Real-Time Modular Deep Neural Network-Based Adaptive Control of Nonlinear Systems

Duc M. Le, Max L. Greene, *Student Member, IEEE*, Wanjiku A. Makumi, and Warren E. Dixon, *Fellow, IEEE*

***Abstract*—A real-time deep neural network (DNN) adaptive control architecture is developed for uncertain control-affine nonlinear systems to track a time-varying desired trajectory. A Lyapunov-based analysis is used to develop adaptation laws for the output-layer weights and develop constraints for inner-layer weight adaptation laws. Unlike existing works in neural network and DNN-based control, the developed method establishes a framework to simultaneously update the weights of multiple layers for a DNN of arbitrary depth in real-time. The real-time controller and weight update laws enable the system to track a time-varying trajectory while compensating for unknown drift dynamics and parametric DNN uncertainties. A non-smooth Lyapunov-based analysis is used to guarantee semi-global asymptotic tracking. Comparative numerical simulation results are included to demonstrate the efficacy of the developed method.**

***Index Terms*—Adaptive control, deep neural networks, Lyapunov methods, nonlinear control systems.**

I. Introduction

NEURAL networks (NNs) have gained popularity due to their ability to approximate nonlinear functions. Conventional NNs can approximate functions to a prescribed accuracy [1] and [2]; however, recent evidence indicates deep neural networks (DNNs) exploit more complex learning features that can potentially improve function approximation performance [3]. Although DNNs may potentially approximate the nonlinear dynamics of a system more accurately, it is difficult to derive real-time adaptation laws for DNNs with multiple layers because the uncertain ideal weights are nested within a collection of nonlinear activation functions.

Manuscript received March 4, 2021; revised May 6, 2021; accepted May 10, 2021. Date of publication May 17, 2021; date of current version June 25, 2021. This work was supported in part by the Office of Naval Research under Grant N00014-13-1-0151; in part by NEEC under Award N00174-18-1-0003; in part by AFOSR under Award FA9550-18-1-0109 and Award FA9550-19-1-0169; in part by AFRL under Award FA8651-19-2-0009; and in part by NSF under Award 1762829. Recommended by Senior Editor C. Seatzu. *(Corresponding author: Duc M. Le.)*

The authors are with the Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: ledan50@ufl.edu; maxgreene12@ufl.edu; makumiw@ufl.edu; wdixon@ufl.edu).

Digital Object Identifier 10.1109/LCSYS.2021.3081361

Results in [4]–[7] leverage Lyapunov-based analysis to develop multi-timescale DNN-based controllers containing real-time and offline iterative learning components. The output-layer weights of the DNN are adjusted online (i.e., in real-time) using NN-based adaptive control techniques. Concurrent to real-time execution, data is collected and DNN training algorithms, such as gradient descent and stochastic gradient descent (see [4], [5], [7] and [8, Ch. 8]), are used to iteratively update the inner-layer DNN weights. Since DNN learning algorithms are performed iteratively, the inner-layer weights are not updated continuously in real-time. The benefit of iterative learning is that the system performance improves with the quality of the DNN estimate. However, improving the quality of the DNN estimate may require a large training data set to capture the nonlinearities of the dynamics and significant computational resources to adjust the inner-layer weights. The DNN-based methods in [4]–[7] also raise questions regarding the inner-layer weights updates such as: when to collect data, what is the most efficient way to retrain the inner-layer weights, when should the inner-layer weights be updated in the implemented adaptation law, etc.

While such open questions are topics for further investigation, this letter investigates general characteristics and structures of inner-layer adaptation laws. Specifically, this letter develops general constraints on the inner-layer adaptation laws to update the inner-layer weight estimates in real-time. A Lyapunov-based analysis is used to develop a continuous adaptation law to estimate the output-layer weights. However, unlike previous methods, this letter provides a first insight into the development of Lyapunov-based adaptive update laws for both the inner-layer DNN weights as well as the output-layer weights. Inspired by modular adaptive control designs in [9]–[12], general constraints on the inner-layer DNN weight update laws are developed that enable modular design and selection of update laws. The developed DNN-based modular adaptive architecture allows more flexibility when selecting inner-layer DNN weight update laws.

In arbitrary width and depth DNNs, there may be hundreds or thousands of inner-layer weights. Simultaneously updating all the inner-layer weights online may be computationally intractable in real-time or undesired. Hence, the developed method provides a switched framework that provides design guidelines that can be used in future research efforts to guide

inner-layer weight adaptive update laws. In doing so, the inner-layer weight update laws may be arbitrarily switched on and off to allocate computational resources while updating the desired weights. Additionally, inner-layer weights may dropout, or be selectively turned off to prevent over-fitting and improve overall function approximation performance [13].

Results such as [6], [14], and [15] develop a robust sliding mode method to achieve asymptotic tracking with NN feedforward controllers. Like the aforementioned results, the developed method uses a sliding mode control term to yield asymptotic tracking in the presence of the NN reconstruction error, but also uses switched adaptation laws for the inner-layer weights. Hence, this letter leverages a nonsmooth Lyapunov-like analysis [16] to guarantee asymptotic tracking of a desired trajectory. Unlike existing works, the developed modular adaptive architecture incorporates the inner-layer weights of an arbitrarily deep DNN into a Lyapunov-based analysis to develop and characterize a general class of suitable inner-layer DNN adaptation laws. Moreover, sufficient conditions are developed to guarantee the tracking objective is achieved, despite the arbitrary number of inner-layers and switched update laws.

