Deep Neural Network-Based Approximate Optimal Tracking for Unknown Nonlinear Systems

Max L. Greene, *Member, IEEE*, Zachary I. Bell, *Member, IEEE*, Scott Nivison, *Member, IEEE*, and Warren E. Dixon, *Fellow, IEEE*

***Abstract*—The infinite horizon optimal tracking problem is solved for a deterministic, control-affine, unknown nonlinear dynamical system. A deep neural network (DNN) is updated in real time to approximate the unknown nonlinear system dynamics. The developed framework uses a multitimescale concurrent learning-based weight update policy, with which the output layer DNN weights are updated in real time, but the internal DNN features are updated discretely and at a slower timescale (i.e., with batch-like updates). The design of the output layer weight update policy is motivated by a Lyapunov-based analysis, and the inner features are updated according to existing DNN optimization algorithms. Simulation results demonstrate the efficacy of the developed technique and compare its performance to existing techniques.**

***Index Terms*—Adaptive control, neural networks, nonlinear control, reinforcement learning.**

I. Introduction

Reinforcement learning (RL) is a technique that facilitates adaptation in many computational problems, such as robotics, video game playing, supply chain management, and automatic control. Generally, RL-based agents interact with an environment, sense the state of the system, and perform an action that seeks to minimize or maximize a cost function [1]. The cost depends on the environment, state, and previous action(s) of the system. RL, unlike supervised learning, can evaluate the performance of a particular action without a teacher. This makes RL well-posed to determine policies in which examples, or models, of desired behavior do not exist. These qualities have motivated the use of RL to obtain online approximate solutions to optimal control problems for systems with finite state-spaces as shown in [2].

The solution to the Hamilton–Jacobi–Bellman (HJB) equation is the optimal value function, which is used to determine the optimal control policy [3]. However, the HJB equation is a nonlinear partial differential equation that generally does not have an analytical solution. RL can be used to approximate the solution to the HJB using an approximate dynamic programming (ADP)-based approach in [2], [4], [5], [6], [7], and [8]. If the value function is successfully approximated, then a stabilizing optimal control policy can be determined.

An indirect measure of a given policy's (sub)optimality, called the Bellman error (BE), is derived from the HJB equation. In conventional ADP approaches, the BE is evaluated at on-trajectory points. In the model-based ADP method developed in [9] and used this work, the BE is calculated at on- and off-trajectory points. This process is called BE extrapolation. BE extrapolation can provide simulation of experience; however, methods, such as simulation of experience and BE extrapolation require a model of the system's dynamics. Results, such as [10], use a model-free approach to solve the Hamilton–Jacobi–Isaacs equation. However, Modares et al. [10] rely on an initially stabilizing control policy and a sufficiently large set of data pairs, which are collected online, to successfully approximate the optimal control policy.

Using a model for methods, such as simulation of experience or BE extrapolation enables faster learning in comparison to model-free methods. However, the need for a model can limit robustness and applicability. Motivated by this issue, the model-based ADP methods in [9] and [11] use a data-driven concurrent learning (CL)-based system identifier (see [12] and [13]) to simultaneously approximate the drift dynamics and, subsequently, the optimal control policy. Using a CL-based adaptation law provides guarantees on system parameter convergence, which are not obtained via traditional gradient or least-squares-based update laws. The result in [11] uses a CL-based policy to update the weights of a single hidden-layer NN in real time. However, recent evidence indicates that deep neural networks (DNNs) utilize a more complex structure to potentially improve the function approximation performance [14].

The results in [15] leverage a multitimescale DNN-based model reference adaptive controller. Similarly, the method in [16] uses a multitimescale DNN to approximate the unknown system dynamics, which, with a robust sliding-mode controller, facilitates a trajectory tracking objective. In [16], a gradient-based adaptation policy is used to update the output layer weights of the DNN in real time. Simultaneous to real-time execution, input–output data is stored and used to update the inner layer features using traditional offline DNN function approximation training methods. The inner layer features are updated iteratively (i.e., not in real time); specifically, the inner layer features are instantaneously implemented when the inner layer DNN update policies complete retraining based on user-defined criteria. Iteratively updating the inner layer features introduces discontinuities into the adaptation algorithm; these discontinuities propagate into the closed-loop error system. Hence, the Lyapunov-based stability result from [11] cannot be easily extended. A more rigorous Lyapunov-like analysis that considers piecewise-in-time discontinuities in the dynamics is required. Furthermore, the adaptive update policy in [16] cannot be easily extended to

Manuscript received 30 November 2022; accepted 3 February 2023. Date of publication 20 February 2023; date of current version 26 April 2023. This work was supported in part by AFOSR under Grant FA9550-19-1-0169, in part by AFRL under Grant FA8651-21-F1027, and in part by ONR Grant N00014-21-1-2481. Recommended by Senior Editor Tetsuya Iwasaki and Guest Editors George J. Pappas, Anuradha M. Annaswamy, Manfred Morari, Claire J. Tomlin, Rene Vidal, and Melanie N. Zeilinger. *(Corresponding author: Max L. Greene.)*

Max L. Greene is with the Aurora Flight Sciences, Cambridge, MA 02142 USA (e-mail: greene.max@aurora.aero).

Zachary I. Bell is with the Munitions Directorate, Air Force Research Laboratory, Eglin AFB, Navarre, FL 32566 USA (e-mail: zachary.bell.10@us.af.mil).

Scott Nivison is with the Johns Hopkins University Applied Physics Laboratory, Fort Walton Beach, FL 32578 USA (e-mail: scott.nivison@jhuapl.edu).

Warren E. Dixon is with the Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: wdixon@ufl.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TAC.2023.3246761.

Digital Object Identifier 10.1109/TAC.2023.3246761

facilitate system identification within the ADP framework. The result in [16] does not guarantee convergence of the output layer weights to their respective ideal values. To prove stability of the overall system with an ADP-based controller, the adaptive update policy of the output layer DNN weights must include a CL modification from [13], which further complicates the stability analysis (cf., model-based ADP analyzes in [9] and [11]).

A novel aspect of this article is the development of a multitimescale DNN system identifier to approximate the solution to the optimal trajectory tracking problem using an online model-based RL framework. Unlike the output layer DNN weight update policy in [16], the developed output layer DNN weight update policy is augmented with a CL-based term to facilitate parameter convergence, which is necessary to prove that the trajectories of the system are uniformly ultimately bounded (UUB). Furthermore, iteratively updating the inner layer features of the DNN introduces piecewise-in-time discontinuities into the dynamics, which must be considered in the stability analysis. A Lyapunov-based stability analysis proves that, while performing continuous-time updates to the output layer weights, along with iterative updates to the inner layer features of the DNN, the applied control policy converges to within a neighborhood of the optimal control policy, and the state trajectory converges to within a neighborhood of the desired state trajectory. The simulation results show that one iteration of DNN retraining improves tracking performance by 10.7%.

II. Problem Formulation

Consider a class of nonlinear control-affine systems[1]

$$\dot{x} = f(x) + g(x)u \tag{1}$$

where $x \in \mathbb{R}^n$ denotes the system state, $u \in \mathbb{R}^m$ denotes the control input, $f : \mathbb{R}^n \to \mathbb{R}^n$ denotes the drift dynamics, and $g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ denotes the control effectiveness matrix with $n \geq m$ and the pseudoinverse of $g(x)$ exists. Let $x_d \in \mathbb{R}^n$ denote a time-varying continuously differentiable desired state trajectory, and $e \triangleq x - x_d$ quantifies the error between the actual and desired state. The following assumptions facilitate the formulation of the approximate optimal tracking controller [11].

Assumption 1: The function f is a locally Lipschitz function and $f(0) = 0$. Furthermore, $\nabla_x f : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ is continuous.

