Received April 5, 2022, accepted April 17, 2022, date of publication April 22, 2022, date of current version April 29, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3169798

Learning an Accurate State Transition Dynamics Model by Fitting Both a Function and Its Derivative

YOUNGHO KIM[1], HOOSANG LEE[1], AND JEHA RYU[1,2], (Member, IEEE)

[1]Robot Reinforcement Learning Laboratory, School of Integrated Technology, Gwangju Institute of Science and Technology, Gwangju 61005, South Korea
[2]Artificial Intelligence Graduate School, Gwangju Institute of Science and Technology, Gwangju 61005, South Korea.

Corresponding author: Jeha Ryu (ryu@gist.ac.kr)

This work was supported by the Development of a Remote Dismantling Training System with Force-Torque Responding Virtual Nuclear Power Plant funded by the Korea Institute of Energy Technology Evaluation and Planning under Project 20201510300280.

ABSTRACT Learning accurate state transition dynamics model in a sample-efficient way is important to predict the future states from the current states and actions of a system both accurately and efficiently in model-based reinforcement learning for many robotic applications. This study proposes a sample-efficient learning approach that can accurately learn a state transition dynamics model by fitting both the predicted next states and their derivatives. The derivatives of the feedforward neural network output (next states) with respect to the inputs (current states and actions) are computed using chain rules. In addition, the effect of the activation functions on the learning derivatives are illustrated via sum of elementary sine functions example and the values are compared with various other activation functions with respect to accuracy. The proposed learning approach exhibits significant improvement in accuracy for both one-step and multi-step prediction cases with a six-degree-of-freedom manipulation robot (UR-10) in both simulation and real environments.

INDEX TERMS Activation function, model-based reinforcement learning, neural network, state transition dynamics model.

I. INTRODUCTION

In model-based reinforcement learning (MBRL) for robotic applications [1], learning accurate state transition dynamics model (STDM) or equations of motion (EOM) is important to predict the future states from the current states and actions of a system accurately. If a learned STDM is very accurate for all states and actions, then well-established control strategies, such as linear quadratic control for deterministic dynamics or linear quadratic Gaussian for stochastic dynamics can be used to obtain an optimal controller. If a learned dynamics model is not accurate, the compounding error increases rapidly resulting in the use of replan strategy such as model predictive control (MPC). The better the accuracy of STDM, the more future steps can be accurately predicted, so the re-planning can be more effective. In addition to the accuracy aspect of learning, sample efficiency (e.g., the use of a smaller number of training trajectories in training) of learning is an important consideration because a safe and cost-effective collection of many meaningful datasets is often difficult in real-world problems. Therefore, training accurate dynamics models with low sample complexity is challenging.

Many MBRL algorithms use naïve black-box approaches with a feedforward neural network (FNN) architecture for learning STDM [2], [3] assuming the Markov decision process in which STDM can predict the next states as outputs given the current states and actions as inputs using supervised learning with a collected rollout dataset, i.e., $(s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$. However, naïve black-box STDM learning usually requires a large meaningful dataset for accurate future state prediction.

Studies on improving accuracy in the naïve black-box STDM are summarized in [4]. Most of these studies used model ensemble methods that averaged the predicted states of several learned neural networks, such as deterministic models [5] or probabilistic models [6] to model the uncertainty and improve accuracy. However, both the prediction speed and memory size of ensemble methods are linearly proportional to the number of models; hence, these methods may not be suitable for real-time control.

The associate editor coordinating the review of this manuscript and approving it for publication was Felix Albu.

In contrast to the naïve black-box STDM approaches that do not need any prior knowledge, the white-box approach learns a known architecture such as the EOM, which can be derived from physics laws. In one such approach, a neural network (NN) may learn unknown dynamics terms [7], [8], such as an inertia matrix, Coriolis matrix, or gravity vector. However, these white-box approaches need additional post-processes to predict the next states. First, the current accelerations must be obtained from the EOM resulting in the NN becoming ill-posed [7]. Second, the accelerations must be integrated numerically to obtain the next states, which requires additional processing time. These post-processes are time-consuming with higher degrees of freedom (DOF), and thus may hinder real-time tasks. In another approach, physical parameters (i.e., mass, length) in an EOM may be learned [9]. However, this system identification approach requires carefully designed suitable control input excitation for good converged parameter identification [7], [10].

Meanwhile, to improve the accuracy of the function approximation in NN theory, there have been studies in which both the function values and derivatives have been fitted simultaneously [11]–[13]. In this "*derivative learning approach*," the NN outputs are differentiated by the NN inputs; then, the error between the ground truth of the derivatives and differentiated values is minimized as an additional loss term. For example, in fluid dynamics simulations, the NN outputs are velocities whereas the inputs are temporal and spatial variables (x, y, and z coordinates) [14]. In this case, the derivatives of NN outputs with respect to the inputs are directly computed by automatic differentiation [18].

Thus, this derivative learning approach may be used to learn STDMs accurately if the ground truth derivatives (velocities and accelerations) are obtainable in robot dynamics and control problems. However, the derivative learning approach in NN theory may not be directly applicable to STDM learning. Unlike previous derivative learning research, e.g., dynamics simulation of differential equations, the outputs of the NN are the next states (poses and velocities) in the STDM whereas the inputs of the NN are the current states and actions. Therefore, the derivatives of the outputs (next states) from the NN with respect to the inputs (current states and actions) are not velocities and accelerations, respectively. However, the ground truth of derivatives that we can utilize are velocities and accelerations obtained by time differentiation of the poses and velocities. Therefore, this derivative mismatch problem should be solved using the derivative learning approach. Moreover, the type of activation function plays an important role in the derivative learning process in terms of accuracy and convergence because some activation functions cannot produce a large target derivative value. However, the effects of various activation functions on derivative learning of NNs have never been investigated in the existing derivative learning approaches.

To solve the above-mentioned problems with the existing derivative learning approaches, this study proposes a novel STDM learning approach that uses derivative information to improve the accuracy of the naïve black-box STDM approaches. To deal with the derivative mismatch problem, the proposed approach uses chain rule to obtain the derivatives of the predicted next states. In addition, we have investigated the effects of the activation functions in the learning process of the derivatives because the activation functions can significantly influence the accuracy and sample efficiency of STDM learning.

To the best of our knowledge, the concept of derivative learning applied to STDM learning has been discussed in this study for the first time. The main contributions of this study can be summarized as follows:

1) To improve the accuracy and sample efficiency of the naïve black-box STDM approaches, we have proposed a novel STDM learning approach in which the chain rule is used to compute the derivatives of the FNN output (next states) with respect to the inputs (current states and actions) to overcome the derivative mismatch problem. Then, both the function and derivative values are used as loss functions.

2) There are no investigations about the effect of activation functions in previous studies on derivative learning. We have assessed the effect of activation functions on derivative learning and empirically compared several activation functions with respect to accuracy.

3) In addition, most of the experiments in previous studies are only demonstrated in simulation. However, we have demonstrated the effectiveness of the proposed approach by learning a real 6-DOF robot STDM while approximating the real accelerations with the desired planned smooth accelerations.

The remainder of this paper is organized as follows: Section II briefly explains the basic concept of fitting derivatives using the Taylor expansion theory. Section III analyzes the effect of activation functions on the derivatives of the NN and compares the performance of various activation functions for an example case of a sum of elementary sine functions. Section IV introduces the proposed derivative learning approach for robot STDM learning. Section V presents the experimental results of the proposed learning approach for the UR-10 manipulator example. The effectiveness of the proposed approach is shown by comparing both one-step and multi-step prediction accuracies with those of the naïve black-box approaches. Finally, Section VI summarizes the conclusions of the study along with future works.

