2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Model Extraction Attacks Against Reinforcement Learning Based Controllers

Momina Sajid, Yanning Shen, and Yasser Shoukry

Abstract— **We introduce the problem of model-extraction attacks in cyber-physical systems in which an attacker attempts to estimate (or extract) the feedback controller of the system. Extracting (or estimating) the controller provides an unmatched edge to attackers since it allows them to predict the future control actions of the system and plan their attack accordingly. Hence, it is important to understand the ability of the attackers to perform such an attack. In this paper, we focus on the setting when a Deep Neural Network (DNN) controller is trained using Reinforcement Learning (RL) algorithms and is used to control a stochastic system. We play the role of the attacker that aims to estimate such an unknown DNN controller, and we propose a two-phase algorithm. In the first phase, also called the offline phase, the attacker uses side-channel information about the RL-reward function and the system dynamics to identify a set of candidate estimates of the unknown DNN. In the second phase, also called the online phase, the attacker observes the behavior of the unknown DNN and uses these observations to shortlist the set of final policy estimates. We provide theoretical analysis of the error between the unknown DNN and the estimated one. We also provide numerical results showing the effectiveness of the proposed algorithm.**

I. Introduction

Cyber-Physical Systems (CPS) control the vast majority of today's critical infrastructure, and these CPSs are vulnerable to attacks that range from denial-of-service [1], replay attacks [2], man-in-the-middle attacks [3], and false data injection attacks [4]. Given the safety-criticality nature of CPSs, it is unsurprising to see the current interest in developing control-theoretic techniques to protect feedback controllers in these systems against such attacks [5]–[7].

Motivated by these vulnerabilities, in this paper, we introduce the model extraction attacks on CPSs. In machine learning literature, model extraction attacks refer to algorithms that aim to extract (or estimate) the parameters of complex Deep Neural Network (DNN) models by observing input-output data collected from these DNNs. These attacks have been widely studied in the supervised learning setup [8], [9]. In the CPS context, DNNs are widely used as feedback controllers thanks to the recent success of using Deep Reinforcement Learning (DRL) to train such complex feedback controllers. In the DRL setting, gradient-descent-based algorithms aim to train a DNN that maximizes a given reward function. In this spirit, we study model extraction attacks on feedback control systems, which can play a significant role in understanding the vulnerabilities of CPSs. First, while the control algorithms are sophisticated and proprietary in some cases, the more common case is that the algorithm parameters constitute intellectual property as they are based on careful modeling, analysis, training, and validation processes at the often considerable expense of time and resources. Hence, extracting such controllers provides a significant loss in learning intellectual properties. Second, extracted feedback controllers can be reverse-engineered to identify weaknesses in CPSs, predict their future behavior, and launch complex attacks.

In this paper, we play the role of an attacker whose goal is to estimate (or extract) an unknown DNN controller using input-output observations collected from the DNN along with side-channel information about the reward function used to train the DNN. Several technical challenges arise in this setup. First, the attacker cannot control the quality and quantity of the input-output data collected from the unknown DNN. In scenarios where the amount of input-output observation is relatively small, or these observations do not cover the entire space of input-outputs, it is hard to construct function estimators that provide small bounded error guarantees away from the provided input-output samples. Second, the reward function provides limited knowledge about the unknown DNN. This stems from the fact that DRL algorithms aim to solve non-convex problems (due to the nonlinearity of the DNNs). Hence, the exact parameters of the DNN depend on a set of unknown parameters like the exact optimization algorithm used, the initial DNN parameters, the learning rates, the stopping criteria, and other hyperparameters.

To that end, we propose a two-phase algorithm. In the first phase, the attacker analyzes the reward function to obtain a set of candidate function estimators. These candidate estimators correspond to the reward function's possible local and global maxima. We propose novel techniques based on multi-kernel learners to approximate the landscape of local and global maxima and use non-convex constraint solvers to estimate these critical points with bounded error. Note that such analysis of the landscape of critical points can be carried over without access to input-output observations from the unknown DNN. Hence, we refer to this phase as the offline phase. In the second phase, also called the online phase, the attacker uses the input-output observations to shortlist the number of candidate function estimators. We provide theoretical guarantees on the estimation error between the unknown DNN and the estimated one for both around the observed input-output data and away from this data. We also show numerical examples to illustrate the efficiency of the proposed algorithm.

II. Problem Statement

A. Notations

Let $\mathbb{R}$ and $\mathbb{N}$ denote the set of real numbers and natural numbers, respectively. We use $\bigwedge$ and $\bigvee$ to represent the logical AND and logical OR operators, respectively. For a vector $x \in \mathbb{R}^n$, we use $\|x\|_2$ to denote the Euclidean norm of x and x_i to denote the ith element of x. Given the two vectors $x \in \mathbb{R}^{n_x}$ and $y \in \mathbb{R}^{n_y}$, we use (x, y) to denote the vector $[x^T y^T]^T \in \mathbb{R}^{n_x+n_y}$. Finally, we denote by $|\Gamma|$ the cardinality

The authors are with the Department of Electrical Engineering and Computer Science, University of California, Irvine, CA 92697, USA (e-mail: {msajid,yannings,yshoukry}@uci.edu).

This work was partially sponsored by the NSF award #CNS-2002405, #ECCS-2207457 and C3.AI Digital Transformation Institute.

2023 62nd IEEE Conference on Decision and Control (CDC) | 979-8-3503-0124-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/CDC49753.2023.10383661

of a set Γ and by $\mathbb{B}_r(c)$ the two-norm ball centered at c with radius r, i.e., $\mathbb{B}_r(c) = \{x \in \mathbb{R}^n | \ \|x - c\|_2 \leq r\}$.

B. System Dynamics

We consider non-linear, stochastic, discrete-time dynamic systems of the form:

$$\mathbb{P}(s^{(k+1)} \mid s^{(k)}, a^{(k)}) = f_d(s^{(k)}, a^{(k)}), \tag{1}$$

where $s^{(k)} \in S \subset \mathbb{R}^q$ is the state of the system at time $k \in \mathbb{N}$, S is the state space which is assumed to be compact, and $a^{(k)} \in A \subset \mathbb{R}^m$ is the action at time $k \in \mathbb{N}$. We assume the dynamical system is controlled by a deep neural network-based policy $\pi_{\theta^*}\colon S \longrightarrow A$ which is a memory-less, state-feedback, non-linear controller. An F-layer NN is specified by composing F layer functions (or just layers). A layer ω with $\mathfrak{i}_\omega$ inputs and $\mathfrak{o}_\omega$ outputs is specified by a weight matrix $W^\omega \in \mathbb{R}^{\mathfrak{o}_\omega \times \mathfrak{i}_\omega}$ and a bias vector $b^\omega \in \mathbb{R}^{\mathfrak{o}_\omega}$ as follows:

$$L_{\theta^\omega} : z \mapsto \phi(W^\omega z + b^\omega), \tag{2}$$

where ϕ is a nonlinear function, z is the input to the NN, and $\theta^\omega \triangleq (W^\omega, b^\omega)$ for brevity. Thus, an F-layer ReLU NN is specified by F layer functions $\{L_{\theta^\omega} : \omega = 1, \ldots, F\}$ whose input and output dimensions are composable: that is, they satisfy $\mathfrak{i}_\omega = \mathfrak{o}_{\omega-1}$, $\omega = 2, \ldots, F$. Specifically:

$$\pi_\theta(s) = (L_{\theta^F} \circ L_{\theta^{F-1}} \circ \cdots \circ L_{\theta^1})(s), \tag{3}$$

where we index a ReLU NN function with a list of parameters $\theta \triangleq (\theta^1, \ldots, \theta^F)$.

