SN Computer Science manuscript No.
(will be inserted by the editor)

Neural Model Extraction for Model-based Control of a Neural Network Forward Model

Shuhei Ikemoto* · Kazuma Takahara · Taiki Kumi · Koh Hosoda

Received: date / Accepted: date

Abstract Neural networks have been widely used to model nonlinear systems that are difficult to formulate. Thus far, because neural networks are a radically different approach to mathematical modeling, control theory has not been applied to them, even if they approximate the nonlinear state equation of a control object. In this research, we propose a new approach - i.e., neural model extraction, that enables model-based control for a feed-forward neural network trained for a nonlinear state equation. Specifically, we propose a method for extracting the linear state equations that are equivalent to the neural network corresponding to given input vectors. We conducted simple simulations of a two degrees-of-freedom planar manipulator to verify how the proposed method enables model-based control on neural network forward models. Through simulations, where different settings of the manipulators state observation are assumed, we successfully confirm the validity of the proposed method.

Keywords Neural network · Model based control

1 Introduction

In contrast to the conventional approach where robots have been designed to simplify their mathematical modeling, new approaches in robotics, such as biologically inspired and soft robotics, do not prioritize ease of modeling. These approaches aim to elucidate how material properties, actuator characteristics, and mechanical

This work was supported by JSPS KAKENHI Grant Number 18H01410, 19K22875, and 19H01122.

S. Ikemoto
Kyushu Institute of Technology & Research Center for Neuromorphic AI Hardware
2-4, Hibikino, Wakamatsu, Kitakyushu, Fukuoka 808-0196 Japan.
Tel.: +81-93-695-6099
E-mail: ikemoto@brain.kyutech.ac.jp

K. Takahara, T. Kumi, and K. Hosoda
Osaka University, 1-3 Machikaneyama, Toyonaka, Osaka 560-8531 Japan.

structures, which constitute components of a robot, contribute to simply achieving a given task. [1–3].

Although many successful research results have been reported in this new robotics field, the absence of a control theory applicable to these robots has been shelved as a problem. The main reason for this is the difficulty in obtaining their state equations as, in these new approaches, the robots are not typically designed for simpler and more accurate mathematical modeling. Therefore, developing their controllers and analyzing their properties cannot rely on well-established control theory. Machine learning is one of the most promising approaches to develop controllers for complex soft robots without relying on control theory [4–6]. However, from the viewpoint of understanding, controlling, and analyzing the dynamic characteristics of soft and bio-inspired robots, which are all needed to establish the theory of soft and bio-inspired robotics, there is not yet a satisfactory alternative to the role of the conventional control theory in robotics. The new approach dealing with this shelved problem will play a key role in establishing the theory of soft robotics.

Neural networks (NNs) have been used to approximate forward/inverse models of robots based on data of their motions [7, 8]. In addition, it is expected that recent great strides in deep learning techniques will strongly enhance the applicability and usefulness of NN [9, 10]. NNs play an important role in controlling soft and bio-inspired robots where mathematical modeling is difficult but for which it is easy to gather data for learning. NNs have the potential to fully capture the complex dynamic properties of these robots as a complete forward model. Conversely, the comprehension of the features of the soft robots themselves is still lacking because analysis of the NN is not typically available. Thus, even though the NN has a function equivalent to that of a state equation, theories well-established in robotics for state equations can still not be applied for controlling and analyzing systems based on NN models.

Reconsidering why control theory is still not applicable, even if an NN is trained to be functionally equivalent to a state equation, it is arguable that models obtained via NNs are regarded as radically different expressions to classical mathematical models. However, in principle, because an NN is just a parametric model of a nested mapping consisting of a weighted sum and an activation function, the NN model is a possible mathematical expressions of the state equation. Therefore, the actual reason why an NN is not deemed as a mathematical model is thought to be the complexity of the expression.

In this study, we propose a new approach - neural model extraction - that extracts mathematical expressions from NNs to apply model-based theories. To ensure the success of this approach, extracted mathematical expressions should be formed that are easy to handle in the model-based theory, which will be applied. Therefore, we propose a method that efficiently extracts linear models from a basic fully connected feed-forward NN, in accordance with changes in the input vector. The core idea is very simple. The method requires an NN using piece-wise linear activation functions, such as the family of rectified linear units [11–13]. In this case, the nonlinearity of the activation function comes from changing which piece is applied to the weighted sum; i.e., once an input vector specifies which piece is applied to the weighted sum for all hidden units, the mapping of the NN can be written identically as a form of linear regression. We assume that the NN was trained to learn the nonlinear state equation of a robot whose dynamics

are extremely difficult to mathematically model, such as for soft and bio-inspired robots. A linear state equation corresponding to an input vector can be obtained in each step. The linear control theory can then be applied for controlling and analyzing soft robots at that particular moment. This would greatly contribute to the introduction of control theory for soft and bio-inspired robotics.

This paper is organized as follows: In the next section, related works concerning applications of machine learning in robot control are introduced to clarify the novelty of the neural model extraction approach. In Section 3, the proposed method, which extracts linear state equations from an NN trained for a nonlinear state equation, is explained in detail. To validate the proposed method, we conducted several simulations that highlight the advantages. In Section 4, the simulations results are presented. Section 5 discusses important future works and concludes by considering how the proposed method applies to a wide variety of soft robots.

2 Related Work

Controlling a complex robot whose dynamics are nonlinear and difficult to formulate mathematically has been a central issue in the fields of robotics, control, and machine learning. Furthermore, the current emergence of new approaches in robotics, such as bio-inspired robotics and soft robotics, will bring even more attention and importance to this issue. So far, many studies have been conducted and successful robotic applications have been demonstrated. In this section, we introduce related works to clarify where the proposed method is placed relative to other works in this field.

First, we would like to approximately categorize related works into two groups by focusing on where, in the controlled system, a machine learning technique was used. Machine learning techniques have been used as controllers/predictors in controlled systems [7,14]; i.e., the categorization is retrieved from the viewpoint of whether there is a functional approximation regarding a state equation in the control theory. It would be summarized as follows:

1. A method that uses a parameterized controller, evaluates the control result, and then computes the optimal control, without modeling the object itself.
2. A method that obtains a forward model of an object and then determining the control input by another optimization process based on the model.

The first group of works focuses on deriving a controller directly through an optimization process. In particular, this problem is well formulated in the typical setup of reinforcement learning. Advanced reinforcement learning methods have been applied to robot control problems, with remarkable results [15,16]. In addition, according to the recent advent of deep learning techniques, reinforcement learning methods that can use images as a state by using a convolutional neural network, have attracted increasing attention [17–19]. This type of robot control problem, which directly optimizes the controller, can be deemed as a black-box optimization problem. For instance, the covariance Matrix adaptation evolution strategy has recently attracted attention[20,21]. Not only optimizing a controller through trials but also constructing a controller from a dataset, was also intensively studied. This approach is often referred to as inverse model learning [22, 23]. In this approach, regression methods play a crucial role in expressing inverse

models. The Bayesian approach is a sound method for tackling this problem and has shown its generality and applicability in several works[24, 25]. Conversely, NNs have also been widely employed. A distinctive feature of these studies is that the NN is often adopted in parallel with a basic feedback controller. They were trained for the inverse model, which can ideally fully implement feed-forward control of the object [26–28]. Similarly, the approach referred to as active learning demonstrates a method combining reinforcement learning and inverse model learning [29].

The second group of works is based on forward model learning. Forward models are typically limited to predicting future states and thus concurrent learning of controllers based on these forward models has often been addressed [30, 31]. A method that simultaneously learns both the forward and inverse models of a robot is thus considered to be in this group of approaches[32, 33]. These methods use NNs and recent advancements in deep learning involve enhancing this type of method [34]. A different approach for exploiting a forward model in robot control is to use it as a stable and efficient evaluation of an inverse model under training. Model-based reinforcement learning is a typical method used in this class of approaches[35]. In these methods, a forward model of a robot is expressed as a probabilistic process and prediction of the robot with a given policy over a certain period is used to evaluate the policy for numerical optimization. Similar to model-free reinforcement learning techniques, model-based techniques have also attracted increased attention in robotics [36, 37]. Here, deep learning plays a key role in expanding model-based reinforcement learning to a wider set of applications [38, 39]. Another type of method included in this approach is model predictive control based on a forward model approximated by an NN [40, 41]. Similar to model-based reinforcement learning, these methods use forward models to predict future states for several control inputs and compare their results with the desired states. In addition, the impact of recent deep learning developments can be found in this approach. For instance, deep convolutional autoencoders were exploited to extract latent features from images [42, 43] and deep convolutional neural networks were used to directly approximate a forward model for model predictive control [44].

