Proceedings of Machine Learning Research vol vvv:1–12, 2025

CNN-based Adaptive Controller with Stability Guarantees

Myeongseok Ryu DDING_98@GM.GIST.AC.KR and **Kyunghwan Choi** KHCHOI@GIST.AC.KR
School of Mechanical and Robotics Engineering, Gwangju Institute of Science and Technology, 61005 Gwangju, Republic of Korea

Abstract

This study proposes a convolutional neural network (CNN)-based adaptive controller wtih three notable features: 1) it determines control input directly from historical sensor data; 2) it learns the desired control policy during real-time implementation without using a pretrained network; and 3) the asymptotic tracking error convergence is proven during the learning process. An adaptive law for learning the desired control policy is derived using the gradient descent optimization method, and its stability is analyzed based on the Lyapunov approach. A simulation study using a control-affine nonlinear system demonstrated that the proposed controller exhibits these features, and its performance can be tuned by manipulating the design parameters. In addition, it is shown that the proposed controller has a superior tracking performance to that of a deep neural network (DNN)-based adaptive controller.

Keywords: Neuro-adaptive control, convolutional neural network (CNN), Lyapunov approach, asymptotic convergence

1. Introduction

Neural networks (NNs) have been widely used in control applications as a function approximator for adaptive control Esfandiari et al. (2022), state estimation Boo (2009), and so on. These studies provided the ultimate boundedness of tracking or estimation errors based on the Lyapunov-based stability analysis, but were limited to investigating shallow NNs with one hidden layer.

Deep neural networks (DNNs) with multiple hidden layers are more expressive and can provide better performance than shallow NNs. However, due to the difficulty in deriving an adaptation law ensuring stability, designing a stable DNN-based controller is generally considered challenging. There have been a few attempts at overcoming this challenge, including developing Lyapunov-based adaptation laws for controllers based on DNNs or theirs variations Patil et al. (2021); Griffis et al. (2023); Hart et al. (2023). A Lyapunov-based adaptation law for a DNN-based controller was derived in Patil et al. (2021), which ensured asymptotic convergence of tracking error. Griffis et al. (2023) utilized long short-term memory (LSTM), a type of DNN, in designing an adaptive controller and presented the ultimate boundedness of tracking error based on Lyapunov analysis. The Lyapunov-based approach was extended to using physics-informed LSTM for an adaptive controller and therein demonstrated its asymptotic stability Hart et al. (2023).

While DNN-based adaptive control with a stability guarantee has been studied by pioneering works Patil et al. (2021); Griffis et al. (2023); Hart et al. (2023), convolutional neural networks (CNNs), which are well known for their spatial feature capturing ability, have not been as actively investigated in control applications, as they have been in computer vision applications. Nonetheless, motivated by the feature-capturing capability, the use of CNNs as a basis for end-to-end controllers has been studied Rausch et al. (2017); Jhung et al.; Liu and Dong (2017); Nobahari and Seifouripour (2019). These end-to-end controllers determine the control input directly from the sensor data with

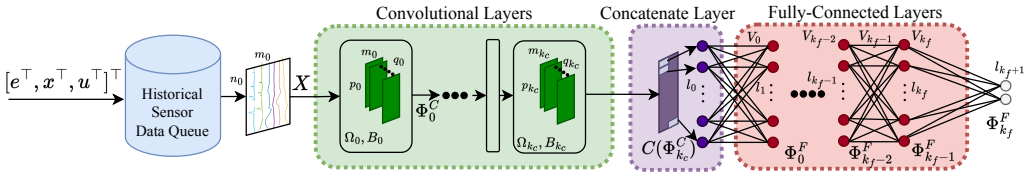

Figure 1: CNN architecture used in the proposed controller.

features that are extracted by CNNs. In Rausch et al. (2017); Jhung et al., CNN-based end-to-end controllers were trained offline to produce the desired steering behavior based on 2D images from the camera. Liu and Dong (2017); Nobahari and Seifouripour (2019) generated pseudo 2D images by stacking historical system input or output data and used them to train CNN-based end-to-end controllers offline to behave as target controllers. However, none of the works above considered online adaptation or stability analysis of CNN-based end-to-end controllers.

This study presents a CNN-based adaptive controller with a stability guarantee. The proposed controller uses 2D images generated by stacking historical sensory data as an input matrix to convolutional layers (CVLs). The output of CVLs is input to the following fully-connected layers (FCLs), which produce the controller output. An adaptation law of network weights are implemented to learn the desired control policy and is derived using the gradient descent optimization method. The stability of the adaptation laws is proven based on the Lyapunov analysis by showing that the tracking error is asymptotically convergent and the network weights and biases are bounded. A part of the proposed controller is designed based on these findings, such as Jacobians with respect to FCLs from Patil et al. (2021) and stabilizing techniques for the adaptation law from Boo (2009); Esfandiari et al. (2022); however, the remainder of the proposed controller, particullary the components rely on the mathematical formulation and adaptation law of the overall CNN architecture, has been solely developed by the present authors. A simulation study using a control-affine nonlinear system is performed to demonstrate that the proposed CNN-based controller ensures asymptotic convergence of tracking error during online adaptation without using a pretrained network. In addition, the tracking performance of the proposed controller is compared with that of the DNN-based adaptive controller, which is a form in which the CVLs are not included in the proposed controller.