II. BASIC CONCEPT OF DERIVATIVE LEARNING

In this section, we have briefly explained the basic concept of fitting both the function values and derivatives using the Taylor expansion theory to enhance the sample efficiency and accuracy compared to those achievable using the naïve black-box approach. Here, the derivatives of FNN are obtained by differentiating the predicted outputs from the NN with respect to the NN inputs. Using the Taylor expansion, an arbitrary analytic function $f(x)$ can be approximated as a sum of the power series at a local point $a_i \in x$ $(i = 1, \ldots, n)$:

$$f(x) \approx f(a_i) + \left.\frac{\partial f(x)}{\partial x}\right|_{a_i} (x - a_i) + \mathit{High\ order\ term}. \tag{1}$$

In approximating $f(x)$ by *NN(x)*, naïve black-box approaches use only function values ($f(a_i)$) as training data, which minimizes only the zeroth-order term error ($NN(a_i) - f(a_i)$) in (1). This learning usually requires lots of training data for accurate approximation. However, if the ground truth derivatives ($\left.\frac{\partial f(x)}{\partial x}\right|_{a_i}$) are available, the NN can also reduce the first-order term error ($\left.\frac{\partial NN(x)}{\partial x}\right|_{a_i} - f'(a_i)$) by fitting the derivatives of the NN having the slope information at the training data a_i. Fitting both the zeroth-order and first-order terms of the exact function $f(x)$ can enable more accurate function approximation. Therefore, this derivative learning approach can learn more accurate NN models with the same number of training data (trajectory), thus resulting in higher sample efficiency.

Fig. 1 illustrates the learning process of the derivative learning approach for a fifth-order polynomial function ($f(x)$). This figure compares the function-only learning (blue line) with the derivative learning (orange line) processes. The blue line approximates only the ground truth function values on the training data, whereas the orange line approximates both the ground truth function values and derivatives on the training data. As training progresses (as the number of epochs increases), the *derivative learning NN* approaches the ground truth curve (dotted line), whereas the naïve *NN* (*function-only learning*) is still far from the ground truth curve. Therefore, the derivative learning approach can make more accurate predictions.

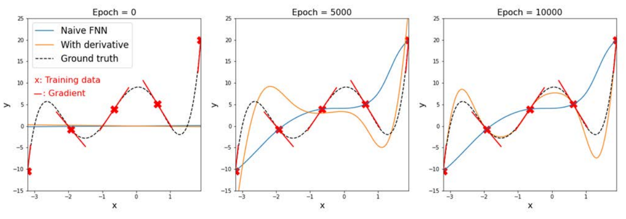


FIGURE 1. Learning processes of a polynomial function using the function-only learning approach (blue) and proposed derivative learning approach (orange) with same number of training data points.

III. EFFECT OF ACTIVATION FUNCTION ON THE DERIVATIVES OF A NEURAL NETWORK

A. EFFECT OF ACTIVATION FUNCTION ON THE DERIVATIVES OF A NEURAL NETWORK

When the derivatives of a NN are used for the training process, the effect of activation functions on those derivatives need to be investigated because NN is a function of the activation function, weight, and bias. In this study, we have used the simple FNN architecture shown in Fig. 2 for basic investigation in which x is the input, $\hat{y}$ is the predicted output, (y, y') are the ground truth function and derivative values, respectively, b is the bias, w is the weight, and g is an activation function.

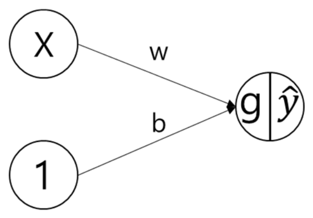


FIGURE 2. Simple FNN architecture.

The predicted function value ($\hat{y}$) is represented as

$$\hat{y} = g(h) = g(wx + b), \quad \text{where } h = wx + b. \tag{2}$$

Then, the derivative ($\frac{\partial \hat{y}}{\partial x}$) of FNN is represented as

$$\frac{\partial \hat{y}}{\partial x} = \frac{\partial g(h)}{\partial h}\frac{\partial h}{\partial x} = \frac{\partial g(h)}{\partial h} w. \tag{3}$$

The derivative loss function ($L_{derivative}$) is defined as the mean squared error (MSE) given by

$$L_{derivative} = \frac{1}{2}\left(\frac{\partial \hat{y}}{\partial x} - y'\right)^2. \tag{4}$$

Note that the derivative of FNN is decomposed into the derivative of activation function ($\frac{\partial g(h)}{\partial h}$) and weight parameter ($w$). To illustrate the effect of activation function on the derivative of FNN, consider a specific activation function such as hyperbolic tangent function *Tanh*, which is commonly used in fully connected layers. This activation function may not fit effectively for the large derivatives since the slope of $Tanh\,(wx + b)$ for a range of weight w and input x (with $b = 0$) asymptotically converges to zero when the weight w and input x go to infinity as shown in Fig. 3. This behavior is mainly due to the exponential decay of $\frac{\partial g(h)}{\partial h}$ as shown in Fig. 4. (b). Therefore, a large weight w alone cannot produce a large value of the derivative of *Tanh.* This behavior may prevent the derivative loss shown in (4) from being minimized to zero if the target derivatives are large. This problem had been pointed out in derivative learning [12] with respect to large target derivatives even though the *Tanh* activation function has been successfully employed to fit functions and their derivatives (e.g., [12]) for years.

On the contrary, if the derivative of an activation function ($\frac{\partial g(h)}{\partial h}$) converges to 1, then the derivative of FNN ($\frac{\partial g(h)}{\partial h} w$) can fit an arbitrary target derivative y' by updating the weight w by the learning process. Therefore, the effects of various activation functions need to be investigated in-depth in the derivative learning of STDM where the target derivatives may be large.

We have investigated three groups of activation functions: (1) sigmoid-like group (*Sigmoid, Tanh*), (2) piecewise-linear group (*ReLU and Leaky ReLU*), and (3) nonlinear group (*Swish, SeLU, ELU, Softplus, and GeLU*). Each activation function and its derivative are shown in Fig. 4.

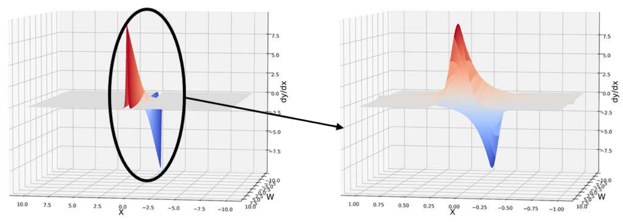


FIGURE 3. **Derivative of *Tanh*($wx + b$), $x, w \in [-10, 10]$.**

Although sigmoid-like group is also a nonlinear function, we have distinguished these nonlinear activation functions into sigmoid-like groups and nonlinear groups.