We assume a reinforcement learning algorithm was used to train the policy and obtain the optimal parameters θ^* that maximizes the discounted, average reward, i.e.:

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmax}} \ J(\theta) = \underset{\theta \in \Theta}{\operatorname{argmax}} \ \underbrace{\mathbb{E} \sum_{k=0}^{H} \gamma^k r\left(s^{(k)}, \pi_\theta(s^{(k)})\right)}_{J(\theta)}, \tag{4}$$

where $\Theta \subset \mathbb{R}^{\mathfrak{o}_1 \times (\mathfrak{i}_1 + 1)} \times \ldots \times \mathbb{R}^{\mathfrak{o}_F \times (\mathfrak{i}_F + 1)} \subseteq \mathbb{R}^l$ is a compact set with $l = (\mathfrak{o}_1 \times (\mathfrak{i}_1 + 1)) + \ldots + (\mathfrak{o}_F \times (\mathfrak{i}_F + 1))$ is the total number of parameters in the neural network, the expectation $\mathbb{E}$ is taken over the trajectories of the dynamical system (1), $r : S \times A \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discounting factor.

C. Threat Model and Problem Definition

We assume the role of an attacker who is interested in constructing an estimate $\hat{\theta} \in \Theta$ of the NN policy parameters θ^* such that:

$$\max_{s \in S} \|\pi_{\hat{\theta}}(s) - \pi_{\theta^*}(s)\|_2 \leq \psi, \tag{5}$$

for any attacker-selected accuracy $\psi \in \mathbb{R}$ with $\psi > 0$.

We assume the attacker can observe the input-output behavior of the unknown policy π_{θ^*}, i.e., $\mathcal{D} = \left\{(s^{(k)}, \pi_{\theta^*}(s^{(k)})\right\}$ for time step $k = 0, \ldots, N$. We emphasize that we place no requirements on the quantity and coverage of the observations $\mathcal{D}$. That is important from the attacker's point-of-view since the attacker can not observe the system's behavior in all states. Nevertheless, the attacker's objective in (5) insists on achieving a uniform, attacker-selected error bound across the entire state space. It is direct to show that the requirement in (5) can not be achieved using the observations $\mathcal{D}$, which necessitates the need for additional information. In particular, we assume the attacker has access to a noisy version of the objective function J called $\overline{J}$ where:

$$\max_\theta \|\overline{J}(\theta) - J(\theta)\| = \max_\theta \|m(\theta)\| \leq \overline{m}, \tag{6}$$

where $\overline{m}$ is assumed to be a known upper bound. Such an unknown error function $m(\theta)$ captures the attacker's imperfect information about the system dynamics f_d, the reward function r, the initial conditions and the approximation of the expectation $\mathbb{E}$ that were used during the training of the policy π_{θ^*}. Hence, whenever the attacker tries to evaluate $J(\theta)$, the attacker will observe an incorrect value of $\overline{J}$ instead, as captured by the unknown error function $m(\theta)$.

III. Offline Phase: Approximating the Landscape of Critical Points

To find an estimate $\pi_{\hat{\theta}}$ that satisfies (5), we propose a two-phase algorithm. In the first phase, the attacker uses the imperfect information about $J(\theta)$ to compute several candidate estimates, namely $\pi_{\hat{\theta}_1}, \pi_{\hat{\theta}_2}, \ldots$ without the use of any data collected from the unknown policy. To that end, we first recall that the unknown policy π_{θ^*} is the solution to the non-convex optimization problem defined in (4). Gradient-descent-based reinforcement learning algorithms terminate when they reach one global or local maxima. Since the non-convex problem in (4) may have several local/global maxima, our objective is to enumerate all these critical points and thus obtain one candidate estimate $\pi_{\hat{\theta}}$ for each of the identified local/global maxima.

Formally, we define g as the derivative of the objective function J with respect to the neural network parameters θ, i.e.:

$$g(\theta) = \frac{\mathrm{d}}{\mathrm{d}\theta} J(\theta). \tag{7}$$

Indeed, critical points where $g(\theta) = 0$ are the candidate values for the unknown θ^*. Nevertheless, the attacker has only access to an imperfect version of J and hence we define:

$$\overline{g}(\theta) = \frac{\mathrm{d}}{\mathrm{d}\theta} \overline{J}(\theta) = \frac{\mathrm{d}}{\mathrm{d}\theta}(J(\theta) + m(\theta)). \tag{8}$$

Our objective is to construct an approximate for g using the information of $\overline{g}$ and isolate candidates for the critical points of g. In particular, we will approximate g using multi-kernel regression algorithms. Multi-kernel-based estimators provide closed-form expressions that can be exploited for our analysis. To that end, we proceed as below.

A. Step 1: Use multi-kernel regression to estimate $g(\theta)$:

Using a grid of size η, we uniformly sample from the space of parameters Θ and we denote by $\tilde{\theta}^{(1)}, \ldots, \tilde{\theta}^{(M)}$, the sampled neural network parameters. For each sample, the attacker aims to evaluate $J(\tilde{\theta}^{(i)})$ using the imperfect knowledge of the dynamics and the reward function and obtains $\overline{J}(\tilde{\theta}^{(i)})$ which can be used to approximate the derivative g by using the information of $\overline{g}$. Specifically $\overline{g} = (\overline{g}_1, \ldots, \overline{g}_l)$

will be approximated by $\widetilde{g}_j$ as:

$$\widetilde{g}_j(\tilde{\theta}) = \frac{\overline{J}(\tilde{\theta}+c_j)-\overline{J}(\tilde{\theta})}{c} = \frac{J(\tilde{\theta}+c_j)+m(\tilde{\theta}+c_j)-J(\tilde{\theta})-m(\tilde{\theta})}{c}, \quad (9)$$

where $j \in \{1,...,l\}$ and $c_j \in \mathbb{R}^l$ is defined as a vector of all zeros except for the j-th element, which is equal to c. The collected samples $\{(\tilde{\theta}^{(i)}, \widetilde{g}_j(\tilde{\theta}^{(i)}))\}_{i=1}^M$ are used to learn a set of Multi-Kernel Learner (MKL)-based function estimators $\widehat{g}_j(\theta)$ of the form:

$$\widehat{g}_j(\theta) = \alpha_j^T k(\theta) \qquad j \in \{1,...,l\}, \quad (10)$$

where $k(\theta) = \left(\kappa(\theta,\tilde{\theta}^{(1)}),\ldots,\kappa(\theta,\tilde{\theta}^{(M)})\right)$ is the kernel vector[1] with $\kappa(\theta,\tilde{\theta}) : \mathbb{R}^l \times \mathbb{R}^l \to \mathbb{R}$ is a symmetric positive semidefinite basis (so-termed kernel) function, which measures the similarity between θ and $\tilde{\theta}$. With some abuse of notation, the vector $\alpha_j \in \mathbb{R}^M$ is the solution to the optimization problem:

$$\alpha_j = \underset{\alpha_j \in \mathbb{R}^M}{\operatorname{argmin}} \sum_{i=1}^{M} \|\widehat{g}_j(\tilde{\theta}^{(i)}) - \widetilde{g}_j(\tilde{\theta}^{(i)})\|_2 \quad j \in \{1,...,l\}. \quad (11)$$

Solving (11) has been widely studied in the literature; see for example [10]. We can bound the error between g and its estimate $\widehat{g}$ as follows.