2. Problem Formulation

In this section, the following notation is defined and then the problem formulation is presented. $\odot$ and $\otimes$ denote the Hadamard and Kronecker products, respectively Bernstein (2009). $x_{(i)}$ denotes the i^{th} element of vector x, and $x_{(i:j)} \triangleq [x_{(i)}, x_{(i+1)}, \ldots, x_{(j)}]$ where $i \leq j$. For $x \in \mathbb{R}^{pq}$ and $pq = nm$, $\text{reshape}(x, n, m) \triangleq [x_{(1)}, \ldots, x_{(n)}; x_{(n+1)}, \cdots, x_{(2n)}; x_{(2n+1)}, \cdots, x_{(nm)}]^\top$. $A_{(i,j)}$ is the i^{th} row-j^{th} column element of matrix A. $\text{vec}(A) \triangleq [A_{(1,1)}, \ldots, A_{(1,m)}, A_{(2,1)}, \cdots, A_{(n,m)}]^\top$ for $A \in \mathbb{R}^{n\times m}$. $\text{row}_i(A) \triangleq [A_{(i,1)}, A_{(i,2)}, \cdots, A_{(i,m)}]$ denotes the i^{th} row of $A \in \mathbb{R}^{n\times m}$. $\text{row}_{(i:j)}(A) \triangleq [\text{row}_i(A)^\top, \text{row}_{i+1}(A)^\top, \ldots, \text{row}_j(A)^\top]^\top$ denotes a matrix consisting of rows i to j of $A \in \mathbb{R}^{n\times m}$. $\overleftarrow{\prod}_{l=k}^{n} A_l \triangleq \begin{cases} A_n A_{n-1} \cdots A_k, & \text{if } n \geq k \\ 1, & \text{if } n < k \end{cases}$, where k and n are positive integers and A_l is a matrix.

Consider a control-affine nonlinear system modeled as

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

where $x \in \mathbb{R}^n$ denotes the state, $u \in \mathbb{R}^n$ denotes the control input, and $f : \mathbb{R}^n \to \mathbb{R}^n$ denotes an unknown smooth nonlinear function. Note that many mechanical systems can be modeled as Euler-Largrange system and can be transformed into control-affine system with unit control gain matrix Ryu et al. (2013). System (1) can be reformulated as

$$\dot{x} = A_c x + f_c(x) + u,$$

where $f_c(x) = f(x) - A_c x$ and $A_c \in \mathbb{R}^{n\times n}$ is a Hurwitz designer matrix.

The desired control policy u^* is defined by feedback linearization control with the knowledge of function $f(x)$ as follows: $u^* = -f_c(x) + \dot{x}_d - A_c x_d - k_s \text{sgn}(e)$, where $x_d \in \mathbb{R}^n$ is the desired state trajectory, which is continuously differentiable; $e \triangleq x - x_d$ is the tracking error; and $k_s \in \mathbb{R}_{>0}$ is the control gain. The desired control policy is an ideal framework, but its solution unknown due to the use of the unknown function $f(x)$. The goal is to design a controller that can learn this desired control policy online with a guarantee of stability.

3. Controller Design

This section presents a CNN-based adaptive controller. The CNN architecture $\Phi : \mathbb{R}^{n_0\times m_0} \times \mathbb{R}^{p_0\times m_0\times q_0} \times \mathbb{R}^{q_0} \times \cdots \times \mathbb{R}^{p_{k_c}\times m_{k_c}\times q_{k_c}} \times \mathbb{R}^{q_{k_c}} \times \mathbb{R}^{(l_0+1)\times l_1} \times \cdots \times \mathbb{R}^{(l_{k_f}+1)\times l_{k_f+1}} \to \mathbb{R}^{l_{k_f+1}}$ illustrated in Fig. 1 consists of the CVLs, concatenate layer and FCLs.

3.1. Convolutional Layers

The CVLs are represented recursively as

$$\Phi^C_{j_c} = \begin{cases} O(\phi^C_{j_c}(\Phi^C_{j_c-1}), \Omega_{j_c}, B_{j_c}), & j_c \in [1, \ldots, k_c] \\ O(\phi^C_0(X), \Omega_0, B_0), & j_c = 0 \end{cases},$$

where $X \in \mathbb{R}^{n_0\times m_0}$ denotes the network input matrix, $O : \mathbb{R}^{n_{j_c}\times m_{j_c}} \times \mathbb{R}^{p_{j_c}\times m_{j_c}\times q_{j_c}} \times \mathbb{R}^{q_{j_c}} \to \mathbb{R}^{n_{j_c+1}\times m_{j_c+1}}$ denotes the CNN operator (see Appendix A), and $\phi^C_{j_c} : \mathbb{R}^{n_{j_c}\times m_{j_c}} \to \mathbb{R}^{n_{j_c}\times m_{j_c}}$ denotes the matrix activation function (i.e. $\phi^C_{j_c}(\Phi^C_{j_c-1})_{(i,j)} = \sigma_{j_c}(\Phi^C_{j_c-1_{(i,j)}})$) for some activation functions $\sigma_{j_c} : \mathbb{R} \to \mathbb{R}$. The first activation function ϕ^C_0 should be a bounded nonlinear function to ensure that the input to the first CNN operator is bounded. In this study, $\alpha_1 \tanh(\cdot)$ is selected with $\alpha_1 \in \mathbb{R}_{>0}$. The filter set Ω_{j_c} contains q_{j_c} filters $W^{(i)}_{j_c} \in \mathbb{R}^{p_{j_c}\times m_{j_c}}$, $\forall i \in [1, \ldots, q_{j_c}]$ and is represented as $\Omega_{j_c} = \{W^{(1)}_{j_c}, \ldots, W^{(q_{j_c})}_{j_c}\}$, where superscript i denotes the filter index. The bias vector B_{j_c} consists of q_{j_c} biases.

