

Convex Optimization-based Policy Adaptation to Compensate for Distributional Shifts

Navid Hashemi¹, Justin Ruths² and Jyotirmoy Deshmukh¹

¹ University of Southern California, Los Angeles CA, USA

² University of Texas at Dallas, Dallas TX, USA

Abstract. Many real-world systems often involve physical components or operating environments with highly nonlinear and uncertain dynamics. A number of different control algorithms can be used to design optimal controllers for such systems, assuming a reasonably high-fidelity model of the actual system. However, the assumptions made on the stochastic dynamics of the model when designing the optimal controller may no longer be valid when the system is deployed in the real-world. The problem addressed by this paper is the following: Suppose we obtain an optimal trajectory by solving a control problem in the training environment, how do we ensure that the real-world system trajectory tracks this optimal trajectory with minimal amount of error in a deployment environment. In other words, we want to learn how we can adapt an optimal trained policy to distribution shifts in the environment. Distribution shifts are problematic in safety-critical systems, where a trained policy may lead to unsafe outcomes during deployment. We show that this problem can be cast as a nonlinear optimization problem that could be solved using heuristic method such as particle swarm optimization (PSO). However, if we instead consider a convex relaxation of this problem, we can learn policies that track the optimal trajectory with much better error performance, and faster computation times. We demonstrate the efficacy of our approach on tracking an optimal path using a Dubin’s car model, and collision avoidance using both a linear and nonlinear model for adaptive cruise control.

1 Introduction

Systems operating in highly uncertain environments are often modeled as stochastic dynamical systems that satisfy Markov assumptions, i.e. Markov decision processes (MDPs). Given a state s_t (i.e., the state at time t), a (discrete-time) MDP defines a distribution on s_{t+1} conditioned on s_t and the control action at time t (denoted a_t). We call this distribution the transition dynamics. For such systems, a number of model-based and data-driven control design methods have been explored to learn an optimal policy (i.e. a function from the set of states to the set of actions) that minimizes some control-theoretic cost function [14]. Model-based methods explicitly, and data-driven methods implicitly, assume a specific distribution for the transition dynamics. However, when the system is deployed in the real-world, this distribution may not be the same; this change in distribution is called a *distribution shift*.

The fundamental problem addressed by this paper is adapting a pre-learned control policy to compensate for distribution shifts. While it is possible to re-train the control policy on the new environment, it is typically expensive to learn the optimal control policy. However, a crucial observation that we make is that while learning an optimal policy is expensive, learning a reasonable high-fidelity model of the transition dynamics may be feasible. We call such a learned model a *surrogate model*. In this paper, we show that under certain kinds of distribution shifts, the problem of adapting an existing optimal policy to the new deployment environment can be framed as a nonlinear optimization problem over the optimal trained trajectory and the surrogate model. Furthermore, we show that if the surrogate is a neural network (with rectified linear unit or ReLU based activation), then there is a convex relaxation of the original optimization problem. This convex relaxation permits an efficient procedure to find a modified action that minimizes the error between the optimal trained trajectory and the system trajectory in the deployment environment. Finally, we empirically show that if the trained trajectory meets desired objectives of safety, then such policy adaptation can provide safety in the deployment setting.

The main technical idea in our work is inspired by recent work in [8], where the authors proposed an efficient method to provide probabilistic bounds on the output of a neural network, given a Gaussian distribution on its inputs. We show how we can use this result to propagate the effects of a distribution shift. However, the result in [8] does not consider the problem of finding optimal actions (which is a non-convex problem). In this research, we propose a methodology to convexify this result for finding optimal actions. We demonstrate our technique on a tracking problem using a Dubin’s car model and a collision avoidance problem that uses adaptive cruise control.

The rest of the paper is organized as follows. In Section 2 we discuss the preliminaries, terminology and technical notation. In Section 3, we discuss our policy adaptation approach, and provide experimental results in Section 4. We conclude with related work in Section 5.

2 Preliminaries

Notation. A multi-variate Gaussian distribution is denoted as $\mathcal{N}(\mu, \Sigma)$, where μ and Σ represent the mean vector and covariance, respectively. For a Gaussian-distributed random vector $r \in \mathbb{R}^n$, we denote its mean value by μ_r and its covariance by Σ_r . Let $c \in \mathbb{R}^n$, then an ellipsoid centered at c with the *shape matrix* Ω is denoted as $\mathcal{E}(c, \Omega)$, i.e., $\mathcal{E}(c, \Omega) = \{x \mid (x - c)^\top \Omega^{-1} (x - c) \leq 1\}$. Given a non-convex set, \mathcal{Y} we use the notation $\mathcal{H}(\mathcal{Y})$ to denote the set of ellipsoids that contain \mathcal{Y} , i.e., $\{\mathcal{E}(c, \Omega) \mid \mathcal{Y} \subseteq \mathcal{E}(c, \Omega)\}$.

Markov Decision Process, Optimal Policy. We now formalize the notion of the type of stochastic dynamical systems that we address in this paper as a Markov Decision process.

Definition 1 (Markov Decision Process (MDP)). *A Markov decision process is a tuple $M = (S, A, T, \iota)$, where S and A denote the set of states and*

actions respectively, $T(s' | s, a)$ is the probability distribution on the next state conditioned on the current state and action, and ι is a distribution on S that is sampled to identify an initial state of the MDP³.

In our approach, we are interested in *finite-horizon* trajectories sampled from the MDP’s transition dynamics. A *policy* $\pi(a | s)$ of the MDP is a distribution on the set of actions conditioned on the current state. Given a fixed policy of the MDP, a T -length *trajectory* (denoted τ) or behavior of the MDP is a sequence of states s_0, \dots, s_T such that $s_0 \sim \iota$, and for all $t \in [0, T-1]$, $s_{t+1} \sim T(s' | s_t, \pi(s_t))$. In control-design problems, we assume that there is a *cost function* J on the space of trajectories that maps each trajectory to real value. An optimal policy π^* is defined as the one that minimizes the expected value of the cost function over trajectories starting from a state s_0 sampled according to the initial distribution ι .

Distribution shifts. Obtaining an optimal control policy is often a computationally expensive procedure for MDPs where the underlying transition dynamics are highly nonlinear. Several design methods, both model-based methods such as model-predictive control [10], stochastic optimal control [2], and model-free methods such as data predictive control [12] and deep reinforcement learning [17] have been proposed to solve the optimal control problem for such systems. Regardless of whether the method is model-based or model-free, these methods explicitly or implicitly assume a model or the distribution encoded by the transition dynamics of the environment. A key issue is that this distribution may change once the system is deployed in the real-world. To differentiate between the *training* environment and the *deployment* environment, we use T_{trn} and T_{dpl} to respectively denote the transition distributions.