Notation: For a square matrix $A \in \mathbb{R}^{n\times n}$, the trace of A is denoted by $\text{tr}(A)$. Let $0_{n\times m}$ denote an $n \times m$ matrix of zeros. Let $\text{vec}(\cdot)$ denote the vectorization operator that transforms a matrix into a column vector, e.g., for a matrix $A \in \mathbb{R}^{n\times m}$, $\text{vec}(A) \in \mathbb{R}^{nm}$.

II. Problem Formulation

A. System Dynamics

Consider a control-affine nonlinear dynamic system modeled as

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

where $x : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ denotes the state, $f : \mathbb{R}^n \to \mathbb{R}^n$ denotes unknown, locally Lipschitz drift dynamics, $g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ denotes the known control effectiveness matrix,[1] and $u : \mathbb{R}_{\geq 0} \to \mathbb{R}^m$ denotes the control input. To facilitate the subsequent control design, the following assumption is made on the control effectiveness matrix. The control effectiveness matrix $g(x)$ is assumed to be full-row rank for all $x \in \mathbb{R}^n$. The right pseudo inverse of $g(x)$ is denoted by $g^+ : \mathbb{R}^n \to \mathbb{R}^{m\times n}$, where $g^+(\cdot) \triangleq g^T(\cdot)(g(\cdot)g^T(\cdot))^{-1}$ is assumed to be bounded given a bounded argument.

B. Control Objective

The control objective is to track a user-defined time-varying trajectory $x_d : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ despite unknown system drift dynamics. The desired trajectory and its time derivative are assumed to be continuous and bounded, i.e., $x_d, \dot{x}_d \in \mathcal{L}_\infty$. To quantify the tracking objective, the tracking error $e : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ is defined as

$$e \triangleq x - x_d. \tag{2}$$

[1]While the developed method does not account for an uncertain control effectiveness matrix for simplicity and to better focus the result on the unique specific contributions, the method in [6] can be used with the developed method to approximate the uncertain control effectiveness matrix online.

III. Control Design

A. Feedforward DNN Estimate

NN-based adaptive control architectures are well-suited for uncertain or unstructured models, as in (1) where the drift dynamics $f(\cdot)$ are unknown. Using the universal function approximation property in [2], a DNN-based feedforward estimate of the drift dynamics is developed in this section. Let $\Omega \subset \mathbb{R}^n$ be a compact simply connected set and define $\mathbb{C}(\Omega)$ as the space where $f : \Omega \to \mathbb{R}^n$ is continuous. The universal function approximation property states there exist ideal weights and basis functions such that the drift dynamics $f(x) \in \mathbb{C}(\Omega)$ can be represented as

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

where $W^* \in \mathbb{R}^{L\times n}$ denotes the unknown ideal output-layer weight matrix of the DNN, $\sigma^* : \mathbb{R}^p \to \mathbb{R}^L$ denotes the unknown vector of ideal activation functions corresponding to the output-layer of the DNN, $L \in \mathbb{Z}_{>0}$ denotes the user-defined number of neurons used in the output-layer, $\varepsilon : \mathbb{R}^n \to \mathbb{R}^n$ denotes the unknown function reconstruction error, and $\Phi^* : \mathbb{R}^n \to \mathbb{R}^p$ denotes the inner-layers of the DNN containing unknown ideal weight matrices and activation functions. Specifically, the ideal inner DNN Φ^* can be expressed as

$$\Phi^*(x) \triangleq \big(V_k^{*T}\phi_k^* \circ V_{k-1}^{*T}\phi_{k-1}^* \circ \cdots \circ V_1^{*T}\phi_1^*\big)\big(V_0^{*T}x\big), \tag{4}$$

where $k \in \mathbb{Z}_{>0}$ denotes the user-defined number of inner-layers, $V_j^* \in \mathbb{R}^{L_j\times L_{j+1}}$ for all $j \in \{0, \ldots, k\}$ denotes the j^{th} inner-layer ideal weight matrix, and $\phi_j^* : \mathbb{R}^{L_j} \to \mathbb{R}^{L_j}$ for all $j \in \{1, \ldots, k\}$ denotes the j^{th} inner-layer vector of ideal activation functions, and the symbol $\circ$ denotes function composition, e.g., $(g \circ h)(x) = g(h(x))$. The user-selected parameters $L_j \in \mathbb{Z}_{>0}$ for all $j \in \{1, \ldots, k\}$ denote the number of neurons in the j^{th} inner-layer. Note that $L_0 = n$ and $L_{k+1} = p$.

Based on (3), the DNN feedforward estimate of the drift dynamics $\hat{f} : \mathbb{R}^n \to \mathbb{R}^n$ is defined as

$$\hat{f}(x) \triangleq \hat{W}^T\hat{\sigma}\Big(\hat{\Phi}(x)\Big), \tag{5}$$

where $\hat{W} : \mathbb{R}_{\geq 0} \to \mathbb{R}^{L\times n}$ denotes the output-layer weight matrix estimate, $\hat{\sigma} : \mathbb{R}^p \to \mathbb{R}^L$ denotes the user-selected vector of activation functions, and $\hat{\Phi} : \mathbb{R}^n \to \mathbb{R}^p$ denotes the estimated inner-layers of the DNN. The inner-layer DNN estimate is defined as

$$\hat{\Phi}(x) \triangleq \Big(\hat{V}_k^T\hat{\phi}_k \circ \hat{V}_{k-1}^T\hat{\phi}_{k-1} \circ \cdots \circ \hat{V}_1^T\hat{\phi}_1\Big)\Big(\hat{V}_0^T x\Big), \tag{6}$$