3.2. Fully-Connected Layers

The output matrix of the CVLs is input to the concatenate layer $C(\Phi^C_{k_c}) = [\text{vec}(\Phi^{C\top}_{k_c})^\top, 1]^\top$ before the input layer of FCLs for compatibility between the CVLs and FCLs.

The FCLs are represented recursively as $\Phi^F_{j_f} = \begin{cases} V_{j_f}^\top \phi^F_{j_f}(\Phi^F_{j_f-1}), & j_f \in [1,\dots,k_f] \\ V_0^\top C(\Phi^C_{k_c}), & j_f = 0 \end{cases}$, where $V_{j_f} \in \mathbb{R}^{l_{j_f}+1 \times l_{j_f+1}}$ denotes the weight matrix, $\phi^F_{j_f} : \mathbb{R}^{l_{j_f}} \to \mathbb{R}^{l_{j_f+1}}$ denotes the vector activation function defined by $\phi^F_{j_f} \triangleq \phi^F_{j_f}(\Phi^F_{j_f-1}) = [\sigma_{j_f}(\Phi^F_{j_f-1_{(1)}}), \sigma_{j_f}(\Phi^F_{j_f-1_{(2)}}) \cdots, \sigma_{j_f}(\Phi^F_{j_f-1_{(l_{j_f})}}), 1]^\top$ for some nonlinear activation functions $\sigma_{j_f} : \mathbb{R} \to \mathbb{R}$ such as $\tanh(\cdot)$. Note that $C(\Phi^C_{k_c})$ and $\phi^F_{j_f}$ are augmented by 1 to consider the bias of the preceding input layer as a weight in the weight matrix. Hereafter, weights refer to the trainable variables in filters, bias vectors, and weight matrices for simplicity. The output of the FCLs is the final output of the CNN architecture and is represented as $\Phi(X, \Omega_0, B_0, \dots, \Omega_{k_c}, B_{k_c}, V_0, \cdots, V_{k_f}) \triangleq \Phi^F_{k_f}$.

3.3. Control Law Development

In the controller, the CNN architecture is designed to approximate the lumped term $\Lambda \triangleq f_c(x) - \dot{x}_d + A_c x_d$ in the desired control policy u^*. According to the universal approximation theorem Zhou (2020), there exist ideal bounded network weights $\Omega^*_{j_c}, B^*_{j_c}, \forall j_c \in [0,\dots,k_c]$ and $V^*_{j_f}, \forall j_f \in [0,\dots,k_f]$, such that $||\Lambda - \Phi^*|| \leq \bar{\epsilon}$ where $\Phi^* \triangleq \Phi(X, \Omega^*_0, B^*_0, V^*_0, \dots, \Omega^*_{k_c}, B^*_{k_c}, V^*_{k_f})$ for a positive constant $\bar{\epsilon}$. In other words, there exist some positive constants $\bar{\Omega}$, $\bar{B}$ and $\bar{V}$, such that $||\Omega^*_{j_c}|| \leq \bar{\Omega}$, $||B^*_{j_c}|| \leq \bar{B}$ and $||V^*_{j_f}|| \leq \bar{V}$ for all $j_c \in [0,\dots,k_c]$, $j_f \in [0,\dots,k_f]$.

Then, the lumped term Λ can be approximated as follows:

$$u^* = -\Phi^* - \epsilon - k_s \text{sgn}(e),$$

where ϵ is the approximation error such that $||\epsilon|| \leq \bar{\epsilon}$. The CNN architecture adapts its weights to converge to the ideal values while used in the control input as

$$u = -\hat{\Phi} - k_s \text{sgn}(e), \tag{2}$$

where $\hat{\Phi} \triangleq \Phi(X, \hat{\Omega}_0, \hat{B}_0, \hat{V}_0, \dots, \hat{\Omega}_{k_c}, \hat{B}_{k_c}, \hat{V}_{k_c})$ denotes the CNN architecture with weights being adapted. The control input u results in the following error dynamics:

$$\dot{e} = A_c e - k_s \text{sgn}(e) + \tilde{\Phi} + \epsilon, \tag{3}$$

where $\tilde{\Phi} \triangleq \Phi^* - \hat{\Phi}$.

3.4. Network Input Matrix Design

Any 2D image data can be used as the input matrix of this CNN architecture if the data include sufficient information for estimating the system dynamics. For instance, concatenated camera images and time-stacked system input and output data constitute sufficient datasets. This study adopts the latter, and the resulting input matrix is defined as

$$X(t) \triangleq [\xi(t), \xi(t - T_s), \cdots, \xi(t - (n_0 - 1)T_s)]^\top \in \mathbb{R}^{n_0 \times m_0},$$

where $\xi = \alpha_2[e^\top, x^\top, u^\top]^\top \in \mathbb{R}^{2n+m}$, α_2 denotes a positive constant, $T_s \in \mathbb{R}_{>0}$ denotes the data stacking time, and n_0 denotes the number of stacks. Parameter α_2 prevents X from being saturated or distorted by the activation functions.

4. Weight Adaptation Law

Jacobians with respect to the CVLs and FCLs are derived first, followed by a derivation of the weight adaptation law based on the gradient descent optimization method using the Jacobians.

