Automatica 189 (2026) 113037

Contents lists available at ScienceDirect

Automatica

journal homepage: www.elsevier.com/locate/automatica

Lyapunov-based dropout deep neural network (Lb-DDNN) adaptive controller☆

Saiedeh Akbari *, Emily J. Griffis, Omkar Sudhir Patil, Warren E. Dixon

Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville FL 32611-6250, USA

ARTICLE INFO

Article history:
Received 4 July 2024
Received in revised form 20 May 2025
Accepted 14 March 2026
Available online 2 May 2026

Keywords:
Deep neural networks
Dropout
Adaptive control
Lyapunov methods
Nonlinear control systems

ABSTRACT

Deep neural networks (DNNs) are well-established tools for function approximation. However, DNNs are known to suffer from issues such as overfitting and co-adaptation, which affect the generalization and overall performance of the DNN. One powerful regularization technique to mitigate the overfitting and co-adaptation issues is dropout. In the dropout technique, randomly selected neurons are ignored or "dropped out", meaning that their contributions to the downstream neurons are temporarily removed, which helps mitigate overfitting and co-adaptation of those neurons. In this paper, a Lyapunov-based dropout DNN (Lb-DDNN) regularization method is developed to deactivate the weights associated with stochastically selected neurons at each layer of the DNN. The Lb-DDNN is modeled using a switched mechanism involving an ensemble of DNN-based controllers that each approximate the unknown nonlinear dynamics individually. Switching between each individual DNN is conducted through a general state- and time-dependent switching signal. Simultaneously, an analytical Lyapunov stability-driven weight adaptation law is developed that adapts the weights of the DDNN online. Asymptotic convergence of the tracking error is ensured through a Lyapunov-based switched stability analysis. An extensive ablation study is provided based on simulations of a five-dimensional nonlinear system to investigate the effects of the various dropout training parameters on the performance of the Lb-DDNN controller. Simulation results indicate a 34.56% improvement in the tracking error, a 43.69% improvement in the function approximation error, and 33.88% lower control effort when compared to a baseline Lyapunov-based DNN (Lb-DNN) adaptive controller without dropout regularization.

© 2026 Elsevier Ltd. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

1. Introduction

Deep neural networks (DNNs) serve as universal function approximators (Anderson, 1995), and they have become a popular means for identification and control of nonlinear systems with unknown dynamics with unstructured uncertainties (Abbeel et al., 2010; Bansal et al., 2016; Li et al., 2017; OConnell et al., 2022; Punjani & Abbeel, 2015; Shi et al., 2019; Zhou et al., 2017). Typically, DNN models in DNN-based control methods are trained offline using numerical optimization methods applied to prior collected datasets to identify the unknown nonlinearities in the model dynamics (Brunton & Kutz, 2019, Section 6.6). The trained DNN is then incorporated as a feedforward term with static weight estimates in the controller to compensate for unstructured uncertainties in the dynamics. The DNN weights are not updated online regardless of system performance and lacks robustness to changes in the dynamics. In contrast, Lyapunov-based DNN control methods employ a weight update law derived through a Lyapunov-based analysis (Griffis et al., 2023; Hart et al., 2023; Joshi & Chowdhary, 2018, 2019; Le et al., 2022; O. Patil et al., 2022; O.S. Patil et al., 2022). Developments in Joshi and Chowdhary (2018, 2019) involve real-time updates of the outer-layer weights while training the inner-layer weights offline using pre-existing data. More recent developments in Griffis et al. (2023), Hart et al. (2023), O.S. Patil et al. (2022) and O. Patil et al. (2022) use Lyapunov-based methods to develop adaptation laws for all layer weights of the DNN (i.e., Lyapunov-based deep neural networks (Lb-DNNs)). In O. Patil et al. (2022), an Lb-DNN is compared to an offline DNN and the Lyapunov-based iterative adjustment DNN method of Sun et al. (2022), highlighting improved performance in both tracking and function approximation. Furthermore, the result in Lamb et al. (2023) shows that the method developed in Sun et al. (2022) outperforms traditional controllers

☆ This research is supported in part by AFRL, United States project FA8651-24-1-0018, AFOSR, United States grant FA9550-19-1-0169, and Office of Naval Research, United States grant N00014-21-1-2481. 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 the sponsoring agencies. The material in this paper was not presented at any conference . This paper was recommended for publication in revised form by Associate Editor Raul Ordonez under the direction of Editor Andrea Serrani.

* Corresponding author.
E-mail addresses: akbaris@ufl.edu (S. Akbari), emilygriffis00@ufl.edu (E.J. Griffis), patilomkarsudhir@ufl.edu (O.S. Patil), wdixon@ufl.edu (W.E. Dixon).

https://doi.org/10.1016/j.automatica.2026.113037
0005-1098/© 2026 Elsevier Ltd. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

such as PID, model reference adaptive control (MRAC), and Deep MRAC (DMRAC) in terms of trajectory tracking capabilities.

Both offline and online training can lead to degraded performance due to data overfitting in fully-connected DNNs. Another challenge that decreases generalization and performance of the trained DNN is co-adaptation, where multiple neurons, or even entire layers, become overly reliant on each other during the training process (Charu, 2018). One effective approach to address these issues is through dropout regularization. Dropout was originally introduced by G. Hinton in Hinton et al. (2012) to prevent co-adaptation of feature detectors and improve the generalization performance of DNNs. Dropout regularization involves stochastically dropping out neurons during training, which helps prevent overfitting, enhances the overall function approximation performance, and efficiently allocates computational resources while updating the network's weights (Le et al., 2022) and Srivastava et al. (2014). By setting the activation of certain individual weights to zero, dropout induces sparse representation in the network which reduces co-dependency in neurons. Dropouts can be viewed as training an ensemble of multiple DNNs with smaller width that are trained independently. Independence in the training has a regularizing effect and provides better generalization to new information. This intuitive reasoning is also applicable for using dropout in DNN-based adaptive control, since dropouts mitigate co-adaptation by reducing the number of weights influencing an adaptation law. In Wei et al. (2020), the dropout is shown to have an implicit regularizing effect in the sense that the randomness of dropouts essentially introduces a small exploratory perturbation in the training process.

Although dropout regularization has been used for offline training of DNNs in results such as Ba and Frey (2013) and Dahl et al. (2013), its application has been limited in real-time adaptive control settings. The Lb-DNN control architecture developed in O. Patil et al. (2022) and O.S. Patil et al. (2022) improved the tracking and function approximation performance of the controller compared to its predecessor in Sun et al. (2022). The result in O.S. Patil et al. (2022) uses a residual neural network (ResNet), which is different from the DNN architecture because it involves shortcut connections. These connections address the vanishing gradient problem by allowing gradients to bypass some layers during backpropagation. Instead of learning a direct mapping, the layers learn a residual mapping, and the shortcut adds the input to this output, ensuring that even if gradients through the learned layers are small, a signal can still reach earlier layers. Although ResNet architectures address the vanishing gradient issue, similar to DNNs, ResNets are still subject to overfitting and co-adaptation. This paper aims to advance the dropout technique for Lb-DNNs, initially introduced in O.S. Patil et al. (2022), by using time- and state-dependent randomization matrices to address these challenges. However, applying dropout regularization on real-time adaptive control methods are accompanied with hurdles such as problem formulation and design in such a way that enables use of the universal function approximation property in a tractable form for dropouts and designing an appropriate dropout strategy that is amenable for the adaptation law design and Lyapunov-based stability analysis. The idea of using dropouts in Lb-DNNs introduces various open questions pertaining to a real-time learning setting such as when or how often to perform dropouts, when to deactivate them, how many neurons to dropout, etc.

This paper introduces a dropout training algorithm for a DNN-based adaptive controller that updates analytical DNN weights in real-time using a constructed Lyapunov-based adaptation law (i.e., a Lyapunov-based Dropout Deep Neural Network (Lb-DDNN) architecture). The proposed technique involves the selective inactivation, or dropout, of weights associated with randomly selected neurons within each DNN layer. To incorporate dropout into the adaptive DNN architecture and facilitate the design of the stability-driven weight adaptation laws, a recursive representation of the Lb-DDNN model is developed using randomization matrices. The designed Lb-DDNN architecture is modeled as a collection of smaller DNNs that switches between sets of the selected nodes. The switching between smaller DNNs is incorporated into the stability analysis and stability-driven weight adaptation laws through a switched systems approach. Through a non-smooth Lyapunov-based stability analysis, the designed controller and weight adaptation law are guaranteed to ensure asymptotic tracking error convergence to zero.

To address the aforementioned open questions on implementing dropouts in practice, an extensive ablation study is provided based on simulations on a five-dimensional nonlinear system to investigate the effects of the various dropout training parameters on the performance of the Lb-DDNN controller. Specifically, the simulation experiments study the influence of the deactivation time of dropout, the frequency of changing dropout configuration, and the number of neurons dropping out. Additional simulations are performed to compare the Lb-DDNN adaptive controller with the baseline adaptive DNN controller developed in O.S. Patil et al. (2022) and to compare the function approximation performance of these two controllers on an off-trajectory dataset. Moreover, the performance of the Lb-DDNN adaptive controller with a narrow NN adaptive controller and sparse Lb-DNN architecture are studied. Additional simulations are included to compare the performance of the developed Lb-DDNN with a pruning method. Although pruning methods have been extensively investigated for offline learning problems, these methods do not apply for online learning. Therefore, an online pruning algorithm is introduced for comparison to the developed Lb-DDNN. The simulation results show a 34.56% improvement in the tracking error, a 43.69% improvement in the function approximation error, and 33.88% lower control effort in the proposed controller when compared to the baseline controller in O.S. Patil et al. (2022). The effects of the rest of the ablation study parameters are provided in Section 5.2.