As explained thus far, a variety of approaches have been investigated and it is expected that forward/inverse model learning will allow the control of complex soft and bio-inspired robots. Despite this, understanding the properties of soft and bio-inspired robots and reflecting the knowledge on robotic theory to design and control them are still unaddressed problems. Hence, a theory is required corresponding to control theory for conventional robotics; the approach proposed in this paper, referred to as neural model extraction, focuses on this and can be categorized into the second group because an NN trained for a nonlinear state equation of a robot is required. However, in contrast with the works described above, which mainly focused on how to train an NN, the proposed approach focuses on how to use the NN after training, namely how to extract mathematical models from the NN to apply control theory. Therefore, the approach presented in this paper, which aims at developing the theory and method for soft and bio-inspired robotics based on well-established control theory, is considered to have a unique focus and should be further categorized into a new group.

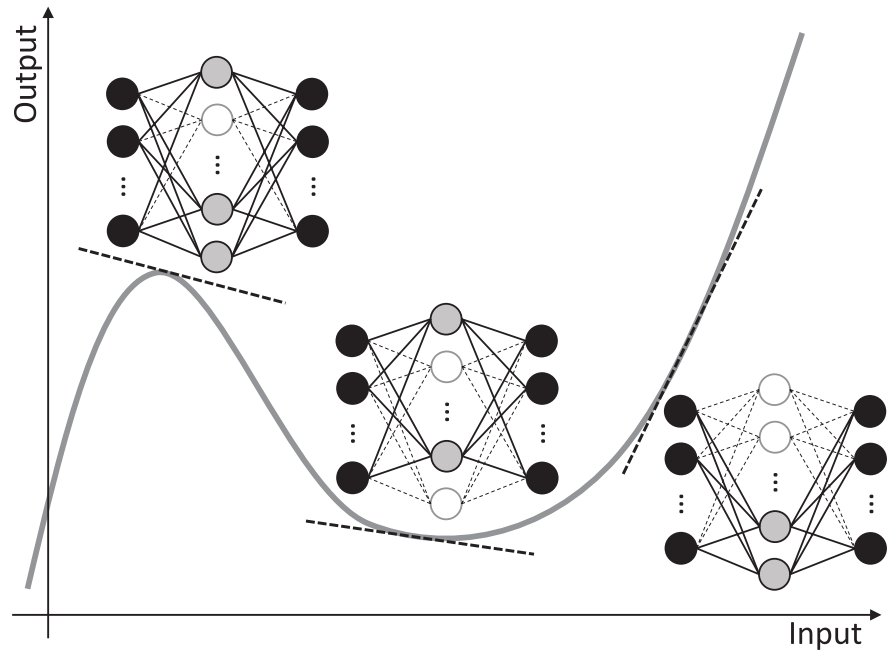

Fig. 1 A schematic representation of the fundamental idea for the extraction of linear state equations. If a piece-wise linear function consisting of N linear functions is employed as an activation function, once an input vector is given, hidden units are equivalent to one of these functions. For example, in this figure, ReLU is assumed to be employed and white and gray circles indicate hidden units that output either zero or non-zero for a particular input, respectively. In this case, white units and the connected weights, depicted by dashed lines, can be ignored. Therefore, the remaining subnetwork governs the entire network at this moment, and can be clearly written in the form of linear regression. This corresponds to obtaining the first-order approximation of the nonlinear state equation around the given input.

3 Proposed Method

In this section, as a concrete example of neural model extraction, we propose a method that extracts an input vector dependent linear state equation from an NN trained over a nonlinear state equation. To achieve robot control using the proposed method, three steps are required.

The first step is generally called forward model learning. In this study, we assume that a forward model corresponds to a nonlinear state equation of a robot. To enable the application of the proposed method, the NN is assumed to be fully connected, feed-forward, and employing a piece-wise linear function as its nonlinear activation function.

The second step aims to extract mathematical models from the NN, which is the core of the proposed method. In particular, it is based on the fact that the activation function can be deemed as a linear function when an input vector determines which part of the piece-wise linear activation function transforms the weighted sum. The term "*activation pattern*" is used to indicate the pattern consisting of which piece of the piece-wise linear activation function is currently used in each of the units. Fig.1 shows the schematic of this step when the rectified linear

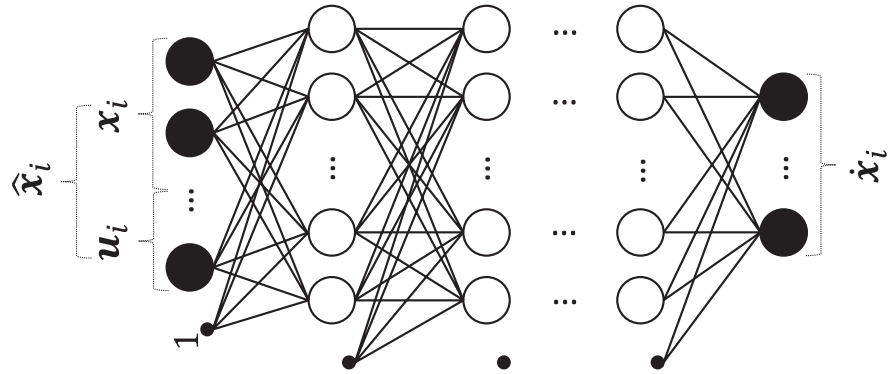

Fig. 2 The structure of the network. To approximate a robot's nonlinear state equation $\dot{x}(t) = f(x(t), u(t))$ by a function $g(\hat{x}(t))$ provided by an NN, definitions of the input and the output vector of the NN are designed to be consistent with those of the nonlinear state equation. In addition, the activation function is assumed to be a piece-wise linear function.

unit is employed as the activation function. Note that this process, which only requires knowing the activation pattern, is simple and has a low computational cost and is thus suitable for being run repeatedly at each variation of the activation pattern.

The third step is to design the controller based on the extracted linear model for a particular instance. In this step, a well-established control theory can be used. For instance, we can use a linear quadratic regulator (LQR) to constitute the feedback controller that determines the control input for a specific instant. In the following subsection, these three steps will be explained in more detail.

3.1 Forward Model Learning

Let $q(t) \in \mathbb{R}^N$ be a column vector that specifies the configuration of a robot. Using $q(t)$, the state is defined as a column vector $x(t) = (q(t)^T, \dot{q}(t)^T)^T \in \mathbb{R}^{2N}$. Defining an input vector as a column vector $u(t) \in \mathbb{R}^M$, an NN for the forward model learning is shaped as receiving a $2N + M$ dimensional column vector consisting of the state and input vectors and outputting a $2N$ dimensional column vector that represents $\dot{x}(t)$.

Fig.2 shows the configuration of the network structure. We denote the input vector concatenating $x(t)$ and $u(t)$ as $\hat{x}(t)$. Assuming an NN provides a mapping $g(\cdot)$, the purpose of the learning is to approximate a robot's nonlinear state equation $\dot{x}(t) = f(x(t), u(t))$ by the function $g(\hat{x}(t))$ such that $f(x(t), u(t)) \approx g(\hat{x}(t))$ holds.

In general, the output of an NN y corresponding to an input $\hat{x}$ is formulated as follows:

$$\begin{aligned} y &= h_{K+1} = W_{out} h_K + b_{out} \\ h_i &= \phi(s_i) \\ s_i &= W_i h_{i-1} + b_i \\ h_0 &= \hat{x} \end{aligned} \tag{1}$$

where K, ϕ, and h_i indicate the number of hidden layers, nonlinear activation function, and outputs of the hidden layers, respectively. In addition, $0 < i \leq K$ holds. As shown in Eq.1, a layer consists of a weighted sum and a nonlinear mapping; $g(\hat{x})$ provided by the NN is just the repetition of this operation. The sole requirement for applying the proposed method is to employ a piece-wise linear function as the activation function.