Assumption 2: The function g is a locally Lipschitz function, has full column rank for all $x \in \mathbb{R}^n$, and is bounded such that $\underline{g} \leq \|g(x)\| \leq \overline{g}$, where $\underline{g} \in \mathbb{R}_{>0}$ is the infimum overall x of the minimum singular values of $g(x)$, and $\overline{g} \in \mathbb{R}_{>0}$ is the supremum overall x of the maximum singular values of $g(x)$.

Assumption 3: The desired trajectory is bounded from above by a positive constant $\overline{x_d} \in \mathbb{R}$ such that $\sup_{t\in\mathbb{R}_{\geq 0}} \|x_d\| \leq \overline{x_d}$.

Assumption 4: There exists a locally Lipschitz function $h_d : \mathbb{R}^n \to \mathbb{R}^n$, such that $h_d(x_d) \triangleq \dot{x}_d$ and $g^+(x_d)g(x_d)(h_d(x_d) - f(x_d)) = h_d(x_d) - f(x_d)$, $\forall t \in \mathbb{R}_{\geq 0}$, where $g^+ : \mathbb{R}^n \to \mathbb{R}^{m\times n}$ is defined as $g^+(x) \triangleq (g^T(x)g(x))^{-1}g^T(x)$. It follows that $\sup_{t\in\mathbb{R}_{\geq 0}} \|g^+(x_d)\| \leq \overline{g_d^+}$.

Remark 1: Assumptions 2–4 are the typical assumptions necessary to facilitate the transformation of this problem from a time-varying tracking problem to an time-invariant optimal control problem, as outlined in [17]. Assumptions 3 and 4 can be satisfied based on the user selection of x_d.

Based on Assumptions 2–4, the control policy $u_d : \mathbb{R}^n \to \mathbb{R}^m$, which tracks the desired trajectory (i.e., trajectory tracking component of the controller), is $u_d(x_d) \triangleq g^+(x_d)(h_d(x_d) - f(x_d))$. However, $u_d(x_d)$ cannot be calculated if the drift dynamics f are unknown. Hence, an implementable approximation of the trajectory tracking controller component $\hat{u}_d$ is subsequently defined in Section III. Motivated by the desire to transform the time-varying tracking problem into an infinite horizon regulation problem, we follow the development in [17] to rewrite (1) as

$$\dot{\zeta} = F(\zeta) + G(\zeta)\mu \tag{2}$$

where $\zeta \in \mathbb{R}^{2n}$ is the concatenated state vector $\zeta \triangleq [e^T, x_d^T]^T$, $\mu \triangleq u - u_d(x_d)$ is the transient component of the controller, $F : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ is defined as

$$F(\zeta) \triangleq \begin{bmatrix} f(e + x_d) - h_d(x_d) + g(e + x_d)u_d(x_d) \\ h_d(x_d) \end{bmatrix} \tag{3}$$

and $G : \mathbb{R}^{2n} \to \mathbb{R}^{2n\times m}$ is defined as

$$G(\zeta) \triangleq \left[g(e + x_d)^T, \mathbf{0}_{m\times n}\right]^T. \tag{4}$$

From Assumption 2, it follows that $0 < \|G(\zeta)\| \leq \overline{G}$, where $\overline{G} \in \mathbb{R}_{>0}$.

A. Control Objective

The control objective is to find a control policy u that minimizes the cost functional

$$J(\zeta, \mu) = \int_0^\infty r(\zeta(\tau), \mu(\tau))\, d\tau \tag{5}$$

subject to (2) while eliminating the tracking error, where $r : \mathbb{R}^{2n} \times \mathbb{R}^m \to \mathbb{R}_{\geq 0}$ is the instantaneous cost, which is defined as $r(\zeta, \mu) \triangleq \overline{Q}(\zeta) + \mu^T R\mu$, where $\overline{Q} : \mathbb{R}^{2n} \to \mathbb{R}_{\geq 0}$ is a positive semidefinite (PSD) cost function, and $R \in \mathbb{R}^{m\times m}$ is a constant user-defined positive definite (PD) symmetric cost matrix. Let $Q(e) = \overline{Q}(\zeta)\, \forall \zeta \in \mathbb{R}^{2n}$, $e \in \mathbb{R}^n$, where $Q : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ is a PD function.[2]

Property 1: The function $\overline{Q}$ satisfies $\underline{q}(\|e\|) \leq \overline{Q}(\zeta) \leq \overline{q}(\|e\|)$ for $\underline{q}, \overline{q} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$.

The scalar infinite-horizon value function for the optimal solution, i.e., the cost-to-go, denoted by $V^* : \mathbb{R}^{2n} \to \mathbb{R}_{\geq 0}$, is given by $V^*(\zeta) = \min_{\mu(\tau)\in U} \int_t^\infty r(\zeta(\tau), \mu(\tau))d\tau$, where $U \subseteq \mathbb{R}^m$ denotes the action space. If the optimal value function is continuously differentiable, then the optimal control policy V^* is a solution to the corresponding HJB equation

$$0 = \nabla_\zeta V^*(\zeta)\left(F(\zeta) + G(\zeta)\mu^*(\zeta)\right) + \overline{Q}(\zeta) + \mu^*(\zeta)^T R\mu^*(\zeta) \tag{6}$$

which has the boundary condition $V^*(0) = 0$, and the optimal policy $\mu^* : \mathbb{R}^{2n} \to \mathbb{R}^m$ is $\mu^*(\zeta) = -\frac{1}{2}R^{-1}G(\zeta)^T(\nabla_\zeta V^*(\zeta))^T$.

B. Value Function Approximation

The optimal control policy can be derived from the HJB equation in (6); however, the optimal control policy requires knowledge of the optimal value function. Parametric methods can be used to approximate the optimal value function over a compact domain $\Omega \subset \mathbb{R}^{2n}$.[3] Since the function V^* is continuous and an approximation is sought on

[1] For notational brevity, the trajectory $x(t)$, where $x : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, is denoted as $x \in \mathbb{R}^n$ and referred to as x instead of $x(t)$. For example, an equation of the form $f + h(y, t) = g(x)$ should be interpreted as $f(t) + h(y(t), t) = g(x(t))\, \forall t \in \mathbb{R}_{\geq 0}$. The gradient $[\frac{\partial f(x,y)}{\partial x_1}^T, \ldots, \frac{\partial f(x,y)}{\partial x_n}^T]^T$ is denoted by $\nabla_x f(x, y)$. $\|\cdot\|$ denotes both the Euclidean norm for vectors and Frobenius norm for matrices. $\mathbf{1}_{n\times m}$ and $\mathbf{0}_{n\times m}$ denote matrices of ones and zeros with n rows and m columns, respectively. $I_{n\times n}$ denotes an $n \times n$ identity matrix. $\mathrm{vec}(\cdot)$ denotes the vectorization operator.

[2] $\overline{Q}$ is PSD and Q is PD so that the desired trajectory x_d is not penalized and the error e is penalized, e.g., let $\overline{Q}(\zeta) = e^T Qe + x_d^T \mathbf{0}_{n\times n} x_d$.

[3] The subsequent stability analysis in Theorem 1 proves that if ζ is initialized within an appropriately-sized subset of Ω, then it will remain in Ω.

the compact set Ω, the Stone–Weierstrass Theorem is used to express the optimal value function in (2) in Ω as

$$V^*(\zeta) = W^T\sigma(\zeta) + \epsilon(\zeta) \quad \forall \zeta \in \Omega \tag{7}$$

where $W \in \mathbb{R}^L$ is an unknown constant weight vector, $\sigma : \mathbb{R}^{2n} \to \mathbb{R}^L$ is a user-defined vector of activation functions, and $\epsilon : \mathbb{R}^{2n} \to \mathbb{R}$ is the bounded function approximation error.