Proposition III.1. *Under the following assumptions:*

1) $\exists R \in \mathbb{R}$ *such that* $\|r(s,a)\|_2 \le R \quad \forall (s,a) \in S \times A$,
2) $\exists G \in \mathbb{R}$ *such that* $\|\frac{\partial}{\partial\theta}\pi_\theta(s)\|_2 \le G \quad \forall s \in S$,
3) $\exists L \in \mathbb{R}$ *such that* $\|\frac{\partial^2}{\partial\theta^2}\pi_\theta(s)\|_2 \le L \quad \forall s \in S$,

then the component-wise maximum estimation error is bounded as:

$$\max_{\theta\in\Theta} \|g_j(\theta) - \widehat{g}_j(\theta)\|_2 \le \underbrace{cL_{g_j} + \frac{2\overline{m}}{c} + \zeta_j + \eta(L_{\widehat{g}_j} + L_{\widetilde{g}_j})}_{e_j}, \quad (12)$$

where $j \in \{1,\ldots,l\}$, and L_{g_j}, $L_{\widehat{g}_j}$ and $L_{\widetilde{g}_j}$ are the Lipschitz constants of g_j, $\widehat{g}_j$ and $\widetilde{g}_j$ respectively, while ζ_j is the maximum sample error at the jth component, defined as $\zeta_j = \max_{i\in\{1,\ldots,M\}} \|\widehat{g}_j(\tilde{\theta}^{(i)}) - \widetilde{g}_j(\tilde{\theta}^{(i)})\|_2$.

Under the assumptions in Proposition III.1, the Lipschitz constant L_g (and hence the Lipschitz constant L_{g_j} of the component-wise function g_j) can be bounded as $L_g \le \frac{H^2G^2R^2+L^2R^2}{1-\gamma^4}$. The Lipschitz constant $L_{\widehat{g}}$ can be directly computed from the knowledge of the Lipschitz constants of the individual kernel function κ and the vectors $\alpha_1,\ldots,\alpha_l$. Finally, the Lipschitz constant $L_{\widetilde{g}_j}$ can be computed from the samples $\{(\tilde{\theta}^{(i)}, \widetilde{g}_j(\tilde{\theta}^{(i)})\}_{i=1}^M$ that were used to define the function $\widetilde{g}_j$ using methods from optimization theory [11], [12] since we have samples of the function, not the analytical form of the function.

Remark 1. *The assumptions in Proposition III.1 are standard assumptions in the reinforcement learning literature, see for example [13], [14], and the references within.*

[1]In practice, we use feature-based MKL which ensures that the dimension of the kernel vector k does not increase with the number of samples, see [10].

B. Step 2: Solve a sequence of feasibility problems to identify critical points:

The next step is to use the estimated function $\widehat{g}$ (and the error margin e_j in Proposition III.1) to identify candidate neural network parameters $\hat{\theta}^{(1)}, \hat{\theta}^{(2)}, \ldots$ that are guaranteed to be close enough to the critical points of J. To that end, we solve the following feasibility problem:

$$\exists \hat{\theta}^{(i)} \in \Theta. \left[-\overline{e} \le \widehat{g}(\hat{\theta}^{(i)}) \le \overline{e} \bigwedge \left(\bigvee_{j=2}^{i-1} \|\hat{\theta}^{(i)} - \hat{\theta}^{(j)}\| \ge b \right) \right], \quad (13)$$

while adding the solutions obtained from the previous iterations $\hat{\theta}^{(0)}, \hat{\theta}^{(i-1)}$ to the ith iteration of solving (13) until no more solutions can be found. The $-\overline{e} \le \widehat{g}(\hat{\theta}^{(i)}) \le \overline{e}$ constraint in (13) ensures that the identified $\hat{\theta}^{(i)}$ is close to the local/global maxima of J. In this constraint, the vector $\overline{e} \in \mathbb{R}^l$ is defined as $\overline{e} = (e_1,\ldots,e_j,\ldots,e_l)$ where e_j is the right-hand side of (12). The second set of constraints

Algorithm 1 Offline Phase

Input: Objective function J, Dynamics f_d, Kernel vector $k(\theta)$, Neural network architecture/parameters bound Θ, and attacker-threshold ψ.
Output: Candidate policy estimates $\{\pi_{\hat{\theta}^{(1)}},\ldots,\pi_{\hat{\theta}^{(o)}}\}$

Set $b \le \frac{\psi}{G}$
Sample uniformly from the policy parameter space Θ along a grid of size η to obtain $\tilde{\theta}^{(1)},\ldots,\tilde{\theta}^{(M)}$
for $j \in \{1,...,l\}$ **do**
  for $\tilde{\theta} \in \{\tilde{\theta}^{(1)},\ldots,\tilde{\theta}^{(M)}\}$ **do**
    Evaluate $\overline{J}$ to obtain:

$$\widetilde{g}_j(\tilde{\theta}) = \frac{\overline{J}(\tilde{\theta}+c_j)-\overline{J}(\tilde{\theta})}{c}$$

  end for
  Use the samples $\{(\tilde{\theta}^{(i)}, \widetilde{g}_j(\tilde{\theta}^{(i)})\}_{i=1}^M$ to train a multi-kernel learner and construct the function estimator $\widehat{g}_j$ as:

$$\widehat{g}_j(\theta) = \alpha_j^T k(\theta)$$

end for
Set `Finished`:= FALSE and $i := 0$
while `Finished` = FALSE **do**
  $i := i + 1$
  Use SMT solver to find a feasible solution of:

$$\exists \hat{\theta}^{(i)} \in \Theta. \left[-\overline{e} \le \widehat{g}(\hat{\theta}^{(i)}) \le \overline{e} \bigwedge \left(\bigvee_{j=2}^{i-1} \|\hat{\theta}^{(i)} - \hat{\theta}^{(j)}\| \ge b \right) \right]$$

end while
Set $o :=$ number of solutions found by the SMT solver
return Candidate policy estimates $\{\pi_{\hat{\theta}^{(1)}},\ldots,\pi_{\hat{\theta}^{(o)}}\}$

$\bigvee_{j=2}^{i-1} \|\hat{\theta}^{(i)} - \hat{\theta}^{(j)}\| \geq b$ ensures that the distance between $\hat{\theta}^{(i)}$, at iteration i, and those found in the previous iterations $\hat{\theta}^{(1)}, \hat{\theta}^{(2)}, \ldots, \hat{\theta}^{(i-1)}$ is larger than a user-defined parameter $b \in \mathbb{R}$ with $b > 0$.

For each iteration, $i = 1, 2, \ldots$, the feasibility problem in (13) can be efficiently solved using Satisfiability Modulo Theory (SMT) solvers. SMT solvers are capable of solving feasibility problems that can be encoded as first-order logic formulas over the real numbers and has been widely used to solve numerous problems in computer science and control theory literature [15]–[18]. Examples are dReal which can handle analytic functions [19], PolyAR and PolyARBerNN which are optimized to handle polynomial functions [20] [21], and Z3 which is a generic SMT solver [22]. The next result captures the correctness of this iterative procedure.

