MPC of Uncertain Nonlinear Systems with Meta-Learning for Fast Adaptation of Neural Predictive Models

Yan, Jiaqi; Chakrabarty, Ankush; Rupenyan, Alisa; Lygeros, John

TR2024-115   August 29, 2024

Abstract

In this paper, we consider the problem of reference tracking in uncertain nonlinear systems. A neural State-Space Model (NSSM) is used to approximate the nonlinear system, where a deep encoder network learns the nonlinearity from data, and a state-space component captures the temporal relationship. This transforms the nonlinear system into a linear system in a latent space, enabling the application of model predictive control (MPC) to deter- mine effective control actions. Our objective is to design the optimal controller using limited data from the target system (the system of interest). To this end, we employ an implicit model-agnostic meta-learning (iMAML) framework that leverages information from source systems (systems that share similarities with the target system) to expedite training in the target system and enhance its control performance. The framework consists of two phases: the (offine) meta-training phase learns a aggregated NSSM using data from source systems, and the (online) meta-inference phase quickly adapts this aggregated model to the target system using only a few data points and few online training iterations, based on local loss function gradients. The iMAML algorithm exploits the implicit function theorem to exactly compute the gradient during training, without relying on the entire optimization path. By focusing solely on the optimal solution, rather than the path, we can meta-train with less storage complexity and fewer approximations than other contemporary meta-learning algorithms. We demonstrate through numerical examples that our proposed method can yield accurate predictive models by adaptation, resulting in a downstream MPC that outperforms several baselines.

International Conference on Automation Science and Engineering (CASE) 2024

MPC of Uncertain Nonlinear Systems with Meta-Learning for Fast Adaptation of Neural Predictive Models

Jiaqi Yan, Ankush Chakrabarty, Alisa Rupenyan, and John Lygeros

Abstract— In this paper, we consider the problem of reference tracking in uncertain nonlinear systems. A neural State-Space Model (NSSM) is used to approximate the nonlinear system, where a deep encoder network learns the nonlinearity from data, and a state-space component captures the temporal relationship. This transforms the nonlinear system into a linear system in a latent space, enabling the application of model predictive control (MPC) to determine effective control actions. Our objective is to design the optimal controller using limited data from the target system (the system of interest). To this end, we employ an implicit model-agnostic meta-learning (iMAML) framework that leverages information from source systems (systems that share similarities with the target system) to expedite training in the target system and enhance its control performance. The framework consists of two phases: the (offline) meta-training phase learns a aggregated NSSM using data from source systems, and the (online) meta-inference phase quickly adapts this aggregated model to the target system using only a few data points and few online training iterations, based on local loss function gradients. The iMAML algorithm exploits the implicit function theorem to exactly compute the gradient during training, without relying on the entire optimization path. By focusing solely on the optimal solution, rather than the path, we can meta-train with less storage complexity and fewer approximations than other contemporary meta-learning algorithms. We demonstrate through numerical examples that our proposed method can yield accurate predictive models by adaptation, resulting in a downstream MPC that outperforms several baselines.

I. Introduction

Optimal control for unknown nonlinear systems has been a long-standing challenge in various applications such as robotics, manufacturing systems, and so on [?], [?], [?]. Different learning-based approaches have been explored, such as reinforcement learning, adaptive dynamic programming, and stochastic optimization-based control [?], [?], [?]. Recently, the neural state-space model (NSSM) has gained popularity as an effective tool for tackling nonlinear control problems [?], [?], [?]. This model enhances traditional state-space models by employing neural networks to capture complex and nonlinear relationships within the system. Specifically, the neural network component empowers the model to

learn from data and transform the original state-space into a latent space where the dynamics are reasonably approximated by a linear system [?]. This architecture is grounded in Koopman operator theory, demonstrating that any nonlinear system can be lifted to an infinite-dimensional linear system in a latent space [?]. For the sake of tractable computations, the latent state of NSSMs is chosen to be of finite size [?].

Although NSSMs show good performance in approximating nonlinear systems, their training usually requires a large dataset. In situations where data is scarce, constructing an accurate model and developing an efficient learning-based controller becomes challenging. This difficulty often arises when collecting data from the system of interest, referred to as the target system, is arduous or expensive. In response to this challenge, recent efforts, while not specifically within the domain of NSSMs, aim to leverage data collected from source systems that share similarities with the target system [?], [?], [?]. As an example, the target system could be a physical system, while the source system may be a numerical model or a digital twin of the target system, or a system with modified physical parameters that is easily accessible, allowing for the collection of large amounts of data. The objective is to use this shared dataset to pre-train a model, then fine-tune it on the target system with limited data.

In this paper, we extend the approach of leveraging knowledge acquired from similar systems to expedite training of the target system in the context of NSSM. Different from existing works [?], [?] that use a transfer learning framework, we propose a method by using meta-learning. The distinction between transfer learning and meta-learning is elucidated in [?], and differences in empirical performance between the two classes of methods have been highlighted in [?].