where $\hat{V}_j : \mathbb{R}_{\geq 0} \to \mathbb{R}^{L_j\times L_{j+1}}$ for all $j \in \{0, \ldots, k\}$ denotes the j^{th} inner-layer estimated weight matrix, and $\hat{\phi}_j : \mathbb{R}^{L_j} \to \mathbb{R}^{L_j}$ for all $j \in \{1, \ldots, k\}$ denotes the j^{th} inner-layer vector of activation functions. The design of the update laws on the weight estimates $\hat{W}$ and $\hat{V}_j$ are subsequently defined. The weight estimate mismatch of the ideal output-layer weight $\tilde{W} : \mathbb{R}_{\geq 0} \to \mathbb{R}^{L\times n}$ and weight estimate mismatch of the ideal inner-layer weights $\tilde{V}_j : \mathbb{R}_{\geq 0} \to \mathbb{R}^{L_j\times L_{j+1}}$ for all $j \in \{0, \ldots, k\}$ are defined as

$$\tilde{W} \triangleq W^* - \hat{W}, \tag{7}$$

$$\tilde{V}_j \triangleq V_j^* - \hat{V}_j. \tag{8}$$

It is assumed there exist known constants $\overline{W^*}, \overline{V^*}, \overline{\sigma^*}, \overline{\hat{\sigma}}, \overline{\varepsilon} \in \mathbb{R}_{\geq 0}$ that upper bound the unknown ideal weights W^*, unknown ideal weights V_j^*, unknown ideal bounded activation functions[2] $\sigma^*(\cdot)$, user-selected bounded activation functions $\hat{\sigma}(\cdot)$, and the function reconstruction error $\varepsilon(\cdot)$, respectively, as $\sup_{x\in\Omega} \|W^*\|_F \leq \overline{W^*}$, $\sup_{x\in\Omega,\forall j} \|V_j^*\|_F \leq \overline{V^*}$, $\sup_{x\in\Omega} \|\sigma^*(\cdot)\| \leq \overline{\sigma^*}$, $\sup_{x\in\Omega} \|\hat{\sigma}(\cdot)\| \leq \overline{\hat{\sigma}}$, and $\sup_{x\in\Omega} \|\varepsilon(\cdot)\| \leq \overline{\varepsilon}$ [1].

B. Control Development

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

$$u \triangleq g^{+}(x)\Big(\dot{x}_d - k_1 e - k_s \operatorname{sgn}(e) - \hat{f}(x)\Big), \tag{9}$$

where $k_1, k_s \in \mathbb{R}_{>0}$ are user-defined control gains, and $\operatorname{sgn}(\cdot)$ denotes the signum function. Based on the subsequent Lyapunov-based stability analysis, the output-layer weight estimate update law $\dot{\hat{W}} : \mathbb{R}_{\geq 0} \to \mathbb{R}^{L\times n}$ is designed as

$$\dot{\hat{W}} \triangleq \Gamma_W \hat{\sigma}\Big(\hat{\Phi}(x)\Big) e^T, \tag{10}$$

where $\Gamma_W \in \mathbb{R}^{L\times L}$ denotes a user-defined positive definite gain matrix used to adjust the learning rate of the output-layer weight matrix estimate.

Taking the time derivative of (2) and substituting in (1), (3), (5), and (9) yields the closed-loop error system

$$\dot{e} = W^{*T}\sigma^*\big(\Phi^*(x)\big) + \varepsilon(x) - k_1 e - k_s \operatorname{sgn}(e) - \hat{W}^T \hat{\sigma}\Big(\hat{\Phi}(x)\Big). \tag{11}$$

In [6], the inner-layer weights of the DNN were held constant and only updated discretely with data-driven learning algorithms. Common learning algorithms include gradient descent variants (see [4], [5], [7], and [8, Ch. 8]). These algorithms often use training data sets to update DNN weights through an optimization process in which the algorithms seek to minimize a cost function. However, DNN training algorithms often require large amounts of training data and high computational costs [3], making real-time execution intractable. Motivated to execute real-time learning and allow flexibility in user-selection of training algorithms while maintaining stability guarantees, we develop modular inner-layer DNN weight estimate update laws (see [17] for single-hidden-layer NNs).

For all $j \in \{0, \ldots, k\}$, the j^{th} inner-layer weight update law $\dot{\hat{V}}_j : \mathbb{R}_{\geq 0} \to \mathbb{R}^{L_j \times L_{j+1}}$ is designed as

$$\dot{\hat{V}}_j \triangleq p_j(t) \nu_j(e, t) \mathbf{1}_{\left\{\underline{\hat{V}_j} \leq \left\|\hat{V}_j\right\|_F \leq \overline{\hat{V}_j}\right\}}, \tag{12}$$

where $p_j : \mathbb{R}_{\geq 0} \to \{0, 1\}$ denotes a switching signal that indicates the active inner-layer weight estimate updates, $\mathbf{1}_{\{\cdot\}}$ is the indicator function, $\underline{\hat{V}_j}, \overline{\hat{V}_j} \in \mathbb{R}$ are user-defined constants where $\overline{\hat{V}_j} \leq \overline{V^*}$, and $\nu_j : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^{L_j \times L_{j+1}}$ denotes a user-defined function that satisfies

$$\left\|\nu_j\right\|_F \leq \rho(\|e\|)\|e\|, \tag{13}$$

where $\rho : \mathbb{R}^n \to \mathbb{R}^n$ is a positive, globally invertible, and non-decreasing function.