2. Problem formulation

2.1. Notation

Given the sets $\mathtt{A}$, $\mathtt{B}$, $\mathtt{C}$ and functions $\mathtt{f}_1 : \mathtt{A} \to \mathtt{B}$ and $\mathtt{f}_2 : \mathtt{B} \to \mathtt{C}$, $\mathtt{f}_2 \circ \mathtt{f}_1$ denotes the composition of the functions $\mathtt{f}_1$ and $\mathtt{f}_2$. The composition of $\mathtt{f}_1$ and $\mathtt{f}_2$ is a new function, $\mathtt{f} : \mathtt{A} \to \mathtt{C}$, defined as $\mathtt{f} \triangleq \mathtt{f}_2 \circ \mathtt{f}_1 = \mathtt{f}_2(\mathtt{f}_1(\mathtt{a}))$, for all $\mathtt{a} \in \mathtt{A}$. A zero matrix with dimension of $p \times q$ is denoted as $\mathbf{0}_{p\times q}$. An identity matrix with the dimension of p is represented as I_p. Let $A^+ \in \mathbb{R}^{n\times m}$ denote the right pseudo inverse of the matrix A, where $A^+(\cdot) \triangleq A^\top(\cdot)\left(A(\cdot)A^\top(\cdot)\right)^{-1}$. For $x \in \mathbb{R}$, let the element-wise sign function be defined as $\operatorname{sgn}(x) \triangleq 1$ for $x > 0$, $\operatorname{sgn}(x) \triangleq 0$ for $x = 0$, and $\operatorname{sgn}(x) \triangleq -1$ for $x < 0$. For functions f_1 and f_2, $f_1(x) = \mathcal{O}^k(f_2(x))$ means that there exists some constants $K \in \mathbb{R}_{>0}$ and $x_0 \in \mathbb{R}$ such that $\|f_1(x)\| \le K\|f_2(x)\|^k$ for all $x \ge x_0$. The right-to-left matrix product operator is represented by $\overleftarrow{\prod}$, i.e., $\overleftarrow{\prod_{p=1}^{m}} A_p = A_m \ldots A_2 A_1$, and $\overleftarrow{\prod_{p=a}^{m}} A_p = 1$ if $a > m$. For a matrix $A \triangleq \left[a_{i,j}\right] \in \mathbb{R}^{n\times m}$, where $a_{i,j}$ is the element in the ith row and jth column of the matrix A, the vectorization operator is defined as $\operatorname{vec}(A) \triangleq \left[a_{1,1}, \ldots, a_{n,1}, \ldots, a_{1,m}, \ldots, a_{n,m}\right]^\top \in \mathbb{R}^{nm}$. The Kronecker product for matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$ is represented as $A \otimes B$. From Bernstein (2009, Proposition 7.1.9) and given matrices $A \in \mathbb{R}^{p\times a}$, $B \in \mathbb{R}^{a\times r}$, and $C \in \mathbb{R}^{r\times s}$, the vectorization operator satisfies the property $\operatorname{vec}(ABC) = \left(C^\top \otimes A\right)\operatorname{vec}(B)$. Differentiating $\operatorname{vec}(ABC)$ on both sides with respect to $\operatorname{vec}(B)$ yields the property $\frac{\partial}{\partial \operatorname{vec}(B)}\operatorname{vec}(ABC) = C^\top \otimes A$. Given some sets A and B, a set-valued map F from A to subsets of B is denoted by

$F : A \rightrightarrows B$. The notation $\overline{\text{co}}A$ denotes the closed convex hull of the set A. The notation $\mathbb{B}(x, \delta)$, for $x \in \mathbb{R}^n$ and $\delta > 0$, is used to denote the set $\{y \in \mathbb{R}^n : \|x - y\| < \delta\}$. Consider a Lebesgue measurable and locally essentially bounded function $h : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$. The Filippov regularization of h at $(y, t) \in \mathbb{R}^n \times \mathbb{R}_{\geq 0}$ is defined as the intersection of convex closures of values attained by h in every neighborhood of y omitting sets of measure zero, i.e.,

$$K[h](y, t) \triangleq \bigcap_{\delta > 0} \bigcap_{\text{m}\mathbb{S}_m = 0} \overline{\text{co}}\, h(\mathbb{B}(y, \delta) \setminus \mathbb{S}, t),$$

where $\bigcap_{\text{m}\mathbb{S}=0}$ denotes the intersection over all sets $\mathbb{S} \subset \mathbb{R}^n$ of Lebesgue measure zero (Paden & Sastry, 1987, Eq. 1.2b). The notation $(\cdot) \overset{\text{a.a.t.}}{}$ means that the relation $(\cdot)$ holds for almost all time. The space of essentially bounded Lebesgue measurable functions is represented as $\mathcal{L}_\infty$. Consider a Lebesgue measurable and locally essentially bounded function $f : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$. The function $x : \mathcal{I} \to \mathbb{R}^n$ is called a Filippov solution of $\dot{x} = f(x, t)$ on the interval $\mathcal{I} \subseteq \mathbb{R}_{\geq 0}$ if x is absolutely continuous on $\mathcal{I}$ and $\dot{x} \overset{\text{a.a.t.}}{\in} K[f](x, t)$.

Definition 1. (Generalized Time Derivative (Kamalapurkar et al., 2019, Def. 3)) Let $F : \mathbb{R}^n \times \mathbb{R}_{\geq t_0} \rightrightarrows \mathbb{R}^n$ have nonempty and compact values. The generalized time derivative of a Locally Lipschitz-continuous function $\dot{V}_F : \mathbb{R}^n \times \mathbb{R}_{\geq t_0} \to \mathbb{R}$ defined as

$$\dot{V}_F(x, t) \triangleq \max_{p \in \partial V(x,t)} \max_{q \in F(x,t)} p^\top [q; 1],$$

where ∂V denotes the Clarke gradient of V.

2.2. Dynamic model and control objective

Consider a control-affine nonlinear dynamic system modeled as[1]

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

where $t \in \mathbb{R}_{\geq 0}$ denotes the time, and the state, the unknown locally Lipschitz drift vector field, the known full row rank continuous control effectiveness matrix,[2] and the control input are denoted as $x : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}^n$, $g : \mathbb{R}^n \to \mathbb{R}^{n \times m}$, and $u : \mathbb{R}_{\geq 0} \to \mathbb{R}^m$, respectively. A trajectory tracking control objective is considered where the goal is for the state to converge to a continuously differentiable desired trajectory $x_d : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$. To fulfill the tracking objective, an adaptive Lb-DNN architecture is designed to estimate the unknown drift vector field f. The adaptive Lb-DNN controller and adaptation law are subsequently designed based on the Lyapunov-based stability analysis to ensure asymptotic tracking error convergence. The control objective is quantified by the tracking error $e : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ defined as

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

The following assumption facilitates the subsequent analysis.

Assumption 1. The desired trajectory is designed to be sufficiently smooth such that it can be bounded as $\|x_d(t)\| \leq \overline{x_d}$ and $\dot{x}_d \in \mathcal{L}_\infty$, for all $t \in \mathbb{R}_{\geq 0}$, where $\overline{x_d} \in \mathbb{R}_{>0}$ is a known constant.

[1] A first-order control-affine system is considered only for the sake of ease in illustration. The extension to higher-order systems can be easily performed by transforming the system to a first-order system using the auxiliary error formulation in Xian et al. (2004, Eq. (8)).

[2] For notational simplicity, the control effectiveness matrix is assumed to be known. The technique in Sun et al. (2022) can be used alongside the developed method to approximate an uncertain linearly parameterizable control effectiveness matrix online.

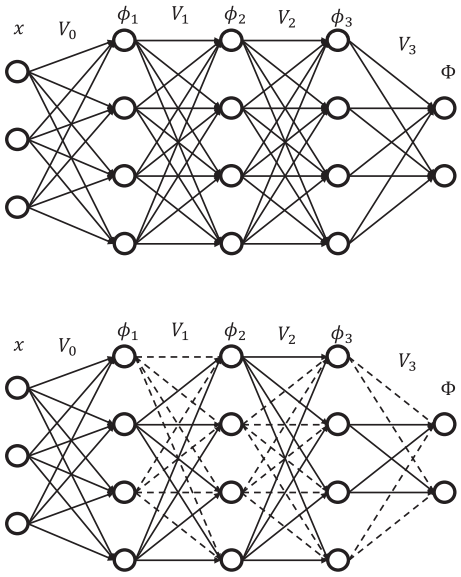


Fig. 1. (Top) The structure of a fully connected DNN with three hidden layers, and (Bottom) the structure of a Lb-DDNN where the dashed lines represent the randomly dropped out weights and the solid lines represent the weights that are selected for adaptation. The structure of Lb-DDNN depicts the dropped weights at a specific switching instant. The dropped neurons change with each switch, and thus, the structure of Lb-DDNN also changes with every switch.

3. Control design

3.1. Lyapunov-based dropout deep neural network architecture

In this section, a Lb-DDNN architecture is developed to compensate for the unknown drift vector field in (1). To design a dropout method for the DNN architecture, weights at each layer of the DNN are randomly selected and zeroed, and the remaining weights are updated according to the stability-driven adaptation law. Omitting randomly selected weights in training attenuate issues such as overfitting and co-adaptation because it mitigates the over-reliance on specific neurons and weights. The reduced overfitting and co-adaptation can yield improved performance and function approximation (Hinton et al., 2012; Srivastava et al., 2014). The developed Lb-DDNN is designed such that randomization matrices are used to incorporate dropout techniques into the online, stability-driven weight adaptation. Different randomization matrices are incorporated in each layer of the architecture, and they are designed such that at each layer, a batch of weights are randomly selected and dropped out, and the rest of the weights are updated based on a Lyapunov-based analysis.[3]

As shown in Fig. 1, let the compositional dropout DNN architecture $\Phi : \mathbb{R}^n \times \{0, 1\}^{\sum_{m=0}^{k} L_m \times \sum_{m=0}^{k} L_m} \times \mathbb{R}^{\sum_{j=0}^{k} L_j L_{j+1}} \to \mathbb{R}^{L_{k+1}}$, be defined as

$$\Phi\left(x, R_{\rho(x,t)}, \theta\right) = \left(R_{\rho(x,t),k} V_k\right)^\top \phi_k \circ \cdots \circ \left(R_{\rho(x,t),1} V_1\right)^\top \phi_1 \circ \left(R_{\rho(x,t),0} V_0\right)^\top x, \tag{3}$$

where $k \in \mathbb{Z}$ denotes the number of the layers in $\Phi\left(x, R_\rho, \theta\right)$, and $\phi_j : \mathbb{R}^{L_j} \to \mathbb{R}^{L_j}$ is the vector of smooth activation functions in the jth layer, for all $j \in \{1, \ldots, k\}$.[4] The number of nodes in

[3] The dropped out neurons can be user-selected instead of randomly selected.