Meta-learning has recently gained widespread application in control and robotics, addressing challenges such as system identification, parameter estimation, and adaptive control [?], [?], [?], [?], [?]. In most of these works, model-agnostic meta-learning (MAML) based solutions are developed, which involves solving a bilevel optimization problem in the meta-training stage. The inner loop of MAML involves calculating and propagating derivatives of the training loss function along the full optimization path (which is the same as the number of inner-loop updates/steps). The outer loop then accumulates information from these inner-loop up-

dates and computes an outer-loop gradient direction. By repeating these two loops, the MAML learner is expected to asymptotically converge to a set of neural weights from which rapid adaptation is possible to any task within the distribution of training tasks. Choosing a large number of inner-loop steps, while useful to understand good outer-loop directions, incurs high memory complexity, and is not practical for a large number of inner-loop adaptation steps. Therefore, existing approaches typically propose approximations to obtain a more efficient solution or to allow for a large number of inner-loop steps. Unfortunately, these approximations are sometimes oversimplifications or valid for specific activation functions (e.g., first-order MAML exploits ReLU properties), and thus, can harm quality. One particular approximation that utilizes the implicit function theorem is called implicit MAML (iMAML), that enables computing the asymptotic inner-loop gradient efficiently [?]. By doing this, the meta-learner can update the outer-loop based on directions obtained by solving the inner-loop over a large number of iterations without requiring high memory, as the method is agnostic to the path taken in the inner-loop and only uses the final gradient direction. As a result, better adaptation performance can be expected.

In this paper, we solve the optimal tracking problem in unknown nonlinear systems by leveraging the iMAML algorithm for system identification, and using this identified predictive model within a receding horizon framework. This is summarized herein:

(i) To efficiently utilize limited samples from the target system, we propose a meta-learning framework that customizes iMAML for adapting neural predictive models. The framework pre-trains an aggregated model from source systems, and then fine-tunes it with a small target dataset and few adaptation steps. (ii) We incorporate the adapted models into MPC, determining optimal inputs for the target system. (iii) Numerical examples show that our method outperforms models trained solely by MAML or with only target system data, along with other baselines.

Organization: Section II introduces the problem of interest. In Section III, we propose the meta-learning-based MPC to solve the problem by training the NSSM with data from source systems. The performance of our algorithm is evaluated through numerical examples in Section IV. Finally, Section V concludes the paper.

II. Problem Formulation

We consider a family of parameterized discrete-time nonlinear systems of the form

$$
\begin{aligned}
x_{t+1} &= f(x_t, u_t, \theta_f), \\
y_t &= g(x_t, u_t, \theta_g),
\end{aligned}
\tag{1}
$$

where $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ are respectively system state and output. Moreover, $u \in \mathcal{C} \subseteq \mathbb{R}^p$ is the input with $\mathcal{C}$ being a compact, convex set. Both f and g are unknown nonlinear functions, and $\theta := [\theta_f; \theta_g] \in \mathbb{R}^w$

denotes a vector of unknown parameters. Despite the unknown dynamics, our objective is to design a controller to asymptotically track some reference signal $\bar{y}$ in the sense that

$$
\lim_{t \to \infty} y_t = \bar{y}_t.
\tag{2}
$$

Since we aim to solve the problem by using a meta-learning framework, we assume that the target system under consideration is similar to some systems, from which we have archival data.

A. Neural State-Space Model (NSSM)

For the unknown system described by (1), we construct a neural state-space model (NSSM) that approximates the dynamics of the target system (the system of interest) parameterized by $\theta = \theta^*$. Based on the NSSM, the optimal controller can be designed using a model-based approach. The true value of θ^* is unknown, but it is drawn from a distribution

$$
\theta^* \sim \Theta.
\tag{3}
$$

The distribution Θ is often informed by domain experience, e.g., a particular parameter may have some interpretation from physics that defines a range of admissible values, and one may sample from a uniform distribution over this range. Therefore, it is not impractical to assume we know Θ, but it is not necessarily for our proposed method: we instead assume that Θ can be reliably sampled from even though its distributional form is unknown.

Let $U(\theta^*, T^*)$ and $Y(\theta^*, T^*)$ represent the input and output trajectories generated from the target system over a horizon T^* and construct a target dataset

$$
\mathcal{D}_{\text{target}} := \{U(\theta^*, T^*), Y(\theta^*, T^*)\}.
\tag{4}
$$

Using this dataset, one can train an NSSM of the form

$$
\begin{aligned}
z_t &= f_{\text{enc}}(U_{t-H+1:t}, Y_{t-H+1:t}), & \text{(5a)} \\
z_{t+1} &= A_z z_t + B_z u_{t+1}, & \text{(5b)} \\
\hat{y}_t &= C_z z_t. & \text{(5c)}
\end{aligned}
$$

where $z \in \mathbb{R}^{n_z}$ is the latent state learned by the encoder network $f_{\text{enc}}(\cdot)$ from historical input and output data:

$$
\begin{aligned}
U_{t-H+1:t} &:= \{u_{t-H+1}, u_{u-H+2}, \cdots, u_t\} \subseteq \mathcal{D}_{\text{target}}, \\
Y_{t-H+1:t} &:= \{y_{t-H+1}, y_{t-H+2}, \cdots, y_t\} \subseteq \mathcal{D}_{\text{target}}.
\end{aligned}
\tag{6}
$$

Here, the latent state dimension $n_z \in \mathbb{N}$ and window length $H \in \mathbb{N}$ are user-defined parameters; these are standard hyperparameters in classical system identification as well. Also, $\hat{y}_t \in \mathbb{R}^m$ is the predicted output, which we require to be close to the true output y_t after NSSM training [?].

