Momentum-Based Variance Reduced Algorithms for Imitation Learning

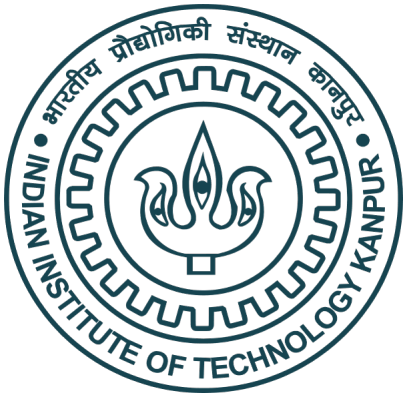
A thesis submitted in fulfilment of the requirements

for the degree of Master of Technology

by

Milind Nakul

18807420



DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

August 2026

Certificate

It is certified that the work contained in this thesis entitled **"Momentum-Based Variance Reduced Algorithms for Imitation Learning"** by **Milind Nakul** has been carried out under my supervision and that it has not been submitted elsewhere for a degree.

August 2026

Prof. Ketan Rajawat
Associate Professor
Department of Electrical Engineering
Indian Institute of Technology Kanpur

Declaration

This is to certify that the thesis titled **"Momentum-Based Variance Reduced Algorithms for Imitation Learning"** has been authored by me. It presents the research conducted by me under the supervision of **Prof. Ketan Rajawat**.

To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and collaborations with appropriate citations and acknowledgments, in line with established norms and practices.

Milind Nakul
BT-MT Dual Degree
Department of Electrical Engineering
Indian Institute of Technology Kanpur
Kanpur 208016

Abstract

Name of the student: **Milind Nakul** Roll No: **18807420**

Degree for which submitted: **M.Tech.** Department: **Electrical Engineering**

Thesis title: **Momentum-Based Variance Reduced Algorithms for Imitation Learning**

Thesis supervisors: **Prof. Ketan Rajawat**

Month and year of thesis submission: **August 2026**

The demand for autonomous intelligent systems has been rapidly increasing in the modern technological world. This has prompted the research community to develop intelligent robotic systems that are capable of working alongside humans and learning from their behaviour. The training of these systems needs to be efficient and robust. This ensures that the agent is able to adapt to slight changes in its working environment. One field that deals with developing algorithms for training such autonomous systems is Imitation learning (IL). It involves an agent learning to imitate the behaviour of an expert. The task is to develop agents that closely mimic the expert given the expert demonstrations or the expert itself. IL has a number of practical applications such as autonomous driving, robotic surgery, gaming, and many more. The quality that makes IL suited for such applications is the availability of huge number of expert demonstrations.

In IL, an important challenge is to develop algorithms that are sample efficient and robust to slight changes in the working environment. These requirements are necessary for allowing any practical application of IL in the above mentioned fields. We aim to address these issues by developing a SGD based algorithm for IL. We first derive the expression for an unbiased estimate of the gradient of the IL objective and plug it into the well known SGD optimization algorithm. The SGD algorithm helps in reducing the number of parameters

that need to be tuned. We also demonstrate the theoretical guarantees of the SGD algorithm for IL. Further, in order to make the algorithm more sample efficient we explore recursive-momentum based variance reduced algorithm for IL namely stochastic recursive momentum (STORM). We also explore some non-smooth non-convex optimization methods and their possible application in developing IL algorithms.

In order to validate our claims we test the developed algorithms in the obstacle avoidance task and classic control environment of OpenAI Gym, the cart pole.

Acknowledgements

I wish to acknowledge and thank everyone who has contributed to this thesis in direct and indirect ways. I feel blessed to have the opportunity to work under my supervisor Prof. Ketan Rajawat. I learned a lot from his simple and decent personality, his research methodology and ways to approach problems. His constant support, motivation, and continuous monitoring have been key guiding forces. I would also like to thank Prof. Rohit Budhiraja, with whom I had the opportunity to work during my undergradute project. The time spent in the several meetings with him and my PhD senior Anupama Rajoriya has been a great learning experience for me. I want to express my gratitude to my PhD mentor, Mohan Krishna. He has excellent research aptitude. Really this work could not have been done without his help, support and continuous encouragement at all the points. I owe special acknowledgement to all my friends for being with me throughout this journey and making this journey wonderful. I am grateful to my parents and family who have supported me at every moment in my life. Finally, I am very thankful to the institution, IIT Kanpur for providing a pleasurable stay and an entirely bright direction to my future.

Contents

List of Figures

List of Tables

Abbreviations

IL	Imitation Learning
RL	Reinforcement Learning
SGD	Stochastic Gradient Descent
STORM	Stochastic Recursive Momentum
BC	Behavioral Cloning
SMILe	Stochastic Mixing Iterative Learning
DAGGER	Dataset Aggregation
RRM	Repeated Risk Minimization
RGD	Repeated Gradient Descent

Symbols

θ	Model parameters
π_θ	Policy
π^*	Expert Policy
τ	Trajectory
$l_t(s_t, \pi_\theta)$	Loss at time step t
F	IL objective
f	Loss corresponding to the stochastic gradient
$p(\tau; \theta)$	Distribution of the trajectories

Dedicated to family.

Chapter 1

Introduction

1.1 Motivation

Learning a behaviour by observation and emulation is called imitation learning. The agent acquires the skill through observing demonstrations of the expert's behaviour. Several methods had been proposed to teach autonomous behaviour, with artificial neural networks, deep learning, reinforcement learning, etc. showing the most promise. The first fully autonomous vehicle was driven using ANN. But, the trained agent was unable to recover from its mistakes. While reinforcement learning's practicality stems from its emphasis on exploration and exploitation during the learning process, the method's computing complexity stems from the fact that it attempts to provide the final policy only after exhaustively examining the state space. The need for knowing a good reward function in reinforcement learning is also a bottleneck in several of its practical applications. In addition to these problems, it is laborious to model the robot's physiology and its surrounding environment within the context of the reinforcement learning framework. This points to the urgent need for a straightforward structure.

As a result of its prospective simplicity in framework, imitation learning has been seen as a promising method for training intelligent robots. Imitation learning is particularly well-suited to scenarios where a robot must work side-by-side a human, such as in the manufacturing, elder care, and service industries, easing the burden on both human workers and carers. Imitation learning is not limited to robots; it can be used in any system that requires autonomous behaviour similar to that of human specialists, including video games and mobile apps. Using imitation learning to improve trajectory planning of autonomous

system is a promising new area of study since it allows for learning with much less data than traditional methods and teaches the agent to correct its own errors.

1.2 Contribution of the Thesis

We study some basic imitation learning algorithms and implement them for the obstacle avoidance task and Gym classical control environments, cart pole. We then look at the basic objective of the IL task and derive an unbiased estimate of the gradient of the objective in the IL problem. We then plug this gradient into the well-known SGD algorithm and develop the SGD for imitation learning algorithm. We then derive the theoretical guarantees for the SGD algorithm for imitation learning. In order to make the proposed algorithm more sample efficient, we explore variance reduced methods. Namely the STORM algorithm and derive the theoretical guarantees for the STORM algorithm. We also explore the domain of non-convex non-smooth optimization and its application in IL. Finally we compare all the proposed algorithms in for two different tasks, namely the obstacle avoidance task and Gym classical control environments, cart pole.

1.3 Organization of the Thesis

In chapter 2, we look at the background for IL. We delve deep into some of the existing methods and their limitations. We also look at some of the optimization algorithms which have been used in this thesis. In chapter 3, we propose the IL objective and derive an unbiased estimator of the gradient of the IL objective. We also propose the SGD and STORM algorithms for IL. We also look at the theoretical guarantees of these algorithms. Finally, we explore the domain of non-convex non-smooth optimization and its application in imitation learning.

In chapter 4, we present environment design for the simulated applications, design choices for imitation learning and parameter selection and several numerical results for the Obstacle avoidance and Cart Pole environments.

In chapter 5, we present the conclusions of our work along with the potential future research directions for this work

1.4 Literature Review

Imitation Learning: Imitation learning is the act of learning to mimic expert behavior. The agent wants to use the training data (expert input-output pairs) in order to learn efficiently a policy which is as close to the expert actions as possible. The most basic algorithm to solve the imitation learning problem is behavioral cloning (BC) [1]. In the method proposed we directly fit a supervised learning model on the expert demonstrations. This particular approach is only effective when dealing with large data sets because small data sets are susceptible to compounding errors due to shift in the distribution [1, 2]. In order to overcome the failure of the BC algorithm, the authors in [1] proposed the forward training and stochastic mixing iterative learning (SMILe) algorithms. The forward training algorithm trains a non-stationary policy for each time step of the learning task. The authors established a sub-linear regret bound for this algorithm in the general case. However, the problem with this algorithm is that it is computationally expensive and needs to iterate over all the time steps of the task before returning the policy. The SMILe algorithm trains a stationary but a stochastic policy. It thus overcomes the problem of iterating over all time steps before returning the policy and thus is applicable in a number of real-world applications. However, due to the stochasticity of the learnt policy the model is not stable [3]. The authors in [4] proposed the search-based structured prediction (SEARN) algorithm. The algorithm starts out by following the expert policy and collects demonstrations iteratively. It takes actions based on a mixture of previously trained policies to collect new episodes. It slowly reduces its dependence on the expert policy. The authors in [5] proposed the reduction based active imitation learning (RAIL) algorithm. It uses an independent identically distributed active learner. It learns a stationary policy which is applicable for all time steps of the learning task. It thus overcomes the limitations of forward training but it has limited practical applications due to the absence of the i.i.d. active learner in real world tasks [3]. The authors in [2] proposed the DAGGER algorithm which is another active imitation learning algorithm. It is closely related to no-regret online learning algorithm [6, 7, 8]. The advantage of DAGGER is that it can teach the agent to learn from previous mistakes [3]. However, it requires the storage of all the previous expert demonstrations and retrains the model on the entire aggregated set of demonstrations which makes it computationally expensive. In DAGGER, the policy that is being learnt may be far from the actual policy space, this limits learning ability and information might not be inferable from the state [3]. In order to overcome this issue the authors in [9], proposed the DAGGER by coaching algorithm. It implements easy to learn actions inferable from the state. However, it still suffers from the computational bottleneck of the

DAGGER algorithm. The authors in [10], proposed the generative adversarial imitation learning algorithm for learning from expert demonstrations. It overcomes the limitations of BC algorithm and uses inverse reinforcement learning (IRL) to extract the underlying cost function and runs reinforcement learning (RL) on the learnt cost function. The authors in [2] propose the DAGGER algorithm as a "no-regret online learning algorithm" for imitation learning. They analyze DAGGER as an online learning algorithm and show that regret of the DAGGER algorithm goes to zero as the number of iterations become very large. In this work we analyze IL from an stochastic optimization perspective.

Performative Prediction: Performative prediction in machine learning refers to the use of machine learning models to make predictions that directly impact the world or the data used to train the model. In other words, the predictions made by the model can affect the very system being modeled, creating a feedback loop that can lead to unintended consequences. When it is ignored, it results in distribution shifts which may result in an erroneous model. An example of performative prediction is using a machine learning model to predict stock prices. If the predictions made by the model are used to inform trading decisions, the act of trading itself can change the market dynamics and therefore impact future predictions made by the model.

The phenomenon of distribution shift can be either due to internal or external sources. The external changes include temporal or spatial factors and have been extensively studied in [11, 12, 13, 14]. In this thesis we consider performative prediction due to internal factors, where the choice of the model induces a change in the distribution. This setting was first introduced in [15]. With the recent rise in the use of ML systems, consideration of performative nature of machine learning models allows developers and practitioners to design and evaluate models that take into account the impact of their predictions on the real world. Recognizing the importance of performative prediction in machine learning can help ensure that these systems are well-designed.

The authors in [15] proposed the RRM and RGD to solve the performative prediction problem. However, these algorithms only converge to a performatively stable point and not the optimal point. In the follow-up work to this in [16], the authors propose the stochastic optimization setting and showed similar results for convergence to a performatively stable point. The authors in [17, 18] analyzed the performative prediction algorithms as biased stochastic optimization. They have shown that the bias in the gradient reduces with convergence. In [19], the authors consider the setting where the current distribution not only depends on the choice of the model but also the previous distributions. Each of these pieces of work identifies a stable point in terms of performance, and if specific assumptions

about the distribution are met, it can be demonstrated that this point is closely related to the optimal point. The authors in [20] propose the performative gradient descent algorithm to find the performatively optimal point by using an unbiased estimate of the gradient.

In this thesis, we assimilate the recent advances in the performative prediction field into IL framework. IL itself can be seen under the performative setting. Hence, we propose a stochastic optimization algorithm for finding the performatively optimal policy in IL.