[4] For DNNs that include different activation functions at each layer, ϕ_j can be modeled as $\phi_j \triangleq \left[\varsigma_{j,1}, \varsigma_{j,2}, \ldots, \varsigma_{j,L_j}\right]^\top$, where $\varsigma_{j,i} : \mathbb{R} \to \mathbb{R}$ denotes the activation function at the ith node of the jth layer, for all $j \in \{1, \ldots, k\}$ and $i \in \left\{1, \ldots, L_j\right\}$.

the jth layer of Φ is denoted by $L_j \in \mathbb{Z}_{>0}$, and $V_j \in \mathbb{R}^{L_j \times L_{j+1}}$ is the weight matrix of the jth layer of DDNN, for all $j \in \{0, \ldots, k\}$. The weights can be represented as a vector $\theta \in \mathbb{R}^{\sum_{j=0}^{k} L_j L_{j+1}}$ defined as $\theta \triangleq \left[\mathrm{vec}(V_0)^\top, \mathrm{vec}(V_1)^\top, \ldots, \mathrm{vec}(V_k)^\top\right]^\top$. Let $\rho : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathcal{I}$ represent a locally bounded time- and state-dependent switching signal, where $\mathcal{I} \triangleq \{1, \ldots, J\}$ is the set of all possible switching indices and J is the total number of switches. For each $i \in \mathcal{I}$, $R_i \in \{0,1\}^{\sum_{m=0}^{k} L_m \times \sum_{m=0}^{k} L_m}$ represents the ith randomization matrix defined as

$$R_i \triangleq \begin{bmatrix} R_{i,0} & \mathbf{0}_{L_0 \times L_1} & \cdots & \mathbf{0}_{L_0 \times L_k} \\ \mathbf{0}_{L_1 \times L_0} & R_{i,1} & \cdots & \mathbf{0}_{L_1 \times L_k} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{L_k \times L_0} & \mathbf{0}_{L_k \times L_1} & \cdots & R_{i,k} \end{bmatrix}, \quad \forall i \in \mathcal{I},$$

where $R_{i,j} \in \{0,1\}^{L_j \times L_j}$ is a diagonal matrix, for $i \in \mathcal{I}$ and $j \in \{1, \ldots, k\}$.

The randomization matrix associated with the input layer $R_{i,0}$ is designed to be an identity matrix for all $i \in \mathcal{I}$ to avoid dropping out any of the inputs. Each of the randomization matrices, $R_{i,j}$ for each $i \in \mathcal{I}$, has a user-selected constant number of ones on the diagonal which is denoted by $n_j \in \mathbb{Z}_{>0}$. The value of n_j and the placement of ones on the diagonal respectively determine how many and which of the nodes are not dropped out and are updated at each of the layers. These selected weights influence the adaptation law for the duration of the time before another batch of weights are selected. The randomization matrices are functions of time, and therefore, the placement of non-zero elements on the diagonal of the matrix $R_{i,j}$ randomly changes after every switch. The dropout can be deactivated once the system reaches the steady state by setting the randomization matrices equal to the identity matrix. Additionally, dropout can be stopped after the transient phase by preserving the last values of the randomization matrices for the rest of time which leads to having a sparse DNN (Greene et al., 2020). These two cases can be considered as the final permutation of R_i for all $i \in \mathcal{I}$.

To illustrate the design of the randomization matrix $R_{\rho(x,t),j}$ and the effect of dropout on the DNN architecture, the following example is provided.

Example 1. Consider $R_{i,2} \in \{0,1\}^{3\times 3}$, the weight matrix $V_2 \in \mathbb{R}^{3\times 2}$, and let $n_2 = 1$. Let the switching signal be only time-dependent. Therefore, $n_2 = 1$ of the elements on the diagonal of $R_{i,2}$ are randomly set to 1 and the others are zeroed. Let the switching between each randomization matrix occur every δt s. The considered permutations of $R_{i,2}$ for $i \in \{1,2,3\}$ are

$$R_{1,2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \; R_{2,2} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$R_{3,2} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Let $v_{p\times q} \in \mathbb{R}$ denote to the weight on the pth row and the qth column of V_2, for $p \in \{1,2,3\}$ and $q \in \{1,2\}$. For $p \in \{1,2,3\}$, let $\phi_{2,p} : \mathbb{R}^{L_2} \to \mathbb{R}^{L_2}$ denote the output of the activation functions of the second layer such that the activation vector is $\phi_2 = \left[\phi_{2,1}, \phi_{2,2}, \phi_{2,3}\right]^\top$. Therefore, in the presence and absence of dropout matrix, the outputs are respectively obtained as

$$\begin{aligned} \left(R_{3,2}V_2\right)^\top \phi_2 &= \begin{bmatrix} v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} \\ v_{3,1} & v_{3,2} \end{bmatrix}^\top \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^\top \begin{bmatrix} \phi_{2,1} \\ \phi_{2,2} \\ \phi_{2,3} \end{bmatrix} \\ &= \begin{bmatrix} v_{3,1}\phi_{2,3} \\ v_{3,2}\phi_{2,3} \end{bmatrix}. \end{aligned} \tag{4}$$

$$V_2^\top \phi_2 = \begin{bmatrix} v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} \\ v_{3,1} & v_{3,2} \end{bmatrix}^\top \begin{bmatrix} \phi_{2,1} \\ \phi_{2,2} \\ \phi_{2,3} \end{bmatrix}$$

$$= \begin{bmatrix} v_{1,1}\phi_{2,1} + v_{2,1}\phi_{2,2} + v_{3,1}\phi_{2,3} \\ v_{1,2}\phi_{2,1} + v_{2,2}\phi_{2,2} + v_{3,2}\phi_{2,3} \end{bmatrix}. \tag{5}$$

Comparing (4) and (5) shows how the dropout method deactivates the activation functions associated with the zeros on the diagonal of the randomization matrix. Since the dropout matrix is randomly generated, a new batch of weights are selected every δt s.

Let $\mathcal{C}(\mathcal{X})$ denote the space of continuous functions over the compact set $\mathcal{X} \subseteq \mathbb{R}^n$, where $x \in \mathcal{X}$. According to the universal function approximation property, the function space of (3) is dense in $\mathcal{C}(\mathcal{X})$ (Kidger & Lyons, 2020, Theorem 1.1). Thus, there exists a vector of ideal weights $\theta^* \in \mathbb{R}^{\sum_{j=0}^{k} L_j L_{j+1}}$ such that $\sup_{x\in\mathcal{X}} \|f(x) - \Phi(x, R_i, \theta^*)\| \leq \bar{\varepsilon}$, for all $i \in \mathcal{I}$, where $\bar{\varepsilon} \in \mathbb{R}$ is the function approximation bound across all dropout configurations. Therefore, the drift vector field, $f(x)$, can be modeled as

$$f(x) = \Phi\left(x, R_{\rho(x,t)}, \theta^*\right) + \varepsilon_{\rho(x,t)}(x), \tag{6}$$

where $\varepsilon_{\rho(x,t)} : \mathbb{R}^n \to \mathbb{R}^n$ denotes the unknown function reconstruction error associated with the switching signal $\rho(x,t)$. For all $i \in \mathcal{I}$, the ith function reconstruction error, i.e., ε_i, can be bounded as $\sup_{x\in\mathcal{X}} \|\varepsilon_i(x)\| \leq \bar{\varepsilon}$.

Remark 1. Since using dropout reduces the total number of active neurons in the network, the minimum function reconstruction error that could theoretically be achieved is potentially larger compared to the DNN architecture without dropout. However, this theoretically minimum function reconstruction error is challenging to achieve in practice due to factors related to the training process such as co-adaptation, overfitting, and lack of sufficient training information (Srivastava et al., 2014). By randomly deactivating neurons every few seconds, the effect of dropout is essentially equivalent to introducing an exploratory perturbation in the control and adaptation laws relative to the DNN architecture (Wei et al., 2020). As evident in the subsequent ablation study in Section 5, the effects of dropout mitigate issues such as overfitting and co-adaptation that exist in Lb-DNN adaptive control methods.

Taking the time-derivative of (2) and substituting (1) and (6) yields the open-loop error system

$$\dot{e}(t) = \Phi\left(x, R_{\rho(x,t)}, \theta^*\right) + \varepsilon_{\rho(x,t)}(x) + g(x)u - \dot{x}_d. \tag{7}$$

Remark 2. The developed Lb-DDNN architecture can be represented as an ensemble of J smaller DNNs. Each of these smaller DNNs can approximate the drift dynamic vector of $f(x)$ with a bounded reconstruction error ε_i. The developed method allows switching between each of the smaller DNNs every δt s, and this switching behavior is accounted for in the baseline subsequent stability analysis.

The following assumption facilitates the subsequent control development.

Assumption 2. The vector of ideal weights can be bounded by a known constant $\bar{\theta} \in \mathbb{R}_{>0}$ as $\|\theta^*\| \leq \bar{\theta}$, Lewis et al. (1996, Assumption 1).[5]

[5] If $\bar{\theta}$ is unknown, a robust adaptive approach similar to Stepanyan and Kurdila (2009) can be used.

3.2. Adaptation law

The tracking objective is to design a controller that allows the states x to track the desired state x_d.[6] To compensate for uncertainties in (1), estimates of the ideal weights in (6) are introduced and updated in real-time based on a Lyapunov stability-driven adaptation law. The vectorized estimate of the weight matrices $\hat{\theta} : \mathbb{R}_{\geq 0} \to \mathbb{R}^{\varphi}$ is defined as $\hat{\theta}(t) \triangleq \left[\operatorname{vec}\left(\widehat{V}_0\right)^\top, \operatorname{vec}\left(\widehat{V}_1\right)^\top, \ldots, \operatorname{vec}\left(\widehat{V}_k\right)^\top\right]^\top$, where $\varphi \triangleq \sum_{j=0}^{k} L_j L_{j+1}$ and $\widehat{V}_j : \mathbb{R}^{L_j \times L_{j+1}}$ represents the estimates of weight matrix, for all $j \in \{0, \ldots, k\}$. Let the weight estimation error $\tilde{\theta} : \mathbb{R}_{\geq 0} \to \mathbb{R}^{\varphi}$ be defined as

$$\tilde{\theta}(t) \triangleq \theta^* - \hat{\theta}(t). \tag{8}$$

To capture the nested nonlinearities resulting from the composition of layers, the DDNN estimate be formulized in a recursive manner (inspired by O.S. Patil et al. (2022)) as

$$\widehat{\Phi}_{i,j} = \begin{cases} \left(R_{i,j}\widehat{V}_j\right)^\top \phi_j\left(\widehat{\Phi}_{i,j-1}\right), & j = \{1, \ldots, k\}, \\ \left(R_{i,0}\widehat{V}_0\right)^\top x, & j = 0, \end{cases} \tag{9}$$

for all $i \in \mathcal{I}$, where $\widehat{\Phi}_{i,j}$ is a shorthand notation for $\widehat{\Phi}_{i,j} \triangleq \Phi_j\left(x, R_i, \hat{\theta}\right)$ such that $\widehat{\Phi}_i = \widehat{\Phi}_{i,k}$, where $\widehat{\Phi}_i \triangleq \Phi\left(x, R_i, \hat{\theta}\right)$.