Note that the NSSM (5) consists of two components: the deep latent encoder f_{enc} in (5a) captures the nonlinearity of the system, while the state-space component in (5b)–(5c) models temporal dependencies in the system. In practical implementations, f_{enc} is trained

from standard deep learning. On the other hand, the state transition in (5b) and (5c) can be modeled by a recurrent neural network (RNN). The RNN layer takes u and z as the current input and hidden state, respectively, and outputs the next prediction $\hat{y}$.

Training (5) involves optimizing the weights of the encoder network $f_{\text{enc}}(\cdot)$, and the elements of the linear decoders A_z, B_z, C_z and D_z; let ω be the set of these parameters. The prediction performance of (5) is evaluated as follows.

1) We first construct a dataset $\mathcal{D}$ from $\mathcal{D}_{\text{target}}$ by collecting trajectories of length $H + T$:

$$U_{t-H+1:t+T} := \{u_{t-H+1}, u_{u-H+2}, \cdots, u_{t+T}\},$$
$$Y_{t-H+1:t+T} := \{y_{t-H+1}, y_{t-H+2}, \cdots, y_{t+T}\}.$$

2) With $U_{t-H+1:t}$ and $Y_{t-H+1:t}$, we then obtain the latent state z_t using (5a).
3) For a prediction horizon of T and with input $U_{t+1:t+T}$, let us recursively compute

$$\hat{Y}_{t+1:t+T} := \{\hat{y}_{t+1}, \hat{y}_{t+2}, \cdots, \hat{y}_{t+T}\}$$

from (5b) and (5c).

4) The performance of the NSSM is evaluated through the following loss function

$$\ell_{\text{SSM}} = \frac{1}{T}||Y_{t+1:t+T} - \hat{Y}_{t+1:t+T}||^2.$$

Obviously, ℓ_{SSM} is determined by both the dataset and the parameters of (5). To make it clearer, we explicitly define $\ell_{\text{SSM}}(\mathcal{D}; \omega)$ as the loss function evaluated on the dataset $\mathcal{D}$ and the parameter ω.

B. NSSM-enabled MPC

Once trained, the NSSM can be used for predictive control tasks, such as within an MPC framework. To leverage this model to calculate the optimal input for the reference tracking, we rewrite (5b) and (5c) in a compact form

$$s_{t+1} = A s_t + B u_{t+1}, \tag{7}$$

where

$$s_{t+1} := \begin{bmatrix} z_{t+1} \\ \hat{y}_{t+1} \end{bmatrix}, \ A := \begin{bmatrix} A_z & 0 \\ C_z A_z & 0 \end{bmatrix}, \ B := \begin{bmatrix} B_z \\ C_z B_z \end{bmatrix}.$$

With this linearized model, we propose to track the reference based on model predictive control (MPC). Notice that the choice of the controller is not limiting for the approach. Here we focus on MPC as it is a suitable approach for trajectory tracking under constraints, as demonstrated in ([?]). It solves the following trajectory tracking problem:

$$\min_{u} \tilde{y}_{t+N|t}^T \cdot P \cdot \tilde{y}_{t+N|t}$$
$$+ \sum_{k=0}^{N-1} (\tilde{y}_{t+k|t}^T \cdot Q \cdot \tilde{y}_{t+k|t} + \Delta u_{t+k|t}^T \cdot R \cdot \Delta u_{t+k|t})$$
$$\text{s.t.} \ (5) \text{ and } u_{t+k|t} \in \mathcal{C}, \ k = 0, \ldots, N-1,$$
$$\tag{8}$$

where for $k = 0, \ldots, N-1$, we define

$$\tilde{y}_{t+k|t} \triangleq \hat{y}_{t+k|t} - \bar{y}_{t+k},$$
$$\Delta u_{t+k|t} \triangleq u_{t+k+1|t} - u_{t+k|t}.$$

Moreover, $Q \geq 0$ and $R > 0$ are design parameters kept fixed, and P is the solution to the discrete-time algebraic Ricatti equation

$$P = A^T \left(P - PB \left(R + B^T PB \right)^{-1} B^T P \right) A + Q.$$

Note that in (5), the nonlinearity is just used to initialize an otherwise linear prediction model. Hence the MPC optimization problem is convex and computationally tractable. Also notice that the formulation (8) is not unique. Other formulations (with different terminal costs, add output constraints, etc.) can also be used here.

The MPC finds the optimal controller that minimizes the cost function in (8), thereby directing the predicted output $\hat{y}$ towards the reference signal $\bar{y}$. When the NSSM accurately captures the nonlinear dynamics in (1), we achieve the reference tracking as claimed in (2).

C. Meta-learning framework

Collecting data is often expensive in many applications. Therefore, $\mathcal{D}_{\text{target}}$ is typically of limited size. Training with limited data can lead to poor prediction performance of NSSMs, which can adversely affect predictive control performance. To address this challenge, we propose a meta-learning framework. The framework is designed to acquire an NSSM, usually offline, that leverages data from multiple source systems that share similarities with the target system.

For instance, the source system could be a numerical model or a digital twin of the target system, and is easily accessible. It could also be a system with modified physical parameters (e.g., different load, friction parameters, etc.) This aggregated model enables quick online adaptation to the target system through few-shot learning, requiring only a small number of data points.

As another motivating example, suppose that we successfully control some manufacturing process governed by a nonlinear system, and have access to data from this system. However, the manufacturing requirements change, which corresponds to a change in the physical parameters in the system, and the controller performance needs to be adjusted. By using meta-learning, we propose a method that achieves this adjustment using very few actual measurements from the new system (the target system), by shifting the data requirements to multiple previously observed systems (the source systems).

