf{I}$ regularly exceeded 10^{10}. This fact is made even more dramatic when we realize that the lower eigenvalues of this matrix are dominated by the condition factor μ and can never approach 0. We feel that this explains why methods based on stochastic gradient descent do not perform well.

Another consequence of this sensitivity is the choice of a fixed set of points $\mathbf{x}_i$ during the learning. We initially experimented with resampling the point set before each iteration or collecting the training points along trajectories of a behaving system, but both approaches turned out to introduce too much noise and prevent good convergence. In all our experiments, the point set was drawn from the quasi-random Halton sequence [28], which provided consistently better results than sampling from a uniform distribution, though not by a large margin.

C. Boundary Conditions

The definition of the value function (1) as an integral into the "future" requires special attention at the boundaries of $X \subseteq \mathbb{R}^d$, the compact set over which we wish to approximate V. The points on ∂X, the boundary surface of $\mathbf{X}$, can be divided into the following three distinct categories

- ∂X_1 Regions where $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ points into X for all $\mathbf{u}$. These regions require no special attention since the integral (1) is well defined.
- ∂X_2 Regions where $\mathbf{f}(\mathbf{x}, \mathbf{u})$ points out of X for all $\mathbf{u}$. Because the integral is no longer defined, the HJB equation does not hold and a terminal cost $\mathcal{L}_T(\mathbf{x})$ must be incurred. In practice, this "clamping" constraint $V(\mathbf{x}) = \mathcal{L}_T(\mathbf{x})$ is enforced by scattering n_b points $\mathbf{x}_{j=1\ldots n_b} \in \partial X_2$ for which the error is defined to be $e_j = (V(\mathbf{x}_j) - \mathcal{L}_T(\mathbf{x}_j))$ rather than the usual HJB residual.
- ∂X_3 Regions where $\mathbf{f}(\mathbf{x}, \mathbf{u})$ points into X for some values of $\mathbf{u}$ and out of X for other values. These regions are the most problematic for our purposes and can sometimes be avoided by a careful choice of X. The solution we used[5] was to modify the dynamics so that if $\mathbf{f}(\mathbf{x}, \mathbf{u})$ points out of X, the component perpendicular to ∂X_3 is discarded and only the parallel component is retained. It should be noted that the resulting discontinuities in $\mathbf{f}$ do not invalidate the existence of HJB solutions.

D. Causes of Divergence

There are several difficulties associated with the approximate numerical solution of the HJB equation. First are multiple solutions admitted by the equation. These are either the result of the aforementioned discontinuities allowed in the solution (see [14] for examples) or the nonuniqueness which results when the minimization in (5) is realized as extremization in (6). This type of nonuniqueness is epitomized by the Riccati equation having two solutions, a positive–definite and a negative–definite one (see Section III-A).

Second are the positive feedback effects brought about by V and $\partial V/\partial \mathbf{x}$ having the same sign in (3). Because $\mathbf{u}^{\mathrm{G}}$ is a function of $\partial V/\partial \mathbf{x}$ in (6), greedy value iteration of type proposed here can lead to divergent phenomena. One type is the classical "rattling" effect, where repeated overshooting of the minimum leads to divergence. In our approach, this effect is mostly controlled by the Levenberg–Marquardt parameter μ. Another type of positive feedback phenomena emerges when using a nonlinear function approximator. Local biases in the approximation or "hidden extrapolation," especially at difficult-to-approximate discontinuous boundaries, can lead to false solutions (see, e.g., Fig. 4).

E. Promoting Convergence by Gradual Complexification

The both two methods proposed in the following involve the gradual modification of some parameter during the convergence loop [i.e., before every $\mathbf{repeat} \rightarrow \mathbf{until}$ iteration, steps 2)–10) in the algorithm], so that the problem is initially an "easier" variant which then progressively approaches the full problem. Although their success is not guaranteed and no general convergence proofs are provided, we show how these techniques enable us to address a wide range of problems with considerable success.

F. Modifying the Horizon Time Scale

Our first method involves the modification of the horizon time scale τ. An inspection of (3) leads to an interpretation of τ as the relative weighting coefficient of instantaneous and future costs. As mentioned previously, the introduction of the normalizing coefficient $1/\tau$ in (1) fixes the units of V to be those of $\mathcal{L}$. Specifically for $\tau = 0$, we have $V = \mathcal{L}$ and the problem reduces to simple function approximation. These insights motivate the following method: First, train the NN approximator with $\tau = 0$ until $V(\mathbf{x}, \mathbf{w})$ has converged to $\mathcal{L}$, and then, increase τ in small increments until the desired value is reached.

For linear–quadratic (LQR) problems, where more than one solution to the HJB equation is possible, starting with $\tau = 0$ is equivalent to starting the convergence process from within the positive–definite cone, thus avoiding the negative–definite solution (see Section III-A). Finally, it is important to note that this method is not appropriate for all types of problems. In many cases, especially when the goal is to bring the state into some target set, $\mathcal{L}$ is discontinuous in X while V is continuous. In these cases, the approximation is actually more difficult for small values of τ, and no advantage is gained by its gradual incrementation.

[5]Due to Rémi Munos.