Based on the stability analysis provided in Section 4, the Lb-DDNN weight estimate $\hat{\theta}$ is updated according to

$$\dot{\hat{\theta}} \triangleq \Gamma_\theta \widehat{\Phi}'^{\top}_{\rho(x,t)} e, \tag{10}$$

where $\Gamma_\theta \in \mathbb{R}^{\varphi \times \varphi}$ denotes a positive-definite adaptation gain matrix, and $\widehat{\Phi}'_i$ denotes the Jacobian of $\widehat{\Phi}_i$, defined as $\widehat{\Phi}'_i \triangleq \frac{\partial \Phi\left(x, R_i, \hat{\theta}\right)}{\partial \hat{\theta}}$. For all $j \in \{0, \ldots, k\}$ and $i \in \mathcal{I}$, the Jacobian $\widehat{\Phi}'_i$ can be represented as $\widehat{\Phi}'_i \triangleq \left[\widehat{\Phi}'_{i,0}, \ldots, \widehat{\Phi}'_{i,k}\right]$, where $\widehat{\Phi}'_{i,j} \triangleq \frac{\partial \Phi_j\left(x, R_i, \hat{\theta}\right)}{\partial \hat{\theta}}$.[7] Taking the partial derivative of (9), applying the chain rule, and incorporating the properties of the vectorization operator, the Jacobians $\widehat{\Phi}'_{i,0}$ and $\widehat{\Phi}'_{i,j}$ can be evaluated for all $i \in \mathcal{I}$ as

$$\begin{aligned} \widehat{\Phi}'_{i,0} &\triangleq \left(\prod_{l=1}^{\overleftarrow{k}} \left(R_{i,l}\widehat{V}_l\right)^\top \widehat{\phi}'_l\right)\left(\left(x^\top R_{i,0}\right) \otimes I_{L_1}\right), \\ \widehat{\Phi}'_{i,j} &\triangleq \left(\prod_{l=j+1}^{\overleftarrow{k}} \left(R_{i,l}\widehat{V}_l\right)^\top \widehat{\phi}'_l\right)\left(\left(\widehat{\phi}_j^\top R_{i,j}\right) \otimes I_{L_{j+1}}\right), \end{aligned} \tag{11}$$

for all $j = \{1, \ldots, k\}$, where $\hat{\phi}_j$ is defined as $\hat{\phi}_j \triangleq \phi_j\left(\widehat{\Phi}_{i,j-1}\right)$ and $\hat{\phi}'_j$ is defined as $\hat{\phi}'_j \triangleq \phi'_j\left(\widehat{\Phi}_{i,j-1}\right)$, for all $i \in \mathcal{I}$.

Remark 3. As shown in Example 1, dropout allows selection and updates of a batch of weights which influence the adaptation law for the duration of the time before the next batch of weights are selected. Due to the randomization matrices in the Jacobians in (11), the adaptation law in (10) is not overly reliant on specific weights and neurons which means reduced interdependence and thus mitigated co-adaptation. For the Lb-DDNN described in Example 1, applying (11) yields $\Phi'_{3,2} = \begin{bmatrix} 0 & 0 & \phi_{2,3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \phi_{2,3} \end{bmatrix}$, whereas it is $\Phi'_{3,2} = \begin{bmatrix} \phi_{2,1} & \phi_{2,2} & \phi_{2,3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \phi_{2,1} & \phi_{2,2} & \phi_{2,3} \end{bmatrix}$ without the dropout. As evident from the provided example, $\Phi'_{3,2}$ has less dependencies on the weights with dropouts compared to the architecture without dropouts. Since the dropout architecture involves switching, the structure of $\Phi'_{i,2}$ is different for every $i \in \{1, 2, 3\}$, which changes the dependencies after every switch. As a result, after multiple switches, all Lb-DDNN weights are updated with less interdependence.

3.3. Closed-loop error system and control design

In Section 3.1, a Lb-DDNN is developed to approximate the unknown field vector. An adaptive Lb-DDNN estimation is designed in Section 3.2 such that all weights of the neural network are updated in real-time based on the Lyapunov stability analysis. In this section, the adaptive Lb-DDNN estimate is incorporated in the developed controller to compensate for the effect of the unknown field vector in (1). Based on the subsequent stability analysis, the controller is design as

$$u(t) \triangleq g^+(x)\left(\dot{x}_d - \widehat{\Phi}_{\rho(x,t)} - k_e e - k_s \operatorname{sgn}(e)\right), \tag{12}$$

for all $i \in \mathcal{I}$, where $k_e, k_s \in \mathbb{R}_{>0}$ are constant control gains. Substituting the controller into the open-loop error system in (7) yields

$$\dot{e}(t) = \Phi\left(x, R_{\rho(x,t)}, \theta^*\right) - \widehat{\Phi}_{\rho(x,t)} + \varepsilon_{\rho(x,t)}(x) - k_s \operatorname{sgn}(e) - k_e e. \tag{13}$$

To facilitate the stability analysis, a first-order Taylor series approximation is applied to $\Phi\left(x, R_{\rho(x,t)}, \theta^*\right) - \widehat{\Phi}_{\rho(x,t)}$ as in Lewis et al. (1996, Eq. 22) to yield

$$\Phi\left(x, R_{\rho(x,t)}, \theta^*\right) - \Phi\left(x, R_{\rho(x,t)}, \hat{\theta}\right) = \widehat{\Phi}'_{\rho(x,t)}\tilde{\theta} + \mathcal{O}^2_{\rho(x,t)}\left(\left\|\tilde{\theta}\right\|\right), \tag{14}$$

where $\mathcal{O}^2_{\rho(x,t)}\left(\left\|\tilde{\theta}\right\|\right) \in \mathbb{R}^n$ denotes the higher-order terms. Substituting (14) into (13) yields

$$\dot{e} = \widehat{\Phi}'_{\rho(x,t)}\tilde{\theta} + \mathcal{O}^2_{\rho(x,t)}\left(\left\|\tilde{\theta}\right\|\right) + \varepsilon_{\rho(x,t)}(x) - k_s \operatorname{sgn}(e) - k_e e. \tag{15}$$

4. Stability analysis

Let the concatenated vector $z : \mathbb{R}_{\geq 0} \to \mathbb{R}^{\psi}$ be defined as

$$z(t) \triangleq \left[e^\top(t), \tilde{\theta}^\top(t)\right]^\top, \tag{16}$$

where $\psi \triangleq n + \sum_{j=0}^{k} L_j L_{j+1}$. Due to the discontinuities in (10) and (15), the subsequent stability analysis focuses on generalized solutions. The convergence properties of each subsystem is subsequently analyzed and used to conclude the properties of the overall switched system. The subsequent analysis uses the switching analysis technique in Kamalapurkar et al. (2019).

Let $F_i : \mathbb{R}^{\psi} \times \mathbb{R}_{\geq 0} \rightrightarrows \mathbb{R}^{\psi}$ be a set-valued map such that the differential inclusion of the form $\dot{z} \in F_i(z, t)$ holds. Using the calculus of $\mathrm{K}[\cdot]$ from Paden and Sastry (1987), a set of regularization of the closed-loop error systems for each $i \in \mathcal{I}$ can be computed as $F_i(z, t) \subseteq F'_i(z, t)$, where

$$F'_i(z, t) = \begin{bmatrix} \widehat{\Phi}'_i\tilde{\theta} + \mathcal{O}^2_i\left(\left\|\tilde{\theta}\right\|\right) + \varepsilon_i(x) - k_e e - k_s \mathrm{K}[\operatorname{sgn}](e) \\ -\Gamma_\theta \widehat{\Phi}'^{\top}_i e \end{bmatrix}. \tag{17}$$

To show that the designed controller in (12) and the Lb-DDNN update law introduced in (10) fulfills the tracking objective, a Lyapunov-based stability analysis is performed using switched

[6] Various robust and adaptive strategies can be applied to achieve this objective, including the use of a shallow neural network. However, DNNs have consistently demonstrated improved theoretical and empirical outcomes when compared to such methods, including shallow neural networks (Lamb et al., 2023; Le et al., 2025; Rolnick & Tegmark, 2018).

[7] When a weight is dropped out, the corresponding weight update law also becomes zero as a result of the multiplication with the randomization matrices in (10). When the weight is not dropped out, the weight update law is no longer zero. Thus, the weight update resumes from the previous value.

systems techniques. Let the Lyapunov function candidate denoted as $V_L : \mathcal{D} \to \mathbb{R}_{\geq 0}$ be defined as

$$V_L(z) \triangleq \frac{1}{2} e^\top e + \frac{1}{2} \tilde{\theta}^\top \Gamma_\theta^{-1} \tilde{\theta}. \tag{18}$$

The Lyapunov function candidate can be lower- and upper-bounded as

$$\underline{\alpha} \|z\|^2 \leq V_L(z) \leq \overline{\alpha} \|z\|^2, \tag{19}$$

where $\underline{\alpha}, \overline{\alpha} \in \mathbb{R}_{\geq 0}$ are known constants defined as $\underline{\alpha} \triangleq \frac{1}{2} \min\left(1, \lambda_{\min}\left(\Gamma_\theta^{-1}\right)\right)$ and $\overline{\alpha} \triangleq \max\left(1, \lambda_{\max}\left(\Gamma_\theta^{-1}\right)\right)$. Based on the subsequent stability analysis, define $\mathcal{S} \triangleq \left\{\varsigma \in \mathbb{R}^\psi : \|\varsigma\| \leq \omega \sqrt{\underline{\alpha}/\overline{\alpha}}\right\}$ as the set of stabilizing initial conditions, where $\omega \in \mathbb{R}_{>0}$ is a positive bounding constant. By the universal function approximation property, (6) holds only on a compact domain $\mathcal{X}$. Consider the set $\Upsilon \subseteq \mathcal{X}$ defined as $\Upsilon \triangleq \{\varsigma \in \mathcal{X} : \|\varsigma\| < \overline{x_d} + \omega\}$. To ensure $x \in \mathcal{X}$ holds, for all $t \in \mathbb{R}_{\geq 0}$, the stability analysis needs to ensure that $z(t)$ is constrained to a closed and bounded set $\mathcal{D} \triangleq \{\varsigma \in \mathbb{R}^\varphi : \|\varsigma\| \leq \omega\}$, where $\chi \in \mathbb{R}_{>0}$ is a bounding constant, for all $t \in \mathbb{R}_{\geq 0}$. The subsequent analysis establishes, based on the growth of V_L, that if z is initialized in $\mathcal{S}$ (which is a strict subset of $\mathcal{D}$), the trajectories of z do not escape $\mathcal{D}$.

