Lyapunov-Based Real-Time and Iterative Adjustment of Deep Neural Networks

Runhan Sun, Max L. Greene, *Student Member, IEEE*, Duc M. Le, Zachary I. Bell, *Member, IEEE*, Girish Chowdhary, *Senior Member, IEEE*, and Warren E. Dixon, *Fellow, IEEE*

***Abstract*—A real-time Deep Neural Network (DNN) adaptive control architecture is developed for general uncertain nonlinear dynamical systems to track a desired time-varying trajectory. A Lyapunov-based method is leveraged to develop adaptation laws for the output-layer weights of a DNN model in real-time while a data-driven supervised learning algorithm is used to update the inner-layer weights of the DNN. Specifically, the output-layer weights of the DNN are estimated using an unsupervised learning algorithm to provide responsiveness and guaranteed tracking performance with real-time feedback. The inner-layer weights of the DNN are trained with collected data sets to increase performance, and the adaptation laws are updated once a sufficient amount of data is collected. Building on the results in (Joshi and Chowdhary, 2019) and (Joshi et al., 2020), which focus on deep model reference adaptive control for linear systems with known drift dynamics and control effectiveness matrices, this letter considers general control-affine uncertain nonlinear systems. The real-time controller and adaptation laws enable the system to track a desired time-varying trajectory while compensating for the unknown drift dynamics and parameter uncertainties in the control effectiveness. A nonsmooth Lyapunov-based analysis is used to prove semi-global asymptotic tracking of the desired trajectory. Numerical simulation examples are included to validate the results, and the Levenberg-Marquardt algorithm is used to train the weights of the DNN.**

***Index Terms*—Adaptive control, deep neural networks, Lyapunov-based analysis.**

Manuscript received October 23, 2020; revised January 6, 2021; accepted January 21, 2021. Date of publication January 28, 2021; date of current version June 24, 2021. This work was supported in part by NSF under Award 1509516; in part by the Office of Naval Research under Grant N00014-13-1-0151; and in part by Air Force Office of Scientific Research under Award FA9550-19-1-0169. Recommended by Senior Editor C. Seatzu. *(Corresponding author: Runhan Sun.)*

Runhan Sun, Max L. Greene, Duc M. Le, and Warren E. Dixon are with the Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: runhansun@ufl.edu; maxgreene12@ufl.edu; ledan50@ufl.edu; wdixon@ufl.edu).

Zachary I. Bell is with the Munitions Directorate, Air Force Research Laboratory, Eglin AFB, FL 32542 USA (e-mail: zachary.bell.10@us.af.mil).

Girish Chowdhary is with the Coordinated Science Laboratory, University of Illinois, Urbana–Champaign, IL 61801 USA (e-mail: girishc@illinois.edu).

Digital Object Identifier 10.1109/LCSYS.2021.3055454

I. Introduction

NEURAL Networks (NNs) have been used in results such as [1] to approximate the receding-horizon (RH) regulator in RH optimal control. More recently, results in [2] and [3] leverage Deep Neural Networks (DNNs) to approximate the MPC laws. However, the NN approximations in [1]–[4] can only provide statistical guarantees of the approximation error. DNN methods can also be used to capture complex features of the dynamics by using back-propagation algorithms that indicate how to update the inner-layer weights [5]. In results such as [5] and [6], the emergence of DNN models with more complex structures improve function approximation performance. Although DNN function approximation methods show improved performance empirically, these methods typically lack performance guarantees because the accuracy of the outputs are probabilistic. As a result, DNN-based methods may have limited adoption for safety-critical applications.

Motivated to ensure performance guarantees, early works in [7]–[10] use Lyapunov-based methods for NN-based adaptive control of unknown nonlinear systems. In [7]–[9], NNs are trained with a gradient descent-based adaptive update law and used as a feedforward control term. Since the update laws are derived from a stability analysis and the NN weights are embedded inside activation functions, it is challenging to derive adaptation laws from a stability analysis beyond a single-hidden-layer.

In [11] and [12], the authors developed a data-driven adaptive learning method called concurrent learning to increase performance of parameter estimation. Concurrent learning leverages recorded input and output data concurrent to real-time execution to apply batch-like updates to adaptive update laws, and has been extended to works in [13] and [14]. Results in [15], [16], and [17] leverage concurrent learning to develop a deep Model Reference Adaptive Control (D-MRAC). Specifically, a gradient descent-based adaptive update law is used to estimate the ideal output-layer weights of a DNN in real-time online, and an offline data-driven method is used to apply batch updates to the inner-layer weights of the DNN for linear systems with known system matrices. The methods were tested on quadrotors and demonstrated that DNN-based adaptive control can significantly improve learning performance [16], [17]. The authors demonstrated that DNN

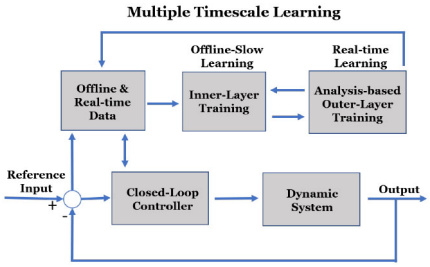

Fig. 1. Multiple timescale learning architecture.

enabled MRAC outperforms shallow NN MRAC, and also showed that the DNN weights cluster in different regions in different operating envelopes of the quadrotor, clearly establishing the learning performance of DNNs [17]. However, the D-MRAC development is specific to linear systems with known system A, B matrices with matched system uncertainty $\Delta(x(t))$, i.e., $\dot{x}(t) = Ax(t) + B(u(t) + \Delta(x(t)))$.