G. Modifying the Stochastic Term

As described in Section I-C, the assumption of stochastic dynamics adds a second derivative term to the HJB equation and has a smoothing effect on the value function. Our second method for promoting convergence involves the gradual reduction of this term.[6] Intuitively, boundaries across which the value function is discontinuous for deterministic dynamics become fuzzy with stochasticity as noise may push the dynamics from one region to another. Besides the dispensation with the formal requirement for "viscosity solutions," smooth functions are easier to approximate and are far less susceptible to the destructive effects of local irregularities, which may form when a continuous function approximates a discontinuous one (as in Gibbs oscillations). In practice, all we need to do to enjoy the benefits of smoothness is to modify step 5) of the algorithm to be

$$e_i \leftarrow V(\mathbf{x}_i, \mathbf{w}) - \tau \frac{\partial V(\mathbf{x}_i, \mathbf{w})}{\partial \mathbf{x}} \cdot \dot{\mathbf{x}}_i - \mathcal{L}\left(\mathbf{x}_i, \mathbf{u}_i^{\mathrm{G}}\right) - \frac{\tau\gamma}{2} \mathrm{tr}\left(\frac{\partial^2 V(\mathbf{x}_i)}{\partial \mathbf{x}^2}\right)$$

which is equivalent to the assumption of white spherical process noise of variance $\mathbf{N} = \gamma \mathbf{I}$. The appropriate derivatives must also be computed when calculating $\partial \mathbf{e}_i / \partial \mathbf{w}$ in step 6)

$$\frac{\partial e_i}{\partial \mathbf{w}} \leftarrow \frac{\partial}{\partial \mathbf{w}} V(\mathbf{x}_i, \mathbf{w}) - \tau \frac{\partial}{\partial \mathbf{w}} \frac{\partial}{\partial \mathbf{x}} V(\mathbf{x}_i, \mathbf{w}) \cdot \dot{\mathbf{x}}_i - \frac{\tau\gamma}{2} \frac{\partial}{\partial \mathbf{w}} \mathrm{tr}\left(\frac{\partial^2 V(\mathbf{x}_i)}{\partial \mathbf{x}^2}\right).$$

It is important to note that while our calculations assume stochastic dynamics and enjoy the benefits of a smooth, differentiable value function, no actual noise is injected anywhere and the algorithm remains deterministic. Another benefit relative to regularization schemes of the Tikhonov, or similar types, is that rather than directly penalizing weights or nonsmoothness as an extra term in the squared error, this type of smoother acts inside the residual term and actually decreases the squared error by orders of magnitude. While these benefits are obvious, we pay a price: A sharp edge in the surface of the value function is often meaningful as a "switching curve," and smoothing it could result in a suboptimal policy, or even the system's failure to achieve its goal. Therefore, we end up with a logic similar to the previous method: Begin with large γ value for improved convergence during the initial approximation iterations, and decrease it gradually through the iterations until it reaches 0 or some other small value.

H. Scheduling and Combining the Methods

For both parameters, we found that parameter modification was best implemented with a sigmoidal schedule, to achieve a converge-track-converge behavior. In the first few iterations, the

[6]The idea of adding this term, hinted at in [29], was suggested to the authors by R. Munos.

values of τ or γ are kept fixed, so that the approximator can converge from its initial conditions to the "easy" problem. Then, the parameter changes gradually while the approximator tracks the resulting changes in the value function. Finally, the parameter remains fixed for the last iterations, allowing maximum convergence for the "hard" problem. The exact rate of change of the parameter during the tracking phase is immaterial, as long as it is slow enough for consecutive realizations of the value function to be good approximations of each other.

When using both methods concurrently, we found that an effective approach is to first let τ increase while γ is held large and constant and then decrease γ only after τ has plateaued. Intuitively, this is because the divergent phenomena described previously are all related to the predictive information contained in V as parameterized by τ. The smoothness benefits arising from γ are used to robustify the convergence while τ is increasing, and should, therefore, be decreased only once τ no longer changes.

III. Experiments

In this section, we demonstrate the proposed method on three optimal control problems. First, we describe the linear quadratic problem, derive the Riccati equation for the discounted horizon case, and discuss the relationship between the horizon time scale and stability. Our solution of a 4-D linear quadratic system is then compared to an accurate solution obtained by standard methods. Next, we present results for the 2-D car-on-the-hill problem. We show how discontinuity of the value function can lead to local divergence, and proceed to use the smoothing parameter γ to resolve this issue. Finally, the 4-D cart-pole problem is described and solved. All experiments were carried out using Pineda-type NNs with $\tanh$ sigmoids, as described in the Appendix.[7] Running time complexity is $O(nq)$ with n the number of sample points and q the number of weights. Experiments were performed on a 3-GHz Pentium 4 CPU.

In both of the nonlinear problems, we follow [12] and [30] and implement a bounded control $|u| < u_{\mathrm{MAX}}$ by using the convex cost function

$$\mathcal{L}_u(u) = c \int_0^u \tanh^{-1}\left(\frac{y}{u_{\max}}\right) dy$$

which, inserted in (6), results in the control law

$$u^{\mathrm{G}} = -u_{\max} \tanh\left(\frac{\tau}{c}\frac{\partial V(\mathbf{x})}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}\right)^{\mathrm{T}}. \qquad (8)$$

We use the terminology of reinforcement learning (reward maximization) rather than control (cost minimization), when doing so aids with the comparison to previous work.