Proposition III.2. *Consider the set of neural network parameters $\{\hat{\theta}^{(1)}, \ldots, \hat{\theta}^{(o)}\}$ computed by iteratively solving the feasibility problem* (13) *for $h = 1, \ldots, o$ until no more solutions can be found, where $b > 0$ is any user-defined threshold. Under the assumptions in Proposition III.1, the following holds:*

$$\exists \hat{\theta}^{(i^*)} \in \{\hat{\theta}^{(1)}, \ldots, \hat{\theta}^{(o)}\} \text{ such that } \|\theta^* - \hat{\theta}^{(i^*)}\|_2 \leq b. \tag{14}$$

Algorithm 1 summarizes the steps in the offline phase, and the following Theorem represents the correctness guarantees of the candidate policies obtained from the offline phase.

Theorem III.3. *Consider an unknown neural network controller π_{θ^*} with θ^* defined as in* (4)*. For any attacker-specified threshold $\psi > 0$, and parameter $b \leq \frac{\psi}{G}$:*

$$\exists \pi_{\hat{\theta}^{i^*}} \in \{\pi_{\hat{\theta}^{(1)}}, \ldots, \pi_{\hat{\theta}^{(o)}}\} \text{ s.t. } \max_{s \in S} \|\pi_{\theta^*}(s) - \pi_{\hat{\theta}^{(i^*)}}(s)\|_2 \leq \psi, \tag{15}$$

where $\{\pi_{\hat{\theta}^{(1)}}, \ldots, \pi_{\hat{\theta}^{(o)}}\}$ is the set of polices computed by Algorithm 1.

In other words, Theorem III.3 ensures that one of the policies computed by Algorithm 1 is guaranteed to satisfy the attacker-provided threshold. Nevertheless, Algorithm 1 is not guaranteed to identify which computed policies are the closest to the unknown policy. This motivates the online phase of our algorithm, discussed in the next section.

IV. Online Phase: Policy Estimate Selection

In this phase, we shortlist the set of final policy estimates π_{Str} from all the candidate policies of the offline phase $\{\pi_{\hat{\theta}^{(1)}}, \ldots, \pi_{\hat{\theta}^{(o)}}\}$. This is done by comparing the outputs of the policies with those observed from the unknown policy. The selection is done based on the user-specified threshold ψ in Theorem III.3. That is, once a new pair $(s^{(k)}, \pi_{\theta^*}(s^{(k)}))$ is observed at time k, we evaluate all the candidate policies to obtain $\pi_{\hat{\theta}^{(i)}}(s^{(k)})$, for $i \in \{1, \ldots, r\}$. Next, we compare these outputs to the observed $\pi_{\theta^*}(s^{(k)})$ and only leave those policies that produce actions within the ψ threshold. Algorithm 2 summarizes the steps in the online phase. It is direct to show that Theorem 3 still holds for the shortlisted policies π_{Str} as captured by the Corollary IV.1.

Algorithm 2 Online Phase

Input: Set of the shortlisted candidate policy estimates π_{Str}, set of the discarded policy estimates π_{Dis}, and attacker-threshold ψ.
Before the first run of the algorithm, these two sets are initialized to $\pi_{\text{Str}} = \{\pi_{\hat{\theta}^{(1)}}, \ldots, \pi_{\hat{\theta}^{(o)}}\}$ and $\pi_{\text{Dis}} = \phi$.
Output: An updated set of shortlisted policy estimate(s) π_{Str} and an updated set of discarded policy estimates π_{Dis}

1: Obtain the samples $(s^{(k)}, \pi_{\theta^*}(s^{(k)}))$ as they become available.
2: **for** $\pi_{\hat{\theta}} \in \pi_{\text{Str}}$ **do**
3: Evaluate the neural network $\pi_{\hat{\theta}}$ to get the estimated control action $\pi_{\hat{\theta}}(s^{(k)})$
4: **if** $\|\pi_{\hat{\theta}}(s^{(k)}) - \pi_{\theta^*}(s^{(k)})\|_2 > \psi$: **then**
4: $\pi_{\text{Dis}} = \pi_{\text{Dis}} \bigcup \{\pi_{\hat{\theta}}(s^{(k)})\}$
4: $\pi_{\text{Str}} = \pi_{\text{Str}} / \{\pi_{\hat{\theta}}(s^{(k)})\}$
5: **end if**
6: **end for**
7: $q = |\pi_{\text{Str}}|$ is number of policies satisfying the threshold
8: **return** π_{Str} and π_{Dis}

Corollary IV.1. *Consider an unknown neural network controller π_{θ^*} with θ^* defined as in* (4)*. For any attacker-specified threshold $\psi > 0$:*

$$\exists \pi_{\hat{\theta}^{i^*}} \in \pi_{Str} \text{ s.t. } \max_{s \in S} \|\pi_{\theta^*}(s) - \pi_{\hat{\theta}^{(i^*)}}(s)\|_2 \leq \psi, \tag{16}$$

where π_{Str} is the set of policies computed by Algorithm 2.

We note that the reduction in the number of shortlisted policies (compared to those produced in the offline phase) depends on the quality and quantity of the collected data. Nevertheless, as we discuss in the next section, even very few samples can be enough to reduce the number of shortlisted policies to a small number.

V. Numerical Experimentation

A. Environment Setup 1

We use the widely used platform, OpenAI Gym utilizing Box2D Physics engine. It is a toolkit for reinforcement learning research [23], which provides various dynamical systems and control tasks, amongst others, for testing. The exact dynamics are unknown to the attacker.

We use the classic inverted pendulum task as an example, shown in Figure 1, with the default parameters of the physical environment in the OpenAI Gym, which results in the following state space dynamics:

$$\frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} x \\ y \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & \frac{3gr}{2l} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{\beta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{3}{ml^2} \end{bmatrix} u, \tag{17}$$

where the state vector $(x, y, \dot{\beta})$ represents the position and the angular velocity of the pendulum, m is the mass and l is the length of the pendulum, gr is the acceleration of gravity, and t is the time step. We used the following reward function:

$$r = -(\beta^2 + 0.1\dot{\beta}^2 + 0.001u^2), \tag{18}$$

to train a neural network controller using actor-critic PPO with $F = 3$ and $l = 11$. Indeed, the learned θ^* is unknown to the attacker.

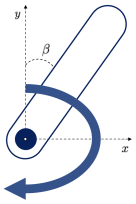

Fig. 1: Inverted Pendulum

B. Numerical Results 1

We first run our offline phase and collect all the candidate policy estimates $\{\pi_{\hat{\theta}^{(1)}}, \ldots, \pi_{\hat{\theta}^{(r)}}\}$. We use $c = 0.05$ and $\psi = 0.04$. We estimated the parameters G, L_g, $L_{\hat{g}}$ and $L_{\tilde{g}}$ from data. Using these parameters, we set $b = 0.02$, and by running Algorithm 1, we obtained $o = 2047$ candidate policies. Figure 2 shows the histogram of the error $\|\theta^* - \hat{\theta}^{(i)}\|_2$ across all identified candidate policies. As can be observed, 5 candidate NN parameters are within the bound b promised by Proposition III.2.