IV. Stability Analysis

Theorem 1: Consider a system modeled by the dynamics in (1) with the initial condition $x(0) \in \Omega$. Then the control input in (9), output-layer weight adaptation law in (10), and the family of potential inner-layer weight adaptation laws that satisfy (12) ensure the closed-loop error system in (11) yields semi-global asymptotic tracking in the sense that $\lim_{t\to\infty} \|e(t)\| = 0$, provided the following sufficient gain condition is satisfied

$$k_s > \overline{W^*}\Big(\overline{\sigma^*} + \overline{\hat{\sigma}}\Big) + \overline{\varepsilon} + 2\overline{V^*}(k+1)\rho\left(\sqrt{\frac{\overline{\alpha}}{\underline{\alpha}}}\|z(0)\|\right), \tag{14}$$

where $\underline{\alpha}, \overline{\alpha} \in \mathbb{R}_{\geq 0}$ are known constants.

Proof: Let $\mathcal{D} \subset \mathbb{R}^{\Psi}$ be a set containing $z = 0_{\Psi\times 1}$ and Ω, where $z : \mathbb{R}_{\geq 0} \to \mathbb{R}^{\Psi}$ denotes a concatenated state defined as $z \triangleq [\, e^T, \operatorname{vec}(\tilde{W})^T, \operatorname{vec}(\tilde{V}_0)^T, \ldots, \operatorname{vec}(\tilde{V}_k)^T \,]^T$, and $\Psi \triangleq n(L+1) + \sum_{j=0}^{k} L_j L_{j+1}$ is defined for notional brevity. Consider the candidate Lyapunov function $V_L : \mathcal{D} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ defined as

$$V_L(z, t) \triangleq \frac{1}{2} e^T e + \frac{1}{2}\operatorname{tr}\Big(\tilde{W}^T \Gamma_W^{-1} \tilde{W}\Big) + \frac{1}{2}\sum_{j=0}^{k} \operatorname{tr}\Big(\tilde{V}_j^T \tilde{V}_j\Big), \tag{15}$$

which satisfies the inequality $\underline{\alpha}\|z\|^2 \leq V_L(z, t) \leq \overline{\alpha}\|z\|^2$, where $\underline{\alpha}, \overline{\alpha} \in \mathbb{R}_{\geq 0}$ are known constants. Let $\zeta : \mathbb{R}_{\geq 0} \to \mathbb{R}^{\Psi}$ be a Filippov solution to the differential inclusion $\dot{\zeta} \in K[h](\zeta, t)$, where $\zeta(t) = z(t)$, the calculus of $K[\,\cdot\,]$ is used to compute Filippov's differential inclusion as defined in [18], and $h : \mathbb{R}^{\Psi} \times \mathbb{R}_{\geq 0} \to \mathbb{R}^{\Psi}$ is defined as $h(\zeta, t) \triangleq [\, \dot{e}^T, \operatorname{vec}(\dot{\tilde{W}})^T, \operatorname{vec}(\dot{\tilde{V}}_0)^T, \ldots, \operatorname{vec}(\dot{\tilde{V}}_k)^T \,]^T$. The generalized time-derivative of V_L along the Filippov trajectories of $\dot{\zeta} = h(\zeta, t)$ is defined by $\dot{\tilde{V}}_L(\zeta, t) \triangleq \bigcap_{\xi\in\partial V_L(\zeta,t)} \xi^T \begin{bmatrix} K[h](\zeta, t) \\ 1 \end{bmatrix}$, where $\partial V_L(\zeta, t)$ denotes Clarke's generalized gradient of $V_L(\zeta, t)$ [19, eq. 13]. Since $V_L(\zeta, t)$ is continuously differentiable in ζ, then $\partial V_L(\zeta, t) = \{\nabla V_L(\zeta, t)\}$, where ∇ denotes the gradient operator. Additionally, the time derivative of V_L exists almost everywhere (a.e.), i.e., $\dot{V}_L(\zeta, t) \overset{\text{a.e.}}{\in} \dot{\tilde{V}}_L(\zeta, t)$ for almost all $t \in \mathbb{R}_{\geq 0}$.

Taking the generalized time derivative of (15), using the trace operator property,[3] and substituting the closed-loop error system in (11), the output-layer adaptive update law in (10), and the inner-layer adaptive update laws in (12) yields

$$\dot{\tilde{V}}_L \subseteq e^T\Big(W^{*T}\sigma^*\big(\Phi^*(x)\big) + \varepsilon(x) - k_1 e - k_s K\big[\operatorname{sgn}(e)\big] - \hat{W}^T K\Big[\hat{\sigma}\Big(\hat{\Phi}(x)\Big)\Big] - \tilde{W}^T K\Big[\hat{\sigma}\Big(\hat{\Phi}(x)\Big)\Big]\Big) - \sum_{j=0}^{k} \operatorname{tr}\left(K\left[\tilde{V}_j^T p_j(t)\nu_j(t)\mathbf{1}_{\left\{\underline{\hat{V}_j} \leq \left\|\hat{V}_j\right\|_F \leq \overline{\hat{V}_j}\right\}}\right]\right). \tag{16}$$

Adding and subtracting $e^T(W^{*T}K[\hat{\sigma}(\hat{\Phi}(x))])$ in (16) yields

$$\dot{\tilde{V}}_L \subseteq e^T W^{*T}\sigma^*\big(\Phi^*(x)\big) + e^T\varepsilon(x) - k_1 e^T e$$