Problem Definition. Suppose we have a system where we have trained an optimal policy π^* under the transition dynamics T_{trn} , and for a given initial state s_0 sampled from ι , we sample an optimal trajectory τ_{opt} for the system using the policy π^* . We denote this as $\tau \sim (\iota, \pi^*)$. Let $\tau = (s_0, s_1, \dots, s_T)$. Let $\tau(t)$ be short-hand to denote s_t . For a trajectory that starts from the same initial state (but in the deployment environment), we want to find the adapted policy $\hat{\pi}$ such that the error between τ_{opt} and the trajectory under T_{dpl} dynamics at time instant $t \in [1, T]$ is small. Formally,

$$\hat{\pi} = \arg \min_{\pi} \mathbb{E}_{\substack{a_t \sim \pi(a | s_t), \\ s_{t+1} \sim T_{dpl}(s' | s_t, a_t)}} \|\tau_{opt}(t+1) - s_{t+1}\| \quad (1)$$

A key challenge in solving the optimization problem in (1) is that T_{dpl} is not known. In this paper, we propose that we learn a *surrogate model* for the deployment transition dynamics. Essentially, a surrogate model is a data-driven

³ Technically, this definition pertains to the transition structure of a stochastic dynamical system. Typically, dynamical systems are defined in terms of difference or differential equations describing the temporal evolution of a state variable. We assume that $T(s' | s, a)$ is thus the infinite set of transitions consistent with any given system dynamics.

model that approximates the actual system dynamics reasonably accurately. There are several choices for surrogate models including Gaussian Processes [1], probabilistic ensembles [4], and deep neural networks (NN). In this paper, we focus on NN surrogates as they allow us to consider convex relaxations of the policy adaptation problem.

Surrogate-based policy adaptation. We now show that surrogate-based policy adaptation can be phrased as a nonlinear optimization problem. First we specify the problem of finding good surrogates. Recall that $T_{dpl}(s_{t+1} | s_t, a_t)$ is assumed to be a time-invariant Gaussian distribution with mean $\mu(s, a)$ and covariance $\Sigma(s, a)$. A surrogate model for the transition dynamics is a tuple $(\mu_{NN}(s, a; \theta_\mu), \Sigma_{NN}(s, a; \theta_\Sigma))$, where μ_{NN} and Σ_{NN} are deep neural networks with parameters θ_μ and θ_Σ respectively. We can train such NNs by minimizing the following loss functions:

$$\mathcal{L}_\mu(\theta_\mu) = \mathbb{E}_{s \sim S, s' \sim T_{dpl}(s' | s, a)} \|\mu_{NN}(s, a; \theta_\mu) - s'\| \quad (2)$$

$$\mathcal{L}_\Sigma(\theta_\Sigma) = \mathbb{E}_{s \sim S} \|\Sigma_{NN}(s, a; \theta_\Sigma) - \Sigma_s(s')\| \quad (3)$$

In the above equations, the expectation is computed by standard Monte Carlo based sampling. In the second equation, Σ_s represents the sample covariance of s' w.r.t. the sample mean.

Assuming that we have learned surrogate models to a desired level of accuracy, the next step is to frame policy adaptation as a nonlinear optimization problem. We state the problem w.r.t. a specific optimal trajectory τ_{opt} sampled from the optimal policy (though the problem generalizes to any optimal trajectory sampled from an arbitrary initial state). Note that $\tau_{opt}(0) = s_0$.

$$\forall t \in [0, T-1] : a_t = \arg \min_{a \in A} \|\tau_{opt}(t+1) - \mu_{NN}(s_t, a_t; \theta_\mu)\| \quad (4)$$

We observe that as the equation above consists of a neural network, it is highly nonlinear optimization problem. In the next section, we will show how we can convexify this problem.

3 Policy adaptation

Solution Overview. The quantity in Eq. (4) being minimized is at each time t , the *residual* error between the optimal trajectory and the mean predicted state by the deployment environment, conditioned on its state and action. Let $r_{t+1} = \tau_{opt}(t+1) - \mu_{NN}(s_t, a_t; \theta_\mu)$. Our main idea is:

1. At any given time t , assume that the state s_t lies in a confidence set described by an ellipsoid $\mathcal{E}(\mu_{s_t}, \Omega_{s_t})$,
2. Assume that the action a_t lies in a confidence set also described by an ellipsoid $\mathcal{E}(\mu_{a_t}, \Omega_{a_t})$,
3. Show that the residual error r_t can be bounded by an ellipsoid, the center and shape matrix of which depends on the action a_t .

4. Find the action a_t that minimizes the residual error by convex optimization.

We now explain each of these steps in sequence. First, we motivate why need to consider confidence sets. Suppose the system starts in state s_0 , then the state s_1 is distributed according to the transition dynamics of the deployment environment. In reality, we are only interested in the next states that are likely with at least probability threshold p . For a multi-variate Gaussian distribution, this corresponds to the sublevel set of the inverse CDF of this distribution, which according to the following lemma can be described by an ellipsoid:

Lemma 1. *A random vector $r \in \mathbb{R}^n$, with Gaussian distribution $r \sim \mathcal{N}(\mu, \Sigma)$, satisfies,*

$$\Pr \left[\frac{1}{\rho_n} (r - \mu)^\top \Sigma^{-1} (r - \mu) \leq 1 \right] = p, \quad (5)$$

where, $\rho_n = \Gamma^{-1}(\frac{n}{2}, \frac{p}{2})$ and $\Gamma^{-1}(\cdot, \cdot)$ indicates the n dimensional lower incomplete Gamma function.

The above lemma allows us to define ellipsoidal confidence sets using truncated Gaussian distributions. An ellipsoidal confidence region with center μ and shape matrix $\rho_n \Sigma$ (where ρ_n is as defined Lemma 1) defines a set with probability measure p .

Now, as the policy we are considering is stochastic (which we also model as a Gaussian distribution), an action that can be taken is described by a conditional Gaussian distribution. Let μ_{a_t} be the mean of the distribution of the action at time t , then all actions with probability greater than p can be described by a ellipsoid confidence set $\mathcal{E}(\mu_{a_t}, \Omega_{a_t})$.

Because the distribution of transition dynamics may have shifted, applying the same action $\pi^*(s_t)$ may result in a residual error r_{t+1} that is unacceptable. So, we want to find a new action, a_t which reduces the residual error. We assume that a_t is in an ellipsoidal uncertainty set by picking actions that have probability greater than a fixed threshold p . We note that the center or the shape matrix of the ellipsoidal set for the action is not known, but is a decision variable for the optimization problem.

We note that the relation between a_t and r_{t+1} is highly nonlinear. However, we show, how we can convexify this problem.

