Recent Advancements in Optimal and Deep Learning

Wanjiku A. Makumi, Omkar Sudhir Patil, Rebecca G. Hart, Emily J. Griffis, and Warren E. Dixon

UF UNIVERSITY of FLORIDA · Duke UNIVERSITY · TEXAS The University of Texas at Austin · UC SANTA CRUZ

Deep Nonlinear Adaptive Control for Unmanned Aerial Systems Operating under Dynamic Uncertainties

Zachary Lamb, Zachary I. Bell, Matthew Longmire,
Prashant Ganesh, Ricardo Sanfelice

Video Credit: Jared Paquet

UF Research & Engineering Education Facility
UNIVERSITY of FLORIDA

UC SANTA CRUZ

UF UNIVERSITY of FLORIDA

Duke UNIVERSITY

TEXAS The University of Texas at Austin

UC SANTA CRUZ

Lyapunov-Based Deep Neural Network-Based Value Function Approximation for Approximate Dynamic Programming

W. A. Makumi, M. L. Greene., Z. I. Bell, W. E. Dixon, “Lyapunov-Based Deep Neural Network-Based Value Function Approximation for Approximate Dynamic Programming,” *In preparation*.

Optimal Control Problem Formulation

- Dynamic system

$$\dot{x} = f(x) + g(x)u \quad \longrightarrow \quad \dot{\zeta} = F(\zeta) + G(\zeta)\mu$$

$$\zeta = \left[e^T, x_d^T\right]^T \qquad \mu = u - u_d(x_d)$$

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

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

- Control objective
  - Find a controller, μ, which minimizes the cost function
- Cost function

$$J(\zeta, \mu) = \int_0^\infty \left(Q(\zeta(\tau)) + \mu(\tau)^T R \mu(\tau)\right) d\tau$$

- Hamilton Jacobi Bellman (HJB) equation

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

R. Kamalapurkar, H. Dinh, S. Bhasin, W. E. Dixon, "Approximate Optimal Trajectory Tracking for Continuous-Time Nonlinear Systems," *Automatica*, Vol. 51, pp. 40-48 (2015).

Optimal Control Problem Formulation

- Dynamic system

$$\dot{x} = f(x) + g(x)u \longrightarrow \dot{\zeta} = F(\zeta) + G(\zeta)\mu$$

$$\zeta = \left[e^T, x_d^T\right]^T \qquad \mu = u - u_d(x_d)$$

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

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

- Control objective
  - Find a controller, μ, which minimizes the cost function
- Cost function

$$J(\zeta,\mu) = \int_0^\infty \Big(Q\big(\zeta(\tau)\big) + \mu(\tau)^T R\mu(\tau)\Big)\, d\tau$$

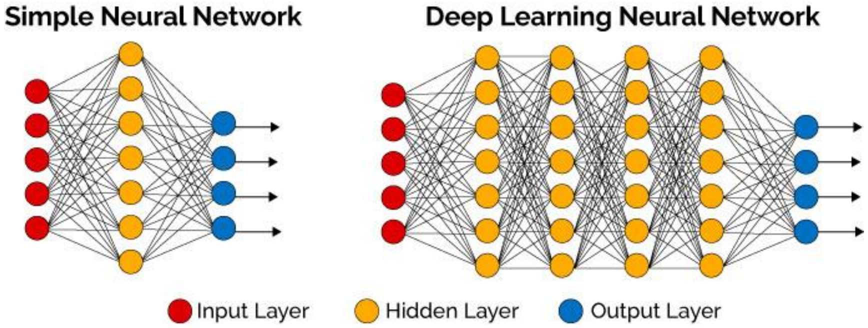
- Hamilton Jacobi Bellman (HJB) equation

$$0 = Q(\zeta) + \mu^*(\zeta)^T R\mu^*(\zeta) + \nabla_\zeta V^*(\zeta)\big(F(\zeta) + G(\zeta)\mu^*(\zeta)\big)$$

R. Kamalapurkar, H. Dinh, S. Bhasin, W. E. Dixon, "Approximate Optimal Trajectory Tracking for Continuous-Time Nonlinear Systems," *Automatica*, Vol. 51, pp. 40-48 (2015).

DNN System Identification

- Previously: Linear parameterization & single-layer NNs
- Lyapunov-based multi-timescale deep neural network (DNN) system identifier
- DNN drift dynamics $f(x) = \theta^T \phi(\Phi(x)) + \epsilon_\theta(x)$
- Drift dynamics approximation $\hat{f}_i(x) = \hat{\theta}^T \phi\left(\widehat{\Phi}_i(x)\right)$



R. Sun, M. Greene, D. Le, Z. Bell, G. Chowdhary, and W. E. Dixon, "Lyapunov-Based Real-Time and Iterative Adjustment of Deep Neural Networks," *IEEE Control Systems Letters*, Vol. 6, pp. 193-198 (2022).

M. Greene, Z. Bell, S. Nivison, and W. E. Dixon, "Deep Neural Network-based Approximate Optimal Tracking for Unknown Nonlinear Systems," *IEEE Transactions on Automatic Control*, to appear.

Multi-timescale Updates

- Output-layer weight updates
  - Online, real-time, adaptive
  - Concurrent learning-based update law

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

- Inner-layer feature updates
  - Iterative, optimization algorithm, batch updates
  - Loss function

$$\mathcal{L}_{i+1}(t) = \frac{1}{M}\sum_{j=1}^{M}\left\|\dot{x}_j - g_j(x_j)u_j - \hat{\theta}^T\phi\left(\widehat{\Phi}_i(x_j)\right)\right\|^2$$

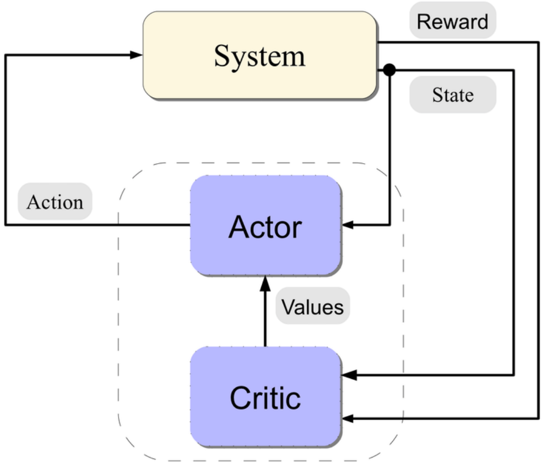
Approximate Dynamic Programming

- Optimal value function (cost-to-go)

$$V^*(x) = \min_{\mu(\tau)\epsilon U} \int_t^{\infty} Q\big(\zeta(\tau)\big) + \mu(\tau)^T R\mu(\tau)\, d\tau$$

- Optimal control policy

$$\mu^*(\zeta) = -\frac{1}{2} R^{-1} G(\zeta)^T \left(\nabla_\zeta V^*(\zeta)\right)^T$$



DNN Value Function Approximation

DNN Optimal Value Function and DNN Optimal Control Policy

$$V^*(\zeta) = W^T\psi\left(\Psi^*(\zeta)\right) + \epsilon_v(\zeta) \qquad \mu^*(\zeta) = -\frac{1}{2}R^{-1}g(\zeta)^T\left(W^T\nabla_\zeta\psi\left(\Psi^*(\zeta)\right) + \nabla_\zeta\epsilon_v(\zeta)^T\right)$$

$\widehat{W}_c$ = Critic weight estimate

$\widehat{W}_a$ = Actor weight estimate

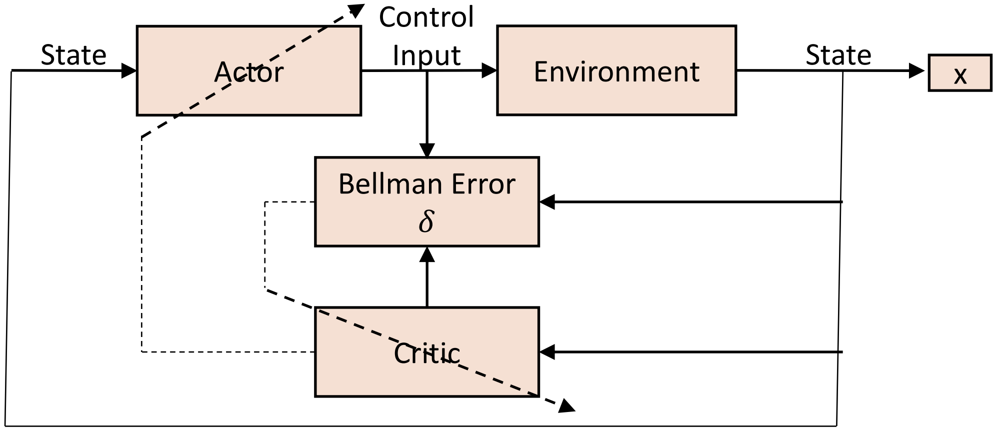
Optimal Value Function and Optimal Control Policy Approximation

$$\hat{V}(\zeta,\widehat{W}_c) = \widehat{\boldsymbol{W}}_{\boldsymbol{c}}^{T}\psi\left(\hat{\Psi}_i(\zeta)\right) \qquad \hat{\mu}(\zeta,\widehat{W}_a) = -\frac{1}{2}R^{-1}g(\zeta)^T\left(\widehat{\boldsymbol{W}}_{\boldsymbol{a}}^{T}\nabla_\zeta\psi\left(\hat{\Psi}_i(\zeta)\right)^T\right)$$