W_{out}, W_i, b_{out}, and b_i, the so-called weight matrices and biases, are parameters to be optimized. For the sake of a simpler explanation, we introduce Θ, which indicates a set of parameters that sufficiently characterizes an NN, defined by the following:

$$\Theta = \{W_{out}, W_1, \cdots, W_K, b_{out}, b_1, \cdots, b_K\} \tag{2}$$

Assuming that actual state transitions $\dot{x}_i^*, 0 \leq i \leq D$ were observed by feeding inputs u_i to a robot at states x_i, the purpose of learning in the simplest case can be written as follows:

$$\Theta^* = \arg\min_{\Theta} \frac{1}{D+1} \sum_{i=0}^{D} \left(g(\hat{x}_i) - \dot{x}_i^*\right)^2. \tag{3}$$

where the input $\hat{x}_i$ indicates $(x_i^T, u_i^T)^T$ in this case. To improve the accuracy and generalization, additional cost function terms and standardizing coefficients have often been introduced. The proposed method does not prevent using these improvements. Similarly, no limitation on the optimization method is imposed by the use of the proposed method. Therefore, learning can be conducted by using state-of-the-art algorithms to obtain better accuracy and generalization. In the next subsection, the procedure applied after learning, i.e., after solving Eq.3 with adequate accuracy, will be explained.

3.2 Model Extraction

We propose a method that, for a given input vector, extracts an equivalent linear model from an NN. Fig.3 provides a representation of the proposed method, which is going to be explained in this subsection.

An NN is identical to a linear regression model if a linear function is employed as the activation function. Therefore, if a neural network that employs a piece-wise linear function as its activation function is used to approximate a nonlinear state equation of a robot, hidden units must use different pieces of linear functions over changes in the input vector. Let us introduce the term "*activation pattern*" to indicate the pattern consisting of which part of the piece-wise linear activation function is currently used in each unit. The above can then be rephrased stating that the activation pattern has to vary in accordance with changes in the input vector to earn nonlinearity. From a different point of view, this means that the neural network can be written down in the form of a linear regression once the activation pattern is identified.

For the sake of explanation, we first specifically define the activation pattern on the i-th layer ϕ_i^a as follows:

$$\phi_i^a = (\phi_{i,1}^a, \phi_{i,2}^a, \cdots, \phi_{i,N_i}^a)^T \tag{4}$$

$$\phi_{i,j}^a = \left.\frac{\partial \phi(z)}{\partial z}\right|_{z=s_{i,j}}$$

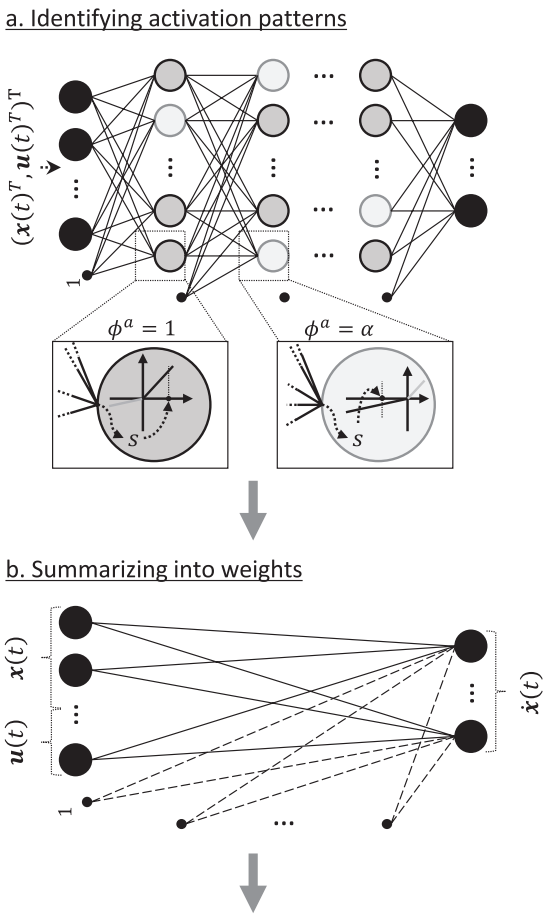

Fig. 3 The schematic representation of the proposed method that, given an input vector, extracts an equivalent linear model from an NN. a) Under a given input vector, the part of the piece-wise linear activation function that each hidden unit belongs to is checked. Here, the leaky ReLU is assumed and dark and light gray units indicate that they are in different parts. b) Once the activation patterns are identified, activation functions are seen as linear functions and interpreted as weights and biases. c) The equivalent linear model at the given input is obtained in equation form

where N_i and $s_{i,j}$ indicate the number of dimensions of s_i and the j-th element of s_i, respectively. For instance, if the Leaky ReLU is employed as the activation function, $\phi^a_{i,j}$ takes either 1 or α according to the sign of $s_{i,j}$. Piece-wise linear functions typically only have a few kinds of derivatives, identifying the activation pattern is straightforward, and can be conducted by simple conditional branching.

Using the activation pattern, the second equation in Eq.1 can be identically written as follows:

$$h_i = \phi_i^a \odot s_i \tag{5}$$

where $\odot$ indicates Hadamard product. Furthermore, Eq.5 is equivalent to:

$$h_i = W_i' h_{i-1} + b_i' \tag{6}$$
$$W_i' = \left[W_{i,j}' \middle| \phi_i^a \odot W_{i,j}, j = 1, \cdots, N_{i-1} \right] \tag{7}$$
$$b_i' = \phi_i^a \odot b_i \tag{8}$$

where $W_{i,j}$ and $W_{i,j}'$ indicate the j-th column of W_i and W_i', respectively. Using Eq.7 and 8, Eq.1 can be differently expressed in the following form:

$$y = g(\hat{x}) = \bar{W}\hat{x}(t) + \bar{b} \tag{9}$$
$$\bar{W} = W_{out}' \prod_{i=1}^{K} W_i' \tag{10}$$
$$\bar{b} = \sum_{i=1}^{K-1} \left(W_{out}' \prod_{j=i+1}^{K} W_j' b_i' \right) + b_{out}. \tag{11}$$

Eq.9-11 show the identical linear regression of the NN. Note that $\bar{W}$ and $\bar{b}$ vary depending on the input $\hat{x}$ because the activation pattern ϕ_i^a depends on s_i.

Considering that the NN was trained by feeding $(x^T, u^T)^T$ as the input $\hat{x}$ and by referring to $\dot{x}^*$ as the target, the NN actually expresses linear state equations as follows:

$$\dot{x}(t) = Ax(t) + Bu(t) + \bar{b} \tag{12}$$
$$\left[A \middle| B \right] = \bar{W} \tag{13}$$

where A and B are submatrices of $\bar{W}$ shaped $2N \times 2N$ and $2N \times M$, respectively. The concrete algorithm of the linear model extraction is shown in Algorithm.1. It is noteworthy that the fifth step is conducted by extremely simple conditional branching if a piece-wise linear activation function is employed.

3.3 Controller Design

A linear model extracted by linear model extraction (LME), as shown in Algorithm 1 can be used to compute a control input. In this study, we simply constitute a controller based on linear control theory every time a new linear state equation is extracted. It is noteworthy that a new linear state equation is obtained almost every time the LME runs with different parameters because the activation pattern is very sensitive to changes in the parameters. Therefore, it could be considered that constructing a controller is required at every control step, as is done for nonlinear model predictive control.