Before we present the convexification of the optimization problem, we need to introduce the notion of the reachable set of residual values. We call this the *residual reach set*. Formally, given ellipsoidal confidence region $\mathcal{E}(\mu_{s_t}, \Omega_{s_t})$ for the state s_t , and the ellipsoidal confidence region $\mathcal{E}(\mu_{a_t}, \Omega_{a_t})$ for the action a_t , the residual reach set \mathcal{R}_{t+1} is defined as follows:

$$\mathcal{R}_{t+1}(\mu_{a_t}, \Omega_{a_t}) = \{ \mu_{NN}(s_t, a_t; \theta_\mu) - \tau_{opt}(t+1) \mid s_t \in \mathcal{E}(\mu_{s_t}, \Omega_{s_t}), a_t \in \mathcal{E}(\mu_{a_t}, \Omega_{a_t}) \} \quad (6)$$

In the above equation, we note that the residual reach set is parameterized by a_t and Ω_{a_t} , and we wish to find the values for a_t and Ω_{a_t} that minimize the size of the residual reach set. However, the residual reach set is a non-convex set. To make the optimization problem convex, we basically approximate the

residual reach set by an ellipsoidal upper bound in the set $\mathcal{H}(\mathcal{R}_{t+1}(\mu_{a_t}, \Omega_{a_t}))$ (the set of all ellipsoidal upper bounds).

We can now express the problem of finding the best adapted action distribution as the following optimization problem:

$$\begin{aligned} (\hat{\mu}_{a_t}, \hat{\Omega}_{a_t}, \hat{\Omega}_{\mathcal{R}_{t+1}}) &= \arg \min_{\mu_{a_t}, \Omega_{a_t}} \mathbf{Logdet}(\Omega_{\mathcal{R}_{t+1}}) \\ &\text{s.t. } \mathcal{R}_{t+1}(\mu_{a_t}, \Omega_{a_t}) \subset \mathcal{E}(\mathbf{0}, \Omega_{\mathcal{R}_{t+1}}) \end{aligned} \quad (7)$$

Consider we set the center of ellipsoidal bound of residual reach set to be 0. This is motivated by the goal that we need to minimize the size of residuals. Equation (7) selects the best action $\hat{a}_t \in \mathcal{E}(\hat{\mu}_{a_t}, \hat{\Omega}_{a_t})$ s.t. the the ellipsoid $\mathcal{E}(\mathbf{0}, \hat{\Omega}_{\mathcal{R}_{t+1}})$ is the smallest ellipsoid that bounds the residual reach set.

The construction of an ellipsoidal bound over the reach-set of a neural network given a single ellipsoidal confidence region is derived in [7]. The author has upgraded this technique later for multiple ellipsoidal confidence regions in Theorem 1 of [11]. We rephrase the key results from these papers in our context in Lemma 2.

Lemma 2. *Suppose $s_t \in \mathcal{E}(\mu_{s_t}, \Omega_{s_t})$, $a_t \in \mathcal{E}(\mu_{a_t}, \Omega_{a_t})$. Then, the residual reach-set $\mathcal{R}_{t+1}(\mu_{a_t}, \Omega_{a_t})$ is upper-bounded by $\mathcal{E}(\mathbf{0}, \Omega_{\mathcal{R}_{t+1}})$ (as defined in (7)) if the constraint in (8) holds. In what follows, $(b_\ell, W_\ell) \in \theta_\mu$ represent the bias vector and the weights of the last layer in μ_{NN} .*

$$\tau_1 M_{s_t} + \tau_2 M_{a_t} + M_\phi - M_{out} \leq 0, \quad \text{for some } \tau_1, \tau_2 \geq 0. \quad (8)$$

Here, M_ϕ is a quadratic constraint proposed in [7], representing ReLU hidden layers in the neural network and⁴,

$$\begin{aligned} M_{s_t} &= \frac{1}{\rho_n} E_1^\top \begin{bmatrix} -\Sigma_{s_t}^{-1} & \Sigma_{s_t}^{-1} \mu_{s_t} \\ \mu_{s_t}^\top \Sigma_{s_t}^{-1} & -\mu_{s_t}^\top \Sigma_{s_t}^{-1} \mu_{s_t} + \rho_n \end{bmatrix} E_1, \\ M_{a_t} &= E_2^\top \begin{bmatrix} -\Omega_{a_t}^{-1} & \Omega_{a_t}^{-1} \mu_{a_t} \\ \mu_{a_t}^\top \Omega_{a_t}^{-1} & -\mu_{a_t}^\top \Omega_{a_t}^{-1} \mu_{a_t} + 1 \end{bmatrix} E_2 \\ E_1 &= \left[\begin{array}{c|c|c} I_n & 0_{n \times m} & 0_{n \times (\sum_{i=2}^{\ell+1} N_i)} \\ \hline 0_{1 \times n+m} & 0_{1 \times (\sum_{i=2}^{\ell+1} N_i)} & 1 \end{array} \right], \\ E_2 &= \left[\begin{array}{c|c|c} 0_{m \times n} & I_m & 0_{m \times (\sum_{i=2}^{\ell+1} N_i)} \\ \hline 0_{1 \times n+m} & 0_{1 \times (\sum_{i=2}^{\ell+1} N_i)} & 1 \end{array} \right] \\ M_{out} &= \begin{bmatrix} C & b \\ 0 & 1 \end{bmatrix}^\top \begin{bmatrix} -\Omega_{\mathcal{R}_{t+1}}^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} C & b \\ 0 & 1 \end{bmatrix}, \end{aligned}$$

and $C = [0 \ 0 \ \dots \ W_\ell]$, $b = b_\ell - \tau_{opt}(t+1)$,

⁴ The parameter N_i in transformation matrices E_1, E_2 is the number of ReLU activations in layer i of μ_{NN} .

In the above lemma, as the adapted action and the shape matrix representing its covariance is assumed to be known M_{a_t} is a fixed matrix; however, in the optimization problem that we wish to solve, in the corresponding matrix M_{a_t} , μ_{a_t} and Ω_{a_t} will appear as variables, which causes the problem to become nonlinear. We can address this by performing two transformations. The first transformation, through a change of variables concentrates the nonlinearity in a single scalar entry of M_{a_t} . We set $U_{a_t} = \tau_2 \Omega_{a_t}^{-1}$, $V_{a_t} = \tau_2 \Omega_{a_t}^{-1} \mu_{a_t}$, and the resulting M_{a_t} is shown as below:

$$\begin{aligned} M_{a_t}^* &= E_2^\top \begin{bmatrix} -U_{a_t} & V_{a_t} \\ V_{a_t}^\top & -(\tau_2 \mu_{a_t}^\top \Omega_{a_t}^{-1}) \left(\frac{\Omega_{a_t}}{\tau_2} \right) (\tau_2 \Omega_{a_t}^{-1} \mu_{a_t}) + \tau_2 \end{bmatrix} E_2 \\ &= E_2^\top \begin{bmatrix} -U_{a_t} & V_{a_t} \\ V_{a_t}^\top & -V_{a_t}^\top U_{a_t}^{-1} V_{a_t} + \tau_2 \end{bmatrix} E_2 \end{aligned} \quad (9)$$

The proposed matrix, $M_{a_t}^*$, is nonlinear where the nonlinearity shows up in the scalar variable $V_{a_t}^\top U_{a_t}^{-1} V_{a_t}$.