4.1. Jacobians with respect to Convolutional Layers

Let Φ_i denote the i^{th} output of Φ. Then, the Jacobians of Φ_i with respect to the weights of CVLs are derived as

$$\begin{aligned}\frac{\partial \Phi_i}{\partial W_{j_c}^{l_k}} &= \sum_{l_i=1}^{n_{j_c+1}} \left(\frac{\partial \Phi_i}{\partial \Phi_{j_c}^C(l_i,l_k)} \text{row}_{(l_i:l_i+p_{j_c}-1)}(\phi_{j_c}^C) \right), \quad \text{with } j_c \in [0,\dots,k_c], \\ \frac{\partial \Phi_i}{\partial B_{j_c}(l_k)} &= \sum_{l_i=1}^{n_{j_c+1}} \left(\frac{\partial \Phi_i}{\partial \Phi_{j_c}^C(l_i,l_k)} \cdot 1 \right), \qquad\qquad \text{with } j_c \in [0,\dots,k_c],\end{aligned} \tag{4}$$

for all $l_k \in [1,\cdots,q_{j_c}]$, $i \in [1,\dots,l_{k_f+1}]$, where $\phi_{j_c}^C \triangleq \phi_{j_c}^C(\Phi_{j_c-1}^C)$ for $j_c \in [1,\dots,k_c]$, $\phi_0^C \triangleq \phi_0^C(X)$, and $\partial\Phi_i/\partial\Phi_{j_c}^C$ denotes the backpropagated gradient of Φ_i with respect to $\Phi_{j_c}^C$, which is obtained using the back-propagation method. The details of the Jacobian calculation are provided in Appendix B.

4.2. Jacobians with respect to Fully-connected Layers

The Jacobians of Φ with respect to the weights of FCLs were derived in Patil et al. (2021) as

$$\begin{aligned}\frac{\partial \Phi}{\partial \text{vec}(V_0)} &= (\overset{\curvearrowleft}{\prod}_{l=1}^{k_f} V_l^\top \phi_l^{F\prime})(I_{l_1} \otimes C(\Phi_{k_c}^C)), \qquad\qquad \text{with } j_f = 0, \\ \frac{\partial \Phi}{\partial \text{vec}(V_{j_f})} &= (\overset{\curvearrowleft}{\prod}_{l=j_F+1}^{k_f} V_l^\top \phi_l^{F\prime})(I_{l_{j_f+1}} \otimes \phi_{j_f}^{F\,\top}), \quad \text{with } j_f \in [1,\dots,k_f],\end{aligned}$$

where $\phi_{j_f}^{F\,\prime} \triangleq \frac{\partial}{\partial x}\phi_{j_f}^F(x)$ is the Jacobian of the activation function with respect to some vector x.

4.3. Derivation of Adaptation Laws

For simplicity, the weights of the FCLs and CVLs are expressed as column vectors: $\theta_F \triangleq [\text{vec}(V_0); \cdots; \text{vec}(V_{k_f})]$ and $\theta_C \triangleq [\text{vec}(W_0^{(1)}); \text{vec}(W_0^{(2)}); \cdots; \text{vec}(W_{k_c}^{(q_{k_c})}); B_0; \cdots; B_{k_c}]$, respectively. Define the weight vector including all the weights as $\theta \triangleq [\theta_F; \theta_C] \in \mathbb{R}^\Xi$, where $\Xi \triangleq \sum_{j_f=0}^{k_f}(l_{j_f}+1)l_{j_f+1} + \sum_{j_c=0}^{k_c}(p_{j_c}m_{j_c}+1)q_{j_c}$ denotes the length of θ. Then, the Jacobian of Φ with respect to the weight vector is represented as

$$\Phi' \triangleq \frac{\partial \Phi}{\partial \theta} = \begin{bmatrix} \frac{\partial \Phi}{\partial \theta_F} & \frac{\partial \Phi}{\partial \theta_C} \end{bmatrix} \in \mathbb{R}^{l_{k_f+1} \times \Xi},$$

where $\partial\Phi/\partial\theta_F = \left[\frac{\partial \Phi}{\partial \text{vec}(V_0)}, \cdots, \frac{\partial \Phi}{\partial \text{vec}(V_{k_f})}\right]$ and $\partial\Phi/\partial\theta_C = [\frac{\partial \Phi_1}{\partial \theta_C}^\top, \cdots, \frac{\partial \Phi_{l_{k_f+1}}}{\partial \theta_C}^\top]^\top$.

Consider a convex objective function $J = \frac{1}{2}e^\top e$, which is defined to minimize the tracking error. The weight adaptation law is derived by employing the gradient descent optimization

method to minimize J. The gradient of J with respect to the $\hat{\theta}$ is $\partial J/\partial\hat{\theta} = [(\partial J/\partial e)(\partial e/\partial\hat{\theta})]^\top = [e^\top(\partial e/\partial\hat{\theta})]^\top$, where $\hat{\theta} \triangleq [\hat{\theta}_F; \hat{\theta}_C]$ denotes the estimate of θ^*. By applying a static approximation (i.e., $\partial\dot{e}/\partial\hat{\theta} = 0$) to (3) and calculating the partial deriavte with respect to $\hat{\theta}$, the term $\partial e/\partial\hat{\theta}$ can be calculated as $\partial e/\partial\hat{\theta} = A_c^{-1}\hat{\Phi}'$, where $\hat{\Phi}' \triangleq \partial\hat{\Phi}/\partial\hat{\theta}$ (see Esfandiari et al. (2022), Boo (2009)). Finally, the weight adaptation law is proposed as

$$\dot{\hat{\theta}} = \text{proj}[-\Gamma(A_c^{-1}\hat{\Phi}')^\top e - \rho||e||\hat{\theta}], \tag{5}$$

where $\Gamma \in \mathbb{R}^{\Xi\times\Xi}$ denotes the learning rate matrix which is symmetric and positive-definite and $\rho \in \mathbb{R}_{>0}$ is the damping factor which corresponds to the e-modification term presented in Esfandiari et al. (2022), Boo (2009).

