ADAPT-TO-LEARN POLICY TRANSFER IN REINFORCEMENT LEARNING AND DEEP MODEL REFERENCE ADAPTIVE CONTROL

BY

GIRISH JOSHI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Aerospace Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Doctoral Committee:

Professor Girish Chowdhary, Advisor
Professor Cedric Langbort, Chair
Professor Naira Hovakimyan
Professor Rayadurgam Srikant
Professor Matthew West

ABSTRACT

Adaptation and Learning from exploration have been a key in biological learning; Humans and animals do not learn every task in isolation; rather are able to quickly adapt the learned behaviors between similar tasks and learn new skills when presented with new situations. Inspired by this, adaptation has been an important direction of research in control as Adaptive Controllers. However, the Adaptive Controllers like Model Reference Adaptive Controller are mainly model-based controllers and do not rely on exploration instead make informed decisions exploiting the model's structure. Therefore such controllers are characterized by high sample efficiency and stability conditions and, therefore, suitable for safety-critical systems. On the other hand, we have Learning-based optimal control algorithms like Reinforcement Learning. Reinforcement learning is a trial and error method, where an agent explores the environment by taking random action and maximizing the likelihood of those particular actions that result in a higher return. However, these exploration techniques are expected to fail many times before exploring optimal policy. Therefore, they are highly sample-expensive and lack stability guarantees and hence not suitable for safety-critical systems.

This thesis presents control algorithms for robotics where the best of both worlds that is "Adaptation" and "Learning from exploration" are brought together to propose new algorithms that can perform better than their conventional counterparts.

In this effort, we first present an Adapt to learn policy transfer Algorithm, where we use control theoretical ideas of adaptation to transfer policy between two related but different tasks using the policy gradient method of reinforcement learning. Efficient and robust policy transfer remains a key challenge in reinforcement learning. Policy transfer through warm initialization, imitation, or interacting over a large set of agents with randomized instances, have been commonly applied to solve a variety of Reinforcement

Learning (RL) tasks. However, this is far from how behavior transfer happens in the biological world: Here, we seek to answer the question: Will learning to combine adaptation reward with environmental reward lead to a more efficient transfer of policies between domains? We introduce a principled mechanism that can **"Adapt-to-Learn"**, which is adapt the source policy to learn to solve a target task with significant transition differences and uncertainties. Through theory and experiments, we show that our method leads to a significantly reduced sample complexity of transferring the policies between the tasks.

In the second part of this thesis, information-enabled learning-based adaptive controllers like **"Gaussian Process adaptive controller using Model Reference Generative Network"** (GP-MRGeN), **"Deep Model Reference Adaptive Controller"** (DMRAC) are presented. Model reference adaptive control (MRAC) is a widely studied adaptive control methodology that aims to ensure that a nonlinear plant with significant model uncertainty behaves like a chosen reference model. MRAC methods try to adapt the system to changes by representing the system uncertainties as weighted combinations of known nonlinear functions and using weight update law that ensures that network weights are moved in the direction of minimizing the instantaneous tracking error. However, most MRAC adaptive controllers use a shallow network and only the instantaneous data for adaptation, restricting their representation capability and limiting their performance under fast-changing uncertainties and faults in the system.

In this thesis, we propose a Gaussian process based adaptive controller called GP-MRGeN. We present a new approach to the online supervised training of GP models using a new architecture termed as Model Reference Generative Network (MRGeN). Our architecture is very loosely inspired by the recent success of generative neural network models. Nevertheless, our contributions ensure that the inclusion of such a model in closed-loop control does not affect the stability properties. The GP-MRGeN controller, through using a generative network, is capable of achieving higher adaptation rates without losing robustness properties of the controller, hence suitable for mitigating faults in fast-evolving systems.

Further, in this thesis, we present a new neuroadaptive architecture: Deep Neural Network-based Model Reference Adaptive Control. This architecture utilizes deep neural network representations for modeling significant nonlin-

earities while marrying it with the boundedness guarantees that characterize MRAC based controllers. We demonstrate through simulations and analysis that DMRAC can subsume previously studied learning-based MRAC methods, such as concurrent learning and GP-MRAC. This makes DMRAC a highly powerful architecture for high-performance control of nonlinear systems with long-term learning properties. Theoretical proofs of the controller generalizing capability over unseen data points and boundedness properties of the tracking error are also presented. Experiments with the quadrotor vehicle demonstrate the controller performance in achieving reference model tracking in the presence of significant matched uncertainties. A software+communication architecture is designed to ensure online real-time inference of the deep network on a high-bandwidth computation-limited platform to achieve these results. These results demonstrate the efficacy of deep networks for high bandwidth closed-loop attitude control of unstable and nonlinear robots operating in adverse situations. We expect that this work will benefit other closed-loop deep-learning control architectures for robotics.

To my wife "Sushma", for her love and support.

ACKNOWLEDGMENTS

It is my pleasure to take this opportunity to thank some of the people who directly indirectly helped and supported me through this process. I owe my deepest gratitude to my advisor, mentor Dr. Girish Chowdhary, for his unfailing support and guidance through my time at the University of Illinois Urbana-Champaign. His leadership skills and ability to find an innovative approach to problem-solving and the ability to look at the big picture and motivate necessary fundamental research will always inspire me. I do not have a dramatic story to share of a tumultuous ride of a Ph.D. student, but rather a happy one. Thanks to my advisor, who has always advised me well and supported financially throughout. Dr. Chowdhary's very open and friendly approach to research discussions, sometime in a very casual setting like at a relaxing game of badminton always helped me innovate and think out the box. While earning my degree, I have made a good friend for life, my advisor!

I should also thank a few people here who had a profound influence on me and my work through personal discussions or the knowledge imparted in their class. I want to thank Dr. Martin Hagan for his incredible lectures in Neural Network and Estimation Theory. He is one of the best teachers I ever had the privilege of listening to. I want to thank Dr. R Srikant, for teaching us to enjoy the elegance of the theory of Reinforcement Learning and Markov Decision Processes. For the love of being in their class, I took the Machine Learning course both under Dr. Bruce Hajek and Dr. Maxim Raginksky. My life research goal will be to, at the least, inculcate a fraction of your passions towards research. I thank Dr. Naira Hovakimyan for teaching me the mathematical intricacies of Adaptive control; it was a dream come true experience attending your course. I would like to thank my committee chair, Dr. Cedric Langbort, and the committee members, Dr. Srikant Raydurgam, Dr. Naira Hovakimyan, and Dr. Mathew West, your reviews and suggestions

on my dissertation helped me refine my work.

My time here at the University of Illinois Urbana-Champaign and Oklahoma State University has been made pleasurable by all my friends and colleagues. I am grateful to my current and past lab-mates and friends including Denis Ospichev, Allan Axelrod, Aaron Havens, Anwesa Choudhuri, Anay Patnaik, Jasvir Virdi, Harshal Maske, Sri Theja Vuppala, Nolan Reploge, Joshua Whitman, Moitreya Chaterjee, M Ugur Ackal, Hunter Young, Aseem Borkar, Prabhat Mishra, Maximillian Muhlegg. You all have been like a family away from home.

I wish to thank all members of administration staff at the Department of Aerospace Engineering, Coordinated Science Lab, College of Engineering, and ISSS who work behind the curtains for the success of a graduate student. A special thanks to Linda Stimson of CSL and Staci L McDannel of Aerospace Engineering.

This journey and achievement would not have been possible without support and encouragement from my loving wife "Sushma". I could not have mustered enough courage leaving a stable job in the Indian Space Research Organization and the family behind, and travelling to unknown land perusing Ph.D. Its Sushma's constant support and greater sacrifice that I could follow my heart. My daughter Aarohi's infectious smile helped me persist through lonely times. Lastly, I wish to thank my parents, in-laws; Joshi's and Koulgi's and my sisters Supriya[2], you all have been immense source of comfort, without which I would be lost.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

Policy Transfer in Reinforcement Learning

$\mathcal{M}$	Markov decision process
$\mathcal{S}$	State Space
$\mathcal{A}$	Action Space
$\bar{\mathcal{A}}$	Normalized Action Space
$\mathcal{P}$	Transition model
$\mathcal{H}$	Horizon length
$\mathcal{R}$	Reward Model
$s, s_k \in \mathcal{S}$	Agent state at step at time k
$s', s_{k+1} \in \mathcal{S}$	Transitioned next state
$a_k \in \mathcal{A}$	Sampled action
$p(s_{k+1}\|s_k, a_k) \in \mathcal{P}$	Transition model
$r_k(s_k, a_k) \in \mathcal{R}$	Reward for state s_k and action a_k
ρ_0	Distribution on Initial states
γ	Discount factor
π	Agent Policy
π_θ	Parameterized policy
π^*	Optimal policy
π_{ad}	Adaptive policy
$\mathcal{K}$	simulation time step
$\mathcal{T}$	Bellman Update Operator

V^{π}	State value function under policy π
Q^{π}	State-Action value function under policy π
A^{π}	Advantage function under policy π
b_k	Baseline correction term
V^*	Optimal State value function
Q^*	Optimal State-Action value function
δ	Temporal Difference error (TD-error)
$\mathcal{M}^S$	Source MDP
$\mathcal{M}^T$	Target MDP
$\hat{\mathcal{P}}^T$	Approximate/Apprentice target Model
$\hat{W}, \hat{\mathcal{B}}$	Approximate/Apprentice target Model Network parameters
χ_T^S	Manifold Alignment
$\hat{s}_k^S, \hat{s}_k^T \in \mathcal{S}$	Projected states in source and target domain
$\mathcal{D}$	Replay Buffer
θ	Policy network parameters
θ^-	Policy network parameters before update
α	Learning rate
$\eta_{\pi}(\theta), J(\theta)$	Total return under policy π
$\eta_{KL,\beta}(\pi_\theta, \pi^*)$	Intrinsic objective
$\bar{\eta}_{KL,\beta}(\pi_\theta, \pi^*)$	Lower bound on Intrinsic objective
β	Reward Mixing co-efficient
ζ_t	Intrinsic reward
r'_t	Total reward
τ^{π_θ}	Trajectory under policy π_θ
d^{π_θ}	State visitation distribution under policy π_θ
∇_θ	Gradient operator
$\zeta\tau$	Placeholder for total return

Model Reference Adaptive control

A	System Dynamics
B	Control Matrix
A_{rm}	Reference system Dynamics
B_{rm}	Reference system control matrix
t	time
$x(t)$	System states
$x_{rm}(t)$	Reference system states
$e(t)$	Tracking error
$\dot{()}$	Derivative operator
$u(t)$	Admissible control vector
$u_{pd}(t)$	Feedback controller
$u_{crm}(t)$	Feed-forward controller
$u_{ad}(t)$	Adaptive controller
k_x, k_r	Feedback and Feed-forward gain
$f(x)$	Nonlinear uncertainty term
L	Lipschitz constant
$\mathcal{D}_x$	Compact domain of states
W^*	Ideal adaptive network parameters
W	Approximate adaptive network parameters
$\tilde{W}$	Parameter error
$\mathcal{W}_b$	Parameter upper bound
$\sigma(x)$	Structured uncertainty feature vector
$\Phi(x)$	Unstructured uncertainty feature vector
$\phi_i(x)$	Elements of unstructured uncertainty feature vector
c_i, σ_i	Latent hyper-parameter of RBF kernel

$\tilde{\epsilon}(x)$	Network appromation error
$\lambda_{min}, \lambda_{max}$	Min and Max Eigen Values
$V(t,x)$	Lyapunov function
Γ	Learning rate
Q, P	Symmetric positive definite matrix
$\mathcal{GP}$	Gaussian Process
$m(.), k(.,.)$	Mean and Co-Variance kernel vector
$\mathcal{H}$	Hilbert Space
G_σ	Linear operator associated with kernel $k(z, z')$
ζ	zero mean Weiner noise
σ	Switching signal
$\mathcal{BV}$	Feature vector buffer
$f_\theta(x)$	Deep neural Network estimate of the function $f(x)$
θ	D-MRAC network parameters
$\Phi_n^\sigma(x) = f_\theta \backslash \theta_n$	Deep feature vector
$Z^M = \{x_i, y_i\}_{i=1}^M$	D-MRAC training data set
$m_{\mathcal{A}}(\epsilon, \delta)$	Sample complexity
N	Number of D-MRAC network parameters

CHAPTER 1

INTRODUCTION

1.1 Adaptation and Learning

Reinforcement learning and Adaptive control are two main approaches in learning control for an agent solving a task. Traditionally the control problems can be divided into two main classes.

- **Regulation and Tracking:** The objective is to follow a reference trajectory.
- **Optimal Control:** The objective is to minimize a cost function, which can be functional of system parameters and not necessarily of the reference signal.

The control algorithms for first kind are well known including Linear quadratic regulators, model reference controls. Where as for the tasks optimizing functionals of system behavior, model free Reinforcement Learning techniques are popular.

In general, learning control refers to the process of acquiring a control strategy for a particular control system and a particular task by trial and error. Learning control is usually distinguished from adaptive control [2] in that the learning system can have rather general optimization objectives—not just, e.g., minimal tracking error—and is permitted to fail during the process of learning, while adaptive control emphasizes fast convergence without failure. Thus, learning control resembles the way that humans and animals acquire new movement strategies, while adaptive control is a special case of learning control that fulfills stringent performance constraints, e.g., as needed in life-critical systems like airplanes.

In this thesis we will address both the strategies for generating control for a given system and task, namely **"Reinforcement Learning"** and **"Adap-**

tive Control". We hypothesize that bring the concepts of **Adaptation** to Reinforcement Learning and **Learning** to Adaptive control we can develop the algorithms which are more efficient than their stand-alone counterparts. We will now introduce Reinforcement Learning and Adaptive control and motivate the ideas of **Adapting-to-Learn** and **Learning-to-Adapt** in control synthesis problems for uncertain dynamical systems.

1.2 Reinforcement Learning

Reinforcement Learning address the problem of optimizing a functional form of the cost objective, to generate the optimal control action for agent trying to solve a given task. The key issue is the constrained optimization; here optimal-control methods based on the calculus of variations and dynamic programming have been extensively studied. When a detailed and accurate model of the system to be controlled is not available, the classical optimal control solution are not feasible and we use approximate dynamic programming which we call Reinforcement Learning.

The usual tracking problems in control assume prior knowledge of a reference trajectory. But note that the for many problems an optimal reference trajectory is not available. For example locomotion of a bipedal robot. where the task is to design a control policy which enables a robot capable of walking. In such cases one may not be able to specify a desired trajectory for the limbs a priori, but one can specify the objective of moving forward, maintaining equilibrium, not damaging the robot, etc. Reinforcement Learning not only learns such optimal reference trajectories but also generate action which enable optimal reference tracking.

Reinforcement Learning (RL) is based on a very intuitive learning behaviors exhibited by biological agents. The RL agent interacts with the it environments, and takes sequential actions with the goal of maximizing a reward signal, which may be time-delayed. If a particular action sequence leads to favourable outcomes the agent is encouraged to take those actions more often. When the agent interacts with environment sufficiently long enough and maximizing the likelihood of favourable action, we claim to learn optimal policy to solve a given task. But this kind of learning requires a designed reward model which captures the the dynamics of the task. Usually for engi-

neering problems we can design such rewards model or can be assumed to be naturally available through interacting environment. For example, an agent could learn to play a game by being told whether it wins or loses, but is never given the "correct" action at any given point in time. The RL framework has gained popularity as learning methods have been developed that are capable of handling increasingly complex problems. However, as the RL agents are more often very slow in learning and mastering difficult tasks, and thus a significant amount of current RL research focuses on improving the speed of learning.

1.2.1 Sample Complexity of Reinforcement Learning

Sample complexity analysis answers the question how much data needs to collected in order to perform learning successfully? Sample complexity in Reinforcement learning is analogous to classical issue for sample complexity in supervised learning, , but is harder because of the larger input-output space and increased realist nature of the problem of the reinforcement learning setting.

The sample complexity for reinforcement learning is usually defined as function of sampling model and performance criteria used. In particular sample complexity can be defined as the number of calls to the sampling model is required to satisfy a specified performance criteria and how this scales with the relevant problem dependent parameters. The dependent parameters for reinforcement learning problems are state and action space size N, $\mathcal{A}$, problem horizon length or number of decision epochs $\mathcal{H}$ or equivalently discount factor γ for infinite horizon problem, performance criteria ϵ used as accuracy parameter and certainty parameter δ. In a policy search setting, where we desire to find a "good" policy among some restricted policy class Π, the dependency on the complexity of a policy class is also relevant.

Most of the existing RL algorithm, the sample complexity of finding a optimal policy has polynomial dependence on the State-Action space $(N, \mathcal{A})$, exponential in planning horizon length $\mathcal{H}$ and linear dependence on policy space size Π. The algorithms whose sample complexity can be bounded by a polynomial in the environment size and approximation parameters, with high probability, are know as Probably Approximately Correct in Markov Deci-

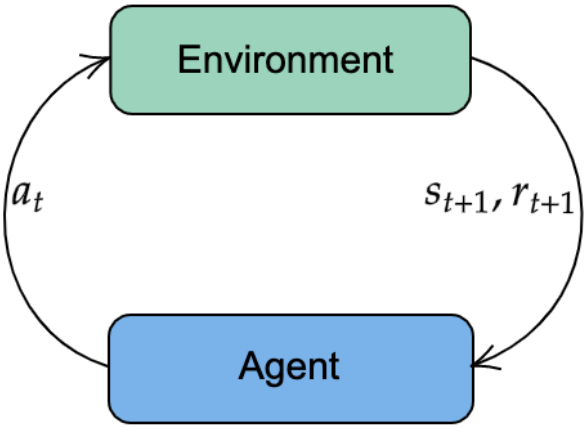


Figure 1.1: Reinforcement Learning

sion Processes (PAC-MDP). For sake of sample complexity analysis the RL algorithms are categorized as Model-based and Model-free algorithms. Difference between model-free and model-based algorithms can be highlighted as, model-based algorithms generally retain some transition information during learning whereas model-free algorithms only keep value-function information.

All algorithms known to be PAC-MDP involve finding solution to a given MDP through value iteration or through dynamic programming of an internal MDP model. Such model based algorithms, include Rmax proposed by Brafman & Tennenholtz's, E3 by Kearns & Singh, and MBIE by Strehl & Littman, are few of the algorithms to mention which provide a sample complexity analysis. These methods are known to have relatively high space and computational complexities of order

$$\tilde{O}\left(N^2 A\right) \tag{1.1}$$

Another class of algorithms, including most forms of Q-learning (Watkins & Dayan, 1992), make no effort to learn a model and can be called model free. The sample complexity of Delayed Q-Leanring is shown to be following by Strehl et.al,

$$\tilde{O}\left(\frac{NAV_{max}^4}{\epsilon^4(1-\gamma)^4}\right) \tag{1.2}$$

This high complexity of reinforcement learning weather model-free or model-based algorithms lead to research in Transfer Learning. Transfer learning can

be defined as improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned. While significant progress has been made to improve learning through transfer in a supervised learning, the idea of transfer learning has only recently been applied to reinforcement learning tasks. We will present some details of Transfer Learning in RL in next section.

1.3 Transfer Learning

Biological agents have inherent ability to evolve and learn from experience. Humans and animals do not learn tasks in isolation. We tend to use our experience and knowledge of related tasks to constantly improve and also learn new but related tasks. This inherent ability to transfer knowledge between tasks, that is, recognize and apply relevant knowledge from previous learning experiences when we encounter new tasks is central point to Transfer Learning.

Contrast to learning in nature Machine learning algorithms in tend to learn tasks in isolation. Transfer learning attempts to change this by developing methods to transfer knowledge learned in one or more source tasks and use it to improve or accelerate learning in a related target task.

The goal of Transfer in Reinforcement Learning is to improve learning in the target task by leveraging knowledge from the source task. The main goal of Transfer Learning is regret minimization, where regret is defined as difference between total achievable maximum reward to achieved reward. There are three most important measures by which we quantify the success of transfer in improving the learning in the target task.

- **Jump-Start** in the achievable performance in the target task using only the transferred knowledge, before any further learning is done, compared to the initial performance of an ignorant agent.

- **Time to Threshold:** The learning time needed by the agent to achieve a pre-specified performance level may be reduced via knowledge transfer. That is how steep is the learning curve. This performance metric is evaluated through time it takes to fully learn the target task given

the transferred knowledge compared to the amount of time to learn it from scratch.

- **Asymptotic Performance**, that is the final performance level achievable in the target task compared to the final level without transfer.

The Transfer Learning in RL can be categorized mainly as Same Domain Transfer and Cross Domain Transfer. The more conventional same domain transfer methods focus on pairs of tasks that are closely related, such as different version of same tasks where agents have the same state $s_t \in \mathcal{S}$ and actions space $a_t \in \mathcal{A}$. A more difficult challenge is to transfer between different domains, where we informally define a domain to be a setting for a group of semantically similar tasks. Cross-Domain transfer require an additional information of inter-task mapping, that is the correspondence between the state an action space of source and target tasks to transfer the policy. Inter-task mapping (which is discussed further in Section) is a way to define how two tasks are related. Most of the time it is assumed the inter-task mapping is provided by the human designer in the loop. However, if the agent is expected to transfer autonomously, such mappings have to be learned. Such an inter-task mapping can be learned in Supervised or Unsupervised setting online or offline. For our experiements on cross domain transfer in this thesis we use a Unsupervised Manifold Alignment technique developed by Wang & Mahadevan [3].

The Transfer Learning involves following steps in transferring skills from source to target domain

- **Source task Selection:** The source selection requires a rational reasoning as, why a policy learned on a given source can prove beneficial in a given target domain learning. More often than not the agent assumes that a human has performed source task selection. Additionally a mechanism can also be designed which selects suitable source task from given set of tasks which avoid negative transfer. While no definitive answer to this problem exists, successful transfer techniques will likely account for specific target task characteristics.

- **Source Policy:** A source policy is defined as a optimal action strategy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ defines how an agent interacts with the environment

mapping perceived system states to actions. It is assumed that an optimal source policy π^* is available which maximizes cumulative reward starting from any state $s_0 \in \mathcal{S}$. There are many possible approaches to learning such a source policy

  - Model-based or Model-Free Algorithms
  - Policy Search Methods
  - Dynamics Programming

- **Inter-task Mapping:** In order to enable transfer learning across tasks with different state and action spaces, one must define how these tasks are related to each other. One way to provide a mapping between State-Action spaces of source and target domains is through Inter-task-mapping χ^S, χ^A. An inter-task mapping is a one-to-one and invertible mapping from source to target domain and visa versa. Such an inter-task mapping can be hand coded or learned in supervised or unsupervised manner over the trajectories collected over both source and target domains.

- **Policy Transfer:** The transfer algorithms can be broadly classified into these methods

  - Life Long Learning [4]
  - Imitation Learning [5, 6, 7, 8],
  - Reward Shaping [9, 10]
  - Representation Transfer [11]

1.3.1 Related work

Deep Reinforcement Learning (D-RL) has recently enabled agents to learn policies for complex robotic tasks in simulation [15, 16, 17, 18]. However, D-RL has been plagued by the curse of sample complexity. Therefore, the capabilities demonstrated in the simulated environment are hard to replicate in the real world. This learning inefficiency of D-RL has led to significant work in the field of Transfer Learning (TL) [19]. A significant body of literature on transfer in RL is focused on initialized RL in the target domain using

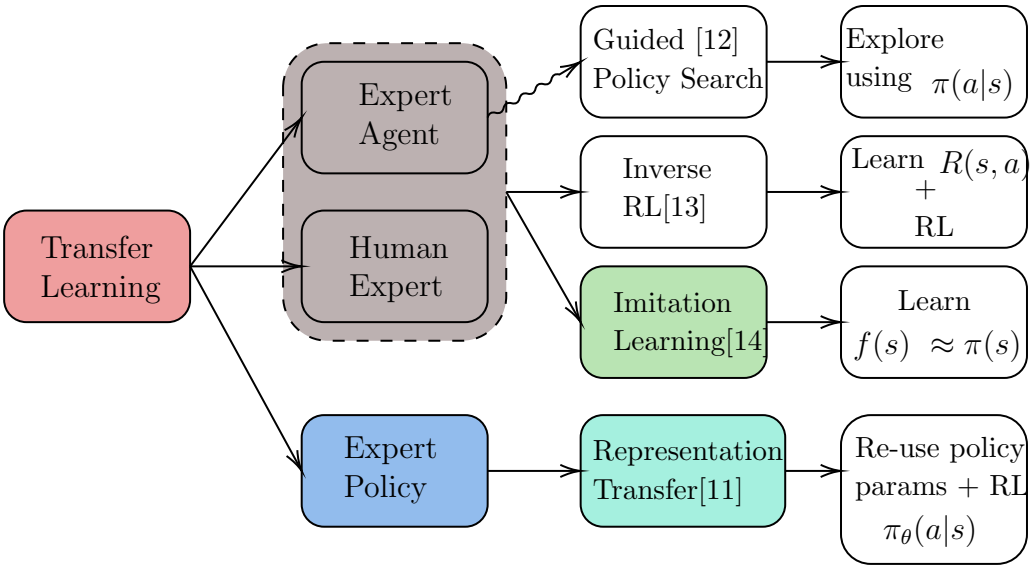


Figure 1.2: Taxonomy of Transfer Learning Algorithms

learned source policy; known as jump-start/warm-start methods [20, 21, 22]. Some examples of these transfer architectures include transfer between similar tasks [23], transfer from human demonstrations [13] and transfer from simulation to real [24, 25, 26]. Efforts have also been made in exploring accelerated learning directly on real robots, through Guided Policy Search (GPS) [27] and parallelizing the training across multiple agents using meta-learning [28, 29, 30]. Sim-to-Real transfers have been widely adopted in the recent works and can be viewed as a subset of same domain transfer problems. Daftry et al. [31] demonstrated the policy transfer for control of aerial vehicles across different vehicle models and environments. Policy transfer from simulation to real using an inverse dynamics model estimated interacting with the real robot is presented by [32]. The agents trained to achieve robust policies across various environments through learning over an adversarial loss is presented in [33]. However, mentioned algorithms and other reported architectures do not *necessarily* lead to improved sample efficiency, handle relatively minor changes in the transition model, and are even known to cause negative transfer [19].

1.4 Adaptive Control

The term **"Adapt"** means to change or alter according to changing situation and purpose. The term **"Adaptive Control"** is used in control engineering for the controller which can learn and adapt their parameters to changing process information in the system. The design of autopilots for high-performance aircraft was one of the primary motivations for active research on adaptive control in the early 1950s. Aircraft operate over a wide range of speeds and altitudes, and their dynamics are nonlinear and conceptually time varying. For such agile system it is found that fixed gain controllers are inadequate to handle all flight regimes. This lead to the development to Adaptive controller which can learn from the system measurement in comparison to a designed reference model and generate control accordingly to cancel the effects of unmodeled dynamics and external disturbances. Thereby Adaptive controllers are preferred when a desired reference tracking is to be achieved in presence of uncertainties. However prior to Adaptive Control there have been few architectures like Integral-Controllers in PID, Gain Scheduling, Robust control architectures like $\mathcal{H}_\infty$ which provide robustness in presence of constant and bounded disturbance.

1.4.1 Model Reference Adaptive control

Model reference Adaptive controller(MRAC) has been a widely studied full-state feedback nonlinear controller, with wide range application in control theory [34, 35, 36, 37]. MRAC have been successfully implemented for control in robotic applications [38], medical processes control [39, 40, 41], Flight system control [42, 43, 44] and many more applications. As the name suggested MRAC architecture aims to design a controller which enables a reference model tracking. That is MRAC enables a system state $x(t)$ track the reference states $x_{rm}(t)$ of an appropriately chosen reference model. The reference model in MRAC is designed to characterize the ideal transient and steady-state response we desire the controlled system to exhibit. MRAC achieves robustness of the baseline controller to the unmodelled dynamics and uncertainty through linearly parametereized adaptive element which aims to cancel the system uncertainties.

Adaptive element in MRAC can be classified as structured and unstruc-

tured networks. In the cases where the uncertainty can be represented as linear weighted combination of the known nonlinear basis functions can be classified as structured uncertainty. The adaptive element in structured uncertainty case can be constructed as linearly parameterized in the known basis function of the uncertainty. In the case where the structure of uncertainty is unknown, the adaptive element is modelled using generic basis functions as polynomials, shallow networks ,radial basis functions or neural networks. We will study these individual choice of basis function in the further part of the thesis. In this thesis we mainly deal with unstructured uncertainty case and study the deep neural network representation of the system uncertainties [45, 46, 38, 47].

The MRAC parameter update law is designed to minimize the tracking error cost defined as $V(t) = e^T(t)e(t)$. The resulting parameter update law is rank-1 update meaning to say it at any given time, the update law looks in one particular direction in parameter space to minimize the tracking error. However this update does not guarantee the convergence in the parameter to their ideal values unless the system is Persistently Exciting(P.E). A detailed mathematical definition of persistency of excitation is given in section-5.6. The P.E condition requires the system states to span the complete spectrum of state space for all arbitrary defined time intervals. Boyd and Shastry have shown the the P.E condition on system states require the exogenous reference signal to be P.E. If the exogenous reference signal $r(t)$ contains as many spectral lines as the number of unknown parameters , then the plant states can be considered as P.E and therefore we can claim exponential parameter convergence. However the condition of P.E on reference signal is restrictive and difficult to monitor online.

Various methods are available in the literature to ensure robustness in the uncertainty estimation despite the lack of P.E of reference signals. A classic $\sigma-$modification to the parameter update law was proposed by Ioannou [48], and $e-$modification was proposed by Narendra [49] which guarantees the adaptive parameters do not diverge even when the system states are not P.E. A projection based modification to the parameter update law is proposed in [41]. The projection operator ensures the parameters remain bounded within a compact set. A $Q-$modification uses integral of tracking error to drive the parameters to a hyper-surface that contains the ideal weights [40]. A $\mathcal{L}_1$ adaptive control is proposed by Cao and Hovakimyan

[50], which uses a high gain learning and low pass filtered adaptive element to ensure robustness and guaranteed transient tracking behaviors. However, due to high learning gains the cannot guarantee parameter convergence even when the system states are P.E. Similarly Nguyen [51] proposed a optimal modification to parameter update which allows use of high learning gains. These proposed methods bound the parameters around a neighborhood of pre-selected value or zero. The boundedness property of the parameters allow us to use Barbalat's Lemma to argue asymptotic convergence in tracking error. But however, these modification can also lead to slow learning or even prevent the adaptive element from estimating constant uncertainties such as trim condition or input biases.

Further more Girish Chowdhary [52] proposed concurrent learning (CL-MRAC) modification to MRAC parameter update law. CL-MRAC use the finitely exciting recorded data concurrently with the current data for adaptation. Finite excitation condition of recorded data is sufficient to guarantee exponential tracking error and parameter convergence in MRAC without requiring stringent P.E condition on reference signal. Authors Chowdhary et.al [53] proposed a Gaussian process Model Reference adaptive control (GP-MRAC) which models the adaptive element as distribution over function rather a deterministic model. GP-MRAC is a non-parametric approach which obviates the requirement to know the domain of operation a priori for feature design. This method also ensures robust learning of system uncertinty using Bayesian inference. In this thesis a new neuroadaptive architecture: Deep Neural Network based Model Reference Adaptive Control (DMRAC) is proposed. The proposed architecture utilizes the power of deep neural network representations for modeling significant nonlinearities while marrying it with the boundedness guarantees that characterize MRAC based controllers. We demonstrate through simulations and analysis that DMRAC can subsume previously studied learning based MRAC methods, such as concurrent learning and GP-MRAC. This makes DMRAC a highly powerful architecture for high-performance control of nonlinear systems with long-term learning properties. The contributions of this thesis is detailed in next section.

1.5 Contribution of This Work

The main contribution of the thesis is in the areas of Policy Transfer in Reinforcement Learning and Adaptive Control. In the controllers available for robotics the model based adaptive controller are at one end of the spectrum and model free Reinforcement learning is at the other. Adaptive controllers take advantage of the model information and carry out tracking error based learning to estimate and cancel the uncertainties online. Thereby enabling to use a baseline controller despite the modelling uncertainty and external disturbances. Since the Adaptive controller are model based and point wise learning in time therefore are highly sample efficient and also enjoys stability and bounded guarantees of the tracking error. These attributes of Adaptive controller make them suitable for safety critical systems like Aerospace applications. At the other end of this spectrum is the model free Reinforcement learning. The RL is trial and error algorithm, which learn the optimal policy through interacting with the environment and over reward feedback. The RL agent is expected to fail multiple times before the agent can find the optimal policy. Hence the RL policy lacks stability during the learning. Also the RL policy is not robust to significant changes in the environment or system parameters. Therefore the application of RL is restricted to simulated agents.

In this thesis the effort is to bring best of both worlds"Adaptation" and "Learning" together in proposing theoretically justified algorithms which perform better than their conventional counterparts. In that effort we borrow control theoretical aspects of Adaptation to Policy transfer in RL and propose two algorithms,

- Model Based Policy Transfer in RL using Target Apprentice.
- Model Free-Adapt to Lean Policy Transfer in RL.

Similarly we borrow the idea of learning over a replay buffer of data collected interacting with the environment and concept of deep Neural networks to Model Reference Adaptive control to propose a new algorithm called

- Deep Model Reference Adaptive Control (DMRAC).

The DMRAC controller uses idea of asynchronous slower update of deep features and faster update of the final layer parameters using classical MRAC

update rule. This time scale separation of update of the network inner layers and last layer allowed us to use a deep neural network in the closed loop of the controller, yet ensure the stability and guaranteed boundedness of the tracking error. The details of the contributions of this thesis are provided in detail in next two subsections.

1.5.1 Adapt to Learn Transfer Learning in RL

Lack of principled mechanisms to quickly and efficiently transfer policies learned between domains has become the major bottleneck in Reinforcement Learning (RL). This inability to transfer or adapt policies is one major reason why RL has still not proliferated physical application like robotics. Since RL agents cannot quickly transfer policies, the agent is forced to learn every task from scratch, which is both time and sample expensive. Warm-start, a method in which weights from one neural network are transferred to another, has been reasonably successful for supervised learning. However, this method can often lead to mixed and even negative results in RL [54, 19].

Our main contribution is an algorithm to transfer policies between tasks with significant differences in state transitions via a policy adaptation mechanism. Unlike the majority of existing work in transfer learning for RL, our approach does not merely use the transferred policy to warm start (initialize the parameter of the target network with learned source network) policy learning in the target domain. Neither does it rely on a multitude of simulations across randomly generated source domains. Instead, we combine supervised reference trajectory tracking and unsupervised reinforcement learning to adapt the source policy to the target domain directly. We show through theory and experiments that our method enjoys significantly reduced sample complexity in solving the task.

In this effort we have proposed both Model-based and Model-Free algorithms to Transfer Policy between source and target tasks.

Model Based-Policy Transfer using Target Apprentice

In this work, we take a different approach from using the source policy to initialize RL in the target. Inspired by the literature in model reference adaptive control (MRAC) [55][56] we propose an algorithm that adapts the source

policy to the target task. We argue that optimal policies retain its optimality across domains that leverage the same physical principle but different state-spaces. We augment the transferred policy by a policy adaptation term, that adapts to the difference between the dynamics of two tasks in target space. The adaptive policy termed as $\pi_{ad}^{(T)}(s)$ is designed to accommodate the difference between the transition dynamics of the projected source and the target task. We prove that such an adapted projected policy to be ϵ-optimal in the target space. The key benefit of this method is that it obviates the need to learn a new policy in the target space, leading to high sample efficient transfer. We prove theoretical bounds on sample efficiency, showing that reduced sample complexity depends polynomially on the cardinality of the replay buffer $|\mathcal{D}|$) and not on the cardinality of the entire state-action pairs of target space ($|\mathcal{S}| \times |\mathcal{A}|$).

Model Free-Adapt-to-Learn Policy Transfer using Target Apprentice

Our main contribution is an algorithm to transfer policies between tasks with significant differences in state transitions via a policy adaptation mechanism. Unlike the majority of existing work in transfer learning for RL, our approach does not just use the transferred policy to warm start (initialize the parameter of the target network with learned source network) policy learning in the target domain [22]; neither does it rely on a multitude of simulations across randomly generated source domains [57]. Instead, we combine supervised adaptation of the source policy and unsupervised reinforcement learning to generate the policy for the target domain. We demonstrate both theoretically and empirically that the proposed method enjoys significantly reduced sample complexity in solving the task.

We posit that the presented method can be the foundation of a broader class of RL algorithms that can choose seamlessly between learning through exploration and supervised adaption resulting in behavioral imitation. Empirically we show that approach is capable of robustly transferring policies between tasks, even in the presence of nonlinear and time-varying differences in the dynamic model of the systems; and theoretically, we ensure $\epsilon-$optimality of such transferred policies in the target domain.

As such, our approach is inspired from two key insights from existing work

in RL and adaptive control. The theory behind adaptive control has been widely studied but has been typically restricted to deterministic dynamical systems with well-defined reference trajectories [55, 56]. We develop adaptive algorithms for reinforcement learning and stochastic MDPs in particular. Imitation Learning (IL) [14, 30] seems to play a crucial part in biological learning, and as such has been widely studied. However, the key is, when presented with a new situation, animals do not just imitate, but quickly adapt, and also learn to get better through further interaction. The ability to adapt and incorporate further learning in a structured way is one fundamental difference between our method and existing imitation learning and Guided Policy Search (GPS) methods [12]. Unlike IL and GPS, our method transfers policies between task with significant differences in the transition model and uncertainty in the parameters in a model agnostic ways. Moreover, by mixing environment reward with intrinsic adaptation rewards, we make sure that we adapt, but also learn in the face of uncertainty to acquire skills beyond what source can teach. We posit that the presented method can be the foundation of a broader class of learning algorithms. By tuning the reward mixing term, the algorithm can choose seamlessly between learning through RL to supervised adaptive imitation. Our approach is capable of robustly transferring policies between tasks, even in the presence of nonlinear and time-varying differences in the dynamic model of the systems. In particular, we show that it suffices to execute adapted greedy policies to ensure $\epsilon-$optimal behavior in the target domain.

1.5.2 Information Enabled Adaptation in Model Reference Adaptive Controller

Deep Neural Networks (DNN) have lately shown tremendous empirical performance in many applications where supervised learning is used, including fields such as computer vision, speech recognition, translation, natural language processing, Robotics, Autonomous driving and many more [58]. Unlike their counterparts such as Gaussian Radial Basis Function networks [59, 60], deep networks learn features by learning the weights of nonlinear compositions of weighted features arranged in a directed acyclic graph [61]. They are indeed much deeper versions of the single hidden layer neural net-

works (SHL-NN) studied in MRAC [62, 63]. It is now pretty clear that deep neural networks are outshining heuristic based regression and classification algorithms as well as RBFNs, GPs, SHL-NNs, and other classical machine learning techniques [64]. However, their utility in the context of control, and especially safety critical control which requires stability guarantees, has been an open question.

Leveraging these successes, there have been many exciting new claims regarding the control of complex dynamical systems in simulation using deep reinforcement learning[65]. However, Deep Reinforcement Learning (D-RL) methods typically do not guarantee stability or even the boundedness of the system during the learning transient. Hence despite significant simulation success, D-RL has seldom been used in safety-critical applications. D-RL methods often make the ergodicity assumption, requiring that there is a nonzero probability of the system states returning to the origin, which in practice is typically enforced by resetting the simulation when a failure occurs. Unfortunately, however, real-world systems do not have this reset option. Unlike, D-RL much effort has been devoted in the field of adaptive control to ensuring that the system stays stable during learning.

In Chapter 5, we present flight test results of Deep MRAC, and present theory that addresses this critical question: How can MRAC utilize deep networks while guaranteeing stability? We present an MRAC architecture that utilizes DNNs as the adaptive element; and an algorithm for the online update of the weights of the DNN by utilizing a dual time-scale adaptation scheme. In our algorithm, the weights of the outermost layers are adapted in real time, while the weights of the inner layers are adapted using batch updates. We also present theory to guarantee Uniform Ultimate Boundedness (UUB) of the entire DMRAC controller.

Our results demonstrate how DNNs can be utilized in stable learning schemes for adaptive control of safety-critical systems such as aircraft. This provides a way to build highly adaptive and long-term learning capable flight controllers. Furthermore, the dual time-scale analysis scheme used by us should be generalizable to other DNN based learning architectures, including reinforcement learning.

1.6 Outline of Thesis

This thesis presents mainly two architectures of Learning through Adaptation in face of Uncertainties: Transfer in Reinforcement Learning and Deep Model Reference Adaptive Control. We begin by discussing basics of Reinforcement learning in Chapter-2. In this chapter we present a taxonomy of various reinforcement learning algorithms. We also present a brief introduction to the state of the art algorithms like value based methods, policy gradient and model based Reinforcement learning algorithm. In Section-2.3, we argue the connection between RL and Adaptive control, and further looking forward the idea of Transfer learning and behavioral adaptation is discussed.

In Chapter-3 & 4 presents details of a new Model-based and Model-free approach to policy transfer in Deep Reinforcement Learning for cross-domain tasks. We present a novel architecture to adapt and reuse the mapped source optimal-policy directly in the related domains. We show the optimal policy from a related source task can be near optimal in the target domain provided an adaptive policy accounts for the model error between the source and target. The main advantage of this policy augmentation over existing transfer techniques, is generalizing the policies across related domains without having to learn in the target tasks. We also show empirically that the presented method enjoys reduced sample complexity depending when compared to stand-alone RL or the state-of-the art Transfer algorithms.

In Chapter-5 the basis of Model reference adaptive controller and stability properties of the controller is presented.

Chapter-6 presents a new architecture for Gaussian Processes Model Reference Adaptive Control (GP-MRAC) trained using a generative network. We will further use the Model reference Generative network for Deep Model Reference Adaptive control in chapter-3. GP-MRAC is a successful method for achieving global performance in the systems enabling adaptive control. GP-MRAC can handle a broader set of uncertainties without requiring apriori knowledge of the domain of operation. However, existing GP-MRAC work requires estimates of the state-derivative, and this is a primary limitation in the implementation of the controller. In this chapter, we alleviate this major limitation by creating Model reference adaptive framework as Generative Network (MRGeN). Our contribution is a generative network architecture for learning Gaussian model to predict system uncertainties without having to

estimate the state derivatives while ensuring that the system stability properties are unaffected. We retain the nonparametric nature of the controller by sharing the kernels between GP's and MRGeN, ensuring global performance and stability guarantees. GP-MRGeN can also be viewed as a method of baseline policy transfers, with applications in Reinforcement Learning.

Further in Chapter-7 presents details of Deep Neural Network based Model Reference Adaptive Control (DMRAC). DMRAC utilizes the power of deep neural network representations for modeling significant nonlinearities while marrying it with the boundedness guarantees that characterize MRAC based controllers. We demonstrate through simulations and analysis that DMRAC can subsume previously studied learning based MRAC methods, such as concurrent learning and GP-MRAC. This makes DMRAC a highly powerful architecture for high-performance control of nonlinear systems with long-term learning properties.

CHAPTER 2

REINFORCEMENT LEARNING

2.1 Introduction

Reinforcement learning (RL) is a machine learning paradigm in which an agent attempts to learn a control policy that can generate the desired sequence of actions for achieving a higher level objective. RL promises to provide a learning mechanism via which autonomous agents can learn to control themselves directly through experience, without requiring manual coding of control policies. Similar to other machine learning paradigms, RL research heavily focuses on end-to-end learning, which in this case is learning of policies directly through experience. Recent successes of RL have shown that agents can learn to decision making and control policies on complex simulations for which it would have been very difficult to manually create control policies. Some examples include chess, go, and more recently complex continuous time simulated domains [65, 66, 67, 15, 16, 17, 18, 68].

2.2 RL in the Markov Decision Process Framework

A stochastic process $\{s_k\}$ is termed Markovian if the conditional probability of a future event s_{k+1} depends only on the event immediately preceding, and not the entire history of events. In particular, $p(s_{k+1}|s_k) = p(s_{k+1}|s_k, s_{k-1}, s_{k-2}, ...)$. The Markov Decision Process (MDP) framework leverages this very key property to lay out a sequential decision making framework: A Markov decision process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, R, \mathcal{P}, \gamma \rangle$, with state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $r(s, a) \in \mathbb{R}$, transition function $p(s_{k+1}|s_k, a_k) \in \mathcal{P}$, and discount factor $\gamma \in [0, 1)$. The goal of the decision making agent is to find a sequence of actions a_k such that the re-

ward is maximized. The reward can be sparse and delayed, that is, it is only available at some states, or only available after a sequence of actions are performed. This makes the MDP sequential decision making problems quite different and rather interesting than one-shot decision making problems such as those studied in the Bandit framework or in linear/non-linear programming. As such, the formulation we discuss below assumes that $\gamma < 1$ and is necessary for infinite horizon sequential decision making problems. In finite horizon problems, such as those studied in [69, 70] γ can take other values.

Given a state s_k, a deterministic non-stationary *policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that outputs an action $a_k = \pi(s_k)$. The goal of MDP problems is to seek out a policy that leads to maximum reward. To further formulate the problem consider that starting from an initial state s_0, the MDP following a policy π will result in a *trajectory* $\zeta = \{[s_0, a_0, r_0], [s_1, a_1, r_1], \cdots\}$. Let us now define the so-called *Q-function* of each state-action pair (s, a) under policy π as the expected sum of discounted reward obtained by starting at any state s and following a policy π thereafter after taking the action $a = \pi(s)$:

$$Q^{\pi}(s, a) = E_{\pi}[\sum_{k=0}^{\infty} \gamma^k r(s_k, a_k) | s_0 = s]. \tag{2.1}$$

Note here that the expectation is on π, that is, following a different policy will lead to a different expected Q value even when one starts on the same state. The Q-function is defined slightly (but very importantly) different than the V-function or the Value function V^{π} for each state s under policy π . The value function is defined only over s (and not over (s, a)) as $V^{\pi}(s) = E_{\pi}[\sum_{k=0}^{\infty} \gamma^k r(s_k, a_k) | s_0 = s]$. That is, it is defined simply as the expected sum of discounted rewards obtained by following the policy π starting from the state s. The goal is to find the optimal policy that maximizes the expected cumulative discounted reward in all states, that is, to find $\pi^*(s) = arg\max_{\pi} V^{\pi}(s)$. Here note that the maximum is over the policy π. The value function of the optimal policy π^* is termed as V^*.

The Bellman equation states that the value of taking action a_k in state s_k under the policy π equals the sum of the immediate reward and the discounted value achieved by following the policy π:

$$Q^{\pi}(s_k, a_k) = r(s_k, a_k) + \gamma Q^{\pi}(x_k, \pi(x_k, a_k)). \tag{2.2}$$

It is rather straight forward to derive this equation, by noting that

$$Q^{\pi}(s_k, a_k) = \sum_{k=0}^{\infty} \gamma^k r(s_k, a_k) = r(s_k, a_k) + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} r(s_k, \pi(s_k)) \tag{2.3}$$

Further expanding the sum leads to the recursion in 2.2.

Similarly, it can be shown that the optimal Q function Q^* follows the Bellman optimality equation $Q^*(x, a) = r(x, a) + \gamma \max_{a'} Q^*(x, a')$. The V function also satisfies the optimality equation $V^*(s_k) = \max_a(r(s_k, a_k) + \gamma V^*(s_{k+1})$, however note that to find the optimal action the knowledge of s_{k+1} or alternatively $\mathcal{P}$ is necessary. On the other hand, with the Q-function

$$\pi^*(s_k) = \arg\max_{\mathrm{a}} Q^*(s_k, a). \tag{2.4}$$

What we have presented is one of the most commonly studied MDP formulations. One key variant is the formulation with stochastic policies, where the policy is a distribution over actions, and given a state s_k the action is sampled from the policy distribution $a_k \sim \pi(s_k)$.

There are many algorithms available to solve the MDP problem in the dynamic programming setting when $\mathcal{R}$ and $\mathcal{P}$ are known. Most of these leverage the fact that the recursion in equation 2.2 is a contraction. Therefore starting with any policy π it is possible to follow first a policy evaluation step to populate the V or the Q function, and then a policy improvement step where for each state the policy $\pi(s) \leftarrow arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(r(s, a) + \gamma V^{\pi}(s'))$. Several of these algorithms are discussed in detail in [71].

In this article, we focus on the so-called Reinforcement Learning (RL) problem, in which $\mathcal{R}$ and $\mathcal{P}$ are assumed to be unknown, and the agent figures out an optimal policy through experience. There are many algorithms available to solve the RL problem, some of which are summarized in Figure-2.1. We provide details of some key methods now.

2.2.1 Q-learning and other value-based methods

Using the Markov assumption on the transition model and the Bellman formula, recursive update expression for learning the state-action value function

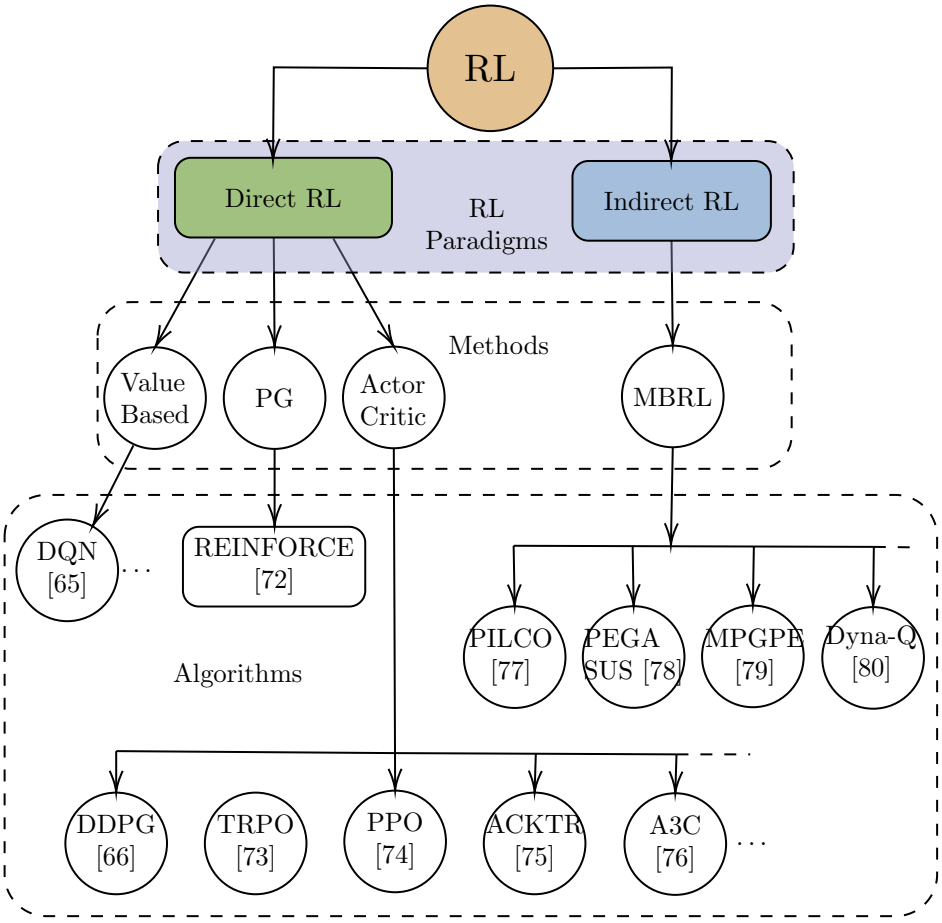


Figure 2.1: Taxonomy of Reinforcement Learning Algorithms

$Q^\pi(s_t, a_t)$ can defined as

$$Q^\pi(s_k, a_k) = r(s_k, a_k) + \gamma Q^\pi(s_{k+1}, \pi(s_{k+1}))$$

The above definition leads us to the famous Q-learning and SARSA algorithms, which use the follwing update form

$$Q^\pi(s_k, a_k) \leftarrow Q^\pi(s_k, a_k) + \alpha\delta$$

where α is the learning rate and δ is defined as temporal-difference (TD-error). Different choice of TD δ error leads to different algorithm

- Q-Learning: $\delta = r(s_k, a_k) + \gamma \max_a Q^\pi(s_{k+1}, a) - Q^\pi(s_k, a_k)$
- SARSA: $\delta = r(s_k, a_k) + \gamma Q^\pi(s_{k+1}, \pi(s_{k+1})) - Q^\pi(s_k, a_k)$

For large or continuous state-action spaces a tabular representation quickly becomes intractable [71]. This issue can be resolved by using a function approximator, such as linear function approaximators [71], Gaussian Processes [81, 60, 82], or neural networks [65, 76, 70, 74, 73]. Deep Q-networks (DQN) is one such algorithm that has solved instability issues of approximate Q-learning with deep neural network function approximators and techniques like experience replay and soft update of the network [65]. DQN uses a cost function of the form

$$J(\theta) = \mathbb{E}_{\{s,a,r,s'\}\sim\mathcal{B}} \left(r + \gamma \max_{a'} Q(s', a', \theta^-) - Q(s, a, \theta) \right)^2$$

where the samples $\{s, a, r, s'\}$ are drawn from the replay buffer $\mathcal{B}$ and θ^- is frozen network parameter from last update.

2.2.2 Policy Gradient (PG) and Actor-Critic methods

Policy gradient method aims at optimizing a parameterized policy $\pi_\theta(a_t|s_t)$ resulting in the higher total discounted return. The total discounted return the policy gradient aims to maximize is defined as

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)} \left[\sum_\tau r(s_k, a_k) \right]$$

where the $\pi_\theta(\tau)$ is the probability of the trajectory and is defined as

$$\pi_\theta(\tau) = \pi_\theta(s_0, a_0, \dots s_T, a_T) = \rho(s_0) \prod_{k=1}^{T} \pi_\theta(a_k|s_k) p(s_{k+1}|s_k, a_k)$$

PG maximizes uses gradient ascent for maximizing the the total expected return $J(\theta)$. The policy gradient update is given as

$$\theta = \theta + \alpha \nabla_\theta J(\theta)$$

where $\nabla_\theta = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\nabla_\theta \log \pi_\theta(\tau) R(\tau) \right]$. The gradient calculation using likelihood-ratio is first introduced in REINFORCE paper by William et.al. [72]. This method makes use of likelihood-ratio trick given by identity

$$\nabla_\theta \pi_\theta(\tau) = \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau)$$

in policy-gradient theorem.

Several variants of the policy gradient can constructed by replacing the total return performance metric in the gradient expression as follows [83]

$$\nabla_\theta = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\nabla_\theta \log \pi_\theta(\tau) \zeta(\tau) \right].$$

Let $\zeta(\tau)$ be the placeholder for total return. Using different estimates of the total return, we can build different policy gradient algorithms, for example $\zeta\tau = R(\tau) = \sum_{k=1}^{T} r(s_k, a_k)$ is the vanilla REINFORCE algorithm which uses bootstrapped total return on trajectory. The causality correction to the total return estimate for gradient variance reduction in REINFORCE use $\zeta\tau = \sum_{k'=t}^{T} r(s_{k'}, a_{k'})$ [72]. A baseline corrected return for variance reduction of gradient estimate employs $\zeta\tau = \sum_{k=1}^{T} r(s_k, a_k) - b_k$, where b_k can be average reward [13]. An actor-critic algorithm uses $\zeta\tau = Q^\pi(s_k, a_k)$ [84], while an Advantage actor critic algorithm would use $\zeta\tau = A^\pi(s_k, a_k)$, where A^π is the advantage function [76].