- Bellman Error (BE)

$$\hat{\delta}(\zeta, \widehat{W}_c, \widehat{W}_a, \hat{\theta}) \triangleq Q(\zeta) + \hat{\mu}(\zeta, \widehat{W}_a)^T R \hat{\mu}(\zeta, \widehat{W}_a) + \nabla_\zeta \hat{V}(\zeta, \widehat{W}_c)\left(\hat{F}_i(\zeta, \hat{\theta}) + G(\zeta)\hat{\mu}(\zeta, \widehat{W}_a)\right)$$

- BE extrapolation
  - Evaluation over user-defined, off trajectory points
  - Satisfies persistence of excitation



Output-Layer Weight Update

- Critic Weight Update Law

$$\dot{\hat{W}}_c(t) = -\eta_{c1}\Gamma\frac{\omega(t)}{\rho(t)}\delta(t) - \eta_{c2}\frac{1}{N}\sum_{e=1}^{N}\frac{\omega_e(t)}{\rho_e(t)}\delta_e(t)$$

On-trajectory points

Off-trajectory points

- Learning Gain Update Law

$$\dot{\Gamma}(t) = \left(\lambda\Gamma(t) - \frac{\eta_{c1}\Gamma(t)\omega(t)\omega(t)^T\Gamma(t)}{\rho(t)} - \eta_{c2}\Gamma(t)\left(\frac{1}{N}\sum_{e=1}^{N}\frac{\omega_e(t)\omega_e^T(t)}{\rho_e(t)}\right)\Gamma(t)\right)\mathbf{1}_{\{\underline{\Gamma}\le\|\Gamma\|\le\overline{\Gamma}\}}$$

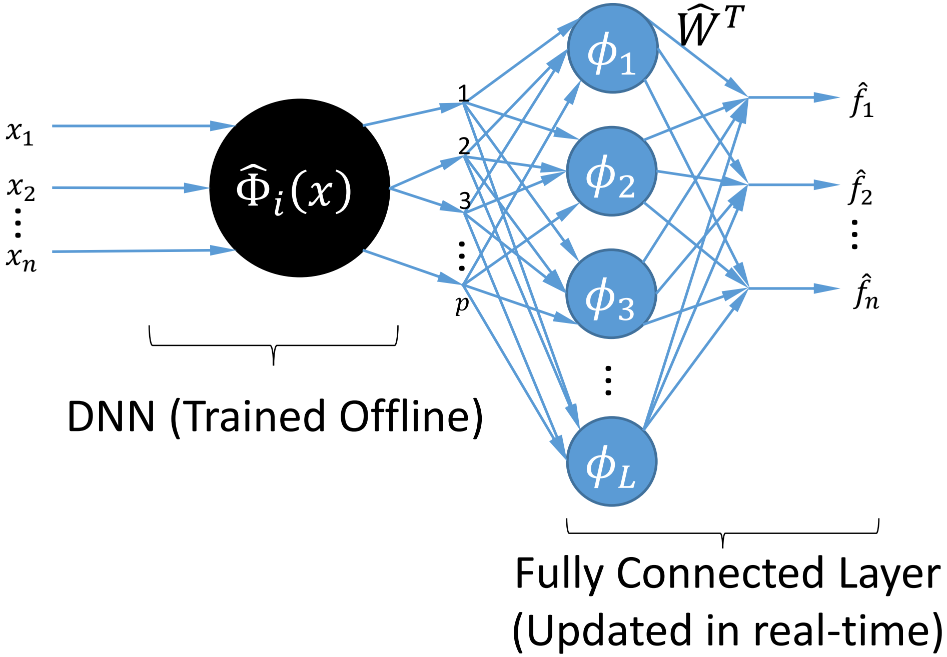
- Actor Weight Update Law

$$\dot{\hat{W}}_a(t) = -\eta_{c1}\left(\hat{W}_a(t) - \hat{W}_c(t)\right) - \eta_{a2}\hat{W}_a(t) + \frac{\eta_{c1}G_{\hat{\psi}}^T(t)\hat{W}_a(t)\omega(t)^T}{4\rho(t)}\hat{W}_c(t)$$
$$+\left(\frac{\eta_{c2}}{N}\sum_{i=1}^{N}\frac{G_{\hat{\psi}e}^T\hat{W}_a(t)\omega_e(t)}{4\rho_e(t)}\right)\hat{W}_c(t)$$

Inner-Layer Feature Update

- Loss function

$$\mathcal{L}_{\delta_{k+1}}(t) = \frac{1}{N}\sum_{k=1}^{N}\left\|\hat{\delta}_e\right\|^2$$

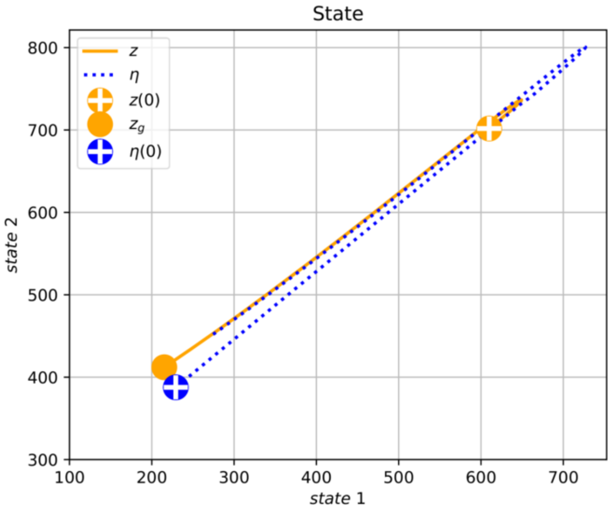


Stability Analysis

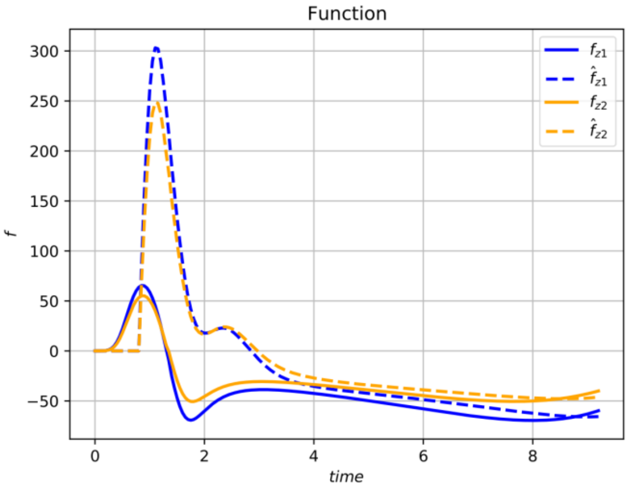
Theorem 1. *Provided the necessary assumptions, gain conditions, and initial conditions hold, then the tracking error $e(t)$, weight estimation errors $\widetilde{W}_c(t)$ and $\widetilde{W}_a(t)$, system ID DNN output-layer weight estimation errors $\tilde{\theta}(t)$, state estimator error $\tilde{x}(t)$, and control policy $\mu(t)$ are uniformly ultimately bounded (UUB).*

- While iteratively retraining DNN terms:
  - $\|e\|$ converges to a neighborhood around zero
  - $\|\tilde{\theta}\|$, $\|\tilde{x}\|$ converge to a neighborhood around zero
  - $\|\widetilde{W}_c\|$, $\|\widetilde{W}_a\|$ converge to a neighborhood around zero
  - $\mu(t)$ is an approximation of the optimal control policy

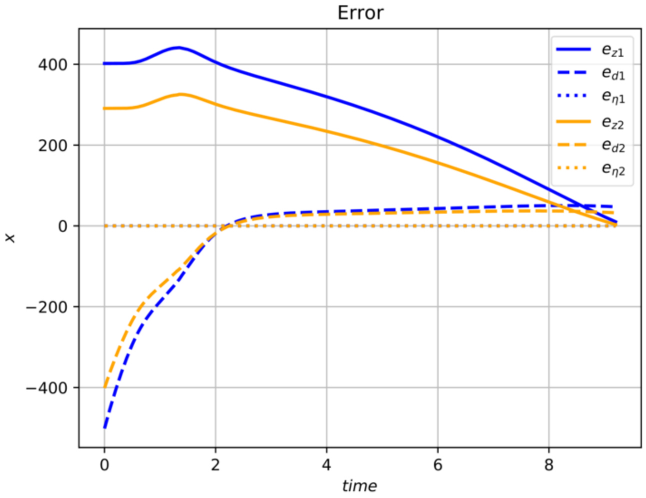
Simulations



Pursing agent (blue) herding an evading agent (orange) to a goal location via interaction dynamics



DNN function approximation for system identification where the true values are shown with solid lines and the estimated values are shown with dashed lines



The evader tracking error steadily decays after the evader initially flees. The auxiliary errors also converge to a small radius of the goal.

Summary

- DNN 1: Approximate the dynamics
  - CL-based adaptive update law
  - CL loss function
- DNN 2: Approximate the value function
  - Actor-critic weight update laws
  - BE loss function
- Future work includes updating the inner-layer features of the DNNs in real-time.

Deep Residual Neural Network (ResNet)-Based Adaptive Control: A Lyapunov-Based Approach

O. Patil, D.M. Le, E.J. Griffis, W. E. Dixon, "Deep Residual Neural Network (ResNet)-Based Adaptive Control: A Lyapunov-Based Approach," *In Proc IEEE Conf. Decis. Control*, 2022.

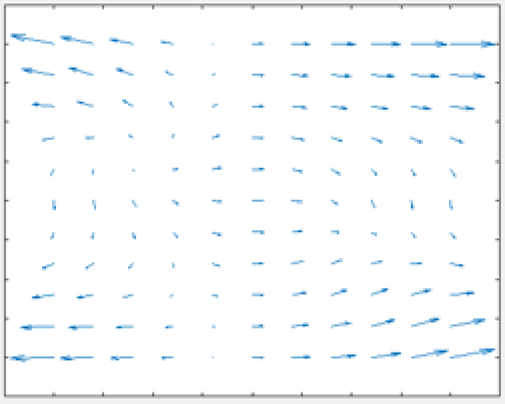
O. Patil, D.M. Le, E.J. Griffis, W. E. Dixon, "Deep Residual Neural Network (ResNet)-Based Adaptive Control: A Lyapunov-Based Approach," *Automatica*, Under Review.