Building on the output-layer weight adjustment strategy in [15] and [16], this letter develops new control design and stability analysis methods for general uncertain nonlinear systems. A Lyapunov-based adaptive control law is developed to estimate the unknown output-layer weights of the DNN using real-time state feedback. Concurrent to real-time execution, data is collected and an offline function approximation method is used to update the estimates of the inner-layer DNN weights. Moreover, this letter considers control-affine dynamics with uncertain state-dependent control effectiveness matrices. To compensate for the uncertain control effectiveness, a novel adaptive update law is developed that has internal feedback. Specifically, the adaptive update law depends on the control input, and hence, is a function of both the input uncertainty estimates and the DNN weight estimates. To account for switching from iterative updates of the DNN weights, a nonsmooth Lyapunov-based analysis is performed to ensure asymptotic tracking of the desired trajectory. The proposed DNN architecture is shown in Figure 1.

II. System Dynamics

Consider a control-affine nonlinear dynamic system modeled as

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

where $x : [t_0, \infty) \to \mathbb{R}^n$ denotes the generalized state, $t_0 \in \mathbb{R}_{\geq 0}$ denotes the initial time, $f : \mathbb{R}^n \to \mathbb{R}^n$ denotes the unknown drift dynamics, $g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ denotes the uncertain control effectiveness matrix, and $u : [t_0, \infty) \to \mathbb{R}^m$ denotes the control input. To facilitate the control development, the following assumption is made.

Assumption 1: The product of the uncertain control effectiveness matrix and control input can be linearly parameterized as

$$g(x(t))u(t) = Y(x(t), u(t), t)\theta, \tag{2}$$

where $Y : \mathbb{R}^n \times \mathbb{R}^m \times [t_0, \infty) \to \mathbb{R}^{n\times q}$ denotes a measurable regression matrix, and $\theta \in \mathbb{R}^q$ denotes a vector of constant unknown parameters.

III. DNN Approximation and Update Policy

Let $\Omega \subset \mathbb{R}^n$ be a compact simply connected set with $x(t) \in \Omega$, and define $\mathbb{S}^n(\Omega)$ as the space where $f(x(t))$ is continuous. There exists ideal weights, ideal basis functions, and an ideal pre-trained DNN such that the drift dynamics $f(x(t)) \in \mathbb{S}^n(\Omega)$ can be represented as [18]

$$f(x(t)) = W^{*T}\sigma^*\left(\Phi^*(x(t))\right) + \varepsilon(x(t)), \tag{3}$$

where $W^* \in \mathbb{R}^{L\times n}$ is an unknown bounded ideal output-layer weight matrix, $\sigma^* : \mathbb{R}^p \to \mathbb{R}^L$ is an unknown bounded vector of the ideal activation functions, $\Phi^*:\mathbb{R}^n \to \mathbb{R}^p$ is the ideal unknown DNN, $\varepsilon : \mathbb{R}^n \to \mathbb{R}^n$ is the bounded unknown function reconstruction error associated with the ideal weights, activation functions, and DNN. The ideal unknown DNN Φ^* can be expressed as $\Phi^*(x(t)) = (W_k^T\phi_k \circ W_{k-1}^T\phi_{k-1} \circ \cdots \circ W_1^T\phi_1)(x(t))$, where $k \in \mathbb{Z}$ denotes the number of inner-layers of the DNN, the symbol $\circ$ denotes function composition, W and $\phi(\cdot)$ denote the corresponding inner-layer weights and activation functions of the DNN, respectively.

The DNN is updated using a multiple timescale approach. The DNN is trained *a priori* using data sets collected from previous experiments, simulation data, etc. Ideally, large data sets from the same dynamic system operating under the same environmental conditions will be available for training the DNN. However, the developed strategy of real-time (Lyapunov-based) adjustment of the output-layer weights provides an advantage of significant flexibility in the training data. For example, as observed in the subsequent simulation, the training data could be from a dynamic system with different parameters (i.e., transfer learning), or could also be initialized with random weights.