- $\zeta\tau = R(\tau) = \sum_{k=1}^{T} r(s_k, a_k)$: REINFORCE (total return on trajectory)
- $\zeta\tau = \sum_{k'=t}^{T} r(s_{k'}, a_{k'})$: Causality correction for gradient estimate variance reduction.
- $\zeta\tau = \sum_{k=1}^{T} r(s_k, a_k) - b_k$: Baseline correction for gradient variance

reduction.

- $\zeta\tau = Q^{\pi}(s_k, a_k)$: Actor-Critic Algorithm
- $\zeta\tau = A^{\pi}(s_k, a_k)$: Advantage Actor-Critic Algorithm

2.2.3 Model based RL or "Indirect RL"

In model-based RL a goal is to attempt to learn $\mathcal{T}$ and $\mathcal{R}$ from state-action-reward tuples and use this in learning the policy. Once $\mathcal{T}$ and $\mathcal{R}$ are learned they may be used in any capacity to solve the MDP. The open ended nature of how to use the model in this setting has brought about much debate and discussion on what constitutes model-based RL, as one could argue that model-free uses the value function and MDP assumption as a "model". Generally model-based RL aims takes advantage of structure in the dynamics and cost model resulting in orders of magnitude improvements in data-efficiency over model-free methods, which is essential for some real-world applications. Modern methods typically fall under two different approaches. The first category strongly resembles system-identification where one iteratively estimates a global dynamics model through interactions and uses the model to plan an optimal policy as in model predictive control (MPC), usually via a shooting method. One example is Probabilistic Ensembles with Trajectory Sampling (PETS) [85]: A probabilistic dynamic model is learned via a model ensemble through environment interactions. The optimal control is given by a sampling based shooting methods using the learned probabilistic model. Another approach is Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control [86]: A probabilistic model is learned using a Gaussian Process in order to account for long-term prediction uncertainty. The optimal control is found using MPC and analytic gradients computed via Maximum Principle.

The second category makes use of global or local models to do policy optimization directly as in the model-free case either by using the model derivatives to compute analytic policy updates or sampling the update using the learned dynamics model. In Probabilistic Inference for Learning Control (PILCO) [77] a probabilistic model learned using a Gaussian process and exploits analytic gradients of resulting distribution to optimize the policy.

In Guided Policy Search (GPS) [12] locally linearized model of the dynamics and cost function are used to iteratively update a global policy.

The Model-based assumption can also become a detriment. The policy is biased by the chosen model parameterization and data-distribution and may result in compounding errors in planning or poor local solutions in policy optimization. This is why the most successful model-based methods emphasize model-uncertainty representation. Many on-going works aim to investigate how to most effectively and generally exploit model assumptions and scale to arbitrarily problems with high-dimensional observations.

2.3 Connections between RL and Adaptive Control

Both RL and adaptive control are frameworks designed to find control policies when the model transitions are not known. Adaptive control has classically considered control of continuous time systems using state-space models and well defined tracking objectives [35, 87, 88, 89, 2], while RL has classically focused on decision making policies in discrete state MDPs. The key power of RL is that reward can be sparse or "delayed", that is, the agent does not need to have a reward at every time-step k, rather only in certain states. For example, an agent trying to find a path on a map need only obtain the reward when it reaches it goal, and not necessarily along the way. This is in contrast with traditional adaptive control problems such as Model Reference Adaptive Control in which the agent receives a tracking error reward at every time step. Traditionally, continuous time RL problems are considered difficult to solve due to: 1. Challenges in learning the right approximate representation for continuous domain state-action spaces [71, 90, 91, 92] and 2. Excessive number of experiments required to handle the explore-exploit dilemma in continuous state-spaces [92, 93, 94]. Recently advances in compute power and training of deep neural networks for supervised learning has rekindled fueled a number of works focused on solving continuous time control problems with RL in simulation [95, 66]. The results are promising, given that the agent can learn high-dimensional control tasks merely from trials, however, transition from simulation to real-world has been a challenge, especially in face of results that show that the learned policies may not transfer well across similar systems or degrade rapidly when parameters are only slightly varied.

In essence, these works have focused on learning control policies through experience, but are not yet equipped to adapt the policies in face of change. There has been interest in also using the RL frameworks for solving classical control problems [96, 97, 98].

2.4 Looking forward: Transfer learning and behavioral adaptation

Given enough samples and time, RL can learn policies for complex tasks. However, much work remains to be done in figuring out principled mechanisms to quickly and efficiently transfer policies learned between domains to bring RL to the real world. A significant body of literature on transfer in RL is focused on initialized RL in the target domain using learned source policy; known as jump-start/warm-start methods [20, 21, 22, 23, 13, 24, 25, 26]. However, warm-start, which has been reasonably successful for supervised learning, can often lead to mixed and even negative results in RL [54, 19]. Efforts are also being made in exploring accelerated learning directly on real robots, through Guided Policy Search (GPS) [27] and parallelizing the training across multiple agents using meta-learning [28, 29, 30]. However, these and other reported architectures do not *necessarily* lead to improved sample efficiency, nor are guaranteed to handle relatively minor changes in the transition model, and are even known to cause negative transfer. In essence, many exciting research directions remain in ensuring RL fails gracefully when the underlying model changes, or has robustness margins like linear controllers. Hierarchical RL and behavioral adaptation are accordingly also receiving increasing attention [54].

CHAPTER 3

MODEL BASED POLICY TRANSFER USING TARGET APPRENTICE

3.1 Introduction

Recent successes in Deep Reinforcement Learning (D-RL) have enabled RL agents to solve complex problems [65, 99]. Despite these advancement, we do not yet understand how to efficiently transfer the learned policies from one task to another [19]. In particular, while some level of progress is made, in transferring RL policies in the same state-space domains, the problem of efficient cross-domain skill transfer is still quite open.

In this chapter, we focus on cross-domain policy transfer. We consider the term "Cross-Domain", in a sense that the source and target tasks exploit the same underlying physical principles, but their state spaces are entirely different. This similarity in dynamical behavior of tasks should help a good RL agent to find it easier to solve a target task after it has learned a related source task. Thereby reducing the effort needed to learn an entirely different and seemingly disconnected task. Such a transfer of knowledge will apply to a broad class of problems of practical interest. We can extend this idea of cross-domain transfer to the problems of transfer from simulation to the real world tasks, simple linear system to complex nonlinear tasks and as an architecture towards robustifying the policies under non-stationary environments. Thereby, we argue that we can efficiently harnessing benefits of RL through transfer for real world engineering problems.

Humans are capable of efficiently and quickly generalizing the learned skills between such related tasks [100]. However, RL algorithms capable of performing efficient transfer of policies without learning in the new domain have not yet been reported. To address this gap, leveraging the notions of apprenticeship learning [7] and adaptive control [55, 56], we present a sample efficient algorithm that can directly transfer by adapting the learned policy

from source to the target task. We show that the presented method enjoys reduced sample complexity, depending polynomially only on the size of the chosen replay buffer ($|\mathcal{D}|$) on which the apprentice model is estimated. And not on the cardinality of the entire state-action pairs of the target space ($|\mathcal{S}| \times |\mathcal{A}|$).

Given an optimal source policy, a target apprentice model and an inter-task mapping, we show that it suffices only to execute greedy actions augmented with an adaptive policy to ensure $\epsilon-$optimal behavior in the target.

3.1.1 State of the Art: Transfer Learning in RL

A significant body of literature in Transfer Learning (TL) in the RL, are focused on using the learned source policy as an initial policy in the target task RL [20, 21, 22]. Refer Supplementary document Section-5 Figure-1 for conventional TL architecture. Examples of TL include a transfer in scenarios where the source and target task are similar, and no mapping of state space is needed [23] or transfer from human demonstrations [13]. However, when the source and target task have different state-action spaces, the policy from source cannot be directly used in the target. In this case, a mapping is required between the state-action space of the corresponding source and target tasks to enable knowledge transfer [19]. The inter-task mapping can be supervised; provided by an agent [101], hand-coded using semantics of state features [102, 23, 103], or unsupervised using Manifold Alignment [3, 22] or Sparse Coding Algorithm [21]. Aforementioned TL methods accelerate the learning and minimize regret as compared to stand alone RL. However, initializing the target task learning with the transferred source policy may not lead to significant sample efficiency in the transfer, or even fail to converge leading to negative transfer [104]. In particular, these approaches do not leverage the fact that both tasks exploit the same physical principle and hence the possibility of reusing source policy in the target domain.

3.2 Transfer Learning with Target Apprentice (TA-TL)

This section proposes a novel transfer learning algorithm capable of cross-domain transfer between two related but dissimilar tasks. The presented ar-

chitecture applies to both continuous and discrete state, action space tasks. Unlike the available state of art TL algorithms, which mainly concentrates on policy initialization in the target task RL [19]; we propose to re-use source policy directly as the optimal policy in the related target domain. We achieve this efficient transfer through on-line correction of the transferred policy with adaptive policy, derived based on model transition error. The presented approach has three distinct phases: Phase -1 involves finding an optimal policy over continuous action space of the source task. For this purpose, we use Deep Deterministic Policy Gradient (DDPG) [99] to solve the source MDP(section-3.4). Phase-2 of the transfer; involves discovering a mutual mapping between state, action space of source and target using Unsupervised Manifold Alignment (UMA) (section-3.5). Phase-3 of transfer, is the adaptation of the mapped source optimal policy through policy augmentation in a new target domain (section-4.3). It is to be noted; we do not engage in policy exploration in target space for transfer, we adapt and reuse the projected source policy in the target space, to achieve near-optimal behavior. Nevertheless, with any further exploration, we can improve upon the projected-adapted policy and aim to achieve an optimal solution, but this exercise is left for follow-on work.

3.3 Markov Decision Process

We assume the underlying problem is defined as Markov Decision Process (MDP). An MDP is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{H}, \rho_0, \mathcal{R})$, where $\mathcal{S}$ denote the state of the system; $\mathcal{A}$ set of continuous actions bounded by $\|\mathcal{A}\|_2 \leq \tau$. $\mathcal{P} = P(s, a, s')$ is a Markovian state transition model, the probability of making transition to s' upon taking action a in state s. $\mathcal{H}$ is solution horizon of MDP, such that MDP terminates after $\mathcal{H}$ steps. ρ_0 is distribution over which initial states are chosen and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is deterministic reward function measuring the performance of agent and is assumed to be bounded s.t $\mathcal{R} \in [0, 1]$. The total return starting from states $s_i \in \rho_0$ is defined as sum of discounted reward $V^{\pi} = \mathbb{E}\left(\sum_{i=t}^{\mathcal{H}} \gamma^{i-t} r(s_i, a_i)\right)$ under some policy π and $\gamma \in [0, 1)$ being the discount factor. A policy $\pi : \mathcal{S} \rightarrow \bar{\mathcal{A}}$ is a mapping from states $\mathcal{S}$ to normalized continuous action space, where $\bar{\mathcal{A}} \in (-1, 1)$. The scaled action from policy is obtained as $a_i \leftarrow \tau\pi(s_i)$, where τ is scale factor.

The agent's goal is to find a policy $\pi^\star$ which maximize the total return.

Let π^*_M be projected optimal policy in target MDP M. The associated optimal value function under M is $V^{\pi^*}_M$ for all $s \in \mathcal{S}$. Let $\mathcal{T} : \mathbb{R}^n \to \mathbb{R}^n$ be the Bellman update operator defined as

$$(\mathcal{T}f)(s) = \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim P(s,a)} f(s') \right] \tag{3.1}$$

The optimal value function $V^{\pi^*}_M$ satisfies the Bellman equation (3.1) such that $V^{\pi^*}_M(s) = \mathcal{T}V^{\pi^*}_M(s)$

We formalize the underlying transfer problem by considering a source and target MDP $\mathcal{M}^S = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{H}, \rho_0, \mathcal{R})^S$, $\mathcal{M}^T = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{H}, \rho_0, \mathcal{R})^T$, with its own state space, action space and transition model respectively. We make the following assumption regarding state and action space of source and target tasks.

Assumption 3.3.1 *The state space can differ in dimensionality and span different coordinate frames in respective domains $\mathcal{S}^{(s)} \in \mathbb{R}^n, \mathcal{S}^{(T)} \in \mathbb{R}^m$. We assume the dimensionality of the continuous action space in the source and target tasks are same $\mathcal{A}^{(T)}, \mathcal{A}^{(S)} \in \mathbb{R}^k$. However, the bounds on action amplitude can be different $||\mathcal{A}^{(S)}||_2 \leq \tau^{(S)}, ||\mathcal{A}^{(T)}||_2 \leq \tau^{(T)}$.*

The different dimension action spaces can also be handled, which adds another layer of hierarchy to manifold alignment and will be considered in the future work. We assume an invertible mapping χ_s provides correspondence between the two state space of source and target model. We will use $\hat{s}^S_i$, $\hat{s}^T_i$ to denote the corresponding projected states from target and source spaces respectively. The transition probabilities $\mathcal{P}^S, \mathcal{P}^T$ also differ. However, we assume the physics of the problem share some fundamental similarities in the underlying principles for transfer to be meaningful.

Assumption 3.3.2 *The transition model for the source task,*

$$s^S_{i+1} = \mathcal{P}^{(S)}(\hat{s}^S_i, a^S_i)$$

is available or that we can sample from a source model simulator. This assumption is not very restrictive, since we can always select design related source task for given target task

We learn an approximation to the target transition model using state-action-state buffer $\mathcal{D} = (s_t, a_t, s_{t+1})_{t=i}^{L}$ collected along the trajectories generated by source policy with added random exploration noise. We call this approximate model $\hat{\mathcal{P}}^{(T)}$ as the apprentice to target.

3.4 Learning Source Policy

The presented approach is based on the deterministic policy optimization for source task. Here we present the transfer in continuous action space domain using deep architecture for policy learning. We consider a standard reinforcement learning setup consisting of an agent interacting with an environment in discrete time steps. Starting from initial state $s_0 \sim \rho_0$, the agent takes series of action a_i at each time step i and receives a scalar reward r_i according to some deterministic reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ for $\mathcal{H}$ steps. The goal is to find policy π_θ, such that total discounted reward along the trajectories is maximized.

$$L(\pi_\theta) = \mathop{\mathbb{E}}_{s_o, s_1, \ldots} \left[\sum_{k=0}^{\mathcal{H}} \gamma^k r_{t+k} \right] = \int_{\mathcal{S}} d^{\pi_\theta} V^{\pi_\theta}(s) ds \tag{3.2}$$

where $V^{\pi_\theta} = \mathop{\mathbb{E}}_{s_{t+1}, s_{t+2}, \ldots} \left[\sum_{t=t}^{\mathcal{H}} \gamma^t r_t | s_t = s, \pi_\theta \right]$, d^{π_θ} is state visitation probability and θ is the policy parameter. The discount factor $\gamma \in [0, 1)$ limits the values of the value function V^{π_θ} to be finite and well-defined. It is shown by [99], a deterministic policy gradient w.r.t. the policy parameters exists. Updating the policy using this gradient, moves the parameters in the direction of maximizing the total discounted rewards. For simplicity, we ignore the discount in the state distribution and write the gradient as

$$\nabla_\theta L(\pi_\theta) = \mathop{\mathbb{E}}_{s \sim d^{\pi_\theta}} \left[\nabla_a Q^{\pi_\theta(s,a)} |_{s=s_t, a=\pi_\theta} \nabla_\theta \pi_\theta |_{s=s_t} \right] \tag{3.3}$$

where $Q^{\pi_\theta}(s_t, a_t) = E_{s_{t+1}}[r(s_t, a_t) + \gamma V^{\pi_\theta}(s_{t+1})]$ is called action-value function. Its output can be interpreted as a value of taking a particular action at a particular state and following the policy thereafter. The DDPG algorithm and implementation details can be found in [99]

3.5 Inter task Mapping through Manifold Alignment

Transfer in RL setting, the source and target task have a different representation of state and action spaces. The cross-domain transfer requires an inter-task mapping to facilitate a meaningful transfer. State space $s^{(S)}$, $s^{(T)}$ belonging to two different manifold, cannot be directly compared. Unsupervised Manifold Alignment (UMA) technique helps to discover alignment between two data sets and provide a one to one and onto inter-task mapping. For ease of exposition we demonstrate the transfer for tasks with same action dimensionality. Problems with distinct, non-analogous action spaces is a straight forward extension and use classification methods to find correspondence between action spaces [105, 11]. Details of the inter-state mapping are provided in [22, 3] and reference therein.

3.6 Transfer learning through policy adaptation

In this section, we present the details of the central idea of Policy Transfer through Adaptation using Target Apprentice (TA-TL). Algorithm-1 details TL through policy adaptation using apprentice model. Algorithm-1 leverages the inter-task mapping detailed in section-3.5 to move back and forth between source and target space for knowledge transfer and adaptive policy learning.

The proposed policy transfer architecture is provided in Algorithm-1 and the working details are as follows. For every initial condition in target task $s_0^T \in \mathcal{S}^{(T)}$; s_0^T are mapped to source space to find the corresponding initial condition of source task using the inverse manifold mapping as $\hat{s}_i^S = {\chi_s}^+(s_i^T)$. Where ${\chi_s}^+$ is the inverse mapping from target to source and $\hat{s}_i^S$ represents the image of s_i^T in source state space. For the mapped state in source task, an optimal action is selected using learned policy $\pi^{(S)}(\hat{s}_i^S)$.

Using this selected action a_i^S the source model at state $\hat{s}_i^{(S)}$ is propagated to s_{i+1}^S. The propagated state in source task is mapped back to the target space using inter-task mapping function $\hat{s}_{i+1}^T = \chi_s(s_{i+1}^S)$ where $\hat{s}_{i+1}^T$ is the image of s_{i+1}^S in the target space.

From Assumption-3.3.1, every selected normalized action $\bar{a}^{(s)} \in \bar{\mathcal{A}}^{(S)}$ of source task has equivalent correspondence in the target space. Using this

Algorithm 1 Transfer Learning through Policy Adaptation

Input: $\pi^{*(S)}(s)$, $\chi_{\mathcal{S}}$ and $\hat{\mathcal{P}}^{(T)}$
Initialize $s_0^T \sim \rho_0$.
while $s_i^{(T)} = terminal$ **do**
  $\hat{s}_i^S = \chi_s{}^+(s_i^T)$ {Project the target task state to source space}
  $a_i^S = \tau^{(S)}\pi^{*(S)}(\hat{s}_i^S)$
  {Generate the optimal action using Source policy}
  $s_{i+1}^S = \mathcal{P}^{(S)}(\hat{s}_i^S, a_i^S)$ {Query the source task model at state $\hat{s}_i^S$ and action a_i^S}
  $\hat{s}_{i+1}^T = \chi_s(s_{i+1}^S)$
  {Project back the source task propagated state to target space}
  $\pi_{ad}^{(T)} = \hat{\mathcal{P}}^{(T)}(s_i^T, a_i^T) - \hat{s}_{i+1}^T$ {Compute the adaptive policy}
  $\pi^{*T} = \tau^{(T)}\pi^{*(S)}(\chi_s(s^S)) + \mathcal{K}\pi_{ad}^{(T)}$ {TL policy for target task}
  $a_i^{*T} \sim \pi^{*T}(s_i)$ {Draw action from policy $\pi^{*T}(s_i)$ and propagate the target model}
end while=0

equivalence of actions in normalized action space, a corresponding scaled action $a_i^T \in \mathcal{A}^{(T)}$ in target space is selected as $a_i^T = \tau^{(T)}\pi^{(S)}(\hat{s}_i^{(S)})$. The selected action for target task is augmented with a policy adaption term $a_{ad}^{(T)} \in \mathcal{A}_{ad}^{(T)}$ derived from adaptive policy,

$$\pi_{ad}^{(T)} = \mathcal{P}_{(S)}^{(T)}(s_i^S, a_i^S) - \hat{\mathcal{P}}^{(T)}(s_i^T, a_i^T) \tag{3.4}$$

$$a_{ad}^{(T)} = \pi_{ad}^{(T)}(s^T) \tag{3.5}$$

where $\hat{\mathcal{P}}^{(T)}(s_i^T, a_i^T)$ is apprentice model and $\mathcal{P}_{(S)}^{(T)}(s_i^S, a_i^S) = \chi_s(\mathcal{P}^{(S)}(s_i^S, a_i^S))$ is the projected source model on to target space. The total transferred policy for solving a related target task is proposed to be a linear combination of mapped optimal policy and an adaptive policy as follows,

$$\pi^{*(T)}(.) = \tau^{(T)}\pi^{*(S)}(\chi_s(.)) + \mathcal{K}\pi_{ad}^{(T)}(.) \tag{3.6}$$

The flow diagram of the proposed transfer algorithm is provided in supplementary Section-5 Figure-2.

3.6.1 ϵ-Optimality of the Projected Adapted Policy

We analyze the admissibility of the target policy (3.6) by proving the $\xi(\epsilon)$-optimal solutions of the projected-adapted source policy in target space. For this purpose we assume the target model to be a nonlinear control affine system and the source model be any general nonlinear plant model. The proof follows from the system theory and universal approximation theorem [106] for single hidden layer networks and is provided in the supplementary document.

Theorem 3.6.1 *For any given small $\epsilon \geq 0$, there exists a $\delta(\epsilon)$ such that the difference between true target model and target apprentice model over the entire reachable state-action space be*

$$d_{\mathcal{M}^*,\hat{\mathcal{M}}} = \sup_{(s,a)\in\mathcal{S}\times\mathcal{A}} \|\mathcal{P}^{(T)}(s,a) - \hat{\mathcal{P}}^{(T)}(s,a)\| \leq \delta(\epsilon) \tag{3.7}$$

Then using the modified policy (3.6) we can show the trajectories in target task under policy $\pi^{*(T)}$ are equivalent to projected optimal source trajectories.

$$s_{i+1}^T = \chi_{\mathbf{s}}\left(\mathcal{P}^{(S)}(s_i^S, a_i^S)\right) + \xi(\epsilon) \tag{3.8}$$

where $\xi(\epsilon) = \chi_{\mathbf{s}}\|\mathcal{P}^{(T)}(s,a) - \alpha\hat{\mathcal{P}}^{(T)}(s,a)\|$

Proof We analyze the admissibility of the augmented policy for the target space. Target model is assumed to be any nonlinear, control affine system and source model be any nonlinear system. The discrete transition model for both source and target model can be considered as follows,

$$\mathcal{P}^{(S)}(s,a) : s_{i+1}^S = \mathcal{F}^{*(S)}(s_i^S, a_i^S) \tag{3.9}$$

$$\mathcal{P}^{(T)}(s,a) : s_{i+1}^T = \mathcal{F}^{*(T)}(s_i^T) + \mathcal{B}a_i^T \tag{3.10}$$

where $s^S \in \mathcal{S}^{(S)}, s^T \in \mathcal{S}^{(T)}$ and $a^S \in \mathcal{A}^{(S)}, a^T \in \mathcal{A}^{(T)}$.

The target apprentice is an approximation to the target model. We retain the control affine property of the target model by using appropriate basis of the single layer neural network, to model the target dynamics. The approximate or the apprentice model of the target can be written as function of

network weights and basis as,

$$\hat{\mathcal{P}}^{(T)}(s,a) : s_{i+1}^{T} = \hat{\mathcal{F}}^{(T)}(s_i) + \hat{\mathcal{B}} a_i^{(T)} \tag{3.11}$$

$$= \left[\hat{W} \;\; \hat{\mathcal{B}}\right] \times \left[\psi(s_i^{(T)}) \;\; a_i^{(T)}\right]^T \tag{3.12}$$

where $\hat{\mathcal{F}}^{(T)}(s_t) = \hat{W}^{(T)}\psi(s_i^{(T)})$ and $\phi = \left[\hat{W} \;\; \hat{\mathcal{B}}\right]$, $\Psi(s_i^T, a_i^T) = \left[\psi(s_i^{(T)}) \;\; a_i^{(T)}\right]$ be target apprentice network weights and basis function.

Sampling the action from modified target optimal policy (3.6) $a_i^{(T)} = \pi^{*(T)}((s_i^T))$ and applying it to target model following holds,

$$s_{i+1}^{(T)} = \mathcal{F}^{*(T)}(s_i^T) + \mathcal{B} a_{S,i}^{(T)} + \mathcal{B}\mathcal{K} a_{ad,i}^{(T)} \tag{3.13}$$

where $a_{S,i}^{(T)} = \tau^{(T)}\pi^{*(S)}(\chi_s(s_i^T))$ is the mapped optimal action to target space corresponding to source optimal policy and $a_{ad,i}^{(T)} = \pi_{ad}^{(T)}(s_i^T)$ is modification term to mapped optimal action to cancel the effects of model error.

From definition of model adaptive policy (3.5) and apprentice model (3.12), above expression can be written as

$$s_{i+1}^{(T)} = \mathcal{F}^{*(T)}(s_i^T) + \mathcal{B} a_{(S),i}^{(T)} + \mathcal{B}\mathcal{K}\left(\mathcal{P}_{(S)}^{(T)}(s_i^S, a_i^S) - \hat{\mathcal{F}}^{(T)}(s_i) - \hat{\mathcal{B}} a_i^{(T)}\right)$$

For choice of policy mixture coefficient $\mathcal{K} = \hat{\mathcal{B}}^{-1}$. The above expression simplifies to,

$$s_{i+1}^{T} = \alpha \mathcal{P}_{(S)}^{(T)}(s_i^S, a_i^S) + \left(\mathcal{P}^{(T)}(s,a) - \alpha\hat{\mathcal{P}}^{(T)}(s,a)\right) \tag{3.14}$$

Where $\alpha = \mathcal{B}\hat{\mathcal{B}}^{-1}$, and for persistently exciting data $\mathcal{D} = [s_i, a_i, s_{i+1}]_{i=1}^{N}$ collected for apprentice estimation, convergence of the parameters to true value can be shown [107], ensuring $\alpha \approx 1$

Using the definition of projected model and (3.10) the above expression (3.14), can be rewritten in terms of source transition model using the inter-task mapping function $\chi_{\mathbf{S}}$ as

$$s_{i+1}^{T} = \chi_{\mathbf{s}}\left(\mathcal{F}^{*(S)}(s_i^S, a_i^S)\right) + \xi(\epsilon) \tag{3.15}$$

where $\xi(\epsilon) = \chi_{\mathbf{s}}\|\mathcal{P}^{(T)}(s,a) - \alpha\hat{\mathcal{P}}^{(T)}(s,a)\|$

Expression (3.15) demonstrates that implementation of the modified opti-

mal policy (3.6) in target task is equivalent to projecting the source optimal trajectories on to the target space, leading to $\xi(\epsilon)$-optimal solutions in the target task.

For given any small "$\delta(\epsilon)$" error in the adaptive policy learning, results in the imperfect model cancellation, which we show leads to suboptimal behavior in target task. With the advantage of high sample efficiency of the proposed technique, near-optimal behavior in the target can be acceptable.

3.7 Theoretical bounds on sample complexity

In this section, we prove theoretical bounds on sample complexity of the proposed transfer algorithm. We show that the sample complexity for the proposed method has no dependence on $|\mathcal{S}|$ and $|\mathcal{A}|$.

Though there is enough empirical evidence that transfer can improve performance in subsequent reinforcement-learning tasks, there has been very little theoretical analysis. Since most of the proposed algorithms approach the problem of transfer as a method of providing good initialization to target task RL, we can expect the sample complexity to be still a function of the cardinality of state-action pairs $|N| = |\mathcal{S}| \times |\mathcal{A}|$. On the contrary, we show the sample complexity of the proposed method is free of $|N|$ thereby providing faster and efficient transfer across the domains. Another reason for our proposed algorithm to be efficient is, we engage in supervised learning of the target apprentice to facilitate the transfer as compared to initialized RL in the state of art TL methods.

As we have seen in the previous section, we do not engage in target task learning, but we learn an apprentice to the target model which assist in reusing the optimal source policy. However, we do not require the estimate of the transition model of the target over the entire state and action space. Instead, we show that it is sufficient to have a good approximate model around the trajectories sampled starting from the state $s_0 \sim \rho_0$, using source policy with added exploration noise as behavioral policy. As we collect enough samples in replay buffer $\mathcal{D}$ to express the state-action pairs of interest, we show the sample complexity of cross-domain policy transfer to be a function of $|\mathcal{D}|$.

Definition 3.7.1 *The replay buffer $\mathcal{D}$ is defined as the set of state-action-state pairs collected over the trajectories generated starting from $s_0 \sim \rho_0$ such that the target apprentice model estimated over this buffer satisfies model approximation identity (3.7).*

3.7.1 Sample Complexity of TA-TL

Let's define MDP M^* and $\hat{M}$ which differ only in their transition models. For analysis, we assume an oracle provides us the true adaption term, such that the resulting transition model is equivalent to the true source transition probability projected on to the target task. Let $\hat{M}$ be the model achieved using the adaption term, estimated interacting with the target model. P and $\hat{P}$ be the transition model associated with $M^*, \hat{M}$ respectively. We analyze the sample complexity of the proposed policy transfer through policy adaptation in achieving the ϵ-optimal return.

Definition 3.7.2 *Given the value function $V^* = V^{\pi^*}$ and model M_1 and M_2, which differ only in the corresponding transition models P_1 and P_2, defined $\forall s, a \in \mathcal{S} \times \mathcal{A}$*

$$d^{V^*}_{M_1,M_2} = \sup_{s,a \in \mathcal{S}\times\mathcal{A}} \left| \mathop{\mathbb{E}}_{s' \sim P_1(s,a)} [V^*(s')] - \mathop{\mathbb{E}}_{s' \sim P_2(s,a)} [V^*(s')] \right|$$

Lemma 3.7.3 *If M^* be the MDP with true projected source model on to the target space $P^{(T)}_{(S)}$ and $\hat{M}$ be the MDP with adapted model $\hat{P}^{(T)}_{(S)}$ estimated over $\mathcal{D}$. Using Definition 4.6.2, $\forall s, a \in \mathcal{S} \times \mathcal{A}$ and any $\delta \in (0, 1)$, w.p at least $1 - \delta$, the following error bound is satisfied*

$$d^{V^*}_{M^*,\hat{M}} \leq \rho_v \delta(\epsilon) \sqrt{\frac{|\mathcal{H}|}{2} log\left(\frac{2}{\delta}\right)}$$

where

$$d^{V^*}_{M^*,\hat{M}} = \sup_{s,a \in \mathcal{S}\times\mathcal{A}} \left| \mathop{\mathbb{E}}_{s' \sim P^{(T)}_{(S)}(s,a)} [V^*(s')] - \mathop{\mathbb{E}}_{s' \sim \hat{P}^{(T)}_{(S)}(s,a)} [V^*(s')] \right|$$

The proof of the above lemma is straight forward using finite differences assumption , Hoeffding's lemma and using the fact that the value function $V^{\pi^*}(s)$ is ρ_v-Lipschitz [108], such that $|\nabla V^{\pi^*}(s)| \leq \rho_v$.

Proof Let the value function $V^{\pi^*} = \sum_{i=1}^{\mathcal{H}} \gamma^i r(s_i, a_i)$ be the total discounted reward over $|\mathcal{D}|$ i.i.d trajectories with action taken according to π^*.

For some $c > 0$ and for states $s' \sim P_{(s)}^{(T)}(s,a)$ and $s'' \sim \hat{P}_{(s)}^{(T)}(s,a)$ the bounded difference for all $s, a \in \mathcal{S} \times \mathcal{A}$ can be written as

$$\|V^*(s') - V^*(s'')\|_\infty \leq c$$

We use the first order Taylor series approximation to bound the above error. Using the fact that V^* is ρ_v-Lipschitz and using the Triangle Inequality we have

$$\begin{aligned} \|V^*(s') - V^*(s'')\|_\infty &\leq \left\| \frac{\partial V^*}{\partial s} \right\|_\infty \|(s' - s'')\|_\infty \\ &= \rho_v d_{\mathcal{M}^*, \mathcal{M}} \\ &= \rho_v \delta(\epsilon) \end{aligned}$$

Hence we have shown that $\|V^*(s') - V^*(s'')\|_\infty$ is bounded in the infinity norm, with the nonnegative difference between $V^*(s')$ and $V^*(s'')$ c being

$$c = \rho_v \delta(\epsilon). \tag{3.16}$$

Using the McDiaramids Inequality on the real valued functions $V^* : \mathcal{S} \to \mathbb{R}$, and using the definition of c from eq. 3.16, completes the proof:

$$\begin{aligned} \left| V^*(s') - \mathop{\mathbb{E}}_{s' \sim \hat{P}_{(S)}^{(T)}(s,a)} [V^*(s')] \right| &\leq c \sqrt{\frac{|\mathcal{H}|}{2} \log\left(\frac{2}{\delta}\right)} \\ &= \rho_v \delta(\epsilon) \sqrt{\frac{|\mathcal{H}|}{2} \log\left(\frac{2}{\delta}\right)} \end{aligned}$$

Lemma 3.7.4 *Given M^*, $\hat{M}$ and value function $V_{M^*}^{\pi^*}$, $V_{\hat{M}}^{\pi^*}$ the following bound holds* $\left\| V_{M^*}^{\pi^*} - V_{\hat{M}}^{\pi^*} \right\|_\infty \leq |\mathcal{D}| d_{M^*, \hat{M}}^{V^*}$

The proof of this lemma is finer version of the simulation lemma. The common results are found in PAC exploration algorithms [109]. Similar results for RL with imperfect model are reported by [110].

Proof Let $\mathcal{T}^*, \hat{\mathcal{T}}$ be the Bellman update operator of M^* and $\hat{M}$ respectively. For the bounded function $V^* : \mathcal{S} \to \mathbb{R}$ for $s, a \in \mathcal{S} \times \mathcal{A}$, lets consider the one

step error

$$\left\|V^*(s) - \hat{\mathcal{T}}V^*(s)\right\|_\infty = \left\|\mathcal{T}^*V^*(s) - \hat{\mathcal{T}}V^*(s)\right\|_\infty$$

The above statement is true due to fact $V^*(s)$ satisfies the Bellman equation

$$V^*(s) = \mathcal{T}^*V^*(s)$$

Using the definition of Bellman operator we can write

$$\begin{aligned} & \left\|\mathcal{T}^*V^*(s) - \hat{\mathcal{T}}V^*(s)\right\|_\infty \\ = & \max_{s,a\in\mathcal{S}\times\mathcal{A}} \left| \mathop{\mathbb{E}}_{s'\sim P^{(T)}_{(S)}(s,a)}[V^*(s')] - \mathop{\mathbb{E}}_{s'\sim \hat{P}^{(T)}_{(S)}(s,a)}[V^*(s')] \right| \\ \leq & d^{V^*}_{M^*,\hat{M}} \end{aligned}$$

Therefore we can bound the difference of total discounted value function over the trajectories generated by source optimal policy $\pi^{(S)*}$ over M^* and $\hat{M}$ as follows.

Add and subtract $\hat{\mathcal{T}}V^*_{M^*}$ and applying the Triangular inequality we can write

$$\begin{aligned} \left\|V^*_{M^*} - \hat{\mathcal{T}}V^*_{\hat{M}}\right\|_\infty &= \left\|V^*_{M^*} - \hat{\mathcal{T}}V^*_{M^*} + \hat{\mathcal{T}}V^*_{M^*} - \hat{\mathcal{T}}V^*_{\hat{M}}\right\|_\infty \\ &\leq d^{V^*}_{M^*,\hat{M}} + \left\|\hat{\mathcal{T}}V^*_{M^*} - \hat{\mathcal{T}}V^*_{\hat{M}}\right\|_\infty \\ &= d^{V^*}_{M^*,\hat{M}} + \max_{s,a\in\mathcal{S}\times\mathcal{A}} \left| \mathop{\mathbb{E}}_{s'\sim \hat{P}^{(T)}_{(S)}(s,a)}[V^*_{M^*}(s')] - \mathop{\mathbb{E}}_{s'\sim \hat{P}^{(T)}_{(S)}(s,a)}[V^*_{\hat{M}}(s')] \right| \end{aligned}$$

Repeating the above steps for $\mathcal{D}$ samples on the support $\mathcal{S} \setminus s_1$ yields the result.

Theorem 3.7.5 *let $\Phi, \rho_j > 0$. let $\mathcal{J}_\phi$ be a convex function and let $\phi^* = arg\min_{\phi:\|\phi\|\leq\Phi} \mathcal{J}_\phi$. Also assume that $\|\nabla_\phi \mathcal{J}_\phi\| \leq \rho_j$ w.p 1. Using SGD for target apprentice learning; at least T iterations are required with $\eta = \sqrt{\frac{\Phi^2}{\rho^2 T}}$, where*

$$T \geq \beta^2 \left(\frac{\Phi|\mathcal{D}|^2}{\epsilon^2}\right)^2 \left(\frac{|\mathcal{H}|}{2}\log\left(\frac{2}{\delta}\right)\right)^2$$

s.t to satisfy ϵ-optimality of the value function as given in Lemma-3.7.4 with the adapted policy.

Proof We prove this theorem using convergence rate of mini-batch SGD algorithm. The convergence proof of SGD is elementary and can found in [111]. Let ϕ^* be minimizer of $\mathcal{J}(\phi)$ and be upper bounded by $\|\phi^*\|_\infty \leq \Phi$. Using learning rate η and assuming $\mathcal{J}(\phi^*) = 0$ we know the following results holds

$$\delta(\epsilon)^2 \leq \frac{\Phi\rho_j}{\sqrt{T}} \tag{3.17}$$

where $\delta(\epsilon)$ is defined in (3.7). Using the Lemma-3.7.4 and letting the error in optimal return for adapted policy be ϵ solve for $\delta(\epsilon)$ as

$$\delta(\epsilon) = \frac{\epsilon}{|\mathcal{D}|\rho_v\sqrt{\frac{|\mathcal{H}|}{2}\log\left(\frac{2}{\delta}\right)}}$$

Using this expression for $\delta(\epsilon)$ in (3.17) and setting $\beta = \rho\rho_v^2$, solving for T yields the desired result.

3.8 Target Task Apprentice Learning

Target apprentice is an approximate model for the target task. The target apprentice model is learned over data collected through random exploration in the target domain around projected source trajectories[7]. From simulation lemma [109] we know that it is sufficient to explore in the vicinity of optimal policies to perform near optimally with the approximated model in the target space.

3.8.1 Apprentice Learning

The training dataset is collected starting from random initial condition sampled according to $s_0^j \sim \rho_0$. Actions are executed according to policy $a_i \sim \tau^T\pi^{(s)}(s_i) + a_\epsilon$ where a_ϵ being exploration noise. The resulting trajectories each of length $\mathcal{H}$ are stored $\mathcal{D}_j := \{s_0, a_0, s_1, a_1 \dots s_H\}_j$. Randomly sampled data-points from $\mathcal{D}$ is divided as training inputs $\{s_t, a_t\}_{t=1}^L$ and target value $\{s_{t+1}\}_{t=1}^L$. Using stored data of state-action-state triplets, we train the dy-

namic model $\hat{f}_\phi(s_t, a_t)$ by minimizing the empirical risk using least square regression for linearly parametrized model.

$$\mathcal{J}(\phi) = \frac{1}{\mathcal{D}_{tr}} \sum_{s_t, a_t, s_{t+1} \in \mathcal{D}_{tr}} \left(\frac{1}{2} \| \hat{f}_\phi(s_t, a_t) - s_{t+1} \|^2 \right) \tag{3.18}$$

The network parameters $\phi = [\hat{W}, \hat{\mathcal{B}}]$ are stored upon convergence. The trained model is evaluated over the trajectories not seen during the training. We evaluate the utility of the projected policy $\hat{\pi}^{(T)} = \tau^{(T)} \pi^{*(S)}(\chi_{\mathbf{s}}(s^T))$ in target model on both true and approximate MDP, $M^{(T)}$ and $\hat{M}^{(T)}$. Utility function $U_M(\pi)$ is defined as average reward accumulated for k trials. If $U_M(\hat{\pi}^{(T)}) - U_{\hat{M}}(\hat{\pi}^{(T)}) \leq \zeta$, the minimization routine return the model parameters ϕ, where ζ is some chosen small threshold on the performance.

It should be noted that while a Deep network is ideally suited for learning the value functions, this is not the case for the apprentice model, which is an approximate model of the true dynamics. A single layer neural network is used with suitable activation functions to model the continuous dynamics of the target task. For tasks with more complex transition models, a deep architecture with multiple layers can also be used [112]. This exercise is underway and left for follow-on work.

3.9 Experiments & Results

The primary results presented in this section is focused on the cross-domain transfer of the policy from the cart-pole to the bicycle balancing task and a transfer from simulation to real world scenario through transfer between inverted pendulum domains with the non-stationary target environment. We also demonstrate the proposed method on simple yet challenging cross-domain transfer from Mountain-Car(MC) to Inverted Pendulum (IP). We also examine the robustness of the presented approach against negative transfer through our final experiment. We evaluate the presented Target Apprentice TL(TA-TL) against existing cross-domain state of the art transfer in RL, Unsupervised Manifold Alignment-TL(UMA-TL) [22] and no transfer Deep-RL(DDPG).

3.10 Same-Domain Transfer

We learn the optimal policy in the source task using FQI. In each problem, distinction in the environment/system parameters makes the source and target tasks different. The target and source domains have the same state-space but different transition models and action spaces. We also do not need target reward model be similar to source task, as the proposed algorithm directly adapts the policy from the source task and does not engage in RL in the target domain.

3.10.1 Grid World to Windy Grid World

The source task in this experiment is Non-Windy (NW) grid world. The state variables describing the system are grid positions. The RL objective is to navigate an agent through obstacles from start to goal position optimally. The admissible actions are up $(+1)$, down (-1), right $(+1)$ and left (-1). The reward function is $+10$ for reaching goal position, -1 for hitting obstacles and 0 everywhere else. The target domain is same as the source but with the added wind which affects the transition model in parts of the state-space (see Figure 3.1b).

The optimal policy in source task (non-windy grid world) $\pi^{*(S)}$ is learned using Q-Iteration. We do not need any inter-task mapping as the source, and target state spaces are identical. We start with 100 randomly sampled starting position and execute policy $\pi^{*(S)}$ in the target domain and collect samples for apprentice model learning. Empirically, we show the proposed method (TA-TL) provides a sample efficient TL algorithm compared to other transfer techniques. Figure 3.1a and 3.1b shows the results of same domain transfer in the grid world, demonstrating TA-TL achieving successful transfer in navigating through the grid with obstacles and wind bias. Figure 3.1c and 3.1d shows the quality of transfer through faster convergence to average maximum reward with lesser training samples compared to UMA-TL and RL methods. The presented algorithm can attain maximum average reward in reaching goal position in $\sim 2 \times 10^4$ steps. UMA-TL and RL algorithm achieve similar performance in $\sim 1.2 \times 10^5$ and $\sim 1.7 \times 10^5$ steps respectively, nearly one order higher compared to proposed TA-TL.

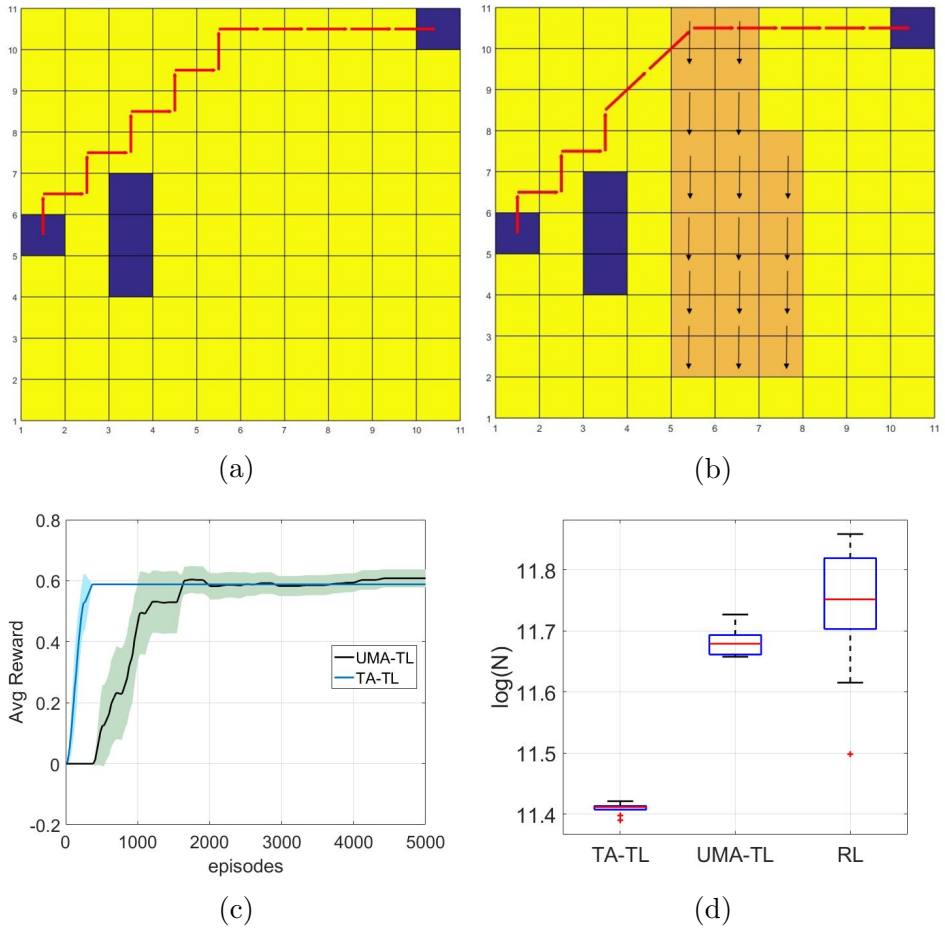
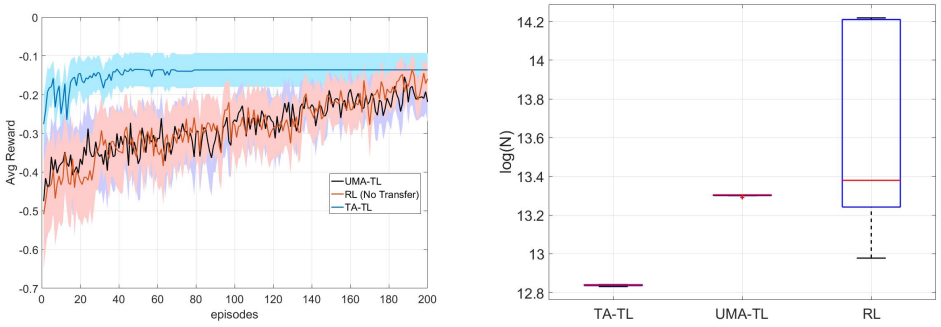


Figure 3.1: Non windy to Windy grid World Transfer:(a) & (b) Agent navigating through grid world in source and target domain (c) Average Rewards & (d) Training length, Comparing quality of transfer for TA-TL and UMA-TL through convergence rate of Average Reward and Training Length

3.10.2 Inverted Pendulum (IP) to time-varying IP

We demonstrate our approach for a continuous state domain, Inverted Pendulum (IP) swing-up and balance Figure-3.2. The source task is the conventional IP domain. The target task differs from the source task in the transition model. The target task is a non-stationary inverted pendulum, where the length and mass of the pendulum are continuously time varying with function $L_i = L_0 + 0.5cos(\frac{\pi i}{50})$ and $M_i = M_0 + 0.5cos(\frac{\pi i}{50})$, where $L_0 = 1$, $M_0 = 1$ and $i = 1 \ldots N$. The state variables describing the system are angle and angular velocity $\{\theta, \dot{\theta}\} \in [-\pi, \pi]$. The RL objective is to swing-up and balance the pendulum upright such that $\theta = 0, \dot{\theta} = 0$. The reward function is selected as $R(\theta, \dot{\theta}) = -10|\theta|^2 - 5|\dot{\theta}|^2$, which yields maximum value at upright position and minimum at the down-most position. The continuous action space is bounded by $T \in [-1, 1]$. Note that the domain is tricky, since full throttle action is assumed to not generate enough torque to be able to swing the pendulum to the upright position, hence, the agent must learn to swing the pendulum back and forth and leverages angular momentum to go to the upright position.

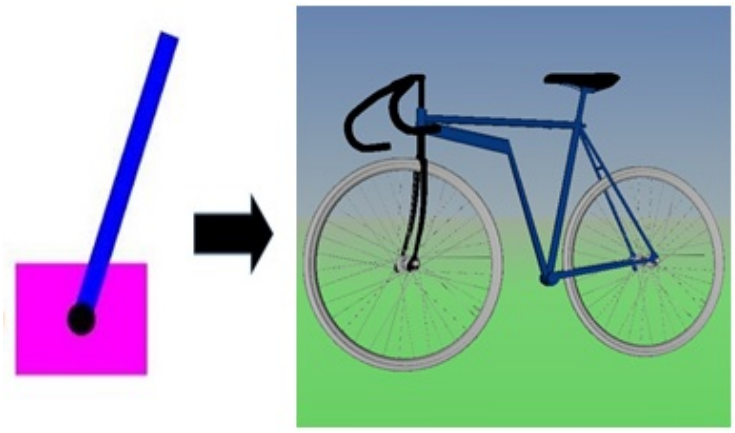


(a) (b)

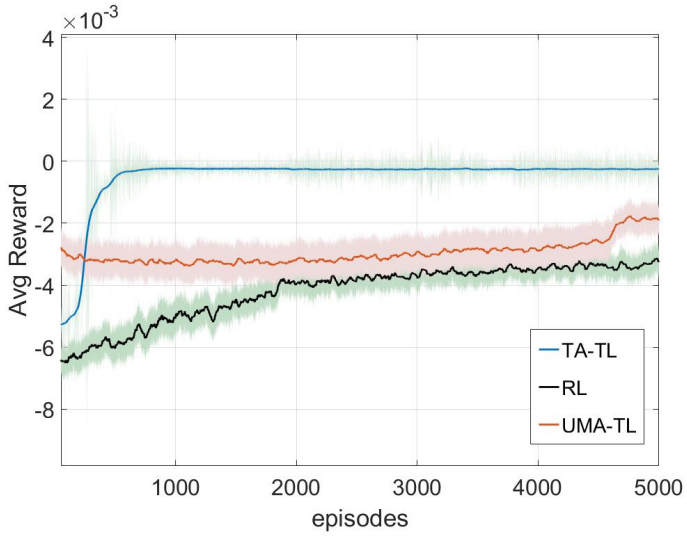
Figure 3.2: Policy Transfer from Inverted Pendulum to Non-stationary Inverted pendulum: (a) Average Rewards and (b) Training length, TA-TL(ATL ours), UMA-TL(Jumpstart-RL) and Stand-alone RL

3.11 Cross Domain Transfer

3.11.1 Cart-Pole to Bicycle:



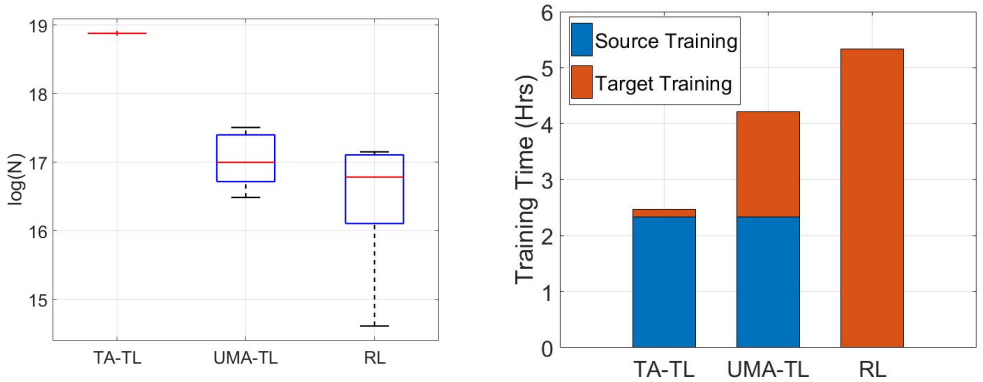
(a)



(b)

Figure 3.3: (a) Cart-Pole and Bicycle Domain (b)Average Rewards for TA-TL (ours), UMA-TL (jumpstart) and RL

Bicycle balancing is a challenging physical problem, especially when the bicycle velocity is below the critical velocity $V_c = 4m/s$ to $5m/s$. We set the bicycle velocity to be $V < V_c$ i.e. $V = 2.778m/s$ such that the bicycle becomes unstable, and active control is required to maintain stability. The simulation itself is very high fidelity and realistic which was designed for



(a) (b)

Figure 3.4: Policy Transfer from Cart-Pole to Bike Balancing: (a) Total simulation time (in seconds) the agent was able to balance the bike in training (b) Total time required to solve the task for TA-TL (ours), UMA-TL (jumpstart) and RL

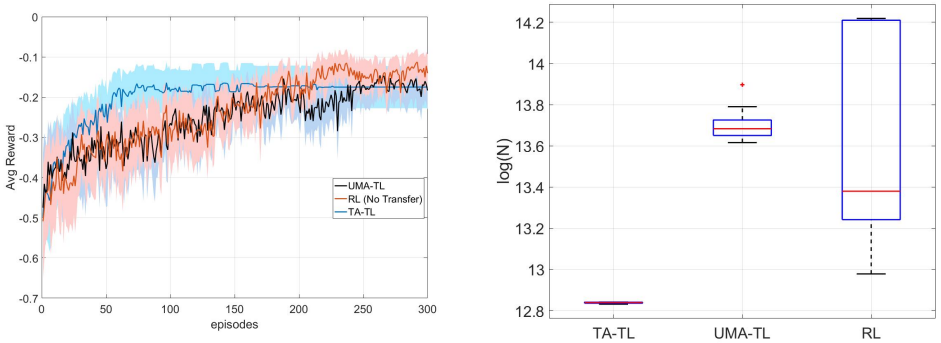
studying the physics of the bicycle.

The states of the bicycle task are angle and angular velocity of the handlebar and the bike from vertical $(\theta, \dot{\theta}, \omega, \dot{\omega})$ respectively. For the given state the agent is in, it chooses an continuous action of applying torque to handlebar, $T \in [-2Nm, 2Nm]$ trying to keep bike upright. The details of bicycle dynamics are beyond the scope of this thesis, interested readers are referred to [113, 114] and references therein.

We use the Cart-Pole as source task for learning to balance a bicycle. The bicycle balance problem in principle is similar to that of cart-pole, since the objective is to keep the unstable system upright. The objective of balance is achieved in both the systems by moving in the direction of fall, which is termed as a non-minimum phase behavior in the controls system literature. The control in the cart pole affects the angle of the pole, by moving the cart such that it is always under the pole. In the bicycle, the control is to move the handlebar in the direction of fall. However, balancing the bike is not straightforward, to turn the bike under itself, one must first steer in the other direction before turning in the direction of fall; this is called counter steering [114]. We observe that both cart pole and bicycle has this commonality in their dynamical behaviors, as both the system have a non-minimum phase that is the presence of unstable zero. This similarity qualifies the cart-pole system as an appropriate choice of source model for bicycle balance task.