Non-convex optimization: The underlying optimization in IL is non-convex, due to the dependence of the distribution on the optimization parameter [2]. Hence we look at some of the works proposed to solve the stochastic non-convex optimization problem. The authors in [21], showed that SGD with an appropriately selected learning rate can find an approximate stationary point at the rate of $O(1/T^{1/4})$. The method proposed in this paper, however, requires the knowledge of the smoothness parameter of the loss function and variance. Adaptive methods have been shown to achieve this rate in a parameter-free manner [22, 23, 24]. Thus there has been a wide use of adaptive methods like AdaGrad [25], Adam [26], and RMSProp [27] for non-convex optimization. Variance reduction techniques for non-convex optimization of finite sum problems was first proposed in [28, 29], having a rate of $O(1/T^{1/4})$. An improved rate of $O(1/T^{3/10})$ was obtained in [30]. The optimal rate of $O(1/T^{1/3})$ and was first obtained in [31, 32]. In [31], the authors obtained the optimal rate for the case of expectation over smooth losses, which captures the finite sum problem in itself. The authors in [33] proposed the STORM algorithm, which attains the optimal rate of $O(1/T^{1/3})$ without the need for maintaining critical points. However, this algorithm requires knowledge about the smoothness parameter of the loss in order to set the hyperparamters. In order to overcome this problem the authors in [34] proposed a fully adaptive version of the STORM algorithm and called it the STORM+ algorithm.

Since the underlying optimization in IL is non-convex, we explore several of these non-convex stochastic optimization techniques in order to solve the optimization problem. We derive an unbiased estimator of the gradient which we plug into several existing algorithms to get the final policy in IL.

Chapter 2

Imitation Learning Formulation and Background

2.1 Imitation Learning

The challenge of imitation learning (IL) is to teach a learner to behave like a human expert. The agent (learner) picks up the skill by seeing and imitating highly skilled experts. Humans and other animals, including fish and frogs, acquire new behaviours mostly through imitation. The fields of neuroscience and behavioural psychology serve as major sources of inspiration for IL. An example IL is autonomous driving shown in Fig.

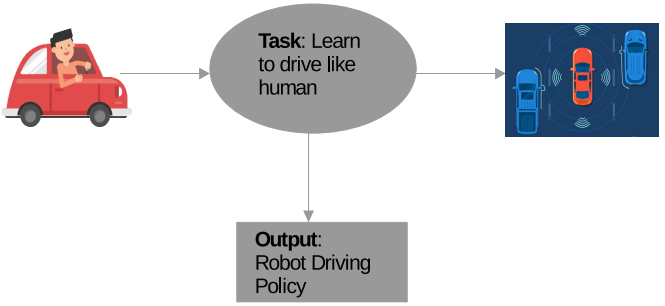


FIGURE 2.1: Imitating the Human Expert.

2.1. In this scenario, an autonomous vehicle learns to drive by observing the actions of a human driver. The human driver is considered the expert, and their actions serve as the training data for the autonomous vehicle.

During the training process, the autonomous vehicle observes the human driver's actions and learns to imitate them. For example while driving if the human driver spots another car approaching it, the human driver must take an action in order to preserve the safety of all the people involved. By observing this action by the human in this situation the autonomus driving car must learn to produce similar responses in similar situations. The vehicle then uses this learned behavior to drive itself in similar situations. Over time, the vehicle may continue to learn and improve its driving skills based on its own experiences on the road.

2.1.1 Methods to solve Imitation Learning

Approaches to solving the IL problem can be broadly classified into to sub-groups, active learning with expert and learning from expert demonstrations. Further learning from expert demonstrations is sub-divided into two categories. Some of the classification of these approaches is shown in Fig. 2.2.

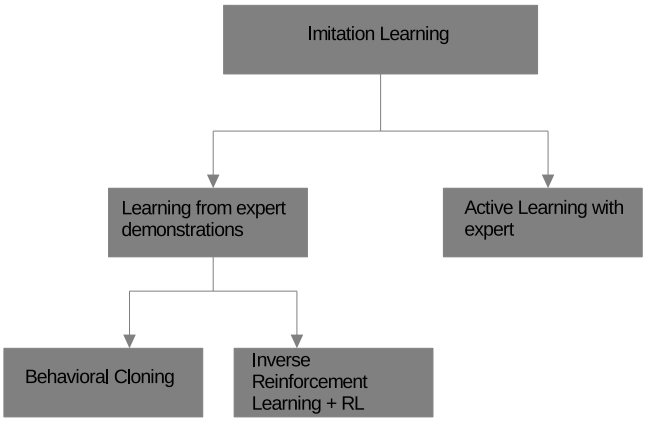


FIGURE 2.2: Methods for Imitation Learning.

Active Learning with an expert

In this method to solve IL, along with expert demonstrations, an active expert or multiple experts are present during training to help with learning. A generic active IL algorithm is shown in Fig. 2.3. The person shown may be another agent or a subject matter expert. Assistance can also be provided in many ways, such as by suggesting the best course of action in response to a specific observation or by suggesting an entirely new course of action. DAGGER [2] is the simplest and most often used approach to active IL since it is an iterative algorithm that learns a deterministic strategy. DAGGER relies

on demonstrations from experts to determine an initial policy. The agent then uses the first policy it has learned to gather roll-outs, or trajectories obtained when following the current policy across the environment.

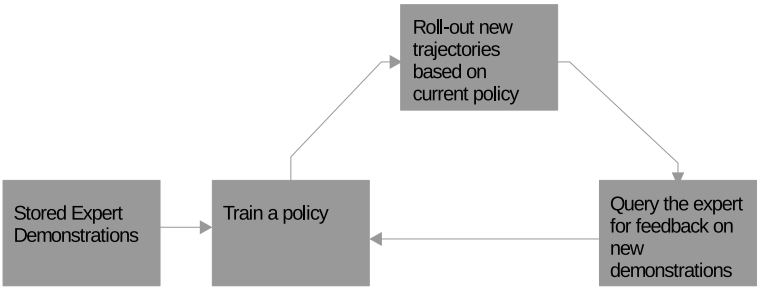


FIGURE 2.3: Active learning with an expert.

The use of active IL has its benefits and drawbacks. The safety of the algorithm is improved both during training and deployment by the presence of expert guidance. Having an expert on hand for long periods of time, however, can be prohibitively expensive and impractical. Thus reducing the computational cost of active imitation learning algorithms becomes an important task. This is the approach that would be followed in our work. We look at active imitation learning algorithms and how to reduce the computational burden of such algorithms.

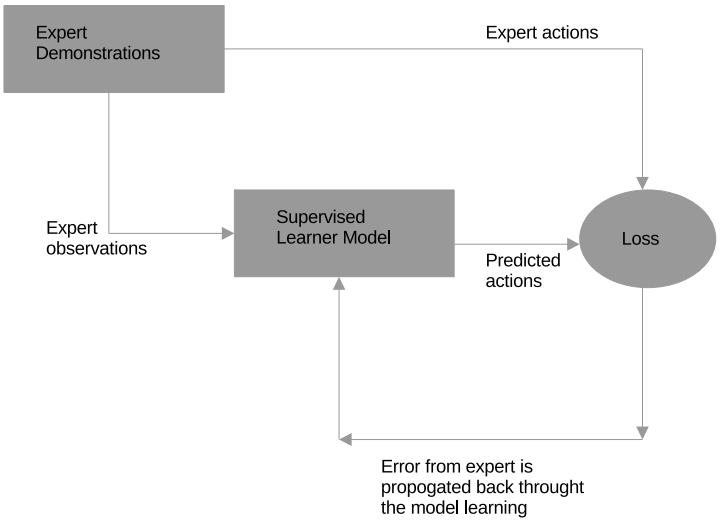
Imitation Learning from Expert Demonstrations

In this setting we are provided with certain number of expert demonstrations and we have to teach the agent a task based on these demonstrations only. There is no active expert guidance available and the expert cannot be queried for some new demonstrations.

(A) **Behavioural Cloning:** A supervised model is learned over (observation, action) pairings in behavioural cloning [1]. However, behavioural cloning is typically only effective when dealing with massive volumes of data because to the compounding mistakes produced by covariate shift [1, 2].

Figure 2.4 illustrates how a forward pass through a supervised learning model uses expert demonstrations to estimate actions. Model parameters are optimized by feeding back the difference between what was expected and what the experts actually did.

(B) **Imitation Learning Using Inverse Reinforcement Learning:** The lack of insight into which actions or behaviours are more costly to the agent and which are less costly is a



5

FIGURE 2.4: Behavioral Cloning.

major limitation of behavioural cloning. The cost of an action is the subjective evaluation of how well (or poorly) it turned out for the agent in light of the information provided by the observer. Learning and development of skills and behaviours rely heavily on cost functions. Both internal and external drives are necessary for success. That is the rationale for the cost function. Recovering the cost function that explains the demonstrations as (near) optimal behaviour is the goal of Inverse Reinforcement Learning (IRL) for imitation learning. Fig. 2.5 summarizes this approach.

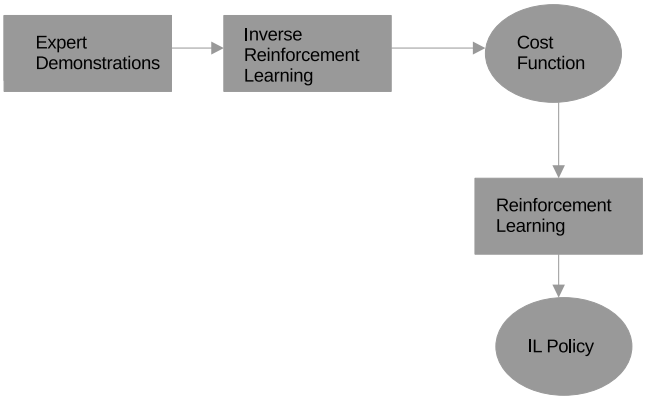


FIGURE 2.5: Imitation Learning using IRL.

2.2 Markov Decision Processes

We now look the description of Markov Decision Processes (MDPs). MDPs provide a mathematical framework for studying decision-making under uncertainty which can be used to model human behaviour in real-world settings. It is commonly used for modelling problems in RL and IL. MDPs are specified by $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{S}_\theta, \gamma, \mathcal{H} \rangle$. The components of a MDP are defined as follows:

1. $\mathcal{S} \subset \mathbb{R}^{D_s}$: This is the state space. The present state of the environment is from this space.
2. $\mathcal{A} \subset \mathbb{R}^{D_a}$: This is the action space. The agent is allowed to pick actions from this compact set.
3. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$: This is transition function that maps a triplet (s, a, s') of states and actions to a probability value in the range $[0, 1]$, representing the probability of transitioning from state s to state s' when taking action a in state s.
4. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$: This is the reward function for having taken an action in a state.
5. $\mathcal{S}_\theta$: This is the initial state distribution.
6. γ: This is the discount factor.
7. $\mathcal{H}$: This is the task horizon.

2.3 Terminology

Some important terminology that we use in this thesis is described below:

- **Policy:** It is a mapping from a state to a distribution over the actions. It is represented as $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. There are three different ways in which the policy representation can be categorized as described in [35]:
    1. Abstraction based on task: In this level of planning, a policy is trained to generate options denoted as o. Policies of action through time are represented by the o choices, which are mapped from the s_t state and c context. At this level of abstraction, each choice is a collection of possible courses of action.

$$\pi : s_t, c \rightarrow [o_1, o_2, \ldots, o_T]. \tag{2.1}$$

2. Abstraction based on trajectory: In this level of planning, a policy is trained to generate entire trajectories. Policies are mapped to trajectories from contexts c.

$$\pi : c \rightarrow \tau. \tag{2.2}$$

3. Abstraction based on action-state: This is a finer level of policy planning. At each time step a policy is trained to get actions a_t from states s_t and context c.

$$\pi : s_t, c \rightarrow a_t. \tag{2.3}$$

In this thesis we consider the action-state level abstraction of the policy. Further categorization of policy can also be done based on stationarity and stochasticity.

1. Based on stationarity: This categorization has two types of policies, stationary and non-stationary. A time-dependent policy is called a non-stationary policy. Since the policy changes depending on the current time step or trajectory phase, most trajectory-based policies are non-stationary. Stationary (or time invariant) policies are not affected by the passage of time, and instead use a single policy to decide the best course of action for each stage along a trajectory.
2. Based on stochasticity: This categorization has two types of policies, stochastic and deterministic. A policy which for a given initial state s_0 and context c produces the same trajectory no matter how many times we roll out the policy is called a deterministic policy. The generated deterministic trajectory is represented as follows:

$$\tau = \pi(s_0, c). \tag{2.4}$$

In action-state level categorization, a deterministic policy gives a fixed action a_t for each time step given a state s_t and context c,

$$a_t = \pi(s_t, c). \tag{2.5}$$

π represents the deterministic function of s_t and c. Deterministic policies do not consider any randomness in the action. The randomness is only due due the transition function. Thus using deterministic policy, distribution of trajectory

$\tau = [s_0, a_0, \ldots, s_T, a_T]$ is as follows:

$$p(\tau) = p(s_0) \prod_{t=1}^{T} p(s_{s+1}|s_t, a_t). \tag{2.6}$$

Here $a_t = \pi_t(s_t)$, which is a deterministic function.