Assumption 5: There exist constants $\overline{W}, \overline{\sigma}, \overline{\nabla_\zeta \sigma}, \overline{\epsilon}, \overline{\nabla_\zeta \epsilon} \in \mathbb{R}_{>0}$ such that the unknown weights W, user-defined activation function $\sigma(\cdot)$, and function approximation error ε, can be bounded such that $\|W\| \le \overline{W}$, $\sup_{\zeta\in\Omega}\|\sigma(\zeta)\| \le \overline{\sigma}$, $\sup_{\zeta\in\Omega}\|\nabla_\zeta\sigma(\zeta)\| \le \overline{\nabla_\zeta\sigma}$, $\sup_{\zeta\in\Omega}\|\epsilon(\zeta)\| \le \overline{\epsilon}$, and $\sup_{\zeta\in\Omega}\|\nabla_\zeta\epsilon(\zeta)\| \le \overline{\nabla_\zeta\epsilon}$ [2, Assumptions 9.1.c-e].[4]

From (6) and (7), the optimal control policy μ^* can be expressed using (7) as

$$\mu^*(\zeta) = -\frac{1}{2}R^{-1}G(\zeta)^T\left(\nabla_\zeta\sigma(\zeta)^T W + \nabla_\zeta\varepsilon(\zeta)^T\right). \tag{8}$$

The ideal weights W in (7) and (8) are unknown; hence, an approximation of W is sought. Using an actor–critic approach (see [2]), the critic weight estimate, $\hat{W}_c \in \mathbb{R}^L$ is used to approximate the optimal value function $\hat{V} : \mathbb{R}^{2n} \times \mathbb{R}^L \to \mathbb{R}$ denoted as

$$\hat{V}\left(\zeta, \hat{W}_c\right) = \hat{W}_c^T\sigma(\zeta). \tag{9}$$

Similarly, an actor weight estimate, $\hat{W}_a \in \mathbb{R}^L$ is used to approximate the optimal control policy $\hat{\mu} : \mathbb{R}^{2n} \times \mathbb{R}^L \to \mathbb{R}$ defined as

$$\hat{\mu}\left(\zeta, \hat{W}_a\right) \triangleq -\frac{1}{2}R^{-1}G(\zeta)^T\left(\nabla_\zeta\sigma(\zeta)^T\hat{W}_a\right). \tag{10}$$

III. DNN System Identification

The HJB equation in (6) is equal to zero under optimal conditions; however, substituting (9) and (10) into $\nabla_\zeta V^*(\zeta)$ and $\mu^*(\zeta)$ results in a residual term called the BE, which is defined in the subsequent section. To compute this residual term, $F(\zeta)$ and $G(\zeta)$, and therefore, the system model (i.e., $f(x)$ and $g(x)$) must be known. If the system model is not known, then online system identification can be used to estimate the model in real time. The ADP result in [11] approximates $f(x)$ with a single hidden-layer NN online and $g(x)$ is known. Recent works indicate that DNNs may potentially improve function approximation performance [14]. The result in [16] develops a multitimescale DNN-based control method to approximate $f(x)$ online, which may improve the approximation of $f(x)$ [14]. The output layer weights of the DNN are adjusted in real time using adaptive update laws motivated by a Lyapunov-based stability analysis. Concurrent to real-time execution, data are collected and DNN training algorithms (e.g., stochastic gradient descent [19, Ch. 8]), iteratively update the inner layer DNN features. Since DNN learning algorithms are performed iteratively, the inner layer weights are not updated in real time; the weights are discretely updated intermittently during task-execution once training is complete. Motivated to apply the aforementioned technique to ADP, this section outlines the necessary steps required to apply multitimescale DNN system identification to ADP.

DNN architectures can approximate continuous functions on a compact set; the ability to do so is based on universal approximation theorems that can be invoked case-by-case for specific DNN architectures [20]. The drift dynamics f can be approximated on a compact set $\mathcal{C} \subset \mathbb{R}^n$ as[5]

$$f(x) = \theta^T\phi(\Phi^*(x)) + \epsilon_\theta^*(x) \quad \forall x \in \mathcal{C} \tag{11}$$

where $\theta \in \mathbb{R}^{h\times n}$ is an unknown bounded ideal output layer weight matrix, $\phi : \mathbb{R}^p \to \mathbb{R}^h$ is an vector of activation functions, $\Phi^* : \mathbb{R}^n \to \mathbb{R}^p$ represents the unknown inner layer features of the DNN, and $\epsilon_\theta^* : \mathbb{R}^n \to \mathbb{R}^n$ is a bounded function approximation error. For example, the unknown inner layer DNN features Φ^* can be expressed as $\Phi^*(x) = V_k\varrho_k(V_{k-1}, \varrho_{k-1}(V_{k-2}, \varrho_{k-2}(\ldots x)))$, where $k \in \mathbb{N}$ denotes the number of inner layers of the DNN, V_k and $\varrho_k(\cdot)$ denote the corresponding inner layer weights and activation functions of the DNN, respectively.

Based on the DNN representation in (11), the ith DNN-based estimate of the drift dynamics $\hat{f}_i : \mathbb{R}^n \times \mathbb{R}^{h\times n}, \to \mathbb{R}^n$ is defined as

$$\hat{f}_i\left(x, \hat{\theta}\right) = \hat{\theta}^T\phi\left(\hat{\Phi}_i(x)\right) \tag{12}$$

where $\hat{\theta} \in \mathbb{R}^{h\times n}$ is the estimate of the ideal output layer weight matrix θ, and $\hat{\Phi}_i : \mathbb{R}^p \to \mathbb{R}^n$ is the ith iteration selection of the inner features with user-selected activation functions and estimated internal-layer weights. To facilitate the convergence of the DNN-based online system identifier, (12) can be used to develop an estimator

$$\dot{\hat{x}} = \hat{f}_i\left(x, \hat{\theta}\right) + g(x)u + k_o\tilde{x} \tag{13}$$

where $\tilde{x} \triangleq x - \hat{x}$, and $k_o \in \mathbb{R}_{>0}$ is a user-selected estimator learning gain.

Assumption 6: Similar to Assumption 5 there exist constant weights θ and positive constants $\overline{\theta}$, $\overline{\phi}$, $\overline{\epsilon_\theta^*}$, and $\overline{\nabla_x\epsilon_\theta^*} \in \mathbb{R}_{\ge 0}$, such that $\|\theta\| \le \overline{\theta}$, $\sup_{x\in\mathcal{C}}\|\phi(x)\| \le \overline{\phi}$, $\sup_{x\in\mathcal{C}}\|\nabla_x\phi(x)\| \le \overline{\nabla_x\phi}$, $\sup_{x\in\mathcal{C}}\|\epsilon_\theta^*(x)\| \le \overline{\epsilon_\theta^*}$, and $\sup_{x\in\mathcal{C}}\|\nabla_x\epsilon_\theta^*(x)\| \le \overline{\nabla_x\epsilon_\theta^*}$ [21, Ch. 4].

Assumption 7: The ith user-selected inner layer features of the Φ^* and $\hat{\Phi}_i$ are selected such that $\Phi^*(x) - \hat{\Phi}_i(x) \le \tilde{\Phi}_i(x)$, where $\tilde{\Phi}_i : \mathbb{R}^n \to \mathbb{R}^p$ is the inner layer DNN function reconstruction error of the ith iteration, and $\sup_{x\in\mathcal{C},\, i\in\mathbb{N}}\|\tilde{\Phi}_i(x)\| \le \overline{\tilde{\Phi}}$, where $\overline{\tilde{\Phi}} \in \mathbb{R}_{\ge 0}$ is a bounded constant for all i. Using the mean value theorem, $\|\phi(\Phi^*(x)) - \phi(\hat{\Phi}_i(x))\| \le \overline{\nabla_x\phi}\,\overline{\tilde{\Phi}}$.[6]

In the developed method, a DNN with uncertain output layer parameters $\hat{\theta}$ is used to facilitate system identification in the sense that $\hat{F}$ approximates F. To enable convergence of $\hat{F}$ to F, CL-based parameter update laws are developed that use recorded data for learning. This CL strategy is leveraged to modify the output layer weight update law in [16]. As shown in the subsequent stability analysis, this modification enables $\hat{\theta}$ to converge to a region containing θ. Specifically, the output layer DNN weight estimates are updated using the CL-based update law

$$\dot{\hat{\theta}} = \Gamma_\theta\phi\left(\widehat{\Phi}_i(x)\right)\tilde{x}^T + k_\theta\Gamma_\theta\sum_{j=1}^{M}\phi\left(\widehat{\Phi}_i(x_j)\right) \cdot\left(\dot{\bar{x}}_j - g_j(x_j)u_j - \hat{\theta}^T\phi\left(\widehat{\Phi}_i(x_j)\right)\right)^T \tag{14}$$

where $\Gamma_\theta \in \mathbb{R}^{h\times h}$ and $k_\theta \in \mathbb{R}_{>0}$ are constant user-selected adaptation gains. Assumption 8 is required to achieve the aforementioned parameter convergence objective. Specifically, Assumption 8 specifies the quality of exploration data that is required by the history stacks in the second term of (14).