In addition, because the parameters contain the input vector itself, updating the input vector by the constructed controller is likely to change the result of the LME. Algorithm.2 shows the procedure to deal with this problem. In this algorithm, the control input is initialized by that at the previous time step and

Algorithm 1: Linear Model Extraction (LME)

Data: $x(t)$ and $u(t)$
Result: A, B, and $\bar{b}$
$h_0 \leftarrow (x(t)^T, u(t)^T)^T$
$K \leftarrow$ the number of hidden layers
for $i \leftarrow 1$ **to** K **do**
  $s_i \leftarrow W_i h_{i-1} + b_i$
  $\phi_i^a \leftarrow \left.\frac{\partial \phi(z)}{\partial x}\right|_{z=s_i}$
  $N_i \leftarrow$ the number of units on the i-th hidden layer.
  $W_i' \leftarrow \left[W_{i,j}' \middle| \phi_i^a \odot W_{i,j}, j = 1, \cdots, N_{i-1} \right]$
  $b_i' \leftarrow \phi_i^a \odot b_i$
$\bar{W} \leftarrow W_{out}' \prod_{i=1}^{K} W_i'$
$\bar{b} \leftarrow \sum_{i=1}^{K-1} \left(W_{out}' \prod_{j=i+1}^{K} W_j' b_i' \right) + b_{out}$
$A \leftarrow$ the $2N \times 2N$ submatrix at the left side of $\bar{W}$
$B \leftarrow$ the rest $2N \times M$ submatrix at the right side of $\bar{W}$
return A, B, and $\bar{b}$

Algorithm 2: Control based on LME

Data: $x(t)$ and $u(t - \Delta t)$
Result: $u(t)$
$n \leftarrow 0$
$N_L \leftarrow$ an integer indicating the limit of loop
repeat
  if $\tilde{A}, \tilde{B}, \tilde{b}$ *exist* **then**
    $A, B, \bar{b} \leftarrow \tilde{A}, \tilde{B}, \tilde{b}$
  else
    $A, B, \bar{b} \leftarrow \mathrm{LME}(x(t), u(t - \Delta t))$
  Constitute a controller based on $\dot{x}(t) = Ax(t) + Bu(t) + \bar{b}$
  Update $u(t)$ based on the controller
  $\tilde{A}, \tilde{B}, \tilde{b} \leftarrow \mathrm{LME}(x(t), u(t))$
  if $n \geq \mathrm{N_L}$ **then**
    $u(t) \leftarrow$ average of the past few candidates of $u(t)$
    break
  else
    $u_n \leftarrow u(t)$
    $n \leftarrow n + 1$
    continue
until $A, B, D = \tilde{A}, \tilde{B}, \tilde{D}$ holds
return $u(t)$

repeatedly updated based on the results of the LME until a consistent linear model is obtained. Although the loop normally ends in a few iterations, a limit N_L is set, and the average of the past few candidates of $u(t)$ is used if the loop reaches this limit.

Once a linear state equation is obtained, a variety of methods can be applied to derive the controller. Specifically, in this study, we employ optimal feedback control, namely, a linear quadratic regulator (LQR). Here, we would like to briefly explain the procedure.

Let x_d be the desired state. Assuming that the extracted linear state equation $\dot{x}(t) = Ax(t) + Bu(t) + \bar{b}$ globally governs the system, $Ax_d + Bu_d + \bar{b} = 0$ holds when the system converges to the desired state. By defining $e(t) = x(t) - x_d$ and $v(t) = u(t) - u_d$, the state equation is transformed to $\dot{e}(t) = Ae(t) + Bv(t)$. In particular, $u_d = -B^{\dagger}(Ax_d + \bar{b})$ is a constant input vector necessary to keep the state at x_d, where $B^{\dagger}$ indicates the Moore-Penrose pseudo-inverse matrix of B.

Let us define a linear quadratic cost function as follows:

$$J = \int_0^{\infty} e(t)^T Q e(t) + v(t)^T R v(t) dt \tag{14}$$

where $Q \geq 0$ and $R > 0$ hold. The state feedback controller that minimizes the cost function is given by:

$$v(t) = -Ke(t) \tag{15}$$

$$K = (R + B^T P B)^{-1} B^T P A \tag{16}$$

where P is the solution of the following Riccati equation:

$$P = Q + A^T P A - A^T P B (R + B^T P B)^{-1} B^T P A. \tag{17}$$

Substituting the result, the control input at the moment $u(t)$ is given by:

$$u(t) = -Ke(t) + ud. \tag{18}$$

Again, this process for constituting the controller and obtaining $u(t)$ is required at almost every control step.

4 Simulation

As explained thus far, the proposed method does not require a mathematical model of a robot but it is still possible to construct a controller based on well-established control theory. This is thought to be a big advantage for controlling and analyzing soft robots. In this section, we conduct a simulation of a simple two degrees-of-freedom (DOF) planar manipulator to validate the proposed method. Although the robot is simple, by assuming several different sensing setups of the robot, the advantage of the approach is successfully highlighted.

In the following subsections, we explain the dynamics of the robot that we simulate, how different types of training data are sampled, and the control results obtained by the proposed method.

4.1 Dynamics of the 2 DOFs planar manipulator

Fig.4 shows the configuration of the 2 DOFs planar manipulator to be simulated. Thanks to its simplicity, the dynamics can be shortly formulated as follows:

$$\theta = (\theta_1, \theta_2)^T \tag{19}$$

$$\tau = (\tau_1, \tau_2)^T \tag{20}$$

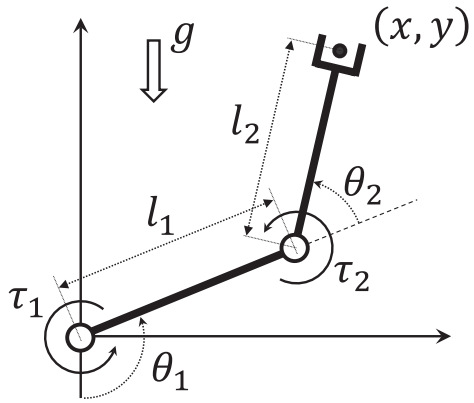

Fig. 4 The configuration of the 2 DoFs manipulator focused in the simulations. Although the end-effector is illustrated, it is not considered in the dynamics. Linkages are assumed to be bars of uniform density.

$$M(\theta)\ddot{\theta} + h(\theta, \dot{\theta}) + g(\theta) = \tau \tag{21}$$

$$M(\theta) = \begin{bmatrix} M_1 & M_2 \\ M_2 & M_3 \end{bmatrix} \tag{22}$$

$$\begin{aligned} M_1 &= m_1 l_{g1}^2 + m_2 l_1^2 + m_2 l_{g2}^2 + I_1 + I_2 \\ &\quad + 2 m_2 l_1 l_{g2} \mathrm{C}_2 \\ M_2 &= m_2 l_{g2}^2 + I_2 + m_2 l_1 l_{g2} \mathrm{C}_2 \\ M_3 &= m_2 l_{g2}^2 + I_2 \end{aligned}$$

$$h(\theta, \dot{\theta}) = \begin{bmatrix} -m_2 l_1 l_{g2} (2\dot{\theta}_1 + \dot{\theta}_2)\dot{\theta}_2 \mathrm{S}_2 \\ m_2 l_1 l_{g2} \dot{\theta}_1^2 \mathrm{S}_2 \end{bmatrix} \tag{23}$$

$$g(\theta) = \begin{bmatrix} (m_2 g l_1 + m_1 g l_{g1})\mathrm{C}_1 + \mathrm{m_2 g l_{g2} rm C_{1,2}} \\ m_2 g l_{g2} \mathrm{C}_{1,2} \end{bmatrix} \tag{24}$$

where I_i is the moment of inertia of the i-th link, and lg_i is the distance from the i-th joint to the center of gravity of the i-th link. In addition, C_1, C_2, $\mathrm{C}_{1,2}$, and S_2 are abbreviations of $\cos\theta_1$, $\cos\theta_2$, $\cos(\theta_1 + \theta_2)$, and $\sin\theta_2$, respectively. Specifically, the physical parameters were configured as $m_i = 1.5$ [Kg], $l_i = 0.35$ [m], $lg_i = l_i/2$ [m], and $I_i = (1/12) m_i l_i^2$ [Kg $\cdot$ m^2] for all $i = 1, 2$. The simulation is conducted based on these equations via the Euler method.

4.2 Forward Model Learning

To highlight the advantage of the proposed method, we assumed three different sensing setups for the robot:

1. Assuming encoders in each joints so that the posture is specified by $\theta = (\theta_1, \theta_1)^T$.
2. Assuming that the position of the end-effector is measured so that the posture is specified by $p = (p_x, p_x)^T$.