Cart pole is characterized by state vector $[x, \dot{x}, \theta, \dot{\theta}]$, i.e., position, velocity of cart and angle, angular velocity of the pendulum. The action space is the force applied to the cart $F \in [-1N, 1N]$. For mapping between the state space of bicycle and cart pole model, we use Unsupervised Manifold Alignment(UMA) to obtain this mapping [3]. We do not report the training time to learn the intertask mapping since it is common to both ATL and UMA-TL methods. Figure 3.3b and 3.4 shows the quality of transfer for ATL through faster convergence to average maximum reward with lesser training samples compared to UMA-TL and RL methods.

3.11.2 Mountain Car (MC) to Inverted Pendulum (IP)



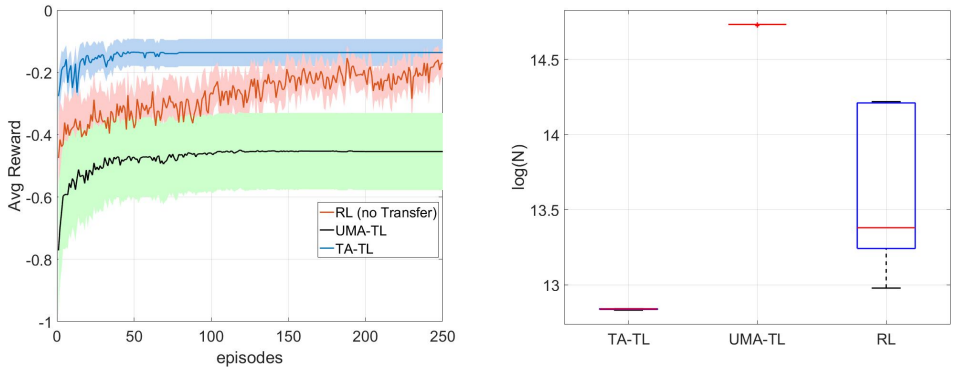
(a) (b)

Figure 3.5: Policy Transfer from Mountain Car to Inverted Pendulum: (a) Average Rewards and (b) Training length

We also demonstrate the cross-domain transfer between mountain car to an inverted pendulum. The source and target task are characterized by different state and action space. The source task MC is a benchmark RL problem of driving an under-powered car up a hill. The dynamics of MC are described by two continuous state variables $(x, \dot{x})$ where $x \in [-1.2, 0.6]$ and $\dot{x} \in [-0.07, 0.07]$ and one continuous action $F \in [-1, 1]$. The reward function is proportional to negative of the square distance of the car from goal position. The target task is conventional IP with state $(\theta, \dot{\theta}) \in (-\pi, \pi)$ and action $T \in [-1, 1]$. We present the performance of transfer methods based on sample efficiency in learning the target task and speed of convergence to maximum average reward. Similar to the bicycle domain transfer Figure 3.5a

and 3.5b shows the quality of transfer for ATL through faster convergence to average maximum reward with lesser training samples compared to UMA-TL and RL methods.

3.11.3 Robustness to Negative Transfer



(a) (b)

Figure 3.6: Negative Transfer Inverted Pendulum: (a) Average Reward and (b) Training length

We demonstrate that the proposed transfer is robust to negative transfers, that is, cases in transfer learning where naive initialization of the target using source policy could be detrimental. We demonstrate this through an inverted pendulum upright balance task with the sign of control flipped. That is, we use inverted pendulum model as both source and target systems but the target is the inverse of the source model with the sign of the control action flipped. It should be noted that dealing with change of sign in control has been considered a highly challenging problem in adaptive control [56]. We demonstrate ATL is immune to negative transfers. Figure 3.6a and 3.6b demonstrate convergence to average maximum reward with lesser training samples for proposed ATL method compared to Initialized-RL(UMA-TL) and standalone-RL methods. It is to be observed that UMA-TL method suffers from a negative transfer phenomenon. The agent under UMA-TL converges to much lower average reward by getting stuck in a local minima and never achieves the upright balance of the pendulum.

Also, the samples observed by UMA-TL in learning the task is much higher compared to no transfer (RL) and proposed ATL methods. If the source

and target are not sufficiently related or the features of source task do not correspond to the target, the transfer may not improve or even decrease the performance in target task leading to negative transfer. We show that the UMA-TL suffers from a negative transfer in this results, where as the performance of presented ATL is much superior compared to UMA-TL and RL(learning from scratch).

CHAPTER 4

ADAPT TO LEARN: POLICY TRANSFER

Efficient and robust policy transfer remains a key challenge in reinforcement learning. Policy transfer through warm initialization, imitation, or interacting over a large set of agents with randomized instances, have been commonly applied to solve a variety of Reinforcement Learning tasks. However, this is far from how skill transfer happens in the biological world: Humans and animals are able to quickly adapt the learned behaviors between similar tasks and learn new skills when presented with new situations. Here we seek to answer the question: Will learning to combine adaptation and exploration lead to a more efficient transfer of policies between domains? IN previous chapter we introduced a model based policy transfer. In this chapter we will introduce a Model-Free **"Adapt-to-Learn"** policy transfer architecture; which will adapt the source policy to learn to solve a target task with significant transition differences and uncertainties. We show through theory and experiments that our method leads to a robust policy transfer algorithm with a significantly reduced sample complexity of transferring the skills between the tasks.

4.1 Introduction

Adapt-to-Learn is inspired by the fact that combined adaptation of behaviors and learning through experience is a primary mechanism of learning in biological creatures [115, 116]. Imitation Learning (IL) [14, 30] also seems to play a very crucial part in skill transfer, and as such has been widely studied in RL. In control theory such adaptation in reference tracking problems has been typically restricted to deterministic dynamical systems with well-defined reference trajectories [55, 56]. Inspired by this ability of biological creatures, we seek to answer the question: Will learning to combine adapta-

tion and learning from exploration lead to more efficient transfer of policies between domains? We demonstrate that the ability to adapt and incorporate further learning can be achieved through optimizing over combined environmental rewards with intrinsic adaptation rewards. Thereby, we ensure that the agent quickly adapts and also learns to acquire skills beyond what the source policy can teach. We experiment the presented policy transfer algorithm between tasks with significant differences in the transition models, which otherwise, using IL or Guided Policy Search (GPS)[12], fail to produce any stable solution.

4.2 Preliminaries

Consider an infinite horizon MDP defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho_0, \gamma)$, where $\mathcal{S}$ denote set of continuous states; $\mathcal{A}$ is a set of continuous bounded actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ is state transition probability distribution of reaching s' upon taking action a in s, $\rho_0 : \mathcal{S} \to \mathbb{R}_+$ is the distribution over initial states s_0 and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ is deterministic reward function.

Let $\pi(a|s) : \mathcal{S} \times \mathcal{A} \to [0, 1]$ be stochastic policy over continuous state and action space. The action from a policy is a draw from the distribution $a_i \sim \pi(a_i|s_i)$. The agent's goal is to find a policy $\pi^\star$ which maximize the total return. The total return under a policy π is given as,

$$\eta_\pi(s_0) = \mathbb{E}_{s_0, a_0, \ldots} \left(\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right). \tag{4.1}$$

where, $s_0 \sim \rho_0$, $a_t \sim \pi(a_t|s_t)$ and $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$.

We will use the following definition of the state value function V^π and state-action value function Q^π defined under any policy π

$$V^\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots} \left(\sum_{i=0}^{\infty} \gamma^i r(s_{i+t}, a_{i+t}) \right),$$

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left(\sum_{i=0}^{\infty} \gamma^i r(s_{i+t}, a_{i+t}) \right).$$

We now formalize the problem of policy transfer: Consider a source MDP $\mathcal{M}^S = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho_0, \gamma)^S$, and a target MDP $\mathcal{M}^T = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho_0, \gamma)^T$,

each with its own state-action space and transition model respectively. We will mainly focus on the problem of same domain transfer, where the state and action space are analogous $\mathcal{S}^{(S)} = \mathcal{S}^{(T)} = \mathcal{S} \subseteq \mathbb{R}^m$ and $\mathcal{A}^{(T)} = \mathcal{A}^{(S)} = \mathcal{A} \subseteq \mathbb{R}^k$, but the source and target state transition models differ significantly due to unmodeled dynamics or external environment interactions. Note that the presented method can also be extended to handle cross-domain transfer using domain alignment techniques such as manifold alignment (MA) [3]. Cross-domain transfer results using the presented Adapt-to-Learn (ATL) algorithm with MA are provided in the supplementary document.

Let π^* be a parameterized optimal stochastic policy for source MDP $\mathcal{M}^S$. The source policy π^* can be obtained using any available RL methods [72, 73, 74]. We do not assume to have access to the parameters of the source optimal policy. We assume source policy to be a stochastic black-box neural network which given states spits out actions. We use the Proximal Policy Optimization (PPO) [74] algorithm to generate the source optimal policy, and a warm-started PPO-TL [22] as the state of the art TL solution against which our ATL algorithm is compared.

4.3 Adapt-to-Learn: Policy Transfer in RL

In the presented transfer algorithm, we approach the problem of transfer learning for RL through adaptation of previously learned policies. Our goal is to show that an algorithm that can judiciously combine adaptation and learning is capable of avoiding brute force random exploration to a large extent and be significantly less sample expensive.

Towards this goal, our approach is to enable the agent to learn the optimal mixture of adaptation (which we term as *behavioral imitation*) and learning from exploration. Our method differs from existing transfer methods that rely on Warm-Started TL or policy imitation [30, 14] in a key way: Unlike imitation learning, we do not aim to emulate the source optimal policy; instead, we try mimic the source transitions under source optimal policy.

4.3.1 Policy Adaptation for Transfer

ATL has two components to policy transfer, Adaptation and Learning. We begin by mathematically describing our approach of adaptation for policy transfer in RL and state all the necessary assumptions in Section 4.3.1. We then develop the Adapt-to-Learn algorithm in Section 4.3.2 by adding the learning from random exploration component.

Adaptation of policies aims to generate the policy π_θ such that at every given $s \in \mathcal{S}^T$ the target transition approximately mimics the source transitions under source optimal policy as projected onto the target manifold. We can loosely state the adaptation objective as

$$\mathcal{P}^T(.|s, \pi_\theta(a|s)) \approx \mathcal{P}^S(.|\hat{s}, \pi^*(a'|\hat{s})), \;\; \forall s \in \mathcal{S}^T, \tag{4.2}$$

Where $\hat{s} = \chi_T^S(s)$, i.e. target state projected onto source manifold. The mapping χ_T^S is a bijective Manifold Alignment (MA) between source and target, state-action spaces $\{\mathcal{S}, \mathcal{A}\}^S$, $\{\mathcal{S}, \mathcal{A}\}^T$. The MA mapping is required for cross-domain transfers [22]. For ease of exposition, we focus on the same domain transfer and assume MA mapping to be identity, such that $\chi_T^S = \chi_S^T = I$.

Note that our goal is not to directly imitate the policy itself, but rather use the source behavior as a guideline for finding optimal transitions in the target domain. This behavioral imitation objective can be achieved by minimizing the average KL divergence between point-wise local target trajectory likelihoods $p^T(s'_{i+1}|s_i, \pi_\theta(s_i))$, and the source transition under source optimal policy $p^S(s'_{i+1}|s_i, \pi^*(s_i))$ as described below.

We define the state transition likelihood as conditional probability of landing in the state s' starting from state s_t under some action $a_t \sim \pi_\theta(a_t|s_t)$. Unlike state transition probability, the state s' is not the next transitioned state but a random state at which this probability is evaluated (Hence s' need not have the time stamp $t+1$). This adaptation objective can be formalized as minimizing the average KL-divergence [73] between source and target transition trajectories as follows,

$$\begin{aligned} \eta_{KL}(\pi_\theta, \pi^*) &= D_{KL}(p_{\pi_\theta}(\tau) \| q_{\pi^*}(\tau)), \\ \eta_{KL}(\pi_\theta, \pi^*) &= \int_{\mathcal{S},\mathcal{A}} p_{\pi_\theta}(\tau) \log \left(\frac{p_{\pi_\theta}(\tau)}{q_{\pi^*}(\tau)} \right) d\tau \end{aligned} \tag{4.3}$$

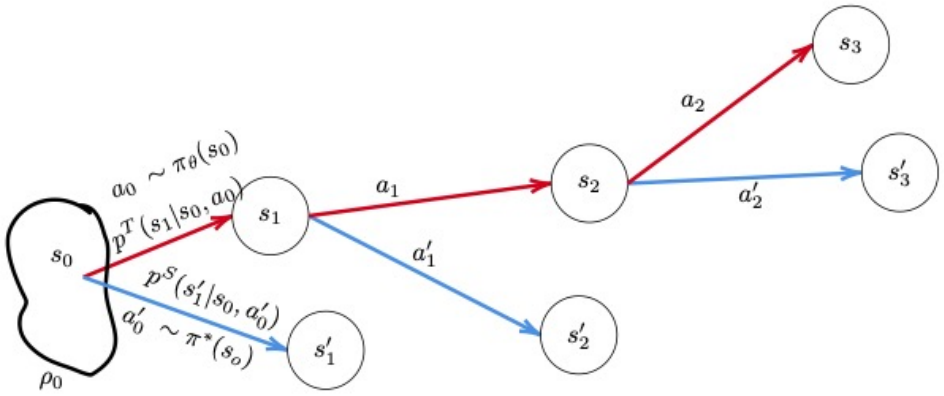


Figure 4.1: Target Trajectory under policy π_θ and local trajectory deviation under source optimal policy π^* and source transition p^S

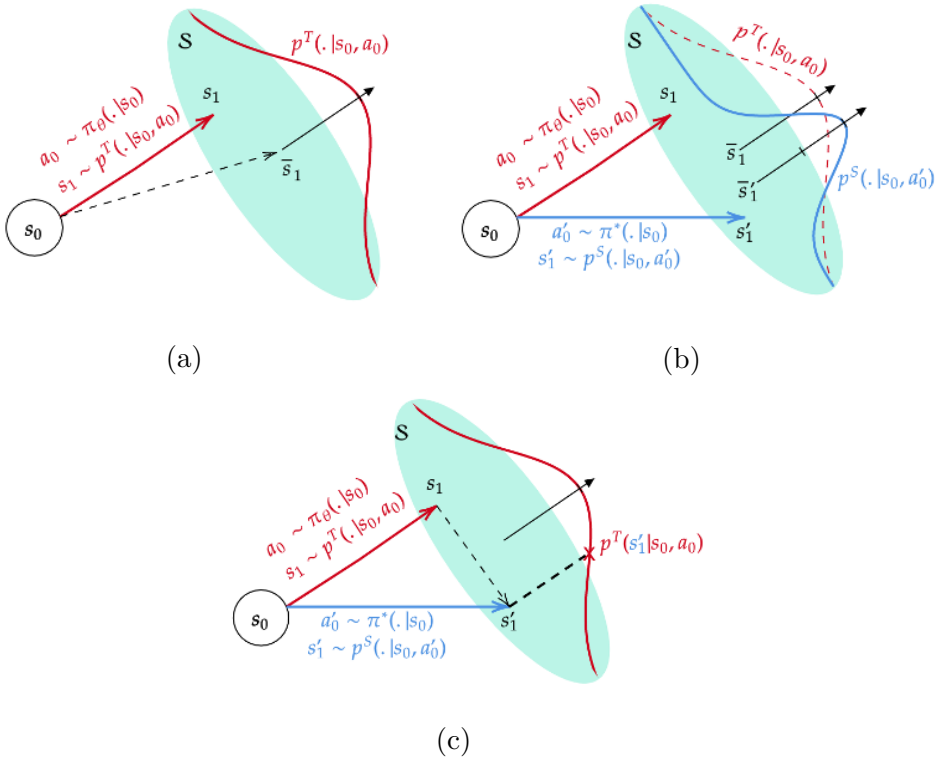


Figure 4.2: Intrinsic KL divergence Objective: (a) One step Transition starting from state s_0 under policy π_θ under target Transition model (b) One step simulated transition from state s_0 under source optimal policy π^* and source transition model (c)Likelihood of landing in state $s_1^{'}$ starting from state s_0 using policy π_θ and target transitions

where τ is the trajectory in the target domain under the policy $\pi_\theta(a|s)$ defined as $\tau = (s_0, s_1, s_2, \dots)$.

The probability of the trajectory $p_{\pi_\theta}(\tau)$ under policy $\pi_\theta(a_t|s_t)$ and target transition $p^T(.|s_t, a_t)$ can be written as

$$p_{\pi_\theta}(\tau) = \rho(s_0) \prod_{t=0}^{\infty} \pi_\theta(a_t|s_t) p^T(s_{t+1}|s_t, a_t). \tag{4.4}$$

Similarly $q_{\pi^*}(\tau)$ can be defined as probability of trajectory deviations at every state $s_t \in \tau$ when the source optimal policy $\pi^*(a'_t|s_t)$ is used in place of target policy $\pi_\theta(a_t|s_t)$, and the states evolve according to the source transitions model $p^S(.|s_t, a'_t)$:

$$q_{\pi^*}(\tau) = \rho(s_0) \prod_{t=0}^{\infty} \pi^*(a'_t|s_t) p^S(s'_{t+1}|s_t, a'_t). \tag{4.5}$$

Unlike the conventional treatment of KL-term in the RL literature, the transition probabilities in the KL divergence in Eq-(4.3) are treated as transition likelihoods and the transitioned state s_{t+1} as a random variable. The policy π_θ learns to mimic the optimal transitions by minimizing the KL distance between the transition likelihoods of the target agent reaching the reference states $\{s'_i\}_{i=1}^{\infty}$ and the source transitions evalauted at $\{s_i\}_{i=0}^{\infty}$.

Using the definition of the probabilities of the trajectories under π_θ and π^* (Equation-(4.4) & (4.5)) the log term in the KL divergence of the trajectory (4.3) can be simplified as follows

$$\log\left(\frac{p_{\pi_\theta}(\tau)}{q_{\pi^*}(\tau)}\right) = \log\left(\frac{\rho(s_0)\pi(a_0|s_0)p^T(s'_1|s_0, a_0)\dots}{\rho(s_0)\pi^*(a'_0|s_0)p^S(s'_1|s_0, a'_0)\dots}\right) \tag{4.6}$$

Assumption 4.3.1 *We assume that optimal source policy is known, and thus, the optimal actions can be chosen with a very high probability* $\pi^*(a'_t|s_t) \approx 1$, $\forall s_t \in \mathcal{S}^T$.

We need this assumption only for deriving the expression for intrinsic reward, which is presented further in this section. In the empirical evaluation of the algorithm, the source policy $\pi^*(a'_t|s_t)$ is used as a stochastic policy to facilitate exploration. However, Assumption-4.3.1 also helps us to extend the proposed transfer architecture to deterministic source policy or source

policy derived from a human expert; since a human teacher cannot specify the probability of choosing an action.

Using Eq-(4.6) and Assumption-4.3.1 i.e. $\pi^*(a'_t|s_t) = 1 \forall s_t$ we derive a surrogate loss $\bar{\eta}_{KL}(\pi_\theta, \pi^*)$ which lower bounds the true KL loss Eq-(4.3), such that,

$$\bar{\eta}_{KL}(\pi_\theta, \pi^*) \leq \eta_{KL}(\pi_\theta, \pi^*),$$

The expression for surrogate loss is defined follows,

$$\bar{\eta}_{KL}(\pi_\theta, \pi^*) = \mathop{\mathbb{E}}_{s_t, a_t \sim \tau} \left(\sum_{t=0}^{\infty} \log \left(\frac{\pi_\theta(a_t|s_t) p^T(s'_{t+1}|s_t, a_t)}{p^S(s'_{t+1}|s_t, a'_t)} \right) \right), \tag{4.7}$$

$$\bar{\eta}_{KL}(\pi_\theta, \pi^*) = \mathop{\mathbb{E}}_{s_t, a_t \sim \tau} \left(\sum_{t=0}^{\infty} \zeta_t \right), \tag{4.8}$$

where

$$\zeta_t = \log \left(\frac{\pi_\theta(a_t|s_t) p^T(s'_{t+1}|s_t, a_t)}{p^S(s'_{t+1}|s_t, a'_t)} \right).$$

4.3.2 Combined Adaptation and Learning

We achieve Adaption and Learning simultaneously by augmenting environment reward $r_t \in \mathcal{R}^T$ with intrinsic reward ζ_t. By optimizing over behavioral imitation intrinsic return and cumulative future environmental reward simultaneously, we achieve transferred policies that try to both follow source advice and learn to acquire skills beyond what source can teach. This trade-off between learning by exploration and learning by adaptation can be realized as follows:

$$\bar{\eta}_{KL,\beta} = \mathop{\mathbb{E}}_{s_t, a_t} \left(\sum_{t=0}^{\infty} \gamma^t ((1-\beta) r_t - \beta \zeta_t) \right). \tag{4.9}$$

For consistency of the reward mixing, the rewards r_t, ζ_t are normalized to form the total reward as follows

$$r'_t = (1-\beta) r_t - \beta \zeta_t. \tag{4.10}$$

where the term β is the mixing coefficient. We learn β for optimal mixing of adaptation and learning over episodic data collected interacting with environment[117].

Assumption 4.3.2 *To calculate the intrinsic reward ζ_t, the true transition distribution for source and the target are unknown. However, we assume both source and target transition models follow a Gaussian distribution centered at the next propagated state with fixed variance "σ", which is chosen heuristically.*

Though Assumption-4.3.2 might look unrealistic, but is not very restrictive. We empirically show that for all the experimented MuJoCo environments, a bootstrapped Gaussian transition assumption is good enough for ATL agent to transfer the policy efficiently.

Using the above assumption, we can approximate the individual KL term (intrinsic reward) as follows,

$$\zeta_t = \log\left(\pi_{\theta^-}(a_t|s_t)e^{-(s_{t+1}-s'_{t+1})^2/2\sigma^2}\right). \tag{4.11}$$

The individual terms in the expectation ζ_t represent the distance between two transition likelihoods of landing in the next state s'_{t+1} starting in s_t and under actions a_t, a'_t. The target agent is encouraged to take actions that lead to states which are close in expectation to a reference state provided by an optimal baseline policy operating on the source model. By doing so, we are providing a possible direction of search for higher environmental rewards r_t.

4.3.3 Actor-Critic

Using the definition of value function, the objective Eq-(4.9) can be written as

$$\bar{\eta}_{KL,\beta} = V^{\pi_\theta}(s). \tag{4.12}$$

We can rewrite the expectation as sum over states and actions as follows:

$$\bar{\eta}_{KL,\beta} = \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q^{\pi_\theta}(s,a), \tag{4.13}$$

where $d^{\pi_\theta}(s)$ is the state visitation distributions [72].

Considering the case of an off-policy RL, we use a behavioral policy $\psi(a|s)$ for generating trajectories. This behavioral policy is different from the policy $\pi_\theta(a|s)$ to be optimized. The objective function in an off-policy RL measures the total return over the behavioral state visitation distribution and actions,

while the mismatch between the training data distribution and the true policy state distribution is compensated by importance sampling estimator as follows,

$$\bar{\eta}_{KL,\beta} = \sum_{s \in \mathcal{S}} d^{\pi_{\theta^-}}(s) \sum_{a \in \mathcal{A}} \left(\psi(a|s) \frac{\pi_\theta(a|s)}{\psi(a|s)} Q^{\pi_{\theta^-}}(s,a) \right), \tag{4.14}$$

where θ^- is the parameter before update and is known to us. Using the previously updated policy as the behavioral policy i.e $\psi(a|s) = \pi_{\theta^-}(a|s)$, the objective expression can be rewritten as,

$$\bar{\eta}_{KL,\beta} = \mathop{\mathbb{E}}_{s_t \sim d^{\pi_{\theta^-}}, a_t \sim \pi_{\theta^-}} \left(\frac{\pi_\theta(a|s)}{\pi_{\theta^-}(a|s)} \hat{Q}^{\pi_{\theta^-}}(s,a) \right). \tag{4.15}$$

Note that we use an estimated state-action value function $\hat{Q}$ rather than the true value function Q because the true value function is usually unknown.

The mixed reward r'_t is used to compute the "critic" state-action value function $\hat{Q}$. Further, any policy update algorithm [72, 74, 73, 84] can be used to update the policy in the direction of optimizing this objective.

Learning over a mixture of intrinsic and environmental rewards helps in the directional search for maximum total return. The source transitions provide a possible direction in which maximum environmental reward can be achieved. This trade-off between directional search and random exploration is achieved using the mixed reward Eq-(4.10). Therefore the proposed source aided policy search in the target domain leads to a more sample efficient algorithm compared to any stand-alone RL policy search methods.

4.4 Optimization of Target Policy

In the previous section, we formulated an Adapt-to-Learn policy transfer algorithm; we now describe how to derive a practical algorithm from these theoretical foundations under finite sample counts and arbitrary policy parameterizations.

We can, solve the following optimization problem to generate adaptive

policy updates:

$$\begin{aligned} \pi_\theta^{*T} &= \arg\max_{\pi_\theta \in \Pi} \left(\bar{\eta}_{KL,\beta}\right), \\ \beta &\leftarrow \beta + \bar{\alpha}\nabla_\beta \bar{\eta}_{KL,\beta}, \\ s.t & \quad 0 \leq \beta \leq 1. \end{aligned}$$

If calculating the above expectation is feasible, it is possible to maximize the objective in Eq-(4.15) and move the policy parameters in the direction of achieving a higher total discounted return. However, this is not generally the case since the true expectation is intractable. Therefore a common practice is to use an empirical estimate of the expectation to do approximate planning.

Algorithm 2 Adapt-to-Learn Policy Transfer in RL

Require: $\pi^*(.|s), p^S$ {Source Policy, source simulator}

1: Initialize $s_0^T \in \rho_0$. {Draw initial state from the given distribution in target task}
2: **for** $i = 1 \leq K$ **do**
3: **while** $s_i \neq terminal$ **do**
4: $a'_i \sim \pi^*(a'_i|s_i)$ {Generate the optimal action using Source policy}
5: $a_i \sim \pi_\theta(a_i|s_i)$ {Generate the action using the π_θ}
6: $s_{i+1} \sim p^T(s_{i+1}|s_i, a_i)$ {Apply the action a_i at state s_i in the target task}
7: $s'_{i+1} \sim p^S(s'_{i+1}|s_i, a'_i)$ {Apply the action a'_i at state s_i in the source simulator}
8: $\zeta_t = \pi_\theta(a_i|s_i)e^{-(s_{i+1}-s'_{i+1})^2/2\sigma^2}$ {Compute the point-wise KL divergence intrinsic reward term}
9: $Z_i = (\{s_i, r_i, \zeta_i, a_i, a'_i\})$ {Incrementally store the trajectory for policy update}
10: **end while**
11: $P_{Z^n_{train}}(\nabla_\theta \bar{\eta}_{KL,\beta})$ {Form the Empirical loss}
12: $\theta' \leftarrow \theta + \alpha P_{Z^n_{train}}\left(\nabla_\theta \bar{\eta}_{KL,\beta}\right)$ {Maximize the total return to update the policy}
13: Collect test trajectories Z^n_{test} using $\pi_{\theta'}$
14: $\beta \leftarrow \beta + \bar{\alpha} P_{Z^n_{test}}\left(\nabla_\beta \bar{\eta}_{KL,\beta}\right)$ {Maximize the total return for optimal mixing coefficient}
15: **end for**=0

4.4.1 Sample-Based Estimation of the Gradient

The previous section proposed an optimization method to find the adaptive policy using KL-divergence as an intrinsic reward, enforcing the target transition model to mimic the source transitions. This section describes how this objective can be approximated using a Monte Carlo simulation. The approximate policy update method work by computing an estimator of the gradient of the return and plugging it into a stochastic gradient ascent algorithm

$$\pi_\theta^{*T} = arg\max_{\pi_\theta \in \Pi} P_{Z^n}(\bar{\eta}_{KL,\beta}), \tag{4.16}$$

$$\theta \leftarrow \theta + \alpha \hat{g}.$$

where α is the learning rate and $\hat{g}$ is the empirical estimate of the gradient of the total return $\bar{\eta}_{KL,\beta}$.

The gradient of the total discounted return is calculated as follows. Lets take the derivative of the total return term

$$\begin{aligned} \nabla_\theta(\bar{\eta}_{KL,\beta}) &= \nabla_\theta V^{\pi_\theta}(s) = \nabla_\theta \left(\sum_a (\pi_\theta(a|s) Q^{\pi_\theta}(s,a)) \right) \\ \nabla_\theta V^{\pi_\theta}(s) &= \sum_a \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) + \sum_a \pi_\theta(a|s) \nabla_\theta Q^{\pi_\theta}(s,a) \end{aligned} \tag{4.17}$$

Using the following definition in above expression,

$$Q^{\pi_\theta}(s_i, a) = p^T(s_{i+1}, |s_i, a)(r + \gamma V_\theta^\pi(s_{i+1}))$$

We can rewrite the gradient to total return over policy π_θ as,

$$\begin{aligned} \nabla_\theta V^{\pi_\theta}(s_0) &= \sum_a \nabla_\theta \pi_\theta(a|s_0) Q^{\pi_\theta}(s_0, a) \\ &+ \sum_{s_1} p^T(s_1, |s_0, a) \sum_a \pi_\theta(a|s_0) \nabla_\theta (r_0 + \gamma V_\theta^\pi(s_1)) \end{aligned} \tag{4.18}$$

As the reward r_t is independent of θ, we can simplify the above expression

and can be re-written as

$$\begin{aligned}\nabla_\theta V^{\pi_\theta}(s_0) &= \sum_a \nabla_\theta \pi_\theta(a|s_0) Q^{\pi_\theta}(s_0, a) \\ &+ \sum_{s_1} \gamma p^T(s_1, |s_0, a) \sum_a \pi_\theta(a|s_0) \nabla_\theta V_\theta^\pi(s_1)\end{aligned} \tag{4.19}$$

As we can see the above expression has a recursive property involving term $\nabla_\theta V^{\pi_\theta}(s)$. Using the following definition of a discounted state visitation distribution d^{π_θ}

$$\begin{aligned}d^{\pi_\theta}(s_0) &= \rho(s_0) + \gamma \sum_a \pi(a|s_0) \sum_{s_1} p^T(s_1|s_0, a) \\ &+ \gamma^2 \sum_a \pi(a|s_1) \sum_{s_2} p^T(s_2|s_1, a) \dots\end{aligned} \tag{4.20}$$

we can write the gradient of transfer objective as follows,

$$\nabla_\theta(\eta_{KL,\beta}) = \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \tag{4.21}$$

Considering an off-policy RL update, where π_{θ^-} is used for collecting trajectories over which the state-value function is estimated, we can rewrite the above gradient for offline update as follows,

Multiplying and dividing Eq-4.21 by $\pi_{\theta^-}(a|s)$ and $\pi_\theta(a|s)$ we form a gradient estimate for offline update,

$$\nabla_\theta(\eta_{KL,\beta}) = \sum_{s \in \mathcal{S}} d^{\pi_{\theta^-}}(s) \sum_{a \in \mathcal{A}} \pi_{\theta^-}(a|s) \frac{\pi_\theta(a|s)}{\pi_{\theta^-}(a|s)} \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} Q^{\pi_{\theta^-}}(s, a) \tag{4.22}$$

where the ratio $\left(\frac{\pi_\theta(a|s)}{\pi_{\theta^-}(a|s)}\right)$ is importance sampling term, and using the following identity the above expression can be rewritten as

$$\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} = \nabla_\theta \log \pi_\theta(a|s)$$

$$\nabla_\theta(\eta_{KL,\beta}) = \mathop{\mathbb{E}}_{s_t \sim d^{\pi_{\theta^-}}, a_t \sim \pi_{\theta^-}} \left(\frac{\pi_\theta(a|s)}{\pi_{\theta^-}(a|s)} Q^{\pi_{\theta^-}}(s, a) \nabla_\theta \log \pi_\theta(a|s) \right) \tag{4.23}$$

The gradient estimate over i.i.d data from the collected trajectories is com-

puted as follows:

$$\begin{aligned}\hat{g} &= P_{Z^n}(\nabla_\theta \bar{\eta}_{KL,\beta}), \\ &= \hat{\mathbb{E}}_{s_t,a_t}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta^-}(a|s)}\hat{Q}^{\pi_{\theta^-}}(s,a)\nabla_\theta \log \pi_\theta(a|s)\right), \end{aligned} \tag{4.24}$$

$$P_{Z^n}(\nabla_\theta \bar{\eta}_{KL,\beta}) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\pi_\theta(a_i|s_i)}{\pi_{\theta^-}(a_i|s_i)}\hat{Q}^{\pi_{\theta^-}}(s_i,a_i)\nabla_\theta \log \pi_\theta(a_i|s_i)\right) \tag{4.25}$$

where P_{Z^n} is empirical distribution over the data $(Z^n : \{s_i, a_i, a_i'\}_i^n)$.

Note that the importance sampling based off-policy update objective renders our discounted total return and its gradient independent of policy parameter θ. Hence the gradient of state-action value estimates $\nabla_\theta \hat{Q}^{\pi_{\theta^-}}(s,a)$ with respect to θ is zero in the above expression for total gradient.

4.5 Learning the Mixing Coefficient

A hierarchical update of the mixing coefficient "β" is carried out over n-test trajectories, collected using the updated policy network $\pi_{\theta'}(a|s)$. Where θ' is parameter after the policy update step. We use stochastic gradient ascent to update the mixing coefficient β as follows

$$\beta \leftarrow \beta + \bar{\alpha}\hat{g}_\beta, \;\; s.t \;\; 0 \leq \beta \leq 1.$$

where $\bar{\alpha}$ is the learning rate and $\hat{g}_\beta = P_{Z^n_{test}}(\nabla_\beta \bar{\eta}_{KL,\beta})$ is the empirical estimate of the gradient of the total return $\bar{\eta}_{KL,\beta}(\pi_{\theta'}, \pi^*)$. The details of computing the estimate $\hat{g}_\beta$ over data $(Z^n_{test} : \{s_i, a_i, a_i'\}_i^T)$ are as follows,

A hierarchical update of the mixing coefficient "β" is carried out over n-test trajectories, collected using the updated policy network $\pi_{\theta'}(a|s)$. The mixing coefficient β is learnt by optimizing the return over trajectory as follows,

$$\beta = arg\max_\beta(\bar{\eta}_{KL,\beta}(\pi_{\theta'}, \pi^*))$$

Where θ' is parameter after the policy update step.

$$\beta = arg\max_{\beta} \mathop{\mathbb{E}}_{s_t, a_t \sim \tau} \left(\sum_{t=1}^{\infty} \gamma^t r'_t \right)$$

We can use gradient ascent to update parameter β in direction of optimizing the reward mixing as follows,

$$\beta \leftarrow \beta + \bar{\alpha} \nabla_{\beta} (\bar{\eta}_{KL,\beta}(\pi_{\theta'}, \pi^*))$$

Using the definition of mixed reward as $r'_t = (1-\beta) r_t - \beta \zeta_t$, we can simplify the above gradient as,

$$\beta \leftarrow \beta + \bar{\alpha} \mathop{\mathbb{E}}_{s_t, a_t \sim \tau} \left(\sum_{t=1}^{\infty} \gamma^t \nabla_{\beta} (r'_t) \right)$$

$$\beta \leftarrow \beta + \bar{\alpha} \mathop{\mathbb{E}}_{s_t, a_t \sim \tau} \left(\sum_{t=1}^{\infty} \gamma^t (r_t - \zeta_t) \right)$$

We use stochastic gradient ascent to update the mixing coefficient β as follows

$$\beta \leftarrow \beta + \bar{\alpha} \hat{g}_{\beta}, \;\; s.t \;\; 0 \leq \beta \leq 1.$$

where $\bar{\alpha}$ is the learning rate and $\hat{g}_{\beta} = P_{Z^n_{test}}(\nabla_{\beta} \bar{\eta}_{KL,\beta})$ is the empirical estimate of the gradient of the total return $\bar{\eta}_{KL,\beta}(\pi_{\theta'}, \pi^*)$. The gradient estimate $\hat{g}_{\beta}$ over data $(Z^n_{test} : \{s_i, a_i, a'_i\}^T_i)$ is computed as follows,

$$\hat{g}_{\beta} = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{\mathcal{H}} \gamma^t (r_t - \zeta_t) \right)$$

where $\mathcal{H}$ truncated trajectory length from experiments.

As we can see the gradient of objective with respect to mixing coefficient β is an average over difference between environmental and intrinsic rewards. If $r_t - \zeta_t \geq 0$ the update will move parameter β towards favoring learning through exploration more than learning through adaptation and visa versa.

As β update is a constrained optimization with constraint $0 \leq \beta \leq 1$. We handle this constrained optimization by modelling β as output of Sigmoidal

network parameterized by parameters ϕ.

$$\beta = \sigma(\phi)$$

And the constrained optimization can be equivalently written as optimizing w.r.to ϕ as follows

$$\phi \leftarrow \phi + \bar{\alpha}\hat{g}_{\beta}\nabla_{\phi}(\beta), \;\; where \;\; \beta = \sigma(\phi)$$

The reward mixing co-efficient β learned for HalfCheetah, Hopper and Walker2d envs is provided in Figure-4.3. For all the experiments we start with $\beta = 0.5$ that is placing equal probability of learning through adaptation and learning through exploration. As we can observe the reward mixing leans towards learning through adaptation for HalfCheetah and Hopper envs. Whereas, as for Walker2d the beta initially believes learning from exploration more, but quickly leans toward learning from source policy and adaptation.

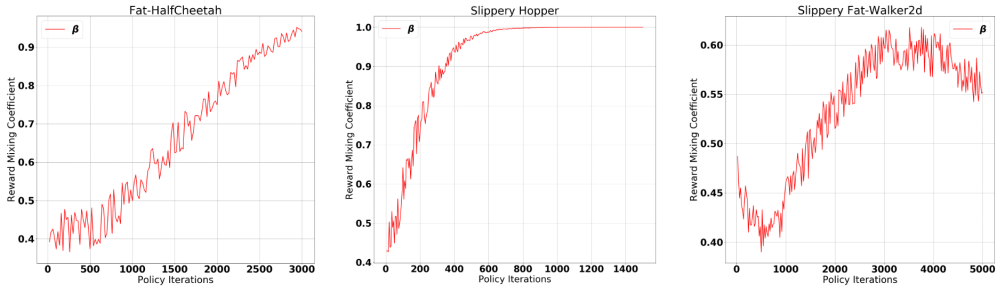


Figure 4.3: The Reward Mixing Co-efficient β for HalfCheetah, Hopper and Walker2d environment learnt over trajectories collected interacting with envs.

4.6 Theoretical bounds on sample complexity

Although there is some empirical evidence that transfer can improve performance in subsequent reinforcement-learning tasks, there are not many theoretical guarantees. Since many of the existing transfer algorithms approach the problem of transfer as a method of providing good initialization to target task RL, we can expect the sample complexity of those algorithms to still be a function of the cardinality of state-action pairs $|N| = |\mathcal{S}| \times |\mathcal{A}|$.

	Hopper	Walker2d	HalfCheetah
State Space	12	18	17
Control Space	3	6	6
Number of layers	3	3	3
Layer Activations	tanh	tanh	tanh
Total num. of network params	10530	28320	26250
Discount	0.995	0.995	0.995
Learning rate (α)	1.5×10^{-5}	8.7×10^{-6}	9×10^{-6}
β initial Value	0.5	0.5	0.5
β-Learning rate ($\bar{\alpha}$)	0.1	0.1	0.1
Batch size	20	20	5
Policy Iter	3000	5000	1500

Table 4.1: Policy Network details and Network learning parameter details

On the other hand, in a supervised learning setting, the theoretical guarantees of the most algorithms have no dependency on size (or dimensionality) of the input domain (which is analogous to $|N|$ in RL). Having formulated a policy transfer algorithm using labeled reference trajectories derived from optimal source policy, we construct supervised learning like PAC property of the proposed method. For deriving, the lower bound on the sample complexity of the proposed transfer problem, we consider only the adaptation part of the learning i.e., the case when $\beta = 1$ in Eq-(4.9). This is because, in ATL, adaptive learning is akin to supervised learning, since the source reference trajectories provide the target states given every (s_t, a_t) pair.

Suppose we are given the learning problem specified with training set $Z^n = (Z1, \dots Z_n)$ where each $Z_i = (\{s_i, a_i\})_{i=0}^n$ are independently drawn trajectories according to some distribution P. Given the data Z^n we can compute the empirical return $P_{Z^n}(\bar{\eta}_{KL,\beta})$ for every $\pi_\theta \in \Pi$, we will show that the following holds:

$$\|P_{Z^n}(\bar{\eta}_{KL,\beta}) - P(\bar{\eta}_{KL,\beta})\| \leq \epsilon. \tag{4.26}$$

with probability at least $1 - \delta$, for some very small δ s.t $0 \leq \delta \leq 1$. We can claim that the empirical return for all π_θ is a sufficiently accurate estimate of the true return function. Thus a reasonable learning strategy is to find a $\pi_\theta \in \Pi$ that would minimize empirical estimate of the objective Eq-(4.9).

Theorem 4.6.1 *If the induced class of the policy π_θ:$\mathcal{L}_\Pi$ has uniform con-*

vergence property in empirical mean; then the empirical risk minimization is PAC. s.t

$$P^n(P(\bar{\eta}_{KL,\hat{\pi}^*}) - P(\bar{\eta}_{KL,\pi^*}) \geq \epsilon) \leq \delta. \tag{4.27}$$

and number of trajectory samples required can be lower bounded as

$$n(\epsilon, \delta) \geq \frac{2}{\epsilon^2(1-\gamma)^2} \log\left(\frac{2|\Pi|}{\delta}\right). \tag{4.28}$$

Proof Fix $\epsilon, \delta > 0$ we will show that for sufficiently large $n \geq n(\epsilon, \delta)$

$$P^n(P(\bar{\eta}_{KL,\hat{\pi}^*}) - P(\bar{\eta}_{KL,\pi^*}) \geq \epsilon) \leq \delta \tag{4.29}$$

Let $\pi^* \in \Pi$ be the minimizer of true return $P(\bar{\eta}_{KL})$, further adding and subtracting the terms $P_{Z^n}(\bar{\eta}_{KL,\hat{\pi}^*})$ and $P_{Z^n}(\bar{\eta}_{KL,\pi^*})$ we can write

$$\begin{aligned} P(\bar{\eta}_{KL,\hat{\pi}^*}) - P(\bar{\eta}_{KL,\pi^*}) &= P(\bar{\eta}_{KL,\hat{\pi}^*}) - P_{Z^n}(\bar{\eta}_{KL,\hat{\pi}^*}) \\ &\quad + P_{Z^n}(\bar{\eta}_{KL,\hat{\pi}^*}) - P_{Z^n}(\bar{\eta}_{KL,\pi^*}) \\ &\quad + P_{Z^n}(\bar{\eta}_{KL,\pi^*}) - P(\bar{\eta}_{KL,\pi^*}) \end{aligned} \tag{4.30}$$

To simplify, the three terms in the above expression can be handled individually as follows,

1. $P(\bar{\eta}_{KL,\hat{\pi}^*}) - P_{Z^n}(\bar{\eta}_{KL,\hat{\pi}^*})$
2. $P_{Z^n}(\bar{\eta}_{KL,\hat{\pi}^*}) - P_{Z^n}(\bar{\eta}_{KL,\pi^*})$
3. $P_{Z^n}(\bar{\eta}_{KL,\pi^*}) - P(\bar{\eta}_{KL,\pi^*})$

Lets consider the term $P_{Z^n}(\bar{\eta}_{KL,\hat{\pi}^*}) - P_{Z^n}(\bar{\eta}_{KL,\pi^*})$ in the above expression is always negative semi-definite, since $\hat{\pi}^*$ is a maximizer wrto $P_{Z^n}(\bar{\eta}_{KL})$, hence $P_{Z^n}(\bar{\eta}_{KL,\hat{\pi}^*}) \leq P_{Z^n}(\bar{\eta}_{KL,\pi^*})$ always, i.e

$$P_{Z^n}(\bar{\eta}_{KL,\hat{\pi}^*}) - P_{Z^n}(\bar{\eta}_{KL,\pi^*}) \leq 0$$

Next the 1st term can be bounded as

$$P(\bar{\eta}_{KL,\hat{\pi}^*}) - P_{Z^n}(\bar{\eta}_{KL,\hat{\pi}^*}) \leq \sup_{\pi \in \Pi}[P_{Z^n}(\eta_{KL}) - P(\bar{\eta}_{KL})]$$

$$\leq \sup_{\pi \in \Pi} \|P_{Z^n}(\bar{\eta}_{KL}) - P(\bar{\eta}_{KL})\|$$

Similarly upper bound can be written for the 3rd term Therefore we can upper bound the above expression as

$$P(\bar{\eta}_{KL,\hat{\pi}^*}) - P(\bar{\eta}_{KL,\pi^*}) \leq 2 \sup_{\pi \in \Pi} \|P_{Z^n}(\bar{\eta}_{KL}) - P(\bar{\eta}_{KL})\|$$

From Equation-(4.29) we have

$$\sup_{\pi \in \Pi} \|P_{Z^n}(\bar{\eta}_{KL}) - P(\bar{\eta}_{KL})\| \geq \epsilon/2 \tag{4.31}$$

Using McDiarmids inequality and union bound, we can state the probability of this event as

$$P^n(\|P_{Z^n}(\bar{\eta}_{KL}) - P(\bar{\eta}_{KL})\| \geq \epsilon/2) \leq 2|\Pi| e^{-\frac{n\epsilon^2}{2C^2\mathcal{H}^2}} \tag{4.32}$$

The finite difference bound

$$C = \frac{1}{1-\gamma}$$

Equating the RHS of the expression to δ and solving for n we get

$$n(\epsilon, \delta) \geq \frac{2}{\epsilon^2(1-\gamma)^2} \log\left(\frac{2|\Pi|}{\delta}\right) \tag{4.33}$$

for $n \geq n(\epsilon, \delta)$ the probability of receiving a bad sample is less than δ.

Env	Property	source	Target	%Change
Hopper	Floor Friction	1.0	2.0	+100%
HalfCheetah	gravity	-9.81	-15	+52%
	Total Mass	14	35	+150%
	Back-Foot Damping	3.0	1.5	-100%
	Floor Friction	0.4	0.1	-75%
Walker2d	Density	1000	1500	+50%
	Right-Foot Friction	0.9	0.45	-50%
	Left-Foot Friction	1.9	1.0	-47.37%

Table 4.2: Transition Model and environment properties for Source and Target task and % change

4.6.1 ϵ-Optimality result under Adaptive Transfer-Learning

Consider MDP M^* and $\hat{M}$ which differ in their transition models. For the sake of analysis, let M^* be the MDP with ideal transition model, such that target follows source transition p^* precisely. Let $\hat{p}$ be the transition model achieved using the estimated policy learned over data interacting with the target model and the associated MDP be denoted as $\hat{M}$. We analyze the ϵ-optimality of return under adapted source optimal policy through ATL.

Definition 4.6.2 *Given the value function $V^* = V^{\pi^*}$ and model M_1 and M_2, which only differ in the corresponding transition models p_1 and p_2. Define $\forall s, a \in \mathcal{S} \times \mathcal{A}$*

$$d^{V^*}_{M_1,M_2} = \sup_{s,a \in \mathcal{S}\times\mathcal{A}} \left| \mathop{\mathbb{E}}_{s' \sim P_1(s,a)}[V^*(s')] - \mathop{\mathbb{E}}_{s' \sim P_2(s,a)}[V^*(s')] \right|.$$

Lemma 4.6.3 *Given M^*, $\hat{M}$ and value function $V^{\pi^*}_{M^*}$, $V^{\pi^*}_{\hat{M}}$ the following bound holds $\left\| V^{\pi^*}_{M^*} - V^{\pi^*}_{\hat{M}} \right\|_\infty \leq \frac{\gamma\epsilon}{(1-\gamma)^2}$*

where $\max_{s,a} \|\hat{p}(.|s,a) - p^*(.|s,a)\| \leq \epsilon$ and $\hat{p}$ and p^* are transition of MDP $\hat{M}, M^*$ respectively.

The proof of this lemma is based on the simulation lemma [1]. Similar results for RL with imperfect models were reported by [110].

Proof For any $s \in \mathcal{S}$

$$\begin{aligned}
&|V^{\pi^*}_{\hat{M}}(s) - V^{\pi^*}_{M^*}(s)|_\infty \\
&= |r(s,a) + \gamma \left\langle \hat{p}(s'|s,a), V^{\pi^*}_{\hat{M}}(s') \right\rangle \\
&-r(s,a) - \gamma \left\langle p^*(s'|s,a), V^{\pi^*}_{M^*}(s') \right\rangle |_\infty
\end{aligned}$$

Add and subtract the term $\gamma \left\langle p^*(s'|s,a), V^{\pi^*}_{\hat{M}}(s') \right\rangle$

$$\begin{aligned}
&= \quad |\gamma \left\langle \hat{p}(s'|s,a), V^{\pi^*}_{\hat{M}}(s') \right\rangle - \gamma \left\langle p^*(s'|s,a), V^{\pi^*}_{\hat{M}}(s') \right\rangle \\
&+ \gamma \left\langle p^*(s'|s,a), V^{\pi^*}_{\hat{M}}(s') \right\rangle - \gamma \left\langle p^*(s'|s,a), V^{\pi^*}_{M^*}(s') \right\rangle |_\infty \\
&\leq \gamma| \left\langle \hat{p}(s'|s,a), V^{\pi^*}_{\hat{M}}(s') \right\rangle - \left\langle p^*(s'|s,a), V^{\pi^*}_{\hat{M}}(s') \right\rangle | \\
&+ \gamma| \left\langle p^*(s'|s,a), V^{\pi^*}_{\hat{M}}(s') \right\rangle - \gamma \left\langle p^*(s'|s,a), V^{\pi^*}_{M^*}(s') \right\rangle |_\infty \\
&\leq \gamma|\hat{p}(s'|s,a) - p^*(s'|s,a)|_\infty |V^{\pi^*}_{\hat{M}}(s')|_\infty \\
&\quad +\gamma|V^{\pi^*}_{\hat{M}}(s) - V^{\pi^*}_{M^*}(s)|_\infty
\end{aligned}$$

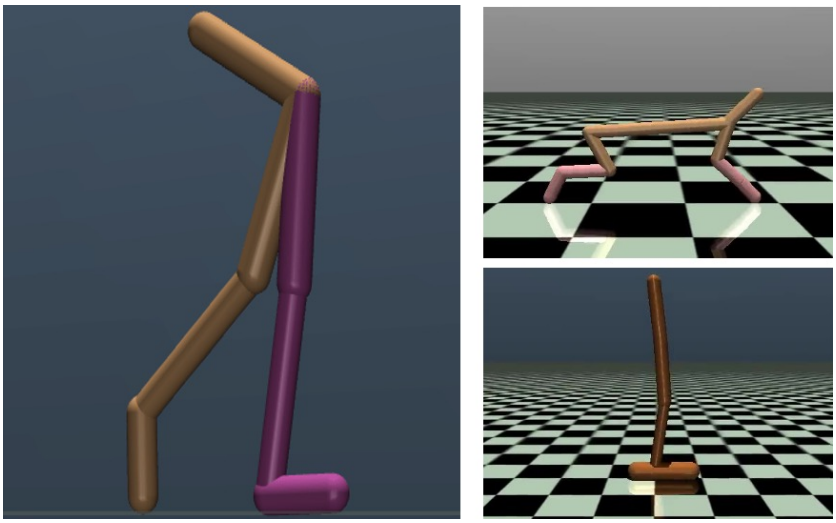


Figure 4.4: 2D robot models used for locomotion experiments. From left to right clockwise: Walker2d, Half-Cheetah, Hopper. These tasks are challenging to control as they are under-actuated and have contact discontinuities.

Using the definition of ϵ in above expression, we can write

$$|V^{\pi^*}_{\hat{M}}(s) - V^{\pi^*}_{M^*}(s)|_\infty \leq \gamma\epsilon|V^{\pi^*}_{\hat{M}}(s')|_\infty + \gamma|V^{\pi^*}_{\hat{M}}(s) - V^{\pi^*}_{M^*}(s)|_\infty$$

Therefore

$$|V^{\pi^*}_{\hat{M}}(s) - V^{\pi^*}_{M^*}(s)|_\infty \leq \frac{\gamma\epsilon|V^{\pi^*}_{\hat{M}}(s')|_\infty}{1-\gamma}$$

Now we solve for expression $|V^{\pi^*}_{\hat{M}}(s')|_\infty$. We know that this term is bounded as

$$|V^{\pi^*}_{\hat{M}}(s')|_\infty \leq \frac{R_{max}}{1-\gamma}$$

where $R_{max} = 1$, therefore we can write the complete expression as

$$|V^{\pi^*}_{\hat{M}}(s) - V^{\pi^*}_{M^*}(s)|_\infty \leq \frac{\gamma\epsilon}{(1-\gamma)^2}$$

4.7 Policy transfer in simulated robotic locomotion tasks

To evaluate Adapt-to-Learn Policy Transfer in reinforcement learning, we design our experiments using sets of tasks based on the continuous control

environments in MuJoCo simulator [118]. Our experimental results demonstrate that ATL can adapt to significant changes in transition dynamics. We perturb the parameters of the simulated target models for the policy transfer experiments (see Table-4.2 for original and perturbed parameters of the target mode). To create a challenging training environment, we changed the parameters of the model such that the optimal source policy alone without any learning cannot produce any stable results (see source policy performance in Figure-4.5). We compare our results against two baselines: (a) Initialized Reinforcement learning (initialized PPO) (Warm-Start-RL [22]) (b) Stand-alone reinforcement policy learning (PPO) [74].

We experiment with the ATL algorithm on Hopper, Walker2d, and HalfCheetah Environments (Refer Figure-4.4). The states of the robots are their generalized positions and velocities, and the actions are joint torques. High dimensionality, non-smooth dynamics due to contact discontinuity, and being under-actuated systems make these tasks very challenging. We use deep neural networks to represent the source and target policy, the details of which are in the Table-2 Supplementary document. The following models are included in our evaluation:

***Slippery Hopper*:** is defined through 11-dimensional state space and 3-dimension action space, with reward function defined as $r(t) = (s_{t+1} - s_t)/dt - 10^{-3}\|a\|_2$, and a bonus of +1 for being in a non-terminal state. The simulation is terminated upon reaching 1000 steps or hopper toppling. The target model differs in the floor friction and foot joint damping.

***Slippery Fat-Walker2d*:** is defined through 17-dimensional state space and 6-dimension action space, with reward function and termination condition defined the same as Hopper. The target model differs in the model density and floor friction.

***Fat HalfCheetah*:** is defined through 17-dimensional state space and 6-dimension action space, with reward function defined as $r(t) = (s_{t+1} - s_t)/dt - 0.1\|a\|_2$. The simulation is terminated upon reaching 1000 steps. The target model differs in the floor friction coefficient, gravity, and mass.