The estimated weight vector $\hat{\theta}$ remains in set $\Theta_\theta = \{\theta \mid ||\theta|| \leq \bar{\theta}\}$ (i.e., $\sup_{\hat{\theta}\in\Theta_\theta}||\hat{\theta}|| \leq \bar{\theta} \in L_\infty$) by the projection operator (Appendix E, Eq. (E.4) in Krstic et al. (1995)). As mentioned in Section 3.3, the ideal weights are bounded, it follows that ideal weight vector $\theta^* \triangleq [\theta_F^*; \theta_C^*]$ is also bounded by a positive constant $\bar{\theta}$ such that $||\theta^*|| \leq \bar{\theta}$. Therefore, weights estimate error $\tilde{\theta} \triangleq \theta^* - \hat{\theta}$ is also bounded.

Moreover, there exists a positive constant Φ'_M such that $||\hat{\Phi}'|| \leq \Phi'_M$, because the activation functions and their gradients are bounded for some bounded inputs, the input matrix X and the weight estimates are bounded.

5. Stability Analysis

The estimation error of the CNN architecture is expressed as follows using the first-order Taylor series approximation:

$$\tilde{\Phi} = \Phi(X, \theta^*) - \Phi(X, \hat{\theta}) = \hat{\Phi}'\tilde{\theta} + \mathcal{O}(||\tilde{\theta}||^2), \tag{6}$$

where $\mathcal{O}(\cdot)$ denotes a bounded higher-order error. This exapansion is reasonable since Φ is combination of affine operators and $\tanh(\cdot)$ which are smooth function (i.e., their high-order derivatives are bounded.). The error dynamics (3) are reexpressed as follows using the relationship (6):

$$\dot{e} = A_c e - k_s \text{sgn}(e) + \hat{\Phi}'\tilde{\theta} + \Delta,$$

where $\Delta \triangleq \epsilon + \mathcal{O}(||\tilde{\theta}||^2) \leq \bar{\Delta} \in L_\infty$ denotes a lumped error. The following theorem establishes the asymptotic tracking error convergence of the proposed controller.

Theorem 1 *For the dynamical system in* (1)*, the proposed controller in* (2) *and adaptation law* (5) *ensure asymptotic tracking error convergence, in the sense that* $e \to 0$ *as* $t \to \infty$*, provided that* $\beta_1\beta_2^2 + \bar{\Delta} \leq k_s$ *where* $\beta_1 = \rho||\Gamma^{-1}||$ *and* $\beta_2 = \{\Phi'_M(||A_c^{-1}|| + 1) + \beta_1\bar{\theta}\}/2\beta_1$.

Proof Let $V : \mathbb{R}^{n+\Xi} \to \mathbb{R}_{>0}$ denote the candidate Lyapunov function defined as $V = (1/2)e^\top e + (1/2)\tilde{\theta}^\top\Gamma^{-1}\tilde{\theta}$. Using $\tilde{x}^T\hat{x} \leq ||\tilde{x}||(\bar{x} - ||\tilde{x}||)$, $x^T\text{sgn}(x) = ||x||_1 \geq ||x||$ for $x \in \mathbb{R}^n$, *Lemma E.1* in

Krstic et al. (1995), and $\dot{\theta}^* = 0$, the time-derivative of V is derived as

$$\begin{aligned}\dot{V} =& e^\top A_c e + e^\top \Big(- k_s \text{sgn}(e) + \hat{\Phi}'\tilde{\theta} + \Delta \Big) - \tilde{\theta}^\top \Gamma^{-1} \text{proj}\Big[- \Gamma \hat{\Phi}'^\top {A_c^{-1}}^\top e - \rho ||e|| \hat{\theta} \Big] \\ \leq& e^\top A_c e - k_s ||e|| + ||e|| (\Phi'_M ||\tilde{\theta}|| + \bar{\Delta}) + \Phi'_M ||\tilde{\theta}|| ||A_c^{-1}|| ||e|| + \rho ||\Gamma^{-1}|| ||\tilde{\theta}|| ||e|| (\bar{\theta} - ||\tilde{\theta}||) \\ \leq& e^\top A_c e + ||e|| \Big\{ - k_s + \bar{\Delta} + ||\tilde{\theta}|| \Big(\Phi'_M (||A^{-1}|| + 1) + \rho ||\Gamma^{-1}|| (\bar{\theta} - ||\tilde{\theta}||) \Big) \Big\} \\ \leq& e^\top A_c e + ||e|| \Big\{ - \beta_1 (||\tilde{\theta}|| - \beta_2)^2 + \beta_1 \beta_2^2 + \bar{\Delta} - k_s \Big\},\end{aligned} \tag{7}$$

where $\beta_1 \triangleq \rho ||\Gamma^{-1}||$ and $\beta_2 \triangleq \{\Phi'_M (||A^{-1}|| + 1) + \beta_1 \bar{\theta}\}/2\beta_1$ are positive constants. By selecting k_s such that $\beta_1 \beta_2^2 + \bar{\Delta} \leq k_s$, (7) yields

$$\dot{V} \leq e^\top A_c e - \beta_1 ||e|| (||\tilde{\theta}|| - \beta_2)^2 \leq e^\top A_c e \triangleq -W(e). \tag{8}$$