Based on (3), the DNN approximation of the drift dynamics $\hat{f}_i : \mathbb{R}^n \to \mathbb{R}^n$ can be represented as $\hat{f}_i(x(t)) = \widehat{W}^T(t)\hat{\sigma}_i(\widehat{\Phi}_i(x(t)))$, where $\widehat{W} :[t_0, \infty) \to \mathbb{R}^{L\times n}$ is the estimate of the ideal output-layer weight matrix, $\hat{\sigma}_i$: $\mathbb{R}^p \to \mathbb{R}^L$ and $\widehat{\Phi}_i : \mathbb{R}^n \to \mathbb{R}^p$ are the i^{th} activation functions and estimates of σ^* and Φ^*, respectively, and $i \in \mathbb{N}$ is the DNN estimate update index.[1] The mismatch between the ideal output-layer weights and the weight estimates $\widetilde{W}$: $[t_0, \infty) \to \mathbb{R}^{L\times n}$ is defined as

$$\widetilde{W}(t) \triangleq W^* - \widehat{W}(t). \tag{4}$$

Assumption 2: Using the universal function approximation property there exists known constants $\overline{W^*}$, $\overline{\sigma^*}$, $\hat{\bar{\sigma}}$, $\bar{\varepsilon} \in \mathbb{R}_{>0}$ such that the unknown ideal weights W^*, unknown ideal activation functions $\sigma^*(\cdot)$, user-selected activation functions $\hat{\sigma}_i(\cdot)$, the unknown ideal DNN $\Phi^*(\cdot)$, and the function reconstruction error $\varepsilon(\cdot)$ can be upper bounded such that $\sup_{x(t)\in\Omega} \|W^*\| \leq \overline{W^*}$, $\sup_{x(t)\in\Omega} \|\sigma^*(\cdot)\| \leq \overline{\sigma^*}$, $\sup_{x(t)\in\Omega,\forall i} \|\hat{\sigma}_i(\cdot)\| \leq \hat{\bar{\sigma}}$,[2] and $\sup_{x(t)\in\Omega} \|\varepsilon(\cdot)\| \leq \bar{\varepsilon}$.

[1] The subscript i on $\hat{\sigma}$ and $\widehat{\Phi}$ represents the i^{th} training iteration activation functions and estimated DNN, respectively. The explicit expression for $\widehat{\Phi}_i(x(t))$ can be expressed as $\widehat{\Phi}_i(x(t)) = (\widehat{W}_{i,k}^T\hat{\phi}_{i,k} \circ \widehat{W}_{i,k-1}^T\hat{\phi}_{i,k-1} \circ \cdots \circ \widehat{W}_{i,1}^T\hat{\phi}_{i,1})(x(t))$, where $\widehat{W}$ and $\hat{\phi}(\cdot)$ denote the corresponding estimated inner-layer weights and activation functions of the corresponding training iteration, respectively.

[2] For common activation functions, e.g., hyperbolic tangent function, sigmoid function, radial basis function, $\overline{\sigma^*} = \hat{\bar{\sigma}} = L$.

A priori training provides $\widehat{\Phi}_1(\cdot)$ and $\widehat{W}(t_0)$. The offline DNN training can be achieved by using different techniques. For example, [15] and [16] use a Stochastic Gradient Descent (SGD) based generative network architecture to generate estimates of matched system uncertainty, and the SGD update policy depends on a stochastic estimation of the expected value of the gradient of the loss function over a training set. When the offline DNN training phase is completed, an adaptive control law will be implemented for the system described in (1) to generate the output-layer DNN weight estimates, i.e., $\widehat{W}(t)$ for all $t \geq t_0$. Simultaneous to the online execution, data is collected and offline function approximation methods are used to update estimates on the inner-layer DNN weights.

IV. Control Design

A. Control Objective

The control objective is to ensure the trajectory of the system in (1) tracks a desired sufficiently smooth time-varying trajectory $x_d : [t_0, \infty) \to \mathbb{R}^n$. To quantify the tracking objective, a tracking error $e : [t_0, \infty) \to \mathbb{R}^n$ is defined as

$$e(t) \triangleq x(t) - x_d(t). \tag{5}$$

B. Control Development

To facilitate the subsequent control development, the product of the estimated control effectiveness matrix and the control input can be written as

$$\hat{g}(x(t))u(t) = Y(x(t), u(t), t)\hat{\theta}(t), \tag{6}$$

where $\hat{g} : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ denotes the estimate of the control effectiveness matrix. The parameter estimation error $\tilde{\theta} : [t_0, \infty) \to \mathbb{R}^q$ is defined as

$$\tilde{\theta}(t) \triangleq \theta - \hat{\theta}(t), \tag{7}$$

where $\hat{\theta} : [t_0, \infty) \to \mathbb{R}^q$ denotes the parameter estimate.

Assumption 3: The estimate of the control effectiveness matrix $\hat{g}$ is a full-row rank matrix for $t \geq t_0$, and the right pseudo inverse of $\hat{g}(\cdot)$ is denoted by $\hat{g}^+ : \mathbb{R}^n \to \mathbb{R}^{m\times n}$, where $\hat{g}^+(\cdot) \triangleq \hat{g}^T(\cdot)(\hat{g}(\cdot)\hat{g}^T(\cdot))^{-1}$ is bounded given a bounded argument.