Introduction

Consider the dynamic model

$$\dot{x} = \underbrace{f\left(x\right)}_{} + u$$

Unknown drift vector field approximated using a DNN



Control objective : $e \triangleq x - x_d(t) \to 0$ as $t \to \infty$ where $x_d(t) \in \Omega$, a known compact set

Introduction

- Challenges with offline methods
  - Open-loop in weights
  - Richness of data
  - Required knowledge of state-derivative
- Lewis et. al. (1996) provided real-time adaptation laws for a single-hidden-layer neural network-based adaptive controller
- Our recent work (Patil et. al., 2022) provided Lyapunov-derived adaptation laws for fully-connected DNNs with an arbitrary number of layers
- However, fully-connected DNNs are susceptible to the problem of vanishing or exploding gradients

Fully-Connected DNN Controller

- Given the DNN architecture

$$\Phi^{\theta}(\eta) \triangleq \left(V_k^T \phi_k \circ \ldots \circ V_1^T \phi_1\right)\left(V_0^T \eta\right)$$

the control and adaptation law in Patil et. al. (2022) is given by

$$u \triangleq \dot{x}_d - \rho(\|e\|)e - k_1 e - k_s \operatorname{sgn}(e) - \Phi^{\hat{\theta}}(x_d)$$

$$\dot{\hat{\theta}} \triangleq \operatorname{proj}\left(\Gamma\left(\frac{\partial \Phi^{\hat{\theta}}(x_d)}{\partial \hat{\theta}}\right)^T e\right)$$

where

$$\hat{\theta} \triangleq \left[\operatorname{vec}(\hat{V}_0)^T, \ldots, \operatorname{vec}(\hat{V}_k)^T\right]^T$$

$$\hat{\varphi}_j \triangleq \varphi_j(x_d, \hat{V}_0, \ldots, \hat{V}_j)$$

$$\hat{\varphi}'_j \triangleq \varphi'_j(x_d, \hat{V}_0, \ldots, \hat{V}_j)$$

$$\frac{\partial \Phi^{\hat{\theta}}(x_d)}{\partial \hat{\theta}} = \left[\left(\frac{\partial \Phi^{\hat{\theta}}(x_d)}{\partial \operatorname{vec}(\hat{V}_0)}\right), \ldots, \left(\frac{\partial \Phi^{\hat{\theta}}(x_d)}{\partial \operatorname{vec}(\hat{V}_k)}\right)\right]$$

$$\frac{\partial \Phi^{\hat{\theta}}(x_d)}{\partial \operatorname{vec}(\hat{V}_0)} = \left(\overset{\curvearrowleft}{\prod}{}_{l=1}^{k} \hat{V}_l^T \hat{\varphi}'_l\right)\left(I_{L_1} \otimes x_d^T\right)$$

$$\frac{\partial \Phi^{\hat{\theta}}(x_d)}{\partial \operatorname{vec}(\hat{V}_j)} = \left(\overset{\curvearrowleft}{\prod}{}_{l=j+1}^{k} \hat{V}_l^T \hat{\varphi}'_l\right)\left(I_{L_1} \otimes \hat{\varphi}_j^T\right)$$

ResNet Architecture

- ResNets contain shortcut connections
- A ResNet can be modeled using fully-connected blocks as

$$\Phi^{\theta}(\eta_1) \triangleq \eta_m + \Phi_m^{\theta_m}(\eta_m)$$

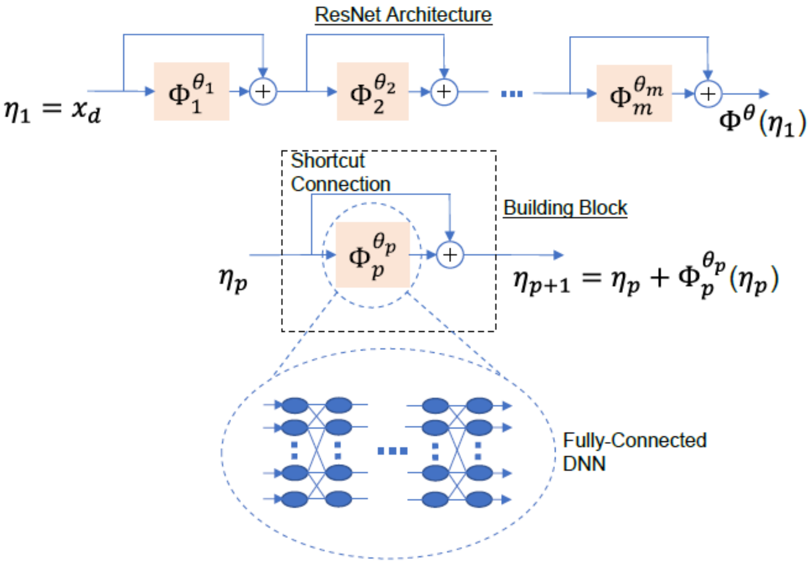
$$\eta_p = \begin{cases} \eta_{p-1} + \Phi_{p-1}^{\theta_{p-1}}(\eta_{p-1}), & p \in \{2, \ldots, m\}, \\ x_d, & p = 1. \end{cases}$$

- Each fully-connected block can be expressed using the recursive relation

$$\Phi_p = V_{k,p}^T \varphi_{k,p}$$

$$\varphi_{p,j} \triangleq \begin{cases} \phi_{p,j}\left(V_{p,j-1}^T \varphi_{p,j-1}\right), & j \in \{1, \ldots, k_p\}, \\ \eta_p, & j = 0. \end{cases}$$

where k_p denotes the depth of the p^{th} block



Adaptation Law

$$\dot{\hat{\theta}} \triangleq \operatorname{proj}\left(\Gamma \Phi'^{T} e\right)$$

where $\Phi' \triangleq \frac{\partial \Phi^{\hat{\theta}}(x_d)}{\partial \hat{\theta}}$ can be computed using the chain rule as

$$\Phi' = \begin{bmatrix} \Phi'_1, & \ldots, & \Phi'_m \end{bmatrix}$$

$$\Phi'_p = \left(\overset{\curvearrowleft}{\prod}{}_{l=1}^{k_p} \left(I_n + \Xi_l \right) \right) \Lambda_p$$

Prevents vanishing gradients

$$\Lambda_{p,0} = \left(\overset{\curvearrowleft}{\prod}{}_{l=1}^{k_p} \hat{V}_{p,l}^{T} \hat{\varphi}'_{p,l} \right) \left(I_{p,L_{p,1}} \otimes \hat{\eta}_p^{T} \right)$$

$$\Lambda_{p,j} = \left(\overset{\curvearrowleft}{\prod}{}_{l=j+1}^{k_p} \hat{V}_{p,l}^{T} \hat{\varphi}'_{p,l} \right) \left(I_{p,L_{p,j+1}} \otimes \hat{\varphi}_{p,j}^{T} \right)$$

$$\Xi_p = \left(\overset{\curvearrowleft}{\prod}{}_{l=1}^{k_p} \hat{V}_{p,l}^{T} \hat{\varphi}'_{p,l} \right) \hat{V}_{p,0}^{T}$$

Simulations

- Dynamics

$$n = 10$$
$$f(x) = Ay(x)$$
$$A \in \mathbb{R}^{n\times 6n} \text{ with all elements from } U(0, 0.1)$$
$$y(x) \triangleq \left[x^T, \tanh(x)^T, \sin(x)^T, \operatorname{sech}(x)^T, (x\odot x)^T, (x\odot x\odot x)^T\right]^T$$

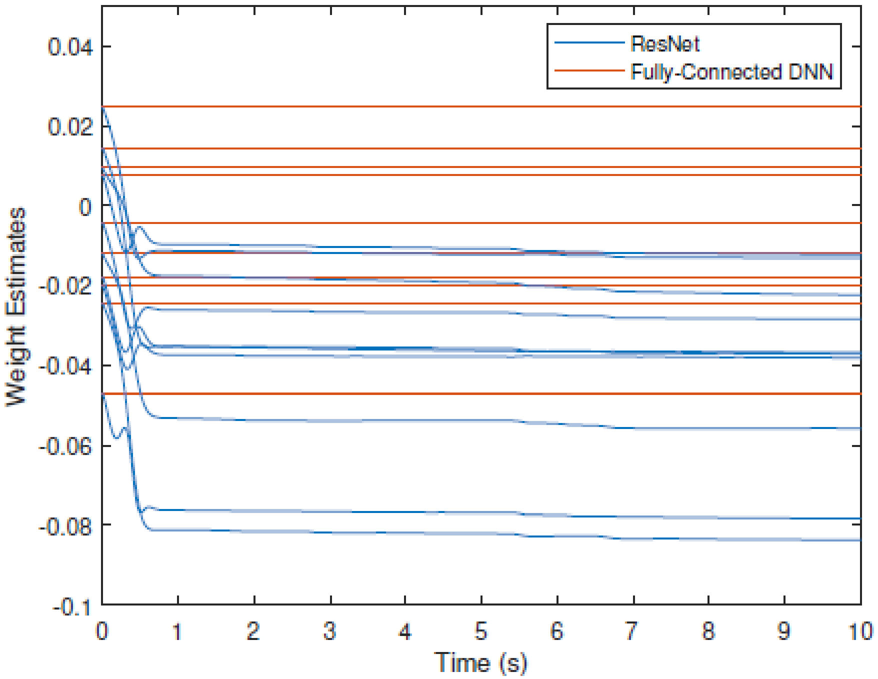
- Reference Trajectory

$$x_d(t) = [0.5 + \sin(\omega_1 t), \ldots, 0.5 + \sin(\omega_n t)]$$
$$\omega_1, \ldots, \omega_n \sim U(0, 20)$$