Inequality (8) implies that $\int_{t_0}^{t} W(e(\tau))\, d\tau$ is finite. By Barbalat's Lemma Khalil (2002), $W(e) \to 0$ as $t \to \infty$. Therefore, $e \to 0$ as $t \to \infty$. ■

Remark 2 *The design parameters n_0 and T_s of the input matrix X play a crucial role in determining the resolution and size of the information provided to the CVLs. The smaller T_s is, the higher the resolution becomes, while also limiting the temporal range of the information. As n_0 increases, the input matrix possesses information for a longer time, which also results in higher computational costs.*

Remark 3 *As mentioned in Remark 3.3 of Esfandiari et al. (2022), the stability and convergence speed in the learning phase are affected by design parameters Γ, ρ, and A_c. Increasing the learning rate Γ increases the convergence speed, but some oscillations can occur in the transient response. The damping factor ρ can help the learning phase to be robust to NN approximation errors Ge et al. (2013), but the weights may converge far from the optimal weights. The further the A_c's eigenvalues are from the imaginary axis in the negative direction, the faster the tracking error dynamics are, but the slower the weights are updated due to the use of A_c^{-1} in the adaptation law (5).*

6. Simulations

Comparative simulations were performed to demonstrate the efficacy of the proposed CNN-based adaptive controller and analyze its properties. An example for system (1) was employed from Patil et al. (2021) as $\dot{x} = f(x) + u$ with

$$f(x) = \begin{bmatrix} x_1 x_2 \tanh(x_2) + \text{sech}(x_1) \\ \text{sech}^2(x_1 + x_2) - \text{sech}^2(x_2) \end{bmatrix}, \tag{9}$$

where $x = [x_1, x_2]^\top$, $u = [u_1, u_2]^\top$. The initial state value was $x(0) = [1, 2]^\top$, and the desired trajectory was $x_d(t) = [\sin(2t), -\cos(t)]^\top$.

Six controllers are compared: the proposed controller with default parameters (CNN1), four variations thereof (CNN2 to CNN5), and the comparison controller (DNN). CNN1 had the following

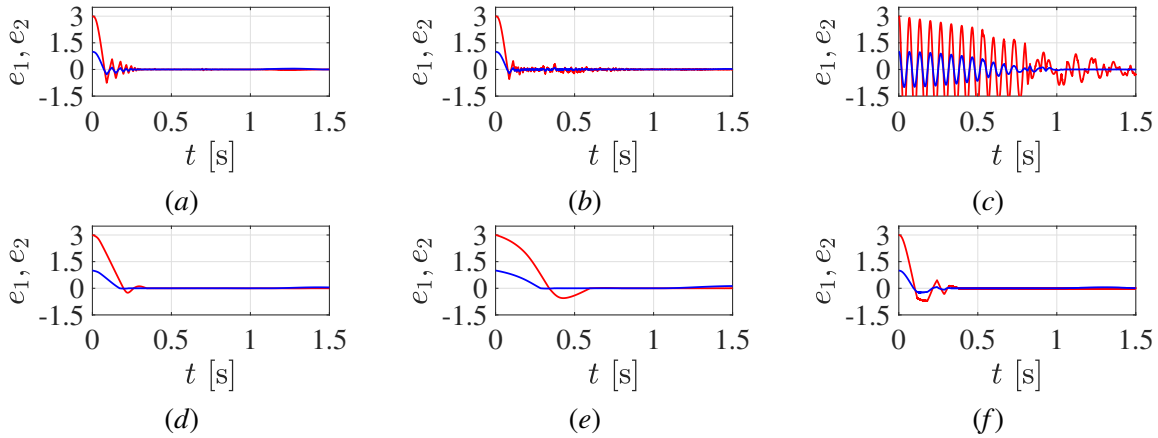

Figure 2: Tracking errors of (a) CNN1, (b) CNN2, (c) CNN3, (d) CNN4, (e) CNN5, and (f) DNN. The blue and red solid lines represent e_1 and e_2, respectively.

control gain and design parameters: $k_s = 1, \rho = 10$, and $A_c = -10I$, where $I = [1, 0; 0, 1]$. The CNN architecture of CNN1 was designed as follows: $k_f = 2, k_c = 1, T_s = 0.1, n_0 = 10$, $\alpha_1 = 100$, $\alpha_2 = 0.01$; each FCL had 8 nodes; the first CVL had 2 filters whose dimensions were (5×6); the second layer had 2 filters whose dimensions were (3×2); the input matrix X had the dimensions of (10×6); all trainable variables were initialized from a uniform distribution in the interval $(-0.1, 0.1)$ and not pretrained; and $\tanh(\cdot)$ was selected as the FCLs and CVLs's activation function.

CNN2 used a smaller stacking time value (i.e. $T_s = 0.01$) compared to CNN1; CNN3 replaced the adaptation law of CNN1 with one from Patil et al. (2021) (i.e., $A_c = -I, \rho = 0$); CNN4 used a larger damping factor value (i.e. $\rho = 50$) compared to CNN1; CNN5 employed eigenvalues that are more negative for A_c (i.e. $A_c = -50I$) compared to those used by CNN1; and DNN was defined by excluding the CVLs from the proposed CNN architecture. DNN had 4 FCLs with $8, 8, 8$ and 4 nodes using $\alpha_1 \tanh(\xi(t)/\alpha_2)$ as the input vector. The other parameters were the same as CNN1. The numbers of nodes in layers were chosen as described above so that the total number of weights of DNN (250) was almost similar to those of CNN1 (244).