Assume that we have access to a dataset containing input and output trajectories generated by N_s different source systems in the form of (1), where the k-th system is parameterized by some vector θ_k sampled from the distribution Θ. It is important to know that θ_k need not to be known. We represent the source dataset as

$$\mathcal{D}_{\text{source}} = \{(U(\theta^k, T^k), Y(\theta^k, T^k))\}_{k=1}^{N_s} := \{\mathcal{D}^k\}_{k=1}^{N_s}, \tag{9}$$
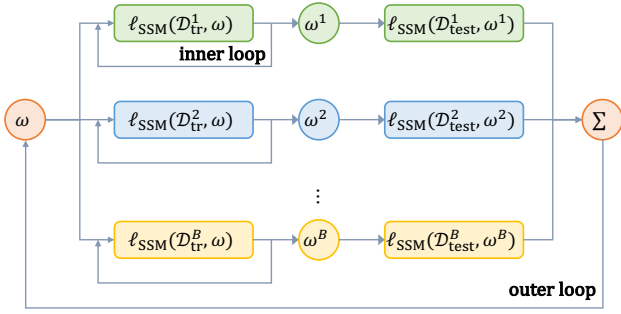
Fig. 1: Information flow in the meta-training phase.

where T^k is the length of trajectory produced by the k-th system, and $\theta_k \sim \Theta$ for each $k \in \{1, \cdots, N_s\}$. Our objective is to learn an aggregated NSSM by leveraging the source data $\mathcal{D}_{\text{source}}$. This aggregated model should have the ability to be quickly adapted to other systems only using small datasets and without explicitly estimating θ. Later, we discuss how to update the dataset by incorporating the trajectories produced by MPC on the source systems.

III. Meta-Learning-Based MPC Design

We tailor the meta-learning algorithm iMAML [?] to address the problem of reference tracking on the target system. The framework consists of two phases: the meta-training phase learns the aggregated NSSM using $\mathcal{D}_{\text{source}}$, while the meta-inference phase adapts this model to the target system using $\mathcal{D}_{\text{target}}$.

A. Bi-level optimization problem in meta-learning

The phase of meta-training involves two loops: the outer-loop updates the parameters of the NSSM based on the performance across multiple source systems and the inner-loop adapts the model to individual source systems. This enables the meta-learning algorithm to learn from multiple systems and generalize well to new datasets.

Specifically, as shown in Fig. 1, the outer loop provides an aggregated representation ω of the NSSM (5). In each iteration of the outer loop, we sample a batch of trajectories $\{\mathcal{D}^b\}_{k=1}^B$ from the source dataset $\mathcal{D}$source. Then the inner-loop trains individual weights ω^b using the aggregated representation and the source data. After this, the outer loop updates the aggregated representation ω using these individual values. The process is repeated until convergence.

For each $\mathcal{D}^b$, we partition it into a training set $\mathcal{D}_{\text{tr}}^b$ and a testing set $\mathcal{D}_{\text{test}}^b$. The training set is utilized to adapt the aggregated representation ω and produce the task-specific parameter ω^b, tailored to the system (1) parameterized by θ^b. To be specific, within the context of iMAML, the inner-loop solves the following optimization problem:

$$\omega^b(\omega) := \operatorname*{argmin}_{\psi}\ \ell_{\text{SSM}}(\mathcal{D}_{\text{tr}}^b; \psi) + \frac{\gamma}{2}\|\psi - \omega\|^2. \quad (10)$$

Here, ω is the weight of the aggregated NSSM, and the regularization term $\|\psi - \omega\|^2$ encourages the finding of an optimal solution within a (ideally, small) neighborhood of ω, and $\gamma > 0$ controls the regularization strength. As we will shown later in Sections III-B and III-D, this regularization enables us to express the outer-loop gradient term in closed-form by exploiting the implicit function theorem. By doing so, the gradient can be calculated exactly, without relying on the inner-loop optimization path but solely on the optimal solution. As a result, there is no need to store the full optimization path or approximate the gradient, as required in MAML [?]. This approach reduces memory consumption and enhances predictive performance, compared to MAML whose memory complexity is high for a large number of inner-loop steps.

The testing set $\mathcal{D}_{\text{test}}^b$ is employed in the outer loop to update the aggregated representation ω. This is achieved by evaluating the prediction performance of the NSSM across multiple systems, as defined below

$$\min_{\omega}\ \frac{1}{B} \sum_{b=1}^{B} \ell_{\text{SSM}}(\mathcal{D}_{\text{test}}^b; \omega^b(\omega)). \quad (11)$$

As shown in (10), $\omega^b(\omega)$ is the optimal task-specific parameter adapted from ω. Therefore, the objective of the outer loop is to learn a set of parameters ω that can produce good task-specific parameters after adaptation. For simplicity, we will denote $\omega^b(\omega)$ as ω^b in the rest of this paper.