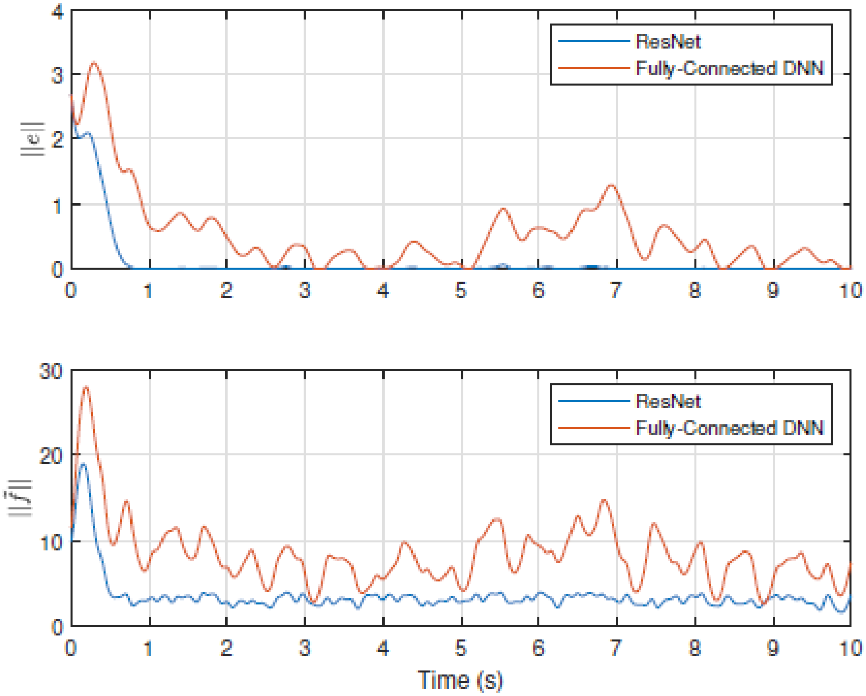
- ResNet configuration is selected with 20 hidden layers, a shortcut connection across each hidden layer, and 10 neurons in each layer.
- For a fair comparison, weights are initialized using Monte Carlo approach with 10,000 iterations to minimize

$$J = \int_0^{10} \left(e^T(t)Qe(t) + u^T(t)Ru(t)\right), \quad Q = I_{10}, R = 0.01 I_{10}$$

Simulations



Plot of weight estimates with ResNet and fully-connected DNN



Plot of tracking and function approximation errors with ResNet and fully-connected DNN

Architecture	$\|e_{\mathrm{rms}}\|$	$\|\tilde{f}_{\mathrm{rms}}\|$	$\|u_{\mathrm{rms}}\|$
ResNet	0.420	4.265	24.290
Fully-Connected	0.832	9.348	24.711

Conclusions and Future Work

- ResNet-based adaptive controller was developed for control-affine nonlinear systems
- ResNet provided twofold improvement in tracking and function approximation performance in comparison to an equivalent fully-connected DNN
- Future work may involve concurrent learning-based adaptation for DNNs

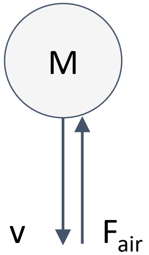
Deep Lyapunov-Based Physics-Informed Neural Networks (DeLb-PINN) for Adaptive Control Design

Rebecca G. Hart, Omkar Sudhir Patil, Emily J. Griffis, and Warren E. Dixon, “Deep Lyapunov-Based Physics-Informed Neural Networks (DeLb-PINN) for Adaptive Control Design”,*IEEE Conf. Decis. Control*, Under Review.

Physics Informed Machine Learning

Motivation: To impose constraints derived from known physical laws on the learning algorithm to reduce the possible solution space and eliminate invalid solutions resulting from noisy data

Historical Example



Aristotle (~330 B.C)
The speed of a falling object is proportional to the weight of the object

Galileo (~1589)
All objects fall at the same rate irrespective of their mass

Connection to Machine Learning

A model based purely on measurement would yield the Aristotelian theory of gravitation for a falling object [1].

A challenge in machine learning and artificial intelligence it's heavy reliance on data which can be noisy or scarce

[1] de Silva BM, Higdon DM, Brunton SL, Kutz JN. Discovery of Physics From Data: Universal Laws and Discrepancies. Front Artif Intell. 2020 Apr 28

Contribution

Many physics inspired learning algorithms have been shown to be useful to approximate solution to various forms of differential equations

- Many are not constructed for control applications

Deep Neural Networks which have high function approximation capabilities, have been integrated in control algorithms and yielded an improvement in tracking capabilities

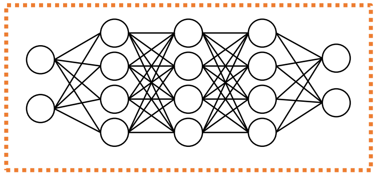
- These DNNs are still data-based algorithms

Deep Lyapunov-Based Physics Informed Neural Network (DeLb-PINN) architecture combines the function approximation performance of DNNs while leveraging knowledge of the system

Contribution

Previous continuous-time adaptation laws for all layers DNN-based controllers do not leverage known physics of the system.

$$\underbrace{M(q)\ddot{q} + V_m(q,\dot{q})\dot{q} + G(q) + F\left(\dot{q}\right) + \tau_d\left(t\right)} = \tau\left(t\right)$$

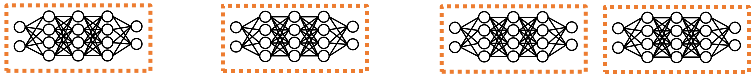


Using the Data Based Approach:
The entire function is approximated by a DNN

Contribution

DeLb-PINN architecture uses the structure of a governing Euler-Lagrange and uses DNN representations to approximate unknown matrix structures.

$$\underbrace{M(q)}\ddot{q}+\underbrace{V_m(q,\dot{q})}\dot{q}+\underbrace{G(q)}+\underbrace{F\left(\dot{q}\right)}+\tau_d\left(t\right)=\tau\left(t\right)$$



Using the Physics Inspired Approach:
Only unknown matrices are approximated using DNNs

DNNs can only approximate vectors, not matrices

System Dynamics and Control Objective

System Dynamics

Consider a general uncertain Euler-Lagrange system modeled as

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + G(q) + F(\dot{q}) + \tau_d(t) = \tau(t)$$

Design a controller to track a desired trajectory $q_d : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$

$$\begin{aligned} e &\triangleq q - q_d \\ r &\triangleq \dot{e} + \alpha e \end{aligned}$$

Closed Loop Error System after some algebraic manipulation can be written as

$$\begin{aligned} M(q)\dot{r} = {} & \tau(t) - \tau_d(t) - V_m(q, \dot{q}) + (M_e - M(q_d))(\ddot{q}_d - \alpha\dot{e}) \\ & + (V_{me} - V_m(q_d, \dot{q}_d))(\dot{q}_d - \alpha e) + G_e - G(q_d) + F_e - F(\dot{q}_d) \end{aligned}$$

Mathematical Challenges

Using the universal function approximation property, the unknown terms can be modeled as

$$\begin{aligned}
\operatorname{vec}\left(M\left(q_d\right)\right) &= \Phi_M\left(x_M, \theta_M^*\right) + \varepsilon_M\left(x_M\right) \\
\operatorname{vec}\left(V_m\left(q_d, \dot{q}_d\right)\right) &= \Phi_{V_m}\left(x_{V_m}, \theta_{V_m}^*\right) + \varepsilon_{V_m}\left(x_{V_m}\right) \\
G(q_d) &= \Phi_G\left(x_G, \theta_G^*\right) + \varepsilon_G\left(x_G\right) \\
F(\dot{q}_d) &= \Phi_F\left(x_F, \theta_F^*\right) + \varepsilon_F\left(x_F\right)
\end{aligned}$$

Recall

$$M\left(q_d\right)\boxed{\left(\ddot{q}_d - \alpha\dot{e}\right)}$$

$$V_m\left(q_d, \dot{q}_d\right)\boxed{\left(\dot{q}_d - \alpha e\right)}$$

Properties $\operatorname{vec}(A) \triangleq \left[a_{1,1}, \ldots, a_{1,m}, \ldots, a_{n,m}\right]^\top$ $\operatorname{vec}(ABC) = \left(C^\top \otimes A\right)\operatorname{vec}(B)$

$$M\left(q_d\right)\left(\ddot{q}_d - \alpha\dot{e}\right) \longrightarrow \left(\left(\ddot{q}_d - \alpha\dot{e}\right)^\top \otimes I_n\right)\left(\Phi_M\left(x_M, \theta_M^*\right) + \varepsilon_M\left(x_M\right)\right)$$

$$V_m\left(q_d, \dot{q}_d\right)\left(\dot{q}_d - \alpha e\right) \longrightarrow \left(\left(\dot{q}_d - \alpha e\right)^\top \otimes I_n\right)\left(\Phi_{V_m}\left(x_{V_m}, \theta_{V_m}^*\right) + \varepsilon_{V_m}\left(x_{V_m}\right)\right)$$

