CONTEXT AWARE ADAPTIVE POLICY SELECTION

A Dissertation
Presented to
The Academic Faculty

By

Anthony Liu

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science in the
College of Computing

Georgia Institute of Technology

May 2020

Copyright © Anthony Liu 2020

CONTEXT AWARE ADAPTIVE POLICY SELECTION

Approved by:

Dr. Byron Boots
Paul G. Allen School of Computer Science and Engineering
University of Washington

Dr. Mark Riedl
College of Computing
Georgia Institute of Technology

Date Approved: April 30, 2020

ACKNOWLEDGEMENTS

I would like to thank my mentor Dr. Byron Boots for his continuous support and professional guidance throughout my time in his lab. He has been an incredible source of knowledge in the classroom and in my research, and he has provided me with many opportunities to grow as a researcher. I am also grateful the guidance of Nolan Wagner, who has provided numerous keen insights and constructive suggestions with this work.

Additionally, I would like to thank my research partner Hemanth Sarabu for his invaluable contributions in helping with the formulation of our approach in this document, as well as with the collection of preliminary results. Specifically, the results presented in this work with respect to the Mars Rover simulations is derived from Hemanth's work during his internship with JPL, and Hemanth has provided remarkable guidance and helpful contributions in extending our research to future work with MPC and Autorally. Regarding the Mars Rover simulations, I would also like to acknowledge Shreyansh Daftry's and Dr. Masahiro Ono's help as they provided noteworthy guidance over the work done with the rover.

Furthermore, I would like to thank all of my lab mates in the Robot Learning Lab and the students in the Georgia Tech AI community for being and inspiring group of colleagues and peers. I have grown a lot during my time here, and will always remember the great times that I have had.

Last but not the least, I would like to thank my family for supporting me throughout my undergraduate career.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In of optimal control and reinforcement learning, the difference in the performance of a state-of-the-art policy and a mediocre one is minuscule in comparison to their difference in amortized computational cost. Further, in certain situations the mediocre policy will be able to perform as well as the state-of-the-art policy, and it will all the while have used significantly less computational resources to do so. This phenomenon is a consequence of the necessity for additional compute to solve difficult scenarios; however, while sparsely occurring, these scenarios can be catastrophic for most planning tasks. In this work, we focus on addressing this imbalance between performance and computational cost in the context of planning. We combine ideas that have been prevalent in other machine learning problems and in Hierarchical Reinforcement Learning, and propose a Context-Aware Adaptive Policy Selector (CAAPS). We utilize our selector to create a meta-policy which can minimize these catastrophic states (thus maximizing the policy's ultimate performance), while also minimizing the computational cost necessary to run the policy. Our meta-policy accomplishes this by adaptively selecting from a set of pre-trained candidate policies which vary in performance and complexity, and we show that in certain environments we are able to plan trajectories at near-optimal performance while minimizing the amortized computational cost.

CHAPTER 1
INTRODUCTION

In this work, we investigate the trade-off between computational complexity and performance in control problems, and we propose techniques to adaptively maximize a controller's performance while minimizing its computational cost.

1.1 Motivation

Many complex tasks can be formalized under the form of optimal control problems with discrete-time dynamics and transitional costs. These control problems can vary from a wide range of areas including optimally playing difficult games with massive state spaces such as Go [1, 2] to routing network packets [3] to controlling autonomous vehicles [4, 5, 6, 7, 8, 9, 10, 11]. Within these problems, various techniques have attracted significant attention from researchers, and two key subjects that have risen here are deep reinforcement learning (Deep RL) and model predictive control (MPC). Between these, Deep RL methods infer closed-loop policies for stochastic optimal control problems using sample trajectories gathered from simulations or real systems [12], while MPC methods exploit an explicit formulation of the environment and solves for optimal trajectories in an open-loop, receding horizon manner [13]. While the research surrounding MPC has primarily been dedicated towards stabilizing complex control problems, much of the emphasis in RL research has been in the scalability and efficiency of the learning process. Yet, a topic that has seen little recent work in both areas is the concept of balancing complexity and difficulty with adaptive computation in real world systems.

In optimal control problems, there is often a large variance in the difficulty of calculating the optimal trajectory for a given state [14]. For example, in the instance of an autonomous vehicle, it would be considerably easier to keep the vehicle in the bounds of a road than it would be to avoid obstacles such as trees or debris. For controllers that attempt to solve such problems, there is an order of magnitude difference between the complexity and computational cost of a policy that reasonably handles these tasks versus a policy performing on state-of-the-art benchmarks [15, 14]. Because of the varying degrees of difficulty in examples, most techniques in general struggle to solve harder examples without exponentially increasing their complexity. In other problem spaces such as image recognition, we have seen various techniques to create prediction models that use the minimum necessary complexity per input in order to achieve an accurate prediction [15, 16, 17, 18, 19]. Since many physical robotics environments are constrained on both compute resources and energy capacity, the benefits of such techniques in optimal control is abundantly clear; however, there is little previous work in applying this idea to controllers in physical systems.

1.2 Contributions

In this thesis, we will propose a simple Context-Aware Adaptive Policy Selection (CAAPS) algorithm which learns a policy selector that samples from a pool of pre-trained policies in order to produce an adaptive meta-policy. Our technique exploits the fact that complex controllers are only required for difficult states, while simple controllers will be sufficient for most situations, thus our meta-policy outputs optimal trajectories at reduced amortized computational cost. Because of this, we can achieve faster control decisions while still designing for the worst-case and free up limited compute resources for auxiliary tasks or improved battery life.

We will demonstrate the feasibility of CAAPS by training a policy selector on two state-of-the-art path planners on the JPL ground rover simulator [20, 21]. Notably, our meta-policy is able to match the performance of the best policy in the set, while spending roughly the same amount of computational time as the cheaper policy.

Finally, we will apply CAAPS to complex RL tasks in several OpenAI Gym environments [22], and we will demonstrate that by varying the parameterization of the algorithm we can yield varying meta-policies that optimize the amortized computational cost to various degrees, while minimizing the reduction in performance. Additionally, we highlight the limitations that come into play with regards to the characteristics of the candidate policies, and we show their effects on the performance of our meta-policy.

1.3 Overview of thesis

In Chapter 2, we will introduce past research on adaptively balancing the trade-off between performance and computational cost in the context of other Machine Learning domains. In Chapter 3, we will present the necessary technical background for the rest of the paper, as well as describe our approach for designing our policy selection algorithm in the context of various environments. In Chapter 4, we will outline our initial experiments with learning a meta-policy selector. Then, in Chapter 5, we present our final results with a general meta-policy selector, as outline our measures to validate these results. Finally in Chapter 6, we summarize our findings and contributions, and describe future work on this subject.

CHAPTER 2
LITERATURE REVIEW

2.1 Adaptive Computation in Machine Learning

Adaptive computation refers to the ability for an algorithm to dynamically adapt to its environment and provide different results depending on the context of its input. In the context of this work, we refer to adaptive computation techniques as methods to dynamically change the complexity of a function in order to minimize the computational work needed to output a desired result. In traditional machine learning problems, a fair amount of progress has been made in this regard, and the most significant contributions fit into either *prediction cascades* or *anytime predictions*.