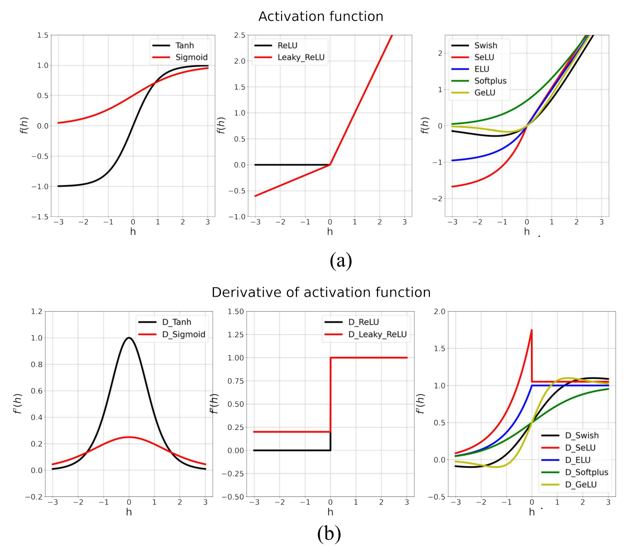


FIGURE 4. **(a) Activation functions and (b) their derivatives.**

From these function and derivative behaviors, both the *Sigmoid* and *Tanh* activation functions have similar effects, as discussed previously. The piecewise-linear group (*ReLU and Leaky ReLU*), which was proposed to solve the gradient vanishing problem in deep layers, shows that the derivative converges to 1 in the positive region. However, the piecewise linear function (in particular, *ReLU*) has a weak expressive capability in a shallow FNN [15]. Thus, it may not approximate functions appropriately in a shallow FNN, which is typically used in STDM learning. In contrast, the nonlinear group (*Swish, SeLU, ELU, Softplus, and GeLU*) the functions are nonlinear and their derivatives also converge to 1. Among these, *Swish, Softplus, and GeLU* activation functions are smooth even in the first order derivative, which is a desirable property in fitting the derivative in derivative learning.

Next, consider the gradient of the derivative loss with respect to the weights in (5):

$$
\begin{aligned}
&\frac{\partial L_{derivative}}{\partial w} \\
&= \left(\frac{\partial \hat{y}}{\partial x} - y'\right) \left(\frac{\partial}{\partial w}\left(\frac{\partial \hat{y}}{\partial x} - y'\right)\right) \\
&= \left(\frac{\partial g\,(h)}{\partial h} w - y'\right) \left(\frac{\partial}{\partial w}\left(\frac{\partial g\,(h)}{\partial h} w\right)\right) \\
&= \left(\frac{\partial g\,(h)}{\partial h} w - y'\right) \left(\frac{\partial}{\partial w}\left(\frac{\partial g\,(h)}{\partial h}\right) w + \frac{\partial}{\partial w}(w)\,\frac{\partial g\,(h)}{\partial h}\right) \\
&= \left(\frac{\partial g\,(h)}{\partial h} w - y'\right) \left(\frac{\partial}{\partial w}\left(\frac{\partial g\,(h)}{\partial h}\right) w + \frac{\partial g\,(h)}{\partial h}\right) \\
&= (\text{error of } \textit{derivatives}) * B(w, h), \qquad (5)
\end{aligned}
$$

where

$$\text{error of derivatives} = \frac{\partial g(h)}{\partial h} w - y'$$
$$B(w, h) = \frac{\partial}{\partial w}\left(\frac{\partial g(h)}{\partial h}\right) w + \frac{\partial g(h)}{\partial h}.$$

This gradient information is used to update the weight through backpropagation. Note that the error of the derivative loss in (5) may not be reflected in the backpropagation process when $B(w,\ h)$ converges to zero (i.e., the gradient vanishing problem) for some activation functions, such as *Tanh* and *Sigmoid* as shown in Fig. 5. The values of $B(w,\ h)$ are also shown for other types of activation functions. This figure shows that the weights cannot be updated when h goes to infinity for sigmoid-like activation functions, whereas the weights can be updated in certain negative regions as well as in the whole positive region for the nonlinear group. These behaviors demonstrate that the term $B(w,\ h)$ can affect the performance of derivative learning.

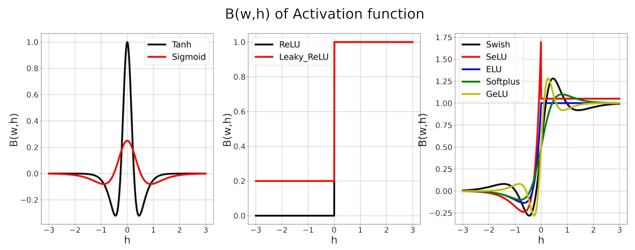


FIGURE 5. ***B*(w, h) for various activation functions.**

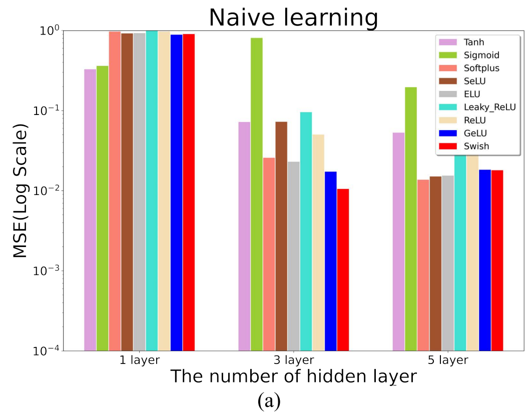
For deep networks, analysis of the effects of activation functions on the derivative of FNN is complex. Therefore, the next subsection shows the numerical behavior of various activation functions in sum of elementary sine functions example to highlight the effect of activation functions with FNN.

B. NUMERICAL EXAMPLE

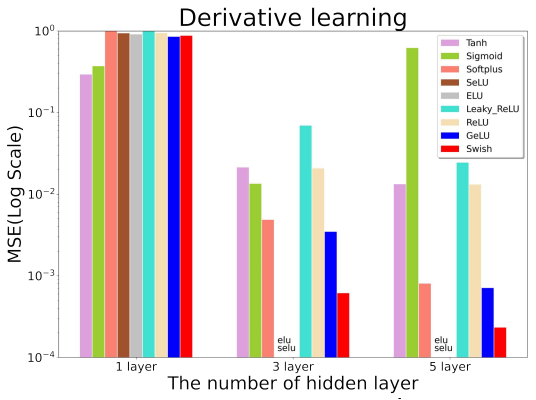
To understand and compare the effects of the activation function on the performance of naïve and derivative learning approaches, we have compared the performance of various activation functions (*Tanh, Sigmoid, ReLU, Leaky ReLU, Softplus, SeLU, ELU, GeLU, and Swish*) with the sum of elementary sine functions given in (6) as a ground truth function. This example function is chosen to illustrate how derivative learning can fit even a very complex function in a sample-efficient way. For a less complex function like robot motions, see section V.

$$
\begin{aligned}
&\mathrm{y} = \sin(\mathrm{x}) + \sin(0.5\mathrm{x}) + \sin(0.1\mathrm{x}); \\
&\quad \text{where} \quad x \in [-30\ \mathrm{pi} \sim 30\ \mathrm{pi}]. \qquad (6)
\end{aligned}
$$

For simplicity of comparison, the FNN model considered here is a FNN with 100 nodes in each hidden layer. MSE was chosen as the performance metric and was compared for the number of hidden layers (1, 3, and 5). The FNN model was trained five times with randomly initialized weights using the Xavier initialization. Then, the best model was selected with the lowest validation loss and was tested using the test data. The dataset (x, y, y') was gathered from (6) by uniform interval sampling from the function domain $x \in [-30\,\mathrm{pi} \sim 30\,\mathrm{pi}]$. The training, validation, and test data corresponded to 100, 1000, and 4000 points, respectively. We intentionally used a small number of training data (only 100) to show the sample efficiency of the proposed approach because a large training dataset may not exhibit the effectiveness of the proposed approach in general. In addition, we have set the derivative loss weight parameter α to 0.001, which is found by trial and error in the range of [0.1,0.01,0.001,0.0001]. Very small parameters have no effect on derivative learning, whereas very large parameters hinder the training of original loss (function value).



(a)



(b)

FIGURE 6. MSE vs activation function for (a) naïve and (b) derivative learning.

For the sum of elementary sine functions example, Fig. 6 shows the 5-average MSE for both naïve and derivative learning results versus several activation functions. The activation function significantly affects derivative learning. In addition, the effect is more significant for deeper layers. Note that the MSE could not be converged with *ELU* and *SeLU* activation functions in derivative learning. We guess this is because the slopes (blue and red lines) at the zero h change abruptly (not smooth) as shown in Fig. 4. (b) (rightmost figure).