A. Linear Quadratic System

Let us derive the discounted Riccati equation. For linear dynamics $\mathbf{f}(\mathbf{x},\mathbf{u}) = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$ and quadratic costs $\mathcal{L}(\mathbf{x},\mathbf{u}) = \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{u}^{\mathrm{T}}\mathbf{R}\mathbf{u}$ with $(\mathbf{A},\mathbf{B})$ stabilizable, $\mathbf{R} > 0$ and $\mathbf{Q} \geq 0$, the value function is itself quadratic $V(\mathbf{x}) = (1/\tau)\mathbf{x}^{\mathrm{T}}\mathbf{S}\mathbf{x}$. Premultiplication by $1/\tau$, which amounts to a rescaling of $\mathbf{S}$, is added for easier comparison to standard notations. Inserting $\mathbf{f}$ and $\mathcal{L}$ in (6), the greedy policy is $\mathbf{u}^{\mathrm{G}} = -\tau\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{S}\mathbf{x}$. Substituting in (2), we have

$$\frac{1}{\tau}\mathbf{x}^{\mathrm{T}}\mathbf{S}\mathbf{x} = 2\mathbf{x}^{\mathrm{T}}\mathbf{S}(\mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{S}\mathbf{x}) + \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} + \mathbf{x}^{\mathrm{T}}\mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{R}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{S}\mathbf{x}.$$

Differentiating twice w.r.t. $\mathbf{x}$, dividing by two, and rearranging

$$\left(\mathbf{A}^{\mathrm{T}} - \frac{1}{2\tau}\mathbf{I}\right)\mathbf{S} + \mathbf{S}\left(\mathbf{A} - \frac{1}{2\tau}\mathbf{I}\right) - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{S} + \mathbf{Q} = 0$$

By comparison to the usual Riccati equation $\mathbf{A}^{\mathrm{T}}\mathbf{S} + \mathbf{S}\mathbf{A} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{S} + \mathbf{Q} = 0$, we see that a discounted linear quadratic system is equivalent to a nondiscounted system with a friction-like damping factor of $-(1/2\tau)\mathbf{I}$ added to the dynamics. The Riccati equation, essentially a quadratic equation, admits two solutions, the positive–definite "correct" solution, and a destabilizing negative–definite solution. This fact serves to motivate the method of increasing τ during the convergence. For a small enough τ, the dynamical system is extremely stable and any initial guess value function will provide a stable control. As τ is increased, the stabilizing controller found at the previous stages serves as "scaffolding" for the current stage. This is quite similar to other techniques [4], [24], [31], where the preexistence of a stabilizing controller is a condition for its further improvement. Another way of showing the same effect is to rescale $\mathbf{S} \leftarrow \tau\mathbf{S}$ and to see that $\tau = 0 \Rightarrow \mathbf{S} = \mathbf{Q}$, which is a positive–definite initial condition.

We experimented with the 4-D linear quadratic system given by

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -2 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad \mathbf{Q} = \mathbf{I}_4 \quad \mathbf{R} = 1.$$

For $u = 0$, this is a simple energy conserving model of two masses connected by springs. We generated 1000 quasi-uniform points in the ellipsoid of unit mechanical energy. Applying the *Naive* algorithm over these points to the nondiscounted equation ($\tau = \infty$) using a 93-weight NN, we got mixed results. In Fig. 1, we see one run converging to the "correct" positive–definite solution, another to the negative–definite solution, and yet another to no solution at all. Gradually increasing τ from 1 to ∞, as described, always resulted in correct convergence.

B. Car on the Hill

The car-on-the-hill problem, described in [32] and investigated in [11], is a 2-D optimal control problem. A "car" is postulated to roll[8] on a hill of shape

$$H(x) = \begin{cases} x^2 + x, & \text{if } x < 0, \\ x/\sqrt{1+5x^2}, & \text{if } x \geq 0. \end{cases}$$

[7] MATLAB code for implementing differential backpropagation is available at www.alice.nc.huji.ac.il/~tassa/

[8] We make the hairsplitting note that the dynamical equations, used in this paper to allow comparison to previous work and originally given in [32], ignore the centripetal force and are not "correct" in the Newton-mechanical sense.

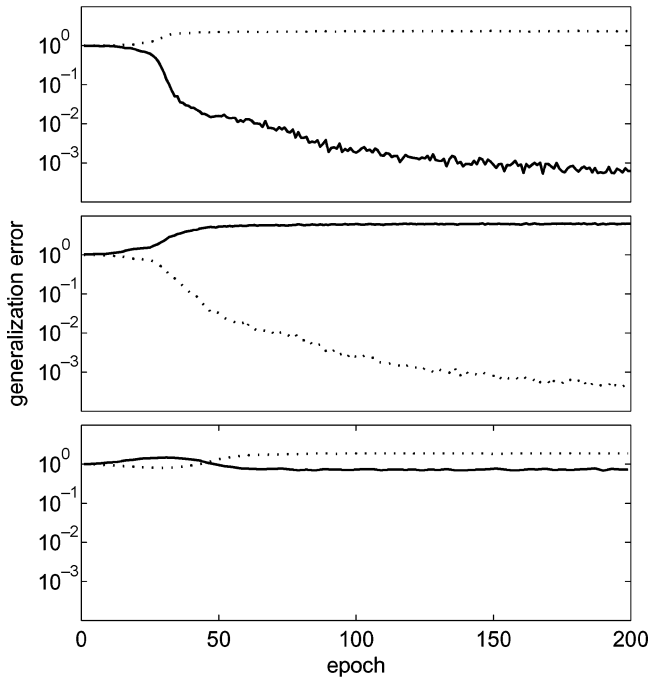

Fig. 1. Performance of three sample runs of the naive algorithm on the linear quadratic system with $\tau = \infty$. At every iteration, we measure the difference between approximated and analytical values over a set of 1000 random points generated anew. Solid (dotted) lines denote the difference between the approximated function and the positive–definite (negative–definite) analytical solution, computed with standard methods. The three figures show a converging, anticonverging, and misconverging run.