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

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

$$-k_s e^T K\big[\operatorname{sgn}(e)\big] - e^T W^{*T} K\Big[\hat{\sigma}\Big(\hat{\Phi}(x)\Big)\Big] - \sum_{j=0}^{k} \operatorname{tr}\Bigg(K\Big[\tilde{V}_j^T p_j(t) v_j(t) \mathbf{1}_{\left\{\underline{\hat{V}_j} \le \left\|\hat{V}_j\right\|_F \le \overline{\hat{V}_j}\right\}}\Big]\Bigg). \quad (17)$$

By the definition of the calculus $K[\,\cdot\,]$, $e^T K[\operatorname{sgn}(e)] = \|e\|$. Using (13), (17) can be upper bounded as

$$\dot{V}_L \overset{\text{a.e.}}{\le} -\|e\|\Big(k_s - \overline{W^*}\Big(\overline{\sigma^*} + \overline{\hat{\sigma}}\Big) - \overline{\varepsilon} - 2(k+1)\overline{V^*}\rho(\|e\|)\Big) - k_1\|e\|^2. \quad (18)$$

To ensure $k_s > \overline{W^*}(\overline{\sigma^*} + \overline{\hat{\sigma}}) + \overline{\varepsilon} + 2(k+1)\overline{V^*}\rho(\|e\|)$, it is required that $\|e\| < \rho^{-1}(\frac{k_s - \overline{W^*}(\overline{\sigma^*} + \overline{\hat{\sigma}}) - \overline{\varepsilon}}{2(k+1)\overline{V^*}})$, which implies $\|z\| < \rho^{-1}(\frac{k_s - \overline{W^*}(\overline{\sigma^*} + \overline{\hat{\sigma}}) - \overline{\varepsilon}}{2(k+1)\overline{V^*}})$. The inequality in (18) can be upper bounded as

$$\dot{V}_L \overset{\text{a.e.}}{\le} -k_1\|e\|^2, \quad \forall z \in \mathcal{D}, \quad (19)$$

where $\mathcal{D} \triangleq \{z \in \mathbb{R}^{\Psi} : \|z\| < \rho^{-1}(\frac{k_s - \overline{W^*}(\overline{\sigma^*} + \overline{\hat{\sigma}}) - \overline{\varepsilon}}{2(k+1)\overline{V^*}})\}$. Then using (15) and (17), V_L is positive definite and non-increasing, which implies $\|z(0)\| \le \sqrt{\frac{V_L(0)}{\underline{\alpha}}}$. Therefore, it is sufficient to show $\|z(0)\| < \sqrt{\frac{\underline{\alpha}}{\overline{\alpha}}}\rho^{-1}(\frac{k_s - \overline{W^*}(\overline{\sigma^*} + \overline{\hat{\sigma}}) - \overline{\varepsilon}}{2(k+1)\overline{V^*}})$, which implies $\mathcal{S} \triangleq \{z \in \mathcal{D} : \sqrt{\frac{\underline{\alpha}}{\overline{\alpha}}}\rho^{-1}(\frac{k_s - \overline{W^*}(\overline{\sigma^*} + \overline{\hat{\sigma}}) - \overline{\varepsilon}}{2(k+1)\overline{V^*}})\}$ is the region where (19) holds, and yields the sufficient gain condition in (14).

From (15) and (19), $V_L \in \mathcal{L}_\infty$, which implies $z \in \mathcal{L}_\infty$, and hence, $e, \tilde{W} \in \mathcal{L}_\infty$. Using (2) and (7) implies $x \in \mathcal{L}_\infty$ and $\hat{W} \in \mathcal{L}_\infty$, respectively. Using (12) and (13), $e \in \mathcal{L}_\infty$ implies $v_j \in \mathcal{L}_\infty$, and by the use of the indicator function $\mathbf{1}_{\{\underline{\hat{V}_j} \le \|\hat{V}_j\|_F \le \overline{\hat{V}_j}\}}$, implies $\dot{\hat{V}}_j \in \mathcal{L}_\infty$ for all $j \in \{0, \ldots, k\}$. Using (6), the fact that $x, \hat{V}_j \in \mathcal{L}_\infty$ implies $\hat{\Phi} \in \mathcal{L}_\infty$. By design, $\dot{x}_d, \hat{\sigma} \in \mathcal{L}_\infty$. Using (9), the fact that $x, e, \dot{x}_d, \hat{W}, \hat{\sigma} \in \mathcal{L}_\infty$ implies $u \in \mathcal{L}_\infty$. Using (10), the fact that $\hat{\sigma}, e \in \mathcal{L}_\infty$ implies $\dot{\hat{W}} \in \mathcal{L}_\infty$. By the LaSalle-Yoshizawa theorem extension for nonsmooth systems in [16] and [20], $k_1\|e\|^2 \to 0$, which implies $\|e(t)\| \to 0$ as $t \to \infty$. ■

V. Simulation

To demonstrate the performance of the developed method, simulations are performed on the nonlinear system from [21], where $f(x) = [-x_1 + x_2, -\frac{1}{2}x_1 + \frac{1}{2}x_2(1 - (\cos(2x_1 + 2)^2))]^T$ and $g(x) = \operatorname{diag}[5, 3]$ are used to model the drift dynamics and control effectiveness in (1). The desired trajectory is $x_d = [3\cos(t), 5\sin(t)]^T$. The initial condition is $x(0) = [3, 0]^T$. The controller gains are selected as $k_s = 0.5$ and $k = 7$.