2.1.1 Prediction Cascades

Prediction cascades were largely popularized by Viola and Jones [23], who constructed a cascade of increasingly complex image classifiers to iteratively reduce the possible false positive predictions for an input image. While Viola and Jones are credited for coining the term cascades, earlier work by Rowley et. al. [16] implement cascading neural networks by having a network arbitrate and merge the output of a set of filter networks that run on different windows of an input image. Unlike these two classic approaches which specifically target the classification task of face detection, later work has explored utilizing prediction cascades with deep neural networks for general multi-class classification problems. Importantly, the majority of these later works focus on using cascades to reduce the amortized computational time for classifying new examples.

The first work worth noting is the work with network selection and evaluation by Bolukbasi et al. [15]. Here, they propose two different schemes of an adaptive cascades. Their

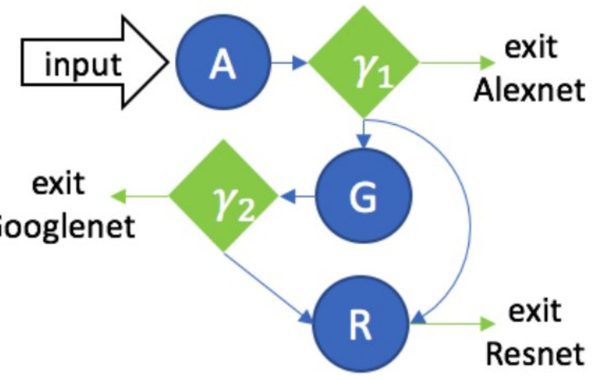


Figure 2.1: Example adaptive network selection topology proposed by Bolukbasi et al. [15] with Alexnet(A), GoogLeNet(G), and Resnet(R). Green γ blocks denote the selection policy. γ_1 evaluates Alexnet, and decides to jump directly to Resnet or send the input to a GoogLeNet$\rightarrow$Resnet cascade that performs a similar evaluation with γ_2.

first contribution adds an evaluation early exit step between the components of a deep network. If an input is correctly classified from just the early layers of the system, the result is exited and they avoid the computation associated with the full evaluation of the network. Second, they propose a method for creating a directed acyclic graph of different networks, in increasing order of complexity, and extend their first contribution to create an adaptive network selection policy that determines which networks to pass new examples through (See Figure 2.1).

Next, Streeter [18] proposes a greedy algorithm to form a cascade from a pool of candidate networks. This proposed algorithm not only automates the construction of the cascade, but provides guarantees that the cascade will be as accurate as a reference target network. The base algorithm that Streeter proposes attaches an auxiliary classifier to predict the accuracy from features of network output (e.g. entropy) for classification tasks and exit at an arbitrary threshold.

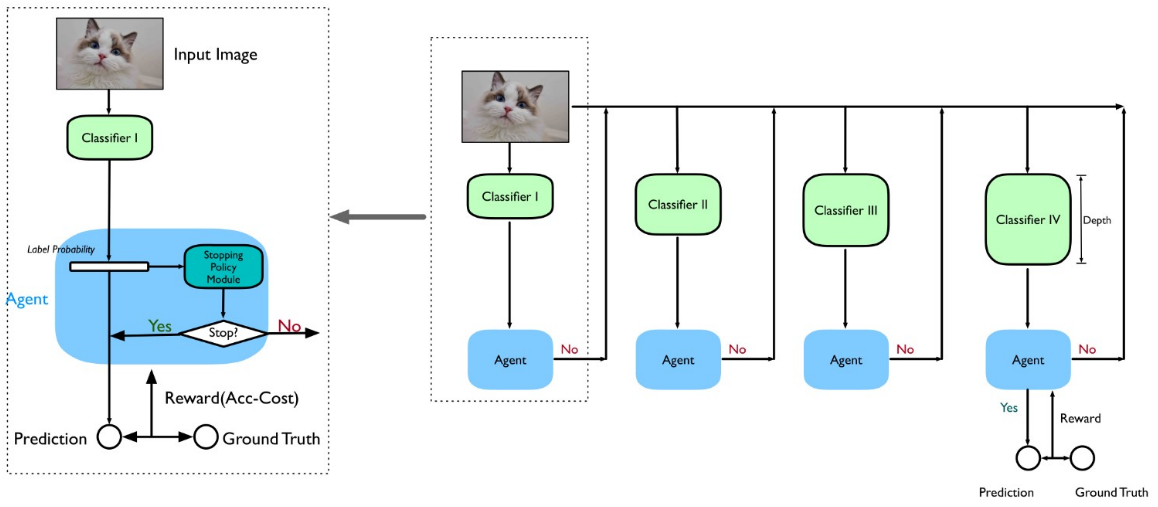


Figure 2.2: Classification cascade using an RL Agent to determine exiting strategy proposed by Guan et al. [17]. Each agent takes the label probability of a classifier's top layer and decides whether to stop or continue.

Guan et al. [17] propose a useful formulation for training a network selection policy using the REINFORCE algorithm (See Section 3.1.3 on Policy Gradient algorithms). Here, they train the network selection policies jointly with the classifiers with the objective to minimize the expected loss, with the energy cost constrained by a desired budget. As shown in Figure 2.2, a separate policy is appended to the end of each classifier, and the policy maps the classifier's output label probability to the decision to exit the cascade or continue. The policies are trained with the following reward function:

$$R_k(x, y, \hat{y}) = -\mathcal{L}(\hat{y}, y) - \alpha \sum_{t-1}^{k-1} \mathcal{F}_t \tag{2.1}$$

Where $\mathcal{F}_t$ is energy cost for classifier t and the k^{th} policy is rewarded with the negative loss minus the total accumulated energy cost from the first step to that policy. It is also worth noting that they explored the case of using pre-trained classifiers in the cascade, and the reported benefits of end-to-end training over the pre-trained scenario were not substantial. Finally, Wang et al. [19] create a similar cascade topology to Guan et al., but they use a differentiable entropy based cost objective which allows for direct cost based optimization instead of reinforcement learning.

2.1.2 Anytime Neural Networks

Anytime predictors serve as a different form of adaptive computation than prediction cascades. Instead of minimizing average test-time computational cost while minimizing sacrifice in accuracy, anytime predictors produces a fast and crude initial prediction and continues to refine and improve it as the computational budget allows. Typically these predictors train auxiliary exit policies attached to various intermediate points along the model. Although not mentioned in their paper, the adaptive early-exit cascade in Bolukbasi et al.'s work [15] mentioned previously can loosely be considered an example of one such Anytime Network. While early work with anytime predictors was in the context of classical machine learning predictors [24, 25], recent work by Huang et al. [26] used clever architecture to derive an anytime image classification predictor. Additionally, Hu et al. [27] derived an adaptive weight scheme for training losses to create generalized anytime neural networks.

In the context of reinforcement learning, there have been various work over the past several decades on applying anytime prediction to classical reinforcement learning (*not deep RL*) [28, 29, 30]. While orthogonal to our work, *anytime learning* is worth mentioning due to their organic nature of iteratively improving on an action.