Assumption 8: A history stack of input–output data pairs $\{x_j, u_j\}_{j=1}^M$ and history stack of numerically computed state derivatives $\{\dot{\bar{x}}_j\}_{j=1}^M$, which satisfies $\lambda_{\min}(\sum_{j=1}^M\phi(\hat{\Phi}_i(x_j))\phi(\hat{\Phi}_i(x_j))^T) >$

[4]Assumption 5 can be satisfied by selecting polynomials as basis functions [18, Th. 1.5].

[5]The subsequent stability analysis in Theorem 1 proves that if x is initialized within an appropriately-sized subset of $\mathcal{C}$, then it will remain in $\mathcal{C}$.

[6]Assumption 7 is a mild assumption that is required because an inner layer features of the DNN parameterization in (11) are not assumed to be known; this assumption is required to introduce a $\tilde{\theta}$ term in the subsequent stability analysis, where $\tilde{\theta} \triangleq \theta - \hat{\theta}$. For typical activation functions (e.g., radial basis functions, sigmoids), Assumption 7 can be easily satisfied with $\overline{\tilde{\Phi}} = 2p$.

0 and $\|\dot{\overline{x}}_j - \dot{x}_j\| < \overline{d}\ \forall j$, are available *a priori* for each index j of x_j, where $\overline{d} \in \mathbb{R}_{>0}$ is a known constant, $\dot{x}_j \triangleq f(x_j) + g(x_j)u_j$, and the operator $\lambda_{\min}(\cdot)$ represents the minimum eigenvalue of the argument [13].[7]

Since the dynamics are unknown, similarly, the trajectory tracking component of the controller $u_d(x_d)$ is not known. Hence, an approximation of the trajectory tracking component of the controller $\hat{u}_d : \mathbb{R}^n \times \mathbb{R}^{h\times n} \to \mathbb{R}^m$ is defined as $\hat{u}_d(x_d, \hat{\theta}) \triangleq g^+(x_d)(h_d(x_d) - \hat{f}_i(x, \hat{\theta}))$. The control policy applied to the system in (1) is

$$u \triangleq \hat{\mu}\left(\zeta, \hat{W}_a\right) + \hat{u}_d\left(x_d, \hat{\theta}\right). \tag{15}$$

While the contribution of this section focuses on updating the output layer weights in real time, updating the inner layer features of the DNN system identifier can lead to improved function approximation. Data stored in the CL history stack can be collected a priori and/or online and can simultaneously update the output layer weights and inner layer features of the DNN (i.e., update $\hat{\theta}$ in real time and update $\hat{\Phi}_i(x)$ from i to $i+1$) iteratively. Following (14) and using the CL history stack, the target dataset is $\{\dot{\overline{x}}_j - g(x_j)u_j\}_{j=1}^M$, and the respective input dataset is $\{x_j\}_{j=1}^M$.

IV. BE Extrapolation

Recall, the HJB equation in (6) is equal to zero under optimal conditions; hence, substituting (9), (10), and the approximated drift dynamics $\hat{f}_i(x, \hat{\theta})$ into (6) results in a residual term $\hat{\delta} : \mathbb{R}^{2n} \times \mathbb{R}^{h\times n} \times \mathbb{R}^L \times \mathbb{R}^L \to \mathbb{R}$, which is referred to as the BE, defined as

$$\begin{aligned}\hat{\delta}\left(\zeta, \hat{\theta}, \hat{W}_c, \hat{W}_a\right) &\triangleq \hat{\mu}\left(\zeta, \hat{W}_a\right)^T R \hat{\mu}\left(\zeta, \hat{W}_a\right) + \overline{Q}(\zeta) \\ &\quad + \nabla_\zeta \hat{V}\left(\zeta, \hat{W}_c\right)\left(\hat{F}_i\left(\zeta, \hat{\theta}\right) + G(\zeta)\hat{\mu}\left(\zeta, \hat{W}_a\right)\right)\end{aligned} \tag{16}$$

where $\hat{F}_i : \mathbb{R}^{2n} \times \mathbb{R}^{h\times n} \to \mathbb{R}^{2n}$ is defined as

$$\begin{aligned}\hat{F}_i\left(\zeta, \hat{\theta}\right) &\triangleq \left[\hat{f}_i\left(e + x_d, \hat{\theta}\right)^T - h_d(x_d)^T\right. \\ &\quad \left. + u_d(x_d)^T g(e + x_d)^T, h_d(x_d)^T\right]^T.\end{aligned} \tag{17}$$

Remark 2: Performing minimization of the BE in (16) results in the broader problem of solving the HJB equation in (6). For general nonlinear systems, the HJB equation lacks a general solution. Often, numerical methods are used offline to solve the HJB equation. For cases with known dynamics, the offline-obtained solution will result in closed-loop stability. However, there are cases, such as the one considered in this article, in which the model is unknown. Because of this, the multitimescale DNN identifier is used to approximate the system dynamics in (1) online and, simultaneously, use this approximation of the model in a model-based RL framework to approximate the solution to the HJB equation in real time. The subsequently defined critic weight update policy in (19) is designed to minimize the BE online.

The BE in (16) indicates how close the actor and critic weight estimates are to their respective ideal weights. The mismatch between the estimates and their ideal values are defined as $\tilde{W}_c \triangleq W - \hat{W}_c$ and $\tilde{W}_a \triangleq W - \hat{W}_a$. Substituting (7) and (8) into (6) and subtracting from (16) yields the analytical form of the BE given by

$$\begin{aligned}\hat{\delta}\left(\zeta, \hat{\theta}, \hat{W}_c, \hat{W}_a\right) =& -\omega^T \tilde{W}_c - W^T \nabla_\zeta \sigma\left(F(\zeta) - \hat{F}_i\left(\zeta, \hat{\theta}\right)\right) \\ &+ \frac{1}{4}\tilde{W}_a^T G_\sigma \tilde{W}_a + O(\zeta)\end{aligned} \tag{18}$$

where $\omega : \mathbb{R}^{2n} \times \mathbb{R}^L \times \mathbb{R}^{h\times n} \to \mathbb{R}^h$ is defined as $\omega(\zeta, \hat{W}_a, \hat{\theta}) \triangleq \nabla_\zeta \sigma(\zeta)(\hat{F}_i(\zeta, \hat{\theta}) + G(\zeta)\hat{\mu}(\zeta, \hat{W}_a))$, $O(\zeta) \triangleq \frac{1}{2}\nabla_\zeta \epsilon(\zeta) G_R \nabla_\zeta \sigma(\zeta)^T W + \frac{1}{4}G_\varepsilon - W^T \nabla_\zeta \sigma(\zeta)\epsilon_\theta^*(e + x_d) - \nabla_\zeta \varepsilon(\zeta) F(\zeta)$, $G_R = G_R(\zeta) \triangleq G(\zeta)R^{-1}G(\zeta)^T$, $G_\sigma = G_\sigma(\zeta) \triangleq \nabla_\zeta \sigma(\zeta) G_R(\zeta) \nabla_\zeta \sigma(\zeta)^T$, and $G_\varepsilon = G_\varepsilon(\zeta) \triangleq \nabla_\zeta \epsilon(\zeta) G(\zeta) \nabla_\zeta \epsilon(\zeta)^T$.