A stochastic policy is a policy which selects an action at each time step according to a probability distribution. Thus given a state s_t and context c the sampled actions might be different depending on the distribution.

$$a_t \sim \pi(s_t, c). \tag{2.7}$$

Given the current state s_t and context c, π indicates a conditional distribution of the action a_t. In this setting in addition to the randomness due to the transition function we also have randomness due to the conditional distribution over the actions. Thus using stochastic policy the trajectory $\tau = [s_0, a_0, \ldots, s_T, a_T]$ distribution as follows:

$$p(\tau) = p(s_0) \prod_{t=1}^{T} p(s_{s+1}|s_t, a_t)\pi_t(a_t|s_t). \tag{2.8}$$

- **Expert Policy:** The implementation of imitation learning requires an expert which guides our agent in the learning task. We denote the policy of the expert as π^* and consider it to be a stationary and deterministic policy.

- **State Distributions:** We consider the following notations related to the distributions induced by a policy π:

  1. In state s, the policy π generates a probability distribution over different actions, which is denoted by π_s.
  2. If a policy π is followed from the initial time step, then d_π^i represents the distribution of states at time step i.
  3. If a policy π is followed from the beginning, then the state visitation frequency over T time steps is represented by d_π, which is obtained by averaging the state distributions d_π^i over T time steps, where i ranges from 1 to T i.e., $d_\pi = \frac{1}{T}\sum_{i=1}^{T} d_\pi^i$.
  4. The distribution of states that would be observed on average if an expert policy is followed throughout all time steps is denoted by d_{π^*}.

- **Cost Functions:** We consider the following cost functions:

  1. The cost of taking an action a in state s is represented by $C(s, a)$ and its value falls in $[0, 1]$. In IL, we aim to minimize the difference between the policy π that we are interested in and π^*, which represents the expert policy. This difference, also known as the loss, is defined as $e(s, a) = \mathbb{I}(a \neq \pi^*(s))$.
  2. The expected immediate cost of taking an action in state s according to policy π is denoted by $C_\pi(s) = \mathbb{E}_{a\sim\pi_s}[C(s, a)]$. In the context of imitation learning, we are interested in the expected loss when using policy π in state s, which is denoted by $e_\pi(s) = \mathbb{E}_{a\sim\pi_s}[e(s, a)]$.
  3. The expected cost of executing the policy π over T time steps is represented by $J(\pi) = T\mathbb{E}_{s\sim d_\pi}[C_\pi(s)]$.

2.4 Existing algorithms for IL

2.4.1 Problem Formulation

The IL objective is to get a policy $\hat{\pi}$ which minimizes the loss function with respect to expert actions, under the distribution of the states induced by rolling out the trajectory using the policy being learnt.

$$\hat{\pi} = \arg\min_{\pi\in\Pi} \mathbb{E}_{s\sim d_\pi}[C_\pi(s)] \tag{2.9}$$

Here d_π is the induced distribution of states obtained using the policy π being learnt. We look at various existing algorithms and how they tackle the problem of imitation learning.

2.4.2 Behavioral Cloning

This is one of the traditional methods used to solve imitation learning problems. It completely ignores the dependence of the current state on the previous predictions made and simply trains the agent on the states encountered by the expert policy. The objective function that this algorithm optimizes is as follows:

$$\hat{\pi}_{sup} = \arg\min_{\pi\in\Pi} \mathbb{E}_{s\sim d_{\pi^*}}[l(s, \pi)] \tag{2.10}$$

Here $l(s, \pi)$ represents the loss function of taking an action which is different expert action. We can see that this algorithm only optimizes under states induced by the expert policy and hence performs badly in cases where states outside the known distribution are encountered. It has been shown in literature that the loss for this algorithm grows quadratically in event horizon T. The event horizon is the duration of one task which has been assigned to the agent.

Intuitively, the reason that the loss function is quadratic in T is that as soon as the algorithm makes a mistake in any one state, it continues to encounter states outside the distribution which was induced by the expert policy and hence the mistake gets compounded. This can be seen as the problem of out of distribution testing which is encountered in alot of machine learning problems, where the distribution on which the agent is tested is different from the one on which it was trained.

2.4.3 Forward Training Algorithm

The basic idea behind Forward Training algorithm is to change the policies slowly so that the learner is given enough time to recover from mistakes. This trains a deterministic but non-stationary policy. We define a new policy for each time step. One limitation of the forward algorithm is that it becomes infeasible to use when the number of time steps T is very large or unknown, since it requires training T distinct policies in sequence and it is not possible to terminate the algorithm before completing all T iterations. This can lead to scalability issues and increased computational complexity, which can make the algorithm impractical for large-scale problems.

2.4.4 Stochastic Mixing Iterative Learning

The problem with behavioral cloning is that it fails to recover from the mistakes it has made because of the out of distribution states it encounters after making a mistake. The SMILe algorithm aims to solve this problem by training a stationary stochastic policy over several iterations. At the start of each iteration a new policy is obtained by stochastically mixing the current policy with the expert policy. This mixing is done in such a way so as to ensure that the contribution of the expert policy reduces, as the number of iterations increase and for $N \to \infty$, the expert policy is completely removed from the final policy learned.

Algorithm 1 SMILe Algorithm

Initialize $\pi^0 \rightarrow \pi^*$
while $i \leq N$ **do**
  Sample a T step trajectory using the current policy
  Get the dataset $D = (s, \pi^*(s))$ using the expert policy
  Train a classifier on the data collected $\pi^{*i} = argmin_{\pi \in \Pi} E_{s \sim D}[e(s, \pi)]$
  Stochastic Mixing as done in [1]: $\pi^i = (1-\alpha)^i \pi^* + \alpha \sum_{j=1}^{i} (1-\alpha)^{j-1} \pi^{*j}$
  Train a policy on the entire dataset D
end while
Remove the expert policy completely from the trained policy.
Return the final trained policy π

2.4.5 Dataset Aggregation

This is an iterative procedure to train a deterministic policy for the imitation learning problems. It provides us with good convergence guarantees under the distribution of states it induces.

The basic idea for this algorithm is to make it robust to out of distribution states which might be encountered by the agent. It starts by training a classifier on the states and actions of the expert policy. Then this trained policy is used to gather a new set of states that might be encountered. Then the expert is queried on the states regarding the actions that it would on these newly encountered states and they are aggregated together with the entire dataset.Then a new policy is trained on the entire dataset. These steps are repeated until we converge to a deterministic policy. Hence over many iterations we train our agent over states that it is likely to encounter during execution hence making it less likely that out of distribution states are encountered by the agent.

Algorithm 2 DAGGER

Ensure: $D \rightarrow \phi$
$\pi \rightarrow \pi^*$
while $i \leq N$ **do**
  Sample a T step trajectory using the current policy
  Get the dataset $D_i = (s, \pi^*(s))$ using the expert policy
  Aggregation step : $D \leftarrow D \cup D_i$
  Train a policy on the entire dataset D
end while
Return the final trained policy π

The DAGGER algorithm operates by collecting a dataset at each iteration using the current policy, and then training the next policy using the combined datasets from all

previous iterations. This approach yields a linear upper bound on the loss function for a fixed number of time steps T. This is achieved by training the policy on a dataset that includes states that are not only generated by the expert policy but also those outside the distribution of states induced by the expert policy. By including such states, the policy is forced to generalize to a broader range of situations, leading to improved performance over the fixed number of time steps.

2.5 Performative Prediction

When the predictions made by our algorithm influences the output in some way, it is called performative prediction. It is a very common phenomenon in many machine learning problems. Performative prediction seeks to actively engage with the world and change it based on its predictions. This can involve making decisions, taking actions, or providing feedback in a way that influences the system being predicted. For example, when a bank uses a model to predict whether a person will default on credit, they might charge higher rates from people at risk of defaulting their credits and thus influence the customer's default risk even further.

Performative prediction has two main sources of influence which change the output. These may be extrinsic or intrinsic. The intrinsic phenomenon happens when the model parameters themselves influence the predictions. IL can been seen as a performative prediction problem due to the intrinsic model parameters because as seen in the objective of (2.9), the output actually depends on the model parameters. Thus we look at the IL problem as performative prediction and try to assimilate the advances made in the performative prediction field into IL.

In order to study the performative prediction problem we first introduce the performative risk as done in [15]:

$$PR(\theta) = \mathbb{E}_{\tau \sim \mathcal{D}(\theta)}[l(\theta, \tau)]. \tag{2.11}$$

Since in the performative risk, the distribution depends on θ, optimizing $PR(\theta)$ is quite challenging. There have been a lot of algorithms introduced which try to minimize the performative risk and these methods have been mentioned in the literature review section. Here, we explore the two difference kinds of solutions possible in the performative prediction framework:

1. Performatively stable (θ_{PS}): An optimizer of $PR(\theta)$ is called performatively stable, if it minimizes the loss under the distribution it induces. Thus it satisfies the following equation,

$$\theta_{PS} = \arg\min_{\theta} \mathbb{E}_{\tau \sim \mathcal{D}(\theta_{PS})}[l(\theta, \tau)]. \tag{2.12}$$

2. Performatively optimal (θ_{PO}): An optimizer of $PR(\theta)$ is called performatively optimal if it satisfies the following:

$$\theta_{PO} = \arg\min_{\theta} \mathbb{E}_{\tau \sim \mathcal{D}(\theta)}[l(\theta, \tau)]. \tag{2.13}$$

The performative optimal and stable solutions are two different concepts of solution. It is not necessary that performatively optimal points are performatively stable or vice versa [15]. However, the authors in [15] have shown that when the underlying distribution $\mathcal{D}$ is ε-sensitive and certain assumptions hold, then θ_{PO} and θ_{PO} are close.

In IL, since the distribution of the trajectories is influenced by the parameter we optimize, it can been seen under the framework of performative prediction. In this thesis, we show that IL algorithm DAGGER is similar to the RRM algorithm proposed in [15] and thus returns a performatively stable point. We also propose algorithms that return performatively optimal point using the gradients of the true IL objective.

2.6 Relevant Optimization Algorithms

In this thesis we look at IL from the perspective of optimization and how to make current active IL algorithms more computationally efficient. Hence, we need to understand the underlying optimization and the relevant algorithms that can be used to solve it. We discuss some of the optimization algorithms such as SGD, Adam [26], RMSProp [27] and PPO [36] algorithms which we have referenced to in this work.

2.6.1 Stochastic Gradient Descent

Gradient Descent is a widely used optimization algorithm for solving the optimization problem $\min_{\theta \in \mathbb{R}^d} f(\theta)$, where $f : \mathbb{R}^d \to \mathbb{R}$ is a function that is twice continuously differentiable. In the stochastic variant of this algorithm called Stochastic Gradient Descent (SGD), the algorithm is applied to a training set and iteratively moves towards the optimal

solution by computing the loss and gradient on a randomly selected subset of the data, rather than on the entire dataset.

When $f(\theta)$ has a finite sum form as expressed like $f(\theta) = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta)$, where f_i loss corresponding to the ith point in the dataset then SGD can be applied. SGD approximates the gradient as $\nabla f(\theta_k) \approx \nabla f_i(\theta_k)$. Thus instead of calculating the actual gradient on the entire dataset, we use the stochastic gradient for obtaining the update direction. This helps in reducing the computational burden of the traditional gradient descent algorithm. However, the variance of the stochastic gradient might hinder the convergence and hence the convergence is not as smooth as the gradient descent algorithm.

2.6.2 Adaptive Moment Estimation (Adam)

Adam [26] was developed as an extension to SGD. It uses adaptive learning rates for each individual parameter based on the previous gradients. In Adam implementation the first and the second order moments of the past gradients are stored in the parameters m_t and v_t. The present gradient is mixed with the past gradients to get the final update direction and learning rate for each parameters. Adam seeks error minima that are as flat as possible.

2.6.3 RMSProp

RMSProp (Root Mean Square Propagation) [27] is an optimization algorithm used in machine learning to improve the training of neural networks. It is a gradient descent-based algorithm that adapts the learning rate of each weight parameter in the network during training, to speed up convergence and avoid oscillations.

The algorithm maintains an exponentially decaying average of the squared gradient for each weight parameter. This average is used to normalize the gradient before updating the parameters. This normalization helps to prevent the learning rate from becoming too large or too small, which can lead to slow convergence or overshooting the minimum. The RMSProp algorithm stores the second moments of the gradients as follows:

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2. \tag{2.14}$$

Then it updates the parameters as follows:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{v_t}} g_t. \tag{2.15}$$

2.6.4 Proximal Policy Optmization (PPO)

Proximal Policy Optimization (PPO) [36] is a reinforcement learning algorithm that is used to train an agent to take actions in an environment in order to maximize a reward signal. It is a policy optimization method that is used to update the policy parameters of an agent in an iterative manner.