2.2 Adaptive Computation in Robotics

Our implementation of our policy selection algorithm fits closely with the formulation of Hierarchical Reinforcement Learning, in which the work for a task is distributed between a hierarchy of sub-policies and meta-policies. However, the majority of recent research in this subject has been with solving complex long-horizon problems with sparse and delayed rewards [31, 32, 33, 34, 35]. The appeal of Hierarchical Reinforcement Learning is that it focuses on decomposing tasks into small hierarchical skills, and training different sub-policies and meta-policies to distribute the work. Generally, this allows for methods to significantly improved sample efficiency [33, 36], increase robustness [34, 37, 36] or allow for the parellization of training process[37]. Yet, there is little work to be found on designing HRL architectures for adaptive computation.

CHAPTER 3
CONTEXT-AWARE POLICY SELECTOR FORMULATION

3.1 Reinforcement Learning Preliminaries

Reinforcement learning is the process of approximating optimal decision-making in physical and artificial environments. In the past, Reinforcement Learning has been used to beat top-ranked Go champions [2], route network packets [3], form stock trading strategies [38], and play complex multiplayer games competitively [32, 39, 40].

Most tasks in reinforcement learning and optimal control can be formalized as a time-discrete sequential decision process. The decision-maker, known as the *agent*, interacts with its surrounding *environment*. In this interaction, an agent at time $t \in T$ will make some observation about the state of its environment, $s_t \in S$, and perform some action $a_t \in A$ upon the environment. In response, the environment will output a reward $r_{t+1} \in \mathbb{R}$ for the action a_t. This interaction loop is shown below in Figure 3.1.

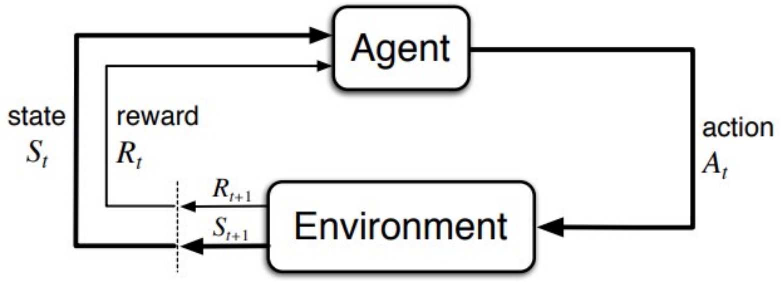


Figure 3.1: The Agent-Environment interaction in a Markov decision process from Reinforcement Learning: An Introduction [12]

Note that notation may vary between reinforcement learning and optimal control tasks. Reiterating from above, for reinforcement learning we model our environment with an agent making a state observation $s_t \in S$, performing an action $a_t \in A$, and receiving a reward $r_{t+1} \in \mathbb{R}$. However for optimal control tasks, we typically describe the task by a controller at state $x_t \in X$ outputting a control $u_t \in U$ and incurring a control cost $c_{t+1} \in \mathbb{R}$. In particular, the distinction between reward and cost is important, as the goal of learning an optimal policy involves maximizing our accumulated reward for reinforcement learning and minimizing our accumulated cost for optimal control.

3.1.1 Markov Decision Processes

The Markov decision process (MDP) is a useful abstraction for modeling stochastic environments. Here, each observation s_t is a *Markov state* if it summarizes all previous observations, so all the necessary information for decision making is retained. If the MDP is fully observable, every state s_t is a markov state and the Markov property holds. This means that the future states and rewards depend only on the current state, and are independent of the previous states and actions that we experience. In this work we will only be dealing with fully observable MDPs, so for the remainder of this thesis we can assume that this property is true.

When we model an environment as an MDP, we define it as a 4-tuple consisting of (S, A, P, R), where

1. S is the set of all valid states.
2. A is the set of all valid actions.
3. $P(s_t, a_t, s_{t+1})$ is the probability that we arrive in state s_{t+1} after taking action a_t in state s_t.
4. $R(s_t, a_t, s_{t+1})$ is the associated immediate reward for arriving in state s_{t+1} after taking action a_t in state s_t.

In tasks which have a finite horizon T, there is a terminal state in which our task ends. For these tasks, we define the sequence of experiences from $t = 1$ to $t = T$ as an episode. Then, within an episode, we can define the total episode return from time t as $R_t = \sum_{k=t}^{T} r_k$. However, when trying to maximize R_t, we can observe that this formulation makes no distinction between if our agent picks up a reward immediately or if our agent decides to sit and wait until the end [12]. Therefore, we define our return using a time-discounted reward $R_t = \sum_{k=t}^{T} \gamma^k r_k$ where $0 < \gamma \leq 1$ is our discounting factor [12]. Lastly, we define the behavior of an agent by a *policy* $\pi(s) = a$ which is a function that maps state to an action.

3.1.2 Value-Based Reinforcement Learning

In Reinforcement Learning, we define a Value function $V^\pi(s)$ as the expected return when following our policy π from a state s,

$$V^\pi(s) = \mathbb{E}_\pi[R_t \mid s_t = s] \tag{3.1}$$

$$= \mathbb{E}_\pi\left[\sum_{k=t}^{T} \gamma^k r_k \mid s_t = s\right] \tag{3.2}$$

Likewise, we define a action-value function $Q^\pi(s, a)$ as the expected return when following our policy π *after* taking action a from state s,

$$Q^\pi(s, a) = \mathbb{E}_\pi[R_t \mid s_t = s] \tag{3.3}$$

$$= \mathbb{E}_\pi\left[\sum_{k=t}^{T} \gamma^k r_k \mid s_t = s\right] \tag{3.4}$$

Since our goal in reinforcement learning is to maximize our award in an episode, we generally want to find the optimal value and action-value functions $V^*(s) = \max_\pi V^\pi(s)$, $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ for every action and state, and thus our reinforcement learning objective is typically to try to estimate $V^*(s)$ or $Q^*(s, a))$. From this, we can get the optimal policy $\pi^*(s, a) = \text{argmax}_\pi(Q^\pi(s, a))$.

3.1.3 Policy Gradient Algorithms

Policy gradient methods are some of the most versatile methods for reinforcement learning due to their fast training speed and flexibility with discrete and continuous spaces. Their objective is to optimize the parameters θ of the policy π_θ in order to maximize the expected discounted reward of running the policy. Since the work in this thesis revolves around episodic environments, we can define our objective with the value function from the agent's initial state, s_0.

$$J(\theta) = V^{\pi}(s_0) \tag{3.5}$$

$$= \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, \pi_\theta(s_t))\right] \tag{3.6}$$

We can use the action-value function definition in equation 3.4 to expand and simplify,

$$J(\theta) = \mathbb{E}\left[r(s_t, \pi(s_t)) + \gamma \sum_{k=t+1}^{T} \gamma^k r_k\right] \tag{3.7}$$

$$= \mathbb{E}\left[r(s_t, \pi(s_t)) + \gamma Q^{\pi}(s_{t+1}, a_{t+1})\right] \tag{3.8}$$

Then, we optimize our policy by taking steps in the direction of the policy gradient $\nabla_\theta J(\theta)$ [41]. If we solve for $\nabla_\theta J(\theta)$ we get,

$$\nabla_\theta J(\theta) = \mathbb{E}_t\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) Q^{\pi}(s_t, a_t)\right] \tag{3.9}$$

From here, we have the framework to dive into any policy gradient algorithm. The policy gradient theorem has given rise to many of these algorithms, which differ in how Q^π is estimated [41]. For example, we could set $Q^{\pi}(s_t, a_t) = \sum_{k=t}^{T} \gamma^k r(s_k, \pi_\theta(s_k, a_k)) = V(s_t)$, and we would have the definition of the **REINFORCE** algorithm [42]. More specifically with **REINFORCE**, after every episode, we would go back through each experience $(s_t, a_t, r_{t+1}, s_{t+1})$ and apply the policy update

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) V^{\pi}(s_t) \tag{3.10}$$

Where α is the update size for each time step.