B. Approximation solutions to the bi-level optimization problem

To efficiently solve the bi-level optimization problem (10) and (11), the solution of (11) can be approached using the gradient decent algorithm:

$$\omega \leftarrow \omega - \beta_{\text{out}} \frac{1}{B} \sum_{b=1}^{B} \nabla_\omega \ell_{\text{SSM}}(\mathcal{D}_{\text{test}}^b; \omega^b), \quad (12)$$

where β_{out} denotes the learning rate for the outer loop. Considering (10), ω^b also depends on ω. Therefore, by applying the chain rule, we expand the gradient term in (12) as

$$\nabla_\omega \ell_{\text{SSM}}(\mathcal{D}_{\text{test}}^b; \omega^b) = \frac{d\omega^b}{d\omega} \cdot P^b. \quad (13)$$

Here,

$$P^b := \nabla_\psi \ell_{\text{SSM}}(\mathcal{D}_{\text{test}}^b; \psi)|_{\psi=\omega^b},$$

which is the value of $\nabla_\psi \ell_{\text{SSM}}(\mathcal{D}_{\text{test}}^b; \psi)$ evaluated at $\psi = \omega^b$. Since P^b can be easily obtained via back propagation on the NSSM, we focus on calculating $\frac{d\omega^b}{d\omega}$.

Recall that ω^b is obtained by solving the inner optimization problem (10), typically through iterative algorithms like gradient descent. Thus, one approach to compute $\frac{d\omega^b}{d\omega}$ is propagating derivatives throughout the iterative process in the inner-loop. As this requires the full path of optimization to be stored in the memory, the

approach becomes intractable if the number of iterative steps is large, or if the inner-loop procedure is non-differentiable. To address this challenge, the following lemma introduces an alternate method to calculate this gradient, which does not rely on the optimization path, but only on the optimal solution ω^b.

Lemma 1 ([?]): Define

$$Q^b := I + \frac{1}{\gamma} \nabla^2_\psi \ell_{\text{SSM}}(\mathcal{D}^b_{\text{tr}}; \psi)|_{\psi=\omega^b}, \quad (14)$$

where $\nabla^2_\psi \ell_{\text{SSM}}(\mathcal{D}^b_{\text{tr}}; \psi)|_{\psi=\omega^b}$ is value of the Hessian matrix $\nabla^2_\psi \ell_{\text{SSM}}(\mathcal{D}^b_{\text{tr}}; \psi)$ evaluated at $\psi = \omega^b$. If Q^b is invertible, then

$$\frac{d\omega^b}{d\omega} = (Q^b)^{-1}. \quad (15)$$

Combining (13) and (15), we obtain

$$\nabla_\omega \ell_{\text{SSM}}(\mathcal{D}^b_{\text{test}}; \omega^b) = (Q^b)^{-1} P^b. \quad (16)$$

As Q^b solely depends on ω^b, Lemma 1 offers a solution for computing $\frac{d\omega^b}{d\omega}$ with reduced memory requirements. Nevertheless, obtaining ω^b, the exact solution to the inner problem, involves solving (10) until convergence, a task often impractical in practice. Therefore, in this paper, we resort to employing the gradient descent algorithm with a finite number of steps to obtain an approximate solution.

On the other hand, calculating the inverse of Q^b can also be expensive, especially for large NSSMs. In order to tackle this issue, we notice that $(Q^b)^{-1} P^b$ is the solution to the following problem:

$$\min_\phi \ \phi^T Q^b \phi - \phi^T P^b. \quad (17)$$

Therefore, in practice, we can also approximate $(Q^b)^{-1} P^b$ by solving (17) with a finite number of iterations, e.g., update the solution by using a few gradient steps. By doing so, one avoids the need to explicitly compute the matrix inverse.

Finally, in order to ensure the compatibility with MPC, we update the source dataset online. After adaptation in each inner loop, we determine the optimal input u^b_* by solving the MPC problem (8) using the NSSM parameterized by ω^b. To balance exploration and exploitation, we let

$$u^b = u^b_* + u^b_0, \quad (18)$$

where u^b_0 is a random variable drawn from a Gaussian distribution, i.e., $u^b_0 \sim \mathcal{N}(0, \Sigma^b)$. Injecting u^b into the system (1) (parameterized by θ^b) yields an output y^b. We then update $\mathcal{D}_{\text{source}}$ by incorporating the most recent trajectory (u^b, y^b). The complete meta-training algorithm is summarized in Algorithm 1.

C. Meta-inference

After the meta-learning phase, we acquire the aggregated NSSM parameterized by ω_∞. The subsequent meta-inference phase adapts this aggregated model to accurately capture the characteristics of the target system

---

Algorithm 1 Meta-training phase

Input: $\omega \leftarrow$ randomly initialize the parameters of the NSSM
Input: $\mathcal{D}_{\text{source}} \leftarrow$ initial source dataset
Input: Regularization strength γ and learning rate β_{out}
  while not converge **do**   ▷ outer-loop
    Sample batch $\{\mathcal{D}^b\}^B_{b=1}$ from $\mathcal{D}_{\text{source}}$
    for $b = 1 : B$ **do**   ▷ inner-loop
      Partition $\mathcal{D}^b$ into $\mathcal{D}^b_{\text{tr}}$ and $\mathcal{D}^b_{\text{test}}$
      $\omega^b \leftarrow$ solving (10) by gradient descent
      $g^b \leftarrow$ solving (17) by gradient descent
      $u^b \leftarrow$ the new input computed by (8) and (18) with the NSSM parameterized by ω^b
      $y^b \leftarrow$ simulating the system (1) using u^b with $\theta = \theta^b$
      Update $\mathcal{D}_{\text{source}}$ by adding (u^b, y^b)
    end for
    $\omega \leftarrow \omega - \beta_{\text{out}} \frac{1}{B} \sum^B_{b=1} g^b$
  end while
Output: $\omega_\infty \leftarrow$ parameters obtained after meta-training phase

---

and achieve the optimal tracking.