Based on the subsequent stability analysis, the control input is designed as

$$u(t) \triangleq \hat{g}^+(x(t))[-ke(t) - k_s\text{sgn}(e(t)) + \dot{x}_d(t) - \widehat{W}^T(t)\hat{\sigma}_i(\widehat{\Phi}_i(x(t)))], \tag{8}$$

where $k, k_s \in \mathbb{R}_{>0}$ are constant control gains, and $\text{sgn}(\cdot)$ denotes the signum function. The weight estimate adaptation law is designed as

$$\dot{\widehat{W}}(t) \triangleq \Gamma_W \hat{\sigma}_i(\widehat{\Phi}_i(x(t)))e^T(t), \tag{9}$$

where $\Gamma_W \in \mathbb{R}^{L\times L}$ denotes a user-defined positive definite, diagonal control gain matrix. The adaptation law for the parameter estimate in (6) is designed as

$$\dot{\hat{\theta}}(t) \triangleq \Gamma_\theta Y^T(x(t), u(t), t)e(t), \tag{10}$$

where $\Gamma_\theta \in \mathbb{R}^{q\times q}$ denotes a user-defined positive definite, diagonal control gain matrix. Taking the time-derivative of (5) and substituting in (1)-(3) and (6)-(8) yields the closed-loop error system

$$\dot{e}(t) = W^{*T}\sigma^*(\Phi^*(x(t))) - \widehat{W}^T(t)\hat{\sigma}_i(\widehat{\Phi}_i(x(t))) + \varepsilon(x(t)) - ke(t) - k_s\text{sgn}(e(t)) + Y(x(t), u(t), t)\tilde{\theta}(t). \tag{11}$$

Recall the initially trained DNN provides initial estimates $\widehat{\Phi}_1(\cdot)$ and $\widehat{W}(t_0)$. During implementation of the real-time controller, the output-layer weights of the DNN are estimated online. Concurrently, the data generated in real-time is stored for additional DNN training. Once a sufficient amount of data (user-defined) is collected to improve the function approximation performance, the inner-layer weights of the DNN will be updated to generate $\widehat{\Phi}_{i+1}(\cdot)$ for all i, i.e., (8)-(10).

V. Stability Analysis

The stability of the DNN-based adaptive tracking controller is established in the following theorem.

Theorem 1: Consider a general nonlinear system modeled by the dynamics in (1) with $x(t_0) \in \Omega$ and satisfying Assumptions 1-3. The control input in (8), the output-layer weight adaptation law in (9), and the parameter estimate adaptation law in (10) ensure the trajectory tracking error defined in (5) yields semi-global asymptotic tracking in the sense that $\lim_{t\to\infty} \|e(t)\| \to 0$, $t \geq t_0$, provided the following gain condition is satisfied

$$k_s > \left(\overline{\sigma^*} + \overline{\hat{\sigma}}\right)\overline{W^*} + \overline{\varepsilon}. \tag{12}$$

Proof: Consider the candidate Lyapunov-like function $V_L : \mathbb{R}^{n(L+1)+q} \times [t_0, \infty) \to \mathbb{R}_{\geq 0}$ defined as

$$V_L(z, t) \triangleq \frac{1}{2}e^Te + \frac{1}{2}\tilde{\theta}^T\Gamma_\theta^{-1}\tilde{\theta} + \frac{1}{2}\text{tr}\left(\widetilde{W}^T\Gamma_W^{-1}\widetilde{W}\right), \tag{13}$$

where $z : [t_0, \infty) \to \mathbb{R}^{n(L+1)+q}$ is defined as $z \triangleq [e^T, \tilde{\theta}^T, \text{vec}(\widetilde{W})^T]^T$ and $\text{vec}(\cdot)$ denotes the vectorization operator. Let $\zeta : [t_0, \infty) \to \mathbb{R}^{n(L+1)+q}$ be a Filippov solution to the differential inclusion $\dot{\zeta} \in K[h](\zeta)$, where $\zeta(t) = z(t)$, the calculus of $K[\cdot]$ is used to compute Filippov's differential inclusion as defined in [19], and $h : \mathbb{R}^{n(L+1)+q} \to \mathbb{R}^{n(L+1)+q}$ is defined as $h(\zeta) \triangleq [\dot{e}^T, \dot{\tilde{\theta}}^T, \text{vec}(\dot{\widetilde{W}})^T]^T$. The time-derivative of V_L exists almost everywhere (a.e.), i.e., for almost all $t \in [0, \infty)$, $\dot{V}_L(\zeta) \overset{\text{a.e.}}{\in} \dot{\tilde{V}}_L(\zeta)$, where $\dot{\tilde{V}}_L(\zeta)$ is the generalized time-derivative of V_L along the Filippov trajectories of $\dot{\zeta} = h(\zeta)$. By [20, Equation 13], $\dot{\tilde{V}}_L(\zeta) \triangleq \bigcap_{\xi\in\partial V_L(\zeta)} \xi^T[K[h]^T(\zeta), 1]^T$, where $\partial V_L(\zeta(t))$ denotes the Clarke generalized gradient of $V_L(\zeta)$. Since $V_L(\zeta)$ is continuously differentiable in ζ, then $\partial V_L(\zeta) = \{\nabla V_L(\zeta)\}$, where ∇ denotes the gradient operator.