Another more advanced policy gradient approach is Advantage Actor-Critic implementations, which not only perform well in parallelized environments [43] but also significantly reduce the variance in the gradient estimates by using an advantage function

$A^\pi(x_t, u_t) = Q^\pi(x_t, u_t) - V^\pi(x_t)$ for their gradient updates [44]. The value function V^π is approximated by a function approximator (the critic), while an actor approximates the policy π_θ, and the policy gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_t \left[\nabla_\theta \log \pi_\theta(u_t \mid x_t) A^\pi(x_t, u_t)\right] \tag{3.11}$$

Among these actor-critic algorithms, Proximal Policy Optimization (PPO) is a commonly used due to its simplicity in comparison to other state-of-the-art policy optimization algorithms, while being incredibly sample efficient. [45].

3.1.4 Evolutionary Algorithms

One of the significant drawbacks to standard Reinforcement Learning techniques is the slow training speed due to requiring small gradient updates [46]. Additionally, the stochastic nature of needing to create a Monte-Carlo estimate of the policy gradient can introduce instability in the training process [47]. This becomes an issue in our case of training a policy selection algorithm, as the transition between different policies in our selection set can easily destabilize training with traditional RL algorithms.

Evolutionary algorithms, such as Cross Entropy Method, do not have this issue since they do not rely on stochastic gradient updates, and they have the additional benefit of being very efficient in parameter search with the provision that the parameter search space is not significantly large [46].

The Cross Entropy Method (CEM) is a commonly used evolutionary algorithm due to its quick convergence and easy implementation [48]. The standard algorithm searches for parameters by randomly sampling a batch of parameters from a random distribution (typically a gaussian distribution) parameterized by an initial set of means μ and variances Σ. Then, it evaluates the fitness of each sample based on a user-defined optimization objective, and updates the μ and Σ using the top performing set of samples in the sampled set [48].

This technique can be easily applied to a MDP for policy optimization by using a variant of $\operatorname{argmax}_\pi V^\pi(s_0)$ as the optimization objective [46, 48].

3.2 Context Aware Policy Selection

To allow for adaptive computing with our policies, we must formulate a meta-policy which utilizes a policy selector that selects the optimal policy from a set of candidate policies,

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(a_t))]. \tag{3.12}$$

In particular, we are selecting the parameterization

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi_\theta(s_t))] \tag{3.13}$$

Since we are focusing on the case where a set of optimal parameters for various parameterizations (models) of the policy π have been learned to reasonable extents and are available to deploy, we call this set of optimal parameters Θ. Owing to the type of model used, each parameterization $\theta^* \in \Theta$ has an associated cost to query during run time and this project aims to learn a stochastic policy selector that regularizes the computational expense associated with the usage of a parameterization at a given time and state. We drop * from θ^* in regards to our candidate policies for the remainder of this document since we provided optimal θs for each parameterization. To this end, we incorporate the expense of using a policy with a certain parameterization $p(\theta)$ into the original cost expression. Ignoring the slight abuse of notation, the reward function is augmented as:

$$\tilde{r} = r(s_t, \pi_\theta) - wp(\theta) \tag{3.14}$$

Here, $p(.)$ is used to weigh the computational cost of using a parameterization θ, and w is scaling factor on $p(\theta)$ that is used to trade computational costs with task costs. We now try to obtain a state-adaptive meta-policy Π with $\theta_t \sim \Pi(x_t)$, (in effect, a policy over policies) at time t using the mapping Π dependent on the current state. Now to select an optimal policy at each state that is also aware of the resources it is using, the optimization problem requires solving the following:

$$\max\ \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \tilde{r}] \tag{3.15}$$

and then we find the optimal meta-policy Π^*:

$$\Pi^* = \underset{\Pi}{\operatorname{argmax}}\, \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi_{\Pi(s_t)}(s_t)) - \gamma^t w p(\Pi(s_t))] \tag{3.16}$$

Note that for this formulation, the parameterizations in Θ and $\tilde{r}$ vary between environment, as different environments have different relationships between reward and complexity.

3.3 Experiment Set Up

In order to demonstrate that we can achieve the optimal meta-policy with a more directed approach, we ran preliminary feasibility studies using the Mars Rover simulator with a policy set Θ of two path planners (See Chapter 4). For simplicity, we created a 4 layer convolutional network policy selection model. We then trained it using a supervised learning approach using data collected from other experiments.

With verification that it is possible to beat cost-reward performance of individual policies, we implement a reinforcement learning method for training a generalizable meta-policy and run the experiments on several OpenAI Gym Environments (See Chapter 5)

CHAPTER 4
PRELIMINARY MARS ROVER EXPERIMENTS

4.1 Preface

For this section, I would like to acknowledge and thank Hemanth Sarabu for his work in conducting these experiments and collecting the results.

4.2 On-board Optimization-based Kinematic Settling (OBKS)

For path planning with the Mars Rover [20], use of approximate clearance techniques unsurprisingly introduces varying levels of conservatism into pose prediction and clearance evaluation algorithms [49]. When used in path planning for collision-checking, algorithms such as Approximate Clearance Evaluation (ACE) is seen to perform better than the state-of-the-art algorithm GESTALT [50, 49] that rejects all paths, including feasible ones that allow the rover to straddle small obstacles and traverse over undulating terrain. By determining a tighter interval on possible states that the rover can occupy, ACE is able reduce this conservatism. It is important to note that the conservatism manifests as false-rejections of candidate paths when used in a motion-primitive path planner. In order to reduce the rate of false-rejections, a more sophisticated algorithm is necessary such that the confidence interval may be further reduced.

High-Performance Spaceflight Computing (HPSC) opens up a variety of avenues to explore owing to the availability of increased computational capabilities. We developed an iterative algorithm for on-board computation that is very much in the vein of those used in ground rover simulation software at JPL [20, 21]. The algorithm named Optimization-Based Kinematic Settling (OBKS) solves a local optimization problem to minimize contact between terrain and rover wheels. It is modelled as a least-squares problem subject to pose constraints on joint angles as determined by rover design limits. As the solution yielded by OBKS is expected to be close to the exact pose of the rover for a given location in a height map, the *interval* of uncertainty is smaller hence, reducing the intrinsic conservatism of the path planner in comparison with ACE.