At each time instant $t \in \mathbb{R}_{\geq 0}$, the estimated BE in (16) and policy in (10) are evaluated using the current system state, critic estimate, actor estimate, and output layer weight estimate matrix to get the instantaneous BE and control policy, which are denoted by $\hat{\delta} \triangleq \hat{\delta}(\zeta, \hat{\theta}, \hat{W}_c, \hat{W}_a)$ and $\hat{\mu} \triangleq \hat{\mu}(\zeta, \hat{W}_a)$, respectively. The system model, which is approximated using the aforementioned DNN-based identifier, can be used to evaluate the BE at off-trajectory states in Ω. The process of evaluating the BE at off-trajectory states is called BE extrapolation. BE extrapolation yields simultaneous exploitation and exploration, which results in faster policy learning over a domain.

To facilitate BE extrapolation, the off-policy trajectories $\{\zeta_e : \zeta_e \in \Omega\}_{e=1}^N$ are selected, where $N \in \mathbb{N}$ denotes the number of extrapolated trajectories in Ω. The state-dependent extrapolation points can be selected a priori by the user or by using an online strategy.[8] Using the extrapolated trajectories $\zeta_e \in \Omega$, the BE in (16) is evaluated such that $\hat{\delta}_e \triangleq \hat{\delta}(\zeta_e, \hat{\theta}, \hat{W}_c, \hat{W}_a)$. Let the tuple $(\Sigma_c, \Sigma_a, \Sigma_\Gamma)$ define the extrapolation data corresponding to Ω such that $\Sigma_c \triangleq \frac{1}{N}\sum_{e=1}^N \frac{\omega_e}{\rho_e}\hat{\delta}_e$, $\Sigma_a \triangleq \frac{1}{N}\sum_{e=1}^N \frac{G_{\sigma e}^T \hat{W}_a \omega_e^T}{4\rho_e}$, and $\Sigma_\Gamma \triangleq \frac{1}{N}\sum_{e=1}^N \frac{\omega_e \omega_e^T}{\rho_e}$, where $\omega_e \triangleq \omega(\zeta_e, \hat{\theta}, \hat{W}_a)$, $\rho_e \triangleq \rho(\zeta_e, \hat{\theta}, \hat{W}_a) = 1 + \nu \omega_e^T \Gamma \omega_e$, and $\Gamma \in \mathbb{R}^{L\times L}$ is a user-initialized learning gain.

To ensure that enough off-trajectory BE extrapolation data is selected to achieve sufficient exploration, Assumption 9 is provided, which facilitates the subsequent stability analysis.

Assumption 9: There exist a finite set of trajectories $\{\zeta_e : \zeta_e \in \Omega\}_{e=1}^N$ such that $0 < \underline{c} \triangleq \inf_{t\in\mathbb{R}_{\geq 0}} \lambda_{\min}\{\Sigma_\Gamma\}$ for all $t \in \mathbb{R}_{\geq 0}$, where $\lambda_{\min}\{\cdot\}$ is the minimum eigenvalue, and the constant $\underline{c}$ is the lower bound of the value of each input–output data pairs' minimum eigenvalues.[9]

V. Actor and Critic Weight Update Laws

Using the instantaneous BE $\hat{\delta}$ and extrapolated BEs $\hat{\delta}_e$, the continuous-time least-squares-based update policies for the critic and actor weights, which are designed based on the subsequent stability analysis, are

$$\dot{\hat{W}}_c = -\eta_{c1}\Gamma \frac{\omega}{\rho}\hat{\delta} - \eta_{c2}\Gamma \Sigma_c \tag{19}$$

$$\dot{\Gamma} = \left(\lambda \Gamma - \eta_{c1}\frac{\Gamma \omega \omega^T \Gamma}{\rho^2} - \eta_{c2}\Gamma \Sigma_\Gamma \Gamma\right)\mathbf{1}_{\{\underline{\Gamma} \leq \|\Gamma\| \leq \overline{\Gamma}\}} \tag{20}$$

$$\begin{aligned}\dot{\hat{W}}_a =& -\eta_{a1}\left(\hat{W}_a - \hat{W}_c\right) - \eta_{a2}\hat{W}_a \\ &+ \eta_{c1}\frac{G_\sigma^T \hat{W}_a \omega^T}{4\rho}\hat{W}_c + \eta_{c2}\Sigma_a \hat{W}_c\end{aligned} \tag{21}$$

[7]Availability of the system identification history stack (i.e., the tuple $\{x_j, u_j, \dot{\overline{x}}_j\}_{j=1}^M$) a priori is not necessary [11]. If the system is sufficiently excited and the history stack is recorded within a finite time, then the developed controller can be used. Switching between a PE-based controller and the developed controller results in a switched subsystem with one switching event. The stability of the overall system is determined from the stability of the individual subsystems.

[8]The design of online strategies to determine extrapolation points could potentially improve learning and is a topic for future research.

[9]Assumption 9 can be verified online. Furthermore, Assumption 9 can be heuristically satisfied by selecting more BE extrapolation points than number of neurons in σ such that $N \gg L$ [9].

where $\eta_{c1}, \eta_{c2}, \eta_{a1}, \eta_{a2}, \lambda \in \mathbb{R}_{>0}$ are constant learning gains, $\overline{\Gamma}$ and $\underline{\Gamma} \in \mathbb{R}_{>0}$ are upper and lower bound saturation constants, and $\mathbf{1}_{\{\cdot\}}$ denotes the indicator function. The indicator function in (20) ensures that $\|\Gamma(t)\|$ is upper and lower bounded by two user-defined saturation gains, $\overline{\Gamma}$ and $\underline{\Gamma} \in \mathbb{R}_{>0}$, to ensure that $\underline{\Gamma} \leq \|\Gamma(t)\| \leq \overline{\Gamma}$ for all $t \in \mathbb{R}_{\geq 0}$. The indicator function in (20) can be removed provided ρ and ρ_i are changed to $\rho = 1 + \nu\omega^T\omega$ and $\rho_i = 1 + \nu\omega_i^T\omega_i$, and additional assumptions are included for the regressors $\frac{\omega}{\rho}$ and Σ_Γ to ensure that Γ is bounded [22].

Remark 3: Under Assumptions 1–4, the optimal value function can be shown to be the unique PD solution of the HJB equation. Approximation of the PD solution to the HJB equation is guaranteed by appropriately selecting initial weight estimates and the Lyapunov-based update laws in (19)–(21) [23].

VI. Stability Analysis

Recall from Property 1 that the function $\overline{Q}$ and, therefore, the optimal value function V^* in (7) is PSD. Hence, V^* is not a valid Lyapunov function. The result in [17] can be used to show that a nonautonomous form of V^*, denoted as $V^*_{na} : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ and defined as $V^*_{na}(e, t) \triangleq V^*(\zeta)$, is PD and decrescent. Furthermore, $V^*_{na}(0, t) = 0$ and there exist class $\mathcal{K}_\infty$ functions $\underline{v}, \overline{v} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ that bound $\underline{v}(\|e\|) \leq V^*_{na}(e, t) \leq \overline{v}(\|e\|) \, \forall e \in \mathbb{R}^n, t \in \mathbb{R}_{\geq 0}$. Hence, $V^*_{na}(e, t)$ is a valid Lyapunov function. Let $Z \in \mathbb{R}^{2n+2L+hn}$ denote a concatenated state defined as $Z \triangleq [e^T, \tilde{W}_c^T, \tilde{W}_a^T, \tilde{x}^T, \mathrm{vec}(\tilde{\theta})^T]^T$. Let $V_L : \mathbb{R}^{2n+2L+hn} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ be a candidate Lyapunov function defined as

$$V_L(Z, t) \triangleq V^*_{na}(e, t) + \frac{1}{2}\tilde{W}_c^T \Gamma(t)^{-1} \tilde{W}_c + \frac{1}{2}\tilde{W}_a^T \tilde{W}_a + \frac{1}{2}\tilde{x}^T\tilde{x} + \frac{1}{2}\mathrm{tr}\left(\tilde{\theta}^T \Gamma_\theta^{-1} \tilde{\theta}\right). \tag{22}$$