We also note that the adapted actions should satisfy actuator bounds $[\ell, \mathbf{u}]$, we include this as a convex constraint below:

$$U_{a_t} \ell \leq V_{a_t} \leq U_{a_t} \mathbf{u}, \quad (10)$$

and we defer the proof to the appendix B. Before stating the final theorem, we make an observation about (7). Without additional constraints, the optimal solution to (7) always returns $\hat{\Omega}_{a_t}$ such that $\text{tr}(\hat{\Omega}_{a_t}) = 0$ (proof in the appendix A). This causes numerical errors, as the inverse of this matrix is appeared in Lemma 2 which is not defined. To avoid such a problem, we impose a tiny lower bound on the trace of this matrix.

Finally, given that all the constraints for the optimization of $\mathcal{E}(\mu_{a_t}, \Omega_{a_t})$ are provided, we can collect them in a convex optimization that results in the modified action set. The following Theorem characterizes the correctness of the modified action and its conservatism. We defer the proof to the appendix.

Theorem 1. *Given the regulation factor $\delta > 0$, and defining $\Omega = \Omega_{\mathcal{R}_{t+1}}^{-1}$, assume decision variables $\tau_1, \tau_2 \geq 0$. Then the following convex optimization,*

$$\begin{cases} \min_{M_\phi, V_{a_t}, U_{a_t}, \tau_1, \tau_2} & -\mathbf{Logdet}(\Omega) \\ \text{s.t.} & -\tau_1 M_{s_t} - E_2^\top \begin{bmatrix} -U_{a_t} & V_{a_t} \\ V_{a_t}^\top & \tau_2 \end{bmatrix} E_2 - M_\phi + M_{out} \geq 0, \\ & U_{a_t} \ell \leq V_{a_t} \leq U_{a_t} \mathbf{u}, \quad \text{tr}(U_{a_t}) \delta \leq \tau_2. \end{cases} \quad (11)$$

results in values (V_{a_t}, U_{a_t}) such that the modified deterministic decision \hat{a}_t^c can be approximated with $\hat{a}_t^c = \hat{\mu}_{a_t} = U_{a_t}^{-1} V_{a_t}$.

Regarding the possible robustness issues with models (adversarial examples) we need to make a comparison between π^* and \hat{a}_t^c as a precautionary measure and select for the best choice for the modified action \hat{a}_t , via,

$$\hat{a}_t = \arg \min_{a \in \{\pi^*, \hat{a}_t^c\}} \|\tau_{opt}(t+1) - \mu_{NN}(s_t, a; \theta_\mu)\|_2. \quad (12)$$

Algorithm 1: Compensation Process for Distributional shifts

Input: s_1 and trained autonomous agent, π^*
Result: Small reachset for residual with its center closer to origin.

- 1 • Sample the optimal trajectory τ_{opt} .
- 2 **foreach** *time step* t **do**
 - 3 • $\left\{ \begin{array}{l} \text{if}(t = 1) \quad [\hat{a}_1 \leftarrow \pi^*] \\ \text{else}(t \geq 2) \left\{ \begin{array}{l} 1\text{- Apply Theorem 1 and compute } \hat{a}_t^c \\ 2\text{- Select the best action between } \hat{a}_t^c \text{ and } \pi^*, \text{ with (12) and return } \hat{a}_t. \end{array} \right. \end{array} \right.$
 - 4 • Employ the observation s_t and \hat{a}_t to characterize the confidence region S_{t+1} Using the deep surrogate (see Appendix D) for the deployment environment.
 - 5 • Record the observation s_{t+1} generated by exertion of \hat{a}_t to the environment
- 6 **end**

See Appendix D for more detail. We summarize the main steps of our proposed method in algorithm 1.

3.1 Scalability

The conservatism of tight ellipsoidal bound approximation introduced in [7] increases with the complexity of neural network’s structure and results in inaccurate solution for Theorem 1. However, for a highly nonlinear deployment environment, this is necessary to train a deep neural network for the surrogate. In response to this problem, (similar to [16]), we utilize an embedder network, \mathcal{M}_p , which maps the state s_t to another space $s'_t \in \mathbb{R}^{n'}$ ($s'_t = \mathcal{M}_p(s_t; \theta_p)$), such that s'_t is more tractable than s_t for training purposes. We next define the surrogate model and its parameters θ_μ based on s'_t as,

$$\mu_{s_{t+1}} = \mu_{NN}(s'_t, \hat{a}_t; \theta_\mu).$$

Given this setting for a highly nonlinear deployment environment, the neural network μ_{NN} is not necessarily a deep neural network. Thus, given the pair (s_t, a_t) as the input vector and s_{t+1} as output, we arrange a training procedure for the function,

$$\mu_{s_{t+1}} = \mu_{NN}(\mathcal{M}_p(s_t; \theta_p), a_t; \theta_\mu)$$

to learn the parameters θ_μ, θ_p together and utilize θ_μ in the convex programming. In another word, given the distribution of s_t and the parameters θ_p , we can approximate the distribution for s'_t with Gaussian mixture model techniques [20] to introduce its confidence region to the convex optimization (through μ_{NN}) for policy modification.

4 Experimental Results

Comparison with PSO. We assume simple car environment and compare the performance of our convex programming technique with Particle Swarm Optimization, (PSO) [13]⁵, on solving the optimization (4). PSO has shown acceptable performance in low dimensional environments. Thus, we plan to show our convex programming technique can outperform PSO even if the scalability is not an important issue. The environment represents the following simple car model:

$$\begin{aligned} \dot{x} &= u\cos(\theta), & \dot{y} &= u\sin(\theta), & \dot{\sin}(\theta) &= \frac{u}{\ell}\tan(\phi)\cos(\theta), \\ \dot{\cos}(\theta) &= -\frac{u}{\ell}\tan(\phi)\sin(\theta) \end{aligned} \tag{13}$$

The system represents a car of length ℓ moving with constant velocity u and driven with control action ϕ . The training environment is characterized by $\ell = 2.5$ and, $u = 4.9$ while the deployment environment is slightly different with $\ell = 2.1$ and $u = 5.1$. We collect a training data set from deployment environment with time step 0.01 second.