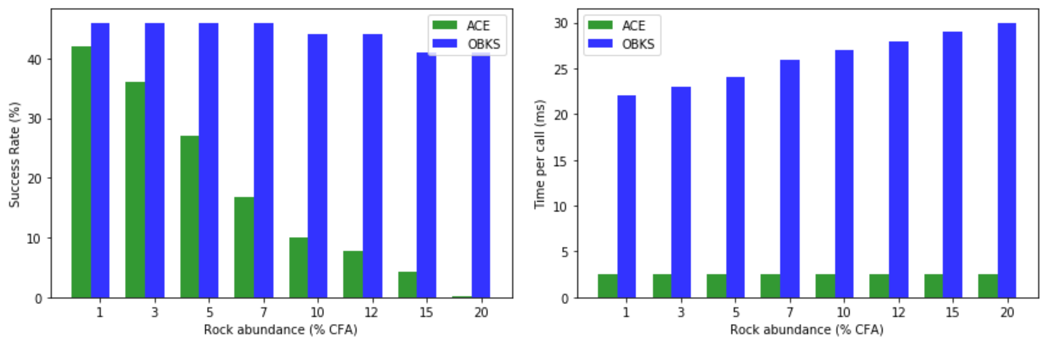


Figure 4.1: Monte Carlo estimates of success rates (left) and query time (right) of ACE and OBKS for random placement on height maps of varying rock abundance (% CFA). A random placement is considered a success if the collision-checking algorithm returns that the location in the map is safe for traversal.

Although the time per query of ACE is lower than that of OBKS (seen in Figure 4.1), the increase in success rate during random placement (seen in Figure 4.1) allows the time taken to generate a path for a 20m traversal to be significantly lower for OBKS in relatively complex terrain (Cumulative Fractional Area greater than 7%). As a consequence of lower conservatism, the OBKS planner is also able to generate paths with lower path inefficiency (seen in Figure 4.2). This algorithm has been implemented on an Nvidia Jetson TX2 board using Ceres solver [51] for Athena Rover and runs with a mean query time of 13.7 ms.

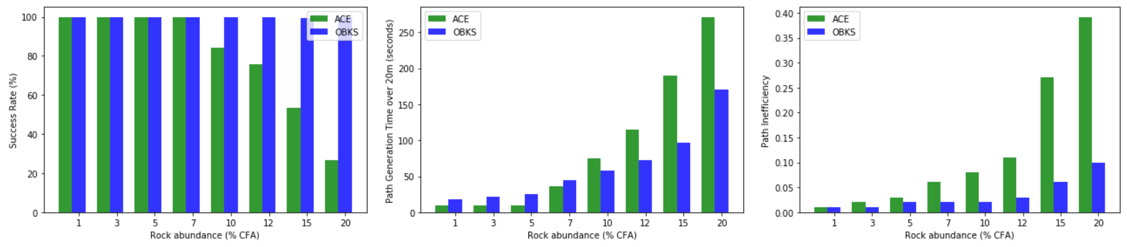


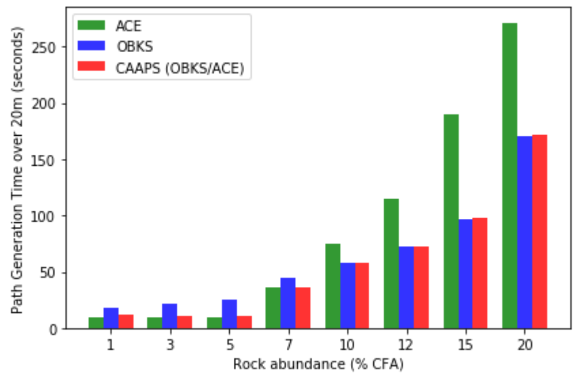
Figure 4.2: Monte Carlo estimates of success rates (left), path generation times (middle), and path inefficiency (right) for an ACE-based planner and an OBKS-based planner for 20m traversals on height maps of varying CFA. A traversal is considered a success if the planner is able to generate a path from the start location to the goal location in the map.

4.3 Mars Rover Path Planning

In this section, we conduct a feasibility study for the formulation of our meta-policy and study this only in the context of computation. Development of OBKS (See section 4.2) for on-board computation on future rover missions provided insight into the algorithm's strengths and weaknesses upon utilization on terrains of varying complexity. While in CFAs greater than 7% the arc-primitive path planner using OBKS outperformed the same planner using ACE for collision-checking, the latter demonstrated an advantage in path generation time with no detriment to path inefficiencies for lower CFAs.

Thus, we developed a supervised learning based meta-policy to select the *optimal* planner for a given state of the JPL Mars rover and its environment. The learned policy is required to select a planner such the path generation time for planner is minimized with a controlled trade-off in probability of success.

We instantiated the meta-policy selector as a 4 layer convolutional neural network which operates on Depth Elevation Maps (DEM) maps as inputs. The model was trained using a supervised learning approach on over 7000 DEM maps spanning 30m x 30m and varying CFAs (1%, 3%, 5%, 7%, 10%, 12%, 15%, and 20%) to achieve mean test ac curacies of 81% and 92% on success prediction and path time generation respectively.



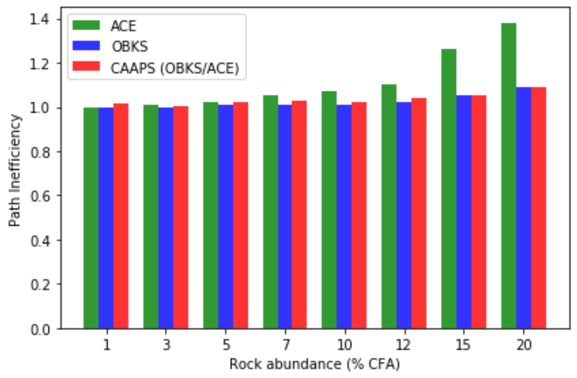


Figure 4.3: Comparison of path inefficiency (left) and path generation time (right) for planner using OBKS, ACE and meta-policy selector that greedily switches between the two.

Preliminary results (summarized in Figure 4.3) indicate that the policy selector is indeed able to select the optimal planner to minimize path generation time (and as a result, path inefficiency due to strong correlation between the two quantities).

CHAPTER 5
CONTEXT-AWARE ADAPTIVE POLICY SELECTION

5.1 Training a policy set

Before we are able to train our policy selector for our Gym benchmarks, we need to pre-train a set of policy networks of varying costs and performance. To achieve this, we implemented an optimized version of PPO2 [45] and trained a batch of baseline policies ranging from linear policies to multi-layer perceptron policies of height 64 and depth 3 on a set of candidate environments. The hyperparameters for training are shown in Table A.1.

Among the results, we observed that for the environments listed in Table 5.1, we were able to see a distinct correlation between each policy's shape and its reward.

Environment	Description	Max episode steps	Reward Threshold
HumanoidBulletEnv-v0	Make a three-dimensional bipedal robot walk forward as fast as possible, without falling over.	1000	N/A
HalfCheetahBulletEnv-v0	Make a two-dimensional cheetah robot robot run as fast as possible.	1000	3000
HopperBulletEnv-v0	Make a two-dimensional one-legged robot hop forward as fast as possible.	1000	2500

Table 5.1: Pybullet Environment Descriptions

In Figure 5.1, we illustrate the energy cost (measured in floating operations per model forward pass) with relation to the average episode return for running the policy. With this, can can create a useful visual measure for evaluating the performance-cost margins of a learned meta-policy in comparison to our baselines. See table B.1 for the precise values of each of the baseline policies.