Algorithm 2 Meta-inference phase with tracking MPC

Input: $\omega_\infty \leftarrow$ parameters obtained after meta-training phase
Input: $\mathcal{D}_{\text{target}} \leftarrow$ initial target dataset
Input: γ
  $\omega^* \leftarrow$ solving (19) by gradient descent   ▷ adaptation
  while not done **do**   ▷ reference tracking
    $u^* \leftarrow$ the optimal solution of (8) based on the NSSM parameterized by ω^*
    Inject u^* to the target system
  end while

Algorithm 2 outlines the meta-inference process; note that no outer-loop is involved in this phase. We only perform an adaptation step similar to (10) to derive the specific NSSM for the target system by using the limited dataset $\mathcal{D}_{\text{target}}$. This is expressed as

$$\omega^* := \underset{\psi}{\text{argmin}} \ \ell_{\text{SSM}}(\mathcal{D}_{\text{target}}; \psi) + \frac{\gamma}{2} ||\psi - \omega_\infty||^2. \quad (19)$$

In practice, we approximate the optimal solution using only a few gradient steps. This corresponds to very few "trials" to correctly adapt the parameters.

Now, with this NSSM tailored for the target system, i.e., the NSSM parameterized with ω^*, we can achieve reference tracking by finding the optimal input through MPC (8).

D. Comparison with MAML

Algorithm 1 is developed based on the iMAML algorithm [?]. Alternatively, one could also consider here the MAML algorithm [?]. Unlike (10), the inner loop of the MAML-based NSSM approximates the solution of

$$\omega^b := \underset{\psi}{\text{argmin}} \ \ell_{\text{SSM}}(\mathcal{D}^b_{\text{tr}}; \psi). \quad (20)$$

Namely, setting γ in (10) as 0. In practice, the algorithm performs the adaptation through the gradient descent:

$$\omega_0^b = \omega,$$
$$\omega_m^b = \omega_{m-1}^b - \beta_{\text{in}} \nabla_{\omega_{m-1}^b} \ell_{\text{SSM}}(\mathcal{D}_{\text{tr}}^b; \omega_{m-1}^b), \ i = 1, \cdots, M,$$
$$\omega^b = \omega_M^b,$$
(21)

where M is the number of inner-loop iterations and β_{in} is the inner-loop learning rate. The outer-loop of MAML aims to solve the same problem as (11). However, since a different cost function is optimized in (20), $\frac{d\omega^b}{d\omega}$ cannot be directly calculated as in (15). The exact calculation involves propagating derivatives along the optimization path, which is not memory-efficient in practice. To tackle this, existing approaches typically resort to approximating (12) in the outer loop update as:

$$\omega \leftarrow \omega - \beta_{\text{out}} \frac{1}{B} \sum_{b=1}^{B} \nabla_{\omega^b} \ell_{\text{SSM}}(\mathcal{D}_{\text{test}}^b; \omega^b), \qquad (22)$$

where it is assumed that

$$\nabla_{\omega} \ell_{\text{SSM}}(\mathcal{D}_{\text{test}}^b; \omega^b) \approx \nabla_{\omega^b} \ell_{\text{SSM}}(\mathcal{D}_{\text{test}}^b; \omega^b). \qquad (23)$$

This approximation results in a tractable solution since $\nabla_{\omega^b} \ell_{\text{SSM}}(\mathcal{D}_{\text{test}}^b; \omega^b)$ can be readily obtained. However, (23) may harm prediction performance of the NSSMs. In contrast, by leveraging the iMAML, our proposed algorithm can calculate the gradient without relying on the optimization path, see (16). Therefore, there is no need to store the full optimization path or approximate the gradient as required in MAML. Consequently, the proposed approach reduces memory consumption and enhances control performance. In Section IV, we will compare the performance of our algorithm and the MAML-based one through numerical examples.

IV. Simulation

In this section, we illustrate the performance of our meta-learned MPC through some numerical examples.

A. Van der Pol oscillators

We first consider a family of van der Pol oscillators. The dynamics of each oscillator is given by

$$\dot{x}_1 = x_2, \ \dot{x}_2 = \theta x_2 \left(1 - x_1^2\right) - x_1 + u,$$
$$y = x, \qquad (24)$$

where θ is the unknown damping ratio. By using meta-learning, we want the approach to work for various different damping ratios and to adapt given the data of several of them. The source and target systems are generated by sampling θ from a Gaussian distribution $\theta \sim \mathcal{N}(0, 1)$.

We select the dimension of the latent state z as $n_z = 5$. The encoder f_{enc} is formed by a neural network comprising one input layer, one hidden layer, and one
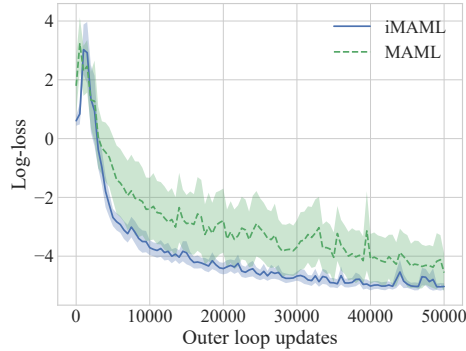

Fig. 2: Comparison of the prediction performance in the meta-training phase. The lines and shaded regions respectively represent the mean and standard deviation across 10 runs.

output layer, with each layer containing 128 neurons and activated by rectified linear units (ReLUs). Notice that we can afford to use a shallow network here due to the explicit incorporation of the adaptation mechanism into the meta-training process. However, if we were required to learn a aggregated model for the entire distribution Θ, we would need more depth and a larger-dimensional latent vector. The matrices in the SSM A_z, B_z and C_z, are randomly initialized. The experiments are repeated for 10 different random seeds.