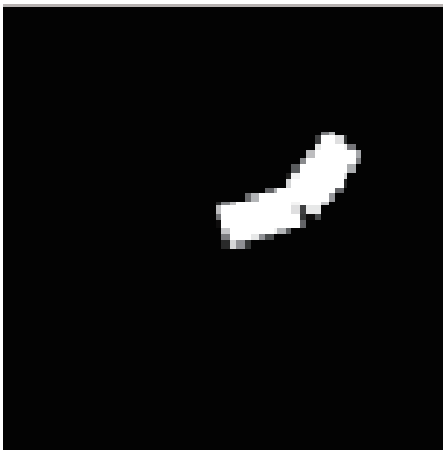

Fig. 5 A top-view image of the 2 DOFs planar manipulator. The image is an 8-bit grayscale 64×64 sized image generated from joint angles θ_1 and θ_2 through OpenGL.

3. Assuming that only the top-view camera image is available so the posture is specified in an unobvious way.

As for the control input, we assume that $u = (\tau_1, \tau_1)^T$ for all three setups.

If joint angles are available by encoders implemented in all joints, the state vector can be defined as $x = (\theta^T, \dot{\theta}^T)^T$. Therefore, an NN forward model trained in this state space provides a mapping from points in the six dimensional input space $\hat{x} = (\theta^T, \dot{\theta}^T, u^T)^T$ to the corresponding points in the four dimensional output space $\dot{x} = (\dot{\theta}^T, \ddot{\theta}^T)^T$, i.e., the same configurations as those shown in 4.1. For the training, 10^6 points were sampled from a uniform distribution defined in intervals $\theta_i \in [0, 135]$ [deg], $\dot{\theta}_i \in [-300, 300]$ [deg/s], and $\tau_i \in [-10, 10]$ [N·m] for all $i = 1, 2$. This means that the data is sampled assuming that the robot arm is a right arm to avoid the appearance of the kinematics redundancy. By initializing θ, $\dot{\theta}$, and τ in Eq.21 using these points, the resultant $\ddot{\theta}$ is computed so that the corresponding target output $\dot{x}^* = (\dot{\theta}^T, \ddot{\theta}^T)^T$ can be obtained. Hereafter, let $\mathcal{D}_J$ be the training data obtained in the state space using joint angles, i.e., $\mathcal{D}_J = \left\{\hat{x}_i, \dot{x}_i^* | 0 \leq i < 10^6\right\}$.

$\mathcal{D}_J$ does not have redundancy and hence the state vector can be defined differently by using the position of the end-effector as $(p^T, \dot{p}^T)^T$. The transformation from $\mathcal{D}_J$ to the training data employing a different state vector is conducted based on the following kinematic relationship:

$$p = \begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} l_1 cos(\theta_1) + l_2 cos(\theta_1 + \theta_2) \\ l_1 sin(\theta_1) + l_2 sin(\theta_1 + \theta_2) \end{bmatrix}. \tag{25}$$

Note that the temporal differentiation of Eq.25 was used for converting $\dot{\theta}$ and $\ddot{\theta}$ into $\dot{p}$ and $\ddot{p}$, respectively. We use $\mathcal{D}_T$ to indicate the data obtained by transforming $\mathcal{D}_J$ as the second setup of the robot.

In addition, as a third setup, we limited the observation to images, thus making the mathematical modeling extremely difficult. Fig.5 shows one of the images taken when the robot arm was randomly driven in the same way as the other setups. A data point in $\mathcal{D}_J$ and $\mathcal{D}_T$ includes acceleration information $\ddot{\theta}$ and $\ddot{p}$ so

at least three images are required to express the point. In particular, for each point in $\mathcal{D}_J$, three kinds of configurations $\theta - \Delta\dot{\theta}$, θ, and $\theta + \Delta\dot{\theta}$, where Δ was set to 0.001, were obtained to transform into three images. Therefore, 3×10^6 images were generated in total. These images were used for training a convolutional autoencoder to derive a two dimensional latent space where the posture of the robot is uniquely expressed and the dynamics can be modeled. To this end, we used a method that trains a convolutional autoencoder with a regulation cost based on a simultaneous forward modeling error of an NN (see [38,45] for the details). By using the trained convolutional autoencoder, encoding and decoding between images and 2 dimensional vectors were made possible.

Fig.6 shows the latent feature figured out from images. Three two dimensional vectors $q(\theta - \Delta\dot{\theta})$, $q(\theta)$, and $q(\theta + \Delta\dot{\theta})$ are obtained from each point in $\mathcal{D}_J$. The third training data $\mathcal{D}_L$ consists of inputs $\hat{x} = (q^T, \dot{q}^T, \tau^T)^T$ and outputs $\dot{x}^* = (\dot{q}^T, \ddot{q}^T)^T$ where $q(t)$, $\dot{q}(t)$, and $\ddot{q}(t)$ were obtained through numerical differentiation. In addition, in parallel with generating training data, corresponding validation data that includes 10^5 points were newly generated in the same manner.

Three kinds of NNs, whose structures are identically configured to have one hidden layer containing 24 units with the leaky ReLU as the activation function, were trained using each of $\mathcal{D}_J$, $\mathcal{D}_T$, and $\mathcal{D}_L$. As shown in Eq.3, the loss function is defined as a simple mean squared error.

Fig.7 shows the history of decreasing loss values in training for both the training and validation data. As a result of training, all NNs were sufficiently trained to show less than 1% in terms of standardized prediction errors. In addition, because loss values evaluated on the validation data appear monotonically decreasing, the learning processes are considered to occur without over-fitting. Therefore, we deemed that each of the NNs were sufficiently trained to approximate the nonlinear state equation of the robot, even if they were based on data employing different expressions.

It is not obvious from Fig.7 but let us roughly consider that the three NNs have almost the same errors in their predictions, i.e., less than 1 %. However, some observations should be made. First, it can be seen that despite all data being generated from systems governed by the same dynamics, these learning processes are largely different. Specifically, learning using $\mathcal{D}_T$, shown as the second subfigure, is slower and takes more than twice the time taken by the other input settings. Second, the validation errors shown in the second and third setup appear to oscillate much more strongly than for the first setup. This implies that the difficulty of forward model learning is strongly dependent not only on the dynamics of the system itself but also on how the dynamics is observed. This observation is strongly relevant for soft robots so will be further discussed in the next section.

4.3 Control by the proposed method

We control the manipulator based on three different NN forward models using the proposed method. To constitute an LQR based on an extracted model, as explained in 3.3, it is necessary to set parameters Q and R. In particular, Q_J, R_J, Q_T, R_T, and Q_L, R_L, defined as follows, were used as Q, R to constitute LQRs