Using the properties of $V^*_{na}(e, t)$ and [24, Lemma 4.3], then (22) be bounded as $\alpha_1(\|Z\|) \leq V_L(Z, t) \leq \alpha_2(\|Z\|)$ for class $\mathcal{K}_\infty$ functions $\alpha_1, \alpha_2 : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. Using (20), the normalized regressors $\frac{\omega}{\rho}$ and $\frac{\omega_e}{\rho_e}$ can be bounded as $\sup_{t \in \mathbb{R}_{\geq 0}} \|\frac{\omega}{\rho}\| \leq \frac{1}{2\sqrt{\nu\underline{\Gamma}}}$ for all $\zeta \in \Omega$ and $\sup_{t \in \mathbb{R}_{\geq 0}} \|\frac{\omega_e}{\rho_e}\| \leq \frac{1}{2\sqrt{\nu\underline{\Gamma}}}$ for all $\zeta_e \in \Omega$. The matrices G_R and G_σ can be bounded as $\sup_{\zeta \in \Omega} \|G_R\| \leq \lambda_{\max}\{R^{-1}\}\overline{G}^2 \triangleq \overline{G_R}$ and $\sup_{\zeta \in \Omega} \|G_\sigma\| \leq (\overline{\nabla_\zeta \sigma G})^2 \lambda_{\max}\{R^{-1}\} \triangleq \overline{G_\sigma}$, respectively, where $\lambda_{\max}\{\cdot\}$ denotes the maximum eigenvalue.

To facilitate the subsequent stability analysis, let $r \in \mathbb{R}_{>0}$ be the radius of a compact ball centered at the origin $\chi \subset \mathbb{R}^{2n+2L+hn}$ centered at the origin, and $l \in \mathbb{R}_{>0}$ is a positive constant that depends on the bounded NN constants in Assumptions 5–7. The sufficient conditions for ultimate boundedness of Z are derived based on the subsequent stability analysis as

$$\eta_{a1} + \eta_{a2} \geq (\eta_{c1} + \eta_{c2}) \frac{\overline{W}\,\overline{G_\sigma}}{\sqrt{\nu\underline{\Gamma}}} \tag{23}$$

$$\underline{c} \geq 4\frac{\eta_{a1}}{\eta_{c2}} + \frac{(\eta_{c1} + \eta_{c2})^2 \overline{W}^2 \overline{G_\sigma}^2}{4\eta_{c2}\nu\underline{\Gamma}(\eta_{a1} + \eta_{a2})} + \frac{3(\eta_{c1} + \eta_{c2})^2 \overline{W}^2 \overline{\nabla_\zeta \sigma}^2 \left(\overline{\phi} + \overline{g_d^+}\,\overline{\phi}\overline{g}\right)^2}{2\eta_{c2} k_\theta \nu\underline{\Gamma} \lambda_{\min}\left\{\sum_{j=1}^M \phi\left(\widehat{\Phi}_i(x_j)\right) \phi\left(\widehat{\Phi}_i(x_j)\right)^T\right\}} \tag{24}$$

$$\nu_l^{-1}(l) < \alpha_2^{-1}(\alpha_1(r)) \tag{25}$$

where ν_l is a subsequently defined PD function.[10]

[10]See [9] for insight into satisfying the conditions in (23)–(25).

The optimal value function is parameterized with a linear combination of weights and basis functions; this has been done in results such as [2]. However, the multitimescale DNN identifier introduces new terms and piecewise-in-time discontinuities into the dynamics. Hence, existing actor–critic approaches cannot be applied to show stability of the closed-loop system. The subsequent Lyapunov-based stability analysis is performed to analyze the convergence and stability properties of the online implementation of (13), (14), and (19)–(21).

Theorem 1: Given the dynamics in (1), that Assumptions 1–9 are satisfied, and that the conditions in (23)–(25) are satisfied, then the tracking error e, weight estimation errors $\tilde{W}_c$ and $\tilde{W}_a$, state estimation error $\tilde{x}$, and output layer weight matrix error $\tilde{\theta}$ are UUB. Hence, the applied control policy $\hat{u}$ converges to a neighborhood of the optimal control policy u^*.

Proof: Using (1), the fact that $\dot{V}^*_{na}(e, t) = \dot{V}^*(\zeta)$, $\dot{V}^*(\zeta) = \nabla_\zeta V^*(\zeta)(F(\zeta) + G(\zeta)\mu)$, (13), (14), (19)–(21), Young's Inequality, nonlinear damping, the class of dynamics in (2), Assumptions 8 and 9, and substituting the sufficient conditions in (23) and (24) yields

$$\dot{V}_L \leq -\nu_l(\|Z\|), \forall \nu_l^{-1}(l) \leq \|Z\| \leq \alpha_2^{-1}(\alpha_1(r)) \quad \forall t \in \mathbb{R}_{\geq 0} \tag{26}$$

where $\nu_l(\|Z\|) \triangleq \frac{1}{2}\underline{q}(\|e\|) + \frac{1}{16}\eta_{c2}\underline{c}\|\tilde{W}_c\|^2 + \frac{1}{16}(\eta_{a1} + \eta_{a2})\|\tilde{W}_a\|^2 + \frac{k_o}{4}\|\tilde{x}\|^2 + \frac{k_\theta}{6}\lambda_{\min}\{\sum_{j=1}^M \phi(\widehat{\Phi}_i(x_j))\phi(\widehat{\Phi}_i(x_j))^T\}\|\mathrm{vec}(\tilde{\theta})\|^2$. Since the discontinuities in the update laws in (13), (14), and (19)–(21) are piecewise continuous in time and (22) is a common Lyapunov function across each DNN iteration i, [24, Th. 4.18] can be invoked to conclude that Z is UUB such that $\limsup_{t\to\infty} \|Z(t)\| \leq \alpha_1^{-1}(\alpha_2(\nu_l^{-1}(l)))$ and $\hat{\mu}$ converges to a neighborhood around the optimal policy μ^*. Since $Z \in \mathcal{L}_\infty$, it follows that $e, \tilde{W}_c, \tilde{W}_a, \tilde{x}, \tilde{\theta} \in \mathcal{L}_\infty$; hence, $x, \hat{W}_c, \hat{W}_a, \hat{\theta} \in \mathcal{L}_\infty$ and $\mu \in \mathcal{L}_\infty$.

Using (26), the result in [24, Th. 4.18] can be invoked to show that every trajectory $Z(t)$ that satisfies the initial condition $\|Z(0)\| \leq \alpha_2^{-1}(\alpha_1(r))$ is bounded for all $t \in \mathbb{R}_{\geq 0}$. That is, $Z \in \chi \, \forall t \in \mathbb{R}_{\geq 0}$. Since $Z \in \chi$ it follows that the individual states of Z lie on compact sets.[11] Furthermore, since $x_d \leq d_d$, then $\zeta \in \Omega$ and $x \in \mathcal{C}$, where Ω is the compact set that facilitates value function approximation, and $\mathcal{C}$ is the compact set that facilitates DNN-based system identification.