The simulation results are presented in Fig. 2, with the tracking errors compared in Table 1, where $e_{1(2)}$ and $\epsilon_{1(2)}$ denote the tracking error and root mean square error (RMSE) of $x_{1(2)}$, respectively. Notably, CNN1, CNN2, and CNN4 showed almost asymptotic convergence, which confirmed the main finding of this study. A detailed comparison is performed from two perspectives in the subsections below.

6.1. Effects of Design Parameters

- Stacking time T_s: CNN2, with a smaller T_s, regulated the tracking errors faster than did CNN1 but provided noisy responses. This is because a small value of T_s provides fine but shortsighted dynamic information, as mentioned in Remark 2.

- Desinger matrix A_c and damping factor ρ: CNN3, with a simpler adaptation law ($A_c = -I$ and $\rho = 0$), showed slow and oscillating convergence. In contrast, other CNNs using more

Table 1: Quantitative Comparison of Tracking RMSE

	CNN1 (Default)	**CNN2** (Smaller T_s)	**CNN3** (Existing)	**CNN4** (Larger ρ)	**CNN5** (More negative A_c)	**DNN** (No CVLs)
ϵ_1	0.0397	0.0384	0.2160	0.0716	0.1291	0.0490
ϵ_2	0.3752	0.3680	2.4030	0.7524	1.5446	0.4757

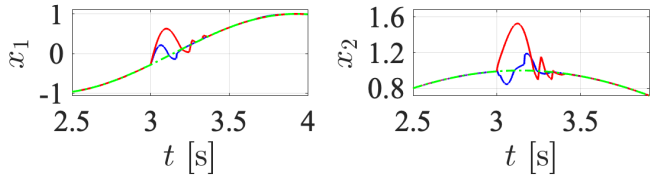

Figure 3: Control results of CNN1 (blue solid line) and DNN (red solid line) under a sudden change in the system at 3 s. The green dash-dotted line denotes the desired trajectory.

negative A_c and nonzero ρ showed faster convergence than did CNN3 and almost no oscillating convergence. The larger ρ was, the slower but more stable the convergence was (see CNN4 vs. CNN1 and CNN2). However, a more negative A_c did not guarantee faster weight's convergence (see CNN5), as mentioned in Remark 3.

6.2. CNN vs. DNN

The strength of the proposed CNN-based controller is derived from utilizing image data containing system historical information, which is useful for approximating a dynamic function of $f_c(x) - \dot{x}_d + A_c x_d$ in u^*. However, the DNN-based controller uses only the current observation and is considered a state feedback controller that has a limited capability to approximate the dynamical function.

The difference in performance between DNN and CNN1 was not significant in the simulation results provided in Fig. 2 and Table 1, although DNN had slightly larger errors than did CNN1. However, the difference can be highlighted by a case study where the system changed suddenly at $t = 3$ from (9) to $\dot{x} = f(x) + g(x) + u$, with

$$g(x) = \begin{bmatrix} 2x_1^2 x_2 + 2\sin(t) + 20 \\ 2x_2^2 \tanh(x_1) + 2\cos(\frac{1}{2}t) + 20 \end{bmatrix}.$$

The case study results, provided in Fig. 3, demonstrated that CNN1 learned the new desired control policy (which depended on the time-dependent functions) faster than DNN did.

7. Conclusion and Future Work

This study presents a CNN-based adaptive controller for tracking control of uncertain control-affine nonlinear systems. The main contributions are threefold: 1) to present an analytical expression of the CNN architecture, 2) to derive an adaptation law based on gradient descent optimization for the CNN architecture, and 3) to prove the stability of the adaptation law, based on Lyapunov analysis.

A simulation study demonstrated that the proposed controller learned the desired control policy and provided asymptotic convergence without using pretrained data. Future work would be development of other CNN variations-based controller with stability guarantee.

Appendix A. CNN Operator

The CNN operator $O(\cdot)$ can be represented as

$$O(X,\Omega,B)_{(i,j)} = 1_{1\times p}(W^{(j)} \odot \text{row}_{(i:i+p-1)}(X))1_{m\times 1} + B_{(j)},$$

where $X \in \mathbb{R}^{n\times m}$, $\Omega \in \mathbb{R}^{p\times m\times q}$, and $B \in \mathbb{R}^q$ denote the input matrix, filter set and bias vector, respectively. The filter set $\Omega = \{W^{(1)},\dots,W^{(q)}\}$ has q filters $W^{(i)} \in \mathbb{R}^{p\times m}$, $\forall i \in [1,\dots,q]$. Note that the operator's output is also bounded if X, Ω, and B are bounded.