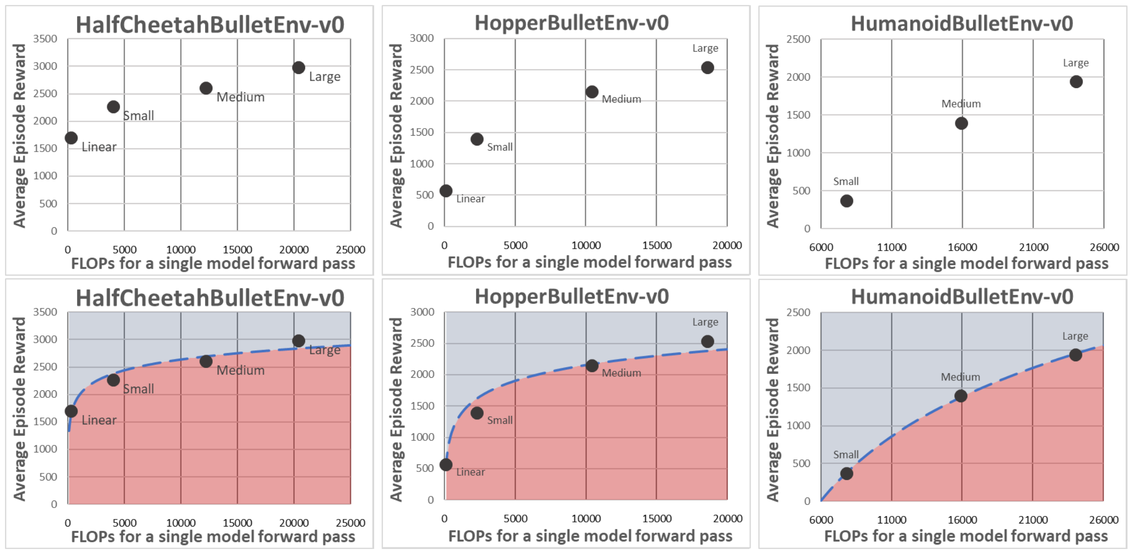


Figure 5.1: **Top**: Plots of *average episode reward* versus *computational cost* (number of FLOPs per forward pass). **Bottom**: Plots of logarithmic linear regression of reward vs. cost points with bounded regions for *superior policies* (blue) and *sub-optimal policies* (red).

5.2 Results

Because training the policy selector using a policy gradient method leads to unstable results, we opted to train the policy using Cross Entropy Method, and since this planner does not actually need to decide the action to run, we are able to keep the parameter space small and converge very quickly. The implementation of our CAAPS algorithm is shown in Section A.3.

The policy selectors that we train all have a single layer of depth $num\_policies \times 4$, and a head which outputs the score of each policy. For policy selection, we take the one-hot max value of the selector output. The results of our experiments are shown below in Figure 5.2, and the numerical values are shown in Table B.2.

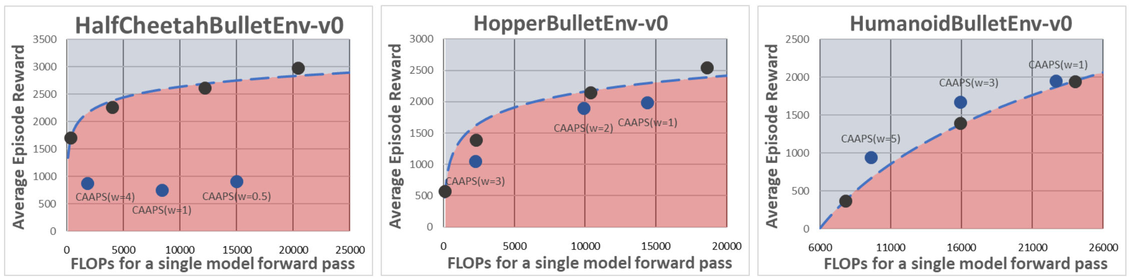


Figure 5.2: Plots CAAPS meta-policy performance versus energy cost, plotted over bounded evaluation regions for *superior policy* (blue) and *sub-optimal policy*

5.3 Discussion

In Figure 5.2, we plot three CAAPS policies trained with varying cost scaling factors w for each test environment. Here, we can observe that the w parameter provides us with the freedom of tuning the amortized energy usage, as scaling the cost to be larger discourages the planner from selecting expensive policies.

Additionally, we observe that each of the the Half-Cheetah and Hopper meta-policies performed worse than the baselines they were derived from. We hypothesize that this negative performance can occur from having non-robust candidate policies, or from having candidate policies which disagree on the optimal action. With the former, we argue that if the individual policies do not generalize to the possible states in the environment, and if the state space which each policy generalizes to has significant variation, then the individual candidate policies may plan trajectories that end in particular states that will lead to catastrophic results for other policies. Alternatively, we argue that a similar catastrophic failure can occur by learning a meta-policy that oscillates between different candidate policies that disagree on the optimal trajectories. In such situation, the process of oscillating between the two competing trajectories will invalidate each of their intended results.

However, regarding the final Humanoid environment, we can say with confidence that we were able to learn a set of parameters to form a meta-policy which was at the minimum able to reproduce the cost-performance margins of our baselines. Notably, we see here that we are even able to learn a set of parameters which resulted in *higher* than baseline performance with respect to their energy cost. We believe that the positive results shown here illustrate potential for the methodology outlined in our work.

CHAPTER 6
CONCLUSION

6.1 Conclusion

In Machine Learning, we can quickly observe how the trade-off between a model's performance and its computational complexity is significant. However, many other research areas have found adaptive computational methods to mitigate this trade off by reducing the computational complexity of a model for easy inputs. Meanwhile, little prior work exists for performing such reductions in optimal control, despite the benefits of freeing up compute power in resource constrained physical systems. Our work here proposes a novel method for reducing the computational power used by optimal control policies, and we show that in certain cases with stochastic policies that we can both reduce the computational load while generally maintaining the optimal performance of our policy. Importantly, in our experiments we highlight the importance of learning a policy-selector which selects from a set of candidate policies which are both, robust and similar. With our preliminary results, we pave the way for further work in adaptive computation for robotics and optimal control.

6.2 Future Work

First, as we have shown with the results in the Hopper and Half-Cheetah environments, there is much work to be done in improving the performance of our methodology. We previously mentioned the potential failure points which can lead to a meta-policy outputting sub-optimal trajectories, which include policy dissimilarity and policy oscillation, and it will be worthwhile to explore these failure points in further detail. Specifically, work should be done in regards to:

1. Developing metrics for describing the similarity between a set of policies in order to

illustrate potential issues with oscillating between sets of policies.

2. Modifying our test environment with random state initializations to learn policies which generalize to a broader subset of the state space.

3. Alternatively to 2., implement various robust adversarial training methods in order to create sets of policies which generalize to a broader subset of the state-space.

4. Increasing the complexity of the policy selector in order to describe the likelihood of success for each candidate policy to a more accurate degree.

5. Enforcing similarity of our candidate policy set by training cheap policies using imitation learning methods on a single complex policy, as opposed to training each policy separately.

6. Using a form of Long-Short Term Memory Network [52] for our meta-policy, and inducing an artificial cost on switching policies in order to minimize oscillating behaviors.