VII. Simulation Example

The following section applies the developed technique to an optimal tracking problem for an autonomous undersea vehicle (AUV) with the instantaneous cost function $r(\zeta, \mu) = e^TQe + \mu^TR\mu$. The system dynamics for the AUV in this example is from [26]. To focus the scope of this simulation section, it is assumed that the AUV is neutrally buoyant while submerged, the center of gravity is below the center of buoyancy on the *z*-axis, and the vehicle model accounts for small roll and pitch angles. The dynamics for the AUV in an irrotational current can be expressed as

$$\dot{\xi} = f_H(\zeta, \nu_c) + f_0(\zeta, \dot{\nu}_c) + g\tau_b \tag{27}$$

where $\xi \triangleq [\eta_{AUV}^T \nu_{AUV}^T]^T \in \mathbb{R}^6$ is the concatenated state vector, $f_0 : \mathbb{R}^6 \times \mathbb{R}^3 \to \mathbb{R}^6$ is the known rigid body drift dynamics, $f_H : \mathbb{R}^6 \times \mathbb{R}^3 \to \mathbb{R}^6$ is the unknown hydrodynamic parameter effects (see [26] for definitions of the states and dynamics). The rigid body dynamics are assumed to be known because they are measurable a priori, whereas the hydrodynamic parameters are not known. The irrotational current vector for this example is $\nu_c = [-0.1, 0.1, 0]^T$.

The time-varying desired trajectory is $\xi_d(t) = [\cos(\frac{\pi}{20}t), \cos(\frac{\pi}{30}t), 0, -\frac{\pi\sin(\frac{\pi}{20}t)}{20}, -\frac{\pi\sin(\frac{\pi}{30}t)}{30}, 0]^T$, and the control

[11]See [25, Algorithm A.2] for discussion on establishing the size of compact sets χ.

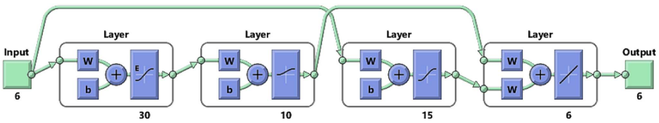

Fig. 1. DNN is composed of four layers, each with 30, 10, 15, and 6 neurons, respectively.

objective is to minimize the infinite horizon cost function in (5). The drift dynamics are unknown and approximated using the developed DNN-based system identification method. The DNN used in this simulation was composed of four layers, each with 30, 10, 15, and 6 neurons, respectively. The DNN architecture is illustrated in Fig. 1. The first, second, and third layers use Elliot symmetric sigmoid, logarithmic sigmoid, and tangent sigmoid activation functions, respectively.[12] The first, second, and third layers include bias terms. The mean squared error was used as the loss function for training. The Levenberg–Marquardt algorithm was used to train the weights of the DNN. For each DNN training iteration, 70% of the data was used for training, 15% was used for validation, and 15% was used for testing.

The controller cost parameters in (5) are $Q = \mathrm{diag}([100,\ 100,\ 200,\ 10,\ 10,\ 50])$ and $R = I_{3\times 3}$. $N = 110592$ BE extrapolation trajectories were selected across the operating domain Ω. The initial conditions used for the simulated system are $\xi(0) = [-1,\ 1.5,\ \frac{3\pi}{4},\ 0,\ 0,\ 0]^T$,$\hat{\xi}(0) = \xi(0)$, and $\Gamma(0) = 5000 \cdot I_{27\times 27}$.[13] Both $\hat{W}_c(0)$ and $\hat{W}_a(0)$ are initialized by solving the algebraic Riccati equation for the linearized rigid body AUV dynamics about the position $\xi = \mathbf{0}_{6\times 1}$. The polynomial basis function σ with 27 elements is used for value function approximation. Each $\hat{\theta}(0) \in \mathbb{R}^{25\times 6}$ is initialized according to its subsequently-defined training method. The gains were selected as $\eta_{c1} = 0$, $\eta_{c2} = 0.5$, $\eta_{a1} = 10$,$\eta_{a2} = 0.1$, $\lambda = 0.025$, $\nu = 0.025$, $\overline{\Gamma} = 5000$,$\underline{\Gamma} = 100$, $k_\theta = 5 \cdot 10^6$, $k_o = 10$, and $\Gamma_\theta = 1$. To facilitate CL, a maximum of 100 state-action pairs are recorded and replaced according to the singular value maximization algorithm defined in [13, Algorithm 1].

This section presents simulation results for exact model knowledge (EMK) ADP, linearly parameterizable (LP) ADP, randomly initialized DNN ADP, transfer learning DNN ADP, and pretrained DNN ADP. All of the ADP methods in this simulation comparison are model-based (i.e., use BE extrapolation). EMK ADP uses EMK of $f(x)$, so the results present the best possible performance for an ADP-based controller for a given set of gains and extrapolation trajectories. LP ADP assumes that $f_0(x)$ is LP (i.e., $f(x) = Y(x)\theta$, where $Y(x)$ exactly parameterizes the dynamics), as typically seen in an adaptive control literature [21, Sec. 3.4.3]. LP requires some, but not EMK, and represents a special case (subset) of the dynamics in (1). For the pretrained DNN ADP method, the DNN is trained a priori using the actual dynamics in (27). The transfer learning DNN ADP method is also based on training the DNN a priori on a system that is similar, but not exactly the same, as the dynamics used during implementation. For the transfer learning case, the current vector $\nu_c = [-0.1, 0.1, 0]^T$ is changed to $\nu_c = [0.1, -0.1, 0]^T$ to represent the uncertainty between the training model and the actual model. The randomly initialized DNN ADP method does not require any prior training, i.e., does not require knowledge of the drift dynamics. The retrained DNN ADP method is initialized as the random DNN ADP method; however, the retrained DNN ADP method updates the inner layer features once online. The retrained DNN ADP method highlights the performance improvements that occur through online iterative adjustment of the inner layer DNN features. For retraining, a history stack of DNN training data is collected for 120 s.[14] After 120 s, the internal DNN weights begin retraining. The DNN is trained for 50 epochs, which takes approximately 12.8 s.

TABLE I
SIMULATION RESULTS AND ADP COMPARISON

Control Type	Total Integral Error	Integral Difference from EMK	Integral Error After Update
EMK ADP	17.89	–	0.00
LP ADP	22.77	5.77	0.01
Pretrained DNN ADP	302.83	287.02	96.81
Transfer Learning DNN ADP	322.15	306.16	125.11
Retrained DNN ADP	352.78	352.03	145.76
Random DNN ADP	370.30	369.55	163.29

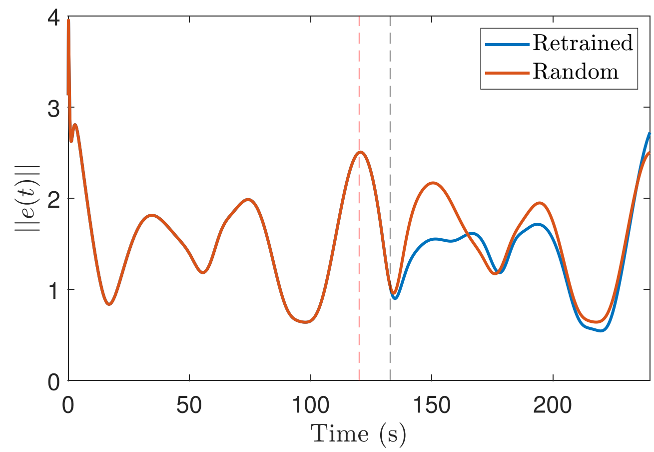

Fig. 2. Error comparisons between the retrained DNN ADP and random DNN ADP methods. The red dashed line at $t = 120$ s represents the beginning of the retraining, and the black dashed line at approximately $t = 12.8$ s represents the end of the retraining and when the new internal DNN weights are implemented. The retrained DNN has improved tracking performance. Since the random DNN and retrained DNN cases are initialized identically, they have identical performance for the first 120 s (i.e., until the inner layer DNN features are updated).