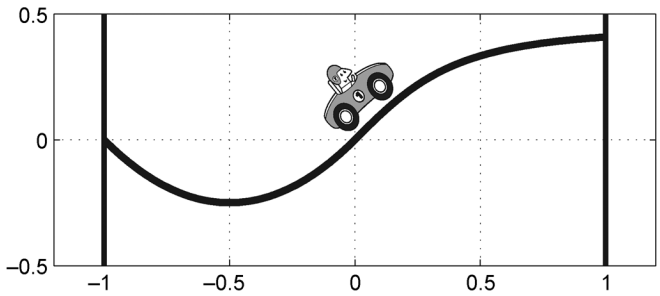

Fig. 2. Car-on-the-hill optimal control problem. The reward is identically zero inside the state space. Terminal rewards of -1 and $1 - \dot{x}/2$ are incurred upon reaching the left and right edges of the state space, respectively.

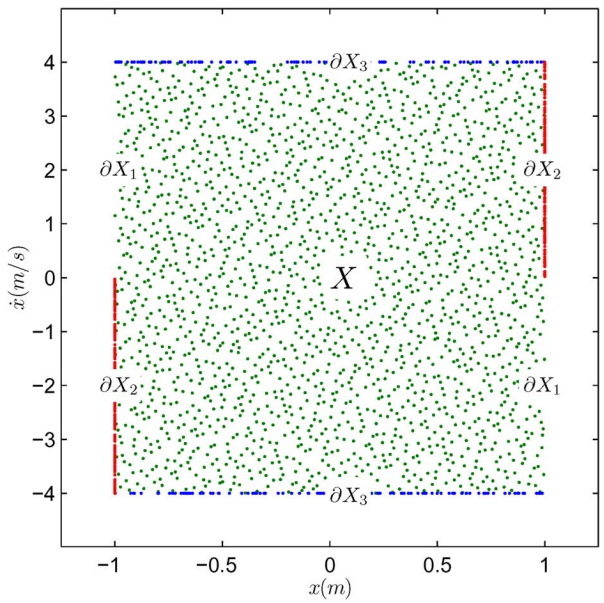

Fig. 3. Illustration of the points where the HJB residual was measured and minimized. Clamping constraints were applied to points on the "outbound" ∂X_2 by minimizing $(V(\mathbf{x}) - r(x))^2$. For points on ∂X_3, the dynamics were modified to never point outside the state space.

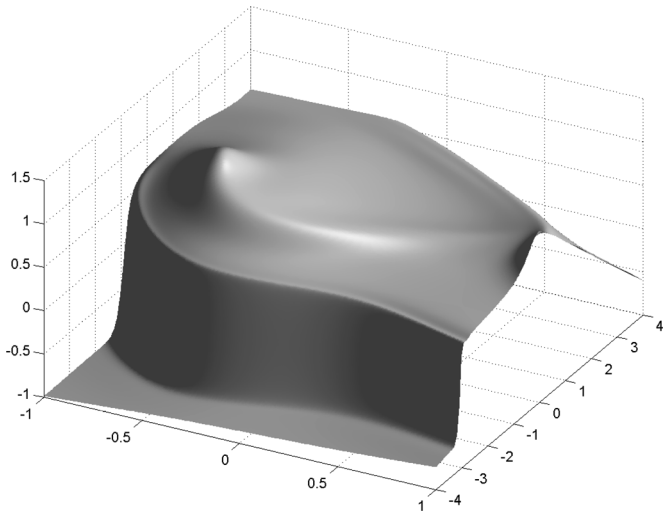

Fig. 4. Local divergence of the learning. The bump in the value function close to $(-0.5\ 0)$ begins as a local extrapolation near the discontinuous boundary and migrates to this position.

The stated goal of reaching the top of the hill with minimal velocity is achieved by setting terminal rewards of $r(x = -1) = -1$ and $r(x = 1) = 1 - \dot{x}/2$ on the left and right edges of the state space, respectively (Fig. 2), and $r(|x| < 1) = 0$ everywhere else. A discrete control space $U = \{-4, 4\}$ is well approximated by using the bounded control (8) with $u_{\max} = 4$ and $c = 10^{-4}$. The volume of state space where we solve this problem is $X = \{(x, \dot{x}) \in \mathbb{R}^2 : |x| < 1_m, |\dot{x}| < 4\text{m/s}\}$. We used three sets of points to find a solution, as shown in Fig. 3. First, we placed 2000 points evenly spread across X, using the pseudorandom Halton sequence. Next, we placed 400 points on the "outbound" $\partial X2$ ($x = \pm 1$ and, respectively, $\dot{x} \lesseqgtr 0$), where any trajectory would inevitably fall out of X. At this points, $V(\mathbf{x}) = r(x)$ was enforced. Finally, we placed 200 points on $\partial X3$, where $|x| < 1$ but $\dot{x} = \pm 4$. In these points, the dynamics were modified by limiting the maximal speed of the car to 4 m/s, a manipulation to the effect of "clipping" the outbound dynamics to be tangent to ∂X. The value function which solves this problem has a discontinuous ridge and a nondifferentiable ridge which make the problem difficult for essentially smooth function approximators like ours. Specifically, when running the *Naive* algorithm, a Gibbs-oscillation-type phenomenon at the discontinuous boundary was found to sometimes evolve into a false stationary point on the $\dot{x} = 0$ line (Fig. 4).