Taking the generalized time-derivative of (13), then substituting in the mismatch between the ideal output-layer weight and the weight estimate in (4), the output-layer adaptation law in (9), the parameter estimate adaptation law in (10), and the closed-loop error system in (11) yields

$$\dot{\tilde{V}}_L(\zeta) \subseteq e^T\left(W^{*T}\sigma^*(\Phi^*(x)) - \widehat{W}^TK\left[\hat{\sigma}_i(\widehat{\Phi}_i(x))\right]\right) + e^T\left(\varepsilon(x) - ke - k_sK\left[\text{sgn}(e)\right]\right) - \text{tr}\left(\widetilde{W}^TK\left[\hat{\sigma}_i(\widehat{\Phi}_i(x))\right]e^T\right). \tag{14}$$

Using the trace operator property,[3] the estimated mismatch for the ideal output-layer weight in (4), and adding and subtracting $e^T W^{*T} K[\hat{\sigma}_i(\widehat{\Phi}_i(x))]$ in (14) yields

$$\dot{\tilde{V}}_L(\zeta) \leq -ke^T e - k_s e^T K\big[\operatorname{sgn}(e)\big] + e^T \varepsilon(x) + e^T W^{*T}\big(\sigma^*\big(\Phi^*(x)\big) - K\big[\hat{\sigma}_i\big(\widehat{\Phi}_i(x)\big)\big]\big). \quad (15)$$

Hence, using the definition of $K[\operatorname{sgn}(e)]$ and Assumption 2, (15) can be upper bounded as

$$\dot{V}_L \overset{\text{a.e.}}{\leq} -k\|e\|^2 - \Big(k_s - \Big(\overline{\sigma^*} + \overline{\hat{\sigma}}\Big)\overline{W^*} - \overline{\varepsilon}\Big)\|e\|. \quad (16)$$

By satisfying the gain condition described in (12), (16) can be further upper bounded as

$$\dot{V}_L \overset{\text{a.e.}}{\leq} -k\|e\|^2. \quad (17)$$

Using (13) and (17) implies $V_L(z,t) \in \mathcal{L}_\infty$, which implies $z(t) \in \mathcal{L}_\infty$. The definition of $z(t)$ implies $e(t), \tilde{\theta}(t), \widetilde{W}(t) \in \mathcal{L}_\infty$. Using (4), (5) and (7) implies $x(t), \hat{\theta}(t), \widehat{W}(t) \in \mathcal{L}_\infty$. Using Assumptions 2 and 3 implies $\hat{\sigma}_i(\cdot)$, $\varepsilon(x(t))$, $\hat{g}^+(x(t))$, $\hat{g}(x(t)) \in \mathcal{L}_\infty$. Since $e(t), \hat{g}^+(x(t)), \widehat{W}(t) \in \mathcal{L}_\infty$, using (8) implies $u(t) \in \mathcal{L}_\infty$. Since $\hat{g}(x(t)), u(t), \hat{\theta}(t) \in \mathcal{L}_\infty$, using (6) yields $Y(x(t), u(t), t) \in \mathcal{L}_\infty$. Furthermore, by the extension of the LaSalle-Yoshizawa theorem for non-smooth systems in [21] and [22], $k\|e\|^2 \to 0$, which implies $\|e(t)\| \to 0$ as $t \to \infty$ and $x(t) \in \Omega$ for all $t \geq t_0$. ■

VI. Simulation

To demonstrate the effectiveness of the developed method, a simulation is performed on a control-affine realization of a two-state Van der Pol oscillator. The dynamics used in the simulation are

$$f(x) = \begin{bmatrix} \mu\Big(x_1 - \frac{1}{3}x_1^3 - x_2\Big) \\ \frac{1}{\mu}x_1 \end{bmatrix}, \quad (18)$$

$$g(x) = \begin{bmatrix} 5 & 0 \\ 0 & 3 \end{bmatrix}, \quad (19)$$

where $x = [x_1, x_2]^T$ and $\mu = 10$. The desired trajectory is $x_d = [5\cos(t), 5\sin(t)]^T$. The initial conditions of the system were $x(0) = [-5, 8]^T$ and $\hat{\theta}(0) = [6, 6]^T$. The user-defined parameters were selected as $k = 75$, $k_s = 0.05$, $\Gamma_W = 500 \cdot I_{13\times 13}$, and $\Gamma_\theta = \operatorname{diag}([0.1, 0.05])$.

The DNN used in this simulation was composed of 4 layers, each with 10, 5, 8, and 2 neurons, respectively. The DNN architecture is illustrated in Figure 2. Each layer is linear and the first, second, and third layers have tangent-sigmoid, logarithmic-sigmoid, and tangent-sigmoid activation functions, respectively. The learning rate (i.e., the learning gain parameter used to determine the step size in retraining the DNN weights at each iteration) was fixed as $\eta = 0.001$. The mean squared error (MSE) was used as the loss function for training. Each training iteration lasted until the MSE (i.e., the loss) was less than 10^{-3}. 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.