First, we evaluate the prediction performance of the NSSM during the meta-training phase. We compare the iMAML-based Algorithm 1, with the MAML-based approach as outlined in Section III-D.

The comparison is depicted in Fig. 2, where the x-axis refers to the number of outer loop updates, that is, (12) and (22) respectively in the two algorithms. In each outer loop, we sample a batch of $B = 16$ datasets from $\mathcal{D}_{\text{source}}$. Moreover, the performance is evaluated by taking the logarithm of $1/B \sum_{b=1}^{B} \ell_{\text{SSM}}(\mathcal{D}_{\text{test}}^b; \omega^b)$, showing the prediction performance of the NSSMs. It is observed that Algorithm 1 outperforms the MAML-based approach throughout the training stage since it computes the derivative $\frac{d\omega^b}{d\omega}$ more accurately and thus requires less iterations for the same loss, as discussed in Section III-D. Therefore, it results in a smaller approximation error of the NSSM.

To evaluate the tracking performance of the proposed algorithm on the target system, we compare against a few approaches, all of which have the same NSSM architecture but are initialized differently:

1) iMAML: The parameters of the NSSM are set with those obtained from the meta-training phase of the iMAML-based algorithm, that is, ω_∞ output by Algorithm 1.
2) MAML: The parameters of the NSSM are set with those obtained from the meta-training phase of the MAML-based algorithm as outlined in Section III-

D. While Fig. 2 shows that iMAML yields less prediction errors over MAML during the meta-training phase, we would like to investigate whether this leads to a better control performance on the target system.

3) Supervised learning: Since there is no meta-training phase, the parameters are chosen randomly. This comparison with Supervised learning highlights whether meta-learning can improve our performance by leveraging the knowledge from source systems.

Starting from these different initial parameters, we train the NSSMs using data from the target system $\mathcal{D}_{\text{target}}$, which only contains 300 data points. Specifically, for iMAML, this refers to run the adaptation step in Algorithm 2. On the other hand, iMAML and supervised learning perform the following update

$$\omega_m^* = \omega_{m-1}^* - \beta_{\text{in}} \nabla_{\omega_{m-1}^*} \ell_{\text{SSM}} \left(\mathcal{D}_{\text{target}}; \omega_{m-1}^* \right), \quad (25)$$

with ω_0^* being the parameters initialized for the NSSMs.

It is evident from Fig. 3 that meta-learning expedites the training of NSSMs by using data from similar systems. Moreover, Fig. 5 shows the tracking performance of different algorithms. Here, we aim to track a circle (centered at the origin with radius 2) by running MPC on the NSSMs trained for 0, 100, and 3000 steps, respectively. From the first column of the figure, we conclude that after the meta-training phase, both iMAML and MAML produce good aggregated models that can capture dynamics of the target system to a certain extent. The performance is further improved after the adaptation steps using target data $\mathcal{D}_{\text{target}}$. However, since supervised learning does not benefit from the similar systems, it performs poorly compared with using meta-learning.

Another observation from comparing iMAML and MAML in Fig. 3 and Fig. 5 is that iMAML tends to exhibit faster convergence and better tracking performance, especially in the initial stages of meta-inference. The enhancement is particularly striking when comparing iMAML and MAML tracking after 10 and 100 steps. This is because that iMAML exhibits superior performance with lower loss during the meta-training phase, as evidenced by Fig. 2. Consequently, it starts with better initial parameters, facilitating more effective adaptation to the target system.

B. Pendulum

We also test the proposed algorithm on the benchmark Pendulum Gym Environment [?]. The environment consists of a pendulum that is free to swing in a vertical plane under the influence of gravity and is controlled by applying a torque to the end point of the pendulum, see Fig. 4. The goal is to keep the pendulum upright, with the least amount of applied torque. The state $x = [\theta; \dot{\theta}]$ consists of the angle and angular velocity of the pendulum, while the input is the torque τ and the
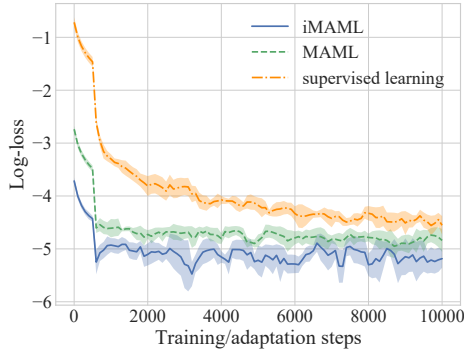

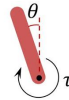Fig. 3: Comparison of the prediction performance on the target system.

Fig. 4: The Pendulum Gym Environment.

output is θ. We collect the input and output data that are contaminated by random noises.