Since the reward function of this problem is discontinuous, setting $\tau = 0$ (and, consequently, $V = r$) would present the

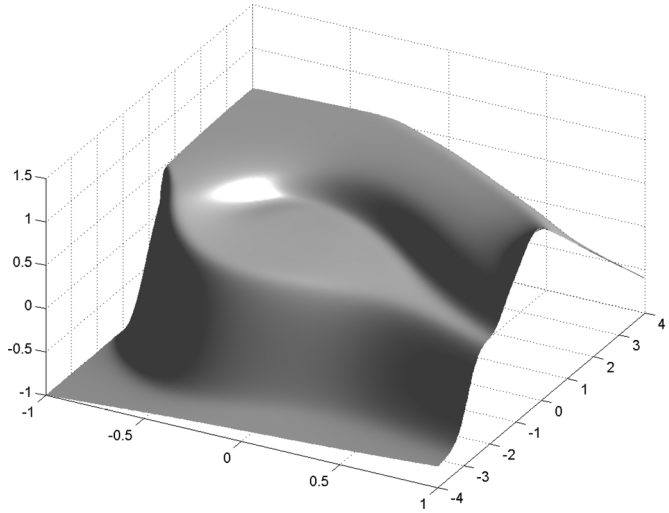

Fig. 5. Smooth value function obtained for $\gamma = 0.02$. We note that the HJB residual for this function was smaller by more than an order of magnitude than for the "sharp" solution.

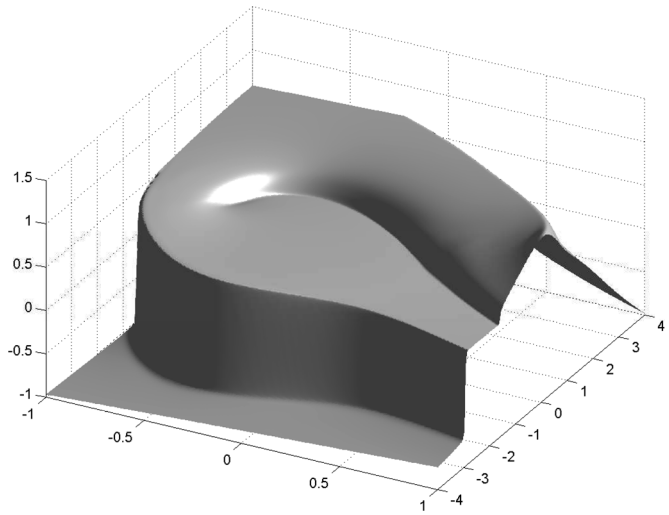

Fig. 6. Approximation of the optimal value function for the car-on-the-hill problem. Compare to [14, Figs. 3 and 4] (available at www.cmap.polytechnique.fr/~munos).

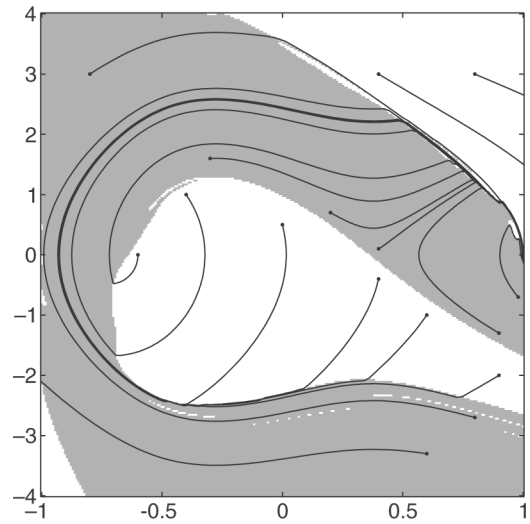

Fig. 7. Approximate optimal policy obtained for the car-on-the-hill problem. Gray and white areas indicate the maximum and minimum control values, respectively. Lines indicate some trajectories integrated using this policy. Compare to [11, Fig. 4].

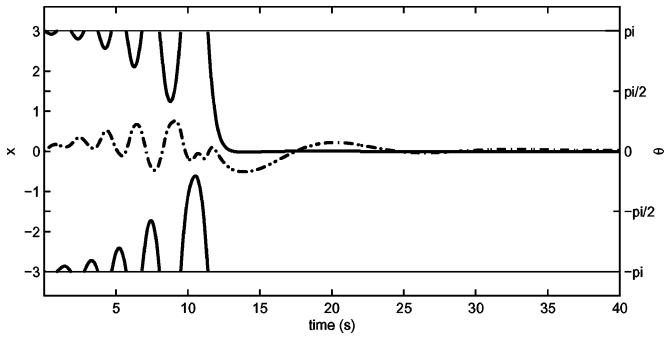

Fig. 8 Time course of the swing-up trajectory for the cart-pole dynamical system, starting from the motionless hanging-down position in the center of the track. The solid line denotes the angle of the pole and the dashed line denotes the position of the cart. Note that the horizon time scale τ is only 1.5 s, far shorter than the total swing-up time.

NN with an unpleasant challenge, and therefore, we are limited in this case to working with the smoothing term only. The smooth value function obtained by setting $\gamma = 0.02$ for all iterations (Fig. 5) has a small HJB residual and is quite immune to the local divergence described previously. By letting γ decrease from 10^{-1} to $2 \cdot 10^{-4}$, we end up with a "sharp" value function (Fig. 6), without exposing ourselves to the danger of local misextrapolation. A typical simulation usually takes about 100 iterations to converge in about 10-min time. The value and policy of Figs. 6 and 7, which were generated with an 800 weight NN, are comparable to the nearly optimal discretization-based solutions in [14], which are described by 66 000 parameters, and appear to be far better than the NN solution therein.