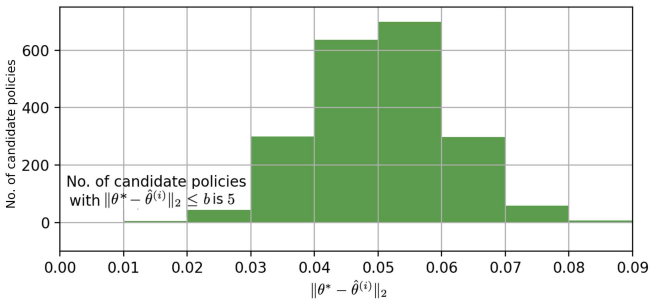

Fig. 2: Histogram of the error $\|\theta^* - \hat{\theta}^{(i)}\|_2$ across all identified candidate policies by Algorithm 1.

Next, we ran Algorithm 2 to get the final set of candidate policy estimates π_{Str}. To illustrate the power of Algorithm 2 in terms of its ability to shortlist the candidate policies, we ran Algorithm 2 for 25 times from different initial conditions of the system. We observed $N = 150$ samples from the unknown RL agent in all runs. In all runs of Algorithm 2, it could eliminate 99% of candidate policies. More specifically, the number of shortlisted policies $q = |\pi_{\text{Str}}|$ varied between 1 and 13, averaging 4.24 across the 25 runs. Figure 3 shows the results of one of these runs which resulted in $\pi_{\text{Str}} = \{\pi_{\hat{\theta}^{(584)}}, \pi_{\hat{\theta}^{(944)}}\}$. In particular, Figure 3 shows the error $\|\pi_{\hat{\theta}^{(i)}}(s^{(k)}) - \pi_{\theta^*}(s^{(k)})\|_2$ across the trajectory of collected observations for both $\pi_{\hat{\theta}^{(584)}}$ and $\pi_{\hat{\theta}^{(944)}}$. As can be observed from the figure, both candidates maintain an error that is below the user-provided threshold of $\psi = 0.04$.

In Figure 4, we randomly sample across the entire state space and plot the error for the two shortlisted candidate policies π_{Str}. As promised by Corollary IV.1, at least one of the identified policies has a maximum error of less than $\psi = 0.04$. More specifically the maximum error for $\pi_{\hat{\theta}^{(584)}} = 0.037$.

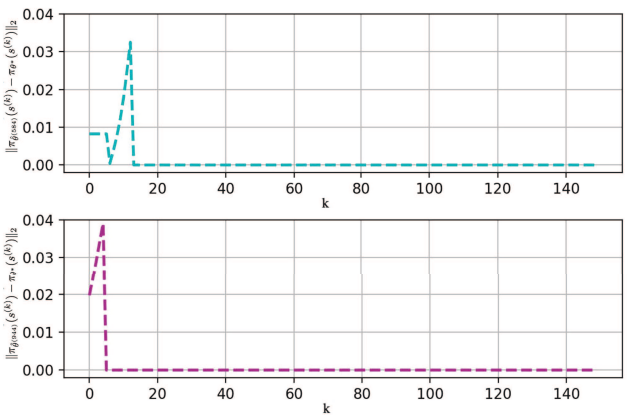

Fig. 3: Error $\|\pi_{\hat{\theta}}(s^{(k)}) - \pi_{\theta^*}(s^{(k)})\|_2$ of all candidate policies $\pi_{\hat{\theta}} \in \pi_{\text{Str}}$, where $\psi = 0.04, o = 2047, q = 2$, and $\pi_{\text{Str}} = \{\pi_{\hat{\theta}^{584}}, \pi_{\hat{\theta}^{944}}\}$, generated by Algorithm 2.

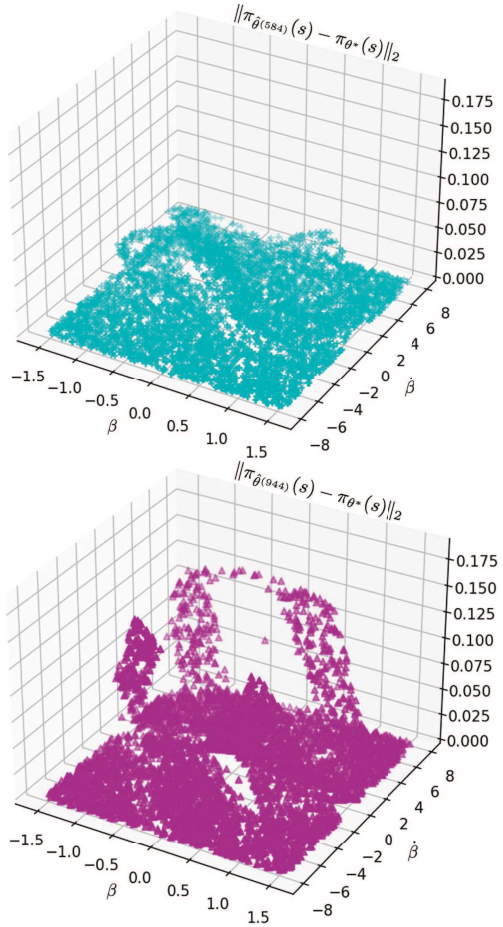

Fig. 4: Error $\|\pi_{\hat{\theta}}(s) - \pi_{\theta^*}(s)\|_2$, over uniform samples across the state space of $\pi_{\hat{\theta}} \in \pi_{\text{Str}}$ generated by Algorithm 2.

C. Environment Setup 2

Next, we used the OpenAI Gym mountain car example where the car is placed stochastically at the bottom of a sinusoidal valley, with the possible actions being acceleration in either direction. The goal is to reach the goal state on top of the right hill, shown in Figure 5. The walls at both the ends are inelastic. The system dynamics follow the following

nonlinear dynamics:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} p \\ v \end{bmatrix} = \begin{bmatrix} v \\ 0.001u - 0.0025\cos(3p) - 0.0025v^2\sin(3p) \end{bmatrix}. \tag{19}$$

The reward function used for training is:

$$\begin{aligned} r &= -0.1u^2 \text{ for each t not reached goal} \\ r &= +100 \quad \text{reached the goal} \end{aligned} \tag{20}$$

We used actor-critic PPO to train a neural network controller with 3 layers and 13 neurons/layer. Indeed the learned θ^* is unknown to the attacker.

This setup involves non-linear dynamics with multiple local optima, making it difficult to find a globally optimal policy to solve the problem. This setup is known to suffer from several local minima, and the reward function is sparse and delayed, providing limited feedback to the agent during the learning process.

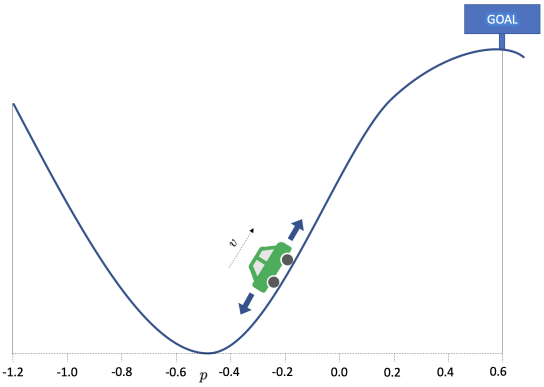

Fig. 5: Mountain Car

D. Numerical Results 2

We first run our offline phase and collect all the candidate policy estimates $\{\pi_{\hat{\theta}^{(1)}}, \ldots, \pi_{\hat{\theta}^{(r)}}\}$. We use $c = 0.05$ and we set $\psi = 0.55$. We estimated the parameters G, L_g, $L_{\hat{g}}$ and $L_{\tilde{g}}$ from data. Using these parameters, we set $b = 0.37$ and by running Algorithm 1, we obtained $o = 9136$ candidate policies. Figure 6 shows the histogram of the error $\|\theta^* - \hat{\theta}^{(i)}\|_2$ across all identified candidate policies. As can be observed, there exist 7 candidate NN parameters that are within the bound b promised by Proposition III.2.