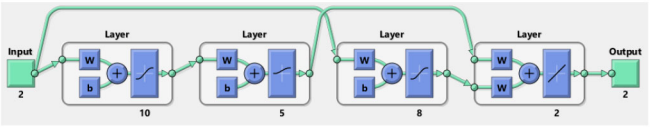

Fig. 2. The DNN is composed of 4 layers, each with 10, 5, 8, and 2 neurons, respectively.

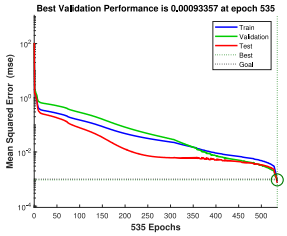

Fig. 3. The MSE for the offline trained DNN with learning rate $\eta = 0.001$ for 535 epochs in logarithmic scale. The best validation performance for DNN1 is MSE = 0.00093357.

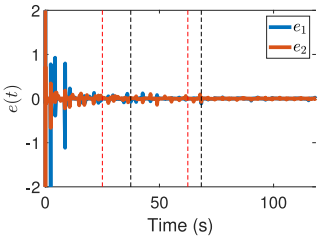

Fig. 4. The tracking error over three iterations of DNN training, i.e., DNN1, DNN2, DNN3. At $t = 0$ seconds, the first iteration of the DNN (DNN1) is deployed. The red dashed line at $t = 25$ seconds represents the beginning of the first retraining (DNN2), and the black dashed line at $t = 37.4$ seconds represents the end of the first retraining. The red dashed line at $t = 62.4$ seconds represents the beginning of the second retraining (DNN3), and the black dashed line at $t = 68.3$ seconds represents the end of the second retraining.

To pre-train the DNN, a 600 second simulation of a system with dynamics in (18) and $\mu = 10$ was performed. Training statistics for the offline training are shown in Figure 3. The real-time controller and the update laws in (8)-(10) are used to update their respective parametric estimates. Concurrent to the real-time controller execution, input-output data is collected to retrain the DNN. As shown in Figures 4–6, the training start time is denoted by the red dashed vertical line and the training completion time is denoted by the black dashed vertical line. The time (and corresponding amount of data) between retraining the inner-layer weights is a user-defined parameter of the simulation. After the prescribed time between retraining elapses, the inner-layer DNN weights begin updating via retraining. In this simulation, the time between retraining is 25 seconds. The first retraining starts at $t = 25$ seconds and ends at $t = 37.4$ seconds. The second retraining starts at $t = 62.4$ seconds and ends at $t = 68.3$ seconds. While the retraining is in process, the real-time controller and update laws continue uninterrupted as described in (8)-(10). Once the retraining is completed, the new inner-layer DNN weights are updated, overwriting the previous values.

[3]For real column matrices $a, b \in \mathbb{R}^n$, the trace of the outer product is equivalent to the inner product, i.e., $\operatorname{tr}(ba^T) = a^T b$.

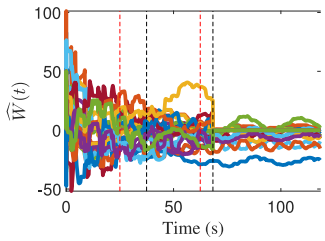

Fig. 5. Weight estimates over three iterations of DNN training, i.e., DNN1, DNN2, DNN3.

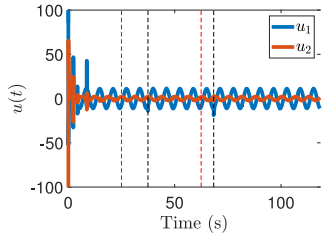

Fig. 6. Applied control input over three iterations of DNN training, i.e., DNN1, DNN2, DNN3.

As shown in Figures 4–6, the time for the MSE to be less than 10^{-3} took 12.4 seconds to complete. After the DNN has completed retraining, the controller implements the inner-layer weights at $t = 37.4$ seconds. After implementing the updated DNN weights, new data is collected for another 25 seconds. To further improve the DNN estimate, a second retraining is performed. During the second training iteration, data from the first 25 seconds and the second 25 seconds are both used. The second retraining took 5.9 seconds. The inner-layer weights from the second retraining are implemented at $t = 68.3$ seconds.

The tracking error performance in Figure 4 indicates that discretely retrained DNNs with an online adaptive output-layer weights are a viable method to perform trajectory tracking. The first iteration of the DNN (DNN1) is the offline generated DNN, DNN2 is the model after the first retraining, and DNN3 is the model after the second retraining. As shown by the root mean squared error (RMSE) in Table I (A), each subsequent DNN training iteration yielded improved performance, where $e \triangleq [\, e_1 \; e_2 \,]^T$. The decrease in error after each retraining is expected since a larger set of system data was used to train the DNN during each retraining. Figure 7 shows the phase plot of the system, and compares the performance of the tracking during the application of each DNN. DNN1 has the worst estimate of the system dynamics. DNN2 and DNN3 show significantly better tracking behavior, which is also reflected in Figure 4. Figure 6 presents the control input to the system for the duration of the simulation. DNN1 poorly approximates the dynamics near $x = [-3.5, 3.5]^T$, and this error is further reflected in Figure 6 with the spikes in control input approximately at $t = 2$ seconds and $t = 8$ seconds.