To establish a standard baseline, we also included the classic cart-pole and Inverted pendulum balancing tasks, based on the formulation [119]. We also demonstrate the cross-domain transfer capabilities using a model-based variant of the proposed algorithm. [54] The results of policy transfer for Cart-Pole to Inverted Pendulum and Inverted Pendulum to Bicycle trans-

fers are provided in the Supplementary. Learning curves showing the total reward averaged across three runs of each algorithm are provided in Figure-4.5. Adapt-to Learn policy transfer solved all the three tasks, yielding quicker learning compared to other baseline methods. These results provide empirical evidence of our hypothesis. Using trajectory KL divergence as intrinsic adaptation reward to adapt source policy to the target, we achieve a more robust and sample efficient policy transfer between two tasks, compared to using a warm-start or standalone RL method. Note that the target domain perturbations introduced are significant enough such that source policy alone without any adaptation in the target domain produced no meaningful results (Figure-4.5). This notion of adaptation in the face of uncertainty is a key advancement over traditional policy transfer, meta-learning, or adversarial RL methods.

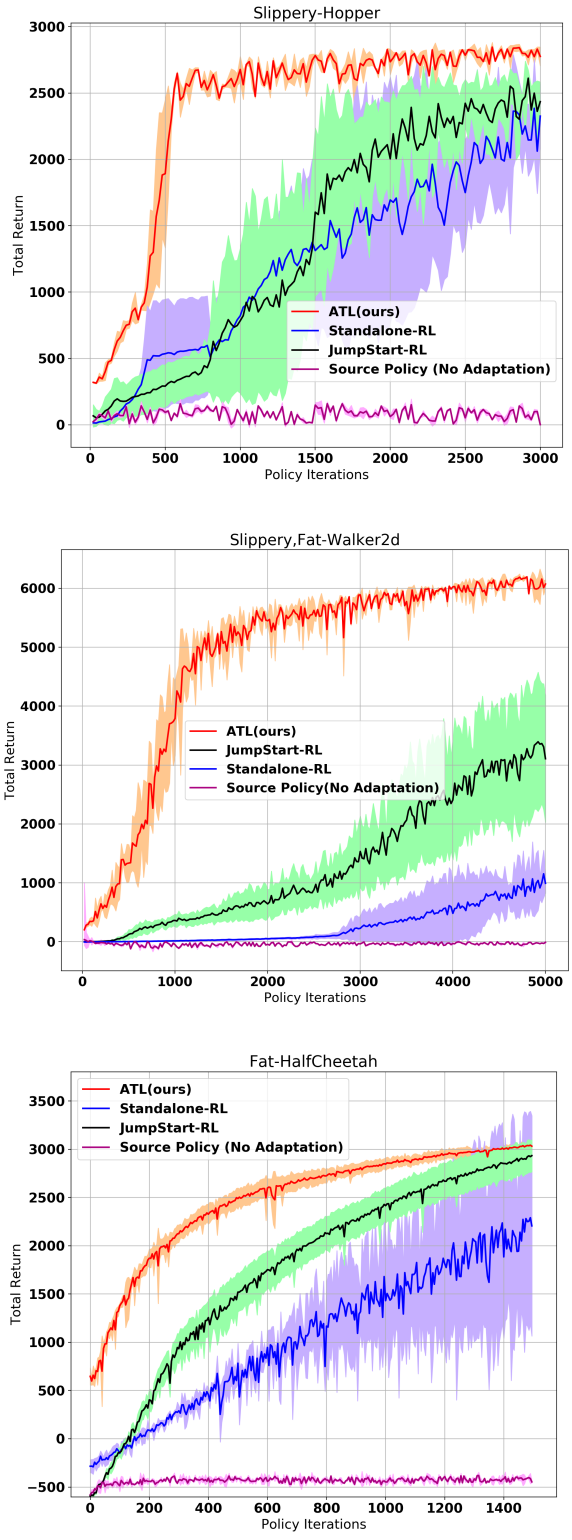


Figure 4.5: Learning curves for locomotion tasks, averaged across three runs of each algorithm: Adapt-to-Learn(Ours), Randomly Initialized RL(PPO), Warm-Started PPO using source policy parameters and Best case imitation learning using Source policy directly on Target Task without any adaptation.

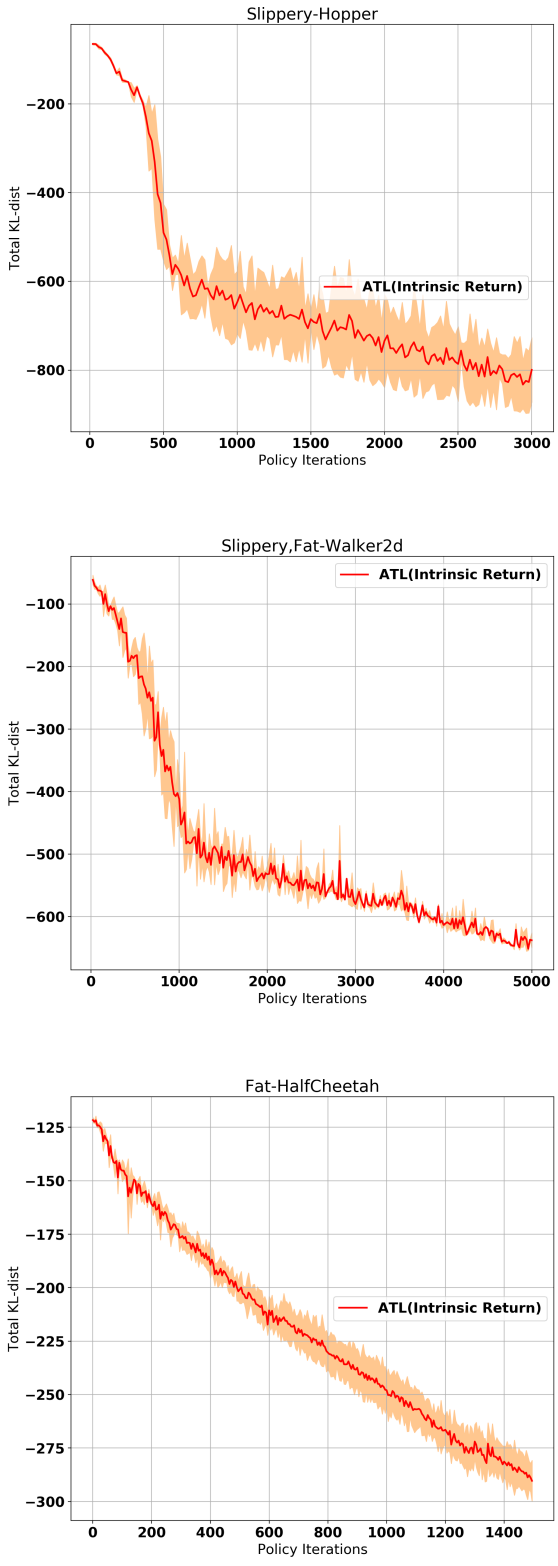


Figure 4.6: Trajectory KL divergence Total Intrinsic Return $\left(-\sum e^{\zeta_t}\right)$ averaged across three runs.

CHAPTER 5

MODEL REFERENCE ADAPTIVE CONTROL

5.1 Introduction

Model Reference Adaptive Control (MRAC) is a leading method for adaptive control that seeks to learn a high-performance control policy in the presence of significant model uncertainties [120, 34, 121]. MRAC has been widely utilized in flight control, for example in [122, 123, 124, 125, 126]. The key idea in MRAC is to find an update law for a parametric model of the uncertainty that ensures that the candidate Lyapunov function is non-increasing. Many update laws have been proposed and analyzed, which include but not limited to σ-modification [48], e-modification [127], and projection based updates [121]. More modern laws extending the classical parametric setting include ℓ_1 adaptive control [128], DF-MRAC [129], and concurrent learning [130] have also been studied.

An another MRAC approach introduced by Chowdhary et.al is the Gaussian Process Model Reference Adaptive Control (GP-MRAC), which utilizes a GP as a model of the uncertainty. A GP is a Bayesian nonparametric adaptive element that can adapt both its weights and the structure of the model in response to the data. The authors and others have shown that GP-MRAC has strong long-term learning properties as well as high control performance [53, 131]. However, GPs can be viewed as "shallow" machine learning models, and do not utilize the power of learning complex features through compositions as deep networks do (see 5.3.3). Hence, one wonders whether the power of deep learning could lead to even more powerful learning based MRAC architectures than those utilizing GPs.

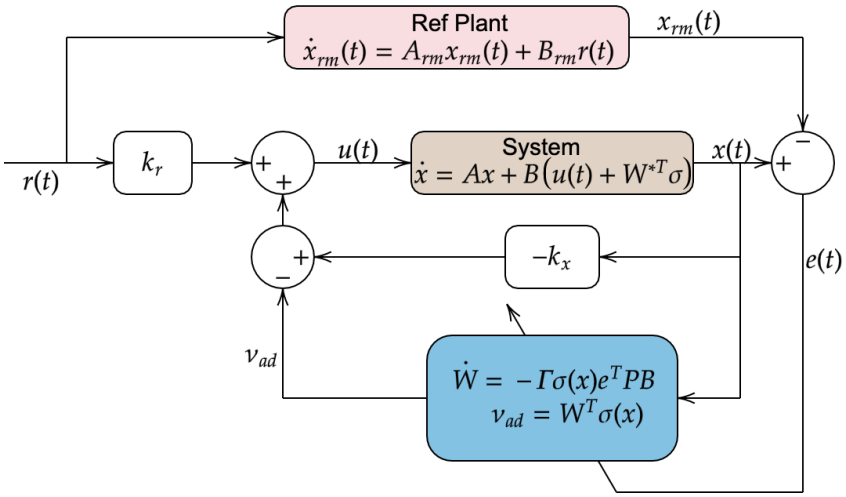


Figure 5.1: Model Reference Adaptive Control Flow Diagram

5.2 Preliminaries

This section discusses the formulation of model reference adaptive control (see e.g. [120, 34]). We consider the following system with uncertainty $\Delta(x)$:

$$\dot{x}(t) = Ax(t) + B(u(t) + f(x)) \tag{5.1}$$

where $x(t) \in \mathbb{R}^n$, $t \geqslant 0$ is the state vector, $u(t) \in \mathbb{R}^m$, $t \geqslant 0$ is the control input, $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$ are known system matrices and we assume the pair (A, B) is controllable. The term $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ is the matched uncertainty term and is assumed to be bounded such that $\|f(x)\|_\infty < B$ and Lipschitz continuous $\|f(x) - f(y)\| \leq L\|x - y\|$ in a compact set $x(t) \in \mathcal{D}_x$, where $\mathcal{D}_x \subset \mathbb{R}^n$ be a compact set. The controller $u(t)$ is assumed to belong to a set of admissible control inputs of measurable and bounded functions, ensuring the existence and uniqueness of the solution to (5.1).

The reference model is assumed to be linear and therefore the desired transient and steady-state performance is defined by a selecting the system eigenvalues in the negative half plane. The desired closed-loop response of the reference system is given by

$$\dot{x}_{rm}(t) = A_{rm}x_{rm}(t) + B_{rm}r(t) \tag{5.2}$$

where $x_{rm}(t) \in \mathcal{D}_x \subset \mathbb{R}^n$ and $A_{rm} \in \mathbb{R}^{n\times n}$ is Hurwitz and $B_{rm} \in \mathbb{R}^{n\times r}$. Furthermore, the command $r(t) \in \mathbb{R}^r$ denotes a bounded, piecewise continuous, reference signal and we assume the reference model (5.2) is bounded input-bounded output (BIBO) stable [120].

The aim is to construct a feedback law $u(t)$, $t \geqslant 0$, such that the state of the uncertain dynamical system (5.1) asymptotically tracks the state of the reference plant model (5.2) despite presence of matched uncertainty.

A tracking control law consisting of linear feedback term $u_{pd} = k_x x(t)$, a linear feed-forward term $u_{crm} = k_r r(t)$ and an adaptive term $u_{ad}(t)$ form the total controller

$$u(t) = u_{pd}(t) + u_{crm}(t) - u_{ad}(t) \tag{5.3}$$

The baseline full state feedback and feed-forward controller is designed to satisfy the matching conditions such that

$$A_{rm} = A - Bk_x \tag{5.4}$$

$$B_{rm} = Bk_r \tag{5.5}$$

The reference model tracking error is defined as

$$e(t) = x_{rm}(t) - x(t) \tag{5.6}$$

Using (5.1) & (5.2) and the controller of form (5.3) with adaptation term $u_{ad}(t)$, the tracking error dynamics can be written as

$$\dot{e}(t) = \dot{x}_{rm}(t) - x(t) \tag{5.7}$$

$$\dot{e}(t) = A_{rm}x_{rm}(t) + B_{rm}r(t) - Ax(t) - B\left(u(t) + f(x)\right) \tag{5.8}$$

Assuming the appropriate feedback $u_{pd} = k_x x(t)$ and feed-forward term $u_{crm} = k_r r(t)$ exists such that the above matching conditions (5.5)-(5.5) are met. We can write the error dynamics as follows,

$$\dot{e}(t) = A_{rm}x_{rm}(t) + B_{rm}r(t) - Ax(t) - B\left(k_x x(t) + k_r r(t) - u_{ad}(t) + f(x)\right) \tag{5.9}$$

$$\dot{e}(t) = A_{rm}x_{rm}(t) + B_{rm}r(t) - \left(A - Bk\right)x(t) - Bk_r t(t) - B\left(f(x) - u_{ad}(t)\right) \tag{5.10}$$

Using the matching condition we can write the error dynamics as,

$$\dot{e}(t) = A_{rm}e(t) + B\left(f(x) - u_{ad}(t)\right) \tag{5.11}$$

5.2.1 Structured Uncertainty

In next two subsection we will study the two possible characterization of the uncertainty term as Structured and Unstructured uncertainty.

Consider the case where the structure of the uncertainty $f(x)$ is known, that is, we known that the uncertainty can be represented as a linear combination of a known continuously differentiable basis function as follows,

Assumption 5.2.1 *The uncertainty $f(x)$ can be linearly parameterized, that is, there exist a unique constant vector $W^* \in \mathbb{R}^{k\times m}$ and a vector of known continuously differentiable basis functions $\sigma(x) = [\sigma_1(x), \sigma_2(x), ..., \sigma_m(x)]$, such that $f(x)$ can be uniquely represented as*

$$f(x) = W^{*T}\sigma(x(t)) \tag{5.12}$$

where $\sigma(x) : \mathbb{R}^n \to \mathbb{R}^k$ is a $k \times 1$ dimensional vector of known basis function, while $W^* \in \mathbb{R}^{k\times m}$ is the matrix of unknown parameters.

A large class of nonlinear uncertainties can be written in the above form. Note that the requirement of existence of unique W^* for a given basis of the uncertainty $\sigma(x(t))$ ensures that the representation of uncertainty is minimal.

The system dynamics with structured uncertainty can be written as,

$$\dot{x}(t) = Ax(t) + B(u(t) + W^{*T}\sigma(x)) \tag{5.13}$$

5.2.2 Unstructured Uncertainty

In the more general case where it is only known that the uncertainty $f(x)$ is bounded, non-destabilizing and continuously differentiable and defined over a compact domain $x(t) \in \mathcal{D}_x \subset \mathbb{R}^n$, the adaptive part of the control law can be formed using Neural Networks (NNs). The uncertainty $f(x)$ is represented as linear combination of designed basis functions and unknown true parameters as follows,

Assumption 5.2.2 *The uncertainty $f(x)$ can be linearly parameterized, that is, there exist a unique constant vector $W^* \in \mathbb{R}^{k\times m}$ and a vector of designed Lipschitz continuous bounded basis function $\Phi(x) = [\phi_1(x), \phi_2(x), ..., \phi_m(x)]$, such that $f(x)$ can be uniquely represented as*

$$f(x) = W^{*T}\Phi(x) + \tilde{\epsilon}(x) \tag{5.14}$$

where $\Phi(x) : \mathbb{R}^n \to \mathbb{R}^k$ is a $k\times 1$ dimensional vector of designed basis function, such that $\Phi(x)$ is bounded on a compact domain $x(t) \in \mathcal{D}_x$ while $W^* \in \mathbb{R}^{k\times m}$ is the matrix of unknown parameters, such that $\|W^*\|_\infty \leq \mathcal{W}_b$. The term $\tilde{\epsilon}(x)$ is the network approximation error due finite parameterization and can be bounded as follow

$$\bar{\epsilon} = \sup_{x\in\mathcal{D}_x} \|\tilde{\epsilon}(x)\| \tag{5.15}$$

In the following section we will present more general approach of uncertainty representation using Neural networks for capturing unstructured uncertainty.

5.3 NN model for Uncertainty Estimation

5.3.1 Radial Basis Function Neural Network

The Radial Basis Functions (RBF) are bell shaped gaussian activations. The argument of the activation function of the hidden layer units represents the Euclidean norm between the input vector and the units' center position. This operation characterizes the exponentially decaying localized nonlinearity of Gaussian functions. The expression for RBF can be written as

$$\phi_i(x, c_i, \sigma_i) \triangleq \exp\left(-\frac{\|x - c_i\|^2}{\sigma_i^2}\right) \tag{5.16}$$

Where c_i are the RBF centroid and σ_i are the bandwidth or the standard deviation of the Gaussian activation.

The output of adaptive element modelled as RBF network, which is essentially mixture of gaussians can be written as follows,

$$\Delta(x) = W^{*T}\Phi(x, \mathbf{c}, \boldsymbol{\sigma}) + \tilde{\epsilon}(x) \tag{5.17}$$

Where $W^* \in \mathbb{R}^{m\times k}$ are ideal weights, $\tilde{\epsilon}(x) : \mathbb{R}^n \to \mathbb{R}^m$ is network approximation error and $\Phi(x) = [1, \phi_1(x, c_1, \sigma_1), \phi_2(x, c_2, \sigma_2), \ldots, \phi_k(x, c_k, \sigma_k)] \in \mathbb{R}^k$ is a vector of known radial basis functions and $\mathbf{c} \in \mathbb{R}^{n\times k}$ and $\boldsymbol{\sigma} \in \mathbb{R}^k$ is vector of centers and standard deviations.

Even though the RBF network is a linear parameterized network, RBF networks can uniformly approximate continuous functions to arbitrary accuracy on a compact domain provided a sufficient number of Gaussian activations are used. The design of RBF network requires a priory knowledge of the domain of operation. This local approximation property of RBF networks is considered one of the reason for RBF being unpopular in function approximation over unknown support.

5.3.2 Single Hidden Layer Neural Network

A Single Hidden Layer (SHL) NN is a non-linearly parameterized map that has also been often used for capturing unstructured uncertainties that are known to be continuous. The input–output map of a SHL network can be represented as

$$y_k = b_w\theta_{wk} + \sum_{j=1}^{n_3} (W_{jk}\Phi_j(z_j)) \tag{5.18}$$

where $k = 1, 2, \ldots n_3$, b_w is the outer layer bias, θ_{wk} is the k^{th} threshold, W_{jk} represents the outer layer weights and Φ_j is the sigmoidal activation functions. The Inner layer output can be written as,

$$\Phi_j(z_j) = \sigma\left(z_j\right) \tag{5.19}$$

With sigmoid activation defined as,

$$\sigma(z) = \frac{1}{1 + e^{-az_j}} \tag{5.20}$$

where, a is the activation potential which can be a distinct value for each neuron. The value z_j is the input to the j^{th} hidden layer neuron, and is given by

$$z_j = b_v\theta_{vj} + \sum_{j=1}^{n_1} (V_{ij}x_i) \tag{5.21}$$

Here $n1$, $n2$ and $n3$ are respectively the number of input and hidden layer neurons, and number of outputs respectively, b_v is the inner layer bias and θ_{vj} is the j^{th} threshold.

Hence in a similar fashion to RBF NN we have that the following approximation holds for all $x \in \mathcal{D}_x \subset \mathbb{R}^n$ where $\mathcal{D}_x$ is compact.

$$f(x) = W^{*T}\Phi(V^{*T}x) + \tilde{\epsilon}(x) \tag{5.22}$$

Where (W^*, V^*) are ideal set of weights that approximates the given function to within an ϵ neighborhood of the function approximation error.

$$\tilde{\epsilon}(x) = \left\| f(x) - W^{*T}\Phi(V^{*T}x) \right\| \tag{5.23}$$

The largest such error $\tilde{\epsilon}(x)$ is given by

$$\bar{\epsilon} = \sup_{x \in \mathcal{D}_x} \left\| f(x) - W^{*T}\Phi(V^{*T}x) \right\| \tag{5.24}$$

And $\bar{\epsilon} = \sup_{x \in \mathcal{D}_x} \|\tilde{\epsilon}(x)\|$ can be made arbitrarily small given sufficient number of hidden layer neurons.

5.3.3 Deep Networks and Feature spaces in Machine Learning

The key idea in machine learning is that a given function can be encoded with weighted combinations of *feature* vector $\Phi \in \mathcal{F}$, s.t

$$\Phi(x) = [\phi_1(x), \phi_2(x), ..., \phi_k(x)]^T \in \mathbb{R}^k$$

Let $W^* \in \mathbb{R}^{k \times m}$ a vector of 'ideal' weights s.t $\|y(x) - W^{*^T}\Phi(x)\|_\infty < \epsilon(x)$. Instead of hand picking features, or relying on polynomials, Fourier basis functions, or comparison-type features used in support vector machines [132, 133] and Gaussian Processes [134], DNNs utilize composite functions of features arranged in a directed acyclic graphs, i.e.

$$\Phi(x) = \phi_n(\theta_{n-1}, \phi_{n-1}(\theta_{n-2}, \phi_{n-2}(...)))) \tag{5.25}$$

where θ_i's are the layer weights. The universal approximation property of the DNN with commonly used feature functions such as sigmoidal, tanh,

and RELU is proved in the work by Hornik's [135] and shown empirically to be true by recent results [136, 137, 138]. Hornik et al. argued the network with at least one hidden layer (also called Single Hidden Layer (SHL) network) to be a universal approximator. However, empirical results show that the networks with more hidden layers, show better generalization capability in approximating complex function. While the theoretical reasons behind better generalization ability of DNN are still being investigated [139], for our purpose, we will assume that it is indeed true, and focus our efforts on designing a practical and stable control scheme using DNNs.

5.4 Universal Approximation Theorem

The dominant theme of the second part of this thesis is the use of Deep Neural Networks to approximate unknown functions of the dynamic states. In order to adopt the different NN models for online adaptation we must ensure that these models are effectively capable of uniformly approximating continuous function over predetermined compact sets.

The Universal Approximation Theorem (UAT) claims a standard multi-layer feed-forward neural network with single hidden layer containing finite number of neurons and arbitrary activation can universally approximate any function in $C(\mathbb{R}^n)$. Kurt Hornik [140] prooved in the paper that the UAT result is not property of the activation function but the mulit-layer feed-forward architectures of the neural networks. The activation function at the output layer is assumed to linear. Without loss of generality the following UAT theory is presented for the single output. Multi output case is a simple extension of the presented result. The UAT is stated as follows,

Theorem 5.4.1 *Let $\phi(.)$ be an arbitrary activation function. Let $X \subseteq \mathbb{R}^n$ and belong to a compact set. The space of continous function on X is defined as $C(X)$. Then $\forall f \in C(X)$ and $\forall \epsilon > 0$, $\exists n \in N$ a_{ij}, b_j and $w_i \in \mathbb{R}$ where $i \in \{1 \dots n\}$, $j \in \{1 \dots m\}$*

$$(A_n f)(x_1, \dots x_m) = \sum_{i}^{n} w_i \phi \left(\sum_{j}^{m} a_{ij} x_j + b_j \right) \tag{5.26}$$

where $(A_n f)$ is the approximation function of f and therefore UAT states

that

$$\|f - A_n f\| \leq \epsilon \tag{5.27}$$

It is necessary the network predicts simultaneously well on all the inputs in compact set. In that case closeness of two function can be measure by uniform distance as follows,

$$\|f - A_n f\| = \sup_{x \in C(\mathbb{R}^n} \|f(x) - A_n f(x)\| \leq \epsilon \tag{5.28}$$

The detailed proof is beyond the scope of this thesis, but a concise proof statement state that. For given topological space Ω and any function $f : \mathbb{R}^n \to \mathbb{R}$. A neural network $A_n f$ is said to universal approximator if $A_n f$ is dense in $C(\Omega)$, the set of continuous functions from Ω to $\mathbb{R}$. For the detailed proof please refer [140, 141].

We will further discuss the network parameter update law for MRAC and necessary condition for the desired convergence in reference tracking error and network parameters.

5.5 Online Parameter Estimation law

Since the the mapping $\Phi(x) \in \mathbb{R}^k$ is known, letting $W(t) \in \mathbb{R}^{k \times m}$ denote the estimate of W^*. The adaptive element in the adaptive controller can be written as,

$$u_{ad}(t) = W^T \Phi(x(t)) \tag{5.29}$$

Using the above definition of adaptive element we can rewrite the total controller as $u(t)$ (5.3) as

$$u(t) = -k_x x(t) + k_r r(t) - W^T \Phi(x(t)) \tag{5.30}$$

Therefore the error dynamics (5.11) can be reduced to

$$\dot{e}(t) = A_{rm} e(t) + B \left(\tilde{W} \Phi(x) + \tilde{\epsilon}(x) \right) \tag{5.31}$$

where $\tilde{W} = W^* - W$ is error in parameter. The estimate to unknown true network parameters W^* are evaluated on-line using the weight update rule (5.32); correcting the weight estimates in the direction of minimizing the

instantaneous tracking error $e(t)$. The resulting update rule for network weights in estimating the total uncertainty in the system is as follows

$$\dot{\hat{W}} = \Gamma proj(\hat{W}, \Phi(x)e(t)'PB) \quad \hat{W}(0) = \hat{W}_0 \tag{5.32}$$

where $\Gamma \in \mathbb{R}^{k\times k}$ and $P \in \mathbb{R}^{n\times n}$ is a positive definite matrix. For given Hurwitz A_{rm} and $Q > 0$ the matrix $P \in \mathbb{R}^{n\times n}$ is a positive definite solution of Lyapunov equation

$$A_{rm}^T P + P A_{rm} + Q = 0 \tag{5.33}$$

Where "*proj*" is the projection operator, and ensure the weight updated according to (5.32) always lie within a compact set $\mathcal{W}_b$.

Let the compact set $\mathcal{W}_b$ be defined as,

$$\mathcal{W}_b \triangleq \{W_i \in \mathbb{R}^k | g(W_i) < c\}, 0 \leq c \leq 1 \forall i = 1, 2 \dots m \tag{5.34}$$

Where $g(W_i)$ is defined as follows,

$$g(W_i) = \frac{(1+\epsilon_w)W^T W - W_{max}^2}{\epsilon_W W_{max}^2} \tag{5.35}$$

The projection operator can be defined as

$$proj(W, y) \triangleq \begin{cases} y & \text{if } g(W) < 0 \\ y & \text{if } g(W) \geq 0 \text{ and } \nabla g^T y \leq 0 \\ y - \frac{\nabla g}{||\nabla g||}\left\langle \frac{\nabla g}{||\nabla g||}, y \right\rangle g(W) & \text{if } g(W) \geq 0 \text{ and } \nabla g^T y > 0 \end{cases}$$

where, $y = \Phi(x)e(t)'PB_i$. More details of projection operator can be found in [35].

5.6 Persistency of Excitation

Consider the problem of parameter estimation using a linear estimator for a function by minimizing its squared loss error. In parameter estimation problems such as flight system identification, the parameters to be estimated directly relate to meaningful physical quantities such as aerodynamic deriva-

tives. Hence, the convergence of the unknown parameters to their true values is highly desirable.

Let us assume the function to be estimated be $f : \mathbb{R}^n \to \mathbb{R}^m$. Let the true function f be of form

$$f(x) = W^{*T}\Phi(x) \tag{5.36}$$

Where the unknown function be linearly parameterized in unknown ideal weight $W^* \in \mathbb{R}^{k\times m}$ and $\Phi(x) : \mathbb{R}^n \to \mathbb{R}^k$ be a nonlinear continuously differentiable basis function.

Let the $W(t)$ be the online estimate of true ideal weights W^*, therfore the online estimate of $f(x)$ can be represented by mapping $\nu : \mathbb{R}^n \to \mathbb{R}^m$ such that,

$$\nu = W^T\Phi(x) \tag{5.37}$$

Lets consider $V(\epsilon_i) : \mathbb{R}^n \to \mathbb{R}$ denote the loss function to be minimized over the set of points $\{x_i\}_{i\in\mathcal{I}}$ to estimate the parameters,

$$\begin{aligned} V(\epsilon_i) &= \frac{1}{2}\sum_{i\in\mathcal{I}} \epsilon_i^T\epsilon_i & (5.38)\\ &= \frac{1}{2}\sum_{i\in\mathcal{I}} \left\|W^T\Phi(x_i) - f(x_i)\right\|_2^2 & (5.39)\end{aligned}$$

If we use the gradient descent algorithm with a fixed step size Γ to minimize V the update rule for W becomes

$$W \leftarrow W\left(I - \Gamma\sum_{i\in\mathcal{I}}\Phi(x_i)\Phi(x_i)^T\right) - \Gamma f(x_i)\Phi(x_i)^T \tag{5.40}$$

Using the expression for $f(x_i) = W^{*T}\Phi(x_i)$ and defining the parameter error $\tilde{W} = W - W^*$, we can rewrite the above update equation for $\tilde{W}$ as follows,

$$\tilde{W} \leftarrow \tilde{W}\left(I - \Gamma\sum_{i\in\mathcal{I}}\Phi(x_i)\Phi(x_i)^T\right) \tag{5.41}$$

which is a discrete-time linear time invariant system. Therefore, a necessary and sufficient condition for the convergence of the algorithm is that

$$\lambda_{max}\left(\sum_{i\in\mathcal{I}}\Phi(x_i)\Phi(x_i)^T\right) < \frac{2}{\lambda_{min}(\Gamma)} \tag{5.42}$$

If the $\Phi(x_i)$ follows the above condition, then we can show that $\tilde{W} \to 0$ as $t \to \infty$, i.e $W \to W^*$ as $t \to \infty$ and the condition (5.42) is known as persistency of excitation of $\Phi(x_i)$

For the case of adaptive control, Boyd and Sastry have shown that the condition on persistency of excitation in the system states $\Phi(x)$ can be related to persistency of excitation in the exogenous reference input $r(t)$ by noting the following: If the exogenous reference input $r(t)$ contains as many spectral lines as the number of unknown parameters, then the plant states are PE, and the parameter error converges exponentially to zero [142].

Definition 5.6.1 *The bounded signal $\Phi(t)$ is exciting over interval $[t, t+T]$,such that $T > 0$ and $t \geq t_0$, if there exists γ such that*

$$\int_t^{t+T} \Phi(\tau)\Phi(\tau)^T d\tau \geq \gamma I \tag{5.43}$$

Definition 5.6.2 *The bounded signal $\Phi(t)$ is persistently exciting over interval $[t, t+T]$,such that $T > 0$ and if for all $t \geq t_0$, if there exists γ such that*

$$\int_t^{t+T} \Phi(\tau)\Phi(\tau)^T d\tau \geq \gamma I \tag{5.44}$$

Note that the above definition requires that the matrix $\int_t^{t+T} \Phi(\tau)\Phi(\tau)^T \in \mathbb{R}^{k\times k}$ be positive definite over any finite interval.

Definition 5.6.3 *Autocovariance: A function $\Phi : \mathbb{R}^n \to \mathbb{R}^k$ is said to have a autocovariance $R_\phi(\tau) \in \mathbb{R}^{k\times k}$ iff*

$$\lim_{t\to\infty} \frac{1}{T} \int_t^{t+T} \Phi(t)\Phi(t+\tau)^T dt = R_\phi(\tau) \tag{5.45}$$

Lemma 5.6.4 *Suppose Φ as autocovariance $R_\phi(\tau)$. Then Φ is Persistently Exciting, iff $R_\phi(0) > 0$*

The proof this lemma can be found in [142].

5.7 Stability and Boundedness

We will introduce a general stability and boundedness proof for nonlinear system. In next section we use this theory to prove closed loop stability of

system under adaptive controllers. Our work eventually leads to multi-layer deep neural network as an online approximator of the uncertainty. With the assumption of unstructured uncertainty and multi-layer network, we will not be able to show asymptotic stability of the tracking, for this reason we precisely state the kind of boundedness we can ensure.

To state the stability and boundedness results we use a general, nonlinear dynamics as follows

$$\dot{x}(t) = f(t, x) \tag{5.46}$$

Definition 5.7.1

The solution of (5.46) are uniformly ultimately bounded (with bound B) if there exits a $\mathcal{B} > 0$ and for any α and $t_0 \in R^+$, there exists a time $T = T(\alpha) > 0$ independent of t_0 such that $\|x(t_0)\| < \alpha$ implies $\|x(t; t_0, x_0)\| < \mathcal{B}$ for all $t > t_0 + T$.

Theorem 5.7.2 *Suppose there exists a Lyapunov function $V(t, x) : \mathcal{D}_x \to \mathbb{R}$ defined on $0 \leq t < \infty$, such that*

$$\begin{aligned} V(t, 0) &= 0 \\ V(t, x) &> 0, x \in \mathcal{D}_x, x \neq 0 \end{aligned}$$

If there exists continuous functions such that, $\alpha(.), \beta(.) \in \mathcal{K}$, such that $V : \mathcal{D}_x \to \mathbb{R}$

$$\alpha(\|x\|) \leq V(t, x) \leq \beta(\|x\|) \forall x \in \mathcal{D} \tag{5.47}$$

$$\dot{V}(t, x) \leq \gamma(\|x\|) \tag{5.48}$$

where $\gamma(.)$ positive continuous function, then solution (5.46) *is uniformly ultimately bounded.*

5.7.1 Lyapunov Stability of MRAC

The on-line adaptive identification law (7.8) guarantees the asymptotic convergence of the observer tracking errors $e(t)$ and parameter error $\tilde{W}(t)$ under

the condition of persistency of excitation [55, 120] for the structured uncertainty. Under the assumption of unstructured uncertainty, we show tracking error is uniformly ultimately bounded (UUB).

Theorem 5.7.3 *Consider the actual and reference plant model (5.1) & (5.2). If the weights parameterizing total uncertainty in the system are updated according to identification law* (5.32) *Then the tracking error and error in network weights* $\|\tilde{e}\|$, $\|\tilde{W}\|$ *are bounded.*

Proof Let $V(e,\tilde{W}) > 0$ be a differentiable, positive definite radially unbounded Lyapunov candidate function,

$$V(e,\tilde{W}) = e^T P e + \frac{1}{2} tr\left(\tilde{W}^T \Gamma^{-1} \tilde{W}\right) \tag{5.49}$$

The time derivative of the lyapunov function (7.13) along the trajectory (7.7) can be evaluated as

$$\dot{V}(e,\tilde{W}) = \dot{e}^T P e + e^T P \dot{e} - \tilde{tr}\left(W^T \Gamma^{-1} \dot{W}\right) \tag{5.50}$$

Using (7.7) & (7.8) in (7.14), the time derivative of the lyanpunov function reduces to

$$\dot{V}(e,\tilde{W}) \;\; = \;\; -e^T Q e + 2 e^T P \epsilon(x) \tag{5.51}$$

Hence $\dot{V}(e,\tilde{W}) \leq 0$ outside compact neighborhood of the origin $e = 0$, for some sufficiently large $\lambda_{min}(Q)$.

$$\|e(t)\| \geq \frac{2\lambda_{max}(P)\bar{\epsilon}}{\lambda_{min}(Q)} \tag{5.52}$$

Using the BIBO assumption $x_{rm}(t)$ is bounded for bounded reference signal $r(t)$, thereby $x(t)$ remains bounded and hence $e(t)$ is bounded. Since $V(e,\tilde{W})$ is radially unbounded the result holds for all $x(0) \in \mathbb{R}^n$. Using the fact, the error in parameters $\tilde{W}$ are bounded through projection operator [143]. Using Lyapunov theory and Barbalat's Lemma [144] we can show that $e(t)$ is uniformly ultimately bounded in vicinity to zero solution. We note that, the second derivative of Lyapunov function is follows,

$$\ddot{V}(e,\tilde{W}) = -2\lambda_{min}(Q)(e(t)\dot{e}(t)) + 2\lambda_{max}(P)\bar{\epsilon}\dot{e}(t) \tag{5.53}$$

$\ddot{V}(e, \tilde{W})$ is bounded due the fact that $\tilde{W}$ is bounded through projection operator in weight update rule [48] and $\bar{\epsilon}$ is a constant, hence from Lyapunov theory and Barbalat's Lemma [144], we can state $\dot{V}(e, \tilde{W})$ is uniformly continuous hence $\dot{V}(e, \tilde{W}) \to 0$ as $t \to \infty$. Using the previous fact with lower bound on error (5.52) we show that $e(t)$ is uniformly ultimately bounded near to zero solution.

Theorem 5.7.4 *For system* (5.1) *with the total controller* (5.3) *and adaptive elements defined as* (5.32) *with reference system* (5.2)*, all signals in the closed-loop system are bounded.*

Proof Theorem 5.7.3 shows that the controlled system state $x(t)$ converges to the reference state $x_{rm}(t)$ in the steady-state. However, during the transient time, $x(t)$ may be far away from $x_{rm}(t)$ due to significant initial errors $e(0)$ and $W(0)$, which may lead to poor transient performance (e.g. convergence rate, smoothness). To show this point, we analyze the transient response of the classical MRAC system considering the norms of $e(t)$ and $W(t)$

We recall the candidate Lyapunov function (5.49) used to prove the system stability under adaptive control. Using the following property of Lyapunov function for stable systems $V(t, x, \tilde{W}) \leq V(t, x(0), \tilde{W}(0))$ we can write

$$\begin{aligned} \|e(t)\|_\infty &\leq \sqrt{\frac{v(t, x(t), \tilde{W}(t))}{\lambda_{min}(P)}} \leq \sqrt{\frac{v(0, x(0), \tilde{W}(0))}{\lambda_{min}(P)}} \\ &\leq \sqrt{\frac{\lambda_{max}(P)\|e\|_\infty^2 + \lambda_{max}\left(\Gamma^{-1}\right)\|\tilde{W}(0)\|_F^2}{\lambda_{min}(P)}} \end{aligned} \tag{5.54}$$

Where $\|\|_F$ is the Frobenius norm. The L_∞ norm of the tracking (5.54) depends on two terms, initial tracking error $e(0)$ and intial weight estimate error $\tilde{W}(0)$. The effects of $\tilde{W}(0)$ on transient error $\|e(t)\|_\infty$ can be reduced by choosing a higher learning gain Γ, i.e. a high gain adaptive law can rapidly suppress the effects of the initial estimation error $\tilde{W}(0)$ in the transient period.

However, a large, high-gain induced, learning rate causes undesirable high-frequency oscillations in adaptive systems, which may excite unmodeled dy-

namics and trigger instability. From (5.32) we can write

$$\begin{aligned}\int_0^T \left\|\dot{\hat{W}}\right\|^2 d\tau &\leq \lambda_{max}^2(\Gamma)\left\|\Phi PB\right\|_\infty^2 \int_0^T \|e\|^2 d\tau \\ &= \lambda_{max}^2(\Gamma)\left\|\Phi PB\right\|_\infty^2 \|e\|_2^2 \end{aligned} \tag{5.55}$$

Using the bound on $\|e(t)\|$ from above expression we can write,

$$\begin{aligned}\left\|\dot{\hat{W}}\right\|_\infty &\leq \sqrt{\lambda_{max}^2(\Gamma)\left\|\Phi PB\right\|_\infty^2 \|e\|_\infty^2} \\ &= \sqrt{\frac{\lambda_{max}^2(\Gamma)\left\|\Phi PB\right\|_\infty^2 \left(\lambda_{max}(P)\|e\|_\infty^2 + \lambda_{max}\left(\Gamma^{-1}\right)\|\tilde{W}(0)\|_F^2\right)}{\lambda_{min}(P)}}\end{aligned} \tag{5.56}$$

Higher learning gain leads to stiff differential equation which lead to instability in numerical integration. The analysis in (5.54) and (5.56) shows the well-known trade-off in the adaptive control design, i.e. the high frequency oscillations in the adaptive system can be reduced by choosing a small learning gain, while a small learning gain leads to sluggish tracking error convergence.

5.8 Evaluation of MRAC through simulation Using Wing-Rock System

In this section we evaluate the MRAC controller on Wing-Rock system [145, 146]. We will experiment with adaptive elements with different basis vector through numerical simulation on a wing rock dynamics model. Wing rock is an interesting phenomena which is caused due to asymmetric stalling on lifting surfaces of agile aircraft. Wing-rock phenomenon is observed on delta wing air crafts. Flying at very high angle of attack at low speeds the aircraft seems to rock in roll direction causing the oscillation to grow and leading to undesired handling characteristics to pilot control. If left uncontrolled, the oscillations caused by wing rock can easily grow unbounded and cause structural damage. Let ϕ denote the roll angle of an aircraft, p denote the roll rate, δ_a denote the aileron control input, then a simplified model for wing

rock dynamics is given by

$$\dot{\phi} = p \tag{5.57}$$

$$\dot{p} = \delta_a + \Delta(x) \tag{5.58}$$

Where $x = [\phi, p]$ is the system state. The system uncertainty is defined as

$$\Delta(x) = W_0 + W_1\phi + W_2 p + W_3|\phi|p + W_4|p|p + W_5\phi^3 \tag{5.59}$$

The original parameters for wing rock motion are adapted from [87], they are $W_0 = 0.0, W_1 = 0.2314, W_2 = 0.6918, W_3 = 0.6245, W_4 = 0.0095, W_5 = 0.0214$. However to make the uncertainty more aggressive and nonlinear, we increased the weights associated with nonlinear terms and added a heavy bias term. The modified true parameters are as follows $W_0 = 1.0, W_1 = 0.2314, W_2 = 0.6918, W_3 = 0.6245, W_4 = 0.1, W_5 = 0.214$. The task of the controller is to follow a reference step command. The reference model chosen is a stable second order linear system with natural frequency of $2 radian/second$ and damping ratio of 0.5. The linear control gains are given by $k = [-4, -2]$ and $k_r = -4$and the learning rate is set to $\Gamma = 100$. Initial conditions for the simulation are arbitrarily chosen to be $\phi = 1 deg$, $p = 1 deg/s$.

5.8.1 Structured Uncertainty

Consider first the case where the structure of the uncertainty is known. We assign the basis function using the known function form of $\Delta(x)$. Therefore the basis $\Phi(x)$ is selected as,

$$\Phi(x) = \left[1, \phi, p, |\phi|p, |p|p, \phi^3\right] \tag{5.60}$$

There fore the adaptive element estimating the uncertainty can be of the form $\nu_{ad} = W^T\Phi(x)$. Since we are assuming structured uncertainty case the true uncertainty can be expressed as $\Delta(x) = W^{*T}\Phi(x)$, where the ideal weights W^* are given in the Section-5.8

The structured uncertainty case has no much of hyper-parameter tuning and we also observe that the controller is robust under high learning gain.

The controller performance in approximating the true uncertainty is much superior compared to unstructured uncertainty case. However this form of controller require an additional information on the structure of uncertainty which might not be available in all cases.

Figure compares the reference model states with the plant states for the baseline adaptive law and Figure provide details of estimated vs true uncertainty in system dyanmics and Figure shows the evolution of the network weights.

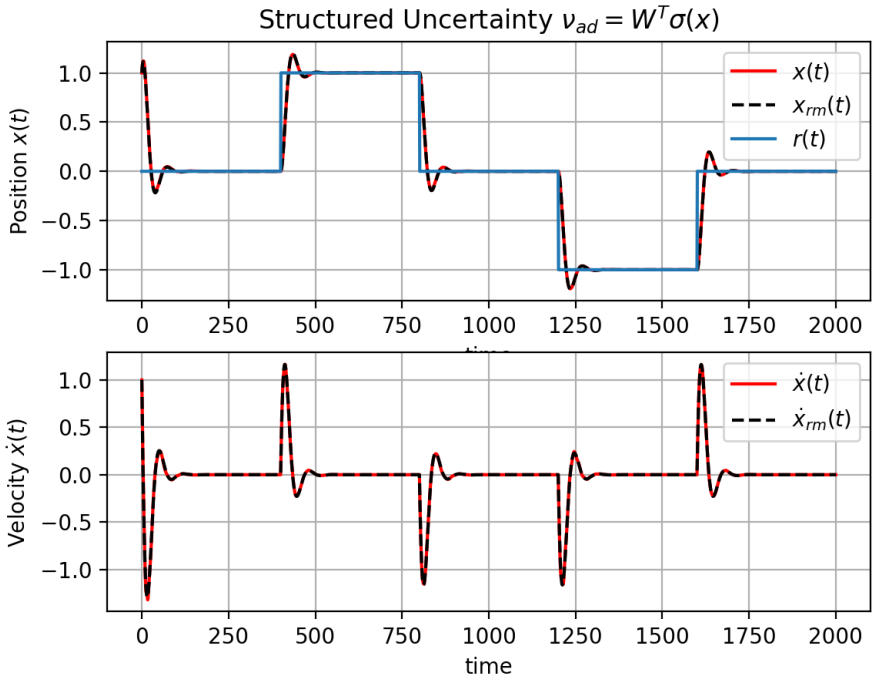


Figure 5.2: Structured uncertainty: position and velocity history under MRAC controller

5.8.2 Unstructured Uncertainty

For the results in this section we assume that it is only known that the structure of the uncertainty is unknown. In unstructured uncertainty case we study two basis function i.e linear in state model $W^T x(t)$ and RBF-NN model $W^T \Phi(x, c, \sigma)$.

Linear in state network is the most popular adaptive network in classical

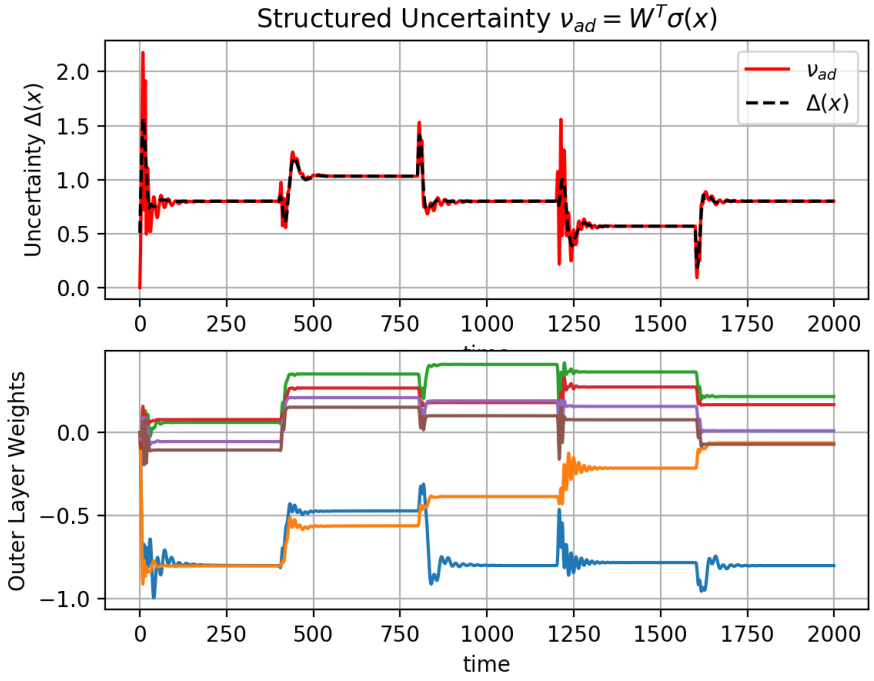


Figure 5.3: Structured uncertainty: Uncertainty estimation and Network weights history

adaptive control. The adaptive element is defined as

$$\nu_{ad} = W_1\phi + W_2 p \tag{5.61}$$

We will show that, a simple model of uncertainty such as $W^T x$ might some time under-represent the nonlinear uncertainty and hence fail to estimate the uncertainty faithfully and thereby lead to a poorly performing controller.

The other general feature vector we will study is RBF-NN. RBF-NN are deemed as universal approximators. RBF-NN are characterized by centers, number of centers and bandwidths of Gaussian features. We consider RBF NN with 10 nodes and uniformly distributed centers over the expected range of the state space $c \in [-1, 1]$ and bandwidth of $\sigma = 0.5$ are used to capture the model uncertainty. Since the ideal weights W in this case are not known, we evaluate the performance of the adaptive law by comparing the output of the RBF-NN with the actual model uncertainty with weights frozen after the simulation run is over.

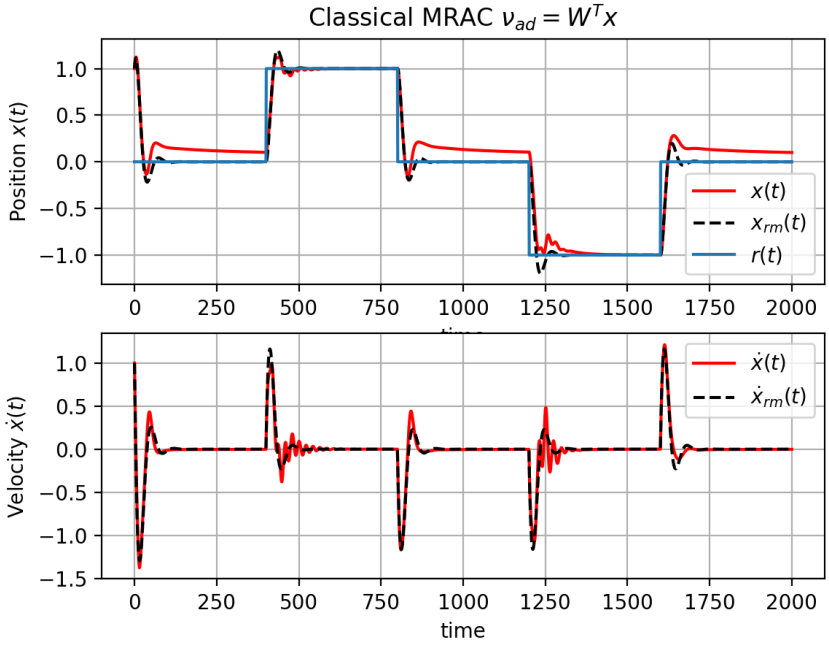


Figure 5.4: Unstructured uncertainty:Linear in state adaptive element-Position and Velocity history under MRAC controller

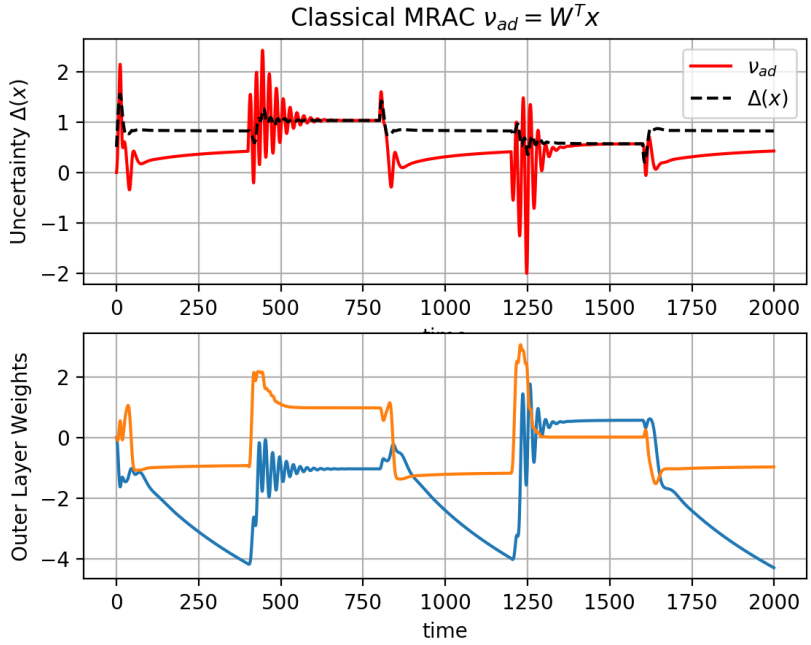


Figure 5.5: Unstructured uncertainty:Linear in state adaptive element-Uncertainty estimation and Network weights history

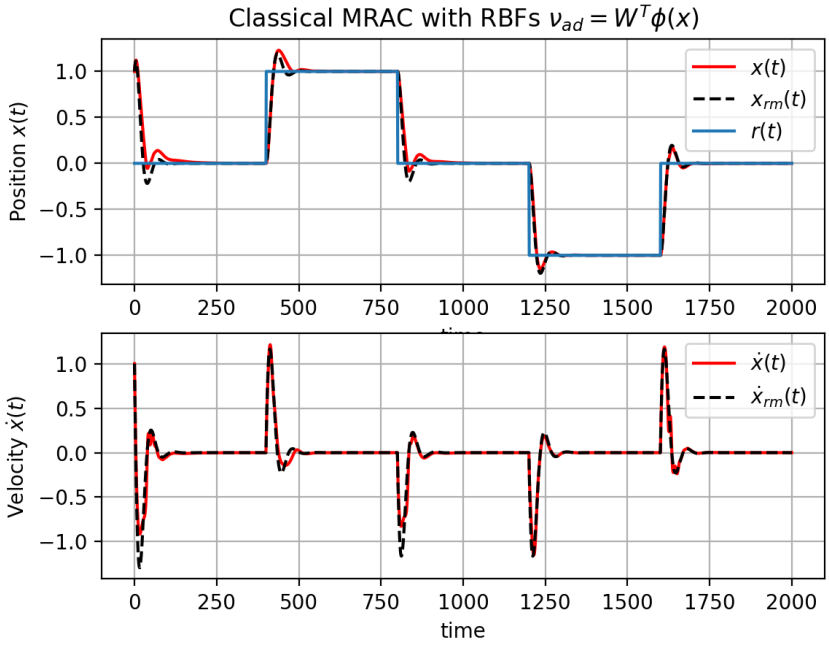


Figure 5.6: Unstructured uncertainty: RBF-NN-Position and Velocity history under MRAC controller

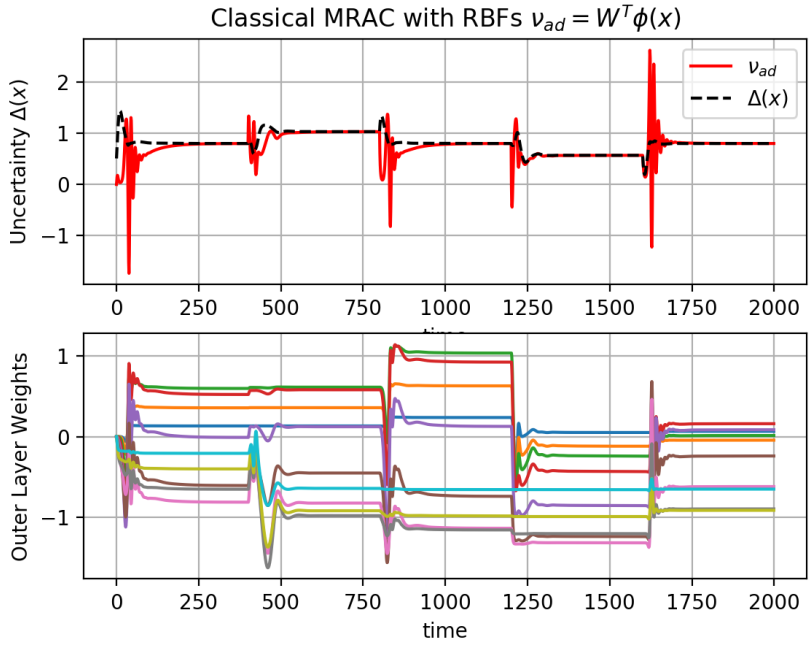


Figure 5.7: Unstructured uncertainty:Linear in state adaptive element-Uncertainty estimation and Network weights history

CHAPTER 6

GAUSSIAN PROCESS MODEL REFERENCE ADAPTIVE CONTROL

6.1 Introduction

In MRAC framework, radial basis function networks (RBFN) are quite widely used as universal-approximator in adaptive models [147][148]. RBFNs remain popular adaptive elements compared to multilayer networks [149, 47], due to their linear in parameter nature, which facilitates stability analysis of real-time controllers. However, the accuracy of RBFN representation greatly depends on pre-allocation of centers in a presumably known domain of operation of the system [150, 52, 151, 152]; this has been a major limitation of RBFN-MRAC [148]. On the contrary Gaussian Process (GP's) adaptive element in MRAC address this limitation of requiring to know the domain of operation by employing the GP nonparametric model to overcome the local approximation properties of RBFNs [134, 53].