To illustrate the modularity of the architecture, two simulation studies are conducted for 60 seconds, each with different inner-layer adaptation laws and structures implemented. During the entire 60 seconds, the output-layer update law in (10) is active in both studies. Due to the large number of weights in this system, only one inner-layer weight update law is active at a time.[4] However, the selection of switching signals that dictates when to update each inner-layer DNN may be arbitrarily selected. Computational resources may be allocated to update a subset of the inner-layer weights, or all inner-layers may be updated arbitrarily at a time. Additionally, inner-layer weights may dropout, or be selectively turned off to prevent over-fitting [13].

To reduce the computational load and show the flexibility in selection of the switching signals to update each inner-layer DNN weight estimate, in both simulation studies, the switching signal is arbitrarily designed as

$$p_j(t) = \begin{cases} 1, & t \in \big[10j, 10(j+1)\big], \\ 0, & \text{else}, \end{cases} \quad (20)$$

for all $j \in \{0, \ldots, 5\}$. Based on the switching signals in (20), each inner-layer weight update law is active for 10 seconds during the duration of the simulation and is activated in consecutive order, i.e., $\hat{V}_0$ is active from 0 to 10 seconds, $\hat{V}_1$ is active from 10 to 20 seconds, etc. If the update law is not active (i.e., $p_j(t) = 0$), then its associated weights are not updated.

The DNN is composed of 6 layers and the hyperbolic tangent function is the activation function for each neuron. Layers 1-6 have 12, 10, 15, 15, 12, and 20 neurons, respectively. The outer-layer weight learning parameter is set to $\Gamma_W = 10 \cdot \mathbf{1}_{L\times L}$, where $\mathbf{1}_{n\times m}$ is an $n \times m$ matrix of ones. The bounds on the inner-layer weights are $\underline{\hat{V}_j} = 10^{-6}$ and $\overline{\hat{V}_j} = 250$ for all $j \in \{0, 1, \ldots, 5\}$. The initial conditions for the output-layer weight estimate $\hat{W}$ and the inner-layer weight estimates $\hat{V}_j$ are randomly selected from a uniform distribution from $[-0.5, 0.5]$.

Various update laws for $\dot{\hat{V}}_{0-5}$ can be selected and designed for learning the inner-layer DNN weights while guaranteeing tracking performance. The constraints in (13) provide general guidelines and enable the user to select or design update laws accordingly, such as gradient tuning laws based on back-propagated errors [22] or a Hebbian tuning law [23].

In the first simulation study (Study 1), the inner-layer weight update laws, which are heuristically selected and inspired by the methods in [24] and [25], are selected as

$$v_j = \Gamma_{V_j} e \cdot \operatorname{re}\Big(\Big(\tanh \circ \hat{\phi}_j^{-1} \circ \hat{V}_j^{+T}\hat{\phi}_{j+1}^{-1} \circ \ldots \cdots \circ \hat{V}_5^{+T}\hat{\sigma}^{-1}\Big)\Big(\hat{W}^{+T}\dot{\bar{x}}\Big)\Big)\hat{V}_j \quad (21)$$

for all $j \in \{0, 1, \ldots, 5\}$, where the inner-layer weight learning parameters are set to $\Gamma_{V_j} = 1000 \cdot \mathbf{1}_{L_j\times 2}$, $\hat{V}_j^+$ is the right-pseudo inverse of $\hat{V}_j$, the $\operatorname{re}(\cdot)$ operator outputs element-wise the real component of each entry in $\hat{V}_j$, and $\dot{\bar{x}} \in \mathbb{R}^n$ denotes the numerically generated state derivative. Due to the projection bounds $\underline{\hat{V}_j}$ and $\overline{\hat{V}_j}$, $\hat{V}_j^+$ exists and is bounded. The selected update law satisfies the modular adaptive control constraint in (13).

[4]There are 955 individual weights in the simulation. As guaranteed by the analysis, the developed method can update each layer separately, i.e., a subset of the total number of weights are updated at a given time. It may be computationally burdensome for some systems to update a large number of weights online. One purpose of this simulation is to highlight the developed method's ability update different DNN layers separately.

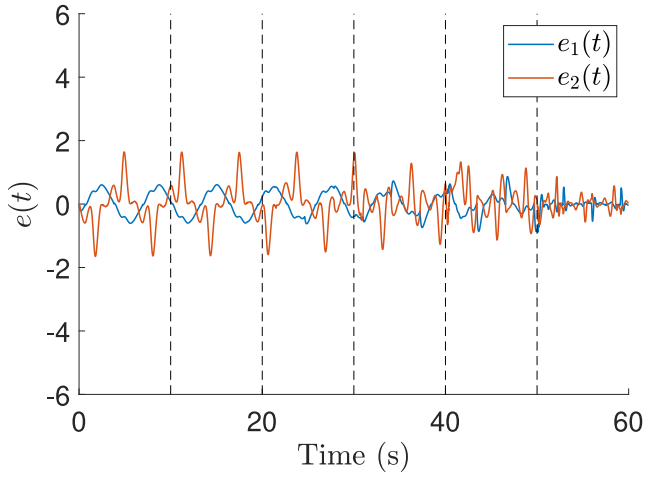

Fig. 1. Tracking error e in the system with the inner-layer DNN weight update law in (21). The vertical lines denote when a new inner-layer update law is activated.