We train two surrogates for the deployment environment μ_{NN}, μ_{NN}^* from deployment environment. The former is utilized in optimization (11) and is a ReLU neural network with dimension [5, 8, 4]. This ReLU neural network is obtained from the proposed procedure in section 3.1. The latter is utilized for comparison discussed in equation (12) and Appendix D, which is a deep **tanh**() neural network of dimension [5, 200, 200, 200, 200, 200, 200, 200, 4]. The results of policy modification are presented in Fig.1. This figure presents the optimal trajectory, τ_{opt} , in green color. This trajectory is simulated with an optimal control and high-fidelity surrogate for the training environment computed from a model-based algorithm. The blue curves are the results of algorithm 1 for (500 steps), which closely tracks the optimal trajectory in all the three states x, y, θ . The red curves represent the deployment environment’s trajectory when there is no policy modification. The run time for convex programming is between [0.005, 0.027] on a personal laptop with YALMIP and MOSEK solver. Thus, we restrict the run time of PSO with 0.027 and employ it for policy modification. In one attempt, we utilize PSO for optimization (4) on the same model with convex programming, μ_{NN} where the resultant trajectory is demonstrated in black. In another attempt we utilize PSO over the deep model μ_{NN}^* and the resultant trajectory is demonstrated in magenta. The results clearly shows our convex programming outperforms the PSO in both cases.

⁵ While it is well-known that nonlinear optimization techniques lack guarantees and can suffer from local minima, techniques like particle swarm work well in practice, especially in low-dimensional systems. Hence, we perform this comparison to show that convexification outperforms state-of-the-art global optimization approaches to residual minimization.

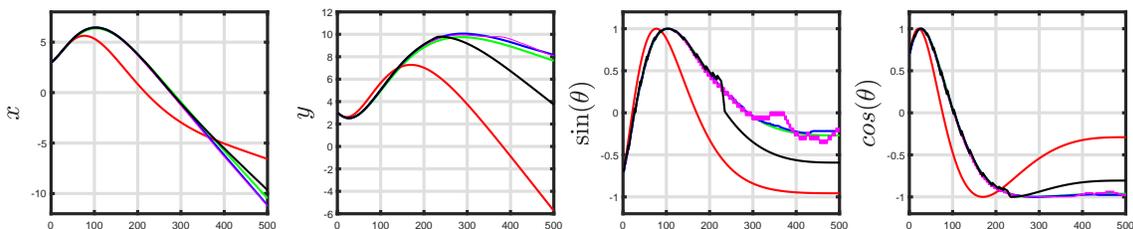


Fig. 1: Shows the comparison between PSO and our convex programming. The green and blue curves are the results of algorithm 1 and optimal trajectory, respectively. The red curves represent the deployment environment’s trajectory when there is no policy modification. We utilize PSO for optimization (4) on the same model with convex programming, μ_{NN} where the resultant trajectory is demonstrated in black. We also utilize PSO over the deep model, μ_{NN}^* and the resultant trajectory is demonstrated in magenta.

Linear Environment of a Car. The training environment is a stochastic linear dynamics as follows:

$$\mathbf{x}_{t+1} = \begin{bmatrix} 1 & 0.1 & 0.0047 \\ 0 & 1 & 0.0906 \\ 0 & 0 & 0.8187 \end{bmatrix} \mathbf{x}_t + \begin{bmatrix} 0.003 \\ 0.0094 \\ 0.1813 \end{bmatrix} u_t + \nu_t,$$

$$\nu_t \sim \mathcal{N} \left(\begin{bmatrix} 0 & 0 & 0.2 \end{bmatrix}^\top, \exp(-8)I_3 \right)$$

The deployment environment is also a stochastic linear dynamics as follows:

$$\mathbf{x}_{t+1} = \begin{bmatrix} 1 & 0.1 & 0.0046 \\ 0 & 1 & 0.0885 \\ 0 & 0 & 0.7788 \end{bmatrix} \mathbf{x}_t + \begin{bmatrix} 0.004 \\ 0.0115 \\ 0.2212 \end{bmatrix} u_t + \eta_t,$$

$$\eta_t \sim \mathcal{N}(\mathbf{0}, \exp(-8)I_3)$$

where the sampling time is $t_s = 0.1$ s. The state $\mathbf{x}_t \in \mathbb{R}^3$ is defined as $\mathbf{x}_t = [x_t, v_t, a_t]^\top$ that are position, velocity and acceleration of car respectively. The scalar action u_t is also bounded within $u_t \in [-3, 3]$. Since the environment is linear, it is not required to use the embedder network. Thus, we train only one surrogate for the deployment environment μ_{NN} with a 2 hidden layer ReLU neural network of dimension $[4, 10, 5, 3]$. We also have access to the model of training environment and a trained optimal feedback policy. Therefore, we perform policy modification through algorithm 1 for deployment environment and the results are presented in Fig.2. In this figure, the green curve presents the simulated optimal trajectory. Blue and red curves also represent the trajectory of deployment environment in the presence and absence of policy modification, respectively. This figure shows the algorithm 1 forces the deployment environment to track the planner τ_{opt} and the policy modification process is successful.

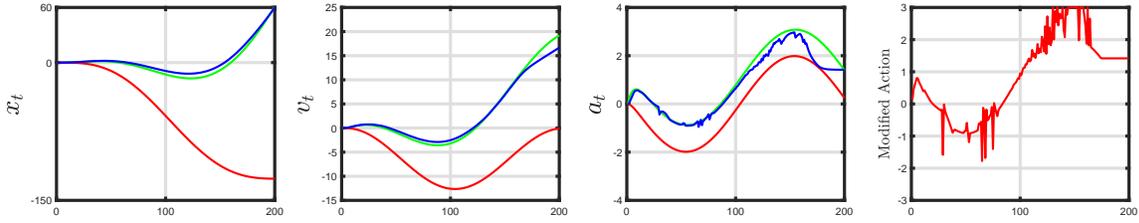


Fig. 2: Shows the results of policy modification on stochastic linear environment of a car. In this figure, the green curve presents the simulated optimal trajectory. Blue and red curves also represent the trajectory of deployment environment in the presence and absence of policy modification, respectively.

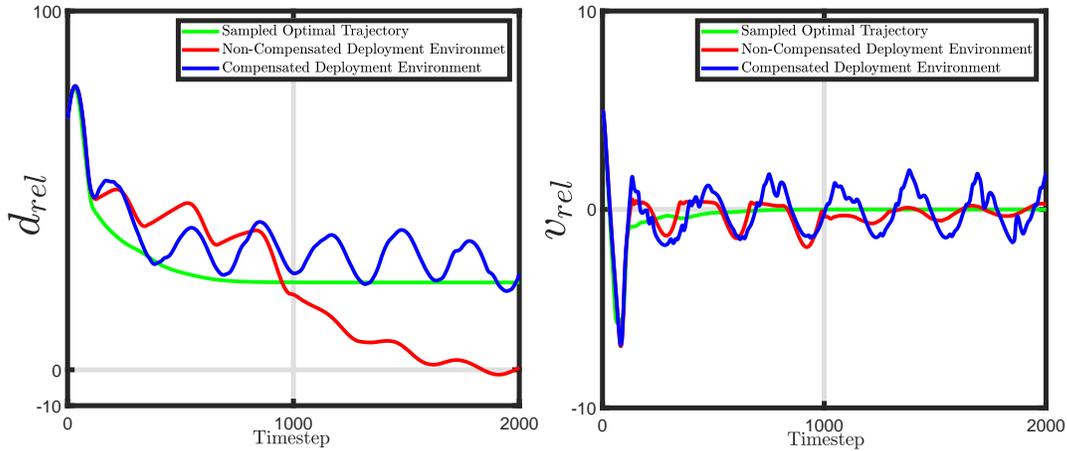


Fig. 3: The green curves represent the optimal trajectory for v_{rel} and d_{rel} , while the red and blue curves present the trajectory of deployment environment without policy adaptation and with adaptation, respectively.