In this chapter, we present a new approach to the on-line supervised training of GP models using a new architecture termed as Model Reference Generative Network (MRGeN). Our architecture is very loosely inspired by the recent success of generative neural network models [153], yet our contributions are in ensuring that inclusion of such a model in closed-loop control does not affect the stability properties. The generative network MRGeN is a neural network model for the system uncertainties, which generates (predicts) the labeled pair of state-uncertainties for GP inference. The MRGeN weights are updated such that network weights are moved in the direction of reducing the reference model tracking error [34, 55].

This work is inspired by previous works on GP-MRAC [53, 154]. GP-MRAC use system acceleration for generating noisy estimate of true model of uncertainty for inferring the GP model. This method has limitations since obtaining high fidelity acceleration information for many systems can be very

difficult due to noisy measurements or lack of sensors. Using MRGeN as a generative model for on-line GP inference has following advantages

1. The presented architecture obviates the necessity of system acceleration for implementation of GP-MRAC. Instead, it uses a fast-trained neural network to predict those values. We demonstrate that the inclusion of this network does not affect the stability properties.

2. Utilizing the GPs based controller [53], we retain the nonparametric nature of the controller by sharing the dynamic centers between GP and MRGeN update, ensuring the controller has global performance guarantees. The number of parameters and their properties are not required to be fixed apriori, rather they grow and adjust with data. We argue that within the class of nonparametric modeling methods, our presented approach leads to a very general data-driven generative models for Gaussian inference.

Another key feature of the presented architecture is the ability to control the rate of adaptation. Since the existing GP-MRAC uses $\dot{x}(t)$ information for training GP model, the rate of learning can only be as fast as the reference model. The original GP-MRAC work did not provide a mechanism for controlling the rate of adaptation. Since we use the MRGeN for generating target values, the rate of adaptation can be independently tuned through a separate adaptation gain value. This feature has the benefit that it can provide faster adaption when it is required to achieve stringent tracking performance specifications. However, fast adaptation using high-gain learning rates has been criticized for causing high-frequency oscillations in the control response, resulting potentially in system instability [155].

To alleviate this issue, we use the fact that GPs are known to have a deep connection with kernel filtering methods[156], in inherently handling noise in training data. Hence this property of GP's ensures that the nonparametric model learns the underlying noise-free model of uncertainty even when MRGeN target values are noisy, ensuring robustness and faster adaptation in the face of high gain learning rates.

6.2 Adaptive Control using GP-MRGeN

To address the issue of on-line supervised training of GP models for adaptive control application, we introduce GP-MRGeN. GP-MRGeN learns underlying smooth model to the system uncertainty using noisy estimate from high gain generative model. While ensuring higher learning rate, this architecture also smooths out high frequency components in the controller, preserving asymptotic stability to system error dynamics. This key feature of the presented framework ensures robustness while adapting faster to abrupt and large changes in system dynamics. The details of Gaussian processes and GP-MRGeN controller is presented in further sections

6.2.1 Gaussian Processes

A GP is defined as collection of random variables such that every finite subset is jointly Gaussian. That is, GP's are completely characterized by their second order statistics [134]. A GP is distribution over functions, i.e. a draw from the GP is a function. When a function $f(x)$ follows a GP model, we can define the term completely with the GP mean and variance as follows,

$$\Delta(.) \sim \mathcal{GP}(m(.), k(., .')) \tag{6.1}$$

where $m(.)$ is the mean of the function and $k(.,.)$ is positive definite, symmetric covariance kernel matrix.

Under GP regression, the mean is assumed to lie in the class of function $\mathcal{H}$, defined as reproducing kernel Hilbert space (RKHS). An RKHS is defined as function space such that $g \in \mathcal{H}$ satisfies following conditions, i.e. $\|g\|_{\mathcal{H}} < \infty$ and $\|g(.)\|^2_{\mathcal{H}} = \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \alpha_i \alpha_j k(z_i, z_j)$, interested readers can find further details on RKHS in [134, 53] and reference therein.

6.2.2 GP Regression

Let $Z_\tau = \{z_1, z_2, \dots z_\tau\}$ be set of discretely sampled state measurements. For each z_i's observed, there is an observed output $y(z_i) = m(z_i) + \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, \omega^2)$. The output y_i's can be stacked as a output vector $y = [y_1, y_2, \dots y_\tau]^T$. We can define the covariance matrix K such that $K_{i,j} =$

$k(z_i, z_j)$. The most common choice of covariance kernel and the one we use in this work is Gaussian RBF kernel defined as $k(z, z') = exp\left(-\frac{\|z-z'\|^2}{2\mu^2}\right)$, where μ is defined as bandwidth of the kernel.

The GP regression fuses RKHS theory and Bayesian linear regression by utilizing the regression model of the form $m(z) = \beta^T \Psi(z) = \sum_{i \in \mathcal{I}} \beta_i \langle \psi(z_i), \psi(z) \rangle$, where $\beta \in \mathcal{F}_Z$ is the vector of weights, belonging to a finite dimensional function space generated over Z such that $\mathcal{F}_Z \subset \mathcal{H}$. $\psi(z_i) \in \mathcal{H}$ is the mapping between data points Z_τ and linear subspace generated via data points.

GP Regression assumes that uncertainty in the data and model follow Gaussian distributions, while modeling the function using mean $\hat{m}(.)$ and a covariance function $\hat{\Sigma}$. Since the observations and the likelihood of the output y_i $p(y_\tau | Z_\tau, \beta)$ is Gaussian, we can set an initial Gaussian prior i.e. $p(\beta) \sim \mathcal{N}(0, \Sigma_\beta)$ and use Bayes' rule to to infer posterior distribution as $p(\beta | Z_\tau, y_\tau)$ with each new observation. Since the posterior is Gaussian, the update generates a revised prior with mean $\hat{m}_\tau$ and covariance $\hat{\Sigma}_\tau$ for next step of inference with new observation.

Given new input $z_{\tau+1}$, the joint distribution of the outputs available up to the time τ, i.e. y_τ and future data $y_{\tau+1}$ under the prior distribution can be written as

$$\begin{bmatrix} y_\tau \\ y_{\tau+1} \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(Z_\tau, Z_\tau) + \omega^2 I & k_{z_{\tau+1}} \\ k_{z_{\tau+1}}^T & k_{\tau+1}^* \end{bmatrix}\right) \tag{6.2}$$

where $k_{z_{\tau+1}} = K(z_{\tau+1}, Z_\tau)$ and $k_{\tau+1}^* = k(z_{\tau+1}, z_{\tau+1})$. The posterior distribution obtained by joint Gaussian prior distribution over observation $z_{\tau+1}$ is computed by

$$p(y_{\tau+1} | Z_\tau, y_\tau, z_{\tau+1}) \sim \mathcal{N}(\hat{m}_{\tau+1}, \hat{\sigma}_{\tau+1}) \tag{6.3}$$

where

$$\hat{m}_{\tau+1} = \beta_{\tau+1}^T k_{z_{\tau+1}} \tag{6.4}$$

$$\hat{\sigma}_{\tau+1} = k_{\tau+1}^* - k_{z_{\tau+1}}^T C_\tau k_{z_{\tau+1}} \tag{6.5}$$

are the updated mean and covariance estimates respectively. where $C_\tau := (K(Z_\tau, Z_\tau) + \omega^2 I)^{-1}$ and $\beta_{\tau+1} = C_\tau y_\tau$.

If $|Z_\tau|$ is finite the posterior mean and covariance is also finite. But since Z_τ and y_τ grow with data, computing the inverse becomes computationally intractable over time. In traditional offline GP regression this is less of a

problem but however, in online setting this linear growth of data poses a computational challenge. To mitigate this issue and use GP-MRGeN in online setting, we will use a simple budgeted kernel method to restrict the number of data points stored for inference. The test for incorporating a new data point into GP model through basis vector ($\mathcal{BV}$) is done by kernel independence test

$$\gamma_{\tau+1} = k^*_{\tau+1} - k^T_{z_{\tau+1}} \alpha_\tau \tag{6.6}$$

where $\alpha_\tau = K^{-1}_{z_\tau} k_{z_{\tau+1}}$. The term $\gamma_{\tau+1}$ characterizes the richness of the space spanned by basis vectors in estimating the target values. The details of qualifying a data point to be added to the budget using kernel independence test and test for removal of old point once the budget size is reached are given in [53]. The algorithm for GP-MRGeN adaptive control is provided in Algorithm-3.

6.2.3 GP-MRGeN Adaptive Controller

GP-MRGeN learns the posterior GP model of the system uncertainties over the MRGeN estimates of the uncertainty $f(x)$ as

$$u_{ad}(t) \sim \mathcal{GP}(\hat{m}(x), k(x, x')) \tag{6.7}$$

where $\hat{m}(x)$ is the estimate of the mean function updated using (6.4). We assume that the high gain MRGeN targets are distributed according to the Gaussian distribution. We use the fact that GP regression can mitigate the noise in the data and can still learn the true underlying model over high-frequency target estimate generated by MRGeN weight update law. This architecture ensures robustness with faster adaptation.

The GP model of the uncertainty is learned over the state measurement $Z_\tau = \{x_1, x_2, \dots x_\tau\}$ and corresponding MRGeN estimate of the uncertainty at each x_i's as the observed output $\mathcal{Y}_\tau = \{y_1, y_2, \dots y_\tau\}^T$. The target y_i are defined as $y_i = W^T \psi(x_i)$, where $\psi(x_i)$ are defined as $k(z_i, z_j) = \langle \psi(z_i), \psi(z_j) \rangle$ and parameter W is updated using the MRAC weight update law (5.32). The MRAC network estimating the target $y_i = W^T \psi(x_i)$ is known as Model Reference Generative Network (MRGeN).

Further using the mean and covariance update given in (6.4) & (6.5) a

smoothed posterior estimate of the true uncertainty is modeled using GP Bayesian inference. The adaptive element $u_{ad}(t)$ can be set equal to the mean of uncertainty i.e. $u_{ad}(t) = \hat{m}(x)$ or to an element drawn from the distribution (6.7). It is shown in the next section, that if the adaptive element is chosen as $u_{ad}(t) = \hat{m}(x)$, the approximation error $\|u_{ad} - f(x)\|$ is bounded and the total controller (5.3) is stable.

Algorithm 3 Gaussian Process Adaptive Control using MRGeN

Input: $\Gamma, \epsilon_{tol}, p_{max}$
while new measurements are available **do**
  Given $z_{\tau+1}$ compute $\gamma_{\tau+1} = k^*_{\tau+1} - k^T_{z_{\tau+1}}\alpha_\tau$.
  if $\gamma_{\tau+1} \geqslant \epsilon_{tol}$ **then**
    Update the weights of MRGeN using weight update rule (7.8)
    Compute $y_{\tau+1} = \hat{W}^T\phi(z_{\tau+1})$
    Add $(z_{\tau+1}, y_{\tau+1})$ to $\mathcal{BV}(\sigma)$
    if $|\mathcal{BV}(\sigma)| > p_{max}$ **then**
      Delete element in $\mathcal{BV}(\sigma)$ based on methods in [53]
    end if
    Increase the switching index σ
    Calculate $\hat{m}_{\tau+1}$ and $\hat{\Sigma}_{\tau+1}$
    $\nu_{ad} = \hat{m}_{\tau+1}$
    Evaluate the total control using (5.3)
  end if
end while=0

6.3 Analysis of Stability

In this section we introduce the stochastic stability theory for switched system. Consider the switched stochastic differential equation of Ito type whose solution are class of continues Markov process. The system of equations are

$$dx(t) = F(t, x(t))dt + G_\sigma(t, x(t))d\xi(t), \quad x(0) = x_0 \tag{6.8}$$

where $x \in \mathbb{R}^{n_x}$, $\xi(t) \in \mathbb{R}^{n_2}$ is Wiener process, $\sigma(t) \in \mathbb{N}$ is the switching index which switches finitely many times in any finite time interval. $F(t, x(t))$ is an n_s-vector function, and $G_\sigma(t, x(t))$ is an $n_s \times n_2$ matrix. Assume that $F(t, 0) = 0$ and $G_\sigma(t, 0) = 0$. Let functions $F(t, x(t))$, $G_\sigma(t, x(t))$ be Lipschitz

for each switching index σ, $\forall x, y$ such that

$$\|F(t,x) - F(t,y)\| + \|G_\sigma(t,x) - G_\sigma(t,y)\| \leq B\|x - y\| \tag{6.9}$$

Under the condition Lipschitz continuity the solution to (6.8) exists. Note the assumption of Lipschitz continuity on G_σ is reasonable for GP formulations, since the terms of G_σ are the differentiable kernel functions.

The following definitions concerning the ultimate boundedness of the solution of (6.8) are introduced.

Definition 6.3.1 *The process $x(t)$ is said to be mean square ultimately bounded uniformly in σ if there exists a positive constant K such that for all $t, x_0 \in \mathbb{R}^{n_s}$ and σ*

$$\lim_{t\to\infty} \mathbb{E}_{x_0}\|x(t)\|^2 \leq K \tag{6.10}$$

Definition 6.3.2 *The process $x(t)$ is said to be exponentially mean square ultimately bounded uniformly in σ if there exist positive constants K, c, and α such that for all $t \in \mathbb{R}^+, x_0 \in \mathbb{R}^{n_s}$, and for all σ*

$$\lim_{t\to\infty} \mathbb{E}_{x_0}\|x(t)\|^2 \leq K + c\|x_0\|^2 e^{\alpha(t)} \tag{6.11}$$

Theorem 6.3.3 *Let the $x(t)$ be the solution of (6.8) and let $V(t,x) : C^{1,2} \in \{\mathbb{R}^+, \mathbb{R}^{n_s}\}$. Lets define the Ito differential generator $\mathcal{L}$ for the smooth function $V(t,x)$ given by*

$$\mathcal{L}V(t,x(t)) = \frac{\partial V}{\partial t} + \sum_j F_j(t,x)\frac{\partial V}{\partial x} + \frac{1}{2}\sum_{i,j}\left[G_\sigma G_\sigma^T\right]_{i,j}\frac{\partial^2 V}{\partial x_i \partial x_j} \tag{6.12}$$

where $[G_\sigma G_\sigma^T]_{ij}$ is i^{th} and j^{th} column of $n_s \times n_s$ matrix $[G_\sigma G_\sigma^T]$. If $V(t,x)$ follows,

1. *$\alpha_1 + c_1\|x\|^2 \leq V(t,x)$ for all $\alpha, c_1 > 0$ and $\mathcal{L}V(t,x(t)) \leq \beta_\sigma - c_2 V(t,x)$, for real β_σ and $c2 > 0$, and all switch states σ; then the process $x(t)$ is mean square ultimately bounded uniformly in σ*

2. *Additionally if $V(t,x) \leq c_3\|x\|^2 + \alpha^2$, then the process x(t) is exponentially mean square ultimately bounded uniformly in σ.*

The proof of the above theorem can be found in [53, 157]

6.4 Stability and Boundedness results for GP-MRGeN

This section provides the stability proof for GP-MRGeN using Algorithm-3. Let $\sigma(t) \in \mathbb{N}$ be the switch index which is updated every time the basis vector set $\mathcal{BV}$ is updated. When the σth set is active the mean function is estimated using the $\mathcal{BV}(\sigma)$ basis vectors, and the corresponding mean is denoted by $\hat{m}^\sigma$. The analysis presented uses the uncertainty modeled as

$$f(z(t)) = m(z(t)) + G_\sigma(t, z(t))d\xi(t) \tag{6.13}$$

where ξ is zero mean Weiner process, G_σ is linear operator associated with kernel $k(z, z')$. Given system (5.1) and reference system (5.2) and for given switch index σ under the control (5.3), the error dynamics can be written as

$$de(t) = A_{rm}e(t)dt + dtB(\epsilon_m^\sigma(t) - G_\sigma(t, z(t))d\xi(t)) \tag{6.14}$$

The above tracking error is achieved using the controller of the form (5.3) and adaptive element is realization of the GP posterior as follows $u_{ad}(z) \sim \mathcal{GP}(\hat{m}^\sigma(z), k(z, z'))$. The error term is $\epsilon_m^\sigma = \hat{m}^\sigma(z) - f(z)$. For the sake of brevity, we drop the time and state dependency of $G_\sigma(t, z(t))d\xi(t)$ in the remaining section.

To prove stability we need to show boundedness of the approximation error $\|\hat{m}^\sigma(z) - f(z)\|$ and will we need the following result, which we will state without the proof.

Theorem 6.4.1 *(Dudley) Lets define a psuedometric,*

$$d_G(t, s) = \sqrt{\mathbb{E}\left[\|G_\sigma d\xi(t) - G_\sigma d\xi(s)\|\right]^2} \tag{6.15}$$

on T. let $N(T, d, \delta)$ be the $\delta-$covering number of the space. The $\delta-$entropy of the space (T, d) is given by $H(T, d, \delta) = logN(T, d, \delta)$. Let $D(T)$ be the diameter of space T with respect to following metric d_G, then following bound holds,

$$\mathbb{E}\sup_{t\in T} G_\sigma d\xi(t) \leq C\int_0^{D(T)} H^{1/2}(T, d, \delta)d\delta \tag{6.16}$$

Proof The proof of the above theorem can be found in [53]

Let $K_{i,j} = k(c_{\vartheta_i}, c_{\vartheta_j})$ be the kernel associated with $\hat{m}^\sigma$ induced by quantization operator $\vartheta : \{1, \ldots, \tau\} \to \{1, \ldots, p_{max}\}$, such that $z_i \mapsto c_{\vartheta_i}$, where

$c_{\vartheta_i} \in \mathcal{D}_x$ are the set of chosen centers $\mathcal{BV}$. Let $\hat{m}^\sigma(z) = \alpha^T \mathbf{k}(c_\vartheta, z)$, where $\alpha = (\sigma_n^{-2}\mathbf{K} + \omega^2 I)^{-1} y$

Theorem 6.4.2 *Let $\hat{m}^\sigma$ and $\Delta(z)$ be defined as above, the MRGeN generated target y be upper bounded as $\|y\|_\infty \leq M^\sigma$ and λ be the minimum of eigen value of K^{-1}. Then the disturbance approximation using GP can be upper bounded as*

$$\|\epsilon_m^\sigma(z)\|_\infty \leq \frac{\sigma_n^{-1}\sqrt{\lambda}}{(\sigma_n^{-2}\lambda + \omega^2)} + \frac{\sigma_n^{-2}\omega^2 M^\sigma}{(\sigma_n^{-2}\lambda + \omega^2)} \tag{6.17}$$

Proof Define the estimate $\hat{m}(z) = (\sigma_n^{-2}\mathbf{K})^{-1} k_z y$. Lets begin with calculating the expectation of approximating term $\hat{m}^\sigma(z)$,

$$\begin{aligned} \mathbb{E}(\hat{m}^\sigma(z)) &= \mathbb{E}((\sigma_n^{-2}\mathbf{K} + \omega^2 I)^{-1} k_z y) \qquad (6.18)\\ &= \mathbb{E}((I + \omega^2(\sigma_n^{-2}\mathbf{K})^{-1})^{-1}(\sigma_n^{-2}\mathbf{K})^{-1} k_z y) \end{aligned}$$

Using the above definition of $\hat{m}(z)$ we can write,

$$\begin{aligned} \mathbb{E}(\hat{m}^\sigma(z)) &= \mathbb{E}((I + \omega^2(\sigma_n^{-2}\mathbf{K})^{-1})^{-1}\hat{m}(z)) \qquad (6.19)\\ &= (I + \omega^2(\sigma_n^{-2}\mathbf{K})^{-1})^{-1}\mathbb{E}(\hat{m}(z)) \qquad (6.20) \end{aligned}$$

Defining the term $\mathbf{W}_\lambda = (I + \omega^2(\sigma_n^{-2}\mathbf{K})^{-1})^{-1}$ we can write

$$\begin{aligned} \hat{m}^\sigma(z) &= \mathbf{W}_\lambda \hat{m}(z) \qquad (6.21)\\ \mathbb{E}(\hat{m}^\sigma(z)) &= \mathbf{W}_\lambda \mathbb{E}(\hat{m}(z)) \qquad (6.22) \end{aligned}$$

Using the previous arguments, lets calculate the mean square error $MSE(\hat{m}^\sigma(z))$, in step towards calculating the bounds on $\epsilon_m^\sigma(z)$

$$\begin{aligned} MSE(\hat{m}^\sigma(z)) &\triangleq \mathbb{E}((\hat{m}^\sigma(z) - \Delta(z))^T(\hat{m}^\sigma(z) - \Delta(z))) \qquad (6.23)\\ &= \mathbb{E}((\mathbf{W}_\lambda \hat{m}(z) - \Delta(z))^T(\mathbf{W}_\lambda \hat{m}(z) - \Delta(z))) \qquad (6.24)\\ &= \mathbb{E}((\hat{m}(z) - \Delta(z))^T \mathbf{W}_\lambda^T \mathbf{W}_\lambda(\hat{m}(z) - \Delta(z))) - \Delta(z)^T \mathbf{W}_\lambda^T \mathbf{W}_\lambda \Delta(z)\\ &\quad -\Delta(z)^T \mathbf{W}_\lambda \Delta(z) - \Delta(z)^T \mathbf{W}_\lambda^T \Delta(z) + \Delta(z)^T \Delta(z) \qquad (6.25) \end{aligned}$$

Simplifying and rearranging the terms we get,

$$\begin{aligned}
&\mathbb{E}((\hat{m}^{\sigma}(z)-\Delta(z)^{T}(\hat{m}^{\sigma}(z)-\Delta(z)) \\
&\quad = \mathbb{E}((\hat{m}(z)-\Delta(z))^{T}\mathbf{W}_{\lambda}^{T}\mathbf{W}_{\lambda}(\hat{m}(z)-\Delta(z))) \\
&\quad\quad +\Delta(z)^{T}(\mathbf{W}_{\lambda}-I_{p\times p})^{T}(\mathbf{W}_{\lambda}-I_{p\times p})\Delta(z) \qquad (6.26) \\
&\quad = \sigma_{n}^{2}\mathrm{tr}(\mathbf{W}_{\lambda}\mathbf{K}^{-1}\mathbf{W}_{\lambda}^{T})+\Delta(z)^{T}(\mathbf{W}_{\lambda}-I_{p\times p})^{T}(\mathbf{W}_{\lambda}-I_{p\times p})\Delta(z) \qquad (6.27)
\end{aligned}$$

Using the identity $\|\mathbf{x}\|_{\infty}\leqslant\|\mathbf{x}\|_{2}$ we can bound the term $\|\epsilon_{m}^{\sigma}(z)\|_{\infty}$ as

$$\begin{aligned}
\|\epsilon_{m}^{\sigma}(z)\|_{\infty} &\leqslant \sup_{z\in\mathcal{D}_{x}}\sqrt{\mathbb{E}((\hat{m}^{\sigma}(z)-\Delta(z)^{T}(\hat{m}^{\sigma}(z)-\Delta(z))} \\
&\leqslant \sup\sqrt{\sigma_{n}^{2}\mathrm{tr}(\mathbf{W}_{\lambda}\mathbf{K}^{-1}\mathbf{W}_{\lambda}^{T})}+\sup\sqrt{\Delta(z)^{T}(\mathbf{W}_{\lambda}-I_{p\times p})^{T}(\mathbf{W}_{\lambda}-I_{p\times p})\Delta(z)} \qquad (6.28)
\end{aligned}$$

Using the definition of $\mathbf{W}_{\lambda}$ and $\lambda=\lambda_{min}(\mathbf{K}^{-1})$ we evaluate each term,

$$\begin{aligned}
\sup\left(\sigma_{n}^{2}\mathrm{tr}(\mathbf{W}_{\lambda}\mathbf{K}^{-1}\mathbf{W}_{\lambda}^{T})\right) &= \frac{\sigma_{n}^{-2}\lambda_{min}(\mathbf{K}^{-1})}{\lambda_{min}(\sigma_{n}^{-2}\mathbf{K}+\omega^{2}I)} \qquad (6.29) \\
&= \frac{\sigma_{n}^{-2}\lambda}{(\sigma_{n}^{-2}\lambda+\omega^{2})^{2}} \qquad (6.30)
\end{aligned}$$

Using the bound $\Delta(z)\leq y\leq M^{\sigma}$ the second term can be simplified as follows,

$$\begin{aligned}
&\sup\left(\Delta(z)^{T}(\mathbf{W}_{\lambda}-I_{p\times p})^{T}(\mathbf{W}_{\lambda}-I_{p\times p})\Delta(z)\right)= \\
&(M^{\sigma})^{2}\left(\frac{\sigma_{n}^{-2}}{(\sigma_{n}^{-2}\lambda+\omega^{2})}-1\right)^{2}\leq(M^{\sigma})^{2}\left(\frac{\sigma_{n}^{-2}\omega^{2}}{(\sigma_{n}^{-2}\lambda+\omega^{2})}\right)^{2}
\end{aligned}$$

Using above expressions the total bound can be written as,

$$\|\epsilon_{m}^{\sigma}(z)\|_{\infty}\leq\frac{\sigma_{n}^{-1}\sqrt{\lambda}}{(\sigma_{n}^{-2}\lambda+\omega^{2})}+\frac{\sigma_{n}^{-2}\omega^{2}M^{\sigma}}{(\sigma_{n}^{-2}\lambda+\omega^{2})} \qquad (6.31)$$

Analyzing the bound we can notice the first term is square root of variance term and second is of the bias, which forms the source of total variation between true signal and its approximation.

The boundedness of tracking error can now be proven as

Theorem 6.4.3 *Consider the system in* (5.1)*, the controller (5.3), and as-*

sume that the uncertainty $\|f(z)\| \leq B$ is bounded and Lipschitz continuous in closed compact domain $z(t) \in \mathcal{D}_x$ and is representable by a GP. Then, Algorithm 1 and the adaptive signal $u_{ad}(z) = \hat{m}^\sigma(z)$ guarantee that the system is mean square uniformly ultimately bounded a.s.

Proof Let $V(e(t)) = 1/2e^T(t)Pe(t)$ be the stochastic Lyapunov candidate, where $P > 0$ satisfies the Lyapunov equation. Note that the Lyapunov candidate is bounded above and below by a quadratic term since $1/2\lambda_{min}(P)e^2 \leq V(e) \leq 1/2\lambda_{max}(P)e^2$. The Ito differential of the Lyapunov candidate along the solution of (6.14) for the σ^{th} system is,

$$\begin{aligned}
&\mathcal{L}V(e) \\
&= \sum_i \frac{\partial V(e)}{\partial e_i} Ae_i + \frac{1}{2}\sum_{i,j}[BG_\sigma(BG_\sigma)^T]_{ij}\frac{\partial^2 V(e)}{\partial e_i \partial e_j} \qquad (6.32)\\
&= \frac{1}{2}e^TQe + e^TPB[\epsilon_m^\sigma(z) + G_\sigma d\xi(t)] + \frac{1}{2}[BG_\sigma(BG_\sigma)^TP] \qquad (6.33)
\end{aligned}$$

Let $c_1 = 1/2\|P\|\|BG_\sigma\|^2$ and $c_2 = \|PB\|$, then

$$\begin{aligned}
&\mathcal{L}V(e) \\
&\leq -\frac{1}{2}\lambda_{min}(Q)\|e\|^2 + c_2\|e\|\left(\|\epsilon_m^\sigma(z) + G_\sigma d\xi(t)\|\right) + c_1 \\
&\leq -\frac{1}{2}\lambda_{min}(Q)\|e\|^2 + c_2\|e\|\left(\|\epsilon_m^\sigma(z)\| + c'\right) + c_1 \qquad (6.34)
\end{aligned}$$

From Theorem-6.4.2 we have $\|\epsilon_m^\sigma(z)\| \leq c_3 + c_4 M^\sigma$, where

$$c_3 = \frac{\sigma_n^{-1}\sqrt{\lambda}}{(\sigma_n^{-2}\lambda + \omega^2)}, \quad c_4 = \frac{\sigma_n^{-2}\omega^2}{(\sigma_n^{-2}\lambda + \omega^2)}$$

therefore using this bound in above expression we can write,

$$\mathcal{L}V(e) \leq -\frac{1}{2}\lambda_{min}(Q)\|e\|^2 + c_2\|e\|\left(c_3 + c_4M^\sigma + c'\right) + c_1 \qquad (6.35)$$

Defining $c_5^\sigma = c_3 + c_4 M^\sigma$, therefore outside the set

$$\Theta^\sigma = \left\{ \|e\| \leq \frac{c_5^\sigma + \sqrt{(c_5^\sigma)^2 + 2 * \lambda_{min}(Q)c_1}}{\lambda_{min}(Q)} \right\} \qquad (6.36)$$

Therefore, $\mathcal{L}V(e) \leq 0$ a.s. Since M^σ depends on y we need to show that y is

bounded. Since we use MRGeN to generate y, this result will be straightforward. From Assumption-5.2.2 and projection operator in weight update rule (7.8), y is upper bounded as $y \leq \mathcal{W}_b^T \|\phi(x)\|_\infty$. Where $\mathcal{W}_b^T$ is boundary point in projection operator[48]. Therefore it follows that Θ^σ is bounded, hence $\|e\|$ is bounded.

Using BIBO stability, for bounded reference signal $r(t)$, $x_{rm}(t)$ remains bounded and therefore $x(t) \in \mathcal{D}_x$ a.s. Since this is true for all arbitrary switching σ, from stochastic stability theory [158] (6.14) is mean square UUB.

6.5 Simulations

In this section, we will evaluate the presented GP-MRGeN adaptive controller using wing rock aircraft dynamic model. Let θ denote the roll attitude of an aircraft, p denotes the roll rate and δ_a denotes the aileron input. The model for wing rock dynamics of a delta wing aircraft is

$$\dot{\theta} = p \tag{6.37}$$

$$\dot{p} = L_{\delta_a}\delta_a + \Delta(x) \tag{6.38}$$

where $L_{\delta_a} = 3$ and $\Delta(x)$ is model for wing rock dynamics and is assumed to be unknown to the controller. We assume a stochastic model to uncertainty $\Delta(x)$, with variance ω^2 and mean

$$\Delta(x) = W_0^* + W_1^*\theta + W_2^* p + W_3^*|\theta|pW_4^*|p|p + W_5^*\theta^3 \tag{6.39}$$

The parameters for the true mean function are motivated from [146] with $W_1^* = 0.2314$, $W_2^* = 0.6918$,$W_3^* = -0.6245$,$W_4^* = 0.0095$,$W_5^* = 0.0214$. In addition, a trim error is introduced by setting $W_0^* = 0.8$. The gain of feedback controller K_p and K_r are set to $K_p = [-16, -4]$ and $K_r = 16$ respectively to ensure the matching condition to the reference model. A second order reference model with natural frequency $4rad/s$ and damping ratio of 0.5 is used. Further stochasticity is added to the system by adding Gaussian white noise to the states with variance $\omega_n = 0.01$. The simulation uses time step of $0.05s$. The maximum number of points (p_{max}) to be stored in $\mathcal{BV}(\sigma)$ is arbitrarily set to 100 and oldest points (OP) were selected to be deleted from

$\mathcal{BV}(\sigma)$ when budget is reached, the details are provided in [53].

The controller is designed to track a stable reference commands $r(t)$. The goal of the experiment is to compare the tracking performance of GP-MRGeN controller with High Gain MRAC controller when the domain operation is unknown. We find that 100 centers are sufficient for a good on-line approximation of the uncertainty. We use the same reference model for both controllers. The learning rate for high gain MRAC and MRGeN generative network for GP training is chosen to be $\Gamma = 5000$, Kernel bandwidth is chosen as $\mu = 0.05$ and tolerance threshold for kernel independence test is selected to be $\epsilon_{tol} = 0.1$ for updating the basis vector $\mathcal{BV}(\sigma)$.

Figure-7.7 and Fig-7.8 show the closed loop system performance in tracking the reference signal for GP-MRGeN and MRAC controller. Note that the control response of the proposed adaptive controller is clearly superior as compared to the standard adaptive controller. GP-MRGeN uses the generative model target at an only small number of discrete time steps to learn the underlying model of the uncertainty. Also, the GP-MRGeN estimate is noise free even when the target values are noisy. The presented controller achieves tighter tracking with smaller tracking error as shown in Fig-7.9 compared to MRAC controller. Figure-6.5 show the total control profile for both the controllers. It can be observed the control history for GP-MRGeN has smaller peaks and less noisy due to smoothing property of the GP estimation. This denoising of the controller demonstrates that proposed method leads to robust adaptation without incurring high-frequency oscillations in the control response. Figure-6.5 shows the network performance in estimating the uncertainty. The GP-MRGeN estimate is noise free and closely approximates the true uncertainty compared to MRAC estimate. The blobs in Fig-6.5 show the time step at which the generative model MRGeN was updated and queried for inferring the posterior GP model over data.

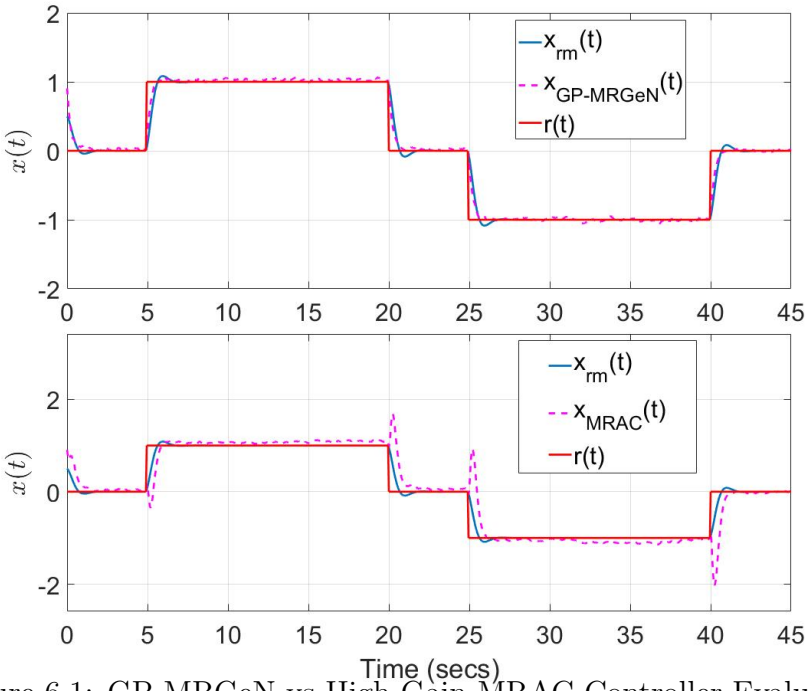


Figure 6.1: GP-MRGeN vs High Gain-MRAC Controller Evaluation on aircraft Wing-Rock dynamics model-Closed-loop system response in roll angle $\theta(t)$

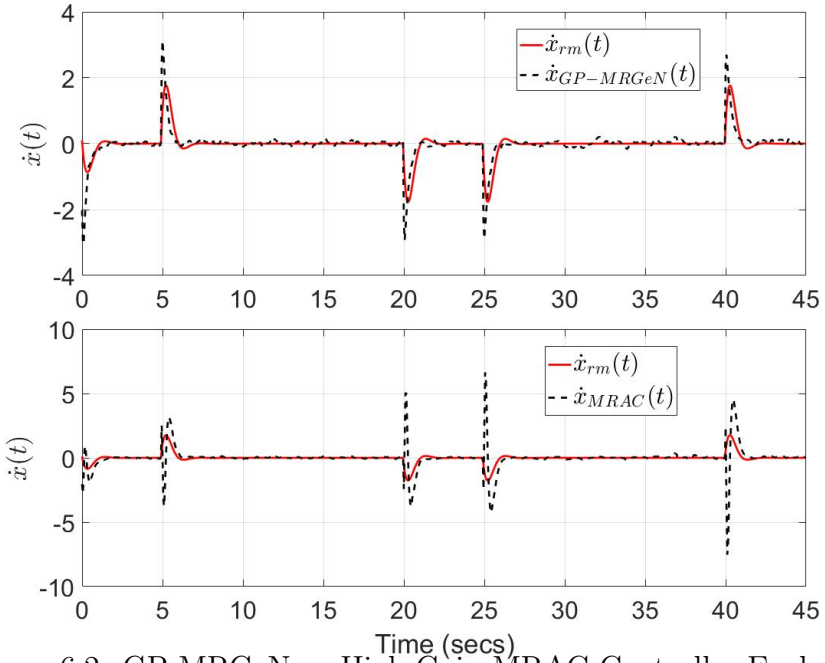


Figure 6.2: GP-MRGeN vs High Gain-MRAC Controller Evaluation on aircraft Wing-Rock dynamics model-Closed-loop system response in roll rate $p(t)$

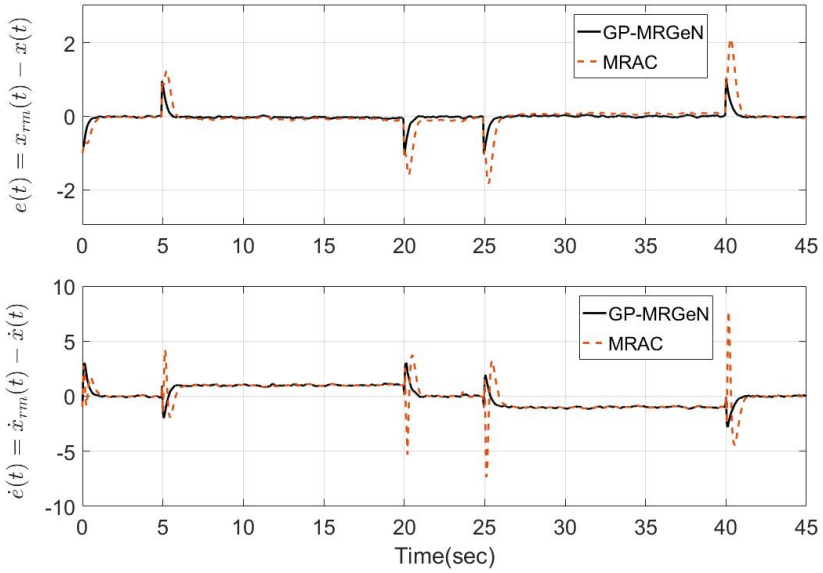


Figure 6.3: GP-MRGeN vs High Gain-MRAC Controller Evaluation on aircraft Wing-Rock dynamics model Tracking Error in $\theta(t)$ and $p(t)$

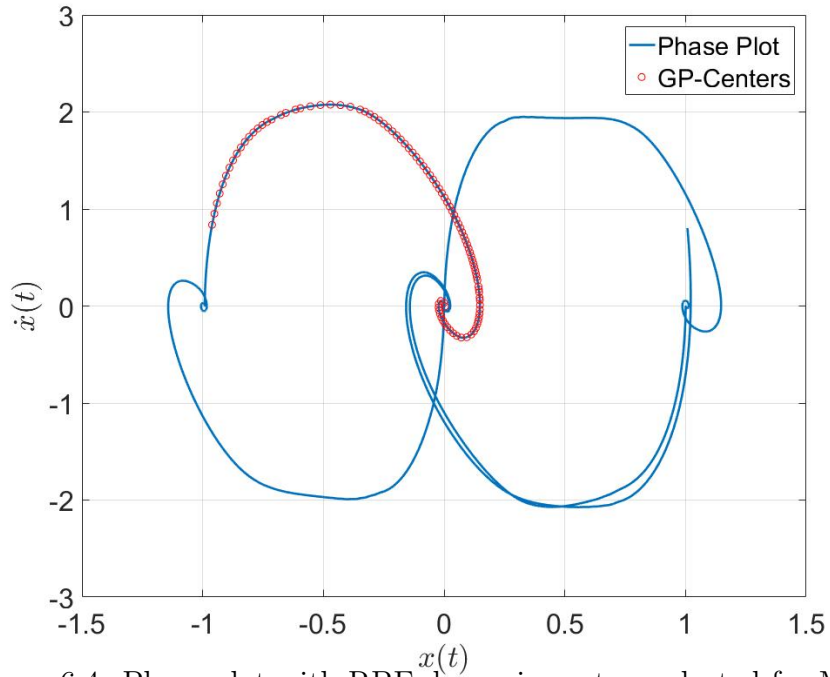


Figure 6.4: Phase plot with RBF dynamic centers selected for MRAC Controller

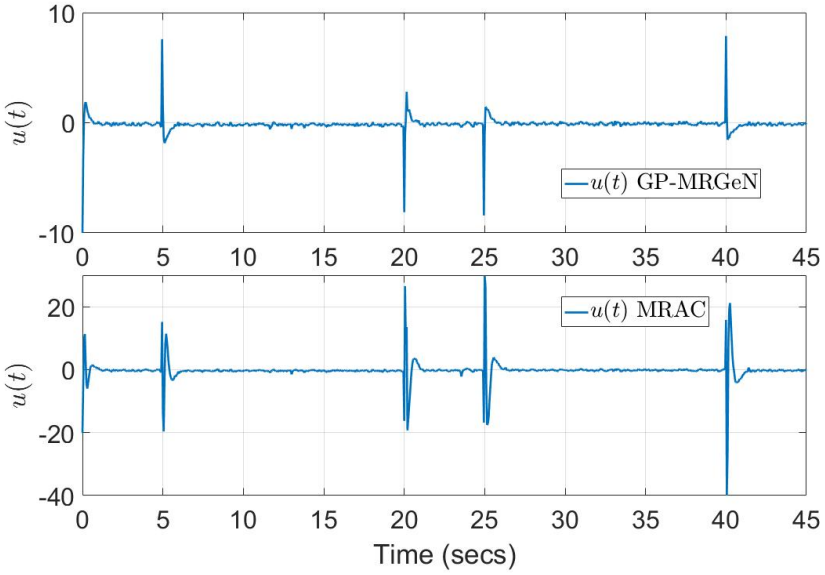


Figure 6.5: GP-MRGeN vs High Gain-MRAC-Total Control input $u(t)$

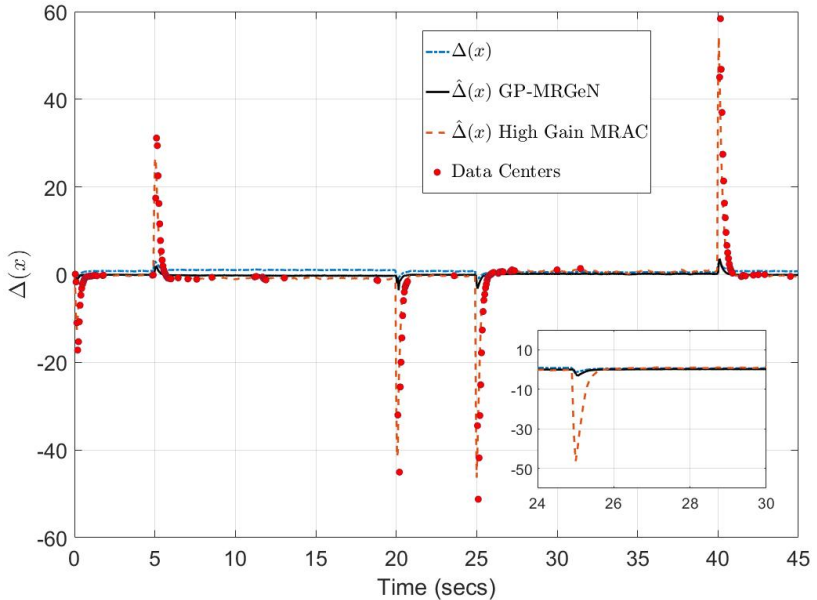


Figure 6.6: GP-MRGeN vs High Gain-MRAC-Uncertainty approximation. The blobs indicate the time step at which generative model is updated and queried for GP training

CHAPTER 7

DEEP MODEL REFERENCE ADAPTIVE CONTROL

Neural networks in adaptive control have been studied for a very long time. The seminal paper by Lewis [62] utilized Taylor series approximations to demonstrate uniform ultimate boundedness with a single hidden neural network. SHL networks are nonlinear in the parameters; hence the analysis previously introduced for linear in parameter, radial basis function neural networks introduced by Sanner and Slotine does not directly apply [59]. The back-propagation type scheme with non-increasing Lyapunov candidate as a constraint, introduced in Lewis' work has been widely used in Neuro-adaptive MRAC. Concurrent Learning MRAC (CL-MRAC) is a method for learning based neuro-adaptive control developed by the author to improve the learning properties and provide exponential tracking and weight error convergence guarantees. However, similar guarantees have not been available for SHL networks. There has been much work, towards including deeper neural networks in control; however, strong guarantees like those in MRAC on the closed loop stability during online learning are not available. Deep MRAC proposes a dual time-scale learning approach which provides such guarantees. Our approach should be generalizable to other applications of deep neural networks, including policy gradient Reinforcement Learning (RL) [159] which is very close to adaptive control in its formulation and also to more recent work in RL for control [97].

7.1 System Description

This section discusses the formulation of model reference adaptive control (see e.g. [120]). We consider the following system with uncertainty $\Delta(x)$:

$$\dot{x}(t) = Ax(t) + B(u(t) + f(x)) \tag{7.1}$$

where $x(t) \in \mathbb{R}^n$, $t \geqslant 0$ is the state vector, $u(t) \in \mathbb{R}^m$, $t \geqslant 0$ is the control input, $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$ are known system matrices and we assume the pair (A, B) is controllable. The term $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ is matched system uncertainty and be Lipschitz continuous and bounded in $x(t) \in \mathcal{D}_x$. Let $\mathcal{D}_x \subset \mathbb{R}^n$ be a compact set and the control $u(t)$ is assumed to belong to a set of admissible control inputs of measurable and bounded functions, ensuring the existence and uniqueness of the solution to (7.1).

The reference model is assumed to be linear and therefore the desired transient and steady-state performance is defined by a selecting the system eigenvalues in the negative half plane. The desired closed-loop response of the reference system is given by

$$\dot{x}_{rm}(t) = A_{rm}x_{rm}(t) + B_{rm}r(t) \tag{7.2}$$

where $x_{rm}(t) \in \mathcal{D}_x \subset \mathbb{R}^n$ and $A_{rm} \in \mathbb{R}^{n\times n}$ is Hurwitz and $B_{rm} \in \mathbb{R}^{n\times r}$. Furthermore, the command $r(t) \in \mathbb{R}^r$ denotes a bounded, piece wise continuous, reference signal and we assume the reference model (7.2) is bounded input-bounded output (BIBO) stable [120].

The true uncertainty $f(x)$ in unknown, but it is assumed to be bounded and continuous over a compact domain $\mathcal{D}_x \subset \mathbb{R}^n$. A Deep Neural Networks (DNN) have been widely used to represent unknown function when the basis vector is not known. Using DNNs, a non linearly parameterized network estimate of the uncertainty can be written as $f_\theta \triangleq \theta_n^T\Phi(x)$, where $\theta_n \in \mathbb{R}^{k\times m}$ are network weights for the final layer and $\Phi_n^\sigma(x) = \phi_n(\theta_{n-1}, \phi_{n-1}(\theta_{n-2}, \phi_{n-2}(...))))$, is a k dimensional feature vector which is function of inner layer weights, activations and inputs. The basis vector $\Phi_n^\sigma(x) \in \mathcal{F} : \mathbb{R}^n \to \mathbb{R}^k$ is considered to be Lipschitz continuous to ensure the existence and uniqueness of the solution (7.1).

7.1.1 Total Adaptive Controller

The aim is to construct a feedback law $u(t)$, $t \geqslant 0$, such that the state of the uncertain dynamical system (7.1) asymptotically tracks the state of the reference model (7.2) despite presence of matched uncertainty.

A tracking control law consisting of linear feedback term $u_{pd} = k_x x(t)$, a linear feed-forward term $u_{crm} = k_r r(t)$ and an adaptive term $u_{ad}(t)$ form the

total controller

$$u(t) = u_{pd}(t) + u_{crm}(t) - u_{ad}(t) \tag{7.3}$$

The baseline full state feedback and feed-forward controller is designed to satisfy the matching conditions such that $A_{rm} = A - Bk_x$ and $B_{rm} = Bk_r$. For the adaptive controller ideally we want $u_{ad}(t) = f(x(t))$. Since we do not have true uncertainty information, we use a DNN estimate of the system uncertainties in the controller as $u_{ad}(t) = f_\theta(x(t))$.

7.1.2 Deep Model Reference Adaptive Control

Unlike traditional MRAC or SHL-MRAC weight update rule, where the weights are moved in the direction of diminishing tracking error, training a deep neural network is much more involved. Feed-Forward networks like DNNs are trained in a supervised manner over a batch of i.i.d data. Deep learning optimization is based on Stochastic Gradient Descent (SGD) or its variants. The SGD update rule relies on a stochastic approximation of the expected value of the gradient of the loss function over a training set or mini-batches.

To train a deep network to estimate the system uncertainties, unlike MRAC we need labeled pairs of state-true uncertainties $\{x(t), f(x(t))\}$ i.i.d samples. Since we do not have access to true uncertainties $(f(x))$, we use a generative network to generate estimates of $f(x)$ to create the labeled targets for deep network training. For details of the generative network architecture in the adaptive controller, please see [131]. This generative network is derived from separating the DNN into inner feature layer and the final output layer of the network. We also separate in time-scale the weight updates of these two parts of DNN. Temporally separated weight update algorithm for the DNN, approximating system uncertainty is presented in more details in further sections.

7.1.3 Online Parameter Estimation law

The last layer of DNN with learned features from inner layer forms the Deep-Model Adaptive Controller(DMRAC). We use the MRAC learning rule to update pointwise in time, the weights of the D-MRGeN in the direction of

achieving asymptotic tracking of the reference model by the actual system.

Since we use the DMRAC estimates to train DNN model, we first study the admissibility and stability characteristics of the generative model estimate $u_{ad}(t) = W^T \Phi_n^\sigma(x)$ in the controller (7.3). To achieve the asymptotic convergence of the reference model tracking error to zero, we use the DMRAC estimate in the controller (7.3) as $u_{ad}(t) = W^T \Phi_n^\sigma(x)$

$$u_{ad}(t) = W^T \phi_n(\theta_{n-1}, \phi_{n-1}(\theta_{n-2}, \phi_{n-2}(...))) \tag{7.4}$$

To differentiate the weights of D-MRGeN from last layer weights of DNN "θ_n", we denote D-MRGeN weights as "W".

Assumption 7.1.1 *Appealing to the universal approximation property of Neural Networks [150] we have that, for every given basis functions $\Phi(x) \in \mathcal{F}$ there exists unique ideal weights $W^* \in \mathbb{R}^{k \times m}$ and $\epsilon_1(x) \in \mathbb{R}^m$ such that the following approximation holds*

$$f(x) = W^{*T} \Phi_n^\sigma(x) + \epsilon_1(x), \;\; \forall x(t) \in \mathcal{D}_x \subset \mathbb{R}^n \tag{7.5}$$

Fact 7.1.2 *The network approximation error $\epsilon_1(x)$ is upper bounded, s.t $\bar{\epsilon}_1 = \sup_{x \in \mathcal{D}_x} \|\epsilon_1(x)\|$, and can be made arbitrarily small given sufficiently large number of basis functions.*

The reference model tracking error is defined as $e(t) = x_{rm}(t) - x(t)$. Using (7.1) & (7.2) and the controller of form (7.3) with adaptation term u_{ad}, the tracking error dynamics can be written as

$$\dot{e}(t) = \dot{x}_{rm}(t) - \dot{x}(t) \tag{7.6}$$

$$\dot{e}(t) = A_{rm} e(t) + B \left(\tilde{W}^T \Phi_n^\sigma(x) + \epsilon_1(x) \right) \tag{7.7}$$

where $\tilde{W} = W^* - W$ is error in parameter.

The estimate of the unknown true network parameters W^* are calculated on-line using the weight update rule (7.8); correcting the weight estimates in the direction of minimizing the instantaneous tracking error $e(t)$. The resulting update rule for network weights in estimating the total uncertainty in the system is as follows

$$\dot{W} = \Gamma proj(W, \Phi_n^\sigma(x) e(t)' P B) \quad W(0) = W_0 \tag{7.8}$$

where $\Gamma \in \mathbb{R}^{k\times k}$ is the learning rate and $P \in \mathbb{R}^{n\times n}$ is a positive definite matrix. For given Hurwitz A_{rm}, the matrix $P \in \mathbb{R}^{n\times n}$ is a positive definite solution of Lyapunov equation $A_{rm}^T P + P A_{rm} + Q = 0$ for given $Q > 0$

Assumption 7.1.3 *For uncertainty parameterized by unknown true weight $W^* \in \mathbb{R}^{k\times m}$ and known nonlinear basis $\Phi(x)$, the ideal weight matrix is assumed to be upper bounded s.t $\|W^*\| \leq \mathcal{W}_b$. This is not a restrictive assumption.*

7.1.4 Lyapunov Analysis

The on-line adaptive identification law (7.8) guarantees the asymptotic convergence of the tracking errors $e(t)$ and parameter error $\tilde{W}(t)$ under the condition of persistency of excitation [55, 120] for the structured uncertainty. Similar to the results by Lewis for SHL networks [47], we show here that under the assumption of unstructured uncertainty represented by a deep neural network, the tracking error is uniformly ultimately bounded (UUB). We will prove the following theorem under switching feature vector assumption.

Theorem 7.1.4 *Consider the actual and reference plant model (7.1) & (7.2). If the weights parameterizing total uncertainty in the system are updated according to identification law* (7.8) *Then the tracking error $\|e\|$ and error in network weights $\|\tilde{W}\|$ are bounded for all $\Phi_n^\sigma \in \mathcal{F}$.*

Proof: The feature vectors belong to a function class characterized by the inner layer network weights θ_i s.t $\Phi_n^\sigma \in \mathcal{F}$. We will prove the Lyapunov stability under the assumption that inner layer of DNN presents us a feature which results in the worst possible approximation error compared to network with features before switch.