Using Definition 1, the generalized time derivative of the Lyapunov function candidate for each $i \in \mathcal{I}$ can be computed as

$$\dot{\tilde{V}}_L(z,t) = \max_{q \in F_i(z,t)} z^\top q.$$

Using (17), $\dot{\tilde{V}}_L(z,t)$ can be upper-bounded as $\dot{\tilde{V}}_L(z,t) \leq \max_{q \in F_i'(z,t)} z^\top q$, and thus

$$\begin{aligned}\dot{\tilde{V}}_L(z,t) &\leq \max_{q \in F_i'(z,t)} z^\top q \\ &= e^\top \left\{\widehat{\Phi}_i' \tilde{\theta} + \mathcal{O}_i^2\left(\left\|\tilde{\theta}\right\|\right) + \varepsilon_i(x)\right\} \\ &\quad - k_e e^\top e - k_s e^\top \mathrm{K}[\mathrm{sgn}](e) \\ &\quad - \tilde{\theta}^\top \left\{\Gamma_\theta^{-1}\left(\Gamma_\theta \widehat{\Phi}_i'^\top e\right)\right\}. \end{aligned} \tag{20}$$

Canceling the cross terms yields

$$\begin{aligned}\dot{\tilde{V}}_L(z,t) &\leq e^\top \left\{\mathcal{O}_i^2\left(\left\|\tilde{\theta}\right\|\right) + \varepsilon_i(x) - k_e e\right\} \\ &\quad - k_s e^\top \mathrm{K}[\mathrm{sgn}](e). \end{aligned} \tag{21}$$

The term $\mathcal{O}_i^2\left(\left\|\tilde{\theta}\right\|\right)$ can be upper-bounded similar to O. Patil et al. (2022, Eq. 18) as $\sup \left\|\mathcal{O}_i^2\left(\left\|\tilde{\theta}\right\|\right)\right\| \leq \Delta$, for all $i \in \mathcal{I}$ and when $\|z(t)\| \in \mathcal{D}$, where $\Delta \in \mathbb{R}_{>0}$ denotes a known constant.

Theorem 1. Consider the dynamical system in (1). The controller designed in (12) with the Lb-DDNN update law in (10) ensures asymptotic tracking error convergence in the sense that $\lim_{t \to \infty} \|e(t)\| = 0$, for all $t \in \mathbb{R}_{\geq 0}$, provided $z(0) \in \mathcal{S}$ and the gain condition $k_s > \overline{\varepsilon} + \Delta$ is satisfied.

Proof. Consider the Lyapunov function candidate in (18) and the inequalities in (20) and (21). Using the bound on the higher order terms, (21) can be further upper-bounded as

$$\dot{\tilde{V}}_L(z,t) \leq -k e^\top e + \left|e^\top\right| (\Delta + \overline{\varepsilon}) - k_s e^\top \mathrm{K}[\mathrm{sgn}](e),$$

when $z(t) \in \mathcal{D}$. Invoking (Krstic et al., 1995, Lemma E.1.IV), $\dot{\tilde{V}}_L$ can be further upper-bounded as

$$\dot{\tilde{V}}_L(z,t) \leq -k_e e^\top e - (k_s - \Delta - \overline{\varepsilon}) \|e\|, \tag{22}$$

when $z(t) \in \mathcal{D}$. By selecting $k_s > \overline{\varepsilon} + \Delta$, (22) can be further upper-bounded as

$$\dot{\tilde{V}}_L(z,t) \overset{\text{a.a.t.}}{\leq} -k_e \|e\|^2, \tag{23}$$

when $z(t) \in \mathcal{D}$. From (18), the inequality obtained in (23), Kamalapurkar et al. (2019, Theorem 2) can be invoked to conclude that $\lim_{t \to \infty} \|e(t)\| = 0$, when $z(t) \in \mathcal{D}$. The desired velocity is bounded by design, and therefore, $x_d, \dot{x}_d \in \mathcal{L}_\infty$. From (23), $e \in \mathcal{L}_\infty$. Therefore, $x \in \mathcal{L}_\infty$. Since $\tilde{\theta} \in \mathcal{L}_\infty$ and using Assumption 2, $\hat{\theta} \in \mathcal{L}_\infty$. Since $\hat{\theta} \in \mathcal{L}_\infty$, and $\widehat{\Phi}_i$ is a smooth function, then $\widehat{\Phi}_i \in \mathcal{L}_\infty$, for all $i \in \mathcal{I}$. Since $\dot{x}_d$, e, $\widehat{\Phi} \in \mathcal{L}_\infty$, then $u \in \mathcal{L}_\infty$. Since $z(t_0) \in \mathcal{S}$, then $\|z(t_0)\| \leq \omega \sqrt{\underline{\alpha}/\overline{\alpha}}$. Using (19), $\|e(t)\| \leq \omega$. Thus, using (2), $\|x\|$ can be bounded as $\|x\| \leq \overline{x_d} + \omega$. Hence, $x \in \Upsilon \subseteq \mathcal{X}$, and therefore the universal function approximation holds.

5. Simulation

5.1. Methods

To demonstrate the performance and efficacy of the Lb-DDNN adaptive controller, an ablation study is provided based on simulations performed on a five-dimensional nonlinear system (i.e., Dynamical Model 1). Using the model in (1), the drift vector field f is

$$f = \begin{bmatrix} x_1 x_2^2 \tanh(x_2) + \sin(x_1)^2 \\ \cos(x_1 + x_2 + x_3)^3 - \exp(x_2)^2 + x_1 x_2 \\ x_3^2 \log(1 + \mathrm{abs}(x_1 - x_2)) \\ \sin\left(x_1 + x_2^2\right)^2 - \exp(x_3)^{1/2} + (x_1 + x_3) x_2 \\ \tanh(x_2) + \cos(x_1)^2 \end{bmatrix},$$

where $x \triangleq [x_1, x_2, x_3, x_4, x_5]^\top : \mathbb{R}_{\geq 0} \to \mathbb{R}^5$ denotes the system state. Simulation experiments are performed for 10 s each with the initial condition $x(0) = [5, 1, -5, 1, 3]^\top$. The desired trajectory is selected as

$$x_d(t) = \begin{bmatrix} 1 + \sin(2t) + \cos(0.5t)^2 \\ 1 - \cos(t) \\ 1 + \sin(3t) + \cos(-2t) \\ 1 + \sin(t)^2 \\ 1 - \sin(2t)^2 \cos(t) \end{bmatrix},$$

and the control effectiveness matrix is selected as $g = I_{5\times 5}$. The DNN used in the simulations has $k = 25$ inner layers with $L_j = 10$ neurons each and hyperbolic tangent activation functions. The switching signal is selected to be only time-dependent for the simulations such that every δt s the matrix R_i switches to R_{i+1}.

The ablation study consists of ten sets of simulations. The first set of simulations are performed to compare the baseline adaptive DNN-based architecture in O.S. Patil et al. (2022) (Lb-DNN 1) and the Lb-DDNN adaptive controller in (10) and (12) (Lb-DDNN 1). The second set of simulations are performed to examine the effect of different values of δt on the performance of the developed method (Lb-DDNN 2, 3). Additionally, for the third set of simulations, a figure is included to show the performance of the Lb-DDNN adaptive controller as δt changes (Lb-DDNN 4). The fourth set of simulations are performed to compare the performance in the absence and presence of dropout deactivation after the transient period (Lb-DDNN 5, 6). The fifth set of simulations are performed to examine the tracking and function approximation performance of the Lb-DDNN in two cases; in the first case (Lb-DDNN 7), the dropout is deactivated after the transient period, in the second (Lb-DDNN 8), the last Lb-DDNN architecture that is used before entering the steady-state is kept for the rest of the simulation. Hence, instead of using the baseline DNN-based adaptive architecture in the steady state, the sparse representation of the whole architecture is utilized. The sixth set of simulations are performed to compare the tracking and function approximation performance of a Lb-DDNN (Lb-DDNN 9) and a narrow DNN (Lb-DNN 2) with width equal to the number of active neurons in the dropout architecture. The seventh

Table 1
Ablation study parameters.

DNN architecture	k_e	k_s	Dropout deactivation/freeze time (s)	δt (s)	Γ (transient phase)	Γ (steady state phase)
Lb-DNN 1	15	0.001	NA	NA	100	100
Lb-DDNN 1	15	0.001	2	0.2	100	20
Lb-DDNN 2	15	0.001	2	0.2	100	20
Lb-DDNN 3	15	0.001	2	0.05	100	20
Lb-DDNN 4	15	0.001	2	0.1–2	100	20
Lb-DDNN 5	15	0.001	NA	0.2	100	20
Lb-DDNN 6	15	0.001	1	0.2	100	20
Lb-DDNN 7	15	0.001	2	0.2	100	20
Lb-DDNN 8	15	0.001	2	0.2	100	20
Lb-DNN 2	5	0.001	NA	NA	40	40
Lb-DDNN 9	5	0.001	NA	0.1	40	20
Lb-DNN 3	15	0.001	NA	NA	100	100
Lb-DDNN 10	15	0.001	2	0.2	100	20
Lb-DDNN 11	15	0.001	2	0.2	100	20
Lb-DNN 4	15	0.001	NA	NA	100	100
Lb-DDNN 12	15	0.001	2	0.2	100	20
Lb-DDNN 13	15	0.001	NA	0.2	100	20
Pruning DNN	15	0.001	NA	0.2	100	20

Table 2
Performance comparison results.