C. Cart Pole Swing-Up

Last, we demonstrate our algorithm on the 4-D system known as the cart-pole dynamical system in the RL community and the inverted-pendulum in the control community, as described in [5] and elsewhere. It consists of a mass (the cart) on a 1-D horizontal track to which another mass (the pole) is attached. The pole swings freely under the effect of gravity. The controller may apply force to the cart in order to achieve the goal of swinging the pole up and then stabilizing it over the cart in the center of the track. The volume of state space where we solve the problem is $X = \{(x, \dot{x}, \theta, \dot{\theta}) \in \mathbb{R}^4 : |x| < 3\text{ m}, |\dot{x}| < 5\text{ m/s}, |\theta| < \pi, |\dot{\theta}| < 10\text{ rad/s}\}$. We used a reward function of $r(x, \theta) = \cos(\theta) - (1/9)x^2$, cart mass 1 kg, pole length 1 m, and pole mass 0.1 kg. The parameters of (8) were $u_{\max} = 3$ N and $c = 0.01$, and the maximal horizon time scale was $\tau = 1.5$ s. The learning took place over 10 000 points generated quasi-uniformly across the entire volume, and another 500 points for clamping constraints at exit regions from the state space. Here, we used both the increase of τ and the decrease of γ. As described earlier, we first increased τ with a high constant value of γ and then decreasing γ after τ had plateaued. Using a 1049-weight NN, a typical simulation took about 250

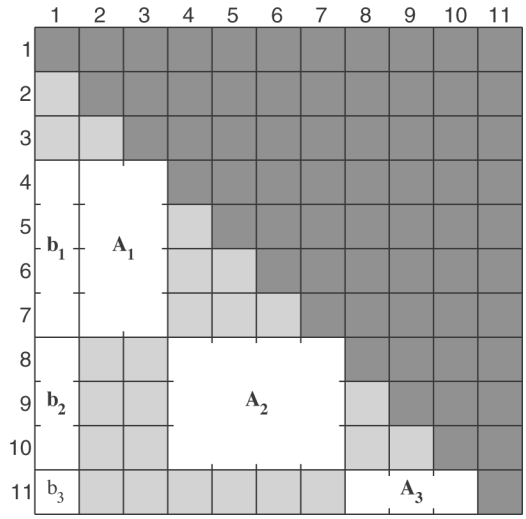

Fig. 9. Pineda weight matrix W for $q = 11$ neurons. Dark-gray squares indicate feedback (recurrent) connections, unused here. Light-gray subdiagonal squares indicate the maximally connected topology enabled by the formalism, of these, indicated by white, are the weights of a regular two hidden-layer network with two input neurons, a four-layer, a three-layer, and a scalar linear output, which would usually be written as $y(\mathbf{x}) = \mathbf{A}_3\sigma(\mathbf{A}_2\sigma(\mathbf{A}_1\mathbf{x}+\mathbf{b}_1)+\mathbf{b}_2)+b_3$. As described in the text, the first neuron is the bias neuron with constant output 1. In all of our experiments, the topology used was the maximal topology minus the weights $\mathbf{W}_{ij}$, for which $j + q + 1 < 2i$.

iterations to converge in about 4 h. In Fig. 8, we show the solution trajectory starting from the difficult position of the pole hanging straight down in the middle of the track with no linear or angular velocity. From this position, the controller must jiggle the cart for a while to accumulate energy in the pole, then swing the pole up and stabilize it around the origin. Quite remarkably, the swing-up process requires almost 30 s to complete which is considerably longer than the horizon time scale $\tau = 1.5$ s.

IV. Conclusion

In this paper, we have shown how using batch least squares minimization of the squared HJB residual is a simple and effective method for solving nonlinear optimal control problems. As a preliminary study, this paper can be extended in numerous ways. The two methods proposed here can probably be put to good use in many other numerical algorithms, and perhaps also in an analytical context. The use of fixed, uniformly distributed points over which the residual is minimized is an obvious place for improvement. A more disciplined approach would be to use some statistically sound method like Kalman filtering to sequentially estimate the update vector $\Delta\mathbf{w}$ at randomly generated points, and accept an update only when the estimate reaches some prescribed confidence threshold. Another approach would be to use advanced stochastic methods like those proposed in [33]. Recent advances in other types of function approximation schemes for the solution of differential equations, such as support vector regression [34] seem promising. Our hope is that by using these or other techniques, even more difficult nonlinear optimal control problems, such as those ostensibly solved by biological nervous systems, might soon become accessible.

Appendix I

The measurement and backpropagation w.r.t. differential quantities in NNs is a rather esoteric art. First proposed indirectly in [20], their description and formulation reappears, apparently independently, every several years [5], [21], [22]. We believe these approximators can find good use in many fields and, therefore, give their explicit formulation, up to the second order. The Pineda feedforward topology, first described in [25] and popularized by [35], is a generalization of classical layered topologies and allows for a maximally connected feedforward topology. Indexing the q neurons of the network in an order allowing feedforward computation, i.e., the ith neuron depending only on neurons $j < i$, we arrange the weights in a strictly lower triangular $q \times q$ matrix $\mathbf{W}$, the weight W_{ij} denoting the connection from neuron j to neuron i.