From Fig. 6, the *Swish* function exhibits the best accuracy (*GeLU* and *Softplus* are the next best) among all the activation functions used in the derivative learning. This is because of the similar smooth derivative shapes of these three activation functions, as seen in Fig. 4. (b) (rightmost figure).

Fig. 7 shows the typical results of one of the activation functions groups (*Tanh, ReLU, Swish*) with five hidden layers. This figure clearly indicates that the prediction of derivative learning is more accurate than that of the naïve (black-box) learning. Specifically, derivative learning with *Swish* activation function almost perfectly approximates the ground truth values.

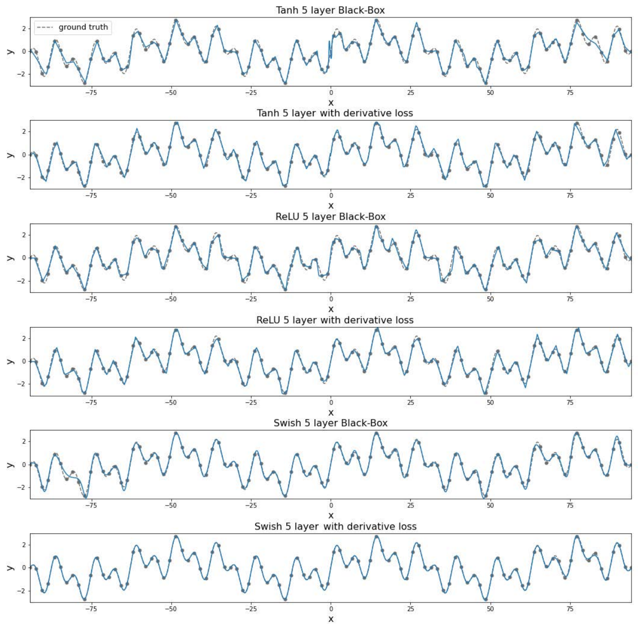


FIGURE 7. Prediction results of the naïve and proposed approaches for different activation functions.

In addition to illustrating matching effects between loss functions for regression (mean squared error (MSE), mean absolute error (MAE), Huber loss) and activation functions in derivative learning, we have also provided simulation results for various combinations between loss and activation functions for the sum of elementary sine functions example with five hidden layers in Table 1, which shows that MSE is the best compared to MAE, MSE, and Huber.

Fig. 8 shows the loss convergence behavior of five randomly initialized weight models with five hidden layers and *Swish* activation function. This figure shows the following. (1) The training and validation losses of the naïve FNN are minimized more slowly compared to those of the proposed approach. (2) The validation loss of the naïve FNN converges

TABLE 1. Loss functions vs activation functions in derivative learning.

5 layer (3-avg)	Tanh	ReLU	Swish
MAE	0.0682	0.0530	0.0962
MSE	**0.0208**	**0.0334**	**0.0006**
Huber	0.0367	0.0468	0.0023

to a higher value (approximately $10E^{-1}$) whereas that of the proposed approach converges to a lower value (approximately $10E^{-2}$), (3) More importantly, the generalization gap [17], difference between training and validation losses is quite small ($10E^{-1}$ vs. $10E^{-2}$) for the proposed approach indicating that it has low variance. In addition, a low training loss means low bias. Therefore, the proposed approach exhibits both low bias and low variance.

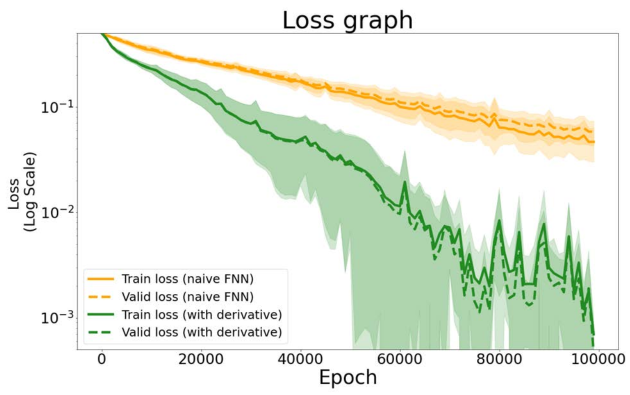


FIGURE 8. Learning curves for naïve and proposed approaches with Swish activation function.

IV. STATE TRANSITION DYNAMICS MODEL LEARNING WITH DERIVATIVE OPTIMIZATION

This section presents the proposed derivative learning approach applied to robot STDM learning for which the target derivatives (ground truth data) must be available. For robot state transition dynamics, the target derivatives are the time derivatives of pose and velocity (i.e., velocity and acceleration: $(\frac{dq_{t+1}}{dt}, \frac{d\dot{q}_{t+1}}{dt})$). However, the derivatives of the network output ($\hat{q}_{t+1}$ & $\hat{\dot{q}}_{t+1}$) with respect to the input (position q_t, velocity $\dot{q}_t$, and action a_t) are $\frac{\partial \hat{q}_{t+1}}{\partial q_t}$, $\frac{\partial \hat{q}_{t+1}}{\partial \dot{q}_t}$, $\frac{\partial \hat{q}_{t+1}}{\partial a_t}$, $\frac{\partial \hat{\dot{q}}_{t+1}}{\partial q_t}$, $\frac{\partial \hat{\dot{q}}_{t+1}}{\partial \dot{q}_t}$, and $\frac{\partial \hat{\dot{q}}_{t+1}}{\partial a_t}$, which are different from the time derivatives. Therefore, the time derivatives need to be computed using the chain rule as derived in (7) and (8) where the subscript t denotes the time variable.

$$\begin{aligned}\frac{\partial \hat{\mathrm{q}}_{\mathrm{t+1}}}{\partial t} &= \frac{\partial \hat{\mathrm{q}}_{\mathrm{t+1}}}{\partial q_t}\frac{\partial \mathrm{q}_{\mathrm{t}}}{\partial t} + \frac{\partial \hat{\mathrm{q}}_{\mathrm{t+1}}}{\partial \dot{q}_t}\frac{\partial \dot{\mathrm{q}}_{\mathrm{t}}}{\partial t} + \frac{\partial \hat{\mathrm{q}}_{\mathrm{t+1}}}{\partial a_t}\frac{\partial \mathrm{a}_{\mathrm{t}}}{\partial t} \\ &= \frac{\partial \hat{\mathrm{q}}_{\mathrm{t+1}}}{\partial q_t}\dot{q}_t + \frac{\partial \hat{\mathrm{q}}_{\mathrm{t+1}}}{\partial \dot{q}_t}\ddot{q}_t + \frac{\partial \hat{\mathrm{q}}_{\mathrm{t+1}}}{\partial a_t}\frac{\partial \mathrm{a}_{\mathrm{t}}}{\partial t} \\ &\approx \frac{\partial \hat{\mathrm{q}}_{\mathrm{t+1}}}{\partial q_t}\dot{q}_t + \frac{\partial \hat{\mathrm{q}}_{\mathrm{t+1}}}{\partial \dot{q}_t}\ddot{q}_t + \frac{\partial \hat{\mathrm{q}}_{\mathrm{t+1}}}{\partial a_t}\frac{\Delta a_t}{\Delta t} \end{aligned} \tag{7}$$