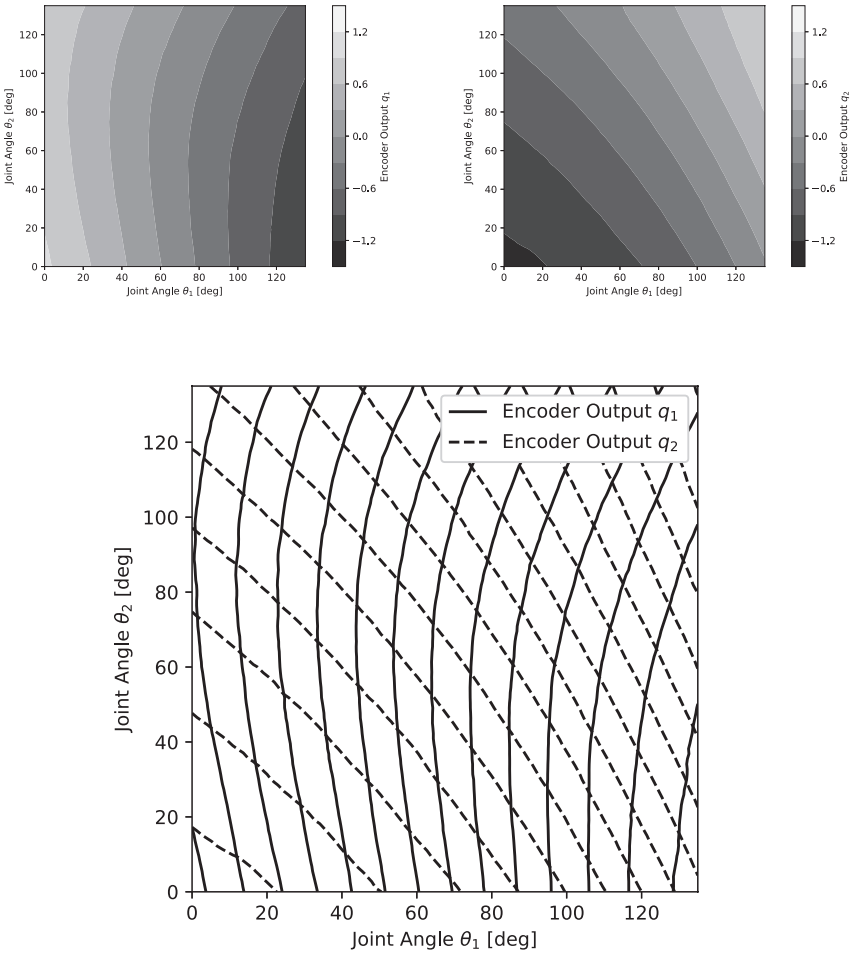

Fig. 6 The latent feature derived from images. For all figures, the horizontal and vertical axes indicate θ_1 and θ_2, respectively. Top-Left: The gradient of the first element of the latent feature. Top-Right: The gradient of the second element of the latent feature. Bottom: Contours of the latent feature $q = (q_1, q_2)^T$. It can be seen that q_1 and q_2 do not directly express θ_1 and θ_2 but q can uniquely specify the posture instead of θ.

based on NNs trained for $\mathcal{D}_J$, $\mathcal{D}_T$, and $\mathcal{D}_L$, respectively.

$$
\begin{aligned}
Q_J &= \mathrm{diag}(5 \times 10^4, 5 \times 10^4, 10, 10) \\
Q_T &= \mathrm{diag}(4 \times 10^4, 4 \times 10^4, 10^{-3}, 10^{-3}) \\
Q_L &= \mathrm{diag}(6 \times 10^2, 6 \times 10^2, 10, 10) \\
R_J &= \mathrm{diag}(10^{-1}, 10^{-1}) \\
R_T &= \mathrm{diag}(10^{-2}, 10^{-2}) \\
R_L &= \mathrm{diag}(1, 1)
\end{aligned}
$$

These parameters were empirically configured through trial and error.

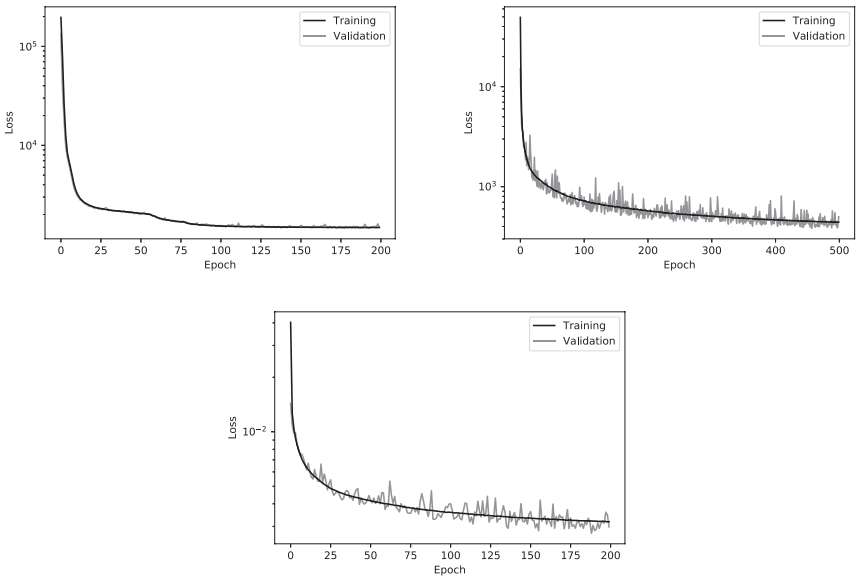

Fig. 7 Learning processes of three kinds of NNs. The first, second, and third subfigures show thee training histories of NNs on $\mathcal{D}_J$, $\mathcal{D}_T$, and $\mathcal{D}_L$, respectively. The horizontal and vertical axes indicate the number of updating steps and the loss value that remains, respectively. A simple mean squared error was adopted as the loss function. The mini-batch size was 64 and the optimization was conducted by the Adam algorithm. The loss values that converged at the end of the training processes reached less than 1% of standard deviations for each data point. As the validation errors showed, over-fitting was not observed.

Fig.8 shows the results of reaching control using the proposed method based on the three NN forward models. Here, the desired and initial postures were set to $\theta_d = (40, 50)^T$ and $\theta(0) = (15, 15)^T$ [deg] for all cases while the results were depicted in the different spaces that were used to specify the posture of the manipulator in the corresponding training data. Therefore, they are all focusing on the same reaching task of the manipulator.

First, from the figure, it can be seen that all results exhibit convergence, indicating that the proposed method can successfully control the manipulator. This provides evidence for the validity of the proposed method, i.e., controllers were achieved in all cases based on linear control theory. For the case where only the end-effector's position is observed and the case where the posture is observed as a top-view image, it is difficult to mathematically describe the dynamics of the manipulator. Therefore, the results shown in the second and third plots clearly show the advantage claimed in this paper.

Second, it can be seen that these results, shown in the second and the third subfigures, are neither as smooth nor as accurate as those obtained by the NN trained for $\mathcal{D}_J$. This may be explained in terms of the aforementioned observation that the validation error histories shown for the second and third input settings appear to oscillate strongly, much more than for the first input setting. The oscillations would imply that the NN forward models do not generalize well and that their control was sometimes conducted based on inaccurate linear state equa-

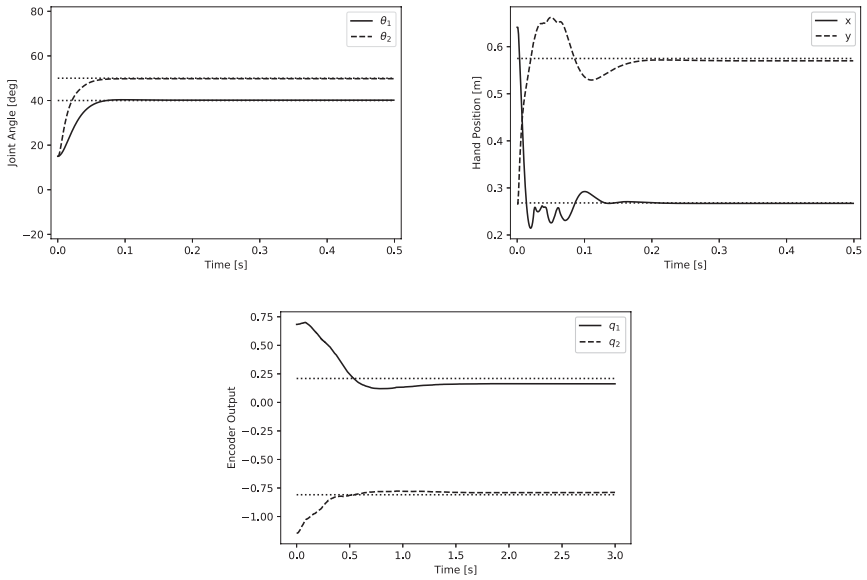

Fig. 8 Results of reaching control by the proposed method. The horizontal and vertical axes indicate the time and values that were employed to express the posture of the manipulator in each training dataset. The desired and initial postures were identically set to $\theta_d = (40, 50)^T$ and $\theta(0) = (15, 15)^T$ [deg] for all cases. The first, second, and third plots show the results of reaching control performed using the NN trained for $\mathcal{D}_J$, $\mathcal{D}_T$, and $\mathcal{D}_L$, respectively. Convergence was confirmed in all cases, validating the proposed method.

tions. Therefore, this suggests that together with the simple loss function some regularization terms will be required to improve the performance of the proposed method.