DNN architecture	RMS $\|e\|$	RMS $\|f - \widehat{\Phi}\|$	RMS $\|u\|$
Lb-DNN 1	1.36	43.51	736.60
Lb-DDNN 1	0.89	24.50	487.06
Lb-DDNN 2	0.93	27.59	520.22
Lb-DDNN 3	1.15	25.80	482.81
Lb-DDNN 5	0.99	27.19	411.13
Lb-DDNN 6	0.90	26.391	513.51
Lb-DDNN 7	0.94	25.18	492.83
Lb-DDNN 8	0.98	30.52	466.35
Lb-DNN 2	1.94	83.51	91.33
Lb-DDNN 9	1.85	82.32	90.15
Lb-DNN 3	1.35	50.50 (off-trajectory)	735.51
Lb-DDNN 10	0.97	22.63 (off-trajectory)	363.38
Lb-DNN 4	0.68	21.68	426.11
Lb-DDNN 12	0.63	17.12	333.66
Pruning DNN	1.48	50.43	821.50
Lb-DDNN 13	1.00	31.76	480.28

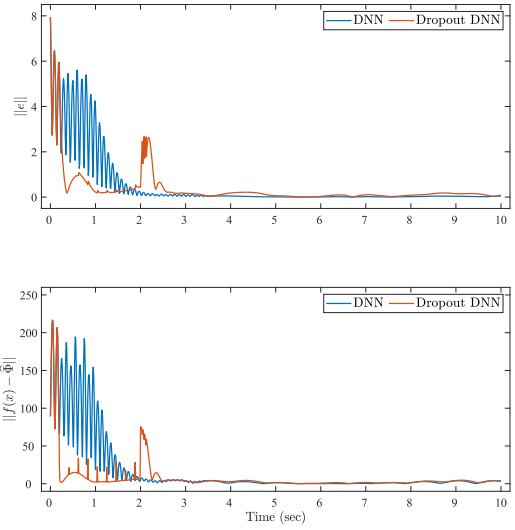


Fig. 2. Performance of the tracking error (top) and function approximation error (bottom) over time comparing the Lb-DDNN controller (Lb-DDNN 1) and the adaptive DNN controller (Lb-DNN 1) developed in O.S. Patil et al. (2022). For Lb-DDNN 1, the dropout deactivated after 2 s.

set of simulations are performed to compare the generalization performance (quantified by the root mean square (RMS) function approximation error $\left\|f(x) - \widehat{\Phi}\right\|$) of the adaptive Lb-DDNN controller (Lb-DDNN 10) compared to the baseline DNN-based adaptive architecture in O.S. Patil et al. (2022) (Lb-DNN 3). For the eighth set of simulations, Fig. 9 indicates the performance of the Lb-DDNN adaptive controller with respect to the number of neurons that are kept in each of the layers (Lb-DDNN 11). The ninth set of simulations are performed to verify the efficacy of the first set of simulation on a different dynamical system (Lb-DDNN 12 and Lb-DNN 4), referred to as Dynamical Model 2. Finally, simulations are performed to compare the performance of the proposed Lb-DDNN adaptive control (Lb-DDNN 13) with a pruning DNN algorithm.

In all simulations, the DNN weight estimates are initialized randomly from the normal distribution $\mathcal{N}(0, 10)$. The tuning parameters for each of the simulations are empirically selected according to Table 1, and the performance results of the simulations are presented in Table 2.

5.2. Results

5.2.1. Comparison with Lb-DNN adaptive controller

As shown in the first subplot of Fig. 2, the tracking error for Lb-DDNN 1 converges significantly faster than Lb-DNN 1. Specifically, Lb-DDNN 1 results in convergence to the final error after approximately 1 s, roughly two times faster than Lb-DNN 1. Once dropout is deactivated, all weights are reactivated, causing a sudden error overshoot for the previously dropped weights because they are now updated based on the feedback. As indicated in Table 2, despite the jump in the tracking error after the deactivation of the dropout[8] of the DNN weights, the Lb-DDNN 1 still yields the norm of the RMS tracking error of 0.89, a 34.56% improvement when compared to Lb-DNN 1. Moreover, Lb-DNN 1 presents more oscillatory behavior within the transient period than Lb-DDNN 1. Lb-DDNN 1 mitigates co-adaptation yielding less oscillatory behavior. However, since the adaptation gain is reduced in the steady-state for Lb-DDNN 1, some residual error is observed, resulting in a longer time for the error to converge to zero. As shown in the second subplot of Fig. 2, the function approximation error for Lb-DDNN 1 rapidly converges after less than 1 s but takes approximately 2 s for Lb-DNN 1 to converge. Although there is a jump in the function approximation error after the deactivation of the dropout method, Lb-DDNN 1 demonstrated a 43.69% improvement in function approximation performance as indicated in Table 2. Thus, Lb-DDNN 1 resulted

[8] The stability analysis holds for both cases deactivating dropout method or keeping dropout throughout the simulation.

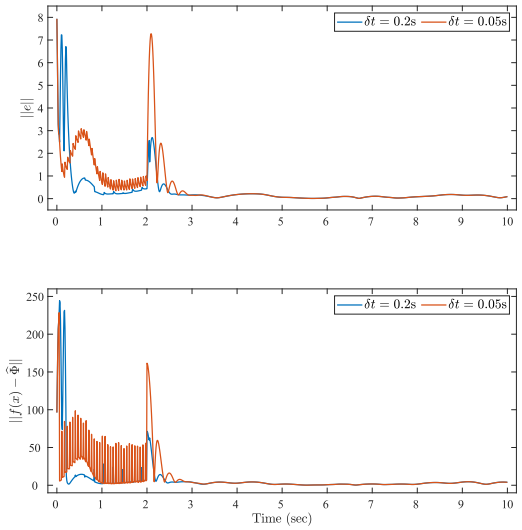


Fig. 3. Performance of the tracking error (top) and function approximation error (bottom) over time for Lb-DDNN with two different δt values (Lb-DDNN 2, 3).

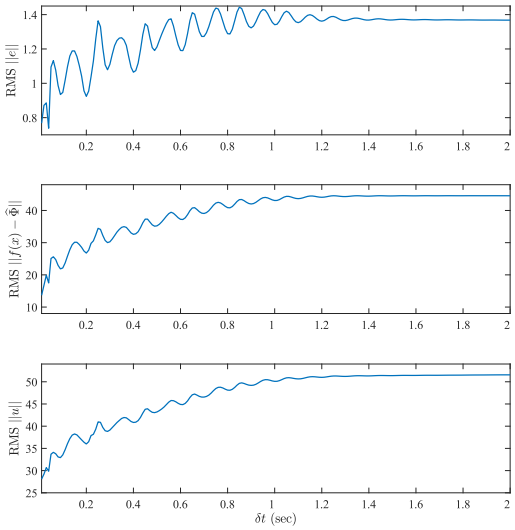


Fig. 4. The RMS of the norm of tracking error (top), RMS of the norm of function approximation error (middle), and RMS of the norm of control effort (bottom) with respect to changes in the value of δt (Lb-DDNN 4).

in better transient behavior and improved tracking and function approximation performance with a 33.88% lower control effort when compared to Lb-DNN 1. Additionally, the computaton time for the Jacobians of Lb-DNN 1 and Lb-DDNN 1 is reported as 0.0026 s and 0.0025 s, respectively. This demonstrates comparable computational performance between the proposed Lb-DDNN and the baseline Lb-DNN.

5.2.2. Effect of dropout frequency

Simulations (Lb-DDNN 2, 3, and 4) also provide insight into the performance of the Lb-DDNN adaptive controller as the dropout frequency varies. The dropout method is deactivated after 2 s Fig. 4 illustrates the RMS of the norm of tracking error, the function approximation error, and control effort for Lb-DDNN 4 as δt increases from 0.1 s to 2 s The results suggest that the increase of δt generally yields decreased performance. Therefore, choosing $\delta t \lesssim 0.2$ s leads to smaller tracking and function approximation errors with less control input. Moreover, choosing $\delta t \gtrsim 1.2$ s yields larger tracking and function approximation errors even with more control effort. A sudden decrease in performance is observed when $\delta t = 0.24$ s.

To examine the effect of selecting different δt on the transient performance, simulations are performed with $\delta t = 0.2$ s (Lb-DDNN 2) and $\delta t = 0.05$ s (Lb-DDNN 3) using the Lb-DDNN controller. As shown in Fig. 3, reducing δt causes more spikes in the tracking and function approximation performances. The increased amount of spikes are due to the switching Lb-DDNN architecture. Reducing δt is found to cause more spikes but slightly lowers the RMS of the norm of the control effort, as shown in Fig. 3 and Table 2. As shown in Section 5.2.1, the introduction of dropout improves tracking and function approximation error performance despite causing discontinuities. However, a sufficiently small dropout frequency can lead to chattering behavior as observed in Fig. 3.

5.2.3. Effect of dropout deactivation time

Traditionally, dropout is only utilized for offline training techniques and not during implementation. The third set of simulations examine the performance of the developed Lb-DDNN controller under two cases; dropping out the neurons for the entire duration of the simulation (Lb-DDNN 5), and deactivating

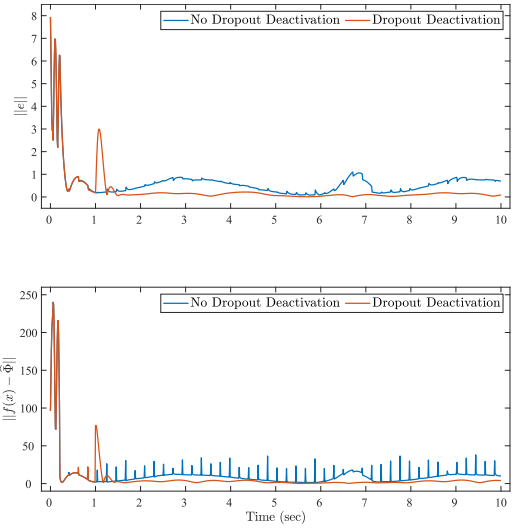


Fig. 5. Performance of the tracking error (top) and function approximation error (bottom) over time in the presence (Lb-DDNN 6) and absence (Lb-DDNN 5) of dropout deactivation in the steady state (after 1 s).

dropout method after 1 s (Lb-DDNN 6). As shown in Fig. 5, for both cases, the difference between the tracking and function approximation performances are insignificant during the first second, as expected. For Lb-DDNN 6, there is an overshoot in both tracking and function approximation errors which does not occur for Lb-DDNN 5. However, not deactivating the dropout in the steady state leads to more spikes in both tracking and function approximation error in steady state. Although dropout is deactivated during steady state, the weights are still updating. Since weights in both architectures are updated with the same adaptive update law, the impact of dropout remains from the transient phase. Therefore, strategically switching to a Lb-DNN in the steady state benefits the Lb-DDNN and eliminates spikes in the steady state. Despite the increase in the control input, Lb-DDNN 6 leads to slightly lower control input and function approximation error as shown in Table 2.

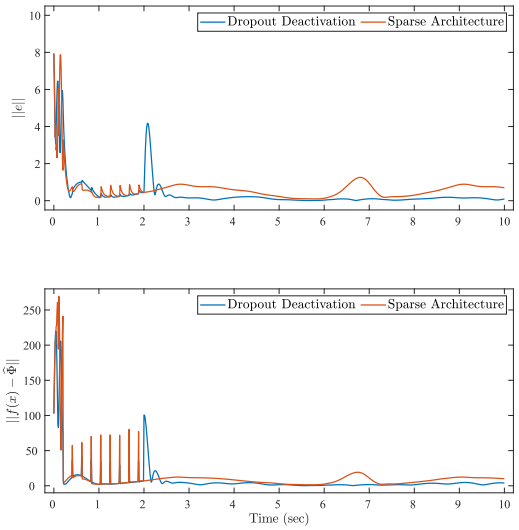


Fig. 6. Performance of the tracking error (top) and function approximation error (bottom) over time with dropout deactivation (Lb-DDNN 7) or preserving the last sparse DNN architecture (Lb-DDNN 8) in the steady state (after 2 s).