Adaptive Cruise Control. Consider the Simulink environment for adaptive cruise control in MATLAB documentation⁶. We consider this trained feedback controller and assume we have access to the model of training environment. We then simulate the optimal trajectory τ_{opt} with model and controller. This controller is trained over 14 hours, which clearly shows how learning a new controller can be expensive and justifies the contribution of our technique. The input of the trained controller is the vector $\mathbf{x} = [\int v_{err}, v_{err}, v_{ego}]^T$. Thus, we take this vector as the state of the environment⁷. This implies lead-car, ego-car and signal

⁶ <https://www.mathworks.com/help/reinforcement-learning/ug/train-ddpg-agent-for-adaptive-cruise-control.html>

⁷ Here v_{err} is a logic based function of $x_{ego}, x_{lead}, v_{ego}$ and v_{lead} . See the MATLAB documentation for more detail. Here, (v_{ego}, v_{lead}) and (x_{ego}, x_{lead}) are the velocity and position for ego and lead car, respectively.

processing block are all together the environment. Consider, this environment is highly nonlinear due to the presence of logic based relations in the signal processing block. The implemented value of V_{set} on the training environment is set on 30 m/s while it is mistakenly set on 34.5 m/s in the deployment environment. This difference characterizes the distributional shift.

Fig.3 shows the evolution of relative velocity and relative position, v_{rel}, d_{rel} between lead and ego cars. The green line shows the simulation for v_{rel}, d_{rel} when the optimal policy, π^* is applied on the training environment. On the other hand, blue and red lines show the evolution of v_{rel}, d_{rel} in the presence and absence of policy modification, respectively. Policy modification process aims to force the states of deployment environment to track optimal trajectory τ_{opt} . Consider the parameter $d_{rel} < 0$ on red line at time $t = 183\text{s}$. Thus, the distributional shift leads to accident in the absence of policy modification. Fig.3 shows, our policy modification technique secures the system against the possible accident.

5 Conclusion and Related Work

Related work. In [5], the authors separate the learned policy from the raw inputs and outputs to ease the transfer from simulation. In [3] a method to use a learned deep inverse dynamics model to decide which real-world action is most suitable to achieve the same state as the simulator is proposed. Mutual alignment transfer learning approaches employ auxiliary rewards for transfer learning under discrepancies in system dynamics for simulation to robot transfer ([21]). Approaches such as [6] compensate for the difference in dynamics by modifying the reward function such that the modified reward function penalizes the agent for visiting states and taking actions in the source domain which are not possible in the target domain. In [19], the authors study robust adversarial reinforcement learning. Inspired with the $\|H\|_\infty$ control idea, they assume the destabilizing adversaries like the gap between simulation and environment as uncertainties and devise a learning algorithm which considers the worst case adversary and is robust against it. Transfer learning has also been investigated in the multi-agent setting [15], where the problem of training agents with continuous actions is studied to ensure that the trained agents can still generalize when their opponent’s policies alter. [18] proposed to use Bayesian optimization (BO) to actively select the distribution of the environment variable that maximizes the improvement generated by each iteration of the policy gradient method. Unlike the authors of [6] who propose a reward modification technique, in this work we propose a policy modification technique to tackle the problem when the environment model in training is different from what is expected.

Conclusion & Broader Impact. In this work, we presented a linearization algorithm for a non-linear system trained by deep neural networks equipped with ReLU activation function and proposed a convex optimization based framework to do the distributional shift compensation on an unknown model with unknown distributional shift. The benefit is the convenience of computation and ability of the controller to respond to the distributional shift instantaneously, with a small cost of added conservatism due to the ellipsoidal bound based linearization.

Future work. Algorithm 1 is currently running the optimization step by step that is a greedy method. We plan to derive a single optimization that covers all the trajectory to improve the result.

References

1. Alpaydin, E.: Introduction to machine learning. MIT press (2020)
2. Bertsekas, D., Shreve, S.E.: Stochastic optimal control: the discrete-time case, vol. 5. Athena Scientific (1996)
3. Christiano, P., Shah, Z., Mordatch, I., Schneider, J., Blackwell, T., Tobin, J., Abbeel, P., Zaremba, W.: Transfer from simulation to real world through learning deep inverse dynamics model. arXiv preprint arXiv:1610.03518 (2016)
4. Chua, K., Calandra, R., McAllister, R., Levine, S.: Deep reinforcement learning in a handful of trials using probabilistic dynamics models. arXiv preprint arXiv:1805.12114 (2018)
5. Clavera, I., Held, D., Abbeel, P.: Policy transfer via modularity and reward guiding. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1537–1544. IEEE (2017)
6. Eysenbach, B., Asawa, S., Chaudhari, S., Salakhutdinov, R., Levine, S.: Off-dynamics reinforcement learning: Training for transfer with domain classifiers. arXiv preprint arXiv:2006.13916 (2020)
7. Fazlyab, M., Morari, M., Pappas, G.J.: Probabilistic verification and reachability analysis of neural networks via semidefinite programming. arXiv preprint arXiv:1910.04249 (2019)
8. Fazlyab, M., Morari, M., Pappas, G.J.: Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. CoRR abs:1903.01287 (2019)
9. Fazlyab, M., Morari, M., Pappas, G.J.: Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. IEEE Transactions on Automatic Control (2020)
10. Garcia, C.E., Prett, D.M., Morari, M.: Model predictive control: Theory and practice—a survey. Automatica **25**(3), 335–348 (1989)
11. Hashemi, N., Fazlyab, M., Ruths, J.: Performance bounds for neural network estimators: Applications in fault detection. In: 2021 American Control Conference (ACC). pp. 3260–3266 (2021). <https://doi.org/10.23919/ACC50511.2021.9482752>
12. Jain, A., Smarra, F., Mangharam, R.: Data predictive control using regression trees and ensemble learning. In: 2017 IEEE 56th annual conference on decision and control (CDC). pp. 4446–4451. IEEE (2017)
13. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN’95-international conference on neural networks. vol. 4, pp. 1942–1948. IEEE (1995)
14. Khalil, I., Doyle, J., Glover, K.: Robust and optimal control. Prentice hall (1996)
15. Li, S., Wu, Y., Cui, X., Dong, H., Fang, F., Russell, S.: Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 4213–4220 (2019)
16. Lusch, B., Kutz, J.N., Brunton, S.L.: Deep learning for universal linear embeddings of nonlinear dynamics. Nature communications **9**(1), 1–10 (2018)
17. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. nature **518**(7540), 529–533 (2015)