A. Regular Forward Propagation

For an input vector $\mathbf{x} \in \mathbb{R}^d$, we fix the outputs of the first $d + 1$ neurons to be the augmented input vector $\mathbf{y}_{1\ldots(d+1)} = (1, x_1, \ldots, x_d)^{\mathrm{T}}$, set $W_{ij} = 0$ for $i \leq (d + 1)$, and then, progressively compute for $i = (d + 2)\ldots q$

$$a_i = \sum_{j<i} W_{ij} y_j, \qquad y_i = \sigma(a_i) \tag{9}$$

where a_i is the input to the ith neuron, and σ the nonlinearity. Here, we assume the scalar output of the network to be the value of the last neuron, with no squashing nonlinearity to avoid bounding the outputs: $F(\mathbf{x}, \mathbf{w}) = y_q = a_q$. By making some of the weights zero, or simply ignoring them in the calculations, any topology can be easily implemented, e.g., the W matrix of a layered network will have a chain of blocks below the diagonal (see Fig. 9).

B. Regular Backpropagation

When calculating $\partial F/\partial W_{ij}$, we first compute $\delta_i \equiv (\partial F/\partial a_i)$ in reverse order, i.e., $\delta_q = 1$, and then, for $i = (q - 1)\ldots 1$

$$\delta_i = \sigma'(a_i) \sum_{j>i} W_{ji}\delta_j \tag{10}$$

and then

$$\frac{\partial F}{\partial W_{ij}} = \frac{\partial F}{\partial a_i}\frac{\partial a_i}{\partial W_{ij}} = \delta_i y_j.$$

Note that in (9) we sum over the ith row of W, the weights into neuron i, and in (10), over the ith column, the weights out of neuron i.

C. First-Order Differential Propagation

We use the superscript $\mu = 1\ldots d$ to denote $\partial/\partial x_\mu$, differentiation w.r.t. the μth coordinate of the input vector. The initial input to the derivative network $\mathbf{y}^{\mu}_{1\ldots(d+1)} =$

$(0,0,\ldots,1,\ldots,0)^{\mathrm{T}}$ is a vector of all zeros with 1 in the $(\mu+1)$th place. Then, $(\partial/\partial x_\mu)F(\mathbf{x},\mathbf{w})$ is given by

$$a_i^\mu = \sum_{j<i} W_{ij} y_j^\mu \qquad y_i^\mu = a_i^\mu \sigma'(a_i).$$

The backpropagation of the quantities $(\partial/\partial W_{ij})(\partial/\partial x_\mu)F(\mathbf{x},\mathbf{w})$ is then given by first setting $\delta_q^\mu = 0$ and computing in reverse order

$$\delta_i^\mu = \sum_{j>i} W_{ji}\left(\delta_j^\mu \sigma'(a_i) + \delta_j \sigma''(a_i) a_i^\mu\right)$$

and then

$$\frac{\partial}{\partial W_{ij}}\frac{\partial F}{\partial x_\mu} = \frac{\partial}{\partial x_\mu}\left(\frac{\partial F}{\partial a_i}\frac{\partial a_i}{\partial W_{ij}}\right) = \delta_i^\mu y_j + \delta_i y_j^\mu.$$

D. Second-Order Differential Propagation

The initial input to the second-order derivative network $\mathbf{y}_{1\ldots(d+1)}^{\mu\nu} = (0,0,\ldots 0)^{\mathrm{T}}$ is now a $d+1$ vector of all zeros. Then, $(\partial/\partial x_\mu)(\partial/\partial x_\nu)F(\mathbf{x},\mathbf{w})$ is given by

$$a_i^{\mu\nu} = \sum_{j<i} W_{ij} y_j^{\mu\nu} \qquad y_i^{\mu\nu} = a_i^{\mu\nu}\sigma'(a_i) + a_i^\mu a_i^\nu \sigma''(a_i).$$

The backpropagation of the quantities $(\partial/\partial W_{ij})(\partial/\partial x_\mu)(\partial/(\partial x_\nu)F(\mathbf{x},\mathbf{w})$ is given by setting $\delta_q^{\mu\nu} = 0$ and then

$$\delta_i^{\mu\nu} = \sum_{j>i} W_{ji}\left(\delta_j^{\mu\nu}\sigma'(a_i) + \delta_j^\mu \sigma''(a_i)a_i^\nu + \delta_j^\nu \sigma''(a_i)a_i^\mu + \delta_j \sigma''(a_i)a_i^{\mu\nu} + \delta_j \sigma'''(a_i)a_i^\mu a_i^\nu\right)$$

and then

$$\frac{\partial}{\partial W_{ij}}\frac{\partial}{\partial x_\mu}\frac{\partial}{\partial x_\nu}F = \delta_i^{\mu\nu} y_j + \delta_i^\mu y_j^\nu + \delta_i^\nu y_j^\mu + \delta_i y_j^{\mu\nu}.$$

ACKNOWLEDGMENT

The authors would like to thank M. Margaliot and R. Munos for their helpful comments.

REFERENCES

[1] M. Crandall and P. Lions, "Viscosity solutions of Hamilton-Jacobi equations," *Trans. Amer. Math. Soc.*, vol. 277, 1983.

[2] J. A. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D. S. Touretzky, and T. K. Leen, Eds. Cambridge, MA: MIT Press, 1995, pp. 369–376.

[3] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Netw.*, vol. 2, pp. 183–192, 1989.

[4] M. Abu-Khalaf and F. L. Lewis, "Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach," *Automatica*, vol. 41, no. 5, pp. 779–791, 2005.

[5] R. Coulom, "Reinforcement learning using neural networks, with applications to motor control," Ph.D. dissertation, Institut National Polytechnique de Grenoble, Grenoble, France, 2002.

[6] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA: Athena Scientific, 1995.

[7] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming*. New York: Elsevier, 1970.

[8] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, pp. 9–44, 1988.