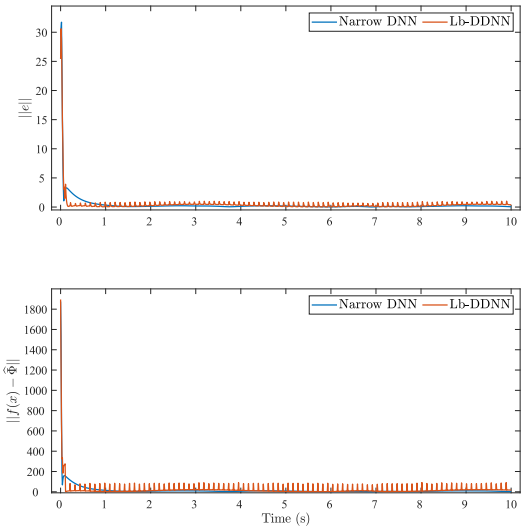


Fig. 7. Performance of the tracking error (top) and function approximation error (bottom) over time for a narrow DNN (Lb-DNN 2) and a dropout DNN (Lb-DDNN 9).

5.2.4. Comparison with sparse Lb-DNN adaptive controller

The fifth set of simulations are performed to compare the performance of the designed Lb-DDNN controller in two different scenarios. In the first scenario, the dropout is completely deactivated after the transient phase (2 sec, Lb-DDNN 7) which means that all the neurons and weights are activated and updated at all times similar to the baseline Lb-DNN architecture in O.S. Patil et al. (2022). Due to the sudden deactivation of dropout, an overshoot is observed in both the tracking and function approximation errors at 2 s as shown in Fig. 6. For the second scenario (Lb-DDNN 8), instead of deactivating the dropout altogether, the last dropout architecture before entering the steady state (i.e., sparse architecture) is used for the rest of the simulation. Therefore, for the rest of the simulations, only the weights that were not dropped in the sparse architecture are updated. Using the sparse network, Fig. 6 indicates that overshoot entering steady state is avoided. Despite the overshoot, both architectures represent approximately similar RMS tracking error performance, function approximation performance, and control effort throughout the steady state.

5.2.5. Comparison with narrow Lb-DNN adaptive controller

The sixth simulation is performed to compare the tracking and function approximation of a Lb-DDNN controller (Lb-DDNN 9) and a narrow DNN which has the same width as the dropout architecture (Lb-DNN 2) with the initial condition $x(0) = [12, 1, 15, -15, 12]^\top$. As shown in Fig. 7, the tracking error of Lb-DDNN 9 converges after 0.15 s while the tracking error of Lb-DNN 2 converges after 1 s. Additionally, the function approximation error for Lb-DDNN 9 converges almost 10 times faster than Lb-DNN 2. Based on Table 2, Lb-DDNN 9 yields lower RMS tracking and function approximation error with approximately the same control effort.

5.2.6. Off-trajectory function approximation comparison with Lb-DNN adaptive controller

The seventh simulation examines the RMS function approximation error performance of Lb-DNN 3 and Lb-DDNN 10 on a test dataset involving 100 random off-trajectory points selected from the uniform distribution $U(-0.25, 0.25)$ to evaluate the function approximation capability of the architecture beyond the explored points (i.e., generalization ability). For Lb-DDNN 10, the dropout is deactivated after the transient phase (2 sec). As the results in Fig. 8 suggest, Lb-DDNN 10 has a noticeably faster convergence during the transient phase compared to Lb-DNN 3 due to the elimination of co-adaptation in the learning phase. Based on the 43.69% improvement in function approximation performance of Lb-DDNN 1 over Lb-DNN 1 and the 55.19% improvement in function approximation performance of Lb-DDNN 9 over Lb-DNN 3, it can be concluded that using dropout method can mitigate the effect of overfitting.

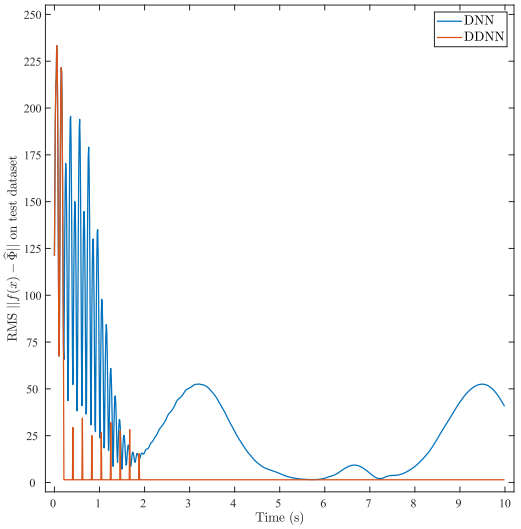


Fig. 8. Comparative plots of the RMS function approximation error norm $\left\|f(x) - \widehat{\Phi}\right\|$ on off trajectory points the Lb-DDNN controller (Lb-DDNN 10), dropout deactivation in the steady state (after 2 s) and the adaptive DNN controller (Lb-DNN 3) developed in O.S. Patil et al. (2022).

5.2.7. Effect of number of dropout neurons

The eighth simulation (Lb-DDNN 11) examines the effect of the number of neurons that are remained on the overall performance of the system. As shown in Fig. 9, as the number of active neurons in each layer of the DDNN architecture increases, the overall performance of the system decreases. As observed in Fig. 9, the RMS of the tracking error is at its lowest when only one

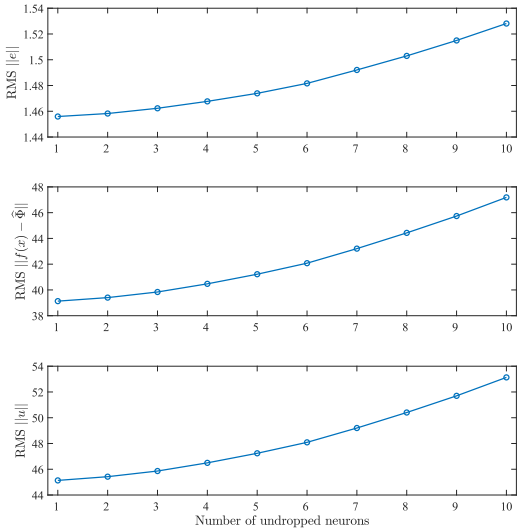


Fig. 9. The RMS of the norm of tracking error (top), RMS of the norm of function approximation error (middle), and RMS of the norm of control effort (bottom) with respect to the number of neurons that are kept in each layer of the DDNN architecture (Lb-DDNN 11).

neuron remains. Essentially, as the DDNN architecture gets closer to the baseline DNN architecture in O.S. Patil et al. (2022) (where all the weights are actively updating), the RMS of the norm of the tracking error, function approximation error, and control input increases.

5.2.8. Lb-DDNN on dynamical model 2

To further demonstrate the efficacy of the developed method, the same architecture, with similar gains and parameters as Lb-DDNN 1, is applied to a different drift dynamic with the initial condition $x(0) = [5, 1, -1]^\top$ and with a different desired trajectory and dimension. The drift vector and the desired trajectory are selected as

$$f = \begin{bmatrix} x_2 \sin(x_2) + x_1 \\ \exp(-x_3) \\ x_1 x_2 + x_3^2 \end{bmatrix}, \quad x_d = \begin{bmatrix} 1 + \cos(0.5t)^2 \\ 1 + \sin(2t) \\ 1 - \cos(t)\sin(t)^2 \end{bmatrix},$$

respectively. For Lb-DDNN 12, dropout is deactivated after 2 s. Based on Table 2, the developed Lb-DDNN 12 continues to exhibit improved performance compared to Lb-DNN 4. The tracking error performances are approximately the same, the function approximation improves by 21.01%, and the control effort is reduced by 21.70%.

5.2.9. Lb-DDNN vs. pruning DNN

Pruning of DNNs has recently emerged as a method for enhancing the energy efficiency of DNN-based applications. This approach involves eliminating redundant or less significant connections, which can potentially boost performance and reduce energy consumption while maintaining approximately equal accuracy (cf., Blalock et al., 2020). Although pruning methods have been extensively investigated for offline learning problems, these methods do not apply for online learning. To construct a pruning method for an online learning-based control setting, Algorithm 1 is introduced. While pruning focuses on investigating the effect of each neuron and reducing the size of the architecture, dropout focuses on temporarily freezing neurons randomly to decrease co-adaptation. Simulation results are provided to compare the tracking and function approximation performances of the developed Lb-DDNN and a momentum-based pruning algorithm described in Algorithm 1.

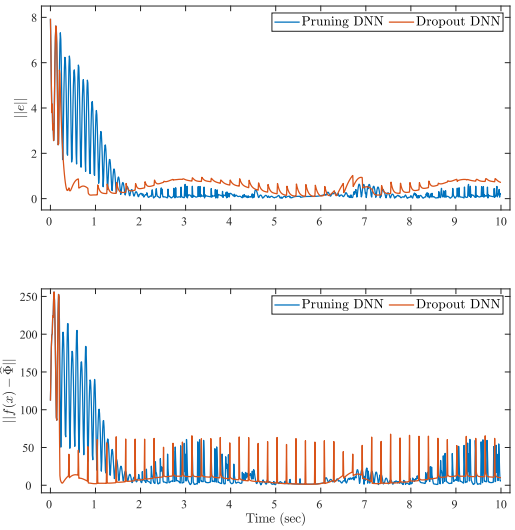


Fig. 10. Performance of the tracking error (top) and function approximation error (bottom) over time comparing the Lb-DDNN 13 and the Pruning DNN in Algorithm 1.

Algorithm 1 Pruning DNN

(1) Every δt seconds:

  (a) Select one element of the pruning matrix P_i , where P_i is an identity matrix of the same size as the randomization matrix R_i.
  (b) Set the selected element to zero.

(2) Keep the selected element unchanged for δt seconds.
(3) After δt seconds,

  (a) Compute the error change as $\delta e = \|e(t - \delta t) - e(t)\|$.
  (b) Compute the momentum as $M(t) = \beta M(t - \delta t) + (1 - \beta)\delta e$, where $M \in \mathbb{R}$ denotes the momentum initialized at zero, and β denotes a user-selected constant.
  (c) Is $M <$ Pruning threshold?

    (i) Yes: set the selected element to zero, permanently.
    (ii) No: set the selected element to one, permanently.

(4) Repeat for the next element in P_i.