Table I compares the performance of each method, and Fig. 2 compares the randomly DNN ADP and retrained DNN ADP methods. The second column compares the total integral error of each simulation (i.e., $\int_0^{240} e(\tau)\, d\tau$). Recall, the EMK ADP method is expected to have the best performance. LP ADP is the best performing method with uncertainty, followed by transfer learning DNN ADP, pretrained DNN ADP, retrained DNN ADP, and random DNN ADP, respectively. The third column of Table I compares the ADP methods with the integral of the difference between their state trajectory and the EMK ADP state trajectory. Similarly, LP ADP performs the best, followed by pretrained DNN ADP, transfer learning DNN ADP, retrained DNN ADP, and random DNN ADP, respectively. While the transfer learning DNN ADP case has a lower integral of error in the second column, the pretrained DNN ADP case performs closer to the EMK ADP case, as seen in the third column. The fourth column of Table I compares the ADP methods after the retrained DNN ADP has completed retraining. Once

[12]The number of neurons per layer and activation functions were selected empirically.

[13]To reduce the computational complexity of the simulation, the least-squares gain matrix is initialized such that $\Gamma(0) = \overline{\Gamma}$.

[14]The time of 120 s was selected because it is the period of x_d. Collecting more data should result in improved training of the inner layer weights at the expense of additional computation time.

retraining is complete, the new internal weights are implemented. After retraining, the difference between the retrained DNN and random DNN controllers is notable. The improved retrained DNN has significantly better performance compared to the random DNN case, and it is comparable to that of the other ADP methods. The integral of error from the time at which the new inner layer weights are implemented to the end of the trial is used to compare the performance of the two techniques. After retraining, the integral of error for the random case is 163.29. The integral of error for the retrained case is 145.76. Hence, the online retraining method empirically improved error tracking by 10.7%. After retraining, the EMK ADP and LP ADP perform the best, followed by pretrained DNN ADP, transfer learning DNN ADP, retrained DNN ADP, and random DNN ADP, respectively. The unsurprising trend in Table I is that if a system has more model knowledge a priori, then performance improves.

These simulation studies confirm the effectiveness of a DNN-based ADP controller with a real-time output layer weight and iterative inner layer feature updates. The benefit of the developed technique is that a component of the drift dynamics f_0 can be approximated without any model knowledge a priori. This simulation example illustrates the well-understood trend that more model knowledge leads to improved controller performance. Since the developed method can the update the DNN features to better approximate the nonlinear drift dynamics, a DNN-based model of the drift dynamics can be learned without pretraining. Unlike existing single-layer NN-based system identifiers, the additional layers of the DNN facilitate improved function approximation. Combining existing data-based deep learning algorithms with adaptive control policies can decrease model uncertainty, enhance the quality of the value function approximation, and ultimately improve the system performance.

VIII. Conclusion

This article develops a framework for using a DNN-based system identifier within a model-based RL ADP framework to solve the infinite horizon optimal tracking control problem. A CL-based continuous-time update law is used to update the output layer weights of the DNN. A Lyapunov-based analysis is performed to prove UUB identification of the DNN weights, trajectory tracking, and approximation of the applied control policy to a neighborhood of the optimal control policy. Simulation results illustrate the performance of the developed method in comparison to existing methods applied to an AUV. Future work will investigate using a DNN to simultaneously approximate the value function in conjunction with a DNN-based system identifier.

Acknowledgment

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of of Aurora Flight Sciences, the Johns Hopkins Applied Physics Laboratory, or the sponsoring agencies.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[2] D. Vrabie, K. G. Vamvoudakis, and F. L. Lewis, *Optimal Adaptive Control and Differential Games by Reinforcement Learning Principles*. London, U.K.: The Institution of Engineering and Technology, 2013.

[3] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton, NJ, USA: Princeton Univ. Press, 2012.

[4] F. L. Lewis and D. Liu, *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*, vol. 17. Hoboken, NJ, USA: Wiley, 2013.

[5] A. Chakrabarty, D. K. Jha, G. T. Buzzard, Y. Wang, and K. G. Vamvoudakis, "Safe approximate dynamic programming via kernelized Lipschitz estimation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 405–419, Jan. 2021.

[6] B. Pang and Z.-P. Jiang, "Adaptive optimal control of linear periodic systems: An off-policy value iteration approach," *IEEE Trans. Autom. Control*, vol. 66, no. 2, pp. 888–894, Feb. 2021.

[7] W. Gao, M. Mynuddin, D. C. Wunsch, and Z.-P. Jiang, "Reinforcement learning-based cooperative optimal output regulation via distributed adaptive internal model," *IEEE Trans. Neural Netw. Learn. Syst*., vol. 33, no. 10, pp. 5229–5240, Oct. 2022.

[8] B. Pang, T. Bian, and Z.-P. Jiang, "Robust policy iteration for continuous-time linear quadratic regulation," *IEEE Trans. Autom. Control*, vol. 59, no. 11, pp. 3051–3056, Nov. 2024.

[9] R. Kamalapurkar, P. Walters, and W. E. Dixon, "Model-based reinforcement learning for approximate optimal regulation," *Automatica*, vol. 64, pp. 94–104, 2016.

[10] H. Modares, F. L. Lewis, and Z.-P. Jiang, "H∞ tracking control of completely unknown continuous-time systems via off-policy reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst*., vol. 26, no. 10, pp. 2550–2562, Oct. 2015.

[11] R. Kamalapurkar, L. Andrews, P. Walters, and W. E. Dixon, "Model-based reinforcement learning for infinite-horizon approximate optimal tracking," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 753–758, Mar. 2017.

[12] G. V. Chowdhary and E. N. Johnson, "Theory and flight-test validation of a concurrent-learning adaptive controller," *J. Guid. Control Dyn.*, vol. 34, no. 2, pp. 592–607, Mar. 2011.

[13] G. Chowdhary, T. Yucelen, M. Mühlegg, and E. N. Johnson, "Concurrent learning adaptive control of linear systems with exponentially convergent bounds," *Int. J. Adapt. Control Signal Process.*, vol. 27, no. 4, pp. 280–301, 2013.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[15] G. Joshi and G. Chowdhary, "Deep model reference adaptive control," in *Proc. IEEE Conf. Decis. Control*, 2019, pp. 4601–4608.

[16] R. Sun, M. Greene, D. Le, Z. Bell, G. Chowdhary, and W. E. Dixon, "Lyapunov-based real-time and iterative adjustment of deep neural networks," *IEEE Control Syst. Lett.*, vol. 6, pp. 193–198, 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9337905/

[17] R. Kamalapurkar, H. Dinh, S. Bhasin, and W. E. Dixon, "Approximate optimal trajectory tracking for continuous-time nonlinear systems," *Automatica*, vol. 51, pp. 40–48, Jan. 2015.

[18] F. Sauvigny, *Partial Differential Equations 1: Foundations and Integral Representations*. Berlin, Germany: Springer, 2012.

[19] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.

[20] P. Kidger and T. Lyons, "Universal approximation with deep narrow networks," in *Proc. Conf. Learn. Theory*, 2020, pp. 2306–2327.

[21] F. L. Lewis, S. Jagannathan, and A. Yesildirak, *Neural Network Control of Robot Manipulators and Nonlinear Systems*. Philadelphia, PA, USA: CRC Press, 1998.

[22] P. Deptula, J. Rosenfeld, R. Kamalapurkar, and W. E. Dixon, "Approximate dynamic programming: Combining regional and local state following approximations," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2154–2166, Jun. 2018.

[23] P. Deptula, Z. Bell, E. Doucette, W. J. Curtis, and W. E. Dixon, "Data-based reinforcement learning approximate optimal control for an uncertain nonlinear system with control effectiveness faults," *Automatica*, vol. 116, pp. 1–10, Jun. 2020.

[24] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2002.

[25] R. Kamalapurkar, P. S. Walters, J. A. Rosenfeld, and W. E. Dixon, *Reinforcement Learning for Optimal Feedback Control: A Lyapunov-Based Approach*. Berlin, Germany: Springer, 2018.

[26] N. Fischer, D. Hughes, P. Walters, E. Schwartz, and W. E. Dixon, "Nonlinear RISE-based control of an autonomous underwater vehicle," *IEEE Trans. Robot.*, vol. 30, no. 4, pp. 845–852, Aug. 2014.