The PPO algorithm can be mathematically summarized as follows:

1. Start with an initial policy parameter θ_0.
2. Collect a set of trajectories using the current policy π_{θ_k}, where k is the current iteration.
3. Compute the advantages A_t for each time step t in the trajectories.
4. Compute the surrogate objective function $L(\theta_k, \theta)$ using the clipped surrogate objective:
   $L(\theta_k, \theta) = \mathbb{E}\left[\min\left(\frac{r_t(\theta)}{r_t(\theta_k)} A_t, \text{clip}\left(\frac{r_t(\theta)}{r_t(\theta_k)}, 1-\epsilon, 1+\epsilon\right) A_t\right)\right]$,
   where $r_t(\theta)$ is the probability of taking the action a_t in state s_t under the policy π_θ, and ϵ is a hyperparameter that controls the extent of clipping.
5. Update the policy parameter θ_{k+1} by maximizing the surrogate objective $L(\theta_k, \theta)$ using a stochastic gradient ascent method.
6. Repeat steps 2-5 for a fixed number of iterations or until convergence is achieved.

In summary, the PPO algorithm aims to improve the policy of an agent by iteratively updating its policy parameters in a way that maximizes a clipped surrogate objective function. The use of a clipped surrogate objective helps to prevent the policy from changing too much from one iteration to the next, which can improve the stability of the learning process. We will be using the PPO algorithm as an expert policy to guide our agent.

2.6.5 Stochastic Recursive Momentum (STORM)

Stochastic Recursive Momentum (STORM) [33] is a stochastic optimization algorithm that is used to minimize a non-convex objective function in machine learning. It is an extension of the classic stochastic optimization algorithm found in most of the problems in machine

learning which require to achieve convergence of a non-convex function to its critical points. The STORM algorithm is designed to converge faster than the classical momentum method and to be less sensitive to hyperparameters. It is a variance reduced algorithm for SGD. It uses the momentum term in order to achieve variance reduction. It uses two gradient computations in each step. This algorithm does not involve computing batch gradients and instead uses adaptive learning rates, which makes it easier to implement and requires fewer hyperparameters to be tuned.

Chapter 3

SGD for Imitation Learning

SGD is a machine learning optimization technique employed to train models. It is derived from gradient descent but differs in how it computes the gradient of the loss function. Rather than calculating the gradient over the complete training set, SGD computes it over a randomly selected subset. In each iteration, the model parameters are updated based on the gradient of the loss function with respect to the parameters. The use of random samples in SGD offers the benefit of reducing the computational burden associated with gradient computation, particularly when dealing with extensive datasets. One of the main benefits of SGD is that it can converge faster than batch gradient descent, especially when the loss function has many local minima. This is because the stochastic nature of the algorithm allows it to jump out of local minima more easily than batch gradient descent. However, the trade-off is that the convergence is not as smooth as batch gradient descent. Overall, stochastic gradient descent is a widely used and effective optimization algorithm for training machine learning models, especially in cases where the data is very large and computationally expensive to process. One of the main problems with the active IL framework, [1, 2], is the computational cost of maintaining an active expert during training. They require alot of calls to the expert in order to converge to a good policy. In order to address the complexity issues of the current active IL algorithms, we propose to use the SGD algorithm in order to relieve the computational burden of the current algorithms without compromising on the accuracy of the final policy obtained. To utilize the SGD algorithm, it is necessary to have an unbiased estimation of the gradient of the IL objective. We derive this in the next section.

3.1 Stochastic Gradient Oracle

In this section we take at closer look at the IL objective and the various optimization algorithms that can be used to optimize the objective function.

3.1.1 Objective

The IL objective is defined as follows:

$$\begin{aligned} \theta^* &= \arg\min_{\theta} \mathbb{E}_{\tau \sim p(\tau;\theta)} \left[\sum_{t=1}^{T-1} l_t(s_t, \pi_\theta) \right] \\ &= \arg\min_{\theta} F(\theta). \end{aligned} \tag{3.1}$$

Here $\tau = (s_1, a_1, \ldots, a_{T-1}, s_T)$, represents the trajectory sampled according to the distribution $p(\tau;\theta)$. First we calculate the loss over the entire trajectory and take the expectation of this loss over the distribution of trajectories which in turn depends on the optimizing parameter θ. θ is the parameters of the policy represented as π_θ. Here $l_t(s_t, \pi_\theta)$ represents the loss of executing the policy π_θ in the state s_t at time step t of the trajectory. The loss function is defined such that it is able to penalize the actions which are far from the actions of the expert policy. We have used the following loss function for the simulations:

$$l_t(s_t, \pi_\theta) = (\pi_\theta(s_t) - \pi^*(s_t))^2. \tag{3.2}$$

Here π^* represents the expert policy.

3.1.2 Derivation of the Stochastic Gradient

In order to apply the SGD algorithm for IL, we need a stochastic gradient for the objective defined in (3.1). We derive this stochastic gradient in this section. For deriving the stochastic gradient we assume that the different trajectories are independent and the action in the present states do not depend on the state or action in future steps. Both these assumptions are quite practical, Expanding the objective of (3.1) and noting that the states at the current time do not depend on the future actions or states, we have the

following:

$$\mathbb{E}_{\tau\sim p(\tau;\theta)}\left[\sum_{t=1}^{T-1} l_t(s_t,\pi_\theta)\right] \tag{3.3}$$

$$= \int \left(\sum_{t=1}^{T-1} l_t(s_t,\pi_\theta)\right)\mu(s_1)\prod_{t=1}^{T-1}\pi_\theta(a_t|s_t)\mu(s_{t+1}|s_t,a_t)ds_1\prod_{t=1}^{T-1} d(s_{t+1}|s_t,a_t)d(a_t|s_t) \tag{3.4}$$

$$= \sum_{t=1}^{T-1}\int l_t(s_t,\pi_\theta)\mu(s_1)\prod_{i=1}^{t-1}\pi_\theta(a_i|s_i)\mu(s_{i+1}|s_i,a_i)ds_1\prod_{i=1}^{t-1} d(s_{i+1}|s_i,a_i)d(a_i|s_i) \tag{3.5}$$

In order to obtain the expression for the gradient we take derivative of the expanded objective with respect to the optimization parameter θ. In the subsequent expressions we represent the derivative with respect to θ as ∇.

$$\nabla\mathbb{E}_{\tau\sim p(\tau;\theta)}\left[\sum_{t=1}^{T-1} l_t(s_t,\pi_\theta)\right] \tag{3.6}$$

$$= \nabla\sum_{t=1}^{T-1}\int l_t(s_t,\pi_\theta)\mu(s_1)\prod_{i=1}^{t-1}\pi_\theta(a_i|s_i)\mu(s_{i+1}|s_i,a_i)ds_1\prod_{i=1}^{t-1} d(s_{i+1}|s_i,a_i)d(a_i|s_i) \tag{3.7}$$

$$= \sum_{t=1}^{T-1}\nabla\int l_t(s_t,\pi_\theta)\mu(s_1)\prod_{i=1}^{t-1}\pi_\theta(a_i|s_i)\mu(s_{i+1}|s_i,a_i)ds_1\prod_{i=1}^{t-1} d(s_{i+1}|s_i,a_i)d(a_i|s_i) \tag{3.8}$$

At this point if all the probability density functions (pdfs) are continuous, then differentiation and integration can be exchanged, yielding:

$$\nabla\mathbb{E}_{\tau\sim p(\tau;\theta)}\left[\sum_{t=1}^{T-1} l_t(s_t,\pi_\theta)\right] \tag{3.9}$$

$$= \sum_{t=1}^{T-1}\int (\nabla l_t(s_t,\pi_\theta))\mu(s_1)\prod_{i=1}^{t-1}\pi_\theta(a_i|s_i)\mu(s_{i+1}|s_i,a_i)ds_1\prod_{i=1}^{t-1} d(s_{i+1}|s_i,a_i)d(a_i|s_i) \tag{3.10}$$

$$+ \sum_{t=1}^{T-1}\int l_t(s_t,\pi_\theta)\mu(s_1)\nabla\left(\prod_{i=1}^{t-1}\pi_\theta(a_i|s_i)\right)\mu(s_{i+1}|s_i,a_i)ds_1\prod_{i=1}^{t-1} d(s_{i+1}|s_i,a_i)d(a_i|s_i)$$

In the second term of the summation, we can use the identity $\nabla p(\theta) = p(\theta)\nabla \log(p(\theta))$, so as to obtain:

$$\nabla \mathbb{E}_{\tau \sim p(\tau;\theta)} \left[\sum_{t=1}^{T-1} l_t(s_t, \pi_\theta) \right] \tag{3.11}$$

$$= \sum_{t=1}^{T-1} \int (\nabla l_t(s_t, \pi_\theta)) \mu(s_1) \prod_{i=1}^{t-1} \pi_\theta(a_i|s_i) \mu(s_{i+1}|s_i, a_i) ds_1 \prod_{i=1}^{t-1} d(s_{i+1}|s_i, a_i) d(a_i|s_i) \tag{3.12}$$

$$+ \sum_{t=1}^{T-1} \int l_t(s_t, \pi_\theta) \nabla \left(\sum_{i=1}^{t-1} \log \pi_\theta(a_i|s_i) \right) \mu(s_1) \prod_{i=1}^{t-1} \pi_\theta(a_i|s_i) \mu(s_{i+1}|s_i, a_i) ds_1 \prod_{i=1}^{t-1} d(s_{i+1}|s_i, a_i) d(a_i|s_i)$$

$$= \sum_{t=1}^{T-1} \int (\nabla l_t(s_t, \pi_\theta)) \mu(s_1) \prod_{i=1}^{t-1} \pi_\theta(a_i|s_i) \mu(s_{i+1}|s_i, a_i) ds_1 \prod_{i=1}^{t-1} d(s_{i+1}|s_i, a_i) d(a_i|s_i) \tag{3.13}$$

$$+ \sum_{t=1}^{T-1} \int l_t(s_t, \pi_\theta) \left(\sum_{i=1}^{t-1} \frac{\nabla \pi_\theta(a_i|s_i)}{\pi_\theta(a_i|s_i)} \right) \mu(s_1) \prod_{i=1}^{t-1} \pi_\theta(a_i|s_i) \mu(s_{i+1}|s_i, a_i) ds_1 \prod_{i=1}^{t-1} d(s_{i+1}|s_i, a_i) d(a_i|s_i).$$

The above expression can be compactly written by substituting back $p(\tau;\theta)$ as:

$$\nabla \mathbb{E}_{\tau \sim p(\tau;\theta)} \left[\sum_{t=1}^{T-1} l_t(s_t, \pi_\theta) \right] = \mathbb{E}_{\tau \sim p(\tau;\theta)} \sum_{t=1}^{T-1} \left[\nabla l_t(s_t, \pi_\theta) + l_t(s_t, \pi_\theta) \sum_{i=1}^{t-1} \frac{\nabla \pi_\theta(a_i|s_i)}{\pi_\theta(a_i|s_i)} \right] \tag{3.14}$$

The unbiased estimator of the gradient of the IL objective is given by :

$$\sum_{t=1}^{T-1} \left[\nabla l_t(s_t, \pi_\theta) + l_t(s_t, \pi_\theta) \sum_{i=1}^{t-1} \frac{\nabla \pi_\theta(a_i|s_i)}{\pi_\theta(a_i|s_i)} \right] \tag{3.15}$$

Then, for a given value of θ, we roll out the trajectory τ. In order to calculate the stochastic gradient, at each state s_t, we need:

1. The loss function and its gradient with respect to θ, i.e., $l_t(s_t, \pi_\theta)$ and $\nabla l_t(s_t, \pi_\theta)$.
2. The action distribution according to the current policy and its derivative with respect to θ, i.e., $\pi_\theta(a_t|s_t)$ in closed for as a function of θ and $\nabla \pi_\theta(a_t|s_t)$.

The first must be obtained from the expert feedback since the loss is always calculated with respect to the expert trajectory, while second follows from the model used. In the case, when we use a neural network parameterized by θ as the policy various automatic differentiation packages such as PyTorch [37] and TensorFlow [38] can be used to obtain the second term. The second term is very similar to the objective maximized in the REINFORCE algorithm of RL. In REINFORCE instead of scaling by the loss, we scale

by the returns of a trajectory. Hence the second term can be obtained by using the REINFORCE trick [20]. The first term of the gradient in (3.16) provides the expert feedback which helps the agent in learning while the second term accounts for the shift in the distribution [20].