For the purpose of this proof let $\Phi_n^{\sigma_j}(x), \Phi_n^{\sigma_i}(x)$ denote feature before and after switch and $\Phi_n^{\sigma_i}(x)$ be the feature after switch. We define the error $\epsilon_2(x)$ as,

$$\epsilon_2(x) = \sup_{\Phi\in\mathcal{F}} \left\| W^T \Phi_n^{\sigma_i}(x) - W^T \Phi_n^{\sigma_j}(x) \right\| \tag{7.9}$$

Similar to Fact-7.1.2 we can upper bound the error $\epsilon_2(x)$ as

$$\bar{\epsilon}_2 = \sup_{x\in\mathcal{D}_x} \|\epsilon_2(x)\|$$

By adding and subtracting the term $W^T\Phi_n^{\sigma_j}(x)$, we can rewrite the error dynamics (7.7) with switched basis as,

$$\dot{e}(t) = A_{rm}e(t)+B\left(W^{*T}\Phi_n^{\sigma_i}(x) - W^T\Phi_n^{\sigma_i}(x) + W^T\Phi_n^{\sigma_j}(x) - W^T\Phi_n^{\sigma_j}(x) + \epsilon_1(x)\right) \tag{7.10}$$

From Assumption-7.1.1 we know there exists a W^* $\forall\sigma$ and $\forall\Phi_n^\sigma \in \mathcal{F}$. Therefore we can replace $W^{*T}\Phi(x)_n^{\sigma_i}$ by $W^{*T}\Phi_n^{\sigma_j}(x)$ and rewrite the Eq-(7.10) as

$$\dot{e}(t) \;=\; A_{rm}e(t) + \tilde{W}^T\Phi_n^{\sigma_j}(x) + W^T(\Phi_n^{\sigma_j}(x) - \Phi_n^{\sigma_i}(x)) + \epsilon_1(x) \tag{7.11}$$

For arbitrary switching σ, for any $\Phi_n^\sigma(x) \in \mathcal{F}$, we can prove the boundedness by considering worst possible approximation error and therefore can write,

$$\dot{e}(t) \;=\; A_{rm}e(t) + \tilde{W}^T\Phi_n^{\sigma}(x) + \epsilon_2(x) + \epsilon_1(x) \tag{7.12}$$

Now lets consider $V(e,\tilde{W}) > 0$ be a differentiable, positive definite radially unbounded Lyapunov candidate function,

$$V(e,\tilde{W}) = e^TPe + \frac{\tilde{W}^T\Gamma^{-1}\tilde{W}}{2} \tag{7.13}$$

The time derivative of the Lyapunov function (7.13) along the trajectory (7.12) can be evaluated as

$$\dot{V}(e,\tilde{W}) = \dot{e}^TPe + e^TP\dot{e} - \tilde{W}^T\Gamma^{-1}\dot{\hat{W}} \tag{7.14}$$

Using (7.12) & (7.8) in (7.14), the time derivative of the lyanpunov function reduces to

$$\dot{V}(e,\tilde{W}) \;=\; -e^TQe + 2e^TP\epsilon(x) \tag{7.15}$$

where $\epsilon(x) = \epsilon_1(x) + \epsilon_2(x)$ and $\bar{\epsilon} = \bar{\epsilon_1} + \bar{\epsilon_2}$.
Hence $\dot{V}(e,\tilde{W}) \leq 0$ outside compact neighborhood of the origin $e = 0$, for some sufficiently large $\lambda_{min}(Q)$.

$$\|e(t)\| \geq \frac{2\lambda_{max}(P)\bar{\epsilon}}{\lambda_{min}(Q)} \tag{7.16}$$

Using the BIBO assumption $x_{rm}(t)$ is bounded for bounded reference signal

$r(t)$, thereby $x(t)$ remains bounded. Since $V(e, \tilde{W})$ is radially unbounded the result holds for all $x(0) \in \mathcal{D}_x$. Using the fact, the error in parameters $\tilde{W}$ are bounded through projection operator [143] and further using Lyapunov theory and Barbalat's Lemma [144] we can show that $e(t)$ is uniformly ultimately bounded in vicinity to zero solution.

From Theorem-7.1.4 & Eq-(7.7) and using system theory [160] we can infer that as $e(t) \to 0$, $W^T \Phi_n^\sigma(x) \to f(x)$ in point-wise sense. Hence D-MRGeN estimates $y_\tau = W^T \Phi_n^\sigma(x_\tau)$ are admissible target values for training DNN features over the data $Z^M = \{\{x_\tau, y_\tau\}\}_{\tau=1}^M$.

The details of DNN training and implementation details of DMRAC controller is presented in following section:

7.2 Adaptive Control using Deep Nets (DMRAC)

The DNN architecture for MRAC is trained in two steps. We separate the DNN into two networks as shown in Fig-7.1. The faster learning outer adaptive network and slower deep feature network. DMRAC learns underlying deep feature vector to the system uncertainty using locally exciting uncertainty estimates obtained using a generative network. Between successive updates of the inner layer weights, the feature provided by the inner deep network is used as the fixed feature vector for outer layer adaptive network update and evaluation. The algorithm for DNN learning and DMRAC controller is provided in Algorithm-4. Through this architecture of mixing two-time scale learning, we fuse the benefits of DNN memory through retention of relevant, exciting features and robustness, boundedness guarantee in reference tracking. This key feature of the presented framework ensures robustness while guaranteeing long term learning and memory in the adaptive network.

Also as indicated in the controller architecture Fig-7.1 we can use contextual state 'c_i' other than system state $x(t)$ to extract relevant features. These contextual states could be relevant model information not captured in system states. For example for an aircraft system, vehicle parameters like pitot tube measurement, the angle of attack, engine thrust and so on. These contextual states can extract features which help in decision making in case of faults. The work on DMRAC with contextual states will be dealt with in

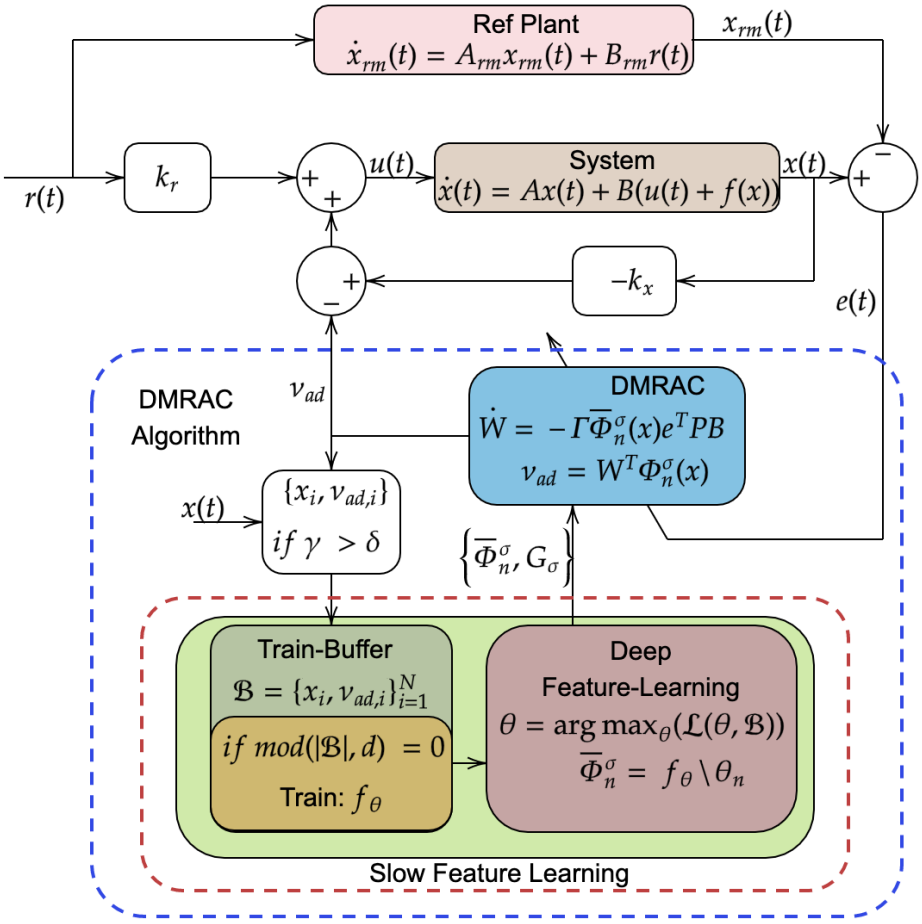


Figure 7.1: DMRAC training and controller details

the follow on work.

The DNN in DMRAC controller is trained over training dataset $Z^M = \{x_i, y_i\}_{i=1}^M$, where the y_i are D-MRGeN estimates of the uncertainty. The training dataset Z^M is randomly drawn from a larger data buffer $\mathcal{B}$. Not every pair of data $\{x_i, y_i\}$ from D-MRGeN is added to the training buffer $\mathcal{B}$. We qualify the input-target pair based on kernel independence test such that to ensure that we collect locally exciting independent information which provides a sufficiently rich representation of the operating domain. Since the state-uncertainty data is the realization of a Markov process, such a method for qualifying data to be sufficiently independent of previous data-points is necessary. The algorithm details to qualify and add a data point to the buffer is provided in detail in subsection 7.2.2.

7.2.1 Details of Deep Feature Training using D-MRGeN

This section provides the details of the DNN training over data samples observed over n-dimensional input subspace $x(t) \in \mathcal{X} \in \mathbb{R}^n$ and m-dimensional targets subspace $y \in \mathcal{Y} \in \mathbb{R}^m$. The sample set is denoted as $\mathcal{Z}$ where $\mathcal{Z} \in \mathcal{X} \times \mathcal{Y}$.

We are interested in the function approximation tasks for DNN. The function $f_{\boldsymbol{\theta}}$ is the learned approximation to the model uncertainty with parameters $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, where $\boldsymbol{\Theta}$ is the space of parameters, i.e. $f_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^m$. We assume a training data buffer $\mathcal{B}$ has p_{max} training examples, such that the set $Z^{p_{max}} = \{Z_i | Z_i \in \mathcal{Z}\}_{i=1}^{p_{max}} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^{p_{max}}$. The samples are independently drawn from the buffer $\mathcal{B}$ over probability distribution P. The hypothesis set, which consist of all possible functions $f_{\boldsymbol{\theta}}$ is denoted as $\mathcal{H}$. Therefore a learning algorithm $\mathcal{A}$ (in our case SGD) is a mapping from $\mathcal{A} : \mathcal{Z}^{p_{max}} \to \mathcal{H}$

The loss function, which measures the discrepancy between true target y and algorithm's estimated target function value $f_{\boldsymbol{\theta}}$ is denoted by $L(y, f_{\boldsymbol{\theta}}(x))$. Specific to work presented in this section using Deep Neural Networks, we use a ℓ_2-norm between values i.e.

$$\mathbb{E}_p(\ell(y, f_{\boldsymbol{\theta}}(x))) = \mathbb{E}_P\left(\|y_i - f_{\boldsymbol{\theta}}(x_i)\|_2\right) \tag{7.17}$$

as loss function for DNN training. The empirical loss (7.34) is used to approximate the loss function since the distribution P is unknown to learning algorithm. The weights are updated using SGD in the direction of negative gradient of the loss function as given in (7.35).

Unlike the conventional DNN training where the true target values $y \in \mathcal{Y}$ are available for every input x, in DMRAC true system uncertainties as the labeled targets are not available for the network training. We use the part of the network itself (the last layer) with pointwise weight updated according to MRAC-rule as the generative model for the data. The D-MRGeN uncertainty estimates $y_i = W^T \Phi(x, \theta_1, \theta_2, \ldots \theta_{n-1})$ along with inputs x_i make the training data set $Z^{p_{max}} = \{x_i, y_i\}_{i=1}^{p_{max}}$. Note that we use interchangably x_i and $x(t)$ as discrete representation of continuous state vector for DNN training. The main purpose of DNN in the adaptive network is to extract relevant features of the system uncertainties, which otherwise is very tedious to obtain without

the limits on the domain of operation.

We also demonstrate empirically, that the DNN features trained over past i.i.d representative data retains the memory of the past instances and can be used as the frozen feed-forward network over similar reference tracking tasks without loss of the guaranteed tracking performance.

7.2.2 Method for Recording Data using MRGeN for DNN Training

In statistical inference, implicitly or explicitly one always assume that the training set $Z^M = \{x_i, y_i\}_{i=1}^M$ is composed on M-input-target tuples that are independently drawn from buffer $\mathcal{B}$ over same joint distribution $P(x, y)$. The i.i.d assumption on the data is required for robustness, consistency of the network training and for bounds on the generalization error [161, 162]. In classical generalization proofs one such condition is that $\frac{1}{p_{max}}\mathbb{X}^T\mathbb{X} \to \gamma$ as $p_{max} \to \infty$, where $\mathbb{X}$ denotes the design matrix with rows Φ_i^T. The i.i.d assumption implies the above condition is fulfilled and hence is sufficient but not necessary condition for consistency and error bound for generative modeling.

The key capability brought about by DMRAC is a relevant feature extraction from the data. Feature extraction in DNN is achieved by using recorded data concurrently with current data. The recorded data include the state x_i, feature vector $\Phi(x_i)$ and associated D-MRGeN estimate of the uncertainty $\Delta'(x_i)$. For a given $\zeta_{tol} \in \mathbb{R}_+$ a simple way to select the instantaneous data point $\{x_i, \Delta'(x_i)\}$ for recording is to require

$$\gamma_i = \frac{\|\Phi(x_i) - \Phi_p\|^2}{\|\Phi(x_i)\|} \geq \zeta_{tol} \tag{7.18}$$

Where the index p is over the data points in buffer $\mathcal{B}$. The above method ascertains only those data points are selected for recording that are sufficiently different from all other previously recorded data points in the buffer. Since the buffer $\mathcal{B}$ is of finite dimension, the data is stored in a cyclic manner. As the number of data points reaches the buffer budget, a new data is added only upon one existing data point is removed such that the singular value of the buffer is maximized. The singular value maximization approach for the

training data buffer update is provided in [163].

Algorithm 4 D-MRAC Controller Training

Input: $\Gamma, \eta, \zeta_{tol}, p_{max}, \sigma$
while New measurements are available **do**
  Update the D-MRGeN weights using Eq:(7.8)
  Compute $y_{\tau+1} = W^T \Phi_n^\sigma(x_{\tau+1})$
  Given $x_{\tau+1}$ compute $\gamma_{\tau+1}$ by Eq-(7.18).
  if $\gamma_{\tau+1} \geqslant \zeta_{tol}$ **then**
    Update $\mathcal{B} : \boldsymbol{Z}(:) = \{x_{\tau+1}, y_{\tau+1}\}$ and X: $\Phi(x_{\tau+1})$
    if $|\mathcal{B}| > p_{max}$ **then**
      Delete element in $\mathcal{B}$ by SVD maximization [163]
    end if
  end if
  if $|\mathcal{B}| \geq M$ **then**
    Sample a mini-batch of data $\boldsymbol{Z}^M \subset \mathcal{B}$
    Train the DNN network over mini-batch data using Eq-(7.35)
    Update the feature vector Φ_n^σ for D-MRGeN network
    Update the Switching signal σ
  end if
end while=0

7.3 Adaptive Control Using Bayesian Deep Neural Networks

In the last sections we presented a MRAC controller using Deep Neural Networks. Despite tremendous success of Deep Neural networks in modelling functions, they suffer from overfitting. When applied to supervised learning problems deep networks are often incapable of correctly assessing the uncertainty in the training data and so make overly confident decisions about the correct class or prediction. To address this issue of over-fitting of deep neural networks in estimating model uncertainty in MRAC we introduce a Bayesian Deep net MRAC. Apart from the benefits of Bayesian learning in adding a natural regularization to feature learning; we can show that using finitely exciting training data a desirable property of "*Induced Persistency of Excitation*" of the features can be obtained. The persistency of excitation property of the deep features ensure the outer layer weights converge to their true values and hence providing a desirable convergence of all signals.

All the layer weights in the Bayesian neural networks are represented by probability distributions over possible values, rather than having a single fixed value as in the traditional neural networks. The learnt deep features and therefore adaptive element will be a stochastic quantity. Therefore we also prove the robustness and stability of the stochastic adaptive controller for given uncertain system under perturbation of the weights. The amount of perturbation each weight exhibits is also learnt in a way that coherently explains variability in the training data. Thus instead of training a single network, the Bayesian neural network trains an ensemble of networks, where each network has its weights drawn from a shared, learnt probability distribution.

In the previous chapter we introduced a Gaussian process Model reference adaptive control where the adaptive element is linear in parameter single layer network. The weight of the network are drawn over a Gaussian distribution, which is updated using Bayes law. We also showed that a linear in parameter, Gaussian prior and Gaussian Likelihood assumption leads to an closed form update of the posterior distribution over the network weights. However for deep neural network Bayesian updated of the posterior belief on network parameters is intractable, due to intractable evidence term. We will see the details of Bayesian update for Deep neural networks in further sections.

7.3.1 Bayesian Deep Neural Networks

Bayesian Neural Networks(BNN) are comprised of a Probabilistic Model and a Neural Network. The intent of such a design is to combine the strengths of Neural Networks and Stochastic modeling. Neural Networks exhibit universal approximation property over continuous functions. Probabilistic models allows us to combine prior and likelihood to produce probabilistic guarantees on it's predictions through generating distribution over the parameters learnt from the observations. Therefore the network can specify the confidence interval over the prediction. Thus BNNs are a unique combination of neural network and stochastic models.

A neural network can be viewed as a probabilistic model $P(\mathcal{Y}|\mathcal{X}, \boldsymbol{\theta})$. We assume y is a continuous variable and $P(\mathcal{Y}|\mathcal{X}, \boldsymbol{\theta})$ is gaussian likelihood. Give the data $Z^M = \{\{x_i, y_i\} \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^M$. The total likelihood can be written

as,

$$p(Z|\boldsymbol{\theta}) = \prod_i \left(p(y_i|x_i, \boldsymbol{\theta})\right) \tag{7.19}$$

Which is function of network parameters Θ. Maximizing this likelihood we achieve a maximum likelihood estimate of the parameters given the data, which is introduced in the previous section. Assuming a gaussian likelihood, performing maximization of negative log likelihood leads to least square regression which can suffer over-fitting.

Using a prior belief on the parameters $p(\Theta)$ and likelihood of the data given a deep neural network model, we can obtain a updated posterior belief on the parameter using Bayes rule as follows,

$$p(\boldsymbol{\theta}|Z) = \frac{p(Z|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_\theta p(Z|\theta)p(\boldsymbol{\theta})d\boldsymbol{\theta}} \tag{7.20}$$

Maximizing the posterior $p(\boldsymbol{\theta}|Z)$ leads to Maximum-a-posterior (MAP) estimate of parameter $\boldsymbol{\theta}$. Computing the MAP estimate has a regularizing effect and can prevent overfitting. The optimization objectives here are the same as for MLE plus a regularization term coming from the log prior. If we had a full posterior distribution over parameters we could make predictions that take weight uncertainty into account by marginalizing the parameters as follows,

$$p(y|x, Z) = \int_\theta p(y|x, \boldsymbol{\theta})p(\boldsymbol{\theta}|Z)d\boldsymbol{\theta} \tag{7.21}$$

This is equivalent to averaging predictions from an ensemble of neural networks weighted by the posterior probabilities of their parameters $\boldsymbol{\theta}$.

7.3.2 Variation Inference

Unlike Gaussian Processes, analytical solution for posterior $p(\boldsymbol{\theta}|Z)$ for a multi-layer neural network is intractable. We therefore have to approximate the true posterior with a variational distribution $q(\boldsymbol{\theta}|\boldsymbol{\zeta})$. The variational distribution is a known function parameterized by variational parameters $\boldsymbol{\zeta}$ which are estimated online. The variational parameters are estimated online by minimizing the Kullback-Leibler (KL) divergence between variational distribution $q(\boldsymbol{\theta}|\boldsymbol{\zeta})$ and true posterior $p(\boldsymbol{\theta}|Z)$. The KL objective can be written

as,

$$KL\left(q(\boldsymbol{\theta}|\boldsymbol{\zeta})\|p(\boldsymbol{\theta}|Z)\right) = \int q(\boldsymbol{\theta}|\boldsymbol{\zeta}) \log\left(\frac{q(\boldsymbol{\theta}|\boldsymbol{\zeta})}{p(\boldsymbol{\theta}|Z)}\right) d\boldsymbol{\theta}$$
$$= \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\zeta})} \log\left(\frac{q(\boldsymbol{\theta}|\boldsymbol{\zeta})}{p(\boldsymbol{\theta}|Z)}\right) \tag{7.22}$$

Using the expression for true posterior (7.20), we can simplify the above expression as

$$KL\left(q(\boldsymbol{\theta}|\boldsymbol{\zeta})\|p(\boldsymbol{\theta}|Z)\right) = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\zeta})} \log\left(\frac{q(\boldsymbol{\theta}|\boldsymbol{\zeta})}{p(Z|\boldsymbol{\theta})p(\boldsymbol{\theta})}p(Z)\right) \tag{7.23}$$
$$= \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\zeta})}\left[\log\left(q(\boldsymbol{\theta}|\boldsymbol{\zeta})\right) - \log\left(p(Z|\boldsymbol{\theta})\right) - \log\left(p(\boldsymbol{\theta}) + \log\left(p(Z)\right)\right)\right] \tag{7.24}$$

Since $\log\left(p(Z)\right)$ is independent of parameters $\boldsymbol{\theta}, \boldsymbol{\zeta}$ we cam simply the above expression as

$$KL\left(q(\boldsymbol{\theta}|\boldsymbol{\zeta})\|p(\boldsymbol{\theta}|Z)\right) = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\zeta})}\left[\log\left(q(\boldsymbol{\theta}|\boldsymbol{\zeta})\right) - \log\left(p(Z|\boldsymbol{\theta})\right) - \log\left(p(\boldsymbol{\theta})\right] + \log\left(p(Z)\right)\right.$$
$$= KL\left(q(\boldsymbol{\theta}|\boldsymbol{\zeta})\|p(\boldsymbol{\theta})\right) - \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\zeta})}\left(\log\left(p(Z|\boldsymbol{\theta})\right)\right) + \log(p(Z)) \tag{7.25}$$

As we know $\log\left(p(Z)\right)$ is independent of parameter $\boldsymbol{\theta}$, to minimize the KL-divergence $KL\left(q(\boldsymbol{\theta}|\boldsymbol{\zeta})\|p(\boldsymbol{\theta}|Z)\right)$, we can minimize only the first two terms in (7.25) which are known as Variational Free Energy $\mathcal{F}(Z, \boldsymbol{\theta}, \boldsymbol{\zeta})$

$$KL\left(q(\boldsymbol{\theta}|\boldsymbol{\zeta})\|p(\boldsymbol{\theta}|Z)\right) = \mathcal{F}(Z, \boldsymbol{\theta}, \boldsymbol{\zeta}) + \log(p(Z)) \tag{7.26}$$

where

$$\mathcal{F}(Z, \boldsymbol{\theta}, \boldsymbol{\zeta}) = KL\left(q(\boldsymbol{\theta}|\boldsymbol{\zeta})\|p(\boldsymbol{\theta})\right) - \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\zeta})}\left(\log\left(p(Z|\boldsymbol{\theta})\right)\right) \tag{7.27}$$

Negative Variational Free Energy is also known as Evidence Lower Bound (ELBO)

$$\mathcal{L}(Z, \boldsymbol{\theta}, \boldsymbol{\zeta}) = -\mathcal{F}(Z, \boldsymbol{\theta}, \boldsymbol{\zeta}) \tag{7.28}$$

Therefore the KL-divergence objective can be written as,

$$KL\left(q(\boldsymbol{\theta}|\boldsymbol{\zeta})\|p(\boldsymbol{\theta}|Z)\right) = -\mathcal{L}(Z, \boldsymbol{\theta}, \boldsymbol{\zeta}) + \log(p(Z)) \tag{7.29}$$

The term $\mathcal{L}(Z, \boldsymbol{\theta}, \boldsymbol{\zeta})$ is known as evidence lower bound since it defines lower bound on the evidence term $p(Z)$ as follows,

$$\mathcal{L}(Z, \boldsymbol{\theta}, \boldsymbol{\zeta}) = \log(p(Z)) - KL\left(q(\boldsymbol{\theta}|\boldsymbol{\zeta}) \| p(\boldsymbol{\theta}|Z)\right) \tag{7.30}$$

Since the KL-divergence term is always positive semi-definite we can write,

$$\mathcal{L}(Z, \boldsymbol{\theta}, \boldsymbol{\zeta}) \leq \log(p(Z)) \tag{7.31}$$

Therefore, the KL divergence between the variational distribution $q(\boldsymbol{\theta}|\boldsymbol{\zeta})$ and the true posterior $p(\boldsymbol{\theta}|Z)$ is also minimized by maximizing the evidence lower bound as follows,

$$\boldsymbol{\zeta} \leftarrow arg \max_{\boldsymbol{\zeta}} \mathcal{L}(Z, \boldsymbol{\theta}, \boldsymbol{\zeta}) \tag{7.32}$$

And the Deep feature parameters are drawn from this variational distribution as follows,

$$\boldsymbol{\theta} \sim q(\boldsymbol{\theta}|\boldsymbol{\zeta}) \tag{7.33}$$

We use Bayes by Backprop [164] algorithm to learn the latent parameters of variational distribution using stochastic gradient descent algorithm whose details are provided in next section.

7.3.3 Stochastic Gradient Descent and Batch Training

Lets consider a deep network model with parameters $\boldsymbol{\theta}$ (or $\boldsymbol{\zeta}$) for bayesian networks, and consider the problem of optimizing a non convex loss function $\mathcal{L}(\boldsymbol{Z}, \boldsymbol{\theta})$, with respect to $\boldsymbol{\theta}$. Let $\mathcal{L}(\boldsymbol{Z}, \boldsymbol{\theta})$ is defined as average of loss over sample of M training data points.

$$\mathcal{L}(\boldsymbol{Z}, \boldsymbol{\theta}) = \frac{1}{M} \sum_{i}^{M} \ell(\boldsymbol{Z_i}, \boldsymbol{\theta}) \tag{7.34}$$

where M denotes the size of sample training set. For each sample size of M, the training data are in form of M-tuple $Z^M = (Z_1, Z_2, \ldots Z_M)$ of $Z-$valued random variables drawn according to some unknown distribution $P \in \mathcal{P}$. Where each $Z_i = \{x_i, y_i\}$ are the labelled pair of input and target values. For each P the expected loss can be computed as $\boldsymbol{E}_p(\ell(\boldsymbol{Z}, \boldsymbol{\theta}))$. The above empirical loss (7.34) is used as proxy for the expected value of loss with

respect to the true data generating distribution.

Optimization based on the Stochastic Gradient Descent (SGD) algorithm uses a stochastic approximation of the gradient of the loss $\mathcal{L}(\boldsymbol{Z}, \boldsymbol{\theta})$ obtained over a mini-batch of M training examples drawn from buffer $\mathcal{B}$. The resulting SGD weight update rule

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \frac{1}{M} \sum_{i}^{M} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{7.35}$$

where η is the learning rate.

7.4 DMRAC weight update using Bayesian Deep Features

In previous section we have presented the DMRAC update rule and its stability characteristics using the standard deep features. In this section we will extend the results to stochastic deep features. The deep feature vector is defined as $\Phi_n^\sigma(x) = f_{\boldsymbol{\theta}} \backslash \theta_n$, where $f_{\boldsymbol{\theta}}$ is the entire network. The deep neural network stripped off its final layer weights form the deep features. Further due to stochastic nature of the network, the feature vector $\Phi_n^\sigma(x)$ is a draw over multivariate normal distribution due to stochastic network weights.

$$\Phi_n^\sigma(x) \sim \mathcal{N}\left(\bar{\Phi}_n^\sigma(x), \left[G_\sigma(x) G_\sigma(x)^T\right] | \boldsymbol{\theta}\right) \tag{7.36}$$

Where $\bar{\Phi}_n^\sigma(x), \left[G_\sigma(x) G_\sigma(x)^T\right]$ are mean and covariance for the feature.

Using the outer layer weights W, updated according to MRAC rule and the deep features from Bayesian deep neural network, we can write adaptive element $u_{ad}(t)$ in the total controller as,

$$u_{ad}(t) = W^T \Phi_n^\sigma(x(t)) \tag{7.37}$$

We will show using stochastic stability theory that the following weight update rule leads to mean square uniform ultimate boundedness almost surely. The weight update rule for the final layer weights using MRAC rule is given as

$$\dot{W}(t) = -\Gamma \bar{\Phi}_n^\sigma(x) e(t)^T P B \tag{7.38}$$

Where $\bar{\Phi}_n^\sigma(x)$ is mean of the features given as

$$\bar{\Phi}_n^\sigma(x) = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\zeta})}\left(\Phi_n^\sigma(x)\right) \tag{7.39}$$

However in case the true expectation is intractable, we can estimate the expectation empirically as follows,

$$\hat{\Phi}_n^\sigma(x) = \frac{1}{N}\sum_{i=1}^{N}\left(\Phi_{n,i}^\sigma(x)\right) \tag{7.40}$$

Where $\Phi_{n,i}^\sigma(x)$ is the i^{th} draw from distribution (7.73) for every given $x(t)$. Therefore we can modify the weight update law as,

$$\dot{W} = proj\left(-\Gamma\left(\frac{1}{N}\sum_{i=1}^{N}\left(\Phi_{n,i}^\sigma(x)\right)\right)e(t)^T PB, W\right)$$

$$\dot{W} = proj\left(-\Gamma\hat{\Phi}_n^\sigma(x)\, e(t)^T PB, W\right)$$

We are using the projection operator to bound the weights. However in the further section we will show that using a stochastic feature presents the property of *Induced Persistency of Excitation*, which ensures the learn weights converge to true weights neighborhood for all switching signal σ.

7.5 Stability Analysis and Sample Complexity for Stochastic-DMRAC

In this section, we present the sample complexity of learning of deep features and stochastic stability results for Stochastic-DMRAC(S-DMRAC) with bayesian deep neural network. In further section we will also analyze the deep features and show that the DMRAC controller is characterized by the memory over previously observed data and demonstrate in simulation that when DMRAC is used as a feed-forward network with frozen weight (no learning) can still produce bounded tracking performance results on reference tracking tasks that are reasonably different from those seen during network training. We ascribe this property of DMRAC to the very low generalization error bounds of the DNN. We will prove this property in two steps. Firstly we will prove

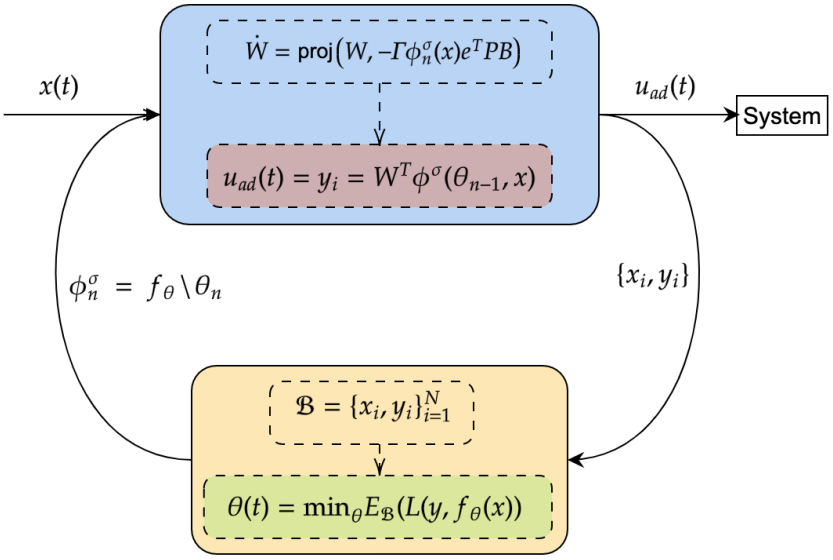


Figure 7.2: DMRAC Update Scheme for Outer-layer weights and Inner-layer Deep feature

the bound on the generalization error of DNN using Lyapunov theory such that we achieve an asymptotic uniform boundedness in tracking error. Further, we will show information theoretically the lower bound on the number of independent datasets we need to train upon before we can claim the DNN generalization error is well below a determined lower level given by Lyapunov analysis.

7.5.1 Stability Analysis

In this section we will introduce the stochastic stability proof for S-DMRAC controller. Using the above stochastic adaptive element (7.37) in the total controller (7.3) we can rewrite the system dynamics (7.1) as follows,

$$\dot{x}(t) = Ax(t) + B(-k_x x(t) + k_r r(t) - u_{ad} + f(x)) \tag{7.41}$$

Using the matchcing condition we can write,

$$\dot{x}(t) = A_{rm}x(t) + B_{rm}r(t) + B(f(x) - u_{ad}) \tag{7.42}$$

Using the expression for stochastic adaptive element we can rewrite the above expression as

$$\dot{x}(t) = A_{rm}x(t) + B_{rm}r(t) + B(f(x) - W^T\Phi_n^\sigma(x)) \tag{7.43}$$

Since we know the the feature vector instance is draw over the distribution of parameters $\boldsymbol{\theta} \sim q(\boldsymbol{\theta}|\boldsymbol{\zeta})$, we can write the random draw of the feature vector as

$$\Phi_n^\sigma(x) = \bar{\Phi}_n^\sigma + G_\sigma\xi \tag{7.44}$$

where $\bar{\Phi}_n^\sigma \in \mathbb{R}^k$ is the mean of feature distribution and $G_\sigma \in \mathbb{R}^k$ is variance and $\xi \sim \mathcal{N}(0, I)$. Using the feature instance we can write the controller expression in the system dynamic as

$$\begin{aligned} \dot{x}(t) &= A_{rm}x(t) + B_{rm}r(t) + B\left(f(x) - W^T\left(\bar{\Phi}_n^\sigma(x) + G_\sigma\xi\right)\right) & (7.45)\\ &= A_{rm}x(t) + B_{rm}r(t) + B\left(f(x) - W^T\bar{\Phi}_n^\sigma(x) - W^TG_\sigma\xi\right) & (7.46)\end{aligned}$$

We assume for every $\bar{\Phi}_n^\sigma$ that there exists an ideal weight W^*, such that,

$$f(x) = W^{*T}\bar{\Phi}_n^\sigma(x) + \epsilon_m^\sigma(x) \tag{7.47}$$

Where $\epsilon_m^\sigma(x) = \Delta(x) - W^{*T}\bar{\Phi}_n^\sigma(x)$ is network approximation error due to universal approximation theorem, and this error can be bounded as,

$$\bar{\epsilon}^\sigma = \sup_{x\in\mathcal{D}} \left\| f(x) - W^{*T}\bar{\Phi}_n^\sigma(x) \right\| \tag{7.48}$$

Using the reference model (7.2), we can write the error dynamics as,

$$\dot{e}(t) = A_{rm}e(t) + B\left(\tilde{W}^T\bar{\Phi}_n^\sigma(x) + \epsilon_m^\sigma(x)\right) + BW^TG_\sigma\xi \tag{7.49}$$

Where $\tilde{W} = W^* - W$. We can write the above tracking error dynamics as diffusion process as follows,

$$de(t) = A_{rm}e(t)dt + B\left(\tilde{W}^T\bar{\Phi}_n^\sigma(x) + \epsilon_m^\sigma(x)\right)dt + BW^TG_\sigma d\xi(t) \tag{7.50}$$

where $d\xi(t)$ is zero mean Wiener process.

Theorem 7.5.1 *Consider the system in* (7.1)*, the control law of* (7.3)*, and assume that the uncertainty* $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ *is bounded and Lipschitz con-*

tinuous on a compact set $x(t) \in \mathcal{D}_x$. Let the reference signal $r(t)$ is such that the state $x_{rm}(t)$ of the bounded input bounded output reference model (7.2) remains bounded in the compact ball $\mathcal{B}_m = \{x_{rm} : \|x_{rm}(t)\| \leq \mathcal{B}_\delta\}$, then the adaptive control $u_{ad} = W^T \Phi_n^\sigma(x)$, guarantee that the system is mean square uniformly ultimately bounded in probability a.s.

Proof Let,

$$V(e_t, \tilde{W}(t)) = \frac{1}{2} e_t^T P e_t + \frac{1}{2} tr\left(\tilde{W}(t)^T \Gamma^{-1} \tilde{W}(t)\right) \tag{7.51}$$

be stochastic lyapunov candidate. The error term e_t is a realization of drift diffusion process, we assume t as index and not function of time, whereas $W(t)$ is a deterministic process $W(t)$ is a function of time. Note that the lyapunov candidate is bounded above below by

$$\frac{1}{2}\underline{\lambda}(P)\|e_t\|^2 + \frac{1}{2}\underline{\lambda}\left(\Gamma^{-1}\right)\|\tilde{W}\|_F^2 \leq V(e_t, \tilde{W}) \leq \frac{1}{2}\bar{\lambda}(P)\|e_t\|^2 + \frac{1}{2}\bar{\lambda}\left(\Gamma^{-1}\right)\|\tilde{W}\|_F^2 \tag{7.52}$$

where $\underline{\lambda}$ and $\bar{\lambda}$ is the minimum and maximum Eigen value operator and $\|\|_F$ is the Frobenius norm. The Ito derivative of the Lyapunov candidate along the error dynamics (7.50) for σ^{th} system,

$$\begin{aligned}
\mathcal{L}V\left(e_t, \tilde{W}\right) &= \frac{\partial V}{\partial t} + \sum_i \frac{\partial V(e_t, \tilde{W})}{\partial e_i} de_t \\
&\quad + \frac{1}{2}\sum_{i,j}[(BW^T G_\sigma)(BW^T G_\sigma)^T]_{ij} \frac{\partial^2 V(e_t, \tilde{W})}{\partial e_i \partial e_j} \qquad (7.53) \\
&= \frac{\partial V}{\partial t} + \sum_i \frac{\partial V(e_t, \tilde{W})}{\partial e_i}\left(Ae_i + B\left(\epsilon_m^\sigma + \tilde{W}^T \bar{\phi}_n^\sigma(\Theta, x)\right)\right) \\
&\quad + \frac{1}{2}\sum_{i,j}[(BW^T G_\sigma)(BW^T G_\sigma)^T]_{ij} \frac{\partial^2 V(e_t, \tilde{W})}{\partial e_i \partial e_j} \qquad (7.54) \\
&= -\dot{W}\Gamma^{-1}\tilde{W} - \frac{1}{2}e^T Q e + \tilde{W}\bar{\phi}_n^\sigma(x) e^T PB + e^T PB\epsilon_m^\sigma(z) \\
&\quad + \frac{1}{2}[(BW^T G_\sigma)(BW^T G_\sigma)^T P] \qquad (7.55)
\end{aligned}$$

Using the weight update law, we can simplify the above expression as follows

$$\mathcal{L}V\left(e_t, \tilde{W}\right) = \frac{1}{2}e^T Q e + e^T PB\epsilon_m^\sigma(z) + \frac{1}{2}[(BW^T G_\sigma)(BW^T G_\sigma)^T P] \tag{7.56}$$

Let $c_1 = 1/2\|P\|\|W\|_F^2\|BG_\sigma\|^2$ and $c_2 = \|PB\|$, then

$$\mathcal{L}V\left(e_t, \tilde{W}\right) \leq -\frac{1}{2}\lambda_{min}(Q)\|e\|^2 + c_2\|e\|\|\epsilon_m^\sigma(z)\| + c_1 \tag{7.57}$$

We know $\sup_{x\in\mathcal{D}}\|\epsilon_m^\sigma(x)\| \leq \bar{\epsilon}^\sigma$, and using the projection operator the outer layer weights $W(t)$ are bounded $\|W\| \leq \mathcal{W}_b$. Therefore using this bound in above expression we can write,

$$\mathcal{L}V\left(e_t, \tilde{W}\right) \leq -\frac{1}{2}\lambda_{min}(Q)\|e\|^2 + c_2\|e\|\bar{\epsilon}^\sigma + c_1 \tag{7.58}$$

Let $c_5^\sigma = c_2\bar{\epsilon}$.

We can define an set outside which the $\mathcal{L}V\left(e_t, \tilde{W}\right) \leq 0$

$$\Theta^\sigma = \left\{\|e\| \geq \frac{c_5^\sigma + \sqrt{\left(c_5^\sigma\right)^2 + 2 * \lambda_{min}(Q)c_1}}{\lambda_{min}(Q)}\right\} \tag{7.59}$$

Therefore, outside the set Θ^σ, $\mathcal{L}V\left(e_t, \tilde{W}\right) \leq 0$ a.s. Using BIBO property of reference model (7.2), when $r(t)$ is bounded, $x_{rm}(t)$ remains bounded within B_m. Solution to (7.1) $x(t) \in \mathcal{D}_x$ a.s. Since this is true for all σ, and because Algorithm guarantees that σ does not switch arbitrarily fast,(7.50) is mean square uniformly ultimately bounded inside of this set Θ^σ a.s.

7.5.2 Sample Complexity of DMRAC

In this section, we will study the sample complexity results from computational theory and show that when applied to a network learning real-valued functions the number of training samples grows at least linearly with the number of tunable parameters to achieve specified generalization error.

The generalization error of a machine learning model is defined as the difference between the empirical loss of the training set and the expected loss of test set [165]. This measure represents the ability of the trained model to generalize well from the learning data to new unseen data, thereby being able to extrapolate from training data to new test data. Hence generalization

error ϵ can be defined as

$$\epsilon = \sup_{x \in \mathcal{D}} \|f(x) - f_{\boldsymbol{\theta}}(x)\| \tag{7.60}$$

where $f_{\boldsymbol{\theta}}(x)$ is neural network output.

Theorem 7.5.2 *Consider a neural network with arbitrary activation functions and an output that takes values in* $[-1, 1]$. *Let* $\mathcal{H}$ *be the hypothesis class characterized by N-weights and each weight represented using k-bits. Then any squared error minimization (SEM) algorithm* $\mathcal{A}$ *over* $\mathcal{H}$, *to achieve a generalization error* (7.60) *admits a sample complexity bounded as follows*

$$m_{\mathcal{A}}(\epsilon, \delta) \leqslant \frac{1}{\epsilon^2}\left(kN \ln 2 + \ln\left(\frac{2}{\delta}\right)\right) \tag{7.61}$$

where N *is total number of tunable weights in the DNN.*

Proof: Let $\mathcal{H}$ be finite hypothesis class of function mapping s.t $\mathcal{H} : \mathcal{X} \to [-1, 1] \in \mathbb{R}^m$ and $\mathcal{A}$ is SEM algorithm for $\mathcal{H}$. Then by Hoeffding inequality for any fixed $f_{\boldsymbol{\theta}} \in \mathcal{H}$ the following event holds with a small probability δ

$$\begin{aligned} & P^m\{|L(\boldsymbol{Z}, \boldsymbol{\theta}) - \mathbb{E}_P(\ell(\boldsymbol{Z}, \boldsymbol{\theta}))| \geq \epsilon\} && (7.62) \\ = \; & P^m\left\{\left|\sum_{i=1}^{m} \ell(\boldsymbol{Z}, \boldsymbol{\theta}) - m\mathbb{E}_P(\ell(\boldsymbol{Z}, \boldsymbol{\theta}))\right| \geq m\epsilon\right\} && (7.63) \\ \leq \; & 2e^{-\epsilon^2 m/2} && (7.64) \end{aligned}$$

Hence

$$\begin{aligned} & P^m\{\forall f_{\boldsymbol{\theta}} \in \mathcal{H}, |\, |L(\boldsymbol{Z}, \boldsymbol{\theta}) - \mathbb{E}_P(\ell(\boldsymbol{Z}, \boldsymbol{\theta}))| \geq \epsilon\} \\ \leq \; & 2|\mathcal{H}|e^{-\epsilon^2 m/2} = \delta && (7.65) \end{aligned}$$

We note that the total number of possible states that is assigned to the weights is $\left(2^k\right)^N$ since there are 2^k possibilities for each weights. Therefore $\mathcal{H}$ is finite and $|\mathcal{H}| \leq 2^{kN}$. The result follows immediately from simplifying Eq-(7.65).

7.6 Persistency of Excitation for S-DMRAC

In multi-layer architecture for uncertainty approximation, it is challenging to ensure the Persistency of Excitation, which result in non P.E deep features and therefore non convergence of the outer-layer weights. One straightforward way to remedy this problem is to inject exogenous perturbations into the dynamics of the gradient descent algorithm so that even the parameters in the hidden layers receive persistent excitation during training.

The traditional procedure to introduce robustness for linear models, which is the addition of regularization term into the loss function, which could be reframed as a method to ensure persistent excitation of the model parameters.

Lets consider $V(\epsilon_i, W) : \mathbb{R}^n \times \mathbb{R}^{k\times m} \to \mathbb{R}$ denote the loss function to be minimized over the set of points $\{x_i\}_{i\in\mathcal{I}}$ and corresponding target values $\{f(x_i)\}_{i\in\mathcal{I}} \in \mathbb{R}^m$. consider the linear regression problem with the squared-error loss

$$V(\epsilon_i, W) = \frac{1}{2}\sum_{i\in\mathcal{I}} \epsilon_i^T \epsilon_i + \mathcal{R}(W) \tag{7.66}$$

The regularization term $\mathcal{R}(W)$ is usually assumed to impose some prior knowledge about W, and it is chosen as a convex function of some norm of W, such as $\|W\|_2^2$, or $\|W\|_1$. This forces the parameters to be close to zero in some norm and restricts the class of functions that can be estimated by the model.

For the ridge regression problem corresponding to the choice $\mathcal{R}(W) = \lambda\|W\|_2^2$ for any $\lambda > 0$

$$V(\epsilon_i, W) = \frac{1}{2}\sum_{i\in\mathcal{I}} \left\|W^T\Phi(x_i) - f(x_i)\right\|_2^2 + \lambda\|W\|_2^2 \tag{7.67}$$

Adding regularization is equivalent adding m pseudo-measurements. We can rewrite the regularization term as follows

$$\lambda\|W\|_2^2 = \sum_{i=1}^{m} \left\|W^T(\sqrt{\lambda}e_i) - 0\right\|_2^2 \tag{7.68}$$

where is $e_i \in \mathbb{R}^k$ is stnadard basis vector. Alternatively, an equivalent prob-

lem could also be formulated as

$$V(\epsilon_i, W) = \frac{1}{2}\mathbb{E}_{\xi_i}\left(\sum_{i\in\mathcal{I}} \left\|W^T(\Phi(x_i) + \xi_i) - f(x_i)\right\|_2^2\right) \tag{7.69}$$

Where $\xi_i \in \mathbb{R}^k$ is and random vector, with

$$\mathbb{E}(\xi_i) = 0, \;\; \mathbb{E}(\xi_i x i_i^T) = \lambda I$$

In particular, the probability distributions for ξ_i could be chosen to be discrete for practical purposes, and the ridge regression could be considered as perturbed version least square regression. However, such methods of adding artificial noise to state in adaptive control might lead to chattering in control and therefore in system response and hence not desirable.

We will now show that Bayesian-Deep neural network is a natural way of adding regularization and the stochastic nature of network weights lead to desirable property of persistency of excitation.

The KL-objective for posterior inference given data in training Bayesian-deep neural network is follows,

$$\mathcal{L}(Z, \boldsymbol{\theta}, \boldsymbol{\zeta}) = \mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\zeta})}\left(\log\left(p(Z|\boldsymbol{\theta})\right)\right) - KL\left(q(\boldsymbol{\theta}|\boldsymbol{\zeta})\|p(\boldsymbol{\theta})\right) \tag{7.70}$$

While maximizing $\mathcal{L}(Z, \boldsymbol{\theta}, \boldsymbol{\zeta})$, we are maximizing the model likelihood given data Z and minimizing KL-divergence between posterior and prior on parameters, which is akin to adding regularization.

Uncertainty in predictions arise from two sources; uncertainty in weights called Epistemic uncertainty and uncertainty coming from the finite excitation in training data know as Aleatoric uncertainty. Epsitemic uncertainty can grows smaller, if we have more training data. Consequently, epistemic uncertainty is higher in regions of no or little training data and lower in regions of more training data. Epistemic uncertainty is covered by the variational posterior distribution. Aleatoric uncertainty is covered by the probability distribution used to define the likelihood function, it cannot be reduced if we get more data. The total uncertainty or randomness in the feature vector due to Epistemic and Aleatoric uncertainty, lead to favorable property of persistency of excitation of Φ_n^σ

Theorem 7.6.1 *Let the Bayesian deep network $f_{\boldsymbol{\theta}}$ is trained over data $Z^{p_{max}} =$*

$\{\{x_i, y_i\} \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^{p_{max}}$, *such that data is collected using kernel independence test. Then the features* $\Phi_n^\sigma = f_{\boldsymbol{\theta}} \backslash \theta_n$ *is persistently exciting.*

Proof We use autocovariance argument to prove the that the stochastic features Φ_n^σ are persistently exciting i.e iff

$$R_{\Phi_n^\sigma}(0) > 0 \tag{7.71}$$

Implies the integral for any given $t \geq t_0$ and over any interval $[t, t+T]$ for any $T > 0$ satisfies,

$$\int_t^{t+T} \Phi_n^\sigma(\tau) \Phi_n^\sigma(\tau)^T d\tau \geq \gamma I \tag{7.72}$$

To prove the fact that deep features of S-DMRAC are PE, we show that the autocovariance $R_{\Phi_n^\sigma}(0)$ exists and $R_{\Phi_n^\sigma}(0) > 0$.

Since the stochastic feature vector are realization of distribution defined as

$$\Phi_n^\sigma(x) \sim \mathcal{N}\left(\bar{\Phi}_n^\sigma(x), \left[G_\sigma(x) G_\sigma(x)^T\right] | \boldsymbol{\theta}\right) \tag{7.73}$$

Therefore the autocovariance of feature vector Φ_n^σ can be defined as,

$$\begin{aligned} R_{q_\theta(\Phi_n^\sigma | x, \theta)}\left(\Phi_n^\sigma(x) | x\right) &= \mathbb{E}_{q_\theta}\left[\left(\Phi_n^\sigma - \mathbb{E}(\Phi_n^\sigma)\right)^2\right] & (7.74) \\ &= \mathbb{E}_{q_\theta}\left(\Phi_n^\sigma \Phi_n^{\sigma T}\right) - \mathbb{E}_{q_\theta}\left(\Phi_n^\sigma\right) \mathbb{E}_{q_\theta}\left(\Phi_n^\sigma\right)^T & (7.75) \end{aligned}$$

Where $q_\theta(\Phi_n^\sigma | x, \theta)$ is the variational predictive posterior distribution defined as,

$$q_{\boldsymbol{\zeta}(\Phi_n^\sigma | x, \theta)} = p(\Phi_n^\sigma | x, \theta) q(\boldsymbol{\theta} | \boldsymbol{\zeta}) \tag{7.76}$$

and $p(\Phi_n^\sigma | x, \theta)$ is the posterior distribution on feature given a input x and network parameters $\boldsymbol{\theta}$

$$p(\Phi_n^\sigma | x, \theta) = \mathcal{N}\left(\bar{\Phi}_n^\sigma(x), \left[G_\sigma(x) G_\sigma(x)^T\right] | \boldsymbol{\theta}\right) \tag{7.77}$$

Using the above definitions of conditional distributions, the autocovariance term (7.75) can be decomposed into ***Aleatoric*** and ***Epistemic*** uncertainties

as follows [166]:

$$
\begin{aligned}
& R_q\left(\Phi_n^\sigma(x)|x\right) = \mathbb{E}_q\left(\Phi_n^\sigma \Phi_n^{\sigma T}\right) - \mathbb{E}_q\left(\Phi_n^\sigma\right)\mathbb{E}_q\left(\Phi_n^\sigma\right)^T \\
\Rightarrow \quad & \int_\Theta \left[diag\left(\mathbb{E}_{p(\Phi_n^\sigma|x,\theta)}\left(\Phi_n^\sigma\right)\right) - \mathbb{E}_{p(\Phi_n^\sigma|x,\theta)}\left(\Phi_n^\sigma\right)\mathbb{E}_{p(\Phi_n^\sigma|x,\theta)}\left(\Phi_n^\sigma\right)^T\right] q_{\boldsymbol{\zeta}}(\boldsymbol{\theta}) d\boldsymbol{\theta} \\
+ \quad & \int_\Theta \left[\mathbb{E}_{p(\Phi_n^\sigma|x,\theta)}\left(\Phi_n^\sigma\right) - \mathbb{E}_{q_\zeta(\Phi_n^\sigma)}\left(\Phi_n^\sigma\right)\right]\left[\mathbb{E}_{p(\Phi_n^\sigma|x,\theta)}\left(\Phi_n^\sigma\right) - \mathbb{E}_{q_\zeta(\Phi_n^\sigma)}\left(\Phi_n^\sigma\right)\right]^T q_{\boldsymbol{\zeta}}(\boldsymbol{\theta}) d\boldsymbol{\theta}
\end{aligned}
\tag{7.78}
$$

The first term of the predictive variance of the variational posterior distribution (7.78) in known as Aleatoric uncertainty coming from finite excitation or variation in the training data. The expectation is with respect to predictive posterior distribution $p(\Phi_n^\sigma|x,\theta)$ integrated over all parameters $\boldsymbol{\theta}$.

The second term of the predictive variance of the variational posterior distribution (7.78) is the epistemic uncertainty. That is uncertainty in predictions that arise from the uncertainty in weights. This kind of uncertainty can be reduced if we get more data. Consequently, epistemic uncertainty is higher in regions of no or little training data and lower in regions of more training data. Epistemic uncertainty is covered by the variational posterior distribution over parameters.

If we chose the training data such that Aleatoric uncertainty is non zero, then we can show that the autocovariance of features Φ_n^σ is non zero every where. We using kernel independence test (7.18) to qualify the data points which are sufficiently distinct from the stored data in buffer $\{Z : x, y \in \mathcal{X}, \mathcal{Y}\}_{i=1}^{p_{max}}$ hence ensuring Aletoric uncertainty is non zero and due to fact that the network weights are stochastic the epistemic uncertainty is non zero anywhere else where training data is not available.

Hence using stochastic Bayesian network and training the network over data points chosen according to kernel independence test (7.18) we can ensure the total variance $R_q\left(\Phi_n^\sigma(x)|x\right) > 0$. An there for due to Lemma-5.6.4 we know that given Φ_n^σ is P.E

7.6.1 Uniform boundedness of the Outer-Layer parameters

Consider the problem of parameter estimation using a linear estimator with stochastic deep features. We learn a given function by minimizing its squared

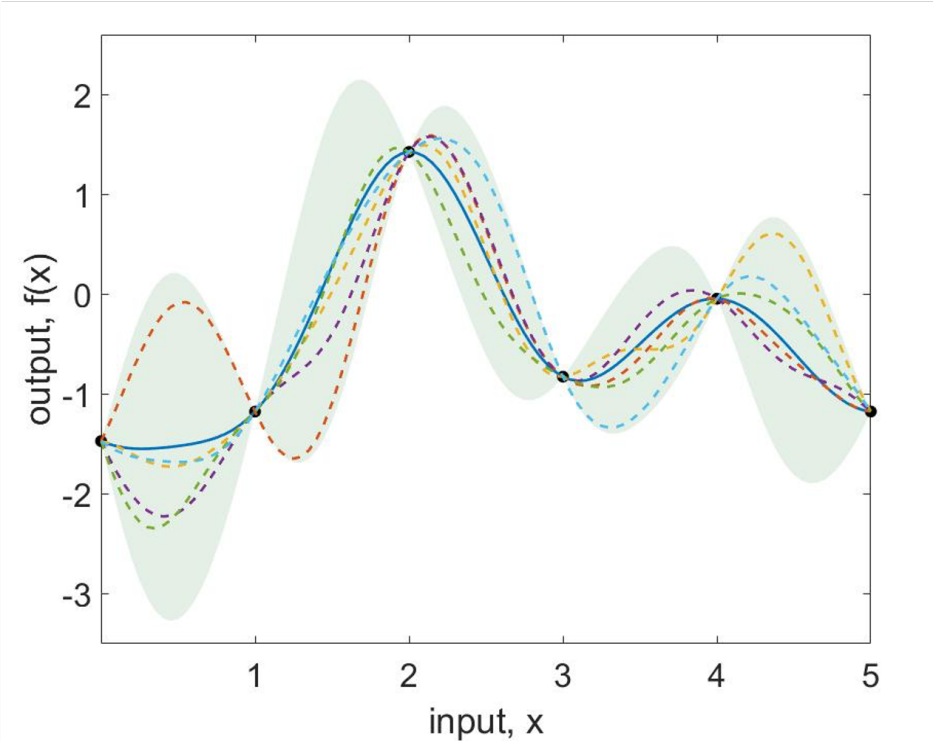


Figure 7.3: output of Bayesian Neural Network: Shaded area is Epistemic uncertainty and Training points (black dots) are chosen according to kernel independence test to ensure non zero Aleatoric uncertainty

loss error.