Appendix B. Jacobians with respect to Convolutional Layers

The Jacobian of Φ with respect to the concatenate layer is represented as

$$\frac{\partial \Phi}{\partial C(\Phi^C_{k_c})} = (\overset{\curvearrowleft}{\prod}_{l=1}^{k_f} \hat{V}_l^\top \hat{\phi}_l^{F\prime}) V_0^\top.$$

The Jacobian of Φ_i with respect to the CVLs' output is

$$\frac{\partial \Phi_i}{\partial \Phi^C_{k_c}} = \text{reshape}\left(\frac{\partial \Phi_i}{\partial C(\Phi^C_{k_c})_{(1:n_{k_c}m_{k_c})}}, n_{k_c}, m_{k_c}\right).$$

The Jacobians of Φ_i with respect to the $j_c{}^{\text{th}}$ activation function of the CVLs can be represented as

$$\begin{aligned}\frac{\partial \Phi_i}{\partial \phi^C_{j_c}} =& \sum_{l_i=1}^{n_{j_c+1}}\sum_{l_j=1}^{m_{j_c+1}}\left(\frac{\partial \Phi_i}{\partial \Phi^C_{j_c(l_i,l_j)}}\frac{\partial \Phi^C_{j_c(l_i,l_j)}}{\partial \phi^C_{j_c}}\right)\\ =&\frac{\partial \Phi_i}{\partial \Phi^C_{j_c(1,1)}}\begin{bmatrix} W^{(1)}_{j_c} \\ 0_{(n_{j_c}-p_{j_c})\times m_{j_c}}\end{bmatrix} + \frac{\partial \Phi_i}{\partial \Phi^C_{j_c(2,1)}}\begin{bmatrix} 0_{(n_{j_c}-p_{j_c})\times 1} & W^{(1)}_{j_c} \\ 0_{(n_{j_c}-p_{j_c})\times (m_{j_c}-1)}\end{bmatrix}\\ &+\cdots+\frac{\partial \Phi_i}{\partial \Phi^C_{j_c(n_{j_c+1},m_{j_c+1})}}\begin{bmatrix} 0_{(n_{j_c}-p_{j_c})\times m_{j_c}} \\ W^{(q_{j_c})}_{j_c}\end{bmatrix}\end{aligned}$$

for all $j_c \in [1,\dots,k_c]$, where $\partial\Phi_i/\partial\Phi^C_{j_c}$ is the Jacobians of Φ_i with respect to $\Phi^C_{j_c}$ and represented as

$$\frac{\partial \Phi_i}{\partial \Phi^C_{j_c}} = \frac{\partial \Phi_i}{\partial \phi^C_{j_c+1}} \odot {\phi^C_{j_c+1}}'(\Phi^C_{j_c})$$

for all $j_c \in [0,\dots,k_c]$. Finally, the Jacobians of Φ_i with respect to the $l_k{}^{\text{th}}$ filters and bias vectors of the j^{th} CVL can be obtained as (4).

References

Neural network-based state estimation of nonlinear systems: application to fault detection and isolation, volume 395. Springer, 2009. ISBN 1441914382.

Dennis S Bernstein. *Matrix mathematics: theory, facts, and formulas*. Princeton university press, 2009. ISBN 1400833345.

Kasra Esfandiari, Farzaneh Abdollahi, and Heidar A Talebi. *Neural network-based adaptive control of uncertain nonlinear systems*. Springer, 2022. ISBN 3030731359.

Shuzhi Sam Ge, Chang C Hang, Tong H Lee, and Tao Zhang. *Stable adaptive neural network control*, volume 13. Springer Science & Business Media, 2013. ISBN 1475765770.

Emily J Griffis, Omkar Sudhir Patil, Zachary I Bell, and Warren E Dixon. Lyapunov-based long short-term memory (Lb-LSTM) neural network-based control. *IEEE Control Systems Letters*, 2023. ISSN 2475-1456.

Rebecca G Hart, Emily J Griffis, Omkar Sudhir Patil, and Warren E Dixon. Lyapunov-based physics-informed long short-term memory (LSTM) neural network-based adaptive control. *IEEE Control Systems Letters*, 2023. ISSN 2475-1456.

Junekyo Jhung, Il Bae, Jaeyoung Moon, Taewoo Kim, Jincheol Kim, and Shiho Kim. End-to-end steering controller with cnn-based closed-loop feedback for autonomous vehicles. In *2018 IEEE intelligent vehicles symposium (IV)*, pages 617–622. IEEE. ISBN 1538644525.

Hassan K Khalil. *Nonlinear systems; 3rd ed.* Prentice-Hall, Upper Saddle River, NJ, 2002. URL https://cds.cern.ch/record/1173048. The book can be consulted by contacting: PH-AID: Wallet, Lionel.

Miroslav Krstic, Petar V Kokotovic, and Ioannis Kanellakopoulos. *Nonlinear and adaptive control design*. John Wiley & Sons, Inc., 1995. ISBN 0471127329.

Xiangdi Liu and Yunlong Dong. A convolutional neural networks approach to devise controller. In *MATEC Web of Conferences*, volume 139, page 00168. EDP Sciences, 2017. ISBN 2261-236X.

Hadi Nobahari and Yousef Seifouripour. A nonlinear controller based on the convolutional neural networks. In *2019 7th International Conference on Robotics and Mechatronics (ICRoM)*, pages 362–367. IEEE, 2019. ISBN 1728166047.

Omkar Sudhir Patil, Duc M Le, Max L Greene, and Warren E Dixon. 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, 2021. ISSN 2475-1456.

Viktor Rausch, Andreas Hansen, Eugen Solowjow, Chang Liu, Edwin Kreuzer, and J Karl Hedrick. Learning a deep neural net policy for end-to-end control of autonomous vehicles. In *2017 American Control Conference (ACC)*, pages 4914–4919. IEEE, 2017. ISBN 150905992X.

Myeongseok Ryu, Donghwa Hong, and Kyunghwan Choi. Constrained optimization-based neuro-adaptive control (conac) for uncertain euler-lagrange systems under weight and input constraints. 2013.

Ding-Xuan Zhou. Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2):787–794, 2020. ISSN 1063-5203.