For the imitation learning objective we have the following:

$$\nabla \mathbb{E}_{\tau \sim p(\tau;\theta)} \left[\sum_{t=1}^{T-1} l_t(s_t, \pi_\theta) \right] = \mathbb{E}_{\tau \sim p(\tau;\theta)} \sum_{t=1}^{T-1} \left[\nabla l_t(s_t, \pi_\theta) + l_t(s_t, \pi_\theta) \sum_{i=1}^{t-1} \nabla \ln \pi_\theta(a_i|s_i) \right], \tag{3.16}$$

$$\nabla F(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} \sum_{t=1}^{T-1} \left[\nabla l_t(s_t, \pi_\theta) + l_t(s_t, \pi_\theta) \sum_{i=1}^{t-1} \nabla \ln \pi_\theta(a_i|s_i) \right],$$

$$\nabla F(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[\nabla f(\theta, \tau)].$$

f represents the loss corresponding to the stochastic gradient and F represents the IL objective. The final update equation is as follows:

$$\theta_{k+1} = \theta_k - \eta_k \nabla f(\theta_k, \tau_k) \tag{3.17}$$

Here k is the index for the iterations, and $\nabla f(\theta_k, \tau_k)$ is the stochastic gradient at the kth iteration. We have the following unbiased property:

$$\mathbb{E}_{\tau_k \sim p(\tau;\theta_k)}[\nabla f(\theta_k, \tau_k)] = \nabla F(\theta_k) \tag{3.18}$$

3.2 SGD Algorithm

In this section we propose the SGD algorithm for IL. We summarize the SGD algorithm

Algorithm 3 SGD Based IL

Initialize θ_1 and $k = 1$.
while $k \leq K$ **do**
  $\tau_k \sim p(\tau; \theta_k)$.
  $\nabla f(\theta_k, \tau_k) = \text{Oracle}(\theta_k, \tau_k)$.
  Update θ as $\theta_{k+1} = \theta_k - \eta_k * \nabla f(\theta_k, \tau_k)$.
  $k = k + 1$.
end while
Return θ_K

for IL in Algorithm 3. At first we randomly initialize the policy parameter as θ_1. We

next collect a trajectory using the current value of the parameter. Next we compute the stochastic gradient using (3.16). Then we update the parameters based on the stochastic gradient. We continue this process until convergence.

3.2.1 Guarantees for the SGD Algorithm

In this section we derive some theoretical guarantees corresponding to the SGD for IL algorithm. We have the following assumptions for the proposed SGD for IL algorithm:

1. We assume that F is L smooth. This means that $\exists L$, such that:

$$||\nabla F(\theta_1) - \nabla F(\theta_2)|| \leq L||\theta_1 - \theta_2||. \tag{3.19}$$

2. Bounded Variance Assumption:

$$\mathbb{E}_{\tau \sim p(\tau;\theta)}[||\nabla f(\theta, \tau)||^2] \leq \sigma^2 + c||\nabla F(\theta)||^2. \tag{3.20}$$

From the L-smoothness of the objective $F(\theta)$, we can write the quadratic upper bound as follows:

$$\begin{aligned} F(\theta_{k+1}) &\leq F(\theta_k) + \langle \nabla F(\theta_k), \theta_{k+1} - \theta_k \rangle + \frac{L}{2}||\theta_{k+1} - \theta_k||^2 \\ &= F(\theta_k) - \eta_k \langle \nabla F(\theta_k), \nabla f(\theta_k, \tau_k) \rangle + \frac{\eta_k^2 L}{2}||\nabla f(\theta_k, \tau_k)||^2 \end{aligned} \tag{3.21}$$

Now taking the conditional expectation with respect to the iterate τ_k given the iterates $\tau_1, \ldots, \tau_{k-1}$. Let us denote this conditional expectation as $\mathbb{E}_{\tau_k|}$.

$$\begin{aligned} \mathbb{E}_{\tau_k|}[F(\theta_{k+1})] &\leq \mathbb{E}_{\tau_k|}[F(\theta_k)] - \eta_k \langle \mathbb{E}_{\tau_k|}[\nabla F(\theta_k)], \mathbb{E}_{\tau_k|}[\nabla f(\theta_k, \tau_k)] \rangle + \frac{\eta_k^2 L}{2}\mathbb{E}_{\tau_k|}[||\nabla f(\theta_k, \tau_k)||^2] \\ &\leq F(\theta_k) - \eta_k \langle \nabla F(\theta_k), \mathbb{E}_{\tau_k|}[\nabla f(\theta_k, \tau_k)] \rangle + \frac{\eta_k^2 L}{2}\mathbb{E}_{\tau_k|}[||\nabla f(\theta_k, \tau_k)||^2] \\ &\leq F(\theta_k) - \eta_k \langle \nabla F(\theta_k), \mathbb{E}_{\tau_k \sim p(\tau;\theta_k)}[\nabla f(\theta_k, \tau_k)] \rangle + \frac{\eta_k^2 L}{2}\mathbb{E}_{\tau_k \sim p(\tau;\theta_k)}[||\nabla f(\theta_k, \tau_k)||^2] \\ &\leq F(\theta_k) - \eta_k ||\nabla F(\theta_k)||^2 + \frac{\eta_k^2 L \sigma^2}{2} + \frac{\eta_k^2 L c}{2}||\nabla F(\theta_k)||^2 \\ &= F(\theta_k) - \eta_k (1 - \frac{\eta_k L c}{2})||\nabla F(\theta_k)||^2 + \frac{\eta_k^2 L \sigma^2}{2} \end{aligned}$$

The second inequality holds because given the previous iterates $\tau_1, \ldots, \tau_{k-1}$, θ_k is a deterministic quantity and can be taken out of the conditional expectation. The third inequality holds because $\mathbb{E}_{\tau_k|} = \mathbb{E}_{\tau_k \sim p(\tau;\theta_k)}$ as shown below,

$$\begin{aligned} p(\tau_k|\tau_1, \ldots, \tau_{k-1}) &= \int p(\tau_k, \theta_k|\tau_1, \ldots, \tau_{k-1}) d\theta_k \\ &= \int p(\tau_k|\theta_k, \tau_1, \ldots, \tau_{k-1}) p(\theta_k|\tau_1, \ldots, \tau_{k-1}) d\theta_k \\ &= \int p(\tau_k; \theta_k)\delta(\theta_k) d\theta_k = p(\tau_k; \theta_k). \end{aligned} \tag{3.22}$$

Now taking the expectation with respect to all the iterates instead of just τ_k and using $\eta_k < \frac{1}{Lc}$, we have the following:

$$\mathbb{E}[F(\theta_{k+1})] \leq \mathbb{E}[F(\theta_k)] - \frac{\eta_k}{2}\mathbb{E}[||\nabla F(\theta_k)||^2] + \frac{\eta_k^2 L\sigma^2}{2} \tag{3.23}$$

Carrying out the telescopic sum we have:

$$\mathbb{E}[F(\theta_{K+1})] \leq F(\theta_1) - \sum_{k=1}^{K} \frac{\eta_k}{2}\mathbb{E}[||\nabla F(\theta_k)||^2] + \sum_{k=1}^{K} \frac{\eta_k^2 L\sigma^2}{2} \tag{3.24}$$

We have $F(\theta^*) \leq \mathbb{E}[F(\theta_{K+1})])$, thus we can write,

$$\begin{aligned} \sum_{k=1}^{K} \eta_k \mathbb{E}[||\nabla F(\theta_k)||^2] &\leq 2(F(\theta_1) - F(\theta^*)) + \sigma^2 L \sum_{k=1}^{K} \eta_k^2 \\ &\leq 2B_F + \sigma^2 L \sum_{k=1}^{K} \eta_k^2. \end{aligned} \tag{3.25}$$

Here, $B_F = F(\theta_1 - F(\theta^*))$. Thus,

$$\min_{1 \leq k \leq K} \mathbb{E}[||\nabla F(\theta_k)||^2] \leq \frac{2B_F + \sigma^2 L \sum_{k=1}^{K} \eta_k^2}{\sum_{k=1}^{K} \eta_k}. \tag{3.26}$$

For $\eta_k = \eta$, we have,

$$\min_{1 \leq k \leq K} \mathbb{E}[||\nabla F(\theta_k)||] \leq \sqrt{\sigma^2 L\eta + \frac{2B_F}{\eta K}} = \epsilon \tag{3.27}$$

We have,

$$\eta = \frac{\epsilon^2}{2\sigma^2 L} \text{ and } K = \frac{8\sigma^2 L B_F}{\epsilon^4}. \tag{3.28}$$

The above analysis provides us with a bound on the gradient norm for the iterates of the SGD for IL algorithm. A bound on the gradient norm can be converted into a bound on the distance between θ_k and θ_{PO} with mild assumptions. Here θ_{PO} represents the performatively optimal point for the IL objective. If the objective function F is α-strongly convex, then we have the result $||\theta_k - \theta_{PO}|| \leq \alpha^{-1}|| \leq \mathbb{E}[||\nabla F(\theta_k)||]$ [20]. Thus with mild assumptions we can show that the proposed algorithm converges to a performatively optimal point.

3.2.2 DAGGER converges to Stable Point

In the previous section, we observed that the suggested SGD for IL reaches a point of optimal performance under certain reasonable assumptions regarding the objective. In this section, we provide a proof demonstrating that the widely recognized DAGGER algorithm, introduced in [2], converges to a point of stability in terms of performance. The RRM algorithm proposed in [15] has the following update:

$$\theta_{k+1} = G(\theta_k) \overset{def}{=} \arg\min_{\theta} \mathbb{E}_{\tau \sim p(\tau;\theta_k)} \left[\sum_{t=1}^{T} l_t(s_t, \pi_\theta) \right]. \tag{3.29}$$

We can say the the DAGGER algorithm of [2], is an aggregated version of the RRM algorithm. We can write the update of the DAGGER as follows:

$$\theta_{k+1} = G_{DA}(\theta_k) \overset{def}{=} \arg\min_{\theta} \frac{1}{k} \sum_{i=1}^{k} \mathbb{E}_{\tau \sim p(\tau;\theta_i)} \left[\sum_{t=1}^{T} l_t(s_t, \pi_\theta) \right]. \tag{3.30}$$

Theorem 3.1. *Let us assume that the loss function $l_t(s_t, \pi_\theta)$ is β-smooth and γ-strongly convex. If the distribution $p(\tau;\theta)$ is ε-sensitive, then we have the following:*

1. $||G_{DA}(\theta) - G_{DA}(\theta')|| \leq \frac{\varepsilon\beta}{\gamma k} \sum_{i=1}^{k} ||\theta_i - \theta_i'||$ *for a sequence of parameters* $\{\theta_i\}_{i=1}^{t}, \theta$ *and* $\{\theta_i'\}_{i=1}^{t}, \theta'$ *in the parameter space.*

Before proceeding to the proof, we define the following lemma:

Lemma 3.2. *A distribution $\mathcal{D}(.)$ is ε-sensitive if and only if the following is satisfied $\forall \theta, \theta'$ in the parameters space:*

$$\sup \left\{ \mathbb{E}_{Z\sim\mathcal{D}(\theta)}[g(Z)] - \mathbb{E}_{Z\sim\mathcal{D}(\theta')}[g(Z)] \leq \varepsilon ||\theta - \theta'|| \right\} \tag{3.31}$$

Proof. Let us fix two sequence of iterates for the DAGGER algorithm and denote them as $\{\theta_i\}_{i=1}^k$ and $\{\theta'_i\}_{i=1}^k$. Let $f(\varphi) = \frac{1}{k}\sum_{i=1}^k \mathbb{E}_{\tau\sim p(\tau;\theta_i)}\left[\sum_{t=1}^T l_t(s_t, \pi_\varphi)\right]$ and $f'(\varphi) = \frac{1}{k}\sum_{i=1}^k \mathbb{E}_{\tau\sim p(\tau;\theta'_i)}\left[\sum_{t=1}^T l_t(s_t, \pi_\varphi)\right]$. Let us assume that f is γ-strongly convex and $G_{DA}(\theta)$ is the unique minimizer for $f(x)$, thus from the strong convexity assumption we have,

$$f(G_{DA}(\theta)) - f(G_{DA}(\theta')) \geq (G_{DA}(\theta) - G_{DA}(\theta'))^T \nabla f(G_{DA}(\theta')) + \frac{\gamma}{2}||G_{DA}(\theta) - G_{DA}(\theta')||^2,$$
$$f(G_{DA}(\theta')) - f(G_{DA}(\theta)) \geq \frac{\gamma}{2}||G_{DA}(\theta) - G_{DA}(\theta')||^2.$$

Combining the two we have,

$$-\gamma||G_{DA}(\theta) - G_{DA}(\theta')||^2 \geq (G_{DA}(\theta) - G_{DA}(\theta'))^T \nabla f(G_{DA}(\theta')). \tag{3.32}$$

Now we know the follwoing holds by the Cauchy-Schwartz inequality:

$$\begin{aligned}(G_{DA}(\theta) - G_{DA}(\theta'))^T (\nabla f(G_{DA}(\theta')) - \nabla f'(G_{DA}(\theta'))) \leq ||G_{DA}(\theta) - G_{DA}(\theta')|| \times \\ ||\nabla f(G_{DA}(\theta')) - \nabla f'(G_{DA}(\theta'))||\end{aligned} \tag{3.33}$$