Let us assume the function to be estimated be $f : \mathbb{R}^n \to \mathbb{R}^m$. Let the true function f be of form

$$f(x) = W^{*T}\bar{\Phi}_n^\sigma(x) + \epsilon(x) \tag{7.79}$$

Where the unknown function be linearly parameterized in unknown ideal weight $W^* \in \mathbb{R}^{k\times m}, \forall\sigma$ and $\bar{\Phi}_n^\sigma(x) : \mathbb{R}^n \to \mathbb{R}^k$ such that

$$\bar{\Phi}_n^\sigma(x) = \mathop{\mathbb{E}}_{\theta}\left(\Phi_n^\sigma(x)\right)$$

Where $\Phi_n^\sigma(x)$ is stochastic nonlinear continuously differentiable basis function which is drawn from distribution (7.73)

Let the $W(t)$ be the online estimate of true ideal weights W^*, therefore the online estimate of $f(x)$ can be represented by mapping $\nu : \mathbb{R}^n \to \mathbb{R}^m$ such

that,

$$\begin{aligned} \nu(x) &= W^T \Phi_n^\sigma(x) && (7.80) \\ &= W^T \left(\bar{\Phi}_n^\sigma(x) + G_\sigma \xi \right) && (7.81) \end{aligned}$$

Let $e(t)$ be defined as,

$$\begin{aligned} e(t) &= f(x) - \nu(x) && (7.82) \\ &= W^{*T} \bar{\Phi}_n^\sigma(x) + \epsilon(x) - W^T \left(\bar{\Phi}_n^\sigma(x) + G_\sigma \xi \right) && (7.83) \\ &= \tilde{W}^T \bar{\Phi}_n^\sigma(x) + \epsilon(x) - W^T \left(G_\sigma \xi \right) && (7.84) \end{aligned}$$

then then the parameter update using gradient descent algorithm is given as

$$\dot{W} = -\Gamma \bar{\Phi}_n^\sigma(x) e(t) \tag{7.85}$$

The parameter error dynamics can be found by differentiating $\tilde{W}$ and using equation (7.85) as,

$$\begin{aligned} \dot{\tilde{W}} &= -\Gamma \bar{\Phi}_n^\sigma(x) e(t) && (7.86) \\ &= -\Gamma \bar{\Phi}_n^\sigma(x) \left(\tilde{W}^T \bar{\Phi}_n^\sigma(x) + \epsilon(x) - W^T \left(G_\sigma \xi \right) \right) && (7.87) \\ &= -\Gamma \bar{\Phi}_n^\sigma(x) \bar{\Phi}_n^\sigma(x)^T \tilde{W} - \Gamma \bar{\Phi}_n^\sigma(x) \epsilon(x) - \Gamma \bar{\Phi}_n^\sigma(x) \left(G_\sigma \xi \right)^T \tilde{W} && (7.88) \end{aligned}$$

This is a linear time varying differential equation in $\tilde{W}$. Furthermore, note that if PE Condition using Theorem-7.6.1 is satisfied, then $\tilde{W}$ is asymptotically bounded near zero solution.

Theorem 7.6.2 *Consider the system model given by equation* (7.79)*, the online estimation model given by equation* (7.81)*, the stochastic deep feature gradient descent weight update law* (7.88)*, and assume that the regressor function* $\Phi_n^\sigma(x)$ *is continuously differentiable and that the measurements* $\Phi_n^\sigma(x) \in \mathcal{D}$ *where* $\mathcal{D} \in \mathbb{R}^m$ *is a compact set. If the* $\Phi_n^\sigma(x)$ *satisfy the PE condition, then the zero solution of the weight error dynamics of equation* (7.81) *is globally uniformly ultimately bounded.*

Proof Consider a quadratic Lyapunov candidate given by

$$V(\tilde{W}) = \frac{1}{2} tr \left(\tilde{W}^T \Gamma^{-1} \tilde{W} \right) \tag{7.89}$$

Such that $V(0) = 0$ and $V(\tilde{W}) > 0, \forall \tilde{W} \neq 0$. Since $V(\tilde{W})$ is quadratic, letting $\lambda_{min}(.)$ and $\lambda_{max}(.)$ denote the operators that return the minimum and maximum eigenvalue of a matrix, we have:

$$\lambda_{min}\left(\Gamma^{-1}\right)\left\|\tilde{W}\right\|_F^2 \leq V(\tilde{W}) \leq \lambda_{max}\left(\Gamma^{-1}\right)\left\|\tilde{W}\right\|_F^2.$$

Differentiating with respect to time along the trajectory (7.88), we have

$$\dot{V}(\tilde{W}) = tr\left(\tilde{W}^T\Gamma^{-1}\dot{\tilde{W}}\right) \tag{7.90}$$

$$= \tilde{W}^T\Gamma^{-1}\left(-\Gamma\bar{\Phi}_n^\sigma(x)\bar{\Phi}_n^\sigma(x)^T\tilde{W} - \Gamma\bar{\Phi}_n^\sigma(x)\epsilon(x) - \Gamma\bar{\Phi}_n^\sigma(x)\left(G_\sigma\xi\right)^T\tilde{W}\right) \tag{7.91}$$

Using the definition of empirical estimate of $\bar{\Phi}_n^\sigma(x)$ in (7.40), the above expression cn be rewritten as,

$$\dot{V}(\tilde{W}) = -\tilde{W}^T\bar{\Phi}_n^\sigma(x)\bar{\Phi}_n^\sigma(x)^T\tilde{W} - \bar{\Phi}_n^\sigma(x)\tilde{W}^T\epsilon(x) - \tilde{W}^T\bar{\Phi}_n^\sigma(x)\left(G_\sigma\xi\right)^T\tilde{W} \tag{7.92}$$

$$\leq -\lambda_{max}(\Omega)\left\|\tilde{W}\right\|^2 - \left\|\bar{\Phi}_n^\sigma\right\|_\infty \mathcal{W}_b\bar{\epsilon}^\sigma \tag{7.93}$$

$$\leq -\frac{\lambda_{max}(\Omega)}{\lambda_{min}\left(\Gamma^{-1}\right)}V(\tilde{W}) - \left\|\bar{\Phi}_n^\sigma\right\|_\infty \mathcal{W}_b\bar{\epsilon}^\sigma \tag{7.94}$$

Where Ω is defined as follows,

$$\Omega = \frac{1}{N}\sum_{i,j=1}^{N}\left(\Phi_{n,i}^\sigma(x)\bar{\Phi}_{n,j}^\sigma(x)^T + \Phi_{n,i}^\sigma(x)(G_\sigma\xi_j)^T\right) \tag{7.95}$$

Using the P.E condition $\Phi_{n,i}^\sigma(x)$, we know that $\lambda_{max}(\Omega) > 0$

Hence from Lyapunov stability theory if

$$\left\|\tilde{W}\right\| \geq \sqrt{\frac{\left\|\bar{\Phi}_n^\sigma\right\|_\infty \mathcal{W}_b\bar{\epsilon}^\sigma}{\lambda_{max}(\Omega)}} \tag{7.96}$$

we have $\dot{V}(\tilde{W}) \leq 0$. Therefore the set $W_\delta = \left\{\left\|\tilde{W}\right\| \leq \sqrt{\frac{\left\|\bar{\Phi}_n^\sigma\right\|_\infty \mathcal{W}_b\bar{\epsilon}^\sigma}{\lambda_{max}(\Omega)}}\right\}$ is positively invariant, hence $\tilde{W}$ approaches the bounded near zero solution exponentially fast and remain are ultimately bounded.

If Theorem-7.6.2 holds, then the adaptive weights $W(t)$ will approach exponentially fast and remain bounded in a compact neighborhood of the ideal weights W^*.

7.7 Evaluation of DMRAC and S-DMRAC Controller

We evaluate the DMRAC and S-DMRAC controller using Wing-Rock system dyanamics in simulation and through flight testing using Quadrotor in Vicon arena. The simulation and flight test results are discussed in the following sections.

7.7.1 S-DMRAC controller evaluation using Wing-Rock system Dynamics

The initial simulation result we present are the S-DMRAC controller evaluation using Wing-Rock dynamics. The details of Wing-Rock dynamics are provided in the Section-5.8. The aim of the controller is to follow the a step reference command under unknown nonlinear uncertainties.

The controller is designed to track a stable reference commands $r(t)$. The goal of the experiment is to compare the tracking performance of S-DMRAC controller with MRAC controller when the domain operation is unknown. We find that 100 centers are sufficient for a good on-line approximation of the uncertainty. We use the same reference model for both controllers. The learning rate for high gain S-DMRAC is chosen to be $\Gamma = 10$, Kernel bandwidth is chosen as $\mu = 0.05$ and tolerance threshold for kernel independence test is selected to be $\epsilon_{tol} = 0.1$ for updating the basis vector $\mathcal{BV}(\sigma)$.

Figure-7.7 and Fig-7.8 show the closed loop system performance in tracking the reference signal for GP-MRGeN and MRAC controller. Note that the control response of the proposed adaptive controller is clearly superior as compared to the standard adaptive controller. GP-MRGeN uses the generative model target at an only small number of discrete time steps to learn the underlying model of the uncertainty. Also, the GP-MRGeN estimate is noise free even when the target values are noisy. The presented controller achieves tighter tracking with smaller tracking error as shown in Fig-7.9 compared to MRAC controller. Figure-6.5 show the total control profile for both the con-

trollers. It can be observed the control history for GP-MRGeN has smaller peaks and less noisy due to smoothing property of the GP estimation. This denoising of the controller demonstrates that proposed method leads to robust adaptation without incurring high-frequency oscillations in the control response. Figure-6.5 shows the network performance in estimating the uncertainty. The GP-MRGeN estimate is noise free and closely approximates the true uncertainty compared to MRAC estimate. The blobs in Fig-6.5 show the time step at which the generative model MRGeN was updated and queried for inferring the posterior GP model over data.

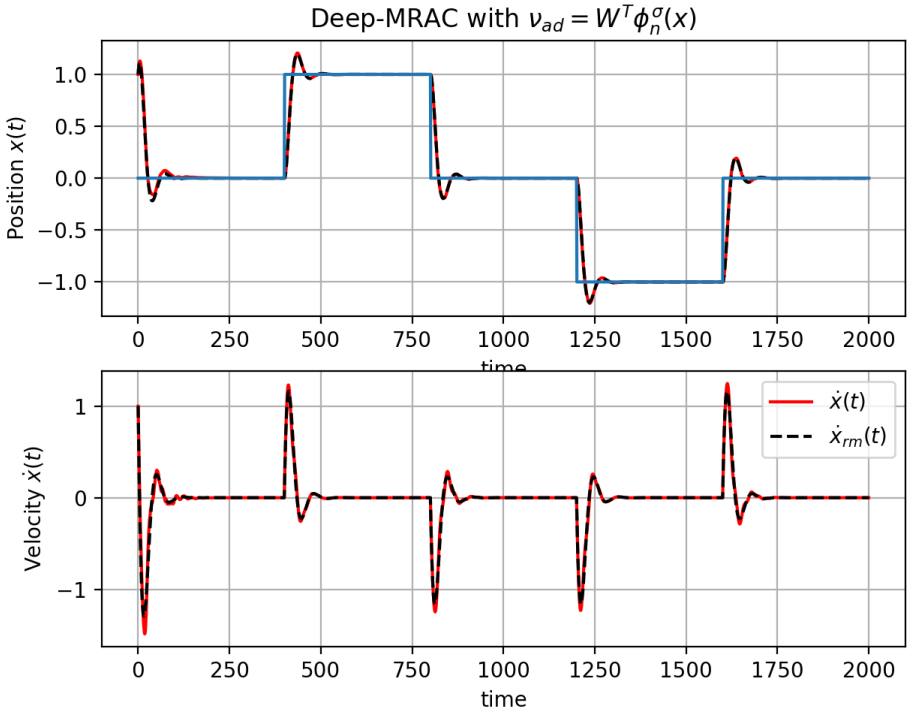


Figure 7.4: Wind-Rock Roll and Roll-rate for step command following using S-DMRAC controller

7.7.2 DMRAC Controller Evaluation on simulated 6-DOF Quadrotor Model

In this section, we will evaluate the presented DMRAC adaptive controller using a 6-DOF Quadrotor model for the reference trajectory tracking problem. The quadrotor model is completely described by 12 states, three position,

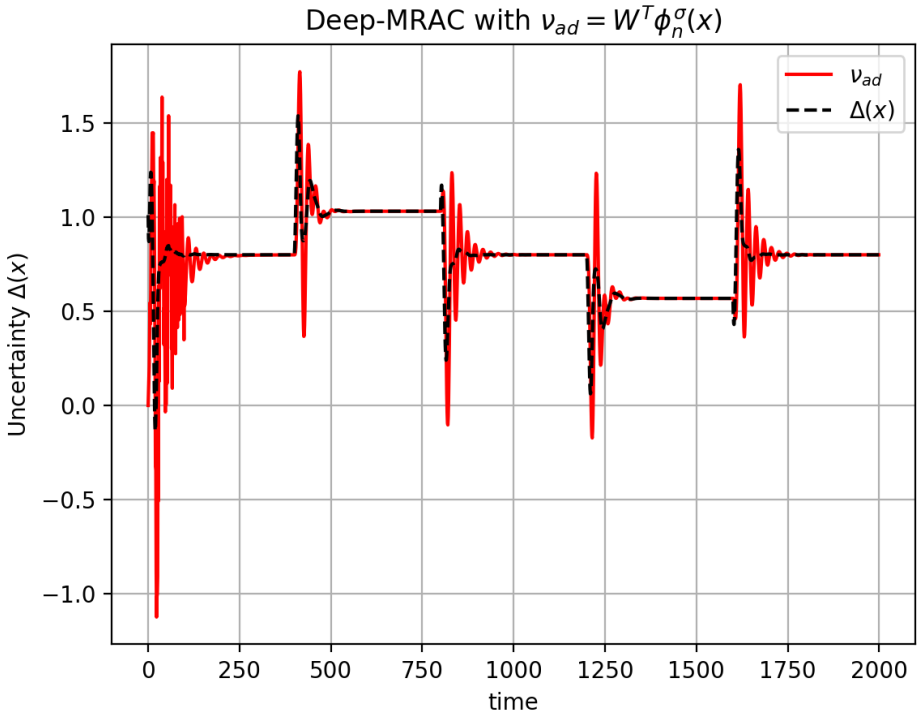


Figure 7.5: Uncertainty Estimate using S-DMRAC Controller

and velocity in the North-East-Down reference frame and three body angles and angular velocities. The full description of the dynamic behavior of a Quadrotor is beyond the scope of this worl, and interested readers can refer to [167] and references therein.

The control law designed treats the moments and forces on the vehicle due to unknown true inertia/mass of the vehicle and moments due to aerodynamic forces of the crosswind, as the unmodeled uncertainty terms and are captured online through DNN adaptive element. The outer-loop control of the quadrotor is achieved through Dynamic Inversion (DI) controller, and we use DMRAC for the inner-loop attitude control. A simple wind model with a boundary layer effect is used to simulate the effect of crosswind on the vehicle.

A second-order reference model with natural frequency $4rad/s$ and damping ratio of 0.5 is used. Further stochasticity is added to the system by adding Gaussian white noise to the states with a variance of $\omega_n = 0.01$. The simulation runs for 150secs and uses time step of $0.05s$. The maximum number of points (p_{max}) to be stored in buffer $\mathcal{B}$ is arbitrarily set to 250, and

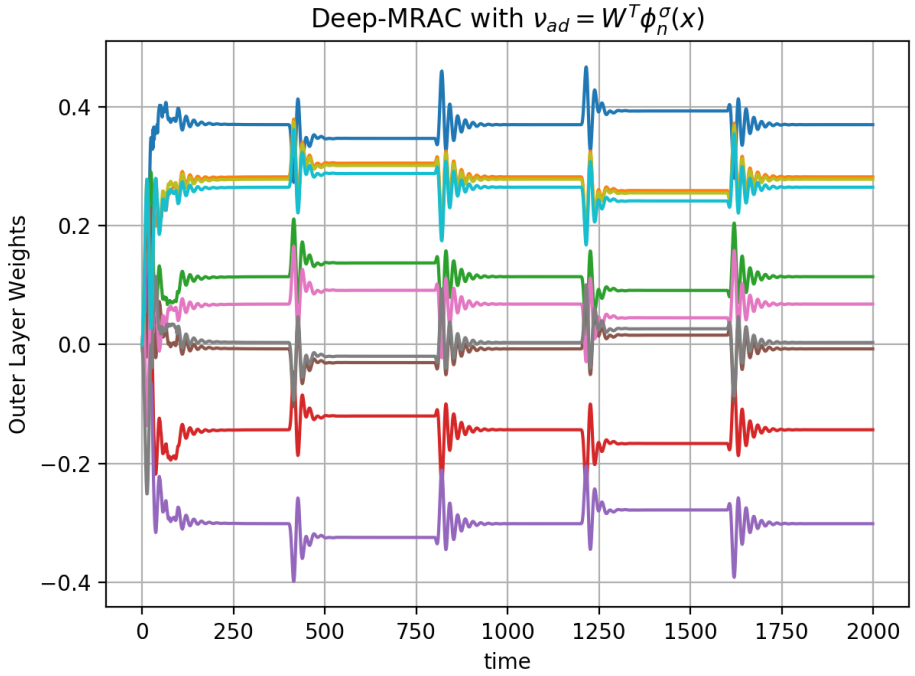


Figure 7.6: Outer-Layer S-DMRAC weights history for Wing-Rock control

SVD maximization algorithm is used to cyclically update $\mathcal{B}$ when the budget is reached, for details refer [163].

The controller is designed to track a stable reference commands $r(t)$. The goal of the experiment is to evaluate the tracking performance of the proposed DMRAC controller on the system with uncertainties over an unknown domain of operation. The learning rate for D-MRGeN network and DMRAC-DNN networks are chosen to be $\Gamma = 0.5 I_{6\times 6}$ and $\eta = 0.01$. The DNN network is composed of 2 hidden layers with $200, 100$ neurons and with tan-sigmoid activations, and output layer with linear activation. We use "Levenberg-Marquardt backpropagation" [168] for updating DNN weights over 100 epochs. Tolerance threshold for kernel independence test is selected to be $\zeta_{tol} = 0.2$ for updating the buffer $\mathcal{B}$.

Figure-7.7 and Fig-7.8 show the closed loop system performance in tracking the reference signal for DMRAC controller and learning retention when used as the feed-forward network on a similar trajectory (Circular) with no learning. We demonstrate the proposed DMRAC controller under uncertainty and without domain information is successful in producing desired reference tracking. Since DMRAC, unlike traditional MRAC, uses DNN

for uncertainty estimation is hence capable of retaining the past learning and thereby can be used in tasks with similar features without active online adaptation Fig-7.8. Whereas traditional MRAC which is "pointwise in time" learning algorithm and cannot generalize across tasks. The presented controller achieves tighter tracking with smaller tracking error in both outer and inner loop states as shown in Fig-7.8 and Fig-7.9 in both with adaptation and as a feed-forward adaptive network without adaptation. The Training, Testing and Validation error over the data buffer for DNN, demonstrate the network performance in learning a model of the system uncertainties and its generalization capabilities over unseen test data.

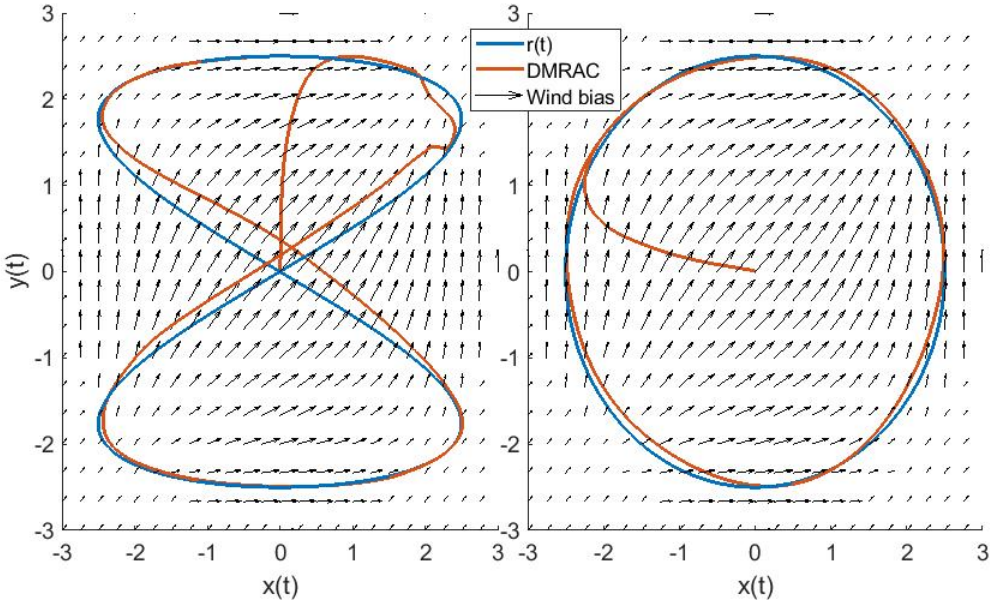


Figure 7.7: DMRAC Controller Evaluation on 6DOF Quadrotor dynamics model, DMRAC Controller on Trajectory tracking control with learning and as frozen feed-forward network (Circular Trajectory)

7.7.3 Flight Test Results-Comparison between PID, MRAC and DMRAC

This section provides the hardware and vehicle details for evaluation of DMRAC for flight control of quadrotor system. The flight tests were done on a commercially available quadcopter platform, Parrot Mambo Mini-Drone 7.12 in a VICON facility. This drone was chosen for flight testing due to its small

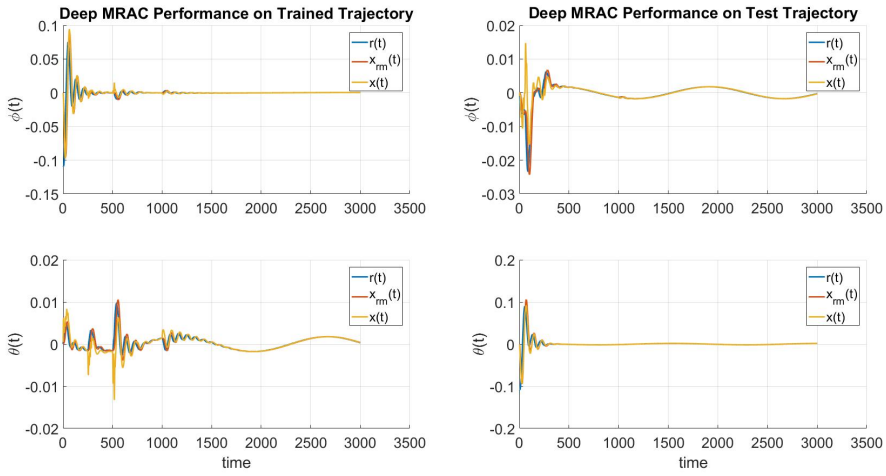


Figure 7.8: Closed-loop system response in roll rate $\phi(t)$ and Pitch $\theta(t)$

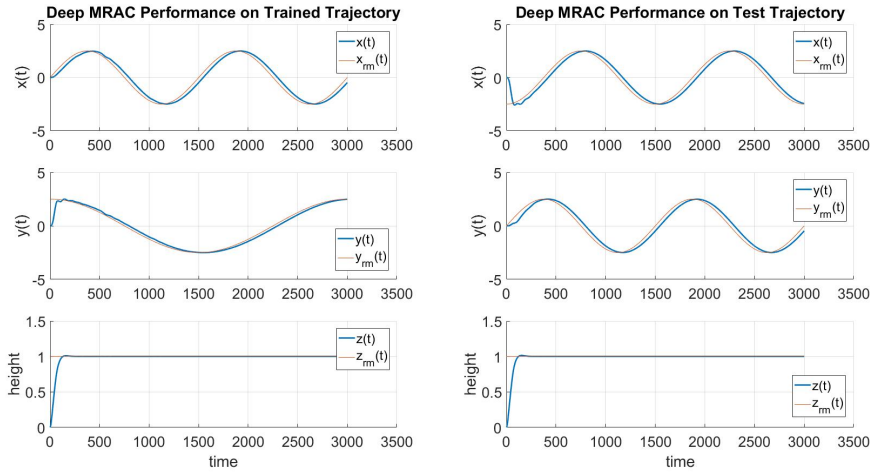


Figure 7.9: Position Tracking performance of DMRAC controller with learning and in learning retention test over Circular Trajectory (b)DNN Training

size, low cost, and relative ease of implementing and testing algorithms. The implementation of DMRAC was done using our onboard, off-board architecture as shown in Figure-7.11. The off-board component consists of software modules which run entirely on a host computer and are responsible for training the neural network using data received from the drone via UDP bluetooth protocol. The data received are the drone's current roll, pitch, yaw angles and the angular rates, and DMRAC estimate of the total uncertainty. The above information is stored in a memory buffer and random batches from this buffer are used to train the deep neural network on the off-board computer.

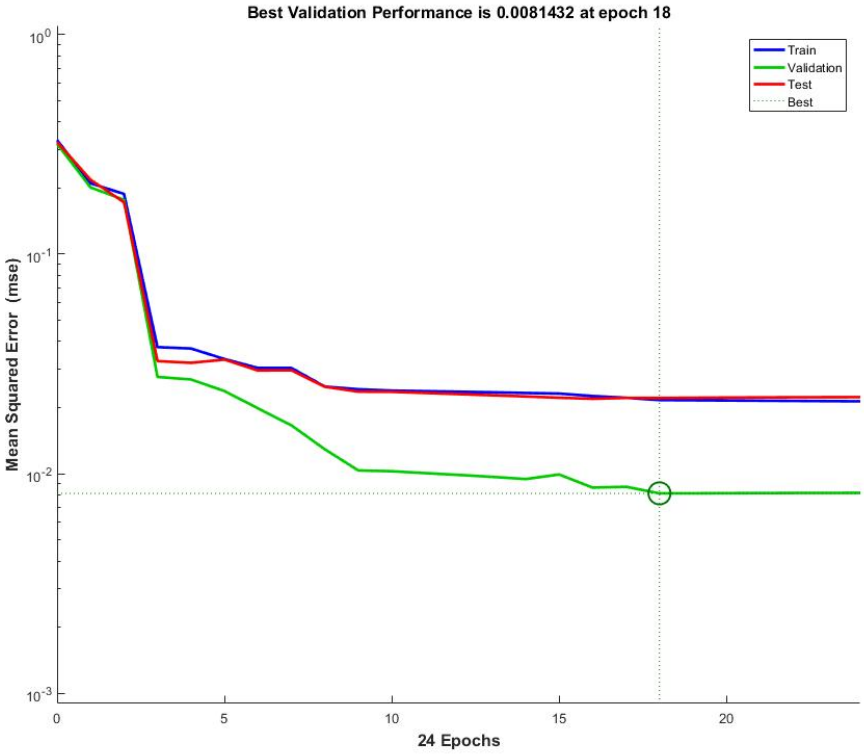


Figure 7.10: (a)Position Tracking performance of DMRAC controller with learning and in learning retention test over Circular Trajectory (b)DNN Training, Test and Validation performance

	Wing-Rock	Quadrotor	Quadrotor-Flight
State Space	2	12	6
Control Space	1	3	3
Num. of layers	5	5	3
Neuron in Hidden Layer	$[100, 200, 100]$	$[100, 200, 100]$	$[20]$
Activations	tanh	tanh	tanh
Final Layer Activation	Linear	Linear	Linear
Outer Layer Learning rate	100	5	5
Feature Learning rate (α)	0.001	0.001	0.001
Buffer Size	10000	10000	10000
Batch size	50	50	50
Epochs	10	10	1

Table 7.1: DMRAC Network details and Network learning parameter details

Moreover, the host computer also receives drone's current position inside the Vicon Arena from the Vicon system via Wi-Fi which is then communicated

to the drone using UDP protocol. Also, as soon as the inner layer weights are

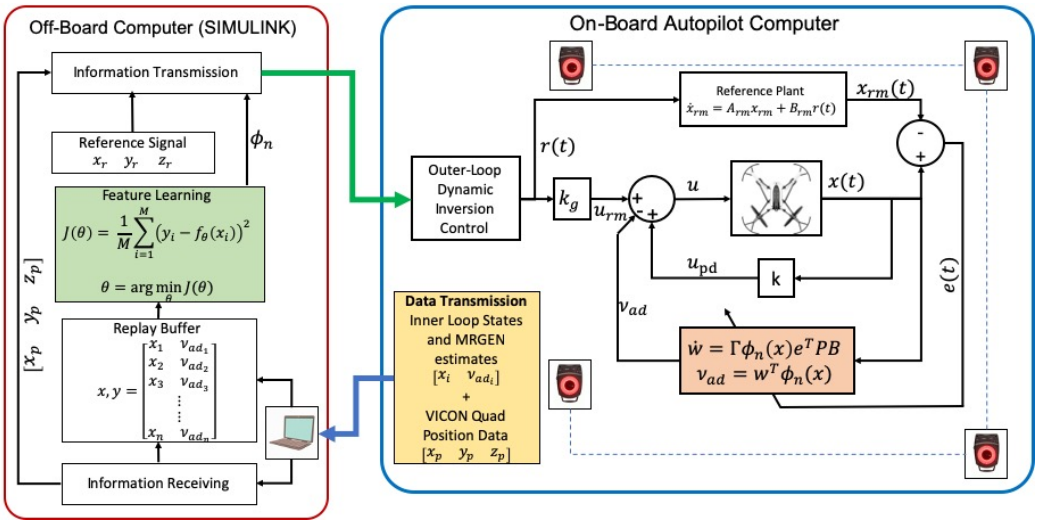


Figure 7.11: Our On-board - Off-board Implementation of Deep Model reference Adaptive controller for Quadrotor control. The system ensures that the outer layer weights can be updated onboard in a manner to ensure Lyapunov stability (at 200Hz), while the inner layer weights are updated asynchronously off-board to improve learning performance.

computed using the data collected from the drone, the host computer communicates those weights back to the vehicle, which are then used in updating the DMRAC controller to produce adaptive torque. The onboard software modules are responsible for state estimation along with computing the entire control effort. The main reason behind implementing DMRAC in the above way is because of the limited processing power of the drone. It is possible that if a more powerful processor can be embedded on-board, the algorithm can be implemented completely onboard. Even in this case however, it is desirable to separate the inner layer learning so that the outer layer weights can be updated to ensure Lyapunov stability and inner layer weight only updated whenever the test data loss values are sufficiently low. Furthermore, in this case, an interesting extension could be to use the presented on-board - off board architecture to asynchronously update the inner-layer weights using data from multiple drones simultaneously. This is left for future work.

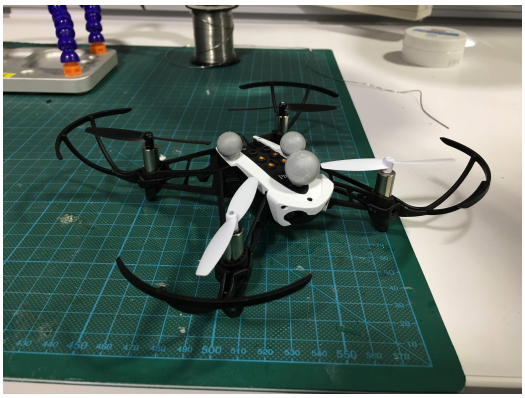


Figure 7.12: The flight platform on which DMRAC is implemented

7.7.4 Flight tests on a figure of 8 trajectory

We experiment with quadrotor to evaluate the controller performance on a figure of 8 reference trajectory tracking. The tuned values of feedback, feedforward gains, and learning rates are kept fixed throughout the following experiments. Figures-3 and Fig-4 show the comparison between each controllers' performance on the nominal baseline task with no external disturbance or faults. Since each controller is tuned to achieve best performance over the given task, the difference between the controller is negligible, and all three controllers perform equally well.

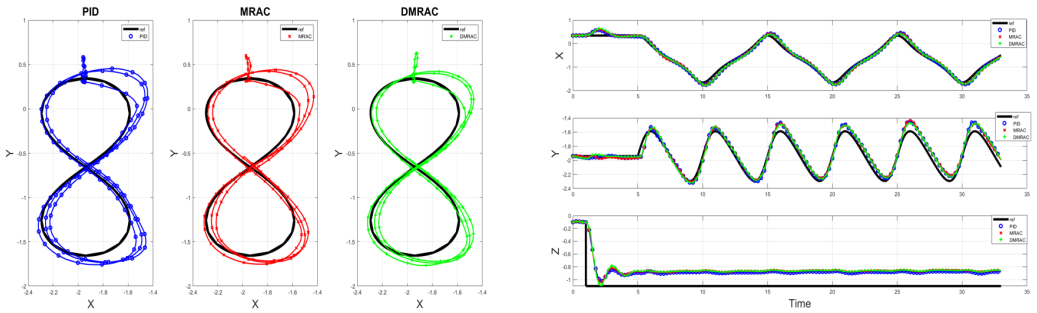


Figure 7.13: Tracking performance on a simple figure of 8 trajectory

7.7.5 Reference trajectory tracking with wind bias

The second task used to evaluate the controllers' performance is reference tracking on a figure of 8 trajectories with wind bias. This task is designed to assess the performance of all three controllers in the case of external disturbance. A wind bias disturbance is simulated, using a fan placed near the

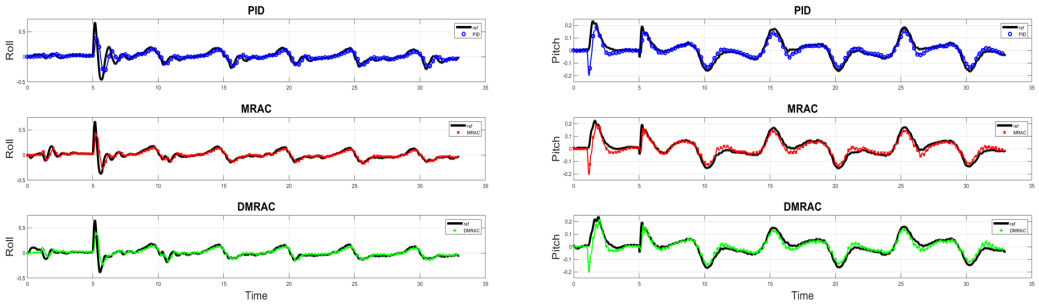


Figure 7.14: Tracking of reference model's roll and pitch signal for a simple figure of 8 trajectory

drone's initial position, and oriented to cause the disturbance along the X-axis. Figures-5 and Fig-6 show a comparison of each controllers' performance on this task. We can observe that DMRAC is more robust and achieves much closer tracking compared to other two algorithms. The tracking error for the DMRAC controller for the inner loop reference roll and pitch states is also much lower compared to MRAC or PID refer Figure-7.16.

We also demonstrate the proposed controller can handle abrupt changes in reference commands. At around 5secs mark, the reference signal changes from step input for height control in the z-axis to figure of 8 trajectory command in the x-y plane. The PID controller experiences high oscillation, whereas MRAC and DMRAC handle the switch much smoothly.

Further we have repeated the above experiment with medium and high wind bias. Following results on a tracking task under external disturbance demonstrates that DMRAC outperforms MRAC and PID refer Figure-7.17-7.20.

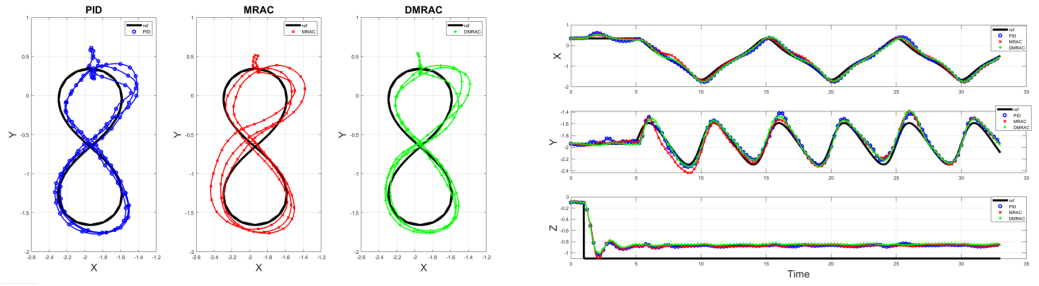


Figure 7.15: Tracking performance under low wind bias

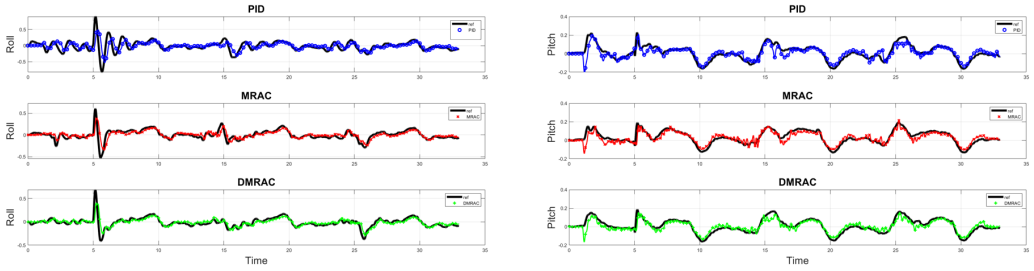


Figure 7.16: Tracking of reference model's roll and pitch signal under low wind bias

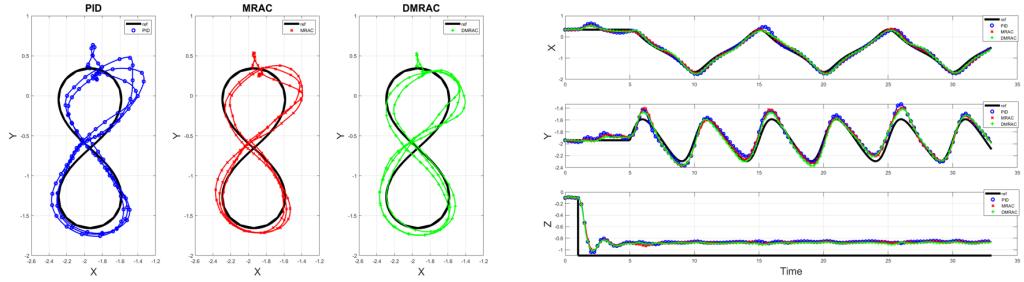


Figure 7.17: Tracking performance under medium wind bias

7.7.6 Reference trajectory tracking under a highly nonlinear disturbance

In this experiment, to simulate a highly nonlinear and unpredictable disturbance, we attach a piece of cloth underneath the air-frame of the quadrotor and is subjected to high wind bias. Erratic flapping of the cloth produced unpredictable disturbance torques and forces. We designed this experiment to push each controller to its limits and was repeated three times to demonstrate the repeatability. We observed that PID failed in all the three experiments, whereas both MRAC and DMRAC gave a stable performance. However, the tracking error observed for MRAC was relatively higher when compared to

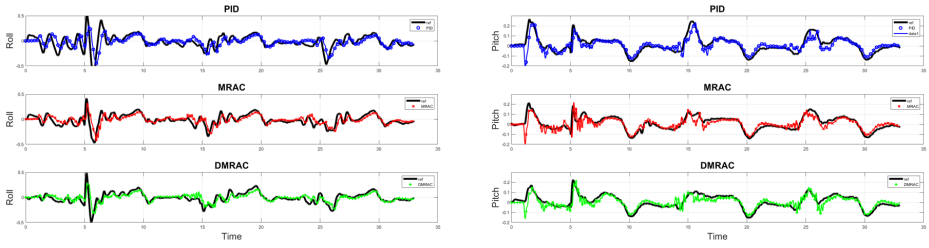


Figure 7.18: Tracking of reference model's roll and pitch signal under medium wind bias

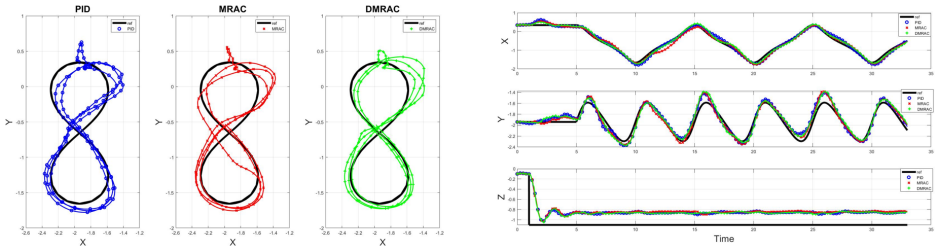


Figure 7.19: Tracking performance under high wind bias

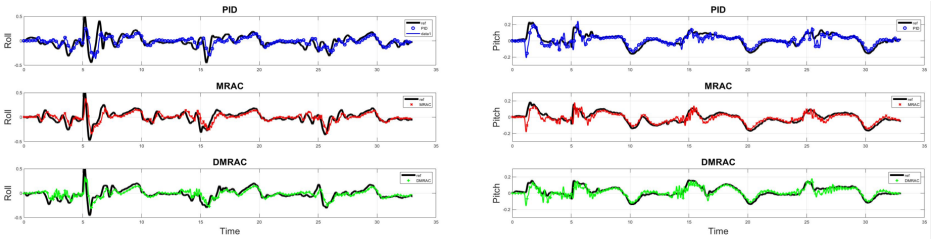


Figure 7.20: Tracking of reference model's roll and pitch signal under high wind bias

DMRAC. Here, we present the best case tracking performance observed for all the three controllers. Figures-7.21 and Fig-7.22 clearly show that PID fails at around the end of the flight with high oscillations, whereas we observe DMRAC tracking under severe disturbance forces appears to be the best followed by MRAC.

Similar results are provided for circular reference trajectory with high wind bias. We observe the PID fails very early in the flight. However adaptive controllers are successful in completing the task, we observe DMRAC outperforms the MRAC with much tighter tracking. Refer Figure:7.23-7.24. The Figure-7.25 plots the control torques generated in Roll and pitch to achieve the trajectory tracking.

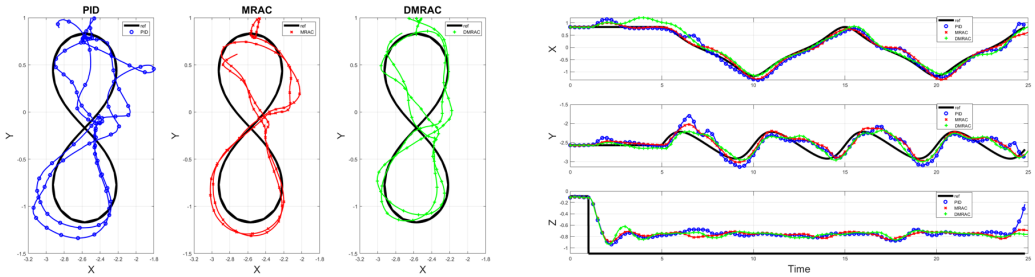


Figure 7.21: Tracking performance under high wind bias with cloth attached underneath the quadcopter

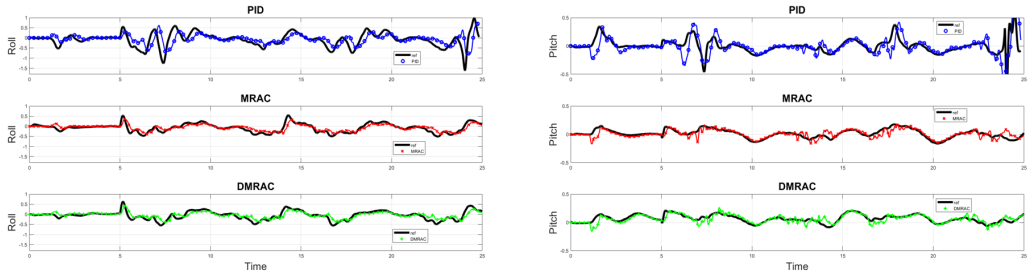


Figure 7.22: Tracking of reference model's roll and pitch signal under high wind bias

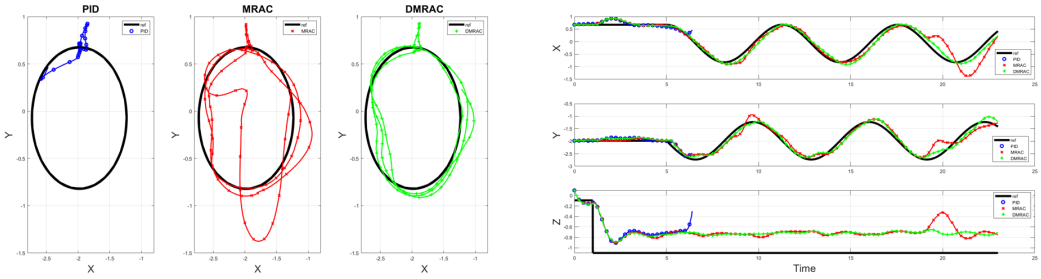


Figure 7.23: Tracking performance for a circular trajectory under high wind bias with cloth attached on drone

7.7.7 Fault tolerance: Rotor blade chipping in mid-flight

To test the fault-tolerance capability of the controllers, we test and compare the performance of PID, MRAC, and DMRAC when rotor chipping occurs during mid-flight. One of the rotor blades is cut in half and is attached back using tape. The quadrotor is commanded to hover at $1m$ above the ground. Due to centrifugal forces, the chipped blade breaks off and causes the fault into the system at an undetermined time. Since this not a controlled fault, to ensure the reliability of the controller, we report controller performance over multiple runs. The results presented in Fig-7.27 clearly shows that DMRAC

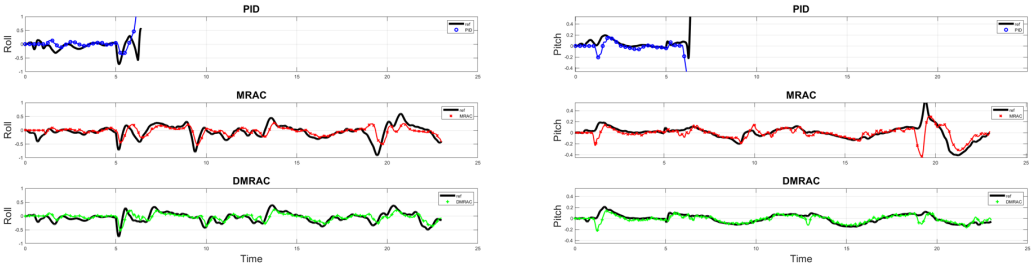


Figure 7.24: Tracking of reference model's roll and pitch under high wind bias with cloth attached on drone

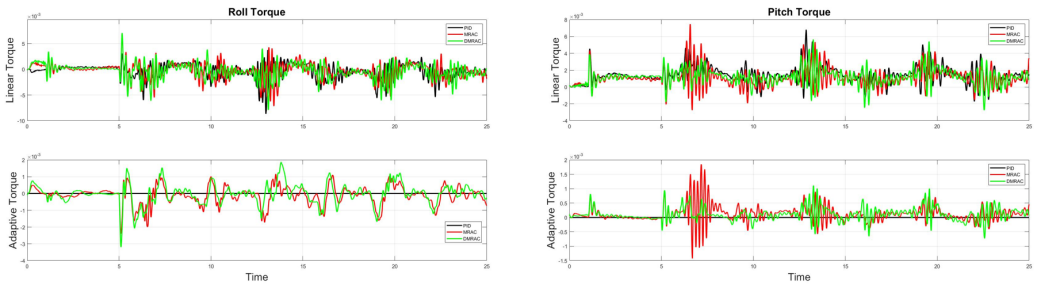


Figure 7.25: Linear and adaptive control torque for each algorithm

outperforms PID and MRAC. In the case of PID, only two runs were carried out, since in both cases, the drone underwent sever oscillation and crashed. Tests conclusively demonstrated that PID is not capable of handling sever faults in the system even with extensive tuning. In the case of MRAC and DMRAC, eight flight tests are carried out. Out of eight test runs, failures were seen twice for MRAC, whereas no failure were observed in the case of DMRAC. Also, on comparing flights where no crash occurs, one can see that MRAC produces poor reference tracking when compared to DMRAC. Refer Figure-7.26 and 7.27 for more detailed results. The figures show mean and variance plots for reference tracking in the x-y-z position for each algorithm.

7.7.8 DMRAC-Learning retention results

In this section, we present the learning retention results of Deep architecture in the model reference adaptive controller. This experiment aims to test the memory associated with deep neural networks in the context of an adaptive controller — the generalizing capability of DMRAC is tested on related but unseen tasks in training. The DMRAC is trained on the labeled pair of input and the output data generated using the model reference generative network. The trained network is then used as a feed-forward function approximator to estimate the adaptive control for reference tracking in a new but similar task. To train the DMRAC neural network, we collect data flying the quadrotor in circles both clockwise and anti-clockwise with and without wind bias. To test this controller performance, we compare it against a tuned PID controller on a new task of tracking the figure of 8 with and without wind bias. Our hypothesis is, since the DMRAC controller is trained over clockwise(cw) and

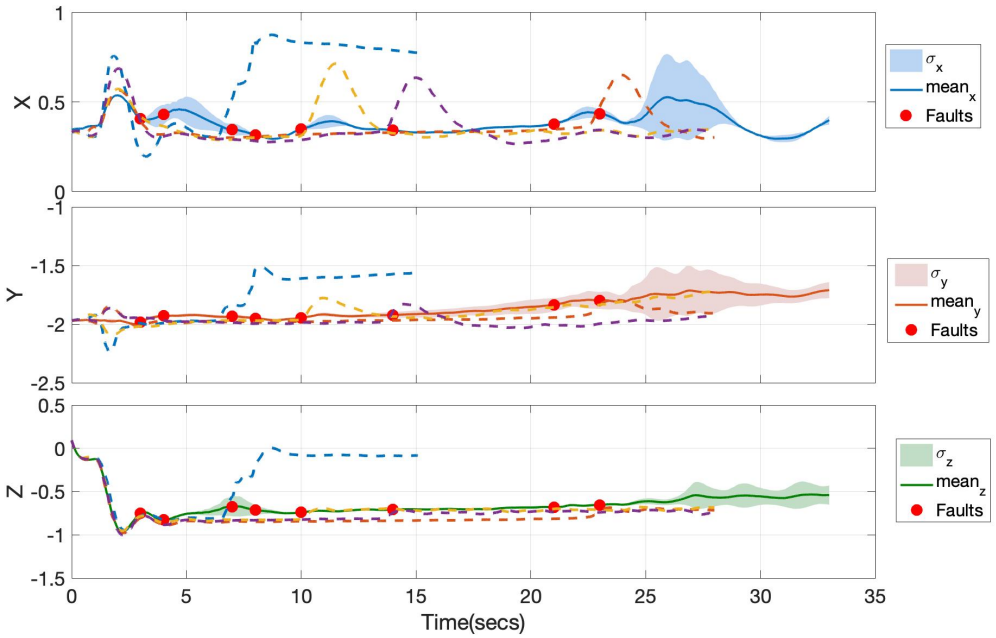


Figure 7.26: MRAC Trajectory tracking performance in X-Y-Z under system fault for eight flight test. Out of eight flights we observe four times the quadrotor either crashed or produced bad tracking (Red dot: Time at which Fault occurred)

anti-clockwise(ccw) trajectories, the controller should be able to generalize to any trajectory formed combining the cw and ccw turns. Since the DMRAC neural network weights are trained off-line before the test flight, the entire controller is hardcoded onto the onboard computer. The parameters, such as PID gains, learning rate, are unchanged and are unchanged from previous experiments.

7.7.9 PID vs Generalization capability of DMRAC

In this experiment, the quadrotor is made to track a similar but unseen trajectory in training (Figure of 8). In Figure:7.28-7.29 we observe that the DMRAC controller generalizes well to a previously unseen reference signal. Comparing to the PID controller, we see that the magnitude of oscillation in roll for the PID is much higher when compared to the generalized controller. Thereby demonstrating the DMRAC not only generalizes well but also proves to be robust.

We also test the DMRAC generalization in the windy case. Here, we can

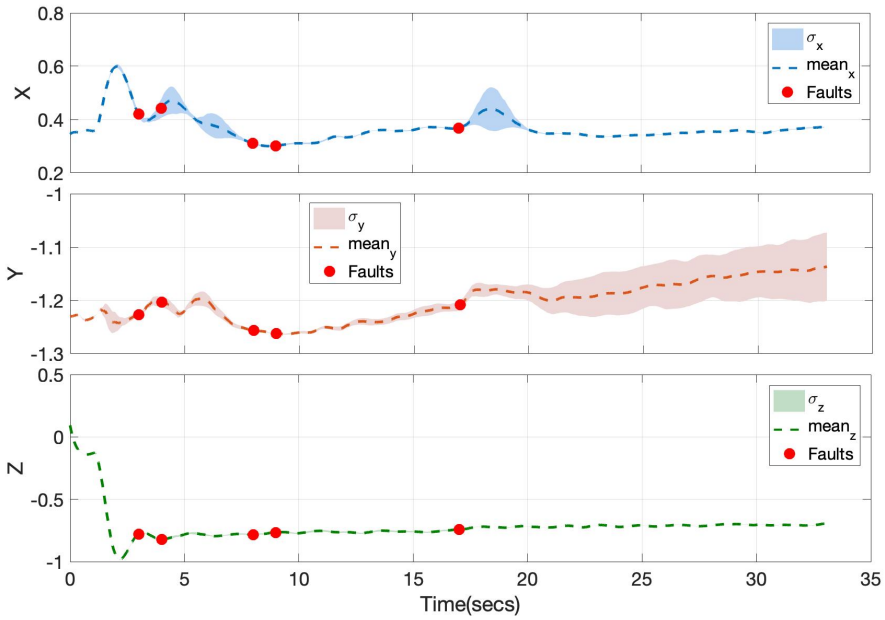


Figure 7.27: DMRAC Trajectory tracking performance in X-Y-Z under system fault for Eight flight test (Red dot: Time at which Fault occurred)

see that the oscillation observed for the generalized controller in tracking is far lower than PID, and it gives much better tracking overall. These experiment demonstrates DMRAC retains the memory of both windy and non-windy cases in form of deep features, and can counter both wind and no wind cases reasonably well even without active online adaptation.