A. Transfer Learning & Random Weights

To further demonstrate the flexibility of the developed real-time Lyapunov-based adjustment of the output-layer weights, two additional simulations were performed. In this section,

TABLE I
ROOT MEAN SQUARED ERRORS (RMSE)

		RMSE e_1	RMSE e_2	RMSE e
A	DNN1	0.18	0.18	0.25
	DNN2	0.01	0.02	0.02
	DNN3	0.01	0.00	0.01
B	DNN1	0.17	0.18	0.24
	DNN2	0.06	0.02	0.07
	DNN3	0.01	0.01	0.02
C	DNN1	0.21	0.22	0.29
	DNN2	0.02	0.00	0.02
	DNN3	0.01	0.01	0.01

RMSE of the tracking error for each simulation: (A) DNN pre-trained with dynamics in (18) parametrized with $\mu = 10$, (B) DNN pre-trained with dynamics in (18) parametrized with $\mu = 1$, (C) DNN initialized with randomly selected inner-layer weights.

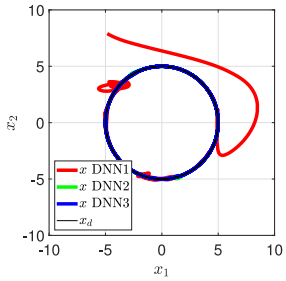

Fig. 7. Phase plot of the dynamics in (18) over three iterations of DNN training, i.e., DNN1, DNN2, DNN3.

transfer learning-based and randomly initialized DNN weights simulations were investigated. Transfer learning in this context is applying the learned DNN model of one system to another system. In the simulation, transfer learning is demonstrated by training a DNN model on a dataset of a system described by the dynamics in (18) with parameter $\mu = 1$, whereas the simulated system has parameter $\mu = 10$.

In the transfer learning-based approach, the DNN is pre-trained with 600 seconds of simulated data from a system with dynamics in (18), but parametrized with $\mu = 1$. Figures 8(a), 9(a), and Table I (B) show the tracking error, phase plot, and RMSE of the transfer learning approach over three iterations of DNN training, respectively. For situations where data cannot be collected *a priori*, initial inner-layer DNN weights can be selected by the user. In the third simulation, instead of pre-training the DNN, a simulation was performed with the initial DNN randomly selected weights. Figures 8(b), 9(b), and Table I (C) show the tracking error, phase plot, and RMSE of this approach over three iterations of DNN training, respectively.

The performance of the transfer learning-based approach and initial randomly selected DNN weights simulations are depicted in Figures 8 and 9. Iterations in the inner-layer weights are shown to improve performance. The first simulation, which was trained with 600 seconds of offline data using the dynamics in (18) with $\mu = 10$ has the best performance with respect to the smallest RMSE within 25

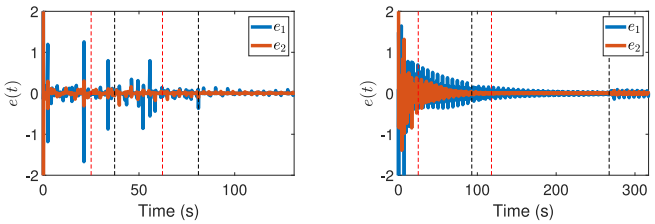

Fig. 8. Tracking error for (a) transfer learning, and (b) randomly selected initial inner-layer weights.

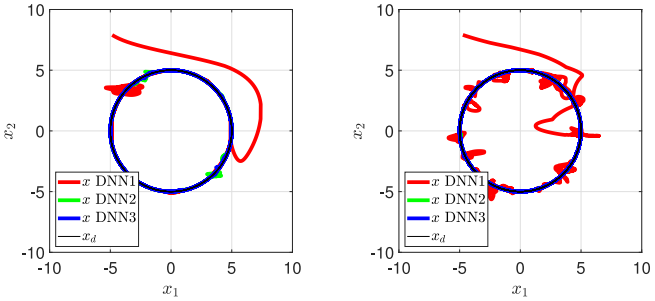

Fig. 9. The phase plot for (a) transfer learning, and (b) randomly selected initial inner-layer weights.

second intervals compared to transfer learning and initial randomly selected DNN weights. Nevertheless, the proposed real-time Lyapunov-based adjustment of the output-layer weights accommodates for different methods to initialize the DNN inner-layer weights.

VII. Conclusion

A multiple timescale DNN-based adaptive control architecture is developed for general nonlinear dynamical systems with unknown drift dynamics and uncertain control effectiveness matrix. Specifically, a Lyapunov-based adaptive update law is developed to estimate the unknown output-layer weights of the DNN and the uncertain control effectiveness matrix in real-time. Simultaneous to real-time execution, data is collected and offline function approximation methods are used to update estimates of the inner-layer weights. A nonsmooth Lyapunov-based analysis is performed to ensure semi-global asymptotic tracking of the desired trajectory. Numerical simulation examples are provided to demonstrate the performance of the proposed architecture.