18. Paul, S., Osborne, M.A., Whiteson, S.: Fingerprint policy optimisation for robust reinforcement learning. In: International Conference on Machine Learning. pp. 5082–5091 (2019)
19. Pinto, L., Davidson, J., Sukthankar, R., Gupta, A.: Robust adversarial reinforcement learning. In: International Conference on Machine Learning. pp. 2817–2826 (2017)
20. Reynolds, D.A., et al.: Gaussian mixture models. Encyclopedia of biometrics **741**(659-663) (2009)
21. Wulfmeier, M., Posner, I., Abbeel, P.: Mutual alignment transfer learning. In: Conference on Robot Learning. pp. 281–290 (2017)

6 Appendix

Appendix A: Generic computational Error

We present a brief summary of [7] and [11] in Appendix E and here we directly focus on the necessary steps for the proof. We denote $\mathcal{M}_t : (S \times A) \rightarrow \mathcal{R}_{t+1}$ as the neural network where represents the residual. This network is equivalent with μ_{NN} but the last bias vector is shifted with $\tau_{opt}(t+1)$. Assume $\mathcal{R}_{t+1}, \mathcal{R}_{t+1}^c$ are the residual reachsets when $(S_t \times A_t)$ and $(S_t \times A_t^c)$ are introduced in, \mathcal{M}_t respectively ($A_t^c \subset A_t$). This implies, if $a_t \in A_t^c$, then $a_t \in A_t$ and therefore, for $A_t = \mathcal{E}(\mu_{a_t}, \Omega_{a_t})$,

$$\begin{bmatrix} a_t \\ 1 \end{bmatrix}^\top \begin{bmatrix} -\Omega_{a_t}^{-1} & \Omega_{a_t}^{-1} \mu_{a_t} \\ \mu_{a_t}^\top \Omega_{a_t}^{-1} & -\mu_{a_t}^\top \Omega_{a_t}^{-1} \mu_{a_t} + 1 \end{bmatrix} \begin{bmatrix} a_t \\ 1 \end{bmatrix} \geq 0,$$

which suffices to say, regarding the optimization (22), the optimal upper-bound for the \mathcal{R}_{t+1} produced by $(S_t \times A_t)$ is in fact a feasible solution for the upper-bound of \mathcal{R}_{t+1}^c obtained from $(S_t \times A_t^c)$ (see the summary of [7,11] in Appendix E). This implies the optimal objective function of optimization (22) in the second problem (taking $(S_t \times A_t^c)$ as input) is less than the optimal objective function of optimization(22) in the first problem (taking $(S_t \times A_t)$ as input). Therefore, we conclude $\mathbf{Logdet}(\Omega_{\mathcal{R}_{t+1}^c}) < \mathbf{Logdet}(\Omega_{\mathcal{R}_{t+1}})$. In another word, we conclude, replacing A_t with A_t^c results in smaller upper-bound on the residual reachset.

Now assume the optimal confidence region for modified action, A_t in optimization (7) contains nonempty subset, $A_t^c \subset A_t$. However, we know the ellipsoidal bound of residual reachset is still reducible by replacing A_t with A_t^c . This is a contradiction because we have already concluded A_t results in smallest upper-bound for the residual reachset. Thus, the optimal confidence region A_t contains no subset and is a singleton. In another word $\text{tr}(\Omega_{a_t}) = 0$.

Appendix B: Construction of Actuator Bound

The proposed action should be inside the following hyper-rectangle: $\ell \leq a_t \leq \mathbf{u}$. In Appendix A we proved that the solution of optimization (11) is a singleton $A_t = \{a_t\}$, therefore we neglect the shape matrix, $\tau_2 U_{a_t}^{-1}$ and only bound the mean value in the mentioned hyper-rectangle, $\ell \leq U_{a_t}^{-1} V_{a_t} \leq \mathbf{u}$, which implies, $U_{a_t} \ell \leq V_{a_t} \leq U_{a_t} \mathbf{u}$

Appendix C. Proof of Theorem 1:

Based on [7] we know the sufficient condition for an ellipsoid $\mathcal{E}(0, \Omega)$ to bound the reachset of the residual is,

$$\tau_1 M_{s_t} + E_2^\top \begin{bmatrix} -U_{a_t} & V_{a_t} \\ V_{a_t}^\top & -V_{a_t}^\top U_{a_t}^{-1} V_{a_t} + \tau_2 \end{bmatrix} E_2 + M_\phi - M_{out} \leq 0 \quad (14)$$

we move the linear terms to the right and keep the nonlinear term at the left,

$$E_2^\top \begin{bmatrix} 0 & 0 \\ 0 & -V_{a_t}^\top U_{a_t}^{-1} V_{a_t} \end{bmatrix} E_2 \leq -\tau_1 M_{s_t} - E_2^\top \begin{bmatrix} -U_{a_t} & V_{a_t} \\ V_{a_t}^\top & \tau_2 \end{bmatrix} E_2 - M_\phi + M_{out} \quad (15)$$

The matrix in the left of inequality is negative definite, therefore if we introduce the new constraint,

$$-\tau_1 M_{s_t} - E_2^\top \begin{bmatrix} -U_{a_t} & V_{a_t} \\ V_{a_t}^\top & \tau_2 \end{bmatrix} E_2 - M_\phi + M_{out} \geq 0 \quad (16)$$

we have satisfied the required constraint in (15). Based on our observations, this new linear constraint will not impose conservatism because the value $V_{a_t}^\top U_{a_t}^{-1} V_{a_t}$ is always near to zero, thus we are neglecting the negative value of the nonlinear term. As we discussed before, we know $\Omega_{a_t} = \tau_2 U_{a_t}^{-1}$ converges to zero, this implies there is a chance for U_{a_t} to become unbounded and this is why an infinitesimal Ω_{a_t} is problematic for our convex optimization. In order to avoid unbounded solution for U_{a_t} , we provide a small lower bound on the $\text{tr}(\Omega_{a_t})$. We know Ω_{a_t} is a positive definite matrix, therefore if $\text{tr}(\Omega_{a_t}^{-1})$ is smaller than a big number, σ , that suffices to have $\text{tr}(\Omega_{a_t})$ to be greater than a small number (lower bound on size of A_t). This can be rephrased with the convex constraint $\text{tr}(U_{a_t}) \leq \tau_2 \sigma$, or in another word, $\text{tr}(U_{a_t}) \delta \leq \tau_2$ where $\delta = \frac{1}{\sigma}$ is preferably a small number. Thus, to justify the presence of convex constraint $\text{tr}(U_{a_t}) \delta \leq \tau_2$, we mention this is just a precautionary measure (δ is very small) to avoid unbounded solutions.