$$\begin{aligned}\frac{\partial \hat{\dot{\mathrm{q}}}_{\mathrm{t+1}}}{\partial t} &= \frac{\partial \hat{\dot{\mathrm{q}}}_{\mathrm{t+1}}}{\partial q_t}\frac{\partial \mathrm{q}_{\mathrm{t}}}{\partial t} + \frac{\partial \hat{\dot{\mathrm{q}}}_{\mathrm{t+1}}}{\partial \dot{q}_t}\frac{\partial \dot{\mathrm{q}}_{\mathrm{t}}}{\partial t} + \frac{\partial \hat{\dot{\mathrm{q}}}_{\mathrm{t+1}}}{\partial a_t}\frac{\partial \mathrm{a}_{\mathrm{t}}}{\partial t} \\ &= \frac{\partial \hat{\dot{\mathrm{q}}}_{\mathrm{t+1}}}{\partial q_t}\dot{q}_t + \frac{\partial \hat{\dot{\mathrm{q}}}_{\mathrm{t+1}}}{\partial \dot{q}_t}\ddot{q}_t + \frac{\partial \hat{\dot{\mathrm{q}}}_{\mathrm{t+1}}}{\partial a_t}\frac{\partial \mathrm{a}_{\mathrm{t}}}{\partial t} \\ &\approx \frac{\partial \hat{\dot{\mathrm{q}}}_{\mathrm{t+1}}}{\partial q_t}\dot{q}_t + \frac{\partial \hat{\dot{\mathrm{q}}}_{\mathrm{t+1}}}{\partial \dot{q}_t}\ddot{q}_t + \frac{\partial \hat{\dot{\mathrm{q}}}_{\mathrm{t+1}}}{\partial a_t}\frac{\Delta a_t}{\Delta t}. \end{aligned} \tag{8}$$

The partial derivatives in (7) and (8) can be computed using automatic differentiation [18]. The measured or approximated target derivatives are used to train each derivative of FNN in the STDM. Note that $(\dot{q}_t, \ddot{q}_t)$ is given (i.e., measured or approximated). However, the action derivatives $(\frac{\partial \mathrm{a}_{\mathrm{t}}}{\partial t})$ are not available (not given or measurable). Therefore, these values need to be approximated by numerical differentiation $(\frac{\partial \mathrm{a}_{\mathrm{t}}}{\partial t} \approx \frac{\Delta a_t}{\Delta t})$. Then, the derivative loss is defined as the MSE between the target derivatives $(\dot{q}_{t+1}, \ddot{q}_{t+1})$ and derivatives of STDM computed in (7) and (8).

The total loss in (9) is then the sum of the two losses (state and derivative losses) with the optimization weight parameter α. Then, the total loss is optimized using Adam optimizer with 0.001 learning rate. The overall architecture of the proposed training framework is summarized in Fig. 9.

$$Loss = Loss_{state} + \alpha Loss_{derivative} \tag{9}$$

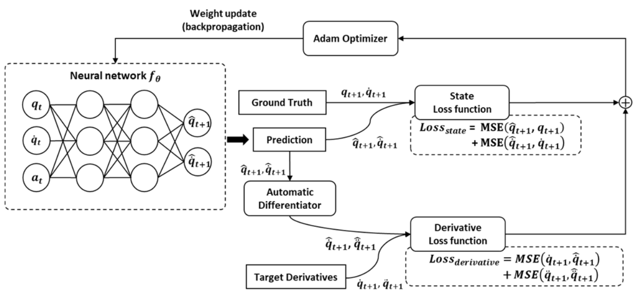

Weight update (backpropagation)
Adam Optimizer
Neural network f_θ
q_t
$\dot{q}_t$
a_t
$\hat{q}_{t+1}$
$\hat{\dot{q}}_{t+1}$
Ground Truth
$q_{t+1}, \dot{q}_{t+1}$
State Loss function
Prediction
$\hat{q}_{t+1}, \hat{\dot{q}}_{t+1}$
$Loss_{state}$ = MSE($\hat{q}_{t+1}, q_{t+1}$) + MSE($\hat{\dot{q}}_{t+1}, \dot{q}_{t+1}$)
Automatic Differentiator
$\hat{\dot{q}}_{t+1}, \hat{\ddot{q}}_{t+1}$
Derivative Loss function
Target Derivatives
$\dot{q}_{t+1}, \ddot{q}_{t+1}$
$Loss_{derivative} = MSE(\dot{q}_{t+1}, \hat{\dot{q}}_{t+1}) + MSE(\ddot{q}_{t+1}, \hat{\ddot{q}}_{t+1})$

FIGURE 9. Proposed training framework for STDM using derivative learning.

V. ROBOT EXAMPLE

To demonstrate the effectiveness of the proposed STDM learning approach, we conducted a series of experiments not only with the simulated 6-DOF UR-10 manipulator to check the feasibility but also with the real 6-DOF UR-10 manipulator to show the practicality of the proposed approach.

A. SIMULATED ROBOT EXPERIMENTS

1) MODEL

The FNNs used for the naïve and proposed approaches consisted of three hidden layers with 100 nodes each, and weight parameter for the derivative loss α was set to 0.001. This FNN model was optimized by trial and error considering both training time and accuracy. The Swish activation function was used in each hidden layer as discussed in the previous section. In addition, all the results were averaged by five randomly initialized models using the Xavier initialization method to neglect initial weight effects.

2) DATA COLLECTION

To check the feasibility of the proposed approach, a nominal robot dynamics model was used in the simulation from which the motion trajectory data (joint angle, angular velocity, and angular acceleration) and action data (joint torque) could be acquired easily. A total of 200 trajectories were generated in joint space by applying joint torques generated by random sine functions as in (10) for 50-time steps using a UR-10 model in MATLAB Simulink. The 200 trajectories with 50-time steps per trajectory were split into three categories: 10, 20, and 40 trajectories (500, 1000, and 2000 data, respectively) for training, 40 trajectories (2000 data) for validation, and 120 trajectories (6000 data) for testing.

$$\tau_i = A_i \sin(f_i t + \emptyset_i) + B_i \quad (i = 1, \ldots, 6), \qquad (10)$$

where

$$A_i = [0, 0.3], \quad f_i = [1, 10], \; B_i = [-1, 1], \; \emptyset_i = [-1, 1].$$

Fig. 10 shows an example of 20 translational (left) and rotational (right) trajectories of training data (blue line) and 120 trajectories of test data (orange line). This figure shows that the large number of test trajectories can cover the end-effector workspace in the task space almost completely.

The prediction performance was evaluated with the best network model having the lowest validation loss of MSE.

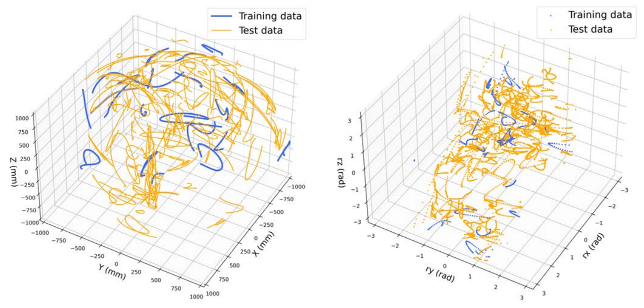


FIGURE 10. Samples of end-effector translational (left) and rotational (right) trajectories in task space.

3) RESULTS

Firstly, we have highlighted the effects of the activation function on prediction error. Fig. 11 shows the one-step prediction errors of joint angles (in radian) and joint angular velocities (in rad/sec) in terms of the average MSE of five trained models for various activation functions when 40 training trajectories and three hidden layers were used. Here, the numbers above the bar graph represent accuracy improvement, which is defined as $\frac{\text{MSE}_{naive} - \text{MSE}_{proposed}}{\text{MSE}_{proposed}} \times 100$. Fig. 11 shows that the proposed derivative learning significantly improves (3-7 times) the prediction accuracy of STDM. This result demonstrates that the proposed approach can significantly improve the prediction accuracy of all the activation functions for a robot like UR-10. This result is qualitatively similar to that for the sum of elementary sine functions presented in Fig. 6. Note that the MSE could not be converged for derivative learning with *ELU* and *SeLU* activation functions as discussed in section III. B.