Next, we ran Algorithm 2 to get the final set of candidate policy estimates π_{Str}. We ran Algorithm 2 for 25 times from different initial conditions of the system. We observed $N = 150$ samples from the unknown RL agent in all runs. In all runs of Algorithm 2, it could eliminate most candidate policies. More specifically, the number of shortlisted policies $q = |\pi_{\text{Str}}|$ varied between 2 and 23, averaging 6.13 across the 25 runs.

Figure 7 shows the results of one of these runs, which resulted in $\pi_{\text{Str}} = \{\pi_{\hat{\theta}6250}, \pi_{\hat{\theta}6252}, \pi_{\hat{\theta}3599}\}$. In particular, Figure 7 shows the error $\|\pi_{\hat{\theta}^{(i)}}(s^{(k)}) - \pi_{\theta^*}(s^{(k)})\|_2$ across the trajectory of collected observations for all shortlisted policies. As can be observed from the figure, all candidates maintain an error that is below the threshold of $\psi = 0.55$. In Figure 8, we randomly sample across the entire state space and plot the error for the shortlisted candidate policies π_{Str}. As promised by Corollary IV.1, at least one of the identified

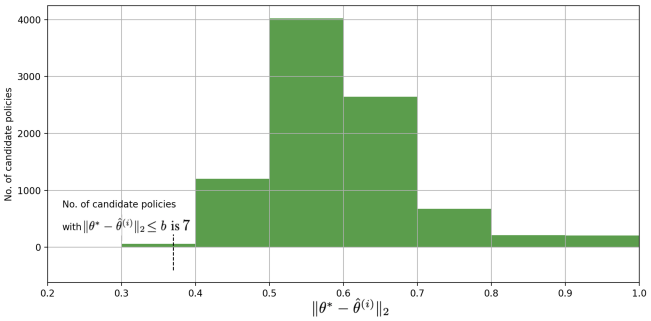

Fig. 6: Histogram of the error $\|\theta^* - \hat{\theta}^{(i)}\|_2$ across all identified candidate policies by Algorithm 1.

policies has a maximum error of less than $\psi = 0.55$. More specifically the maximum error for $\pi_{\hat{\theta}(6250)} = 0.463$.

Figures 7 and 8 highlight our assumption on the attacker's capabilities in Section II.C and, in particular, that we place no requirements on the quantity and coverage of the observations $\mathcal{D}$. As seen in Figure 7, policy $\pi_{\hat{\theta}3599}$ was shortlisted because it met the attacker threshold ψ across this trajectory. Nevertheless, the policy $\pi_{\hat{\theta}3599}$ does not, in general, satisfy the threshold ψ as shown in Figure 8.

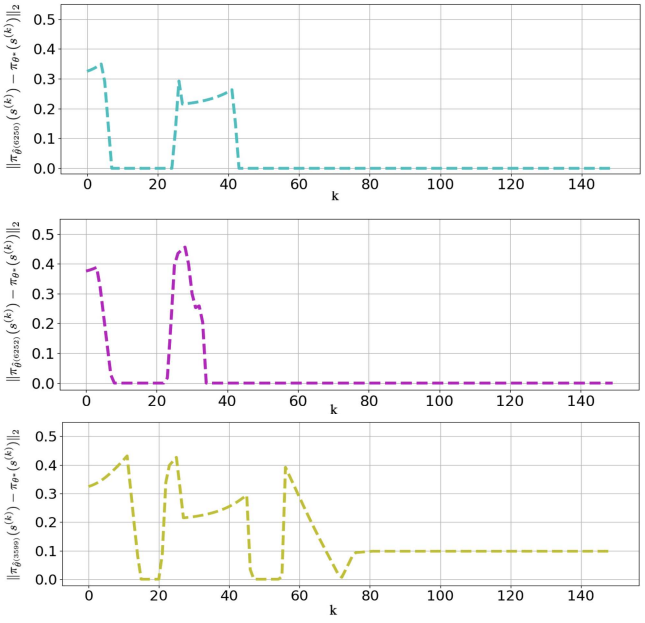

Fig. 7: Error $\|\pi_{\hat{\theta}}(s^{(k)}) - \pi_{\theta^*}(s^{(k)})\|_2$ of all candidate policies $\pi_{\hat{\theta}} \in \pi_{\text{Str}}$, where $\psi = 0.55, r = 9136, q = 3$, and $\pi_{\text{Str}} = \{\pi_{\hat{\theta}6250}, \pi_{\hat{\theta}6252}, \pi_{\hat{\theta}3599}\}$, generated by Algorithm 2.

VI. Conclusion

In this paper, we introduced the model extraction attacks on reinforcement-learning-based neural network controllers. The attacker's objective is to estimate an unknown neural network controller to predict its future control outputs. We proposed a two-phase algorithm. In the first phase, we produce a set of candidate estimates of the unknown controller using the side-channel information about the optimization problem originally used to design that controller. We then

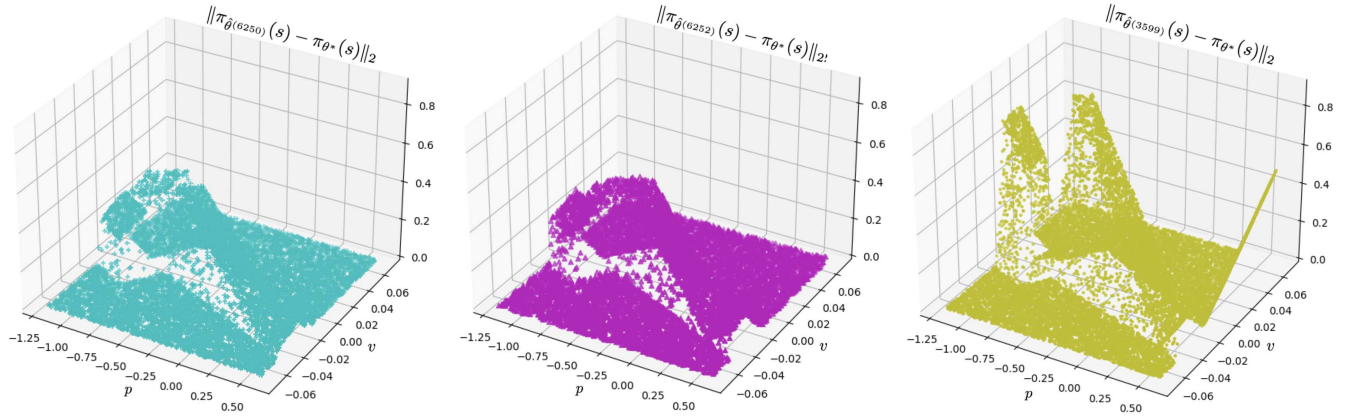

Fig. 8: Error $\|\pi_{\hat{\theta}}(s) - \pi_{\theta^*}(s)\|_2$, over uniform samples across the state space of $\pi_{\hat{\theta}} \in \pi_{\text{Str}}$ generated by Algorithm 2. The global candidate policies $\{\pi_{\hat{\theta}^{6250}}, \pi_{\hat{\theta}^{6252}}\}$ are always well below the user threshold $\psi = 0.55$.

use a limited set of the observed state/control action pairs in the online phase to shortlist the estimates from the offline phase. We provided a theoretical analysis that establishes the correctness of our algorithms. Our numerical experiments on two different sets of tasks show that only a limited amount of observations is needed to produce estimates that are close enough to the unknown controller.