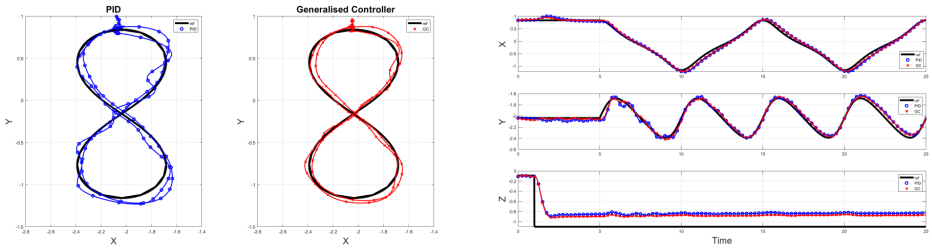


Figure 7.28: Tracking performance on a figure of 8 trajectory

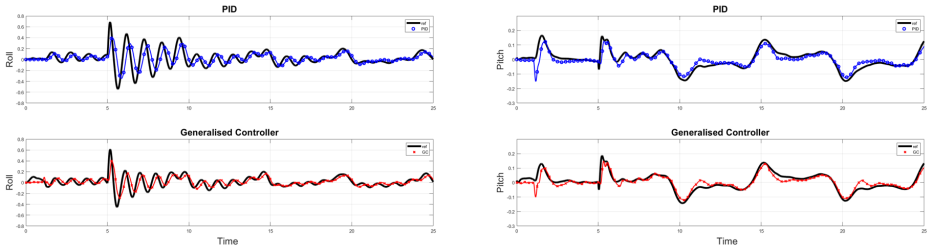


Figure 7.29: Tracking of reference model's roll and pitch for a figure of 8 reference trajectory

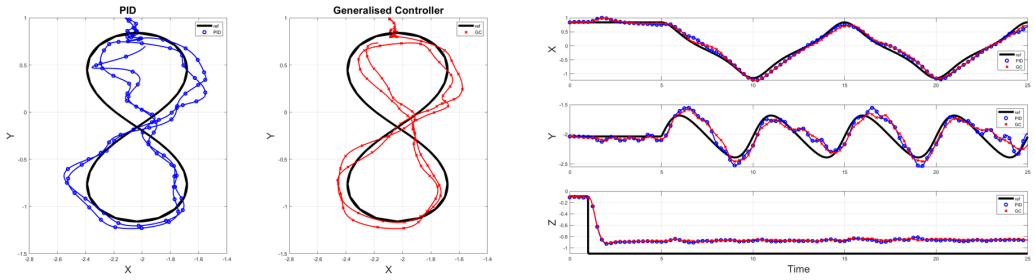


Figure 7.30: Controller performance for clockwise tracking of figure of 8 trajectory under wind bias

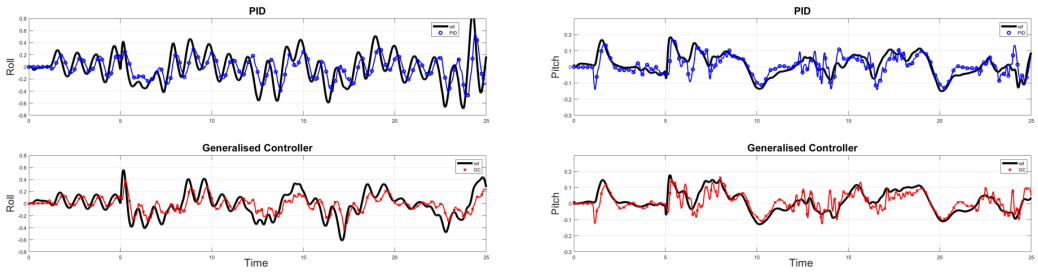


Figure 7.31: Tracking of reference model's roll and pitch for clockwise tracking of figure of 8 under wind bias

CHAPTER 8

CONCLUSION & SUGGESTED FUTURE RESEARCH

The key contributions of this thesis is to show a information enabled adaptation in both learning and control systems can lead to robustfying and efficient use of the baseline policies in face of uncertainties. This hypothesis was presented in two key architectures that is " Policy Transfer in Reinforcement Learning" and "Deep Model Reference Adaptive Control"

8.1 Policy Transfer using Adaptation

We introduced a new transfer learning technique for RL: Adapt-to-Learn, that utilizes combined adaptation and learning for policy transfer to the target tasks. We demonstrated on nonlinear and continuous robotic locomotion tasks that learning to adapt using source policy in the target domain leads to a significant reduction in sample complexity over the prevalent warm-start based approaches. We further also proved theoretical guarantees on the reduced sample complexity of our proposed architecture. There are many exciting directions for future work. A network of policies that can generalize across multiple tasks could be learned based on each new adapted policies. How to train this end-to-end is an important question for meta-learning. The ability of Adapt-to-Learn to handle significant perturbations to the transition model indicates that it should naturally extend to sim-to-real transfer. Indeed we argue that such adaptation is necessary for real-world robotics, as has been established previously in classical domains like flight control. Another exciting direction is to extend the work to other combinatorial domains (e.g., multiplayer games). We expect, therefore, follow on work will find other exciting ways of exploiting such adaptation in RL and machine learning.

We introduced a new transfer learning technique for RL with proven sample

efficiency guarantees. The presented approach demonstrates that an adaptive policy can be learned that makes the transferred source policy near-optimal in the target domain. Our results on realistic, high-fidelity, nonlinear, and continuous domains show that learning an adaptive policy for transfer has significant sample efficiency advantages over the traditional approach of using the transferred policy to initialize RL in the target domain. Unlike the state of art TL methods, we show the transfer is possible without having to learn the target task. Indeed, we proved that the sample complexity of the presented transfer algorithm can be significantly reduced, from being polynomial function of $|\mathcal{S}| \times |\mathcal{A}|$, to $|\mathcal{D}|$. We expect the approach will work for other continuous domains, and with some modifications to combinatorial domains (e.g. games), these are excellent directions for future work.

8.2 Learning over data in Adaptive Control

In chapter 6, we presented a Gaussian Process adaptive controller using model reference generative network architecture to address on-line training of the GP models. The proposed controller uses MRGeN as the generative model to estimate the uncertainty for inferring the GP posterior. We demonstrate unlike GP-MRAC the presented controller does not require state derivative information $\dot{x}(t)$ for training GP model of the uncertainties in the dynamical systems. It is also shown that the presented method leads to a robust adaptive control architecture which allows for fast adaptation while guaranteeing steady-state performance and prove guaranteed uniform bounds on the tracking error. Numerical simulations with wing-rock model demonstrate the controller performance, in achieving reference model tracking in the presence of uncertainty with no prior information on the domain of operation.

A GP is a Bayesian nonparametric adaptive element that can adapt both its weights and the structure of the model in response to the data. The presented GP-MRGeN controller has strong long-term learning properties as well as high control performance [53, 131]. However, GPs can be viewed as "shallow" machine learning models, and do not utilize the power of learning complex features through compositions as deep networks do. Hence, further in chapter 7, utilizing the power of deep learning a even more powerful

learning based MRAC architecture called Deep MRAC is introduced.

In this chapter 7 of the thesis, we presented a DMRAC adaptive controller using model reference generative network architecture to address the issue of feature design in unstructured uncertainty. Deep Neural Networks (DNN) have lately shown tremendous empirical performance in many applications where supervised learning is used, including fields such as computer vision, speech recognition, translation, natural language processing, Robotics, Autonomous driving and many more [58]. Unlike their counterparts such as Gaussian Radial Basis Function networks [59, 60], deep networks learn features by learning the weights of nonlinear compositions of weighted features arranged in a directed acyclic graph [61]. They are indeed much deeper versions of the single hidden layer neural networks (SHL-NN) studied in MRAC [62, 63]. It is now pretty clear that deep neural networks are outshining heuristic based regression and classification algorithms as well as RBFNs, GPs, SHL-NNs, and other classical machine learning techniques [64]. The proposed controller uses DNN to model significant uncertainties without knowledge of the system's domain of operation. We provide theoretical proofs of the controller generalizing capability over unseen data points and boundedness properties of the tracking error. Experiments with the quadrotor vehicle demonstrate the controller performance in achieving reference model tracking in the presence of significant matched uncertainties and also learning retention when used as a feed-forward adaptive network on similar but unseen new tasks. Thereby we claim DMRAC is a highly robust architecture for high-performance control of nonlinear systems with robustness and long-term learning properties. Our results demonstrate how DNNs can be utilized in stable learning schemes for adaptive control of safety-critical systems such as aircraft. This provides a way to build highly adaptive and long-term learning capable flight controllers. Furthermore, the dual timescale analysis scheme used by us should be generalizable to other DNN based learning architectures, including reinforcement learning.

8.3 Feature Analysis and Selection using Deep Learning Approach in Adaptive Control

Deep neural network models have recently drawn lots of attention, as they produce impressive results in many computer vision tasks such as image classification, object detection, image segmentation, speech processing, data encoding etc. However analyzing or interpreting a deep network success in performing these tasks is mathematically not feasible and has become a challenging question. We often treat the deep neural network models as black boxes without clear understanding on internal mechanics. Without a clear understanding of how and why a model works in DMRAC, we cannot clearly understand how and which part of network neurons activate upon seeing input from different flight regime. Also how the network is able to store the relevant features pertaining to different flight regimes and present adaptive controller the appropriate features for the appropriate flight data. As a result, academic researchers and industrial practitioners are facing challenges that demand visualizing these high dimension network data for better understanding and analyzing machine learning models,especially their inner working mechanism.

There are many data visualization and analysis techniques available in literature, which have shown to advance our understanding of deep models. The data visualization techniques are used to analyse the model which achieve the state of the art results in image and video classification [169]. Also there are methods to understand classification [170, 171, 172] and regression models [173]. The visualization techniques can be categorized as point-based and network-based methods. In this thesis we will using point based techniques names t-SNE [174] and PCA [175] for analyzing the DMRAC features.

Point based techniques [173, 176] help visualize the relation between neural network components such as neurons or learned representations though point cloud. The point based technique provide us the method to visualize a high dimension feature in the lower dimensional space. Typically each feature in higher dimension is represented as point in lower dimensional space. While carrying out dimensionality reduction these visualization technique preserve the neighborhood relationship of features in higher dimensions. Features resulted from similar inputs to the network are cluster together in lower dimensional representation. Principle Component Analysis(PCA) [175] and t-

Distributed Stochastic Neighborhood Embedding (t-SNE) [174] are two very important techniques in the point based visualization of higher dimensional data in to lower dimensional feature vectors. These methods will help us explore and analyse the relationship between the cluster in the deep feature for different flight regime the network has seen through training.

8.3.1 t-SNE and PCA

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets [174]. It is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions. Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability. The details of t-SNE are beyond scope of this work and we present a very a brief working of the algorithm, interested readers can refer to [174].

The t-SNE algorithm cluster the points through two steps of processing the data. First, t-SNE constructs a probability distribution over pairs of high-dimensional objects in such a way that similar objects have a high probability of being picked while dissimilar points have an extremely small probability of being picked.

Let $\{X_1, X_2 \dots X_N\} \in \mathbb{R}^k$ are set of N high dimensional objects. t-SNE computes the neighbourhood relation using using the probability p_{ij} which encodes the similarity of the objects x_i and x_j as follows,

$$p_{j|i} = \frac{\exp(\|x_i - x_j\|^2 / \sigma_i^2)}{\sum_{k \neq i} \exp(\|x_i - x_k\|^2 / \sigma_i^2)} \tag{8.1}$$

Where σ_i is bandwidth of the gaussian distribution and can be tuned as perplexity factor in t-SNE algorithm. The condition probability $p_{j|i}$ similarity index of point x_j to point x_i. The total probability is defined as

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{8.2}$$

with $p_{ii} = 0$.

Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map. Let $\{Y_1, Y_2 \dots Y_N\} \in \mathbb{R}^d$ are set of N lower dimensional objects such that $k >> d$. A similar neighbourhood relation is defined between the initially randomly chosen lower dimensional objects using Student-t distribution as follows,

$$q_{j|i} = \frac{\left(1 + \|Y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq i} \left(1 + \|Y_i - Y_k\|^2\right)^{-1}} \tag{8.3}$$

A heavy-tailed Student t-distribution is used to measure similarities between low-dimensional points in order to allow dissimilar objects to be modeled far apart in the map. With q_{ij} defined as above and also in this case set $q_{ii} = 0$. Now minimizing the Kullback–Leibler divergence (KL divergence) between the two distributions with respect to the locations of the points in the map, t-SNE generates a data clustering in lower dimension space which reflects the data similarity relation in higher dimensional space.

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors (each being a linear combination of the variables and containing N observations) are an uncorrelated orthogonal basis set. Further details of PCA can be found in [175] and reference therein.

8.3.2 Feature Visualization for Quadrotor and Wingrock Experiments

We use t-SNE and PCA algorithm to visualize the high dimensional feature vector $\Phi_n^\sigma(x)$ learned using DMRAC and S-DMRAC algorithm for Quadrotor and Wing-Rock control experiments.

Wing-Rock Experiment

We use DMRAC network to learn to control a uncertain Wing-Rock system to track a given mixture of step and sinusoidal reference signal. We collect the finitely exciting data and train the deep features from first 100*secs* and further the deep features are used as frozen network with no active learning. Despite no active learning, only from the memory of previously seen training data the DMRAC controller can generalize and generate control to track any mix of step and sinusoidal reference signals. Figure-8.1 show the

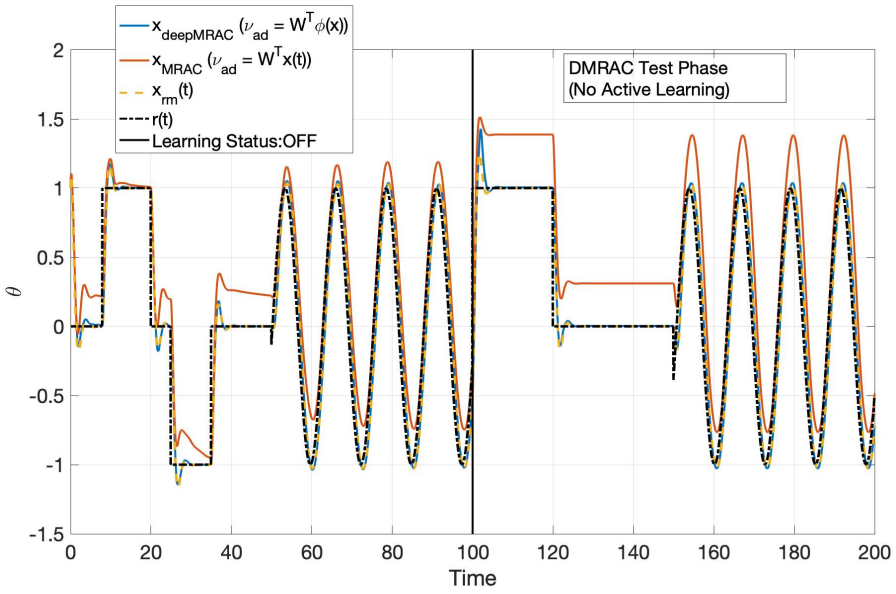


Figure 8.1: MRAC vs DMRAC in Training and Test Phase. For 1st 100*sec* the DMRAC features are trained, further the DMRAC is used as frozen network with no active online learning

controller performance in producing reference tracking. Up to 100*sec* mark in the simulation the data is collected for the DMRAC training using kernel Independence test. After 100*sec* the deep feature update is stopped and we see this part of reference tracking as the test phase. From Figure-8.1 & Figure-8.2 we can see the DMRAC controller is successful in generalizing to similar reference signal without active learning.

The Deep features extracted from the data using DMRAC controller when visualized using t-SNE shows clearly two distinct cluster pertaining to step and sinusoidal reference tracking Figure-8.3. This flight regime specific clustering of the features very clearly support our argument that DMRAC is successful in extracting and memorizing the important features for a given

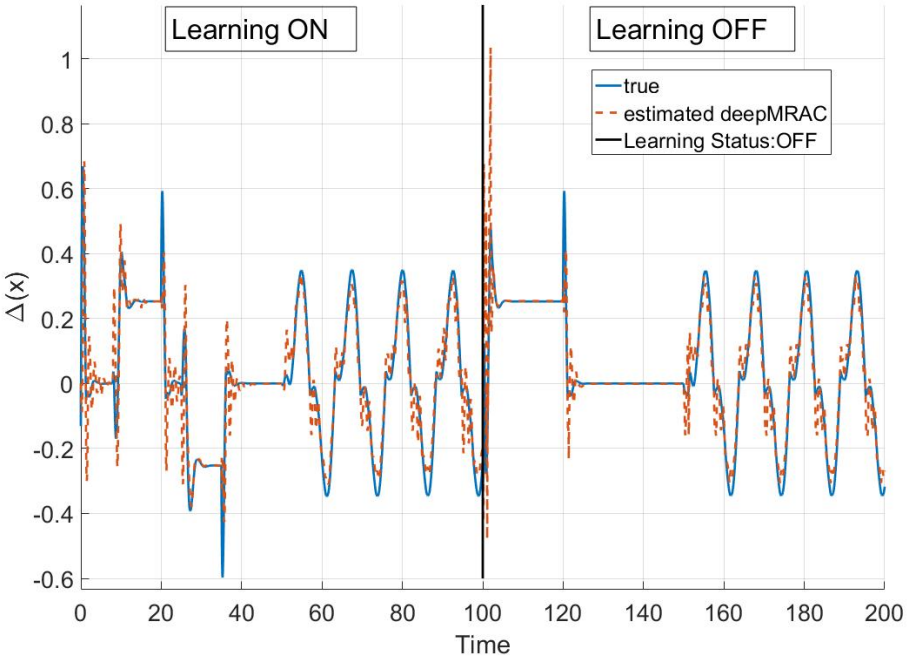


Figure 8.2: DMRAC controller uncertainty estimate while in Training and Test phase

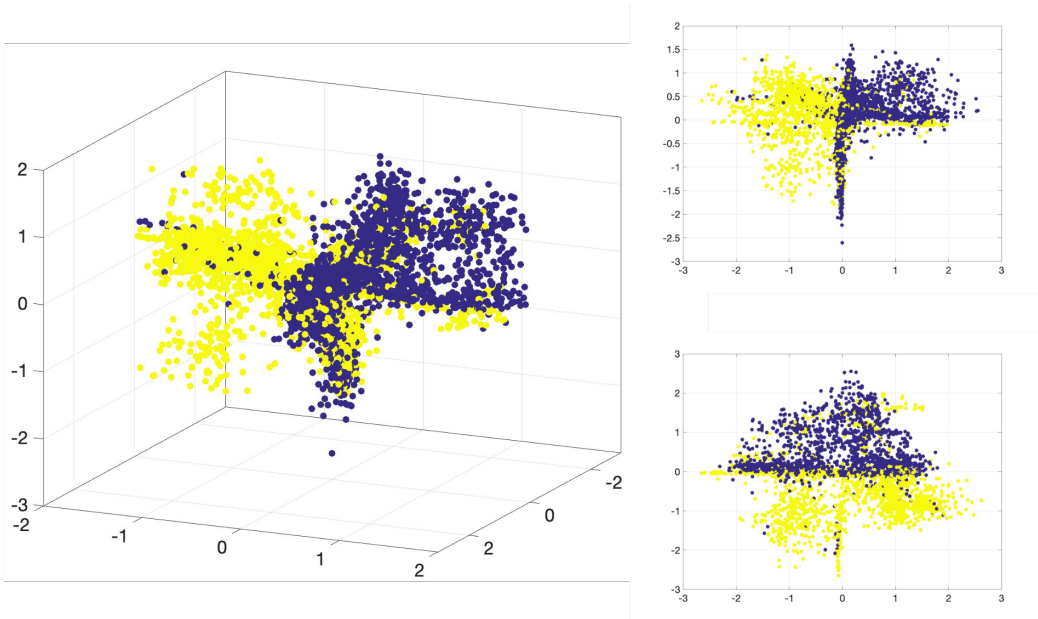


Figure 8.3: DMRAC Features visualization using t-SNE.

problem. And during Test phase when presented with input data for a related task DMRAC can choose relevant feature vector for control synthesis.

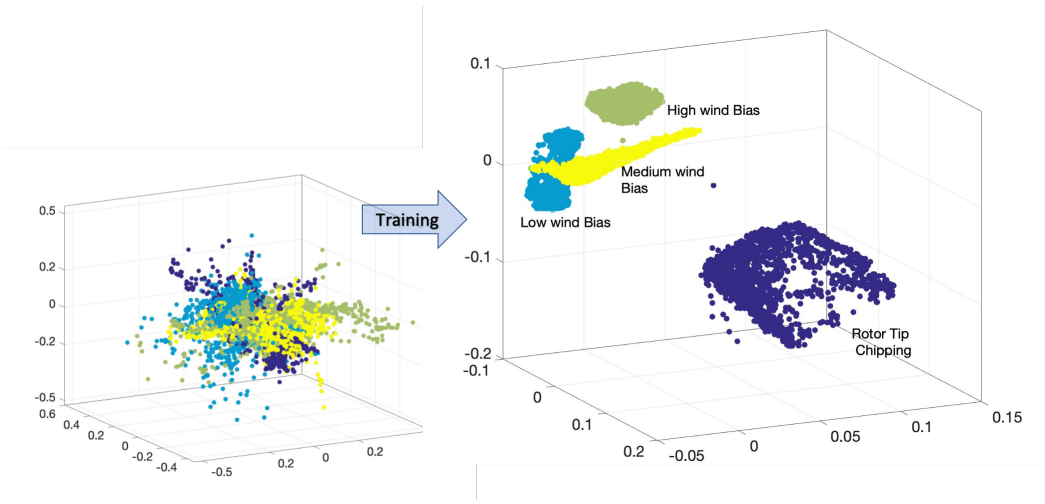


Figure 8.4: DMRAC features before training and after training to handle Low, Medium, High Wind Bias and Rotor Tip breaking case

DMRAC evaluation using Quadrotor under Wind Bias and Rotor blade breaking

Similar to wing-Rock experiment, we used a DMRAC network to control a physical quadrotor tracking figure-8 reference signal under wind bias and Rotor fault. The details of the network architecture are provided in the Table-7.1.

Same Deep model is trained progressively over the collected data from Low, Medium, High wind bias and Fault tolerant Case. Figure-8.4 show the t-SNE visualization of the 20−dimensional feature vector in 3−dimensional scatter-plot. In Figure-8.4 each point denote one feature representation of test state from either of the experiments. As shown in the figure, after training, the spatial separation between classes is significantly improved. This observation provides evidence for the hypothesis that neural networks learn to detect representations that are useful for different flight regimes and uncertainties experienced by the vehicle. Figure-8.5 is the result of principle component analysis and provides the details of important features used in decision making while windy and fault cases of the flight of quadrotor.

The feature visualization also helps with the understanding of similar flight regimes. As we can observe the windy and Fault cases are classified as two very distinct clusters. But the wind cases from Low to Medium to High wind case cluster are together but progressively move away from each other as the wind disturbance is increased.

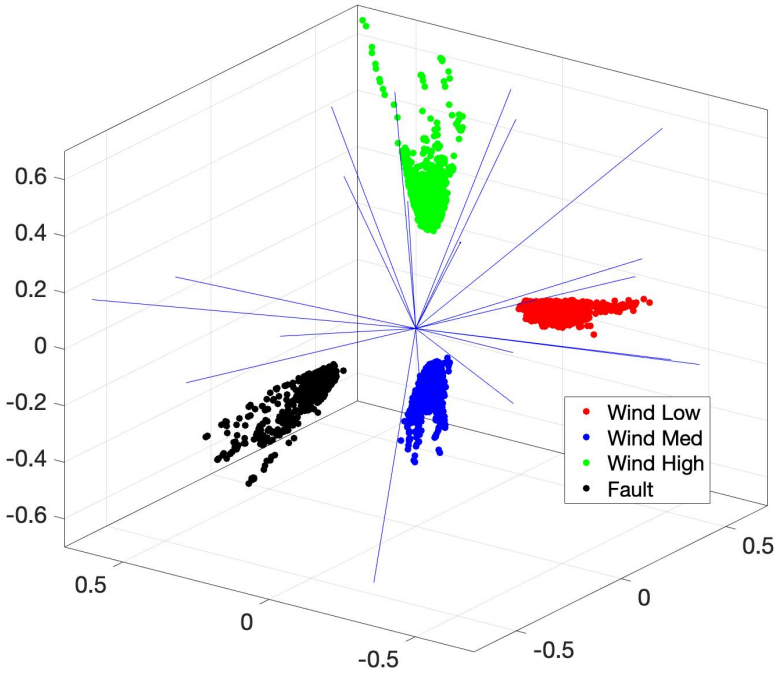


Figure 8.5: PCA for DMRAC features trained to handle Low, Medium, High Wind Bias and Rotor Tip breaking case

The feature analysis might not help us directly to design a DMRAC deep network, but provide us a tool to visualize the high dimensional feature data and give us intuitive perspective on how the features get cluster for different tasks.

APPENDIX A

A.1 Simulation Lemma [1]

Simulation lemma states that if one MDP $\hat{M}$ is sufficiently accurate approximation of the another MDP M, then we can actually approximate the $T-$step return of any policy in M quite accurately by its $T-$step return in MDP $\hat{M}$.

Definition A.1.1 *Let M and $\hat{M}$ be markov decision process over the same state space. Then we can say that $\hat{M}$ is an $\alpha-$approximation of M if,*

For any state s

$$R_M(s) - \alpha \leq R_{\hat{M}}(s) \leq R_M(s) + \alpha \tag{A.1}$$

For any states s_k and s_{k+1} and any action a, the transition models can be written as

$$P_M(s_{k+1}|s_k, a) - \alpha \leq P_{\hat{M}}(s_{k+1}|s_k, a) \leq P_M(s_{k+1}|s_k, a) + \alpha \tag{A.2}$$

The Simulation Lemma, which says that provided $\hat{M}$ is sufficiently close to M in the sense of above definition, the T -step return of policies in $\hat{M}$ and M will be similar

Lemma A.1.2 *Let M be any Markov decision process over N states,*

- *Undiscounted Case: Let $\hat{M}$ be an $O\left(\left(\epsilon/(NT\eta_{max}^T)\right)^2\right)-$approximation of M. Then for any policy $\pi \in \Pi_M^{T,\epsilon/2}$ and for any state s, the return over policy can be written as,*

$$U_M^\pi(s,T) - \alpha \leq U_{\hat{M}}^\pi(s,T) \leq U_M^\pi(s,T) + \alpha \tag{A.3}$$

- *Discounted case: Let* $T > (1/(1-\gamma))\log\left(R_{max}/(\epsilon(1-\gamma))\right)$ *and let* $\hat{M}$ *be an* $O\left(\left(\epsilon/(NT\eta_{max}^{T})\right)^{2}\right)-$*approximation of* M*. Then for any policy* π *and any state* s*, we can write the value function as*

$$V_{M}^{\pi}(s) - \alpha \leq V_{\hat{M}}^{\pi}(s) \leq V_{M}^{\pi}(s) + \alpha \tag{A.4}$$

The proof of simulation lemma is provided in [1].

A.2 Manifold Alignment

[177] A manifold alignment problem arises in many real world applications, where two manifolds (defined by totally different features) need to be aligned with no correspondence information. Solving this problem is hard, since there are two unknown variables in this problem: the correspondence and the transformation. One such example is control transfer between different Markov decision processes (MDPs), where we want to align state spaces of different tasks. In a MDP manifold alignment problem, the state are usually defined in different features and its hard to find correspondence between them. This problem can be more precisely defined as follows: suppose we have two data sets $X = x_1, \ldots, x_m$ and $Y = y_1, \ldots, y_n$ for which we want to find correspondence, our aim is to compute functions α and β to map x_i and y_j to the same space such that $\alpha^T x_i$ and $\beta^T y_j$ can be directly compared.

A semi supervised Manifold alignment is presented by [178]. Given two data sets X and Y along with the additional correspondence between a small subset of training instances $x_i \leftrightarrow y_i$ for $i \in [1, l]$. Semi-supervised alignment directly computes mapping between x_i and y_i minimizing the following cost function.

$$C(f,g) = \mu \sum_{i=1}^{t} (f-g)^2 + 0.5 \sum_{i,j} (f_i - f_j)^2 W_x^{i,j} + 0.5 \sum_{i,j} (g_i - g_j)^2 W_y^{i,j} \tag{A.5}$$

Where f_i is mapping result of x_i and similarly g_i is mapping result of y_i. The first term penalizes the differences between X and Y on the mapping results of the corresponding instances. The second and third terms guarantee that the neighborhood relationship within X and Y will be preserved.

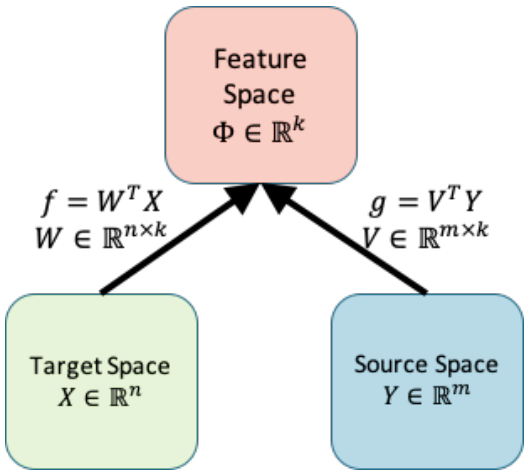


Figure A.1: Manifold alignment between two state spaces defined in different feature spaces

The unsupervised manifold alignment presented by [3] has two fundamental difference compared supervised method. First the correspondence information is not available. The correspondence information in $C(f, g)$ is replaced by soft constraint induced by local geometry. Second the unsupervised manifold alignment seek a linear mapping functions α and β rather than direct embeddings, so that the mapping is defined everywhere. The cost function we want to minimize is as follows:

$$\begin{aligned} C(f,g) &= \mu \sum_{i=1}^{t} \left(\alpha^T x_i - \beta^T y_i\right)^2 W^{i,j} \\ &+ 0.5 \sum_{i,j} \left(\alpha^T x_i - \beta^T x_j\right)^2 W_x^{i,j} + 0.5 \sum_{i,j} \left(\alpha^T y_i - \beta^T y_j\right)^2 W_y^{i,j} \end{aligned} \tag{A.6}$$

The first term of $C(\alpha, \beta)$ penalizes the differences between X and Y on the matched local patterns in the new space. Suppose that local geometry relation representation matrix Rx_i and Ry_j are similar, then weight matrix $W^{i,j}$ will be large. If the mapping results in x_i and y_j are being far away from each other in the new space, the first term will be large. The second and third terms preserve the neighborhood relationship within X and Y. The details of the algorithm and implementation details are presented in [3, 178]

REFERENCES

[1] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine Learning*, vol. 49, no. 2-3, pp. 209–232, 2002.

[2] K. J. Åström and B. Wittenmark, *Adaptive Control*, 2nd ed. Readings: Addison-Weseley, 1995.

[3] C. Wang and S. Mahadevan, "Manifold alignment without correspondence." in *IJCAI*, vol. 2, 2009, p. 3.

[4] S. B. Thrun, "Efficient exploration in reinforcement learning," 1992.

[5] B. Price and C. Boutilier, "Accelerating reinforcement learning through implicit imitation," *Journal of Artificial Intelligence Research*, vol. 19, pp. 569–629, 2003.

[6] U. Syed and R. E. Schapire, "A game-theoretic approach to apprenticeship learning," in *Advances in neural information processing systems*, 2008, pp. 1449–1456.

[7] P. Abbeel and A. Y. Ng, "Exploration and apprenticeship learning in reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 1–8.

[8] J. Z. Kolter, P. Abbeel, and A. Y. Ng, "Hierarchical apprenticeship learning with application to quadruped locomotion," in *Advances in Neural Information Processing Systems*, 2008, pp. 769–776.

[9] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, and others, "Swarmanoid: a novel concept for the study of heterogeneous robotic swarms," *IEEE Robotics & Automation Magazine*, 2012.

[10] M. J. Mataric, "Reward functions for accelerated learning," in *Machine learning proceedings 1994*. Elsevier, 1994, pp. 181–189.

[11] M. E. Taylor, P. Stone, and Y. Liu, "Transfer learning via inter-task mappings for temporal difference learning," *Journal of Machine Learning Research*, vol. 8, no. Sep, pp. 2125–2167, 2007.

[12] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, 2013, pp. 1–9.

[13] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 2219–2225.

[14] Y. Duan, M. Andrychowicz, B. Stadie, O. J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," in *Advances in neural information processing systems*, 2017, pp. 1087–1098.

[15] X. B. Peng, G. Berseth, and M. Van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 81, 2016.

[16] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 41, 2017.

[17] L. Liu and J. Hodgins, "Learning to schedule control fragments for physics-based characters using deep q-learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 3, p. 29, 2017.

[18] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller et al., "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[19] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[20] M. E. Taylor, P. Stone, and Y. Liu, "Value functions for rl-based behavior transfer: A comparative study," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 20, no. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005, p. 880.

[21] H. B. Ammar, K. Tuyls, M. E. Taylor, K. Driessens, and G. Weiss, "Reinforcement learning transfer via sparse coding," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 383–390.

[22] H. B. Ammar, E. Eaton, P. Ruvolo, and M. E. Taylor, "Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment," in *Proc. of AAAI*, 2015.

[23] B. Banerjee and P. Stone, "General game learning using knowledge transfer." in *IJCAI*, 2007, pp. 672–677.

[24] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *arXiv preprint arXiv:1710.06537*, 2017.

[25] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[26] M. Yan, I. Frosio, S. Tyree, and J. Kautz, "Sim-to-real transfer of accurate grasping with eye-in-hand observations and continuous control," *arXiv preprint arXiv:1712.03303*, 2017.

[27] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 156–163.

[28] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with large-scale data collection," in *International Symposium on Experimental Robotics*. Springer, 2016, pp. 173–184.

[29] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7559–7566.

[30] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar, "Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost," *arXiv preprint arXiv:1810.06045*, 2018.

[31] S. Daftry, J. A. Bagnell, and M. Hebert, "Learning transferable policies for monocular reactive mav control," in *International Symposium on Experimental Robotics*. Springer, 2016, pp. 3–11.

[32] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.

[33] M. Wulfmeier, I. Posner, and P. Abbeel, "Mutual alignment transfer learning," *arXiv preprint arXiv:1707.07907*, 2017.

[34] G. Tao, *Adaptive control design and analysis.* John Wiley & Sons, 2003, vol. 37.

[35] G. Tao, *Adaptive Control Design and Analysis.* New York: Wiley, 2003.

[36] G. Tao, S. M. Joshi, and X. Ma, "Adaptive state feedback and tracking control of systems with actuator failures," *Automatic Control, IEEE Transactions on*, vol. 46, no. 1, pp. 78–95, 2001.

[37] G. Chowdhary, "Concurrent Learning for Convergence in Adaptive Control Without Persistency of Excitation," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, 2010.

[38] Y. H. Kim and F. L. Lewis, *High-level feedback control with neural networks.* World Scientific, 1998, vol. 21.

[39] W. M. Haddad, K. Y. Volyanskyy, J. M. Bailey, and J. J. Im, "Neuroadaptive output feedback control for automated anesthesia with noisy eeg measurements," *IEEE Transactions on Control Systems Technology*, vol. 19, no. 2, pp. 311–326, 2010.

[40] K. Y. Volyanskyy, "Adaptive and Neuroadaptive Control for nonnegative and compartmental dynamical systems," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, 3 2010.

[41] K. Y. Volyanskyy, W. M. Haddad, and A. J. Calise, "A new neuroadaptive control architecture for nonlinear uncertain dynamical systems: Beyond σ and $e-$modifications," *IEEE Transactions on Neural Networks*, vol. 20, no. 11, pp. 1707–1723, 2009.

[42] E. N. Johnson, "Limited Authority Adaptive Flight Control," Ph.D. dissertation, Georgia Institute of Technology, School of Aerospace Engineering, Atlanta, GA 30332, 12 2000.

[43] E. N. Johnson and S. K. Kannan, "Adaptive trajectory control for autonomous helicopters," *Journal of Guidance, Control, and Dynamics*, vol. 28, no. 3, pp. 524–538, 2005.

[44] M. Steinberg, "Historical overview of research in reconfigurable flight control," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 219, no. 4, pp. 263–275, 2005.

[45] S. Lee, "Neural network based adaptive control and its applications to aerial vehicles," Ph.D. dissertation, Georgia Institute of Technology, 2001.

[46] S. K. Kannan and E. N. Johnson, "Adaptive control of systems in cascade with saturation," in *49th IEEE Conference on Decision and Control (CDC)*. IEEE, 2010, pp. 42–47.

[47] F. Lewis, "Nonlinear network structures for feedback control," *Asian Journal of Control*, vol. 1, no. 4, pp. 205–228, 1999.

[48] P. A. Ioannou and J. Sun, *Robust adaptive control*. PTR Prentice-Hall Upper Saddle River, NJ, 1996, vol. 1.

[49] K. Narendra and A. Annaswamy, "Robust adaptive control in the presence of bounded disturbances," *IEEE Transactions on Automatic Control*, vol. 31, no. 4, pp. 306–315, 1986.

[50] N. Hovakimyan and C. Cao, *1 Adaptive Control Theory: Guaranteed Robustness with Fast Adaptation*. SIAM, 2010.

[51] N. Nguyen, "Asymptotic Linearity of Optimal Control Modification Adaptive Law with Analytical Stability Margins," in *Infotech@AIAA conference*, Atlanta, GA, 2010.

[52] G. Chowdhary and E. Johnson, "Concurrent learning for convergence in adaptive control without persistency of excitation," in *Decision and Control (CDC), 2010 49th IEEE Conference on*. IEEE, 2010, pp. 3674–3679.

[53] G. Chowdhary, H. A. Kingravi, J. P. How, and P. A. Vela, "Bayesian nonparametric adaptive control using gaussian processes," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 26, no. 3, pp. 537–550, 2015.

[54] G. Joshi and G. Chowdhary, "Cross-domain transfer in reinforcement learning using target apprentice," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7525–7532.

[55] K. J. Åström and B. Wittenmark, *Adaptive control*. Courier Corporation, 2013.

[56] G. Chowdhary, T. Wu, M. Cutler, and J. P. How, "Rapid transfer of controllers between uavs using learning-based adaptive control," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 5409–5416.

[57] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1126–1135.

[58] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[59] R. M. Sanner and J.-J. Slotine, "Gaussian networks for direct adaptive control," *Neural Networks, IEEE Transactions on*, vol. 3, no. 6, pp. 837–863, 11 1992.

[60] M. Liu, G. Chowdhary, B. C. da Silva, S.-Y. Liu, and J. P. How, "Gaussian processes for learning and control: A tutorial with examples," *IEEE Control Systems Magazine*, vol. 38, no. 5, pp. 53–86, 2018.

[61] D. Yu, M. L. Seltzer, J. Li, J.-T. Huang, and F. Seide, "Feature Learning in Deep Neural Networks - Studies on Speech Recognition Tasks," *arXiv e-prints*, p. arXiv:1301.3605, Jan 2013.

[62] F. L. Lewis, "Nonlinear Network Structures for Feedback Control," *Asian Journal of Control*, vol. 1, pp. 205–228, 1999.

[63] G. Chowdhary and E. N. Johnson, "Theory and Flight Test Validation of a Concurrent Learning Adaptive Controller," *Journal of Guidance Control and Dynamics*, vol. 34, no. 2, pp. 592–607, 3 2011.

[64] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury et al., "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal processing magazine*, vol. 29, 2012.

[65] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and others, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[66] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[67] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[68] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser et al., "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[69] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 1995, vol. 1, no. 2.

[70] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*. Athena Scientific Belmont, MA, 1996, vol. 5.

[71] A. Geramifard, T. J. Walsh, S. Tellex, G. Chowdhary, N. Roy, J. P. How et al., "A tutorial on linear function approximators for dynamic programming and reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 6, no. 4, pp. 375–451, 2013.

[72] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[73] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[74] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[75] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," 2017.

[76] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[77] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[78] A. Y. Ng and M. Jordan, "Pegasus: A policy search method for large mdps and pomdps," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 2000, pp. 406–415.

[79] V. Tangkaratt, S. Mori, T. Zhao, J. Morimoto, and M. Sugiyama, "Model-based policy gradients with parameter-based exploration by least-squares conditional density estimation," *Neural networks*, vol. 57, pp. 128–140, 2014.

[80] R. S. Sutton, "Integrated modeling and control based on reinforcement learning and dynamic programming," in *Advances in neural information processing systems*, 1991, pp. 471–478.

[81] G. Chowdhary, M. Liu, R. Grande, T. Walsh, J. How, and L. Carin, "Off-policy reinforcement learning with Gaussian processes," *Automatica Sinica, IEEE/CAA Journal of*, vol. 1, no. 3, pp. 227–238, 2014.

[82] M. Kuss, "Gaussian Process Models for Robust Regression, Classification, and Reinforcement Learning," Ph.D. dissertation, Technische Universität Darmstadt, 2006.

[83] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *CoRR*, vol. abs/1506.02438, 2015.

[84] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7, pp. 1180–1190, 2008.

[85] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 4754–4765.

[86] S. Kamthe and M. Deisenroth, "Data-efficient reinforcement learning with probabilistic model predictive control," in *International Conference on Artificial Intelligence and Statistics*, 2018, pp. 1701–1710.

[87] K. S. Narendra and J. Balakrishnan, "Adaptive control using multiple models," *Automatic Control, IEEE Transactions on*, vol. 42, no. 2, pp. 171–187, 2 1997.

[88] G. Chowdhary, H. A. Kingravi, J. P. How, and P. A. Vela, "Bayesian Nonparametric Adaptive Control Using Gaussian Processes," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 3, pp. 537–550, 3 2015.

[89] A. Calise, N. Hovakimyan, and M. Idan, "Adaptive output feedback control of nonlinear systems using neural networks," *Automatica*, vol. 37, no. 8, pp. 1201–1211, 2001, special Issue on Neural Networks for Feedback Control.

[90] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[91] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[92] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[93] L. Busoniu, R. Babuska, B. D. Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, 1st ed. {CRC} Press, 4 2010.

[94] A. Axelrod and G. Chowdhary, "The Explore-Exploit Dilemma in Non-stationary Decision Making under Uncertainty," in *Handling Uncertainty and Networked Structure in Robot Control*, 1st ed., ser. 2198-4182. Springer, 2015, ch. The Explor.

[95] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[96] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal control*. John Wiley & Sons, 2012.

[97] H. Modares, F. L. Lewis, and M.-B. Naghibi-Sistani, "Integral reinforcement learning and experience replay for adaptive optimal control of partially-unknown constrained-input continuous-time systems," *Automatica*, vol. 50, no. 1, pp. 193–202, 2014.

[98] B. Kiumarsi, F. L. Lewis, H. Modares, A. Karimpour, and M.-B. Naghibi-Sistani, "Reinforcement q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics," *Automatica*, vol. 50, no. 4, pp. 1167–1175, 2014.

[99] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[100] J. Condell, J. Wade, L. Galway, M. McBride, P. Gormley, J. Brennan, and T. Somasundram, "Problem solving techniques in cognitive science," *Artificial Intelligence Review*, vol. 34, no. 3, pp. 221–234, 2010.

[101] L. Torrey, J. Shavlik, T. Walker, and R. Maclin, "Relational macros for transfer in reinforcement learning," in *International Conference on Inductive Logic Programming*. Springer, 2007, pp. 254–268.

[102] Y. Liu and P. Stone, "Value-function-based transfer for reinforcement learning using structure mapping," in *Proceedings of the national conference on artificial intelligence*, vol. 21, no. 1. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006, p. 415.

[103] G. Konidaris and A. G. Barto, "Building portable options: Skill transfer in reinforcement learning." in *IJCAI*, vol. 7, 2007, pp. 895–900.

[104] G. Konidaris, I. Scheidwasser, and A. Barto, "Transfer in reinforcement learning via shared features," *Journal of Machine Learning Research*, vol. 13, no. May, pp. 1333–1371, 2012.

[105] M. E. Taylor and P. Stone, "Cross-domain transfer for reinforcement learning," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 879–886.

[106] T. Chen and H. Chen, "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 911–917, 1995.

[107] Y. Liu and F. Ding, "Convergence properties of the least squares estimation algorithm for multivariable systems," *Applied Mathematical Modelling*, vol. 37, no. 1, pp. 476–483, 2013.

[108] K. Hinderer, "Lipschitz continuity of value functions in markovian decision processes," *Mathematical Methods of Operations Research*, vol. 62, no. 1, pp. 3–22, 2005.

[109] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine Learning*, vol. 49, no. 2, pp. 209–232, Nov 2002. [Online]. Available: https://doi.org/10.1023/A:1017984413808

[110] N. Jiang, "Pac reinforcement learning with an imperfect model," in *Proc. of AAAI*, 2018.

[111] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[112] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," *CoRR*, vol. abs/1708.02596, 2017. [Online]. Available: http://arxiv.org/abs/1708.02596

[113] J. Randlov and P. Alstrom, "Learning to drive a bicycle using reinforcement learning and shaping," in *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998, pp. 463–471.

[114] K. J. Åström, R. E. Klein, and A. Lennartsson, "Bicycle dynamics and control," *IEEE Control Systems Magazine*, vol. 25, no. 4, pp. 26–47, 2005.

[115] J. W. Krakauer and P. Mazzoni, "Human sensorimotor learning: adaptation, skill, and beyond," *Current opinion in neurobiology*, vol. 21, no. 4, pp. 636–644, 2011.

[116] M. J. Fryling, C. Johnston, and L. J. Hayes, "Understanding observational learning: An interbehavioral approach," *The Analysis of Verbal Behavior*, vol. 27, no. 1, pp. 191–203, 2011.

[117] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few-shot learning," 2017.

[118] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[119] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.

[120] P. Ioannou and J. Sun, "Theory and design of robust direct and indirect adaptive-control schemes," *International Journal of Control*, vol. 47, no. 3, pp. 775–813, 1988.

[121] J.-B. Pomet and L. Praly, "Adaptive nonlinear regulation: estimation from the Lyapunov equation," *Automatic Control, IEEE Transactions on*, vol. 37, no. 6, pp. 729–740, 6 1992.

[122] A. J. Calise, M. Sharma, and S. Lee, "Adaptive Autopilot Design for Guided Munitions," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 23, no. 5, 2000.

[123] T. Yucelen and A. Calise, "Derivative-Free model reference adaptive control," vol. 34, no. 8, pp. 933–950, 2012.

[124] G. Chowdhary, E. N. Johnson, R. Chandramohan, S. M. Kimbrell, and A. Calise, "Autonomous Guidance and Control of Airplanes under Actuator Failures and Severe Structural Damage," *Journal of Guidance Control and Dynamics*, 2012.

[125] M. Idan, M. D. Johnson, and A. J. Calise, "A Hierarchical Approach to Adaptive Control for Improved Flight Safety," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 25, no. 6, p. 1012, 2002.

[126] E. N. Johnson and S. Kannan, "Adaptive Trajectory Control for Autonomous Helicopters," *Journal of Guidance Control and Dynamics*, vol. 28, no. 3, pp. 524–538, 5 2005. [Online]. Available: http://doi.aiaa.org/10.2514/1.6271

[127] A. M. Annaswamy and K. S. Narendra, "Adaptive control of simple time-varying systems," in *Decision and Control, 1989., Proceedings of the 28th IEEE Conference on*, 12 1989, p. 1014?1018 vol.2.

[128] N. Hovakimyan and C. Cao, *1 Adaptive Control Theory: Guaranteed Robustness with Fast Adaptation*. SIAM, 2010.

[129] T. Yucelen and A. Calise, "A derivative-free model reference adaptive controller for the generic transport model," in *AIAA Guidance, Control and Navigation Conference*, Toronto, Canada, 8 2010.

[130] G. Chowdhary, T. Yucelen, M. Mühlegg, and E. N. Johnson, "Concurrent learning adaptive control of linear systems with exponentially convergent bounds," *International Journal of Adaptive Control and Signal Processing*, vol. 27, no. 4, pp. 280–301, 2013.

[131] G. Joshi and G. Chowdhary, "Adaptive control using gaussian-process with model reference generative network," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 237–243.

[132] B. Scholkopf, R. Herbrich, and A. Smola, "A Generalized Representer Theorem," in *Computational Learning Theory*, ser. Lecture Notes in Computer Science, D. Helmbold and B. Williamson, Eds. Springer Berlin / Heidelberg, 2001, vol. 2111, pp. 416–426. [Online]. Available: http://dx.doi.org/10.1007/3-540-44581-1_27

[133] B. Schölkopf and A. J. Smola, *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[134] C. E. Rasmussen and C. K. Williams, *Gaussian process for machine learning*. MIT press, 2006.

[135] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[136] H. Mhaskar, Q. Liao, and T. Poggio, "Learning Functions: When Is Deep Better Than Shallow," *arXiv e-prints*, p. arXiv:1603.00988, Mar 2016.

[137] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review," *International Journal of Automation and Computing*, vol. 14, no. 5, pp. 503–519, Oct 2017. [Online]. Available: https://doi.org/10.1007/s11633-017-1054-2

[138] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *arXiv e-prints*, p. arXiv:1611.03530, Nov 2016.

[139] M. Telgarsky, "Benefits of depth in neural networks," *arXiv e-prints*, p. arXiv:1602.04485, Feb 2016.

[140] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[141] G. V.˜Cybenko, "Dynamic Load Balancing for Distributed Memory Multiprocessors," *Journal on Parallel and Distributed Computing*, vol. 7, pp. 279–301, 1989.

[142] S. Boyd and S. Sastry, "Necessary and Sufficient Conditions for Parameter Convergence in Adaptive Control," *Automatica*, vol. 22, no. 6, pp. 629–639, 1986.

[143] G. Larchev, S. Campbell, and J. Kaneshige, "Projection operator: A step toward certification of adaptive controllers," in *AIAA Infotech@ Aerospace 2010*, 2010, p. 3366.

[144] K. S. Narendra and A. M. Annaswamy, *Stable adaptive systems*. Courier Corporation, 2012.

[145] J. Luo and C. E. Lan, "Control of wing-rock motion of slender delta wings," *Journal of Guidance, Control, and Dynamics*, vol. 16, no. 2, pp. 225–231, 1993.

[146] M. M. Monahemi and M. Krstic, "Control of wing rock motion using adaptive feedback linearization," *Journal of guidance, control, and dynamics*, vol. 19, no. 4, pp. 905–912, 1996.

[147] B. Schölkopf, R. Herbrich, and A. J. Smola, "A generalized representer theorem," in *International conference on computational learning theory*. Springer, 2001, pp. 416–426.

[148] R. M. Sanner and J.-J. Slotine, "Stable adaptive control and recursive identification using radial gaussian networks," in *Decision and Control, 1991., Proceedings of the 30th IEEE Conference on*. IEEE, 1991, pp. 2116–2123.

[149] K. Y. Ho et al., *High-level feedback control with neural networks*. World Scientific, 1998, vol. 21.

[150] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural computation*, vol. 3, no. 2, pp. 246–257, 1991.

[151] K. S. Narendra, "Neural networks for control theory and practice," *Proceedings of the IEEE*, vol. 84, no. 10, pp. 1385–1406, 1996.

[152] H. Patino and D. Liu, "Neural network-based model reference adaptive control system," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 30, no. 1, pp. 198–204, 2000.

[153] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[154] R. C. Grande, G. Chowdhary, and J. P. How, "Nonparametric adaptive control using gaussian processes with online hyperparameter estimation," in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*. IEEE, 2013, pp. 861–867.

[155] C. Rohrs, L. Valavani, M. Athans, and G. Stein, "Robustness of continuous-time adaptive control algorithms in the presence of unmodeled dynamics," *IEEE Transactions on Automatic Control*, vol. 30, no. 9, pp. 881–889, 1985.

[156] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the American mathematical society*, vol. 68, no. 3, pp. 337–404, 1950.

[157] Y. Miyahara, "Ultimate BOundedness of the Systems Governed by Stochastic Differential Equations," *Nagoya Math Journal*, vol. 47, pp. 111–144, 1972.

[158] R. Khasminskii, *Stochastic stability of differential equations*. Springer Science & Business Media, 2011, vol. 66.

[159] R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE Control Systems Magazine*, vol. 12, no. 2, pp. 19–22, 1992.

[160] T. Kailath, *Linear systems*. Prentice-Hall Englewood Cliffs, NJ, 1980, vol. 156.

[161] H. Xu and S. Mannor, "Robustness and generalization," *Machine learning*, vol. 86, no. 3, pp. 391–423, 2012.

[162] S. A. van de Geer and P. Bühlmann, "On the conditions used to prove oracle results for the lasso," *Electron. J. Statist.*, vol. 3, pp. 1360–1392, 2009. [Online]. Available: https://doi.org/10.1214/09-EJS506

[163] G. Chowdhary and E. Johnson, "A singular value maximizing data recording algorithm for concurrent learning," in *Proceedings of the 2011 American Control Conference*, June 2011, pp. 3547–3552.

[164] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," *arXiv preprint arXiv:1505.05424*, 2015.

[165] D. Jakubovitz, R. Giryes, and M. R. D. Rodrigues, "Generalization Error in Deep Learning," *arXiv e-prints*, p. arXiv:1808.01174, Aug 2018.

[166] Y. Kwon, J.-H. Won, B. J. Kim, and M. C. Paik, "Uncertainty quantification using bayesian neural networks in classification: Application to ischemic stroke lesion segmentation," 2018.

[167] G. Joshi and R. Padhi, "Robust control of quadrotors using neuro-adaptive control augmented with state estimation," in *AIAA Guidance, Navigation, and Control Conference*, 2017, p. 1526.

[168] H. Yu and B. M. Wilamowski, "Levenberg-marquardt training," *Industrial electronics handbook*, vol. 5, no. 12, p. 1, 2011.

[169] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[170] J. G. S. Paiva, W. R. Schwartz, H. Pedrini, and R. Minghim, "An approach to supporting incremental visual data classification," *IEEE transactions on visualization and computer graphics*, vol. 21, no. 1, pp. 4–17, 2014.

[171] R. Turner, "A model explanation system," in *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2016, pp. 1–6.

[172] F.-Y. Tzeng and K.-L. Ma, *Opening the black box-data driven visualization of neural networks*. IEEE, 2005.

[173] T. Zahavy, N. Ben-Zrihem, and S. Mannor, "Graying the black box: Understanding dqns," in *International Conference on Machine Learning*, 2016, pp. 1899–1908.

[174] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[175] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[176] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea, "Visualizing the hidden activity of artificial neural networks," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 101–110, 2016.

[177] X. Wang, P. Tino, M. A. Fardal, S. Raychaudhury, and A. Babul, "Fast parzen window density estimator," in *International Joint Conference on Neural Networks*, 2009, pp. 3267–3274.

[178] J. Ham, D. D. Lee, L. K. Saul et al., "Semisupervised alignment of manifolds." in *AISTATS*, vol. 120. Citeseer, 2005, pp. 120–127.