[9] W. Schultz, P. Dayan, and P. R. Montague, "A neural substrate of prediction and reward," *Science*, vol. 275, pp. 1593–1599, 1997.

[10] M. G. Lagoudakis, R. E. Parr, and M. L. Littman, "Least-squares methods in reinforcement learning for control," in *Proc. 2nd Hellenic Conf. Artif. Intell.*, 2002, vol. 2308, pp. 249–260.

[11] R. Munos and A. W. Moore, "Variable resolution discretization for high-accuracy solutions of optimal control problems," in *Proc. Int. Joint Conf. Artif. Intell.*, 1999, pp. 1348–1355.

[12] K. Doya, "Temporal difference learning in continuous time and space," in *Advances in Neural Information Processing Systems*, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds. Cambridge, MA: MIT Press, 1996, vol. 8.

[13] G. Tesauro, "Practical issues in temporal difference learning," in *Advances in Neural Information Processing Systems*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. San Mateo, CA: Morgan Kaufmann, 1992, vol. 4, pp. 259–266.

[14] R. Munos, L. Baird, and A. Moore, "Gradient descent approaches to neural net-based solutions of the Hamilton-Jacobi-Bellman equation," in *Proc. Int. Joint Conf. Neural Netw.*, 1999, pp. 1316–1323.

[15] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Proc. 1993 Connectionist Models Summer School*, M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, Eds., 1993, pp. 255–263.

[16] R. Beard and T. McLain, "Successive galerkin approximation algorithms for nonlinear optimal and robust control," *Proc. Int. J. Control: Special Issue Breakthroughs Control Nonlinear Syst.*, vol. 71, no. 5, pp. 717–743, 1998.

[17] J. A. Sethian, *Level Set Methods and Fast Marching Methods*. Cambridge, U.K.: Cambridge Univ. Press, 1999.

[18] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 4–27, Mar. 1990.

[19] C. J. Goh, "On the nonlinear optimal regulator problem," *Automatica*, vol. 29, no. 3, pp. 751–756, 1993.

[20] P. Simard, B. Victorri, Y. LeCun, and J. Denker, "Tangent prop—A formalism for specifying selected invariances in an adaptive network," in *Neural Information Processing Systems*, J. M. R. Lippman and S. J. Hanson, Eds. San Mateo, CA: Morgan Kaufmann, 1992, vol. 4.

[21] J. W. Lee and J. H. Oh, "Hybrid learning of mapping and its Jacobian in multilayer neural networks," *Neural Comput.*, vol. 9, pp. 937–958, 1997.

[22] R. Masuoka, "Neural networks learning differential data," *IEICE Trans. Inf. Syst.*, vol. E83-D, no. 6, pp. 1291–1300, 2000.

[23] W. Fleming and H. Soner, *Controlled Markov Processes and Viscosity Solutions*. New York: Springer-Verlag, 1993.

[24] G. Saridis and C. S. Lee, "An approximation theory of optimal control for trainable manipulators," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-9, no. 3, pp. 152–159, Mar. 1979.

[25] F. Pineda, "Generalization of back-propagation to recurrent neural networks," *Phys. Rev. Lett.*, vol. 19, no. 59, pp. 2229–2232, 1987.

[26] M. S. Bazarraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*. New York: Wiley, 1993.

[27] D. H. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural network by choosing initial values of the adaptive weights," in *Proc. 1st IEEE Int. Joint Conf. Neural Netw.*, 1990, vol. 3, pp. 21–26.

[28] J. H. Halton, "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals," *Numerische Mathematik*, vol. 2, pp. 84–90, 1960.

[29] K. Doya, "Reinforcement learning in continuous time and space," *Neural Comput.*, vol. 12, no. 1, pp. 219–245, 2000.

[30] S. E. Lyshevski and A. U. Meyer, "Control system analysis and design upon the Lyapunov method," in *Proc. Amer. Control Conf.*, Jun. 1995, pp. 3219–3223.

[31] D. Kleinman, "On an iterative technique for Riccati equation computations," *IEEE Trans. Autom. Control*, vol. 13, no. 1, pp. 114–115, Feb. 1968.

[32] A. Moore and C. Atkeson, "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces," *Mach. Learn.*, vol. 21, pp. 1–36, 1995.

[33] N. N. Schraudolph, "Local gain adaptation in stochastic gradient descent ISDIA, Lugano, Switzerland, Tech. Rep. IDSIA-09-99, 1999, p. 8.

[34] M. Lazaro, I. Santamaria, F. Perez-Cruz, and A. Artes-Rodriguez, "Support vector regression for the simultaneous learning of a multivariate function and its derivatives," *Neurocomput.*, vol. 69, pp. 42–61, 2005.

[35] B. A. Pearlmutter, "Fast exact multiplication by the Hessian," *Neural Comput.*, vol. 6, no. 1, pp. 147–160, 1994.

Yuval Tassa received the B.Sc. degree in physics and mathematics from the Hebrew University of Jerusalem, Jerusalem, Israel, in 2003, where currently, he is working towards the Ph.D. degree in neural computation at the Interdisciplinary Center for Neural Computation.

Tom Erez received the B.Sc. degree in mathematics from the Hebrew University of Jerusalem, Jerusalem, Israel, in 2004. Currently, he is working towards the Ph.D. degree in computer science at the Media and Machines lab, Computer Science Department, Washington University in St. Louis, St. Louis, MO.

In 2004–2006, he worked as a Researcher in the multiagent systems division of the ISI Institute, Turin, Italy, as part of the Lagrange and GIACS projects.