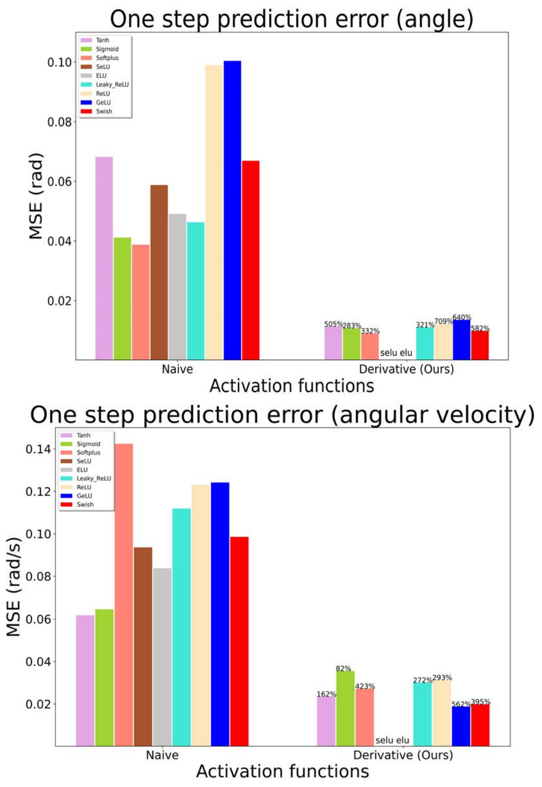


FIGURE 11. MSE vs. activation functions in simulated robot example.

Secondly, we have shown the sample efficiency of the proposed approach through one-step prediction error. Fig. 12 highlights the MSEs (both joint angle and joint angular velocity) of the predicted next states of the naïve and proposed approaches. Although the prediction error of both approaches decreases with an increase in the number of training data, the prediction accuracy of the proposed approach is significantly improved from that of the naïve approach.

Fig. 12 shows that the prediction accuracy of the proposed approach has improved by approximately 11, 5, and 9 times for joint angles and 3, 5, and 6 times for joint angular velocity for 10, 20, and 40 trajectories, respectively. Moreover, the smaller standard deviation of the proposed approach in Fig. 12 indicates that the proposed approach has a more robust prediction capability than the naïve approach. In addition, the validation loss of the proposed approach converges to a lower value than that of the naïve approach, as shown in Fig. 13. This low validation loss also demonstrates that the trained model with a small number (i.e., 40 trajectories) of training data in the proposed approach can generalize to unseen test data (i.e., 120 trajectories) better than the naïve approach.

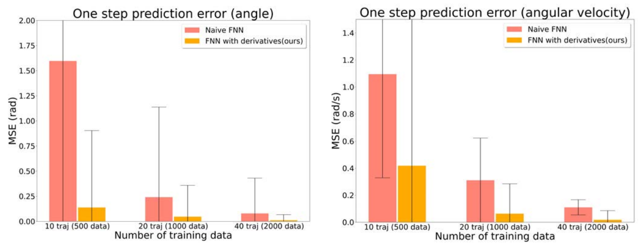


FIGURE 12. MSE vs. number of training data in simulated UR-10.

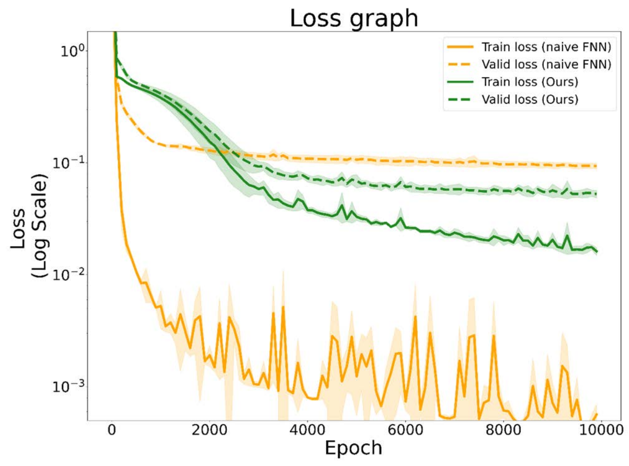


FIGURE 13. Loss behavior vs. training epoch (40 trajectories (2000 data)).

B. REAL ROBOT EXPERIMENTS

The proposed approach requires real acceleration data as the target derivatives in the training process. However, direct measurement of acceleration in the real world is expensive, and the numerical differentiation of velocity is also noisy. To overcome this problem, we have used the accelerations approximated by the desired acceleration planned in the planning stage [19].

1) DATA COLLECTION

In real robots, the training and testing trajectories $(q_t, \dot{q}_t, \ddot{q}_t)$ in joint space should be generated considering potential self-collisions and/or singularity inside workspaces. Moreover, the data should cover the robot workspace as much as possible for generalization capability. Then, once the STDM is learned over the robot workspace, it can be applied in any trajectories. To satisfy these requirements, 100 desired trajectories in a task space for 100 time-steps (thus $100 \times 100 = 10{,}000$ data) were pre-planned in a safe zone using a quintic function with four predefined initial poses and 25 random final poses near each initial state (100 trajectories = 4 initial states multiplied by 25 final states). In this planning, initial and final velocities of 0 along with small magnitude of accelerations were used for quintic function to avoid jerk motions that could induce large errors between the desired and actual accelerations. Fig. 14 shows the planned trajectories.

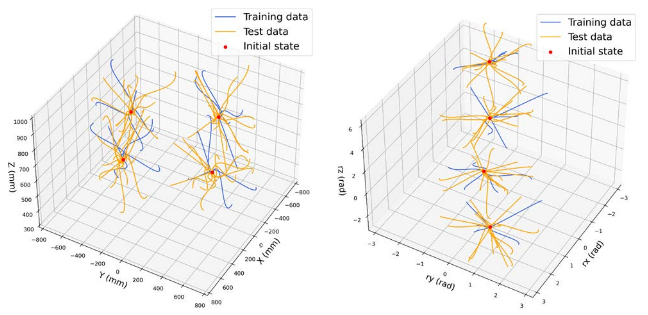


FIGURE 14. Example of end-effector translational (left) and rotational (right) trajectories generated by desired trajectory in a real robot.

To ensure that the real robot follows the planned desired trajectories, a PID controller in the MoveIt ROS package [20] was tuned for small tracking error (MSE of joint angle: 1.818E–5, MSE of joint angular velocity: 0.029). With this setting, the actual states (joint angles and angular velocities) and actions (motor torques) were acquired in a joint space. Fig. 15 shows an example of the actual joint angle (left) and joint angular velocity (right) trajectories of the elbow joint for the planned trajectories with a quintic function in a task space with the real UR10 robot. These trajectories were used along with the desired accelerations to train a NN model. Here, we used an average filter for the measured angular velocity to reduce the noise.

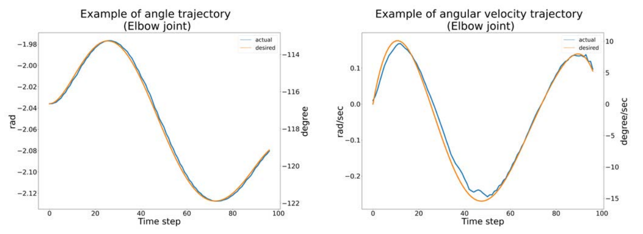


FIGURE 15. Example of desired and actual trajectories of angle (left) and angular velocity (right) of the elbow joint in a real environment.