Control Design

Using the DeLb-PINN architecture the control input is designed as

$$\tau\left(t\right)=\left(\left(\dot{q}_d-\alpha e\right)^{\top}\otimes I_n\right)\boxed{\widehat{\Phi}_{V_m}}+\boxed{\widehat{\Phi}_G}+\boxed{\widehat{\Phi}_F}+\left(\left(\ddot{q}_d-\alpha\dot{e}\right)^{\top}\otimes I_n\right)\boxed{\widehat{\Phi}_M}-k_1r-e$$

$$-\mathrm{sgn}(r)\left(\rho(\|z\|)\|z\|+k_2+k_3\|(\dot{q}_d-\alpha e)^{\top}\otimes I_n\|\right)-k_4\mathrm{sgn}(r)\|(\ddot{q}_d-\alpha\dot{e})^{\top}\otimes I_n\|$$

Where the DNN estimates adapt in real time according to the following update laws

$$\dot{\hat{\theta}}_M=-\mathrm{proj}\left(\Gamma_M\widehat{\Phi}_M'^{\top}\left(\left(\ddot{q}_d-\alpha\dot{e}\right)^{\top}\otimes I_n\right)^{\top}r\right)$$

$$\dot{\hat{\theta}}_F=-\mathrm{proj}\left(\Gamma_F\widehat{\Phi}_F'^{\top}r\right)$$

$$\dot{\hat{\theta}}_{V_m}=-\mathrm{proj}\left(\Gamma_{V_m}\widehat{\Phi}_{V_m}'^{\top}\left(\left(\dot{q}_d-\alpha e\right)^{\top}\otimes I_n\right)^{\top}r\right)$$

$$\dot{\hat{\theta}}_G=-\mathrm{proj}\left(\Gamma_G\widehat{\Phi}_G'^{\top}r\right)$$

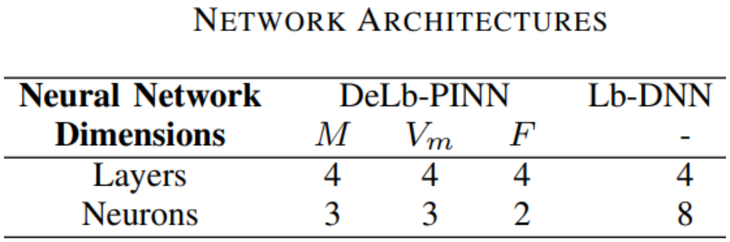
Stability Result

Theorem. For the general Euler-Lagrange system, the developed controller and adaptation laws ensure global asymptotic tracking in the sense that $\|e(t)\| \to 0$ and $\|r(t)\| \to 0$ as $t \to \infty$ provided sufficient gain conditions are met.

Simulations

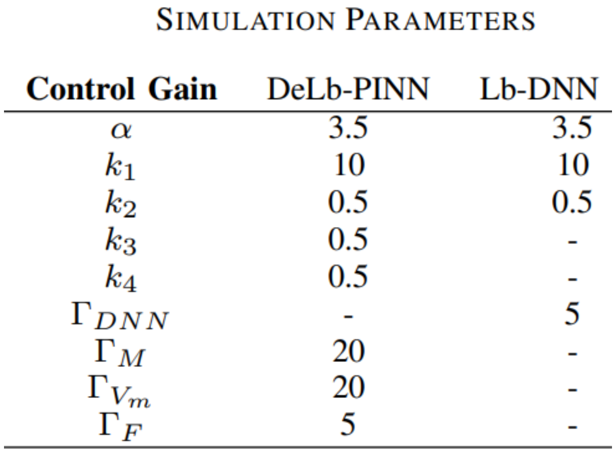
The developed method was implemented on a two-link planer revolute robot and compared to a baseline Lyapunov-based DNN adaptive controller

For a fair comparison, the same number of neurons was used in each architecture and the control gains were kept the same



NETWORK ARCHITECTURES

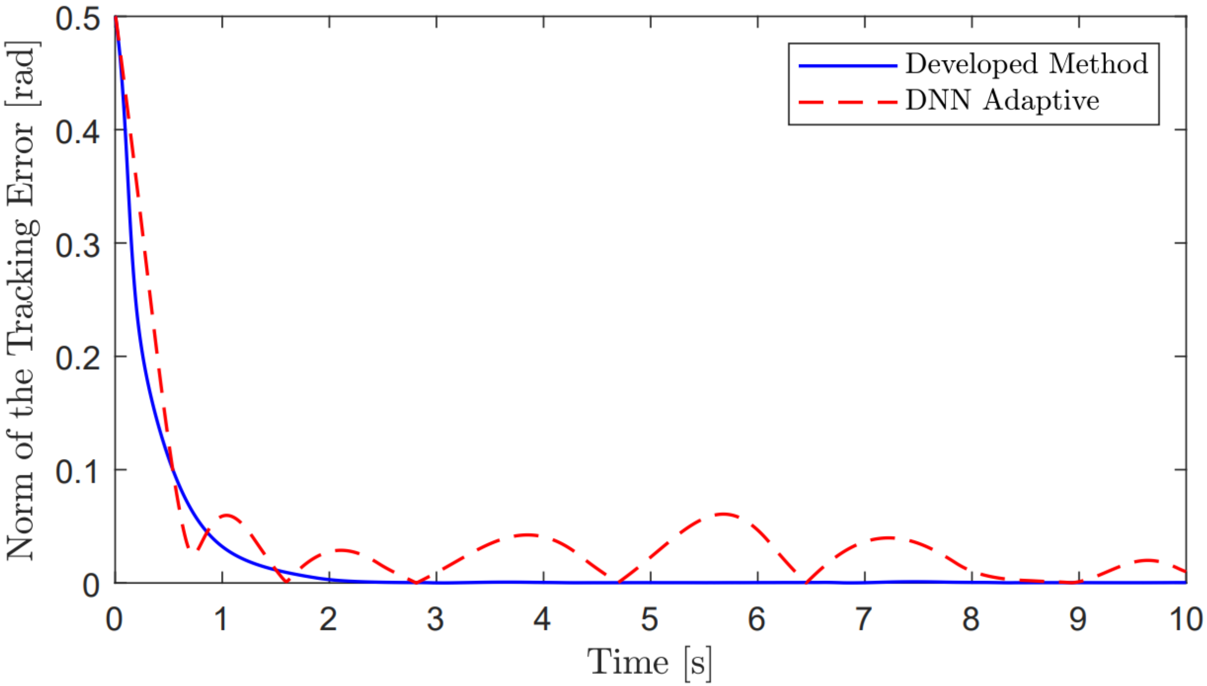
Neural Network	DeLb-PINN			Lb-DNN
Dimensions	M	V_m	F	-
Layers	4	4	4	4
Neurons	3	3	2	8



SIMULATION PARAMETERS

Control Gain	DeLb-PINN	Lb-DNN
α	3.5	3.5
k_1	10	10
k_2	0.5	0.5
k_3	0.5	-
k_4	0.5	-
Γ_{DNN}	-	5
Γ_M	20	-
Γ_{V_m}	20	-
Γ_F	5	-

Simulations



Note: Larger robustifying control gains could yield improved tracking error convergence for the DNN adaptive method, but would likely lead to high-gain oscillatory behavior in the control input

Conclusion

A DeLb-PINN adaptive controller leverages knowledge of the system and the function approximation capabilities of DNNs.

Simulations showed that the developed DeLb-PINN adaptive controller yielded a 19.91% improvement in RMS tracking error compared to a baseline adaptive DNN controller.

Future work would include constraining the output of the DNN to further respect physical properties of the matrices such as positive definiteness.

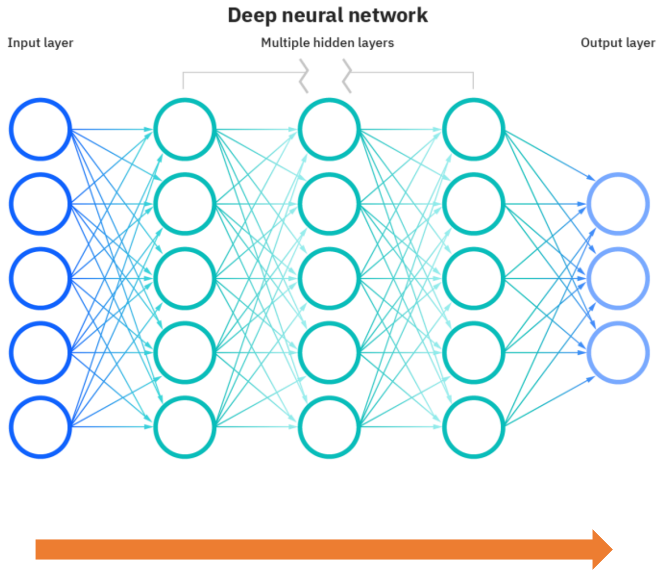
Lyapunov-Based Long Short-Term Memory (Lb-LSTM) Neural Network-Based Control

Emily 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*, Under Review.