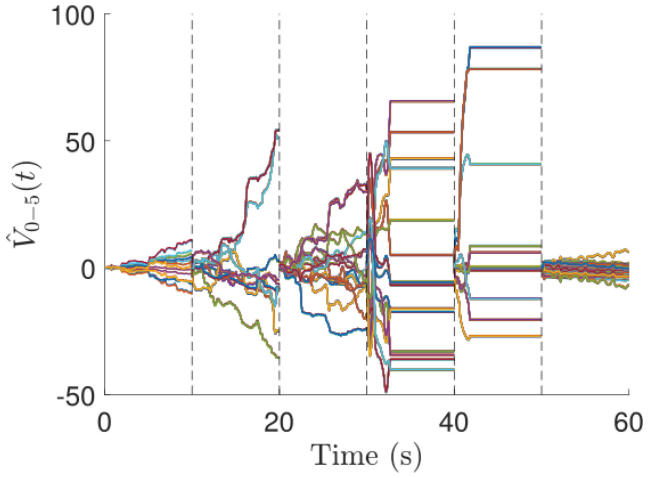

Fig. 2. Value of the active DNN inner-layer weight estimates $\hat{V}_{0-5}$ when each inner-layer update law is active with the inner-layer DNN weight update law in (21). The vertical lines denote when a new inner-layer update law is activated. Based on the switching signals in (20), $\hat{V}_0$ is active from 0 to 10 seconds, $\hat{V}_1$ is active from 10 to 20 seconds, etc.

Figure 1 shows the tracking error $e \triangleq [e_1, e_2]^T$ with the inner-layer DNN weight update law in (21). From initialization to approximately 30 seconds, the system exhibits poor tracking performance with oscillatory behavior. Poor controller performance is likely due to randomly initialized weights which signifies no *a priori* model knowledge in the drift dynamic approximation. However, as each subsequent inner-layer weight matrix $\hat{V}_{0-5}$ is updated, the tracking performance improves and the amplitude of the error signals decreases, which indicates the DNNs improved approximation of the drift dynamics.

Figure 2 shows the online adjustment of the inner-layer weights with the inner-layer DNN weight update law in (21). Each set of weights is divided by a set of black dashed lines. Note that some of the weights are unaffected by the indicator function since the upper bounds $\overline{\hat{V}_j}$ are sufficiently large. However, the weights of layers 3 and 4 are affected by the indicator function. When layers 3 and 4 are activated, their weight estimates quickly reach the bounds defined by the indicator function.

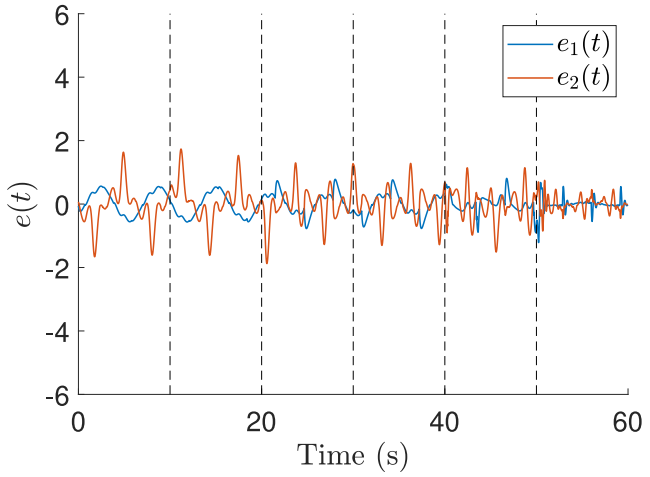

Fig. 3. Tracking error e in the system with inner-layer DNN weight update law in (22). The vertical lines denote when a new inner-layer update law is activated.

To provide a comparison in the selection of inner-layer weight update laws, a second simulation study (Study 2) is performed where the DNN is structured similarly as the first simulation study, and each inner-layer update is activated according to the switching signal in (20). To illustrate the modularity in the selection of inner-layer weight update laws, various inner-layer weight update laws are arbitrarily selected to satisfy (13) and are selected as

$$\begin{aligned}
\nu_0 &= \Gamma_{V_0} \mathrm{e}^{-\frac{e_1^2}{2}} \mathrm{e}^{-\frac{e_2^2}{2}} \|e\|, \\
\nu_1 &= \Gamma_{V_1} \mathrm{e}^{-\frac{e_1^2}{2}} \tanh(e_2) \|e\|, \\
\nu_2 &= \Gamma_{V_2} \tanh(e_1) \tanh(e_2) \|e\|, \\
\nu_3 &= \Gamma_{V_3} \frac{1}{1+\mathrm{e}^{-e_1}} \mathrm{e}^{-\frac{e_2^2}{2}} \|e\|, \\
\nu_4 &= \Gamma_{V_4} \tanh(e_1) \frac{1}{1+\mathrm{e}^{-e_2}} \|e\|, \\
\nu_5 &= \Gamma_{V_5} \frac{1}{1+\mathrm{e}^{-e_1}} \tanh(e_2) \|e\|,
\end{aligned} \tag{22}$$

where the inner-layer weight learning parameters are set to $\Gamma_{V_j} = 1000 \cdot \mathbf{1}_{L_j \times 2}$ for all $j \in \{0, 1, \ldots, 5\}$.

Figure 3 shows the tracking error e with the inner-layer DNN weight update law in (22). Similar to Study 1, the tracking error results in Figure 3 show poor tracking performance at the start of the simulation. However, as the simulation progresses and each subsequent inner-layer is activated, the tracking performance improves.