In this simulation, the pruning threshold is set to 0.1 and the constant β is set to 0.5. Based on Table 2, the tracking error shows a 32.43% improvement, the function approximation improves by 37.02%, and the control effort of the Lb-DDNN is reduced by 41.54% over the Pruning DNN. Based on Fig. 10, the tracking error and function approximation of Lb-DDNN 13 converges to steady state after approximately 0.4 s and 0.25 s, respectively. However, the tracking error and function approximation of the Pruning DNN converges to steady state after approximately 1.7 s and 1.8 s, respectively. The tracking and function approximation performance for both Lb-DDNN 13 and the Pruning DNN in Algorithm 1 are similar in the steady state.

6. Conclusion and future work

This work developed the first Lb-DDNN to compensate for the unknown drift vector field in a general control-affine nonlinear dynamic system for the trajectory tracking objective while mitigating co-adaptation and overfitting. The dropout technique randomly selects a batch of neurons at each layer of the DNN and equate the associated weights to zero while updating the remaining weights according to a Lyapunov-based update law. Every δt s, a new batch of neurons are randomly selected, and at each δt, the remaining weights are actively updated. Through a rigorous switching Lyapunov-based stability analysis, asymptotic convergence of the tracking error is guaranteed in addition to the estimation error of Lb-DDNN weights remaining bounded. Mitigation of co-adaptation is shown theoretically (i.e., (11) and Remark 3), and the mitigation of overfitting is shown in the ablation study (i.e., Section 5.2.6). Simulation results show 34.56% and 43.69% improvement in the tracking error and function approximation error, respectively, with a 33.88% reduced control effort when compared to the baseline adaptive DNN controller in O.S. Patil et al. (2022). Additional simulations showed the effect of dropout during both transient and steady state periods and how modifying dropout parameters can affect system performance.

References

Abbeel, P., Coates, A., & Ng, A. Y. (2010). Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, *29*(13), 1608–1639.

Anderson, J. A. (1995). *An introduction to neural networks*. MIT Press.

Ba, J., & Frey, B. (2013). Adaptive dropout for training deep neural networks. *Advances in Neural Information Processing Systems*, *26*.

Bansal, S., Akametalu, A. K., Jiang, F. J., Laine, F., & Tomlin, C. J. (2016). Learning quadrotor dynamics using neural network for flight control. In *Proc. IEEE conf. decis. control* (pp. 4653–4660).

Bernstein, D. S. (2009). *Matrix mathematics*. Princeton University Press.

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., & Guttag, J. (2020). What is the state of neural network pruning? *Proceedings of Machine Learning and Systems*, *2*, 129–146.

Brunton, S. L., & Kutz, J. N. (2019). *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press.

Charu, C. A. (2018). *Neural networks and deep learning: a textbook*. Spinger.

Dahl, G. E., Sainath, T. N., & Hinton, G. E. (2013). Improving deep neural networks for LVCSR using rectified linear units and dropout. In *2013 IEEE int. conf. on acoust., speech & signal process.* (pp. 8609–8613).

Greene, M. L., Deptula, P., Nivison, S., & Dixon, W. E. (2020). Sparse learning-based approximate dynamic programming with barrier constraints. *IEEE Control Systems Letters*, *4*(3), 743–748.

Griffis, E., Patil, O., Bell, Z., & Dixon, W. E. (2023). Lyapunov-based Long Short-Term Memory (Lb-LSTM) neural network-based control. *IEEE Control Systems Letters*, *7*, 2976–2981.

Hart, R., Patil, O., Griffis, E., & Dixon, W. E. (2023). Deep Lyapunov-based physics-informed neural networks (DeLb-PINN) for adaptive control design. In *Proc. IEEE conf. decis. control* (pp. 1511–1516).

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. ArXiv Preprint.

Joshi, G., & Chowdhary, G. (2018). Adaptive control using Gaussian-process with model reference generative network. In *IEEE conf. decis. control* (pp. 237–243).

Joshi, G., & Chowdhary, G. (2019). Deep model reference adaptive control. In *Proc. IEEE conf. decis. control* (pp. 4601–4608).

Kamalapurkar, R., Rosenfeld, J. A., Parikh, A., Teel, A. R., & Dixon, W. E. (2019). Invariance-like results for nonautonomous switched systems. *IEEE Transactions on Automatic Control*, *64*(2), 614–627.

Kidger, P., & Lyons, T. (2020). Universal approximation with deep narrow networks. In *Conf. learn. theory* (pp. 2306–2327).

Krstic, M., Kanellakopoulos, I., & Kokotovic, P. V. (1995). *Nonlinear and adaptive control design*. New York: John Wiley & Sons.

Lamb, Z., Bell, Z. I., Longmire, M., Paquet, J., Ganesh, P., & Sanfelice, R. (2023). Deep nonlinear adaptive control for unmanned aerial systems operating under dynamic uncertainties. arXiv preprint arXiv:2310.09502.

Le, D., Greene, M., Makumi, W., & Dixon, W. E. (2022). Real-time modular deep neural network-based adaptive control of nonlinear systems. *IEEE Control Systems Letters*, *6*, 476–481.

Le, D., Patil, O., Nino, C., & Dixon, W. E. (2025). Accelerated gradient approach for deep neural network-based adaptive control of unknown nonlinear systems. *IEEE Transactions on Neural Networks and Learning Systems*, *36*(4), 6299–6313.

Lewis, F. L., Yesildirek, A., & Liu, K. (1996). Multilayer neural-net robot controller with guaranteed tracking performance. *IEEE Transactions on Neural Networks*, *7*(2), 388–399.

Li, Q., Qian, J., Zhu, Z., Bao, X., Helwa, M. K., & Schoellig, A. P. (2017). Deep neural networks for improved, impromptu trajectory tracking of quadrotors. In *Proc. IEEE int. conf. robot. autom.* (pp. 5183–5189).

OConnell, M., Shi, G., Shi, X., Azizzadenesheli, K., Anandkumar, A., Yue, Y., & Chung, S.-J. (2022). Neural-fly enables rapid learning for agile flight in strong winds. *Science Robotics*, *7*(66), 6597.

Paden, B. E., & Sastry, S. S. (1987). A calculus for computing Filippov's differential inclusion with application to the variable structure control of robot manipulators. *IEEE Transactions on Circuits and Systems*, *34*(1), 73–82.

Patil, O., Le, D., Greene, M., & Dixon, W. E. (2022). Lyapunov-derived control and adaptive update laws for inner and outer layer weights of a deep neural network. *IEEE Control Systems Letters*, *6*, 1855–1860.

Patil, O. S., Le, D. M., Griffis, E., & Dixon, W. E. (2022). Deep residual neural network (ResNet)-based adaptive control: A Lyapunov-based approach. In *Proc. IEEE conf. decis. control* (pp. 3487–3492).

Punjani, A., & Abbeel, P. (2015). Deep learning helicopter dynamics models. In *Proc. IEEE int. conf. robot. autom.* (pp. 3223–3230).

Rolnick, D., & Tegmark, M. (2018). The power of deeper networks for expressing natural functions. In *Int. conf. learn. represent.*.

Shi, G., Shi, X., OConnell, M., Yu, R., Azizzadenesheli, K., Anandkumar, A., Yue, Y., & Chung, S.-J. (2019). Neural lander: Stable drone landing control using learned dynamics. In *Proc. IEEE int. conf. robot. autom.* (pp. 9784–9790).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958.

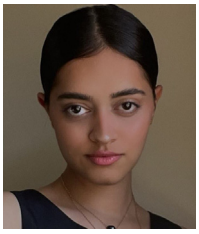
Stepanyan, V., & Kurdila, A. (2009). Asymptotic tracking of uncertain systems with continuous control using adaptive bounding. *IEEE Transactions on Neural Networks*, *20*(8), 1320–1329.

Sun, R., Greene, M., Le, D., Bell, Z., Chowdhary, G., & Dixon, W. E. (2022). Lyapunov-based real-time and iterative adjustment of deep neural networks. *IEEE Control Systems Letters*, *6*, 193–198.

Wei, C., Kakade, S., & Ma, T. (2020). The implicit and explicit regularization effects of dropout. In *Int. conf. on mach. learn.* (pp. 10181–10192). PMLR.

Xian, B., Dawson, D. M., de Queiroz, M. S., & Chen, J. (2004). A continuous asymptotic tracking control strategy for uncertain nonlinear systems. *IEEE Transactions on Automatic Control*, *49*(7), 1206–1211.

Zhou, S., Helwa, M. K., & Schoellig, A. P. (2017). Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking. In *Proc. IEEE conf. decis. cont.* (pp. 5201–5207).



Saiedeh Akbari is a Ph.D. candidate in the department of Mechanical and Aerospace Engineering at University of Florida. Her research focuses on developing adaptive learning-based control strategies for stochastic nonlinear systems. Saiedeh completed her Bachelor of Science in Mechanical Engineering at KN Toosi University of Technology in 2020. During her undergraduate studies, she conducted research on discrete-time sliding mode control for permanent magnet DC motors. She then received her Master of Science in Mechanical Engineering at The University of Alabama in 2023, where she worked on nonlinear adaptive control techniques for hybrid systems, with applications in rehabilitation robotics.



Emily J. Griffis received the B.S. and M.S. degrees in mechanical engineering and the Ph.D. degree from the University of Florida, in May 2020, December 2021, and May 2024, respectively. In 2020, she joined the Nonlinear Controls and Robotics Laboratory, University of Florida, under the supervision of Dr. Warren Dixon to pursue her Ph.D. degree. Her research interests include using adaptive control and deep learning to study Lyapunov-based control of nonlinear and uncertain systems. In 2024, she was awarded the Best Dissertation Award for the Department of Mechanical and Aerospace Engineering.



Omkar Sudhir Patil is a Research Scientist at the University of Florida whose work spans nonlinear control, robotics, and learning-enabled autonomy. His research integrates Lyapunov-based adaptive control, machine learning, and nonlinear dynamics to develop provably stable, safe, and real-time learning methods for uncertain robotic systems. He has pioneered extensions of Lyapunov theory to deep neural networks, enabling online system identification and control with formal guarantees in complex environments. Dr. Patil received his Ph.D. (2023) and M.S. (2022) in Mechanical Engineering from the University of Florida, and his B.Tech. in Production and Industrial Engineering from the Indian Institute of Technology (IIT) Delhi in 2018.



Warren Dixon received his Ph.D. from the Department of Electrical and Computer Engineering from Clemson University. He worked as a research staff member and Eugene P. Wigner Fellow at Oak Ridge National Laboratory (ORNL) for four years until he joined the University of Florida in the Mechanical and Aerospace Engineering Department. His main research interest has been the development and application of Lyapunov-based control techniques for uncertain nonlinear systems. He is an ASME and IEEE Fellow and his collaborative work has received various early and mid-career awards, including various best paper awards.