UF UNIVERSITY of FLORIDA Duke UNIVERSITY TEXAS The University of Texas at Austin UC SANTA CRUZ

Neural Networks (NNs)

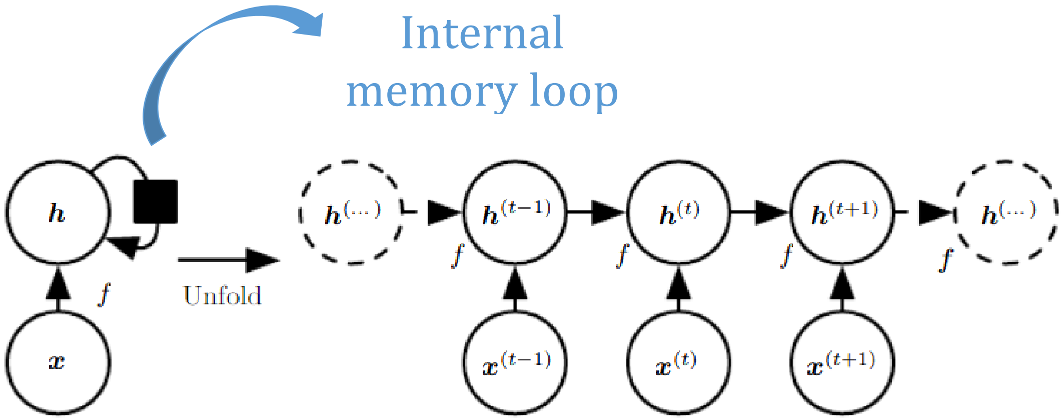
- Most adaptive NN control methods are restricted to feedforward NNs
    - static structures
    - only have access to current state information
- The presence of a memory capable of accessing previous state information both reduces the required data set for training and leads to faster learning.



Flow of information
(no memory loop)

Recurrent Neural Networks (RNNs)

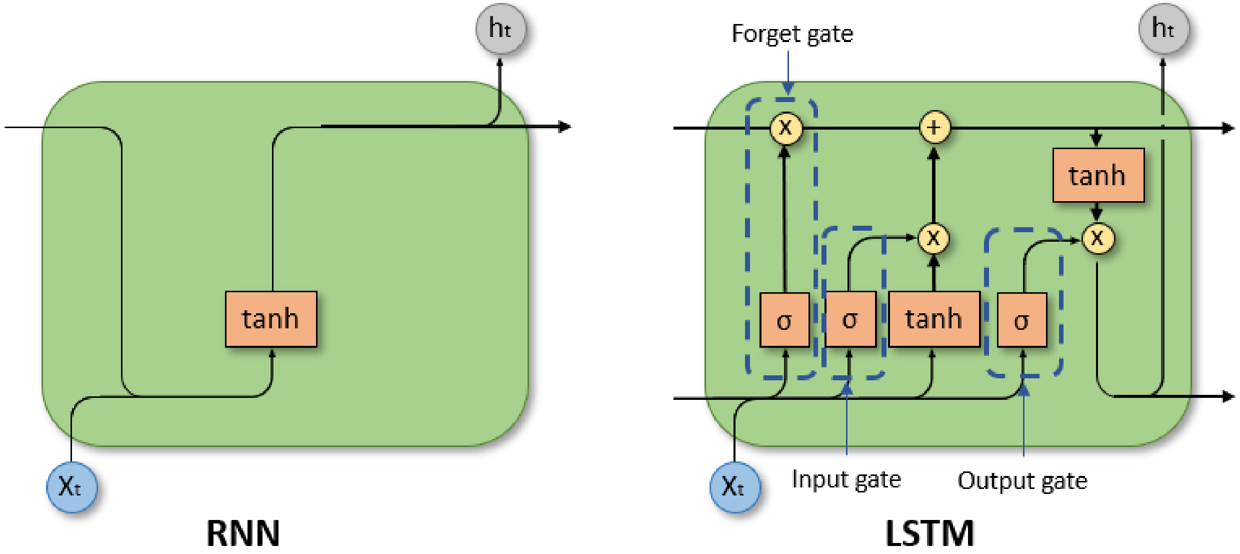
- RNNs can capture the temporal dynamic behavior of an unknown system.
- RNNs have an internal memory that can leverage dependencies in a sequence and increase approximation capabilities, thus improving performance.



Internal memory loop

Long Short-Term Memory (LSTMs)

- The structure of traditional RNNs inhibits their ability to learn long-term time dependencies.
- LSTMs have a better ability to learn long term dependencies, and therefore, have improved memory capability when compared to traditional RNNs.

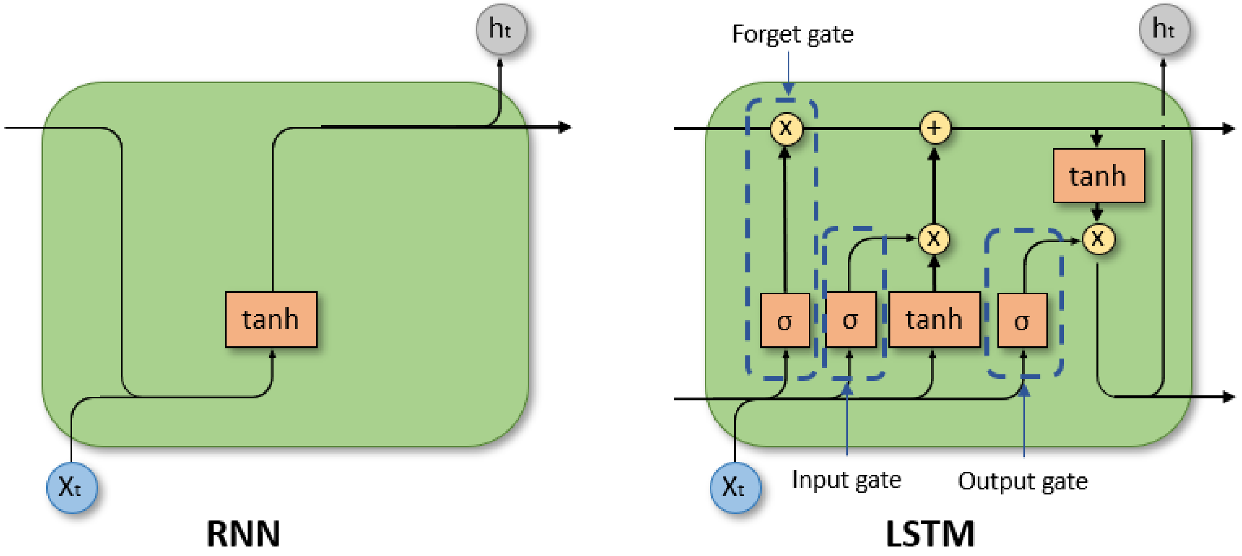


RNN

LSTM

Long Short-Term Memory (LSTMs)

- LSTMs regulate the flow of the gradient along long time sequences by adding an explicit memory through three gate units: the input, forget, and output gates.
  - Retain relevant information and forget irrelevant information stored in the internal memory.



Contribution

- Previous LSTM-based control results use offline optimization techniques to train the LSTM weights.
  - No online learning of the LSTM
- An adaptive Lyapunov-based LSTM (Lb-LSTM) controller is developed for general Euler-Lagrange systems.
  - A continuous-time Lb-LSTM NN is developed to adaptively estimate uncertain model dynamics.
  - Stability-driven adaptation laws adjust the Lb-LSTM weights in real-time.

System Dynamics and Control Objective

- Consider a general uncertain Euler-Lagrange system

$$M(q)\ddot{q} + V_m(q)\dot{q} + F(\dot{q}) + G(q) = \tau$$

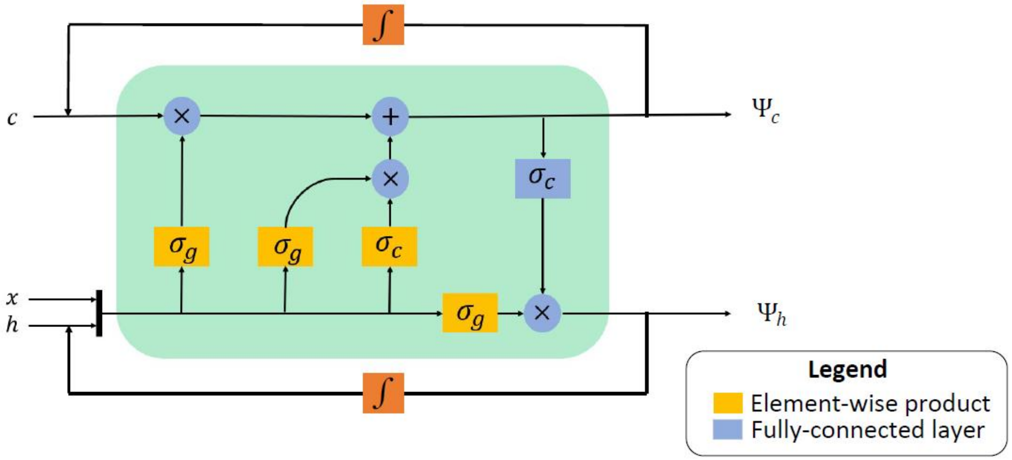
- Design a controller to track a desired trajectory $q_d \in \mathbb{R}^n$

$$e \triangleq q_d - q$$
$$r \triangleq \dot{e} + \alpha e$$

- Design an Lb-LSTM to adaptively learn unknown system dynamics $g(x) = M(q)(\ddot{q}_d + \alpha\dot{e}) + V_m(q,\dot{q})(\dot{q}_d + \alpha e) + F(\dot{q}) + G(q)$.