Table I shows the root mean squared (RMS) error and standard deviation (SD) of the tracking error. The leftmost column indicates the active inner-layer weight law, e.g., the first row is data collected while the $\hat{V}_0$ update law is active. Study 1 uses the heuristically selected update law in (21), and the RMS error and SD corresponding to $\hat{V}_0$ are 0.735 and 0.145, respectively. Study 2 uses the update law in (22),

TABLE I
RMS AND SD ERROR

Active $\hat{V}_j$	Study 1		Study 2	
	RMS Error	SD Error	RMS Error	SD Error
$\hat{V}_0$	0.735	0.145	0.705	0.145
$\hat{V}_1$	0.727	0.142	0.700	0.149
$\hat{V}_2$	0.710	0.136	0.651	0.137
$\hat{V}_3$	0.640	0.116	0.612	0.098
$\hat{V}_4$	0.658	0.087	0.591	0.094
$\hat{V}_5$	0.289	0.039	0.327	0.050

and the RMS error and SD are 0.705 and 0.145, respectively. In Study 1, as each subsequent inner-layer update law is activated, the tracking performance improves and results in an RMS error and SD of 0.289 and 0.039, respectively. Although a similar trend is seen in Study 2, the inner-layer update law in Study 1 outperforms Study 2 which indicates the inner-layer update law yielded better function approximation performance, as expected. However, the tracking objective is achieved in both studies, despite different inner-layer update laws, as guaranteed by the analysis.

VI. CONCLUSION

This letter develops DNN-based modular adaptive control update laws and constraints, which were inspired by existing modular adaptive control constraints, to achieve trajectory tracking objectives. The modular adaptive control framework provides general constraints and enables users to design update laws accordingly. The developed method also allows for different sets of weights to be arbitrarily switched. A simulation study was performed to compare the performance of different inner-layer weight update laws selected. Inner-layer weight update laws were designed and selected that satisfy the developed design constraints. Despite different inner-layer update laws used and arbitrary switching, the implemented controllers achieved the tracking objective.

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 sponsoring agencies.

REFERENCES

[1] F. L. Lewis, S. Jagannathan, and A. Yesildirak, *Neural Network Control of Robot Manipulators and Nonlinear Systems*. Philadelphia, PA, USA: CRC Press, 1998.
[2] N. E. Cotter, "The Stone-Weierstrass theorem and its application to neural networks," *IEEE Trans. Neural Netw.*, vol. 1, no. 4, pp. 290–295, Dec. 1990.
[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[4] G. Joshi and G. Chowdhary, "Deep model reference adaptive control," in *Proc. IEEE Conf. Decis. Control*, 2019, pp. 4601–4608.
[5] 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.
[6] 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.*, early access, Jan. 28, 2021, doi: 10.1109/LCSYS.2021.3055454.
[7] G. Joshi, J. Virdi, and G. Chowdhary, "Asynchronous deep model reference adaptive control," 2020. [Online]. Available: arXiv:2011.02920.
[8] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.
[9] P. Patre, W. Mackunis, K. Dupree, and W. E. Dixon, "Modular adaptive control of uncertain Euler-Lagrange systems with additive disturbances," *IEEE Trans. Autom. Control*, vol. 56, no. 1, pp. 155–160, Jan. 2011.
[10] M. S. De Queiroz, D. M. Dawson, and M. Agarwal, "Adaptive control of robot manipulators with controller/update law modularity," *Automatica*, vol. 35, no. 8, pp. 1379–1390, 1999.
[11] W. E. Dixon, M. S. de Queiroz, D. M. Dawson, and T. J. Flynn, "Adaptive tracking and regulation control of a wheeled mobile robot with controller/update law modularity," *IEEE Trans. Control Syst. Technol.*, vol. 12, no. 1, pp. 138–147, Jan. 2004.
[12] M. Krstic and P. V. Kokotovic, "Adaptive nonlinear design with controller-identifier separation and swapping," *IEEE Trans. Autom. Control*, vol. 40, no. 3, pp. 426–440, Mar. 1995.
[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[14] H. D. Patino, R. Carelli, and B. R. Kuchen, "Neural networks for advanced control of robot manipulators," *IEEE Trans. Neural Netw.*, vol. 13, no. 2, pp. 343–354, Mar. 2002.
[15] M. Chen, S. S. Ge, and B. V. E. How, "Robust adaptive neural network conrtol for a class of uncertain MIMO nonlinear systems with input nonlinearities," *IEEE Trans. Neural Netw.*, vol. 21, no. 5, pp. 796–812, May 2010.
[16] 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.
[17] P. M. Patre, K. Dupree, W. MacKunis, and W. E. Dixon, "A new class of modular adaptive controllers, part II: Neural network extension for non-LP systems," in *Proc. Amer. Control Conf.*, Seattle, WA, USA, Jun. 2008, pp. 1214–1219.
[18] 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.
[19] D. Shevitz and B. Paden, "Lyapunov stability theory of nonsmooth systems," *IEEE Trans. Autom. Control*, vol. 39 no. 9, pp. 1910–1914, Sep. 1994.
[20] 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.
[21] R. Kamalapurkar, J. Rosenfeld, and W. E. Dixon, "Efficient model-based reinforcement learning for approximate online optimal control," *Automatica*, vol. 74, pp. 247–258, Dec. 2016.
[22] P. J. Werbos, "Back propagation: Past and future," in *Proc. Int. Conf. Neural Netw.*, vol. 1, 1989, pp. 1343–1353.
[23] D. O. Hebb, *The Organization of Behavior*. New York, NY, USA: Wiley, 1949.
[24] K.-A. Toh, "Analytic network learning," 2018. [Online]. Available: arXiv:1811.08227.
[25] H.-T. Nguyen, C. C. Cheah, and K.-A. Toh, "An analytic layer-wise deep learning framework with applications to robotics," 2021. [Online]. Available: arXiv:2102.03705.