Additionally, as indicted by our results with learning a policy selector for the Humanoid environment, we have shown that there potential for adaptive policy selection. From here, it will be worth exploring more complex implementations of the policy selection architecture. With this, further experiments can include:

7. Forming the meta-policy into a cascading structure, similar to the structures discussed in Section 2.1.1.

8. Including a desired amortized computation budget as an input to the policy selector and learning adaptive meta-policies constrained to the inputted budget as opposed to learning a single meta-policy constrained by our cost function scaled by w.

9. Exploring methods for training our policy selector that are alternative to evolutionary algorithms, such as hybrid evolutionary policy gradients. [53]

Appendices

APPENDIX A

IMPLEMENTATION DETAILS

Parameter	Pybullet Environments
discounting factor	0.99
entropy coefficient	0.0
learning rate	3×10^{-4}
value coefficient	0.5
max gradient norm	0.5
lambda	0.95
policy cliprange	0.2
value cliprange	0.2
num optimization epochs	10
frames per update	32000
num minibatches per update	32
total updates	600

Table A.1: PPO Hyperparameters used to train HalfCheetahBulletEnv-v0, HopperBulletEnv-v0, and HumanoidBulletEnv-v0 policies

Parameter	Pybullet Environments
num samples	1000
num elite	20
num rollouts per sample	10
num optimization epochs	100
initial parameter variance	10
initial parameter mean	0

Table A.2: CrossEntropyMethod Hyperparameters used to train HalfCheetahBulletEnv-v0, HopperBulletEnv-v0, and HumanoidBulletEnv-v0 policy selectors. See Section A.1 for CrossEntropyMethod implementation.

Implementation	Repo
PPO Policy Implementations	`https://github.com/ajliu/torchrl`
CAAPS	`https://github.com/ajliu/caaps`

Table A.3: Links to implementation code

A.1 Algorithm

Algorithm 1: Initializing cost function $p(^*)$

```
Data: $\Theta$
Result: $p(^*)$
for $\theta \in \Theta$ do
    $c[\theta] \leftarrow [\ln(\texttt{FLOPS}(\theta))]$
end
$c \leftarrow c/\max(c)$
Function $p(\theta) : float$ is
    return $c[\theta]$;
end
return $p(^*)$;
```

As we see in 5.1, the relation ship between cost and reward is exponential. Therefore, when defining our cost function we linearize this relationship by taking the log of the cost.

Algorithm 2: Train policy selector using CEM

```
Data: $\Pi,\ p(^*),\ w,\ n$
Result: $\Pi^*$
$\mu \leftarrow 0$;
$\Sigma \leftarrow 1$;
for i:=1 to $n$ do
    fitness $\leftarrow$ [ ]
    $\theta_\Pi s \leftarrow [\,]$;
    for j:=1 to 100 do
        $\theta_\Pi = \text{SAMPLE}(\mu, \Sigma)$;
        fitness $[j] \leftarrow \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi_{\Pi_{\theta_\Pi}(s_t)}(s_t)) - \gamma^t w p(\Pi_{\theta_\Pi}(s_t))]$
        $\theta_\Pi s\,[j] \leftarrow \theta$;
    end
    elite $\leftarrow \theta s\,[\texttt{ArgSort}(\text{fitness})]$
    $\mu \leftarrow \texttt{Mean}(\text{elite})$;
    $\Sigma \leftarrow \texttt{Var}(\text{elite})$;
end
return $\Pi_\mu$
```

Here, we implement a basic Cross Entropy optimizer which samples rollouts of the environment to calculate 3.16. We found that training using standard reinforcement learning algorithms quickly failed due to the instability of the policy selector while switching between different actions. See Table A.2 for hyperparameters used for CrossEntropyMethod runs.

APPENDIX B

RESULTS

HalfCheetahBulletEnv-v0			
Policy	Reward	Num FLOPS	Shape
Linear	1706	312	[]
Small	2264	4096	[64]
Medium	2614	12288	[64, 64]
Large	2982	20480	[64, 64, 64]
HopperBulletEnv-v0			
Policy	Reward	Num FLOPS	Shape
Linear	569	90	[]
Small	1394	2304	[64]
Medium	2150	10496	[64, 64]
Large	2541	18688	[64, 64, 64]
HumanoidBulletEnv-v0			
Policy	Reward	Num FLOPS	Shape
Small	370	7808	[64]
Medium	1398	16000	[64, 64]
Large	1941	24192	[64, 64, 64]

Table B.1: Baseline Policy results

	HumanoidBulletEnv-v0		
Policy	Reward	Num FLOPS	Selector Flops
CAAPS(w=5)	941	8512.51	1128
CAAPS(w=3)	1678	14892.288	1128
CAAPS(w=1)	1955	21570.56	1128
	HalfCheetahBulletEnv-v0		
Policy	Reward	Num FLOPS	Selector Flops
CAAPS(w=4)	877	841.76	960
CAAPS(w=1)	754	7589.68	960
CAAPS(w=0.5)	914	14090.24	960
	HopperBulletEnv-v0		
Policy	Reward	Num FLOPS	Selector Flops
CAAPS(w=3)	1051	1735.02	608
CAAPS(w=2)	1895	9410.01	608
CAAPS(w=1)	1988	13918.93	608

Table B.2: CAAPS Policy results

REFERENCES

[1] "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[3] X. You, X. Li, Y. Xu, H. Feng, J. Zhao, and H. Yan, "Toward Packet Routing with Fully-distributed Multi-agent Deep Reinforcement Learning," *ArXiv*, vol. abs/1905.0, 2019. arXiv: 1905.03494.

[4] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars," *ArXiv*, vol. abs/1604.0, 2016. arXiv: 1604.07316.

[5] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car," *ArXiv*, vol. abs/1704.0, 2017. arXiv: 1704.07911.

[6] H. M. Eraqi, M. N. Moustafa, and J. Honer, "End-to-End Deep Learning for Steering Autonomous Vehicles Considering Temporal Dependencies," *ArXiv*, vol. abs/1710.0, 2017. arXiv: 1710.03804.

[7] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp, "Off-road obstacle avoidance through end-to-end learning," *Advances in Neural Information Processing Systems*, vol. 18, pp. 739–746, 2005.

[8] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile Autonomous Driving using End-to-End Deep Imitation Learning," *Robotics: Science and Systems*, 2018. arXiv: 1709.07174.

[9] V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, and J. K. Hedrick, "Learning a deep neural net policy for end-to-end control of autonomous vehicles," in *Proceedings of the American Control Conference*, Institute of Electrical and Electronics Engineers Inc., 2017, pp. 4914–4919, ISBN: 9781509059928.

[10] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," *Journal of Machine Learning Research*, vol. 15, pp. 627–635, 2011. arXiv: 1011.0686.

[11] J. Zhang and K. Cho, “Query-Efficient Imitation Learning for End-to-End Autonomous Driving,” *Proceedings of the 31st AAAI Conference on Arificial Intelligence (AAAI-17)*, pp. 2891 –2897, 2016. arXiv: `1605.06450`.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second Edi. 1998, ISBN: 978-0262193986.

[13] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, “Constrained model predictive control: Stability and optimality,” *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[14] C. Richter, “Autonomous navigation in unknown environments using machine learning,” PhD thesis, Massachusetts Institute of Technology, 2017, pp. 5450–5455, ISBN: 9781728148786.

[15] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, “Adaptive neural networks for efficient inference,” in *34th International Conference on Machine Learning, ICML 2017*, vol. 2, 2017, pp. 812–821, ISBN: 9781510855144. arXiv: `1702.07811`.