$$M(q)\dot{r} = \boxed{g(x)} + \tau - V_m(q,\dot{q})r$$

LSTM Model



An LSTM NN can be modeled in continuous-time as

Gate Outputs

$$f(z, W_f) = \sigma_g \circ W_z^\top z$$

$$o(z, W_o) = \sigma_g \circ W_o^\top z$$

$$i(z, W_i) = \sigma_g \circ W_i^\top z$$

$$c^*(z, W_c) = \sigma_c \circ W_c^\top z$$

$z \triangleq [x^\top h^\top]^\top$ for some input x

Cell State and Hidden State Dynamics

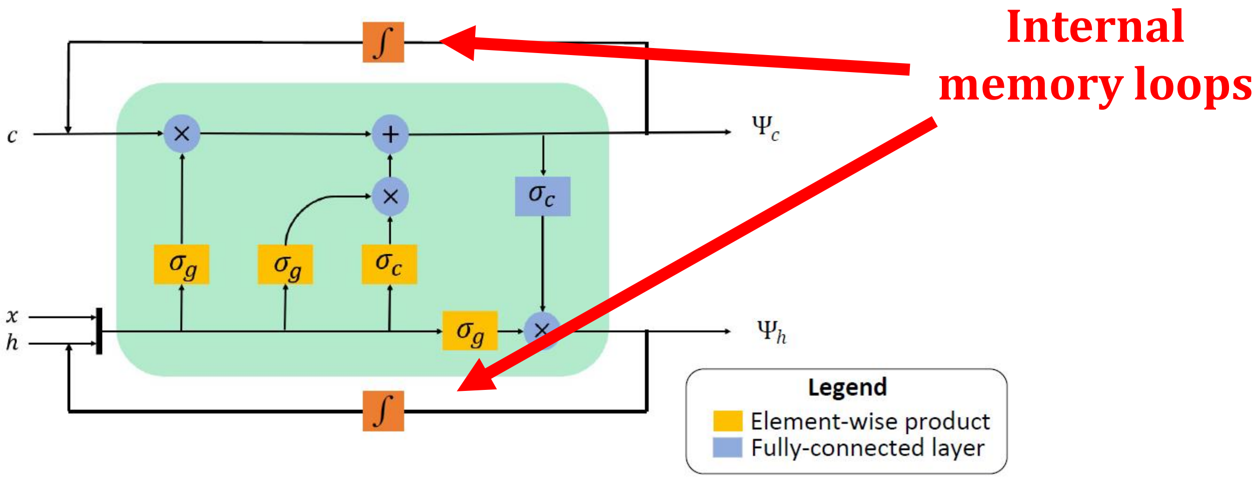
$$\dot{c} = -b_c c + b_c \Psi_c(x, c, h, \theta)$$

$$\dot{h} = -b_h h + b_h \Psi_h(x, c, h, \theta, W_o)$$

$$\Psi_c(x, c, h, \theta) = f(z, W_f) \odot c + i(z, W_i) \odot c^*(z, W_c)$$

$$\Psi_h(x, c, h, \theta, W_o) = o(z, W_o) \odot (\sigma_c \circ \Psi_c(x, c, h, \theta))$$

LSTM Model



Internal memory loops

An LSTM NN can be modeled in continuous-time as

Gate Outputs

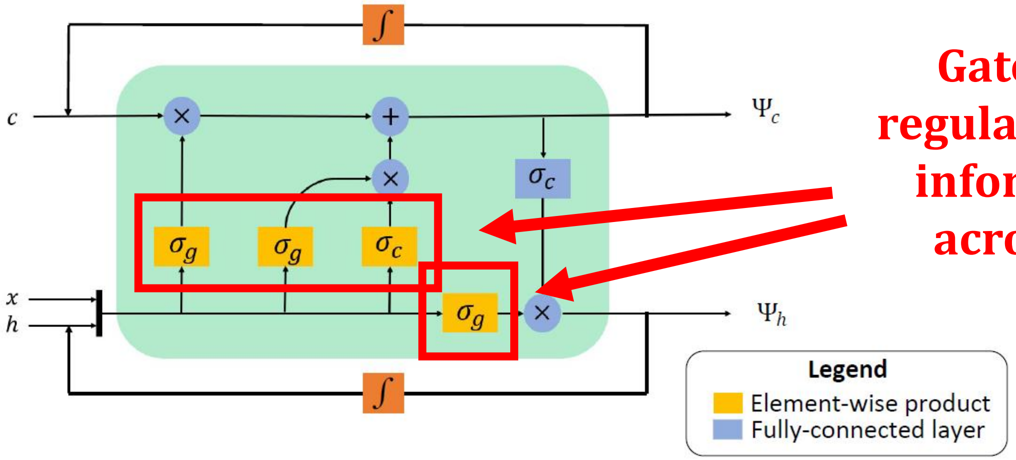
$$f(z, W_f) = \sigma_g \circ W_z^\top z$$
$$o(z, W_o) = \sigma_g \circ W_o^\top z$$
$$i(z, W_i) = \sigma_g \circ W_i^\top z$$
$$c^*(z, W_c) = \sigma_c \circ W_c^\top z$$

$z \triangleq [x^\top h^\top]^\top$ for some input x

Cell State and Hidden State Dynamics

$$\dot{c} = -b_c c + b_c \Psi_c(x, c, h, \theta)$$
$$\dot{h} = -b_h h + b_h \Psi_h(x, c, h, \theta, W_o)$$
$$\Psi_c(x, c, h, \theta) = f(z, W_f) \odot c + i(z, W_i) \odot c^*(z, W_c)$$
$$\Psi_h(x, c, h, \theta, W_o) = o(z, W_o) \odot (\sigma_c \circ \Psi_c(x, c, h, \theta))$$

LSTM Model



Gate units regulate flow of information across cell

An LSTM NN can be modeled in continuous-time as

Gate Outputs	Cell State and Hidden State Dynamics
$f(z, W_f) = \sigma_g \circ W_z^\top z$ $o(z, W_o) = \sigma_g \circ W_o^\top z$ $i(z, W_i) = \sigma_g \circ W_i^\top z$ $c^*(z, W_c) = \sigma_c \circ W_c^\top z$ $z \triangleq [x^\top h^\top]^\top$ for some input x	$\dot{c} = -b_c c + b_c \Psi_c(x, c, h, \theta)$ $\dot{h} = -b_h h + b_h \Psi_h(x, c, h, \theta, W_o)$ $\Psi_c(x, c, h, \theta) = f(z, W_f) \odot c + i(z, W_i) \odot c^*(z, W_c)$ $\Psi_h(x, c, h, \theta, W_o) = o(z, W_o) \odot (\sigma_c \circ \Psi_c(x, c, h, \theta))$

LSTM Model

- LSTM Output

$$\Phi = W_h^T\left(\Psi_h(x, c, h, \theta, W_o) + \sigma \circ W_{FF}^T x\right)$$

  - Fully-connected output layer ensures appropriate dimensions
  - Feedforward component adds generality
- Model unknown system dynamics using LSTM

$$g(x) = \Phi(x, c, h, \theta, W_o, W_h, W_{FF}) + \epsilon(x)$$

Control Design

- Using the adaptive Lb-LSTM term $\widehat{\Phi} = \Phi(x, \hat{c}, \hat{h}, \hat{\theta}, \widehat{W}_o, \widehat{W}_h, \widehat{W}_{FF})$, the control input is designed as

$$\boxed{\tau \triangleq \widehat{\Phi} + k_r r - K_{2,c}\eta_c - K_{2,h}\eta_h + e}$$

- The estimated cell and hidden states evolve as

$$\dot{\hat{c}} = -b_c\hat{c} + b_c(f\left(\hat{z}, \widehat{W}_f\right) \odot c + i\left(\hat{z}, \widehat{W}_i\right) \odot c^*\left(\hat{z}, \widehat{W}_c\right))$$

$$\dot{\hat{h}} = -b_h\hat{h} + b_h(o\left(\hat{z}, \widehat{W}_o\right) \odot \sigma_c \circ \Psi_c(x, \hat{c}, \hat{h}, \hat{\theta})$$

where $\hat{z} \triangleq \left[x^\top \hat{h}^\top\right]^\top$ and $\hat{\theta} \triangleq \left[\widehat{W}_c^\top \widehat{W}_i^\top \widehat{W}_f^\top\right]^\top$.

Control Design

- Using the adaptive Lb-LSTM term $\widehat{\Phi} = \Phi(x, \hat{c}, \hat{h}, \hat{\theta}, \widehat{W}_o, \widehat{W}_h, \widehat{W}_{FF})$, the control input is designed as

$$\boxed{\tau \triangleq \widehat{\Phi} + k_r r - K_{2,c}\eta_c - K_{2,h}\eta_h + e}$$

- The estimated cell and hidden states evolve as

$$\dot{\hat{c}} = -b_c\hat{c} + b_c(f(\hat{z}, \widehat{W}_f) \odot c + i(\hat{z}, \widehat{W}_i) \odot c^*(\hat{z}, \widehat{W}_c))$$

$$\dot{\hat{h}} = -b_h\hat{h} + b_h(o(\hat{z}, \widehat{W}_o)) \odot \sigma_c \circ \Psi_c(x, \hat{c}, \hat{h}, \hat{\theta})$$

where $\hat{z} \triangleq [x^\top \hat{h}^\top]^\top$ and $\hat{\theta} \triangleq [\widehat{W}_c^\top \widehat{W}_i^\top \widehat{W}_f^\top]^\top$.