Acknowledgment

Any opinions, findings and conclusions or recommendations expressed in this letter are those of the author(s) and do not necessarily reflect the views of the sponsoring agency.

References

[1] T. Parisini and R. Zoppoli, "A receding-horizon regulator for nonlinear systems and a neural approximation," *Automatica*, vol. 31, no. 10, pp. 1443–1451, 1995.
[2] B. Karg and S. Lucia, "Efficient representation and approximation of model predictive control laws via deep learning," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3866–3878, Sep. 2020.
[3] X. Zhang, M. Bujarbaruah, and F. Borrelli, "Near-optimal rapid MPC using neural networks: A primal-dual policy learning framework," *IEEE Trans. Control Syst. Technol.*, early access, Oct. 30, 2020, doi: 10.1109/TCST.2020.3024571.
[4] M. Hertneck, J. Köhler, S. Trimpe, and F. Allgöwer, "Learning an approximate model predictive controller with guarantees," *IEEE Control Syst. Lett.*, vol. 2, no. 3, pp. 543–548, Jul. 2018.
[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[6] Q. Teng and L. Zhanga, "Data driven nonlinear dynamical systems identication using multi-step CLDNN," *AIP Adv.*, vol. 9, no. 8, 2019, Art. no. 085311.
[7] F. L. Lewis, "Nonlinear network structures for feedback control," *Asian J. Control*, vol. 1, no. 4, pp. 205–228, 1999.
[8] P. M. Patre, W. MacKunis, K. Kaiser, and W. E. Dixon, "Asymptotic tracking for uncertain dynamic systems via a multilayer neural network feedforward and RISE feedback control structure," *IEEE Trans. Autom. Control*, vol. 53, no. 9, pp. 2180–2185, Oct. 2008.
[9] P. M. Patre, S. Bhasin, Z. D. Wilcox, and W. E. Dixon, "Composite adaptation for neural network-based controllers," *IEEE Trans. Autom. Control*, vol. 55, no. 4, pp. 944–950, Apr. 2010.
[10] W. Mackunis, F. Leve, P. Patre, N. Fitz-Coy, and W. E. Dixon, "Adaptive neural network-based satellite attitude control in the presence of CMG uncertainty," *Aerosp. Sci. Technol.*, vol. 54, pp. 218–228, Jul. 2016.
[11] 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./Apr. 2011.
[12] 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.
[13] Z. Bell, J. Nezvadovitz, A. Parikh, E. M. Schwartz, and W. E. Dixon, "Global exponential tracking control for an autonomous surface vessel: An integral concurrent learning approach," *IEEE J. Ocean Eng.*, vol. 45, no. 2, pp. 362–370, Apr. 2020.
[14] A. Parikh, R. Kamalapurkar, and W. E. Dixon, "Integral concurrent learning: Adaptive control with parameter convergence using finite excitation," *Int. J. Adapt. Control Signal Process.*, vol. 33, no. 12, pp. 1775–1787, Dec. 2019.
[15] G. Joshi and G. Chowdhary, "Deep model reference adaptive control," in *Proc. IEEE Conf. Decis. Control (CDC)*, 2019, pp. 4601–4608.
[16] G. Joshi, J. Virdi, and G. Chowdhary, "Design and flight evaluation of deep model reference adaptive controller," in *Proc. AIAA Scitech Forum*, 2020, p. 1336.
[17] G. Joshi, J. Virdi, and G. Chowdhary, "Asynchronous deep model reference adaptive control," 2020. [Online]. Available: arXiv:2011.02920.
[18] F. L. Lewis, S. Jagannathan, and A. Yesildirak, *Neural Network Control of Robot Manipulators and Nonlinear Systems*. Philadelphia, PA, USA: CRC Press, 1998.
[19] B. E. Paden and S. S. Sastry, "A calculus for computing Filippov's differential inclusion with application to the variable structure control of robot manipulators," *IEEE Trans. Circuits Syst.*, vol. 34, no. 1, pp. 73–82, Jan. 1987.
[20] D. Shevitz and B. Paden, "Lyapunov stability theory of nonsmooth systems," *IEEE Trans. Autom. Control*, vol. 39, no. 9, pp. 1910–1914, Sep. 1994.
[21] N. Fischer, R. Kamalapurkar, and W. E. Dixon, "LaSalle-Yoshizawa corollaries for nonsmooth systems," *IEEE Trans. Autom. Control*, vol. 58, no. 9, pp. 2333–2338, Sep. 2013.
[22] R. Kamalapurkar, J. A. Rosenfeld, A. Parikh, A. R. Teel, and W. E. Dixon, "Invariance-like results for nonautonomous switched systems," *IEEE Trans. Autom. Control*, vol. 64, no. 2, pp. 614–627, Feb. 2019.