Appendix D. Comparison

Due to presence of adversarial examples, we need to certify the modified action performs better than autonomous agent on deployment environment. Since we can not utilize the environment directly for this purpose, we must employ a deep surrogate model for deployment environment. On the other hand, reading through [9] clarifies, although a deep network is accurate, it results in noticeable conservatism for convex programming in Theorem 1. Therefore, we train two networks for surrogate in a highly nonlinear environment. The former will be obtained from Embedded technique in section 3.1 and will be utilized for convex programming. The latter is a very deep network that provides reliability for an accurate comparison between, π^* and d_t^c . We call this deep neural network as, μ_{NN}^* .

Appendix E. Brief summary of [7]

We present the solution summary with parameters of our specific problem for more clarification. Assume the confidence regions for state and actions $s_t \in S_t$, $a_t \in A_t$ are fixed and known. Then the tool provided in [7,11] proposes a convex optimization for tightest ellipsoidal upper-bound over residual's reachset \mathcal{R}_{t+1} . In this research, we add another constraint and fix the center of the mentioned upper-bound on the origin to make it certain that the residual decreases in Euclidean norm. Therefore, the tool [7,11] is utilized to present the tightest upper-bound such that, $\mathcal{R}_{t+1} \subset \mathcal{E}(\mathbf{0}, \Omega_{\mathcal{R}_{t+1}}^{-1})$. We know $a_t \in A_t := \mathcal{E}(\mu_{a_t}, \Omega_{a_t})$ therefore:

$$\begin{bmatrix} a_t \\ 1 \end{bmatrix}^\top \underbrace{\begin{bmatrix} -\Omega_{a_t}^{-1} & \Omega_{a_t}^{-1} \mu_{a_t} \\ \mu_{a_t}^\top \Omega_{a_t}^{-1} & -\mu_{a_t}^\top \Omega_{a_t}^{-1} \mu_{a_t} + 1 \end{bmatrix}}_{Q_1} \begin{bmatrix} a_t \\ 1 \end{bmatrix} \geq 0 \quad (17)$$

We also know $s_t \in S_t := \mathcal{E}(\mu_{s_t}, \rho_n \Sigma_{s_t})$ therefore:

$$\frac{1}{\rho_n} \begin{bmatrix} s_t \\ 1 \end{bmatrix}^\top \underbrace{\begin{bmatrix} -\Sigma_{s_t}^{-1} & \Sigma_{s_t}^{-1} \mu_{s_t} \\ \mu_{s_t}^\top \Sigma_{s_t}^{-1} & -\mu_{s_t}^\top \Sigma_{s_t}^{-1} \mu_{s_t} + \rho_n \end{bmatrix}}_{Q_2} \begin{bmatrix} s_t \\ 1 \end{bmatrix} \geq 0, \quad (18)$$

In the next attempt [7] suggests us to concatenate all the post-activations, in the residual's model \mathcal{M}_t and generate vector $\mathbf{x} = [z^1{}^\top, z^2{}^\top, \dots, z^{\mathcal{L}-1}{}^\top]^\top$. Then they propose a symmetric matrix Q_ϕ which satisfies the quadratic constraint,

$$\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}^\top Q_\phi \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \geq 0. \quad (19)$$

The ultimate goal is to prove the residual $\Delta_{t+1} \in \mathcal{E}(\mathbf{0}, \Omega_{\mathcal{R}_{t+1}})$. Therefore, defining $\Omega = \Omega_{\mathcal{R}_{t+1}}^{-1}$ we should propose a constraint that implies,

$$\begin{bmatrix} r_{t+1} \\ 1 \end{bmatrix}^\top \begin{bmatrix} -\Omega & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} r_{t+1} \\ 1 \end{bmatrix} \geq 0 \quad (20)$$

To propose such a constraint authors in [7] suggest defining the base vector $\mathbf{z} = [s_t^\top, a_t^\top, \mathbf{x}^\top, 1]^\top$ and define the linear transformation matrices E_1, E_2, E_3 and matrix C as,

$$\begin{bmatrix} s_t \\ 1 \end{bmatrix} = E_1 \mathbf{z}, \quad \begin{bmatrix} a_t \\ 1 \end{bmatrix} = E_2 \mathbf{z}, \quad \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = E_3 \mathbf{z}, \quad \begin{bmatrix} r_{t+1} \\ 1 \end{bmatrix} = \begin{bmatrix} C & b \\ 0 & 1 \end{bmatrix} \mathbf{z},$$

and $C = [0 \ 0 \ \dots \ W_\ell]$, $b = b_\ell - \tau_{opt}(t+1)$, $(b_\ell, W_\ell) \in \theta_\mu$ represent the bias vector and the weights of the last layer in μ_{NN} , and add the left side of equations (17), (18), (19) which provides the following inequality:

$$\mathbf{z}^\top \left(\tau_1 \underline{E_1^\top Q_1 E_1}_{M_{s_t}} + \tau_2 \underline{E_2^\top Q_2 E_2}_{M_{a_t}} + \underline{E_3^\top Q_\phi E_3}_{M_\phi} \right) \mathbf{z} \geq 0, \quad (21)$$

for some $\tau_1, \tau_2 \geq 0$. Thus, if the inequality,

$$\mathbf{z}^\top (\tau_1 M_{s_t} + \tau_2 M_{a_t} + M_\phi) \mathbf{z} - \begin{bmatrix} r_{t+1} \\ 1 \end{bmatrix}^\top \begin{bmatrix} -\Omega & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} r_{t+1} \\ 1 \end{bmatrix} \leq 0$$

holds, then the constraint (20) is satisfied. This constraint can be reformulated as,

$$\mathbf{z}^\top \left(\tau_1 M_{s_t} + \tau_2 M_{a_t} + M_\phi - \underbrace{\begin{bmatrix} C & b_{\mathcal{L}} \\ 0 & 1 \end{bmatrix}^\top \begin{bmatrix} -\Omega & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} C & b_{\mathcal{L}} \\ 0 & 1 \end{bmatrix}}_{M_{out}} \right) \mathbf{z} \leq 0$$

thus applying the assumption, $\tau_1 M_{s_t} + \tau_2 M_{a_t} + M_\phi - M_{out} \leq 0$, is sufficient but not necessary to claim (20) is satisfied. Therefore, the convex optimization:

$$\begin{cases} \min_{M_\phi, \tau_1, \tau_2} & -\mathbf{Logdet}(\Omega) \\ \text{s.t.} & \tau_1 M_{s_t} + \tau_2 M_{a_t} + M_\phi - M_{out} \leq 0 \end{cases} \quad (22)$$

presents the suboptimal tightest ellipsoidal upper-bound that is centered on the origin over the residual reachset \mathcal{R}_{t+1} .