Weight Estimates

Weight Adaptation Laws

- The weight adaptation laws are designed as

$$\mathrm{vec}\left(\dot{\hat{\theta}}\right) \triangleq \mathrm{proj}(\Gamma_\theta(b_c\hat{\Psi}_c'^{\top}\eta_c + b_h\hat{\Psi}_{h,\theta}'^{\top}\eta_h + \hat{\Phi}_\theta'^{\top}r - \gamma_\theta \mathrm{vec}(\hat{\theta}))$$

$$\mathrm{vec}\left(\dot{\hat{W}}_o\right) \triangleq \mathrm{proj}(\Gamma_o(b_h\hat{\Psi}_{h,W_o}'^{\top}\eta_h + \hat{\Phi}_{W_o}'^{\top}r - \gamma_o \mathrm{vec}(\hat{W}_o))$$

$$\mathrm{vec}\left(\dot{\hat{W}}_h\right) \triangleq \mathrm{proj}(\Gamma_h(\hat{\Phi}_{W_h}'^{\top}r - \gamma_h \mathrm{vec}(\hat{W}_h))$$

$$\mathrm{vec}\left(\dot{\hat{W}}_{FF}\right) \triangleq \mathrm{proj}(\Gamma_{FF}(\hat{\Phi}_{W_{FF}}'^{\top}r - \gamma_{FF} \mathrm{vec}(\hat{W}_{FF}))$$

Weight Adaptation Laws

- The weight adaptation laws are designed as

$$\operatorname{vec}\left(\dot{\hat{\theta}}\right) \triangleq \operatorname{proj}(\Gamma_\theta(\boxed{b_c \widehat{\Psi}_c'^{\top} \eta_c} + \boxed{b_h \widehat{\Psi}_{h,\theta}'^{\top} \eta_h} + \boxed{\widehat{\Phi}_\theta'^{\top} r} - \boxed{\gamma_\theta \operatorname{vec}(\hat{\theta}))}$$

$$\operatorname{vec}\left(\dot{\widehat{W}}_o\right) \triangleq \operatorname{proj}(\Gamma_o(\boxed{b_h \widehat{\Psi}_{h,W_o}'^{\top} \eta_h} + \boxed{\widehat{\Phi}_{W_o}'^{\top} r} - \boxed{\gamma_o \operatorname{vec}(\widehat{W}_o))}$$

$$\operatorname{vec}\left(\dot{\widehat{W}}_h\right) \triangleq \operatorname{proj}(\Gamma_h(\boxed{\widehat{\Phi}_{W_h}'^{\top} r} - \boxed{\gamma_h \operatorname{vec}(\widehat{W}_h))}$$

$$\operatorname{vec}\left(\dot{\widehat{W}}_{FF}\right) \triangleq \operatorname{proj}(\Gamma_{FF}(\boxed{\widehat{\Phi}_{W_{FF}}'^{\top} r} - \boxed{\gamma_{FF} \operatorname{vec}(\widehat{W}_{FF}))}$$

σ-mod terms

Jacobian-based terms

Main Stability Result

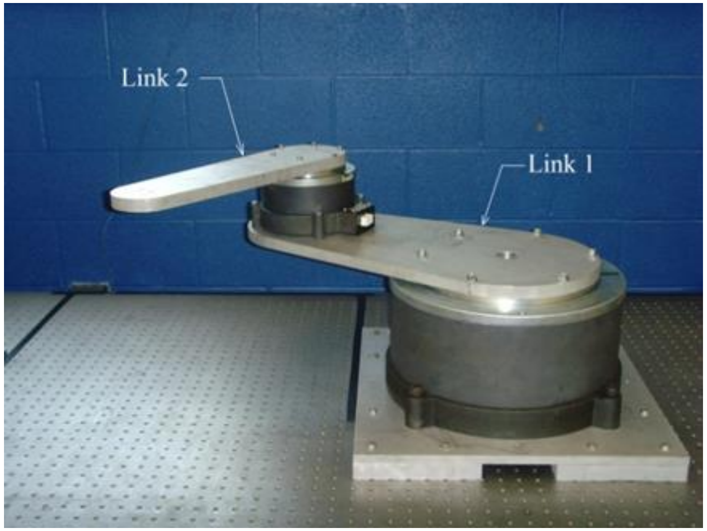
Theorem 1. The adaptive LSTM-based controller and weight adaptation laws ensure the states $\zeta \triangleq [e^\top r^\top \eta_c^\top \tilde{c}^\top\ \eta_h^\top \tilde{h}^\top$ $\mathrm{vec}(\ \tilde{\theta})^\top\ \ \mathrm{vec}(\widetilde{W}_o)^\top\ \mathrm{vec}(\widetilde{W}_{FF})^\top]^\top$ are uniformly ultimately bounded (UUB) in the sense that

$$\|\zeta\| \leq \sqrt{\frac{\beta_2}{\beta_1}\|\zeta(0)\|^2 e^{-\frac{\delta\beta_1}{\lambda}t} + \frac{\delta}{\lambda}\left(1 - e^{-\frac{\delta\beta_1}{\lambda}t}\right)}$$

Simulation Results

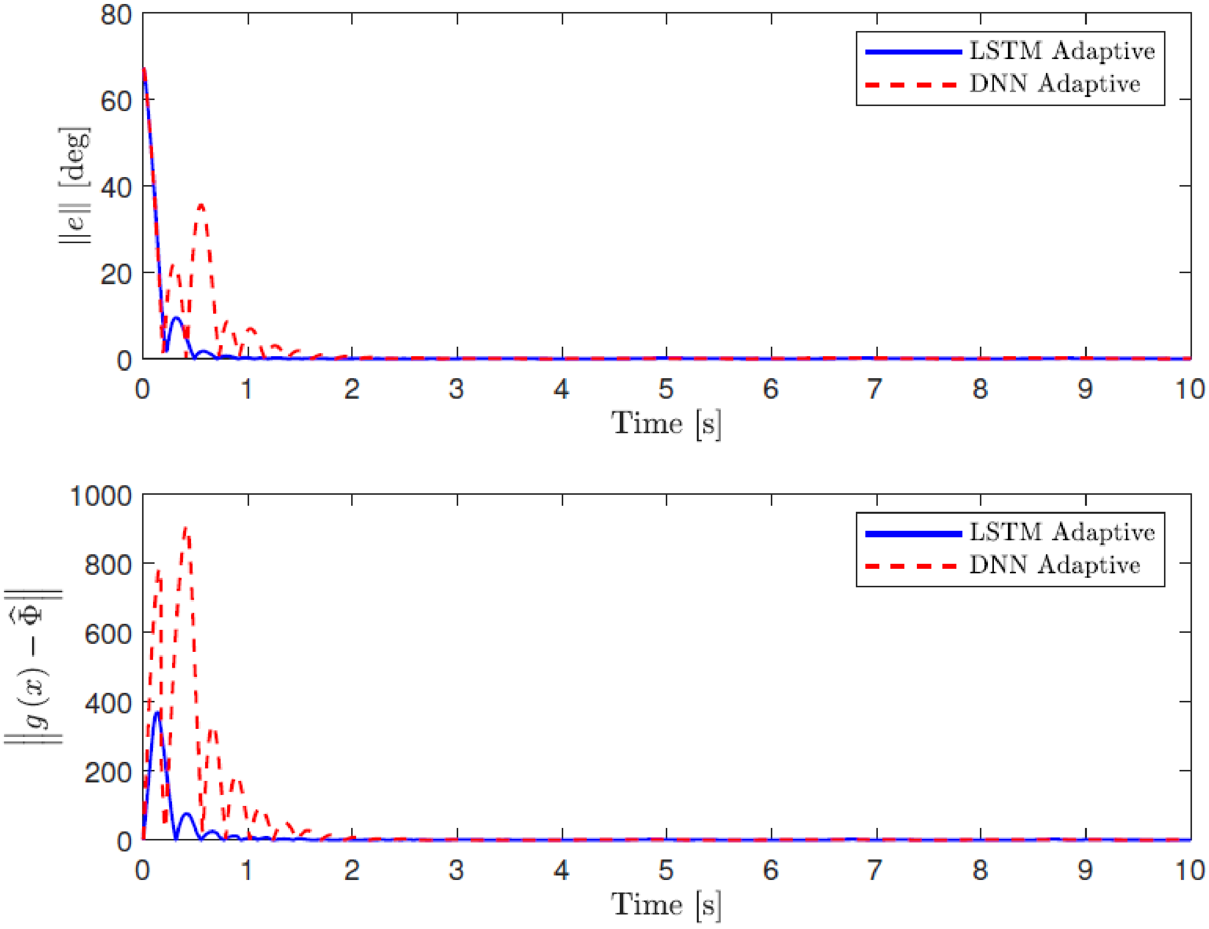
Simulations were performed on a two-link robot manipulator for 10 s.

- Adaptive feedforward DNN used for comparison†
- $q_d(t) = (1 - \exp(-0.1t)) \begin{bmatrix} \frac{\pi}{3}\sin(\frac{\pi}{2}t) \\ \frac{\pi}{3}\sin(\frac{\pi}{2}t) \end{bmatrix}$ [rad]
- $q(0) = [1.0472, -0.5236]^\top$ [rad] and $\dot{q}(0) = [0,0]^\top$ [rad/s]



† O. Patil, D. Le, M. Greene, and W. E. Dixon, "Lyapunov-derived control and adaptive update laws for inner and outer layer weights of a deep neural network," IEEE Control Syst Lett., vol. 6, pp. 1855–1860, 2022.

Simulation Results



Simulation Results

TABLE I.

RMS	Adaptive DNN	Adaptive LSTM	% Improvement
$\|e\|$ [deg]	0.0373	0.0179	52.0542%
$\|g(x)-\widehat{\Phi}\|$	39.2248	8.6457	77.9585%
$\|\tau\|$ [N·m]	23.8107	10.3615	56.4838%

- LSTM provided twofold faster tracking and function approximation error convergence with better transient behavior.
- Twofold and fourfold improvements in the tracking error and function approximation error, respectively with reduced control effort.

Conclusions

- Formulated a continuous-time LSTM model for control of continuous-time systems.
- Developed the Lb-LSTM architecture to capture time-varying effects in the system.
- Stability-driven weight adaptation laws for real-time learning.
- Simulation results yield fourfold improvement in function approximation performance compared to a baseline DNN controller.

Future Work

- Extend developed Lb-LSTM architecture for black-box estimation.
- Develop Lb-LSTM-based observer for uncertain, nonlinear systems.
- Investigate developing Lyapunov-based adaptive architectures for other neural network models
  - Transformers

Thank you

UF UNIVERSITY of FLORIDA

Duke UNIVERSITY

TEXAS The University of Texas at Austin

UC SANTA CRUZ