We construct the source systems by sampling pendulum masses of varying weights and then evaluate on five target tasks involving pendulums of different mass. Due to the space limits, we omit parameters in Algorithm 1 and 2. Fig. 6 shows the prediction performance of NSSM along the meta-training phase. When this phase converges, we apply the aggregated model on target systems. For all target systems, the control error, defined as the deviation of the angle from the center, i.e., $|\theta|$, decreases to below 10^{-4} after 100 adaptation steps. That means, our algorithms are robust to noise and successfully stablize the pendulum by using few samples on target systems. Therefore, meta learning is useful to pre-train a reasonable NSSM that can be adapted to system changes and quickly finds proper controllers.

V. Conclusion

This paper considers the problem of reference tracking in an unknown nonlinear system by using an NSSM. It introduces an iMAML-based MPC algorithm, which comprises two phases. In the meta-training phase, data from source systems is leveraged to pre-train the NSSM, while in the meta-inference phase, this model is quickly adapted to the target system using only a small amount of data. This approach addresses the issue of limited dataset availability for the target system, enabling the development of a precise model that facilitates optimal tracking even with a small dataset. Numerical examples demonstrate the superior performance of the proposed algorithm compared to several baselines. As a future work, we are planning experiments on physical benchmark problems with a view towards transitioning to manufacturing processes.
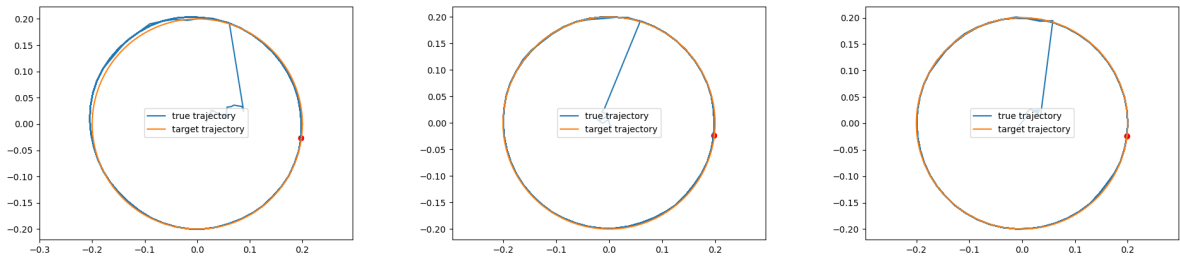
(a) The iMAML-based MPC (Algorithm 2).
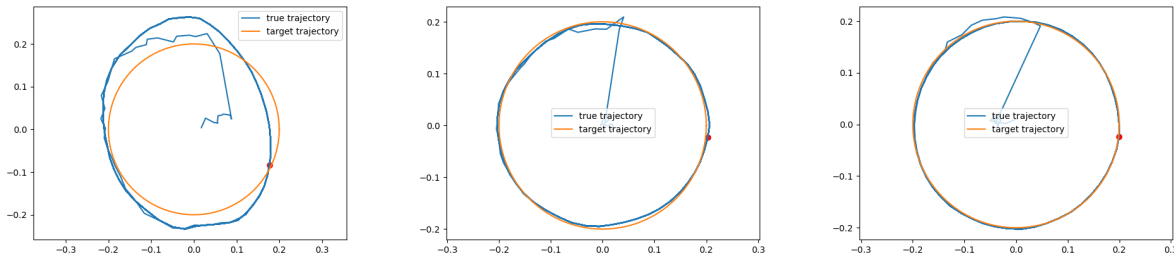
(b) The MAML-based MPC.
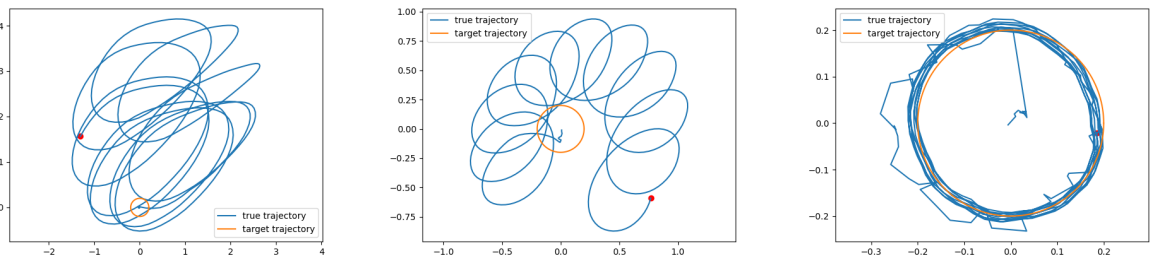
(c) The supervised learning-based MPC.

Fig. 5: Comparison of the tracking performance on the target system, where the three columns show the performance of different algorithms by using the NSSMs adapted after 10, 100, and 3000 steps, respectively. The lines and shaded regions respectively represent the mean and standard deviation across 10 runs.
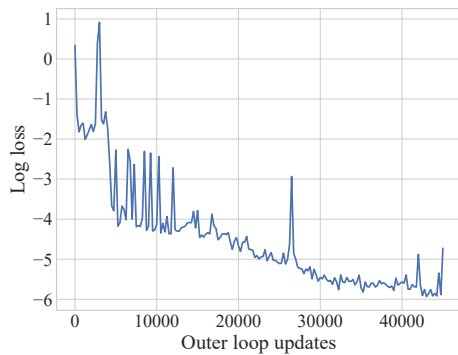

Fig. 6: The prediction performance of NSSM on the pendulum.