2) ONE-STEP PREDICTION EXPERIMENT

In this experiment, we used the same network setup as in the simulated robot experiment both for the naïve and proposed approaches with the optimally selected weight parameter $\alpha = 0.001$. Similar to the simulation case, we selected three groups of activation functions such as sigmoid-like, piecewise, and non-linear groups to show the effect of activation functions. In addition, we selected three different small number of training trajectories (4, 12, and 20 trajectories from a total of 100 trajectories) randomly from each of the four initial states (e.g., 1 of 25 trajectories from each initial state for the case of 4 training trajectories) to show the sample efficiency in terms of the number of trajectories in the proposed approach. More specifically, from each of the four initial states, 1, 3, and 5 trajectories were randomly sampled for the training dataset, 5 trajectories for the validation dataset, and the remaining 15 trajectories for the test dataset.

3) MULTI-STEP PREDICTION EXPERIMENT

Multi-step prediction performance is also important for downstream reinforcement learning performance because, for example, if the accuracy of multi-step prediction is high, then future rewards can be known in advance, thus improving the efficiency of reinforcement learning. Therefore, we compared the multi-step performance against state-of-the-art MBRL method [6]. To compare the multi-step prediction accuracy with that of other black-box approaches [6] for UR-10 robot case, we considered four model types (*D*: deterministic model (naïve FNN), *P*: probabilistic model, *DE*: deterministic ensemble model, and *PE*: probabilistic ensemble model) that have the same notations and model training cases appearing in [6] with five models for the ensemble approach. The proposed approach uses four additional model types (*DD*: deterministic model with derivative learning, *PD*: probabilistic model with derivative learning, *DED*: deterministic ensemble model with derivative learning, *PED*: probabilistic ensemble model with derivative learning). In this multi-step experiment, we used 20 trajectories of training data, 20 trajectories of validation data, and 60 trajectories of test data. For a fair comparison, we used the same network size (3 hidden layers, 100 nodes) and *Swish* activation functions for all models.

4) RESULTS

Fig. 16 shows the one-step prediction errors of joint angles (in radian) and joint angular velocities (in rad/ sec) in terms of average MSE of 5 trained models for the activation functions when 20 training trajectories and 3 hidden layers were used. Fig. 16 shows that the proposed derivative learning still improves the prediction accuracy for most of the activation functions even though the amount of improvement is less than that in simulation. Specifically, *Tanh* activation function degraded the prediction accuracy for joint angles whereas *Swish* activation function improved the prediction accuracy the most by approximately 60% for joint angle and 80% for joint angular velocity. This result was anticipated in the analysis of activation function and simulated robot example. In addition, this result shows that the use of target (desired) derivatives in a real robot instead of true derivatives is effective too.

Fig. 17 shows the sample efficiency and robustness of the proposed approach when *Swish* activation function is used. Even though the improvement is not significant compared to the simulation case, this figure still shows that the prediction accuracy of the proposed approach has improved by approximately 17, 7, and 56% for joint angle and 54, 47, and 77% for joint angular velocity for 3 training datasets, respectively (4, 12, and 20 trajectories). Note that the amount of accuracy improvement depends on the number of samples; however, it is always better than that of naïve FNN.

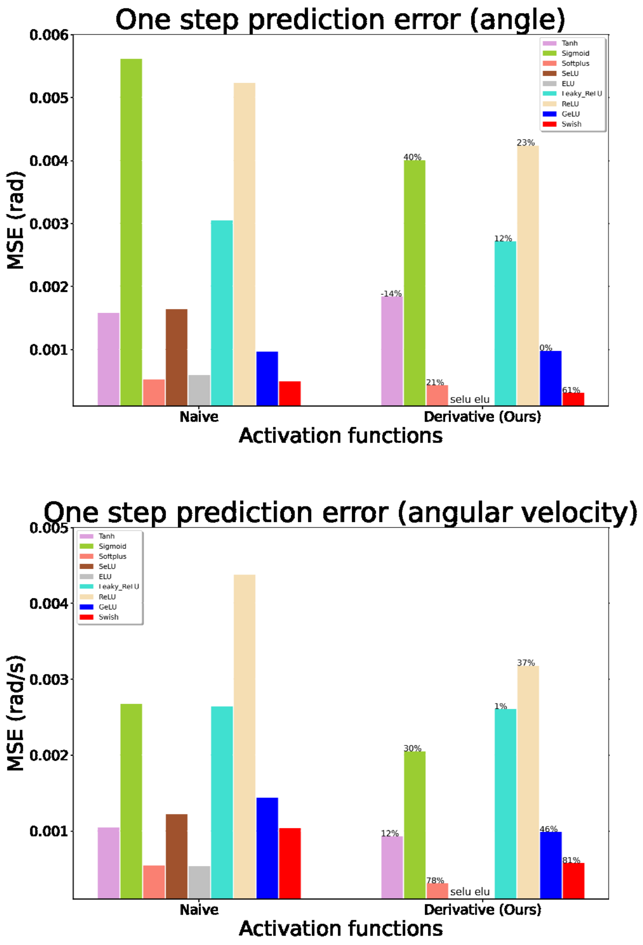


FIGURE 16. MSE vs. activation function in real robot example.

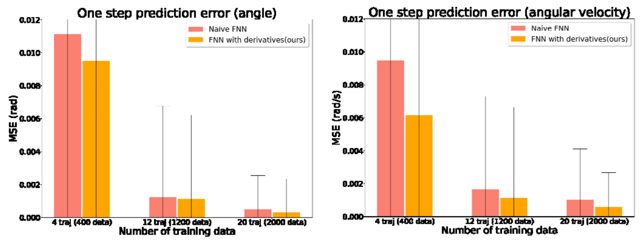


FIGURE 17. One-step prediction vs. number of training data in real robot example.

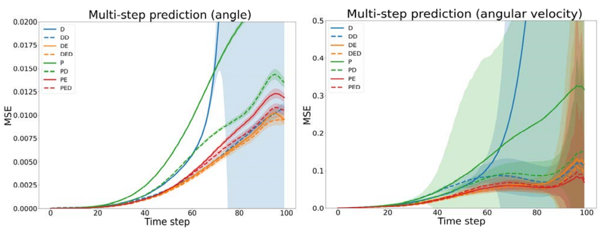


FIGURE 18. (left) Multi-step (up to 100 steps) prediction (angle). (right) Multi-step (up to 100 steps) prediction (angular velocity).

Fig. 18 shows the multi-step (up to 100 steps) prediction accuracy of the abovementioned eight model cases when

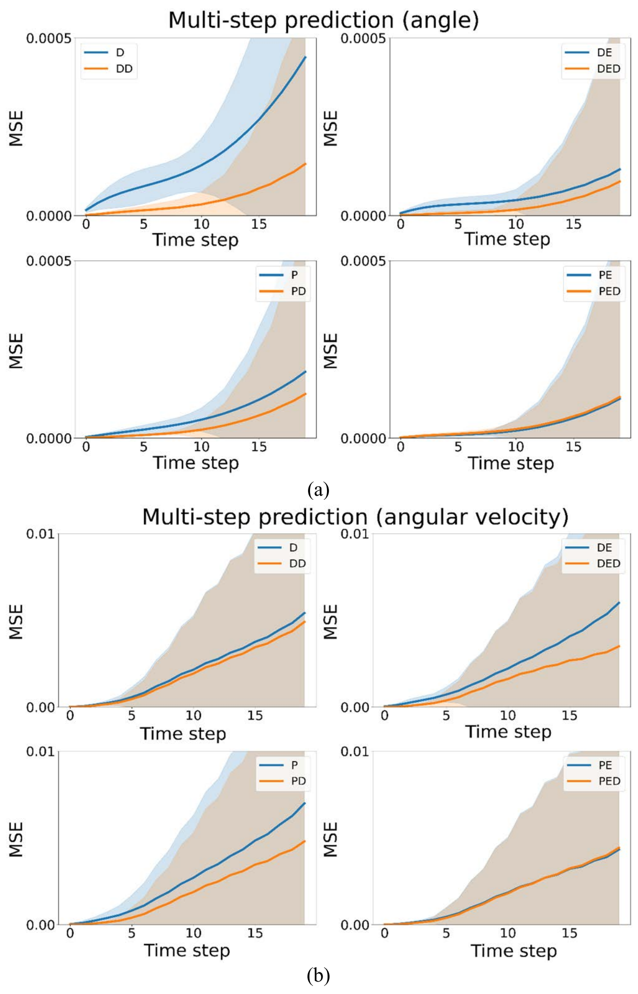


FIGURE 19. (a) Multi-step (up to 20 steps) prediction (angle). (b) Multi-step (up to 20 steps) prediction (angular velocity).