REFERENCES

[1] Y. Yuan, Q. Zhu, F. Sun, Q. Wang, and T. Başar, "Resilient control of cyber-physical systems against denial-of-service attacks," in *2013 6th International Symposium on Resilient Control Systems (ISRCS)*, 2013, pp. 54–59.

[2] Y. Mo and B. Sinopoli, "Secure control against replay attacks," in *2009 47th annual Allerton conference on communication, control, and computing (Allerton)*. IEEE, 2009, pp. 911–918.

[3] P. Griffioen, R. Romagnoli, B. H. Krogh, and B. Sinopoli, "Resilient control in the presence of man-in-the-middle attacks," in *2021 American Control Conference (ACC)*, 2021, pp. 4553–4560.

[4] Y. Shoukry, P. Martin, P. Tabuada, and M. Srivastava, "Non-invasive spoofing attacks for anti-lock braking systems," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2013, pp. 55–72.

[5] Y. Shoukry and P. Tabuada, "Event-triggered state observers for sparse sensor noise/attacks," *IEEE Transactions on Automatic Control*, vol. 61, no. 8, pp. 2079–2091, 2015.

[6] G. Dán and H. Sandberg, "Stealth attacks and protection schemes for state estimators in power systems," in *2010 first IEEE international conference on smart grid communications*. IEEE, 2010, pp. 214–219.

[7] Y. Shoukry and P. Tabuada, "The secure state estimation problem," in *Safety, Security and Privacy for Cyber-Physical Systems*. Springer, 2021, pp. 123–143.

[8] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 3–18.

[9] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction {APIs}," in *25th USENIX security symposium (USENIX Security 16)*, 2016, pp. 601–618.

[10] Y. Shen, T. Chen, and G. B. Giannakis, "Random feature-based online multi-kernel learning in environments with unknown dynamics," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 773–808, 2019.

[11] G. Wood and B. Zhang, "Estimation of the lipschitz constant of a function," *Journal of Global Optimization*, vol. 8, pp. 91–103, 1996.

[12] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[13] Z. Shen, A. Ribeiro, H. Hassani, H. Qian, and C. Mi, "Hessian aided policy gradient," in *International conference on machine learning*. PMLR, 2019, pp. 5729–5738.

[14] M. Papini, D. Binaghi, G. Canonaco, M. Pirotta, and M. Restelli, "Stochastic variance-reduced policy gradient," in *International conference on machine learning*. PMLR, 2018, pp. 4026–4035.

[15] Y. Shoukry, P. Nuzzo, A. Puggelli, A. L. Sangiovanni-Vincentelli, S. A. Seshia, and P. Tabuada, "Secure state estimation for cyber-physical systems under sensor attacks: A satisfiability modulo theory approach," *IEEE Transactions on Automatic Control*, vol. 62, no. 10, pp. 4917–4932, 2017.

[16] Z. Huang, Y. Wang, S. Mitra, G. E. Dullerud, and S. Chaudhuri, "Controller synthesis with inductive proofs for piecewise linear systems: An smt-based algorithm," in *2015 54th IEEE conference on decision and control (CDC)*. IEEE, 2015, pp. 7434–7439.

[17] M. Bahavarnia, Y. Shoukry, and N. C. Martins, "Controller synthesis subject to logical and structural constraints: A satisfiability modulo theories (smt) approach," in *2020 American Control Conference (ACC)*. IEEE, 2020, pp. 5281–5286.

[18] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of model checking*. Springer, 2018, pp. 305–343.

[19] S. Gao, S. Kong, and E. M. Clarke, "dreal: An smt solver for nonlinear theories over the reals," in *International conference on automated deduction*. Springer, 2013, pp. 208–214.

[20] W. Fatnassi and Y. Shoukry, "Polyar: A highly parallelizable solver for polynomial inequality constraints using convex abstraction refinement," *IFAC-PapersOnLine*, vol. 54, no. 5, pp. 43–48, 2021.

[21] W. Fatnassi *et al.*, "Polyarbernn: A neural network guided solver and optimizer for bounded polynomial inequalities," *arXiv preprint arXiv:2204.05365*, 2022.

[22] L. d. Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[24] M. Yano, J. D. Penn, G. Konidaris, and A. T. Patera, "Math, numerics, & programming (for mechanical engineers)," 2012.

APPENDIX

A. Proof of Proposition III.1

Proof. In order to bound the error $\max_{\theta\in\Theta} \|g_j(\theta) - \widehat{g}_j(\theta)\|_2$, we split it into the two parts below:

$$\max_{\theta\in\Theta} \|g_j(\theta) - \widehat{g}_j(\theta)\|_2 \leq \max_{\theta\in\Theta} \|g_j(\theta) - \tilde{g}_j(\theta)\|_2 + \max_{\theta\in\Theta} \|\widehat{g}_j(\theta) - \tilde{g}_j(\theta)\|_2 \quad (21)$$

To bound the first term on the RHS of (21), we follow the same argument used in proving the error bound for sampled derivatives of real-valued functions in [24] as follows:

$$
\begin{aligned}
&\max_{\theta\in\Theta}\|g_j(\theta)-\widetilde{g}_j(\theta)\|_2 \\
&\quad\overset{(a)}{=}\max_{\theta\in\Theta}\|g_j(\theta)-\frac{J(\theta+c_j)-J(\theta)}{c} \\
&\qquad\qquad+\frac{m(\tilde{\theta})-m(\tilde{\theta}+c_j)}{c}\|_2 \\
&\quad\overset{(b)}{\leq}\max_{\theta\in\Theta}\|g_j(\theta)-\frac{J(\theta+c_j)-J(\theta)}{c}\| \\
&\qquad\qquad+\max_{\theta\in\Theta}\|\frac{m(\tilde{\theta})-m(\tilde{\theta}+c_j)}{c}\| \\
&\quad\overset{(c)}{\leq}\max_{\theta\in\Theta}\|g_j(\theta)-\frac{J(\theta+c_j)-J(\theta)}{c}\|+\frac{2\overline{m}}{c} \\
&\quad\overset{(d)}{=}\max_{\theta\in\Theta}\|g_j(\theta)-\frac{1}{c}(g_j(\theta)c+\frac{1}{2}g_j'(\Lambda)c^2\|_2+\frac{2\overline{m}}{c} \\
&\quad=\|\frac{1}{2}g_j'(\Lambda)c\|_2+\frac{2\overline{m}}{c} \\
&\quad\leq\frac{1}{2}\max_{\|\Lambda\|\in[\|\theta\|,\|\theta+c_j\|]}\|g_j{}'(\Lambda)\|_2 c+\frac{2\overline{m}}{c} \\
&\quad\overset{(e)}{\leq} cL_{g_j}+\frac{2\overline{m}}{c} \qquad (22)
\end{aligned}
$$

where g_j' is the derivative of g_j and $\Lambda\in\mathbb{R}^l$ where $\|\Lambda\|\in[\|\theta\|,\|\theta+c_j\|]$. Equation (a) follows from the definition of $\widetilde{g}_j(\theta)$ in equation (9). Inequality (b) uses triangular inequality, and equation (c) uses the bound on $m(\theta)$ as defined in (6). Equation (d) follows from assumptions 2 and 3 in the proposition, which ensures that the function J is twice differentiable and hence once can apply the Taylor series expansion and the Mean-Value Theorem, which ensures the existence of a Λ such that $\|\Lambda\|\in[\|\theta\|,\|\theta+c_j\|]$ for which equality holds. The inequality in (e) follows from assumptions 1-3 in the proposition, which ensure that g is Lipschitz continuous [13, Lemma 4.1] with $L_g\leq\frac{H^2G^2R^2+L^2R^2}{1-\gamma^4}$.