[16] H. A. Rowley, S. Baluja, and T. Kanade, “Neural network-based face detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 23–38, 1998.

[17] J. Guan, Y. Liu, Q. Liu, and J. Peng, “Energy-efficient amortized inference with cascaded deep classifiers,” in *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2018-July, 2018, pp. 2184–2190, ISBN: 9780999241127. arXiv: `1710.03368`.

[18] M. Streeter, “Approximation algorithms for cascading prediction models,” in *35th International Conference on Machine Learning, ICML 2018*, vol. 11, 2018, pp. 7569–7577, ISBN: 9781510867963. arXiv: `1802.07697`.

[19] X. Wang, Y. Luo, D. Crankshaw, A. Tumanov, F. Yu, and J. E. Gonzalez, “IDK Cascades: Fast deep learning by learning not to Overthink,” *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, vol. 2, pp. 580–590, 2018. arXiv: `1706.00885`.

[20] A. Jain, J Guineau, C. Lim, W Lincoln, M. Pomerantz, G. Sohl, and R Steele, “Roams: Planetary Surface Rover Simulation Environment,” *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2003)*, 2003.

[21] A. Jain, J. Balaram, J. Cameron, J. Guineau, C. Lim, M. Pomerantz, and G. Sohl, “Recent developments in the ROAMS planetary rover simulation Environment,” in *IEEE Aerospace Conference Proceedings*, vol. 2, 2004, pp. 861–876, ISBN: 0780381556.

[22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *OpenAI Gym*, 2016. arXiv: `1606.01540`.

[23] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001, pp. I–511–I–518, ISBN: 0-7695-1272-0. arXiv: `arXiv:1011.1669v3`.

[24] Z. Xu, M. J. Kusner, G. Huang, and K. Q. Weinberger, “Anytime representation learning,” in *30th International Conference on Machine Learning, ICML 2013*, 2013, pp. 2113–2121.

[25] Z. Xu, K. Q. Weinberger, and O. Chapelle, “The Greedy Miser: Learning under test-time budgets,” in *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, vol. 2, 2012, pp. 1175–1182, ISBN: 9781450312851. arXiv: `1206.6451`.

[26] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, “Multi-Scale Dense Networks for Resource Efficient Image Classification,” *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2017. arXiv: `1703.09844`.

[27] H. Hu, D. Dey, M. Hebert, and J. A. Bagnell, “Learning Anytime Predictions in Neural Networks via Adaptive Loss Balancing,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3812–3821, 2019. arXiv: `1708.06832`.

[28] A. Bonarini, “Anytime learning and adaptation of structured fuzzy behaviors,” *Adaptive Behavior*, vol. 5, no. 3-4, pp. 281–315, 1997.

[29] L. Breitsameter, “Anytime Learning - The next Step in Organic Computing?” *ArXiv*, vol. abs/1808.0, pp. 1 –7, 2018. arXiv: `arXiv:1808.07590v1`.

[30] J. J. Grefenstette and C. L. Ramsey, “An Approach to Anytime Learning,” in *Machine Learning Proceedings 1992*, Elsevier, 1992, pp. 189–195.

[31] S. Li, R. Wang, M. Tang, and C. Zhang, “Hierarchical Reinforcement Learning with Advantage-Based Auxiliary Rewards,” in *Advances in Neural Information Processing Systems 32*, 2019. arXiv: `1910.04450`.

[32] Z.-J. Pang, R.-Z. Liu, Z.-Y. Meng, Y. Zhang, Y. Yu, and T. Lu, “On Reinforcement Learning for Full-Length Game of StarCraft,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4691–4698, 2019. arXiv: `1809.09095`.

[33] O. Nachum, H. Lee, S. Gu, and S. Levine, “Data-efficient hierarchical reinforcement learning,” in *Advances in Neural Information Processing Systems*, vol. 2018-

Decem, Neural information processing systems foundation, 2018, pp. 3303–3313. arXiv: 1805.08296.

[34] C. Florensa, Y. Duan, and P. Abbeel, "Stochastic neural networks for hierarchical reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2019. arXiv: 1704.03012.

[35] J. Ackermann, *Hierarchical Deep Reinforcement Learning for Multi-Agent Robotic Systems*, 2018.

[36] A. Li, C. Florensa, I. Clavera, and P. Abbeel, "Sub-policy adaptation for hierarchical reinforcement learning," in *International Conference on Learning Representations*, 2020.

[37] R. Nogueira, J. Bulian, and M. Ciaramita, "Learning to Coordinate Multiple Reinforcement Learning Agents for Diverse Query Reformulation," 2018. arXiv: 1809.10658.

[38] Z. Xiong, X.-Y. Liu, S. Zhong, H. Yang, and A. Walid, "Practical Deep Reinforcement Learning Approach for Stock Trading," *ArXiv*, vol. abs/1811.0, 2018. arXiv: 1811.07522.

[39] OpenAI, *OpenAI Five Defeats Dota 2 World Champions*, https://openai.com/blog/openai-five-defeats-dota-2-world-champions/, 2019.

[40] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver, *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*, https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[41] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, 2000, pp. 1057–1063, ISBN: 0262194503.

[42] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[43] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, International

Machine Learning Society (IMLS), 2016, pp. 2850–2869, ISBN: 9781510829008. arXiv: 1602.01783.

[44] T. Degris, P. M. Pilarski, and R. S. Sutton, "Model-Free reinforcement learning with continuous action in practice," in *Proceedings of the American Control Conference*, 2012, pp. 2177–2182, ISBN: 9781457710957.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *ArXiv*, vol. abs/1707.0, 2017. arXiv: 1707.06347.

[46] S. Mannor, R. Rubinstein, and Y. Gat, "The Cross Entropy method for Fast Policy Search," in *Proceedings, Twentieth International Conference on Machine Learning*, vol. 2, 2003, pp. 512–519, ISBN: 1577351894.

[47] D. J. Mankowitz, N. Levine, R. Jeong, Y. Shi, J. Kay, A. Abdolmaleki, J. T. Springenberg, T. Mann, T. Hester, and M. Riedmiller, "Robust Reinforcement Learning for Continuous Control with Model Misspecification," *International Conference on Learning Representations*, 2020. arXiv: 1906.07516.

[48] P. T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, no. 1, pp. 19–67, 2005.

[49] K. Otsu, G. Matheron, S. Ghosh, O. Toupet, and M. Ono, "Fast Approximate Clearance Evaluation for Rovers with Articulated Suspension Systems," *Journal of Field Robotics*, 2018. arXiv: 1808.00031.

[50] S. B. Goldberg, M. W. Maimone, and L. Matthies, "Stereo vision and rover navigation software for planetary exploration," in *IEEE Aerospace Conference Proceedings*, vol. 5, 2002, pp. 2025–2036, ISBN: 078037231X.

[51] S. Agarwal, K. Mierle, and Others, *Ceres Solver*, \url{http://ceres-solver.org}.

[52] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, 2016, pp. 3988–3996. arXiv: 1606.04474.

[53] S. Khadka and K. Tumer, "Evolution-guided policy gradient in reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 2018-Decem, 2018, pp. 1188–1200. arXiv: 1805.07917.