only initial state and action trajectories are given. The solid lines represent the naïve black-box approach (D, DE, P, and PE) whereas the dashed lines represent the proposed approach (DD, DED, PD, and PED), i.e., the derivative loss of the proposed approach is additionally considered in training the naïve black-box approaches. The centerline represents the average MSE, whereas the shaded area represents the standard deviation of 60 test trajectories at each time step. More specifically, Fig. 18 (100-time steps) shows that for very long horizon prediction cases of single models (D and P), the proposed derivative learning approaches (DD: blue dashed line, PD: green dashed line) achieve more stable prediction with small MSEs than the other naive approaches (D: blue solid line, P: green solid line). On the contrary, for the ensemble models, both the proposed (DED, PED) and other approaches (DE, PE) show similar behaviors. However, the computation time of the proposed approach is far less (0.15 s for the five ensemble cases (DE, PE, DED, and PED) vs. 0.033 s for the single model cases (D, P, DD, and PD) for 100-time steps) than that of the ensemble methods because the ensemble methods use multiple models. Therefore, the proposed approach can predict the long-term future accurately with the same inference speed as the single-model approaches (D, P).

To demonstrate the shorter horizon behavior typically used in MPC, MSEs of the angle and angular velocity for 20-time steps are shown in Fig. 19. The angle and angular velocity accuracies of the proposed approach are consistently better than those of the other approaches even though the PE and PED show very similar behaviors.

VI. CONCLUSION

In this study, we proposed a novel STDM learning approach by learning both the function values and its derivatives to improve the STDM accuracy. Additionally, we investigated the effects of the activation functions on the fitting derivatives when a NN learns the derivatives using the proposed derivative learning. We found that the derivative characteristics of the activation functions play an important role in the learning process. For example, the non-linear activation function group, specifically *swish-like*, whose derivative converges to one was suitable for derivative learning. The proposed STDM learning method achieved better accuracy in not only one-step prediction but also multi-step prediction than the naïve black-box approaches for both simulated and real UR-10 manipulators.

In future studies, we will investigate the effect of accuracy of STDM on learning the optimal policy in reinforcement learning settings. A more accurate STDM improves the efficiency of reinforcement learning process.

REFERENCES

[1] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *J. Intell. Robot. Syst. Theory Appl.*, vol. 86, no. 2, pp. 153–173, May 2017.

[2] R. Grzeszczuk, D. Terzopoulos, and G. Hinton, "NeuroAnimator: Fast neural network emulation and control of physics-based models," in *Proc. 25th Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, 1998, pp. 9–20.

[3] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 7579–7586.

[4] A. Plaat, W. Kosters, and M. Preuss, "High-accuracy model-based reinforcement learning, a survey," 2021, *arXiv:2107.08241*.

[5] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model ensemble trust-region policy optimization," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–15.

[6] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 4754–4765.

[7] M. Lutter, C. Ritter, and J. Peters, "Deep Lagrangian networks: Using physics as model prior for deep learning," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–17.

[8] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho, "Lagrangian neural networks," 2020, *arXiv:2003.04630*.

[9] M. Lutter, J. Silberbauer, J. Watson, and J. Peters, "Differentiable physics models for real-world offline model-based reinforcement learning," 2020, *arXiv:2011.01734*.

[10] J. Swevers, C. Ganseman, D. B. Tukel, J. de Schutter, and H. Van Brussel, "Optimal robot excitation and identification," *IEEE Trans. Robot. Autom.*, vol. 13, no. 5, pp. 730–740, Oct. 1997.

[11] V. I. Avrutskiy, "Enhancing function approximation abilities of neural networks by training derivatives," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 2, pp. 916–924, Feb. 2021.

[12] A. Pukrittayakamee, M. Hagan, L. Raff, S. T. S. Bukkapatnam, and R. Komanduri, "Practical training framework for fitting a function and its derivatives," *IEEE Trans. Neural Netw.*, vol. 22, no. 6, pp. 936–947, Jun. 2011.
[13] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Netw.*, vol. 3, no. 5, pp. 551–560, Jan. 1990.
[14] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, Feb. 2019.
[15] L. Lu, "Dying ReLU and initialization: Theory and numerical examples," *Commun. Comput. Phys.*, vol. 28, no. 5, pp. 1671–1706, Jun. 2020.
[16] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017, *arXiv:1710.05941*.
[17] N. S. Keskar, J. Nocedal, P. T. P. Tang, D. Mudigere, and M. Smelyanskiy, "On large-batch training for deep learning: Generalization gap and sharp minima," in *Proc. 5th Int. Learn. Represent. (ICLR)*, 2017, pp. 1–16.
[18] C. C. Margossian, "A review of automatic differentiation and its efficient implementation," 2018, *arXiv:1811.05031*.
[19] A. Colome, D. Pardo, G. Alenya, and C. Torras, "External force estimation during compliant robot manipulation," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2013, pp. 3535–3540.
[20] D. Coleman, I. Şucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: A MoveIt! Case study," *J. Softw. Eng. Robot.*, vol. 5, no. 1, pp. 3–16, May 2014.

YOUNGHO KIM was born in Ulsan, South Korea, in 1992. He received the B.S. degree in mechanical engineering from Konkuk University, Seoul, South Korea, in 2017. He is currently pursuing the integrated M.S. and Ph.D. degrees in intelligent robotics technology with the Gwangju Institute of Science and Technology, Gwangju.

His current research interest includes intelligent robot control with reinforcement learning.

HOOSANG LEE was born in Anyang, South Korea, in 1992. He received the M.S. degree in mechatronics from the Gwangju Institute of Science and Technology, Gwangju, in 2015, and the B.S. degree in robotics from Kwangwoon University, Seoul, South Korea, in 2018. He is currently pursuing the Ph.D. degree in intelligent robotics technology with the Gwangju Institute of Science and Technology.

His current research interests include robot control and robot learning.

JEHA RYU (Member, IEEE) received the B.S. degree in mechanical engineering from Seoul National University, Seoul, South Korea, in 1982, the M.S. degree in mechanical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 1984, and the Ph.D. degree in mechanical engineering from The University of Iowa, Iowa City, IA, USA, in 1991. He is with the School of Mechatronics, Gwangju Institute of Science and Technology (GIST), Gwangju, South Korea, where he is currently a Professor with the School of Integrated Technology and the Artificial Intelligence Graduate School. He has more than 300 of his research articles and reports have been published. His current research interests include basic and applied research on artificial intelligence that can be used to various areas, such as medical and rehabilitation robotics, autonomous vehicles, tele-operation, haptics, virtual reality, augmented reality, intelligent robot controls, and image processing with artificial intelligence.