We also have the following:

$$\begin{aligned}&||\nabla f(G_{DA}(\theta')) - \nabla f'(G_{DA}(\theta'))|| \\ &= \left|\left|\nabla \frac{1}{k}\sum_{i=1}^k \mathbb{E}_{\tau\sim p(\tau;\theta_i)}\left[\sum_{t=1}^T l_t(s_t, G_{DA}(\theta'))\right] - \nabla \frac{1}{k}\sum_{i=1}^k \mathbb{E}_{\tau\sim p(\tau;\theta'_i)}\left[\sum_{t=1}^T l_t(s_t, G_{DA}(\theta'))\right]\right|\right| \\ &\leq \frac{1}{k}\sum_{i=1}^k \left|\left|\nabla \mathbb{E}_{\tau\sim p(\tau;\theta_i)}\left[\sum_{t=1}^T l_t(s_t, G_{DA}(\theta'))\right] - \nabla \mathbb{E}_{\tau\sim p(\tau;\theta'_i)}\left[\sum_{t=1}^T l_t(s_t, G_{DA}(\theta'))\right]\right|\right| \\ &\leq \frac{1}{k}\sum_{i=1}^k \beta\varepsilon||\theta_i - \theta'_i||\end{aligned} \tag{3.34}$$

In the last inequality we have used the ε-sensitivity of the distribution $p(\tau;\theta)$ and the Kantorovich-Rubinstein inequality [15]. Thus we have,

$$||G_{DA}(\theta) - G_{DA}(\theta')|| \times ||\nabla f(G_{DA}(\theta')) - \nabla f'(G_{DA}(\theta'))|| \leq \frac{\beta\varepsilon}{k}||G_{DA}(\theta) - G_{DA}(\theta')|| \sum_{i=1}^{k} ||\theta_i - \theta'_i||.$$

We finally have the following:

$$(G_{DA}(\theta) - G_{DA}(\theta'))^T(\nabla f(G_{DA}(\theta')) - \nabla f'(G_{DA}(\theta'))) \geq -\frac{\beta\varepsilon}{k}||G_{DA}(\theta) - G_{DA}(\theta')|| \sum_{i=1}^{k} ||\theta_i - \theta'_i||.$$

By optimality condition we have $(G_{DA}(\theta) - G_{DA}(\theta'))^T \nabla f'(G_{DA}(\theta')) \geq 0$. Thus we have,

$$(G_{DA}(\theta) - G_{DA}(\theta'))^T \nabla f(G_{DA}(\theta')) \geq -\frac{\beta\varepsilon}{k}||G_{DA}(\theta) - G_{DA}(\theta')|| \sum_{i=1}^{k} ||\theta_i - \theta'_i||. \quad (3.35)$$

Combining this with (3.32), we have,

$$-\gamma||G_{DA}(\theta) - G_{DA}(\theta')||^2 \geq -\frac{\beta\varepsilon}{k}||G_{DA}(\theta) - G_{DA}(\theta')|| \sum_{i=1}^{k} ||\theta_i - \theta'_i||. \quad (3.36)$$

Rearranging we have,

$$||G_{DA}(\theta) - G_{DA}(\theta')|| \leq \frac{\beta\varepsilon}{\gamma k} \sum_{i=1}^{k} ||\theta_i - \theta'_i||. \quad (3.37)$$

We know that $\theta_{k+1} = G_{DA}(\theta_k)$, according to the update equation and $\theta_{PS} = G_{DA}(\theta_{PS})$, according to the definition of performatively stable point [15]. Putting these into the above inequality we have,

$$||\theta_{k+1} - \theta_{PS}|| \leq \frac{\beta\varepsilon}{\gamma k} \sum_{i=1}^{k} ||\theta_i - \theta'_i||. \quad (3.38)$$

This proves that if there is sequence of iterates $\{\theta'_i\}_{i=1}^k$ which lead to a performatively stable point θ_{PS}, then the DAGGER iterates $\{\theta_i\}_{i=1}^k$ remaining close to the performatively stable iterates, will converge to a performatively stable point.

3.2.3 Improvements of the proposed SGD algorithm over DAGGER

1. DAGGER algorithm treats all the state-action pairs as independent, which is not actually true. Inside each trajectory, the current states depend on the previous states

and actions. In the SGD approach, we have treated trajectories as independent and have not assumed anything about the independence of state-action pairs. The derivation of the stochastic gradient is done based on the assumption that the trajectories are independent and the present states depend only on the past states and actions and do not depend on the future states, both of which seem to be practical assumptions.

2. The SGD algorithm is much more computationally efficient compared to the DAGGER algorithm. It does not require the aggregation of previous trajectories and hence the per iteration complexity of the proposed algorithm is much lower compared to DAGGER.

3. From the optimization perspective, DAGGER can be looked as an aggragated version of the RRM proposed in [15]. Hence the DAGGER algorithm only converges to a performatively stable point. While the proposed SGD algorithm converges to a perfomatively optimal point [20].

4. For the SGD algorithm, we can use various variance reduced methods using the stochastic gradient which will help us in reducing the number of calls made to the expert, further reducing the computational burden.

3.3 Stochastic Recursive Momentum (STORM)

In order to make the convergence of the SGD for IL algorithm more stable we explore some variance reduced versions of the SGD algorithm. There have been a lot of adaptive methods recently which seem to outperform the vanilla SGD such as AdaGrad [25], RMSProp [27] and Adam [26]. However, the theoretical guarantess of these adaptive algorithms are not well established. In this section we explore a momentum based variance reduced algorithm for stochastic optimization called STORM. It has sound theoretical guarantees and seems to work well in practice. It is a variance reduced algorithm for SGD. It uses the momentum term in order to achieve variance reduction. It uses two gradient computations in each step. This algorithm doesn't need batch gradient calculations and incorporates adaptive learning rates, which makes it easier to implement and reduces the need for adjusting hyperparameters. This algorithm is summarized in Algorithm 4.

Algorithm 4 STORM for IL

Initialize θ_1, $k = 1$, w, c, and k'.
Sample τ_1.
$G_1 = ||\nabla f(\theta_1, \tau_1)||$.
$d_1 = \nabla f(\theta_1, \tau_1)$
$\eta_0 = \frac{k'}{w^{1/3}}$.
while $k \leq K$ **do**
  $\eta_k = \frac{k'}{(w+\sum_{i=1}^{k} G_i^2)^{1/3}}$.
  Update θ as $\theta_{k+1} = \theta_k - \eta_k * d_k$.
  $a_{k+1} = c\eta_k^2$
  Sample $\tau_{k+1} \sim p(\tau; \theta_{k+1})$.
  $G_{k+1} = ||\nabla f(\theta_{k+1}, \tau_{k+1})||$.
  $d_{k+1} = \nabla f(\theta_{k+1}, \tau_{k+1}) + (1 - a_{k+1})\nabla f(\theta_k, \tau_{k+1})$.
  $k = k + 1$.
end while
Return θ_K

We first look at the intuition behind the STORM algorithm. The typical momentum with SGD is achieved as follows:

$$\begin{aligned} d_k &= (1-a)d_{k-1} + a\nabla f(\theta_k, \tau_k) \\ \theta_{k+1} &= \theta_k - \eta d_k \end{aligned} \tag{3.39}$$

The use of noise in stochastic gradients is known to negate the theoretical benefits of the momentum factor, even though momentum-based SGD has been successful in numerous machine learning applications. According to [33], the STORM algorithm takes a different approach by demonstrating that a variant of momentum can lower gradient variance, rather than relying on the accelerated rates provided by momentum in SGD. The proposed variant, as described [33], is as follows:

$$\begin{aligned} d_k &= (1-a)d_{k-1} + a\nabla f(\theta_k, \tau_k) + (1-a)(\nabla f(\theta_k, \tau_k) - \nabla f(\theta_{k-1}, \tau_k)) \\ \theta_{k+1} &= \theta_k - \eta d_k \end{aligned} \tag{3.40}$$

The only difference is the addition of $(1-a)(\nabla f(\theta_k, \tau_k) - \nabla f(\theta_{k-1}, \tau_k))$ to the momentum part. Assuming smoothness, if an algorithm is converging towards a stable point, then the two terms mentioned will be similar. In this scenario, the algorithm will operate similarly to classic momentum SGD during the final stages of optimization. [33]. We will use the stochastic gradient derived in (3.16), along with the STORM algorithm in order to optimize the IL objective.

3.3.1 Guarantees for non-adaptive STORM

We have the following assumptions while deriving the theoretical guarantees for the STORM algorithm:

1. We assume that f is L smooth. This means that $\exists L$, such that:

$$||\nabla f(\theta_1) - \nabla f(\theta_2)|| \leq L||\theta_1 - \theta_2||. \tag{3.41}$$

   This implies that the IL objective F is also L-smooth.

2. Bounded Variance Assumption:

$$\mathbb{E}_{\tau \sim p(\tau;\theta)}[||\nabla f(\theta, \tau) - \nabla F(\theta)||^2] \leq \sigma^2. \tag{3.42}$$

We use the notations $e_k = \mathbb{E}[||d_k - \nabla F(\theta_k)||^2]$, $\epsilon_k = d_k - \nabla F(\theta_k)$ and $\Delta_k = \mathbb{E}[||\nabla F(\theta_k)||^2]$ in the proof. We will first derive a recursion on e_k, which is termed as the tracking error because it measures how well d_k is able to track the actual gradient $\nabla F(\theta_k)$. The basic outline of the proof is that we first bound e_k and Δ_k and then combine the bounds to get the final result. For the non-adaptive case the following choice of η_k and α_{k+1} is used:

$$\eta_k = \frac{1}{4L}\left(\frac{7}{k+7}\right)^{1/3} \text{ and } \alpha_{k+1} = 16\eta_k^2 L^2 = \left(\frac{7}{k+7}\right)^{2/3}. \tag{3.43}$$

The recursion on e_k is stated in the following lemma:

Lemma 3.3. *With the notations introduced above we have,*

$$e_{k+1} \leq (1 - a_{k+1})^2(1 + 4L^2\eta_k^2)e_k + 4(1 - \alpha_{k+1})^2 L^2 \Delta_k + 2\alpha_{k+1}^2\sigma^2. \tag{3.44}$$

Proof. We have the following relation [33]:

$$\begin{aligned}\epsilon_{k+1} =& (1 - a_{k+1})\epsilon_k + (1 - a_{k+1})(\nabla f(\theta_{k+1}, \tau_{k+1}) - \nabla f(\theta_k, \tau_{k+1}) - \nabla F(\theta_{k+1}) + \nabla F(\theta_k)) \\ &+ a_{k+1}(\nabla f(\theta_{k+1}, \tau_{k+1}) - \nabla F(\theta_{k+1})) = A + B + C.\end{aligned}$$

We denote the first, second and third term of the recursion as A, B, and C respectively. We note that A is independent of τ_{k+1}, hence the expectation with respect to τ_{k+1} and

norm square can be written as follows:

$$\mathbb{E}_{k+1}\left[||A+B+C||^2\right] \leq ||A||^2 + \mathbb{E}_{k+1}\left[2||B||^2 + 2||C||^2\right] \tag{3.45}$$

We have the following:

$$B = \nabla f(\theta_{k+1}, \tau_{k+1}) - \nabla f(\theta_k, \tau_{k+1}) - \mathbb{E}_{k+1}[f(\theta_{k+1}, \tau_{k+1}) - \nabla f(\theta_k, \tau_{k+1})] = X - \mathbb{E}_{k+1}[X]. \tag{3.46}$$

Since the variance of X is always less than or equal to its second moment we have,

$$\mathbb{E}\left[||B||^2\right] \leq \mathbb{E}\left[||f(\theta_{k+1}, \tau_{k+1}) - \nabla f(\theta_k, \tau_{k+1})||^2\right]. \tag{3.47}$$

Taking the full expectation and combining all the things we have the following:

$$e_{k+1} \leq (1-a_{k+1})^2 e_k + 2(1-a_{k+1})^2 \mathbb{E}\left[||\nabla f(\theta_{k+1}, \tau_{k+1}) - \nabla f(\theta_k, \tau_{k+1})||^2\right] \tag{3.48}$$

$$+2a_{k+1}^2 \mathbb{E}\left[||\nabla f(\theta_{k+1}, \tau_{k+1}) - \nabla F(\theta_{k+1})||^2\right]. \tag{3.49}$$

Using the smoothness and the bounded variance assumption, we thus have,

$$e_{k+1} \leq (1-a_{k+1})^2 e_k + 2(1-a_{k+1})^2 L^2 \mathbb{E}\left[||\theta_{k+1} - \theta_k||^2\right] + 2a_{k+1}^2 \sigma^2. \tag{3.50}$$

From the update of the STORM algorithm we have, $\theta_{k+1} - \theta_k = -\eta_k d_k$.

$$e_{k+1} \leq (1-a_{k+1})^2 e_k + 2(1-a_{k+1})^2 \eta_k^2 L^2 \mathbb{E}\left[||d_k||^2\right] + 2a_{k+1}^2 \sigma^2. \tag{3.51}$$

We also have the following bound:

$$||d_k||^2 = ||\epsilon_k + \nabla F(\theta_k)||^2 \leq 2(e_k + \Delta_k). \tag{3.52}$$

Thus combining the above bounds we have the recursion on e_k as stated in the lemma. Now we state the bound on Δ_t in the following lemma.