Third, it should be noted that the cost functions of LQR did not evaluate errors in the joint angles when different state space definitions were employed. For instance, because the NN forward model trained for $\mathcal{D}_T$ employs the definition $x = (p^T, \dot{p}^T)^T$, the cost function of the LQR shown in Eq.14 in this case did not evaluate errors in the joint angles and angular velocities space, but errors in the end-effector's positions and its velocities space. Fig.9 shows how results shown in the second and the third plot of Fig.8 appear in the joint angles space. In particular, the first plot in Fig.9, which corresponds to the second plot of Fig.8, exhibits larger joint angle errors. Not only is the generalization of the NN forward model not good enough but also setting the parameters Q_T and R_T was difficult compared to setting Q_J and R_J; this is thought to be the reason for the reaching performance degradation. In the case where latent features of images were employed to define the state space, it is clear that this problem becomes crucial and Q_L and R_L are difficult to configure to obtain the desired behavior of the manipulator. To address this problem, the method used for finding the latent feature from images, namely the methods proposed in [38,45], would be required to include additional cost functions for training a convolutional autoencoder that remains the intuitive meaning of distances in the latent space. Conversely, this would constitute an

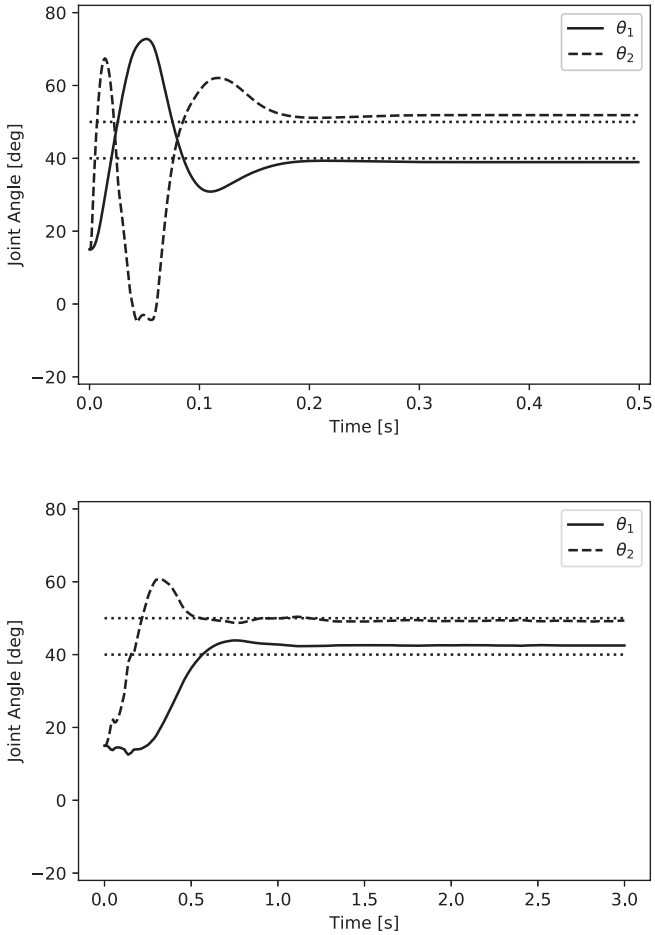

Fig. 9 Time-series of joint angles in reaching control corresponding to the second and third plot of Fig.8. The horizontal and vertical axes indicate the time and angular values, respectively. The first and second plots correspond to the results shown in the second and third subfigures of Fig.8. Errors are larger than those of the corresponding figures in Fig.8 because their cost functions for the LQR did not evaluate errors in the joint angles.

important aspect of the future work of this study, which will be discussed in the next section.

5 Discussion

In Fig.7, the learning processes of the three NNs were largely different, despite all the data being generated from systems governed by the same dynamics. This means that the difficulty of the forward model learning was strongly dependent not only on the dynamics itself but also on how the dynamics were observed to

generate the training data. In contrast to the fact that using an encoder is a typical setup to measure the state of a conventional robot, there is still no standard setup for measuring the state of a soft robot. Therefore, evaluating what kind of sensor information is useful for a soft robot will be an important question in this field. The results shown in Fig.7 would imply the possibility of an evaluation in terms of how easy it is to approximate the dynamics.

From this perspective, it is interesting that the learning process on $\mathcal{D}_L$ is faster than that on $\mathcal{D}_T$. In contrast to $\mathcal{D}_J$ and $\mathcal{D}_T$, which have a physical meaning in each element, $\mathcal{D}_L$ does not have any physical meaning because it was artificially generated through images compressions. On the other hand, as explained, three training data basically contain the same information concerning the posture of the manipulator. Therefore, in principle, converting $\mathcal{D}_T$ to $\mathcal{D}_J$, $\mathcal{D}_J$ to images, and images to $\mathcal{D}_L$ is possible. This means that $\mathcal{D}_L$ can perform as well as $\mathcal{D}_J$ or $\mathcal{D}_T$, regardless of whether it was achievable through training the convolutional autoencoder. In other words, it may be possible to convert sensor signals for defining the state space in a form that is suitable for forward modeling. As mentioned in the previous section, obtaining a latent feature space still provides difficulties in terms of introducing a metric that represents what we intended. However, the above reasoning should suggest an important aspect of future work for expanding the advantages of the proposed approach and method described here.

In this study, we modeled the nonlinear state equation of the robot in continuous-time space. The reason is that a mapping from a current state to the next state $x(t) \rightarrow x(t+1)$ could be almost an identity mapping in the discrete-time model. This slows the learning process and makes expressing the dynamics difficult. Introducing a structure specialized for expressing $x(t+1) = x(t) + \Delta\dot{x}$, such as ResNet [46], is a possible solution to this problem. Even for this case, it is noteworthy that the proposed method is still available and retains the abovementioned advantages.

In this study, to verify the basic properties of the proposed approach, we evaluated only simple examples. Although this example is extremely simple compared to typical soft and bio-inspired robots, the proposed approach of applying model-based control without going through mathematical modeling could be useful for soft and bio-inspired robotics in general. In addition, the results that the state space reconstructed by autoencoders can be used in this approach when it is non-trivial to construct state vectors directly from sensor values and that the difficulty in defining a cost function in the reconstructed state space is useful information for future research in this area. Employing a simulation setting that is more similar to a real soft and bio-inspired robot and conducting an experiment using real soft and bio-inspired robots will be an important future work for this study.

6 Conclusion

In this paper, we proposed a new approach, referred to as neural model extraction, as a method that enables the application of control theory for soft and bio-inspired robots that are difficult to model mathematically. Specifically, as an instance of neural model extraction, we proposed linear model extraction, in which linear regressions are extracted from an NN employing a piece-wise nonlinear activation function. The proposed method can efficiently extract a linear equation that, for that particular input, behaves in a completely identical way to the NN owing to the

property that the derivative of piece-wise linear functions is easily computed for any given input. If an NN is trained for a nonlinear state equation of a robot, the extracted linear models correspond to first-order approximations around inputs to the NN. To validate the proposed method, we conducted a simulation of a simple two DOFs planar manipulator. In particular, we focused on three setups using different sensors to measure the posture of the robot. Through the simulation, the validity was confirmed and the advantage was highlighted. This distinctive feature is beneficial, especially for soft and bio-inspired robots, and its importance is thought to increase in the future.

Conflict of interest

The authors declare that they have no conflict of interest.