Next, we bound the second term on the RHS of (21):

$$
\begin{aligned}
&\max_{\theta\in\Theta}\|\widehat{g}_j(\theta)-\tilde{g}_j(\theta)\|_2 \overset{(f)}{\leq} \\
&\qquad\max_{\theta\in\{\tilde{\theta}^{(1)},\ldots,\tilde{\theta}^{(M)}\}}\|\widehat{g}_j(\theta)-\tilde{g}_j(\theta)\|_2 \\
&\qquad\qquad+\max_{\theta\in\Theta\setminus\{\tilde{\theta}^{(1)},\ldots,\tilde{\theta}^{(M)}\}}\|\widehat{g}_j(\theta)-\tilde{g}_j(\theta)\|_2 \\
&\qquad\overset{(g)}{=}\zeta_j+\max_{\theta\in\Theta\setminus\{\tilde{\theta}^{(1)},\ldots,\tilde{\theta}^{(M)}\}}\|\widehat{g}_j(\theta)-\tilde{g}_j(\theta)\|_2 \\
&\qquad\overset{(h)}{\leq}\zeta_j+\eta L_{\xi_j} \qquad (23)
\end{aligned}
$$

where (f) follows from splitting the error into two parts, one at the training samples $\tilde{\theta}^{(1)},\ldots,\tilde{\theta}^{(M)}$ and another one that represents the error between the training samples. Equation (g) follows from the definition of ζ_j as the maximum sample error. Assuming the function $\xi_j(\theta)=\widehat{g}_j(\theta)-\tilde{g}_j(\theta)$ is Lipschitz continuous with a Lipschitz constant L_{ξ_j}, the maximum error between any two samples is bounded by ηL_{ξ_j} where η is the grid size used to sample $\tilde{\theta}^{(1)},\ldots,\tilde{\theta}^{(M)}$ and hence inequality (h) holds.

What is remaining is to show that the function ξ_j is Lipschitz continuous. Nevertheless, it follows from the definition of ξ_j that it is the summation of two functions $\widehat{g}_j$ and $\tilde{g}_j$. The first function $\widehat{g}_j$ is the linear combination of kernel functions that are Lipschitz continuous and hence $\widehat{g}_j$ is Lipschitz continuous with a constant $L_{\widehat{g}_j}$. Similarly, $\tilde{g}_j$ is the sample approximation of the continuous function g (thanks to the assumptions 1-3) and hence Lipschitz continuous with a constant $L_{\tilde{g}_j}$.

Thus, the Lipschitz constant L_{ξ_j} of ξ_j is equal to $(L_{\widehat{g}_j}+L_{\tilde{g}_j})$. Substituting in equation (23), we conclude that:

$$
\max_{\theta\in\Theta}\|\widehat{g}_j(\theta)-\tilde{g}_j(\theta)\|_2\leq\zeta_j+\eta(L_{\widehat{g}_j}+L_{\tilde{g}_j}) \qquad (24)
$$

The overall component-wise bound is then found by substituting equations (22) and (24) in equation (21) as:

$$
\max_{\theta\in\Theta}\|g_j(\theta)-\widehat{g}_j(\theta)\|_2\leq\underbrace{cL_{g_j}+\frac{2\overline{m}}{c}+\zeta_j+\eta(L_{\widehat{g}_j}+L_{\tilde{g}_j})}_{e_j} \qquad (25)
$$

□

B. Proof of Proposition III.2

Proof. We start by defining the sets below:

$$
\begin{aligned}
S_{\theta^*}&=\{\theta\mid g_j(\theta)=0,\ j\in\{1,\ldots,l\}\} &(26)\\
S_{\widehat{\theta}}&=\{\theta\mid -e_j\leq\widehat{g}_j(\theta)\leq e_j,\ j\in\{1,\ldots,l\}\} &(27)\\
S_{B_{\widehat{\theta}}}&=\mathbb{B}_b(\hat{\theta}^{(1)})\cup\mathbb{B}_b(\hat{\theta}^{(2)})\cup\ldots\cup\mathbb{B}_b(\hat{\theta}^{(o)}) &(28)
\end{aligned}
$$

Indeed, it is direct to show that equation (14) is equivalent to showing that $\theta^*\in S_{B_{\widehat{\theta}}}$. To show the correctness of this statement, we aim to prove the following sequence of set inclusion:

$$
\theta^*\overset{(a)}{\in}S_{\theta^*}\overset{(b)}{\subseteq}S_{\widehat{\theta}}\overset{(c)}{\subseteq}S_{B_{\widehat{\theta}}} \qquad (29)
$$

The set membership (a) follows from the definition of S_{θ^*} along with the fact that θ^* is a critical point of J (recall equation (4)) and hence $g(\theta^*)=\frac{d}{d\theta}J(\theta)|_{\theta=\theta^*}$ is equal to zero. The set inclusion (b) can be shown as follows. First, we note that the following holds for any $\theta\in S_{\theta^*}$:

$$
\|g_j(\theta)-\widehat{g}_j(\theta)\|_2\overset{(e)}{=}\|\widehat{g}_j(x)\|_2\overset{(f)}{=}|\widehat{g}_j(x)|\overset{(g)}{\leq}e_j \qquad (30)
$$

where (e) follows from the definition of S_{θ^*} which ensures that $g_j(\theta)=0$, (f) follows from the fact that $\widehat{g}_j(x)$ is a scalar and hence $\|\widehat{g}_j(x)\|_2=\sqrt{(\|\widehat{g}_j(x)\|_2)^2}=|\widehat{g}_j(x)|$, and (g) follows from (25). Hence we conclude that each $\theta\in S_{\theta^*}$ satisfies the conditions of $S_{\widehat{\theta}}$ which proves (b) above. Finally, the set membership in (c) follows from equation (13), which concludes the proof.

□

C. Proof of Theorem III.3

Proof. It follows from the assumptions in Proposition III.1 that π is a Lipschitz continuous function with a Lipschitz constant G and hence the following holds for any $s\in S$:

$$
\begin{aligned}
\|\pi_{\theta^*}(s)-\pi_{\hat{\theta}^{(i^*)}}(s)\|_2&\overset{(a)}{\leq}G\|\theta^*-\hat{\theta}^{(i^*)}\|_2\\
&\overset{(b)}{\leq}Gb\\
&=\psi
\end{aligned}
$$

where (a) is implied by assumption (2) in Proposition III.1 and (b) follows from Proposition III.2. □