Lemma 3.4. *We have the following bound on Δ_k:*

$$\eta_k \Delta_k \leq 4\mathbb{E}[F(\theta_k) - F(\theta_{k+1})] + 3\eta_k e_k. \tag{3.53}$$

Proof. Using the quadratic upper bound on F, we have,

$$F(\theta_{k+1}) \leq F(\theta_k) + \langle \nabla F(\theta_k), \theta_{k+1} - \theta_k \rangle + \frac{L^2}{2} ||\theta_{k+1} - \theta_k||^2 \tag{3.54}$$

Again using the update equation we have,

$$\begin{aligned} F(\theta_{k+1}) &\leq F(\theta_k) - \eta_k \langle \nabla F(\theta_k), d_k \rangle + \frac{\eta_k^2 L^2}{2} ||d_k||^2 \\ &= F(\theta_k) - \eta_k ||\nabla F(\theta_k)||^2 - \eta_k \langle \nabla F(\theta_k), d_k - \nabla F(\theta_k) \rangle + \frac{\eta_k^2 L^2}{2} ||d_k||^2. \end{aligned} \tag{3.55}$$

Taking the expectation we have,

$$\mathbb{E}[F(\theta_{k+1}) - F(\theta_k)] \leq -\eta_k \Delta_k - \eta_k \mathbb{E}\left[\langle \nabla F(\theta_k), d_k - \nabla F(\theta_k) \rangle\right] + \frac{\eta_k^2 L^2}{2} \mathbb{E}\left[||d_k||^2\right]. \tag{3.56}$$

We have the following bounds:

$$-\mathbb{E}\left[\langle \nabla F(\theta_k), d_k - \nabla F(\theta_k) \rangle\right] \leq (e_k + \Delta_k)/2 \text{ and } \mathbb{E}\left[||d_k||^2\right] \leq 2(e_k + \Delta_k). \tag{3.57}$$

Combining the above bounds we have,

$$\mathbb{E}[F(\theta_{k+1}) - F(\theta_k)] \leq -\frac{\eta_k}{2}\left(1 - \frac{\eta_k L}{2}\right)\Delta_k + \eta_k \left(1 + \frac{\eta_k L}{2}\right) \leq -\frac{\eta_k}{4}\Delta_k + \frac{3\eta_k}{4} e_k. \tag{3.58}$$

Here we have assumed that $\eta_k L < 2$. This gives us the final bound for Δ_k.

Now we assume that $a_{k+1} = 16\eta_k^2 L^2$ and then multiply the bound of Lemma 3.3 by $\eta_k / 2a_{k+1}$ and add it to the bound of Lemma 3.4. Thus we have,

$$\begin{aligned} \mathbb{E}[F(\theta_{k+1}) - F(\theta_k)] + \frac{\eta_k}{2a_{k+1}} e_{k+1} - \frac{\eta_{k-1}}{2a_k} e_k \leq &\left[\frac{\eta_k}{2a_{k+1}}\left((1 - a_{k+1})^2\left(1 + \frac{a_{k+1}}{4}\right) + \frac{3a_{k+1}}{2}\right) - \frac{\eta_{k-1}}{2a_k}\right] e_k \\ &+ \frac{\eta_k}{8}[(1 - a_{k+1})^2 - 2]\Delta_k + \eta_k a_{k+1} \sigma^2. \end{aligned}$$

The choices listed below result in the first term being negative:

$$\eta_k = \frac{1}{4L}\left(\frac{7}{k+7}\right)^{1/3} \text{ and } \alpha_{k+1} = 16\eta_k^2 L^2 = \left(\frac{7}{k+7}\right)^{2/3}. \tag{3.59}$$

Thus we have,

$$\mathbb{E}[F(\theta_{k+1}) - F(\theta_k)] + \frac{\eta_k}{2a_{k+1}} e_{k+1} - \frac{\eta_{k-1}}{2a_k} e_k \leq -\frac{\eta_k}{8}\Delta_k + \eta_k a_{k+1} \sigma^2.$$

Carrying out the telescopic sum we have,

$$\mathbb{E}[F(\theta_{K+1}) - F(\theta_1)] + \frac{\eta_K}{2a_{K+1}} e_{K+1} - \frac{\eta_0}{2a_1} e_1 \leq -\frac{1}{8} \sum_{k=1}^{K} \eta_k \Delta_k + \sigma^2 \sum_{k=1}^{K} \eta_k a_{k+1}.$$

Using the bounded variance assumption we have $e_1 \leq \sigma^2$. Thus we have the following guarantee for the STORM algorithm,

$$\min_{1 \leq k \leq K} \mathbb{E}[||\nabla F(\theta_k)||] \leq \mathcal{O}\left(\frac{\sqrt{LB_F + \sigma^2 \log T}}{T^{1/3}}\right) \tag{3.60}$$

This makes $T = O(\epsilon^{-3})$, for $\min_{1 \leq k \leq K} \mathbb{E}[||\nabla F(\theta_k)||] \leq \epsilon$. This provides us with a rate of convergence of $O(\epsilon^{-3})$ which is much better than the SGD algorithm for IL with a convergence rate of $O(\epsilon^{-4})$. For the fully adaptive case we have the same bound for $\sigma^2 \neq 0$. But for the case $\sigma^2 = 0$, the bound becomes $T = \mathcal{O}(\epsilon^{-2})$ as shown in [33]. Thus the STORM algorithm is adaptive to the variance in the stochastic gradient.

3.4 Stochastic Non-smooth Non-convex Optimization through Online-to-Non-convex Conversion

In the previous sections we have assumed that the objective function is smooth. When the objective is non-smooth, the IL optimization problem presents several challenges, and we examine those challenges in this section. Many contemporary neural networks architectures used in training are not smooth. Due to the absence of smoothness in the objective, the theoretical assurances may not actually apply, and the algorithms described in the previous part can only provide a sense of what might happen when these algorithms are applied in practice. Points ϵ away from the stationary points were easily located with the aid of the smoothness assumption; specifically, a point θ where $||\nabla F(\theta)|| \leq \epsilon$. Since practical optimization is possible even for non-smooth objectives, we require a different set of mathematical tools to study them. We shall use the (δ, ϵ)-stationary point specified in [39]. Searching for (δ, ϵ)-stationary point means that instead of $||\nabla F(\theta)|| \leq \epsilon$, we ask that there is a random variable y supported in a ball of radius δ about θ such that $||\mathbb{E}[\nabla F(y)]|| \leq \epsilon$. Cutkosky et al., in [39] showed that such an (δ, ϵ)-stationary point can be identified with only $\mathcal{O}(\epsilon^{-3}\delta^{-1})$ stochastic gradient evaluations. The primary technique which they used was *online-to-non-convex conversion*, which establishes a connection between non-convex stochastic optimization and online learning.

In order to analyze the non-smooth non-convex optimization we assume that the IL objective satisfies the following mild regularity condition.

Definition 3.5. A differentiable function $F : \mathcal{H} \to \mathbb{R}$ is said to be well-behaved if it satisfies the following condition for all $x, y \in \mathcal{H}$:

$$F(y) = F(x) + \int_0^1 \langle \nabla F(x + s(y - x)), y - x \rangle ds. \tag{3.61}$$

This assumption is used to obtain the theoretical guarantees of finding an (δ, ϵ)-stationary point along with the unbiased and bounded variance assumption of the stochastic gradient. Now we formally define the (δ, ϵ)-stationary point as follows [39]:

Definition 3.6. A point θ is an (δ, ϵ)-stationary point of a differentiable function F if there is a finite subset S of the ball of radius δ centered at θ such that for y selected uniformaly randomly from S, $\mathbb{E}[y] = x$ and $||\mathbb{E}[\nabla F(y)]|| \leq \epsilon$.

3.4.1 Online Learning

In online learning, all the data is not available at once and the learning proceeds in rounds. In each round the algorithm outputs a point Δ_t as its prediction and receives a linear loss $l_t(.) = \langle g_t, . \rangle$, where g_t is the value we are trying to predict. Based on the output of the algorithm, a loss of $l_t(\Delta_t)$ is incurred. The goal of the algorithm is to minimize the regret over T rounds, defined as the difference between its cumulative loss and that of an arbitrary,fixed competitor u:

$$R_T(u) = \sum_{t=1}^{T} \langle g_t, \Delta_t - u \rangle \tag{3.62}$$

Another more challenging objective in online learning setting is the minimization of a K-shifting regret, where the arbitrary competitor is a sequence of K vectors $u^1, \ldots, u^K$ that changes every T iterations:

$$R_T(u^1, \ldots, u^K) = \sum_{k=1}^{K} \sum_{n=(k-1)T+1}^{kT} \langle g_n, \Delta_n - u^k \rangle \tag{3.63}$$

3.4.2 Online-to-non-convex Algorithm

We dissect the steps necessary to make the transition from online to non-convex. The key idea is to treat the shifting regret over linear losses as a problem of minimization of a non-convex and non-smooth function. In particular, consider a $\theta_n = \theta_{n-1} + \Delta_n$ iterative optimization process that updates the previous iterate in the direction Δ_n. For a specific rate of learning, $\Delta_n = -\eta f(\theta_n, \tau_n)$ is the one selected by SGD. However, in online-to-non-convex conversion, the update direction Δ_n is left up to an online learning algorithm $\mathcal{A}$. The online to non-convex conversion is summarized in Algorithm 5.

Algorithm 5 Online to non-convex conversion for IL

Initialize θ_0, $K \in \mathbb{N}$, $T \in \mathbb{N}$, s_n for all n and online algorithm $\mathcal{A}$.
Set $M = K.T$
Set $n = 1$.
while $n \leq M$ **do**
  Get update direction Δ_n from $\mathcal{A}$.
  $\theta_n = \theta_{n-1} + \Delta_n$.
  $w_n = \theta_{n-1} + s_n \Delta_n$.
  Sample $\tau_n \sim p(\tau; \theta_n)$.
  Query the stochastic gradient oracle $f_n = \text{GRAD}(w_n, \tau_n)$.
  Send f_n to $\mathcal{A}$.
  $n = n + 1$.
end while
Set $w_t^k = w_{(k-1)T+t}$ for $k = 1, \ldots, K$ and $t = 1, \ldots, T$.
$\bar{w}^k = \frac{1}{T} \sum_{t=1}^{T} w_t^k$ for $k = 1, \ldots, K$.
Return $\{\bar{w}^1, \ldots, \bar{w}^K\}$.

3.4.3 Theoretical Guarantees of Online-to-non-convex conversion

In order to establish the guarantees for the online-to-non-convex algorithm we first, try to express the objective in terms of the shifting regret of an online learning algorithm.

Theorem 3.7. *Suppose F is well-behaved. We follow the notation of Algorithm 5. Let s_n be independent random variables uniformly distributed in [0, 1]. Set $T, K \in \mathbb{N}$ and $M = KT$. Define $\nabla_n = \int_0^1 \nabla F(\theta_{n-1} + s\Delta_n) ds$. Then we have the following:*

$$\mathbb{E}[F(\theta_M)] = F(\theta_1) + \mathbb{E}\left[\sum_{n=1}^{M} \langle f_n, \Delta_n - u_n \rangle\right] + \mathbb{E}\left[\sum_{n=1}^{M} \langle f_n, u_n \rangle\right]. \tag{3.64}$$

Proof. By the well-behaveness of the objective F we have the following:

$$\begin{aligned} F(\theta_n) - F(\theta_{n-1}) &= \int_0^1 \langle \nabla F(\theta_{n-1} + s\Delta_n, \Delta_n)\rangle ds \\ &= \int_0^1 \langle \nabla_n, \Delta_n)\rangle ds \\ &= \langle f_n, \Delta_n - u_n\rangle + \langle \nabla_n - f_n, \Delta_n\rangle + \langle f_n, u_n\rangle \end{aligned} \tag{3.65}$$

Now we take the sum from $n = 1$ to M. For s being uniformly randomly distributed in $[0, 1]$, we have $\mathbb{E}[f_n] = \mathbb{E}[\nabla F(w_n)] = \mathbb{E}[\int_0^1 \nabla F(\theta_{n-1} + s\Delta_n)ds] = \mathbb{E}[\nabla_n]$. Putting this into the above statement and taking the expectation we have the statement of the theorem. The main benefit of the theorem comes from the fact that the regret of an online learning algorithm corresponds directly to the term $\langle f_n, \Delta_n - u_n\rangle$. A smaller bound on $F(\theta_M)$ is indicative of a lesser regret. Then, by strategically picking u_n, we can connect the average gradients used in the definition of (δ, ϵ)-stationarity with the term $\langle f_n, u_n\rangle$. Then we have the following theorem:

Theorem 3.8. *Assume F is well-behaved. Following the notation in Algorithm 5, let s_n be a uniformly distributed random variable in $[0, 1]$. Set $T, K \in \mathbb{N}$ and $M = KT$. Define $u^k = -D\frac{\sum_{t=1}^T \nabla F(w_t^k)}{||\sum_{t=1}^T \nabla F(w_t^k)||}$ for some $D > 0$ for $k = 1, \ldots, K$. Suppose $Var(f_n) = \sigma^2$. Then we have the following:*