Compliance with Ethical Standards

Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. R. Pfeifer, M. Lungarella, F. Iida, Science **318**(5853), 1088 (2007). URL http://science.sciencemag.org/content/318/5853/1088.abstract
2. D. Rus, M.T. Tolley, Nature **521**, 467 (2015). DOI 10.1038/nature14543. URL https://doi.org/10.1038/nature14543
3. C. Laschi, B. Mazzolai, M. Cianchetti, Science Robotics **1**(1) (2016). URL http://robotics.sciencemag.org/content/1/1/eaah3690.abstract
4. G. Martius, R. Hostettler, A. Knoll, R. Der, in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016), pp. 767–773. DOI 10.1109/IROS.2016.7759138
5. A. Gupta, C. Eppner, S. Levine, P. Abbeel, (2016), pp. 3786–3793. DOI 10.1109/IROS.2016.7759557
6. M. Ishige, T. Umedachi, T. Taniguchi, Y. Kawahara, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018)
7. K.J. Hunt, D. Sbarbaro, R. bikowski, P.J. Gawthrop, Automatica **28**(6), 1083 (1992). DOI https://doi.org/10.1016/0005-1098(92)90053-I. URL http://www.sciencedirect.com/science/article/pii/000510989290053I
8. L. Jin, S. Li, J. Yu, J. He, Neurocomputing **285**, 23 (2018). DOI https://doi.org/10.1016/j.neucom.2018.01.002. URL http://www.sciencedirect.com/science/article/pii/S0925231218300158
9. Y. LeCun, Y. Bengio, G. Hinton, Nature **521**, 436 (2015). DOI 10.1038/nature14539. URL https://doi.org/10.1038/nature14539
10. H.A. Pierson, M.S. Gashler, Advanced Robotics **31**(16), 821 (2017). DOI 10.1080/01691864.2017.1365009. URL https://doi.org/10.1080/01691864.2017.1365009
11. V. Nair, G.E. Hinton, in *Proceedings of the 27th International Conference on International Conference on Machine Learning* (Omnipress, 2010), ICML'10, pp. 807–814. URL http://dl.acm.org/citation.cfm?id=3104322.3104425
12. A.L. Maas, A.Y. Hannun, A.Y. Ng, in *ICML Workshop on Deep Learning for Audio, Speech and Language Processing* (2013)
13. K. He, X. Zhang, S. Ren, J. Sun, CoRR **abs/1502.01852** (2015). URL http://arxiv.org/abs/1502.01852
14. D. Nguyen-Tuong, J. Peters, Cognitive Processing **12**(4), 319 (2011). DOI 10.1007/s10339-011-0404-1. URL https://doi.org/10.1007/s10339-011-0404-1

15. J. Peters, S. Schaal, Neural Networks **21**(4), 682 (2008). DOI https://doi.org/10.1016/j.neunet.2008.02.003. URL http://www.sciencedirect.com/science/article/pii/S0893608008000701
16. M. Gaeta, V. Loia, S. Miranda, S. Tomasiello, Applied Mathematical Modelling **40**(21), 9183 (2016). DOI https://doi.org/10.1016/j.apm.2016.05.049
17. T.d. Bruin, J. Kober, K. Tuyls, R. Babuka, in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016), pp. 3947–3952. DOI 10.1109/IROS.2016.7759581
18. S. Gu, E. Holly, T. Lillicrap, S. Levine, in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (2017), pp. 3389–3396. DOI 10.1109/ICRA.2017.7989385
19. T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, S. Levine, in *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018* (2018), pp. 6244–6251. DOI 10.1109/ICRA.2018.8460756. URL https://doi.org/10.1109/ICRA.2018.8460756
20. F. Stulp, O. Sigaud, in *Proceedings of the 29th International Coference on International Conference on Machine Learning* (Omnipress, USA, 2012), ICML'12, pp. 1547–1554. URL http://dl.acm.org/citation.cfm?id=3042573.3042771
21. F. Stulp, P.Y. Oudeyer, Paladyn **3**(3), 128 (2012). DOI 10.2478/s13230-013-0108-6. URL https://doi.org/10.2478/s13230-013-0108-6
22. D. Nguyen-Tuong, J. Peters, M. Seeger, B. Schölkopf, in *Advances in Computational Intelligence and Learning: Proceedings of the European Symposium on Artificial Neural Networks*. Max-Planck-Gesellschaft (d-side, Evere, Belgium, 2008), pp. 13–18
23. O. Sigaud, C. Salan, V. Padois, Robotics and Autonomous Systems **59**(12), 1115 (2011). DOI https://doi.org/10.1016/j.robot.2011.07.006. URL http://www.sciencedirect.com/science/article/pii/S092188901100128X
24. S. Schaal, C.G. Atkeson, S. Vijayakumar, in *Proceedings in IEEE International Conference on Robotics and Automation*, vol. 1 (2000), vol. 1, pp. 288–293 vol.1. DOI 10.1109/ROBOT.2000.844072
25. D. Nguyen-Tuong, M. Seeger, J. Peters, Advanced Robotics **23**(15), 2015 (2009). DOI 10.1163/016918609X12529286896877. URL https://doi.org/10.1163/016918609X12529286896877
26. H. Miyamoto, M. Kawato, T. Setoyama, R. Suzuki, Neural Networks **1**(3), 251 (1988). DOI https://doi.org/10.1016/0893-6080(88)90030-5. URL http://www.sciencedirect.com/science/article/pii/0893608088900305
27. M. Katayama, M. Kawato, in *Neural Information Processing Systems* (1990)
28. T. Waegeman, F. wyffels, B. Schrauwen, IEEE Transactions on Neural Networks and Learning Systems **23**(10), 1637 (2012). DOI 10.1109/TNNLS.2012.2208655
29. B. Settles, Synthesis Lectures on Artificial Intelligence and Machine Learning **6**(1), 1 (2012)
30. M.I. Jordan, D.E. Rumelhart, Cognitive Science **16**(3), 307 (1992). DOI https://doi.org/10.1016/0364-0213(92)90036-T. URL http://www.sciencedirect.com/science/article/pii/036402139290036T
31. A. Dearden, Y. Demiris, in *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005), IJCAI'05, pp. 1440–1445. URL http://dl.acm.org/citation.cfm?id=1642293.1642521
32. D.M. Wolpert, M. Kawato, Neural Networks **11**(7), 1317 (1998). DOI https://doi.org/10.1016/S0893-6080(98)00066-5. URL http://www.sciencedirect.com/science/article/pii/S0893608098000665
33. M. Haruno, D.M. Wolpert, M. Kawato, Advances in Neural Information Processing Systems pp. 31–37 (1999)
34. A. Lambert, A. Shaban, A. Raj, Z. Liu, B. Boots, 2018 IEEE International Conference on Robotics and Automation (ICRA) pp. 675–682 (2018)
35. A.S. Polydoros, L. Nalpantidis, Journal of Intelligent & Robotic Systems **86**(2), 153 (2017). DOI 10.1007/s10846-017-0468-y
36. T. Hester, M. Quinlan, P. Stone, in *2012 IEEE International Conference on Robotics and Automation* (2012), pp. 85–90. DOI 10.1109/ICRA.2012.6225072
37. D. Martnez, G. Aleny, C. Torras, in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015), pp. 6422–6427. DOI 10.1109/IROS.2015.7354295
38. M. Watter, J. Springenberg, J. Boedecker, M. Riedmiller, in *Advances in neural information processing systems* (2015), pp. 2746–2754

39. A. Nagabandi, G. Kahn, R.S. Fearing, S. Levine, 2018 IEEE International Conference on Robotics and Automation (ICRA) pp. 7559–7566 (2018)
40. D. Soloway, P.J. Haley, in *Proceedings of the 1996 IEEE International Symposium on Intelligent Control* (1996), pp. 277–282. DOI 10.1109/ISIC.1996.556214
41. B. Akesson, H. Toivonen, Journal of Process Control **16**(9), 937 (2006). DOI https://doi.org/10.1016/j.jprocont.2006.06.001. URL http://www.sciencedirect.com/science/article/pii/S0959152406000618
42. K. Kashima, in *IEEE International Conference on Decision and Control* (2016), pp. 5750–5755. DOI 10.1109/CDC.2016.7799153
43. M. Wang, H.X. Li, W. Shen, in *International Joint Conference on Neural Networks* (2016), pp. 3180–3186. DOI 10.1109/IJCNN.2016.7727605
44. I. Lenz, R.A. Knepper, A. Saxena, in *Robotics: Science and Systems* (2015)
45. K. Takahara, S. Ikemoto, K. Hosoda, in *The 15th International Conference on Intelligent Autonomous Systems* (2018)
46. K. He, X. Zhang, S. Ren, J. Sun, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). DOI 10.1109/CVPR.2016.90