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^K \left|\left|\frac{1}{T}\sum_{t=1}^T \nabla F(w_t^k)\right|\right|\right] \leq \frac{F(\theta_0) - F^*}{DM} + \frac{\mathbb{E}[R_T(u^1, \ldots, u^K)]}{DM} + \frac{\sigma}{\sqrt{T}}. \tag{3.66}$$

Proof. Set u_n to u^1 for the first T iterations, u^2 for the second T iterations and so on. Thus, $u_n = u^{mod(n,T)+1}$ for $n = 1, \ldots, M$. Thus we have,

$$\mathbb{E}[F(\theta_M)] = F(\theta_1) + \mathbb{E}[R_T(u^1, \ldots, u^K)] + \mathbb{E}\left[\sum_{n=1}^M \langle f_n, u_n\rangle\right]. \tag{3.67}$$

Since $u^k = -D\frac{\sum_{t=1}^{T}\nabla F(w_t^k)}{||\sum_{t=1}^{T}\nabla F(w_t^k)||}$, $\mathbb{E}[f_n] = \nabla F(w_n)$, and $Var(g_n) = \sigma^2$, we have,

$$\begin{aligned}\mathbb{E}\left[\sum_{n=1}^{M}\langle f_n, u_n\rangle\right] &= \mathbb{E}\left[\sum_{k=1}^{K}\sum_{t=1}^{T}\left\langle g_n + \nabla F(w_t^k) - \nabla F(w_t^k), u^k\right\rangle\right] \\ &= \mathbb{E}\left[\sum_{k=1}^{K}\left\langle \sum_{t=1}^{T} g_n + \nabla F(w_t^k) - \nabla F(w_t^k), u^k\right\rangle\right] \\ &= \mathbb{E}\left[\sum_{k=1}^{K}\left\langle \sum_{t=1}^{T}\nabla F(w_t^k), u^k\right\rangle\right] + \mathbb{E}\left[\sum_{k=1}^{K}\left\langle \sum_{t=1}^{T} g_n - \nabla F(w_t^k), u^k\right\rangle\right]\end{aligned} \tag{3.68}$$

Looking at the second term,

$$\mathbb{E}\left[\sum_{k=1}^{K}\left\langle \sum_{t=1}^{T} g_n - \nabla F(w_t^k), u^k\right\rangle\right] = D\mathbb{E}\left[\sum_{k=1}^{K}\left\langle \sum_{t=1}^{T} -g_n + \nabla F(w_t^k), \frac{\sum_{t=1}^{T}\nabla F(w_t^k)}{||\sum_{t=1}^{T}\nabla F(w_t^k)||}\right\rangle\right] \tag{3.69}$$

$$\leq D\mathbb{E}\left[\sum_{k=1}^{K}\sum_{t=1}^{T}\left|\left|-g_n + \nabla F(w_t^k)\right|\right|\right] \text{ (By Cauchy Schwartz)}$$

$$\leq D\sum_{k=1}^{K}\sqrt{\sum_{t=1}^{T}\mathbb{E}\left[\left|\left|-g_n + \nabla F(w_t^k)\right|\right|^2\right]} \text{(By Jensen's)} \leq DK\sqrt{T}\sigma.$$

Thus we have the following:

$$\mathbb{E}\left[\sum_{n=1}^{M}\langle f_n, u_n\rangle\right] \leq \mathbb{E}\left[-\sum_{k=1}^{K} DT\left|\left|\frac{1}{T}\sum_{t=1}^{T}\nabla F(w_t^k)\right|\right|\right] + DK\sqrt{T}\sigma. \tag{3.70}$$

Putting all the inequalities together and dividing by $KDT = KM$, we get the statement of the theorem.

We now take the online algorithm $\mathcal{A}$, as the online gradient descent (OGD) of [40]. The update $\Delta_{n+1} = \prod_{||\Delta||\leq D}[\Delta_n - \eta f_n]$ is calculated by OGD, which receives as inputs the radius D and the step size η. It can be shown analytically that if $\mathbb{E}[||f_n||^2] \leq G^2$ for all n, then for each u satisfying $||u|| \leq D$, OGD will guarantee1 static regret $\mathbb{E}[R_T(u)] \leq DG\sqrt{T}$. Thus, we obtain a shifting regret of $\mathbb{E}[R_T(u^1, \ldots, u^K)] \leq KDG\sqrt{T}$ by restarting the algorithm every T iterations.

Corollary 3.9. *Let the number of gradient evaluations required be N. We use the assumptions of Theorem 3.8, let us also assume $\mathbb{E}[||f_n||^2] \leq G^2$ and that $\mathcal{A}$ ensures $||\Delta_n|| \leq D$ for some user-specified D for all n and ensures the worst case K-shifting regret bound $\mathbb{E}[R_T(u^1, \ldots, u^K)] \leq KDG\sqrt{T}$ for all $||u^k|| \leq D$. Let $\delta > 0$ be an arbitrary number.*

Let $D = \delta/T, T = \min\left(\left\lceil (\frac{GN\delta}{F(\theta_0)-F^*})^{2/3}\right\rceil, \frac{N}{2}\right)$ *and* $K = \lfloor \frac{N}{T} \rfloor$. *Then for all* k *and* t, $||\bar{w}^k - w_t^k|| \leq \delta$ *and,*

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K}\left|\left|\frac{1}{T}\sum_{t=1}^{T}\nabla F(w_t^k)\right|\right|\right] \leq \frac{2(F(\theta_0)-F^*)}{\delta N} + \max\left(\frac{5G^{2/3}(F(\theta_0)-F^*)^{1/3}}{(N\delta)^{1/3}}, \frac{6G}{\sqrt{N}}\right) \tag{3.71}$$

Proof. We know that $\mathcal{A}$ guarantees $||\Delta_n|| \leq D$, for all $n < n' \leq T + 1 - 1$, we have

$$\begin{aligned} ||w_n - w_{n'}|| &= ||\theta_n - (1-s_n)\Delta_n - \theta_{n'-1} + s_n\Delta_{n'}|| \\ &\leq ||\sum_{i=n+1}^{n'-1}\Delta_i|| + ||\Delta_n|| + ||\Delta_{n'}|| \\ &\leq D(n'-n-1) + 2D = D(n'-n+1) \leq DT = \delta. \end{aligned} \tag{3.72}$$

We have $M = KT \geq N - T \geq N/2$. We also have that $Var(f_n) \leq G^2, \forall n$ (since the variance is less than or equal to the second moment). Applying theorem 3.8 along with $\mathbb{E}[R_T(u^1, \ldots, u^K)] \leq KDG\sqrt{T}$, we have:

$$\begin{aligned} \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K}\left|\left|\frac{1}{T}\sum_{t=1}^{T}\nabla F(w_t^k)\right|\right|\right] &\leq \frac{2(F(\theta_0)-F^*)}{DN} + 2\frac{KDG\sqrt{T}}{DN} + \frac{G}{\sqrt{T}} \\ &\leq \frac{2T(F(\theta_0)-F^*)}{\delta N} + 3\frac{G}{\sqrt{T}} \\ &\leq \frac{2(F(\theta_0)-F^*)}{\delta N} + \max\left(\frac{5G^{2/3}(F(\theta_0)-F^*)^{1/3}}{(N\delta)^{1/3}}, \frac{6G}{\sqrt{N}}\right) \end{aligned} \tag{3.73}$$

where the last inequality is due to the choice of T. If we assume the $F(\theta_0) - F^* \leq B_F$, thus for finding a (δ, ϵ)-stationary point we require $N = O(\epsilon^{-3}\delta^{-1}B_F G^2)$ stochastic gradient evaluations. Thus in the IL setting with a non-convex non-smooth objective for finding a (δ, ϵ)-stationary point $O(\epsilon^{-3}\delta^{-1})$ stochastic gradient valuations are required.

Chapter 4

Numerical Results

In this chapter we present the experimental results of the proposed, SGD, STORM, Adam and RMSProp algorithms for IL and compare it against the existing IL algorithms such as DAGGER. We split this section based on the environments on which we test the algorithms and provide separate results for each of the environments.

4.1 Obstacle Avoidance Environment

4.1.1 Environment Description

In the obstacle avoidance environment, the goal is to reach the endpoint from a fixed starting point while avoiding randomly shifting obstacles as depicted in Fig. 4.1. Here, the difficulty for the agent in learning arises from the fact that the beginning position of the obstacle and the step length for movement in the x and y directions both vary from instance to instance. We are taking an environment with agent from start position $(-2.5, 0)$ need to reach goal at position $(2.5, 0)$. A 0.1 maximum velocity and a 0.05 radius apply to any agent or robot. The 0.4-meter-radius obstacle can travel at a speed of 0.02 metres per second. Each time an obstacle moves, it does so from a different starting position (between -0.5 and 0.5) and takes a different sized random step in the x and y directions (within a predetermined range).

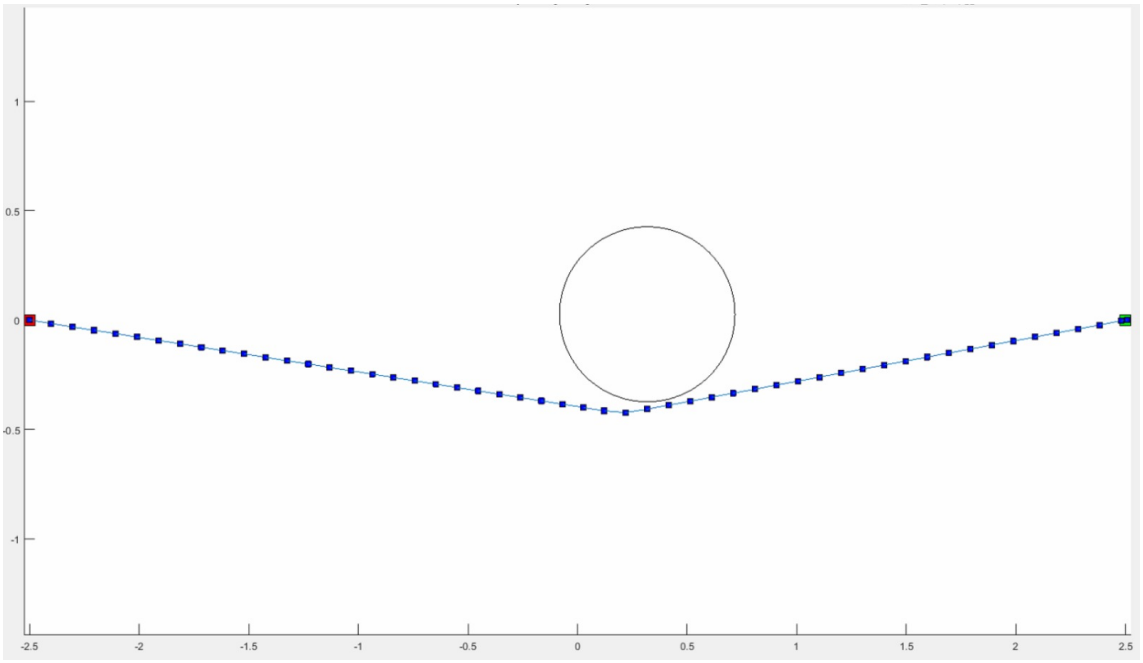


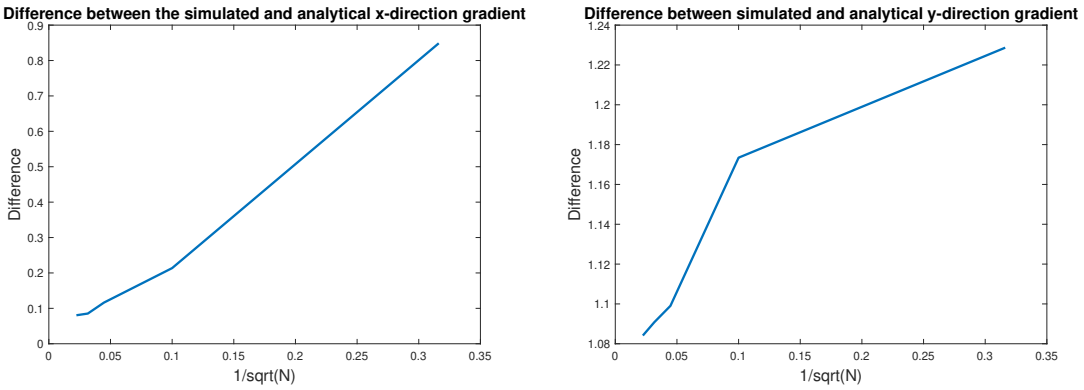
FIGURE 4.1: Obstacle Avoidance Environment

4.1.2 Expert Design

In design of the expert the approach for trajectory planning suggested in [41] is followed for problem formulation which is further solved in MATLAB using fmincon function. Concept of Inverse velocity obstacle is used to design the obstacle avoiding the trajectory. Following formulation is solved to get required agent velocity at each state.

$$\begin{aligned} \min_{u_x,u_y} J =& ||v_{desired} - (v_r + u)||^2 + \lambda||u||^2, \text{subject to,} \\ & \frac{(r^T v)^2}{||v||^2} - ||r||^2 + (R_A + R_B)^2 \leq 0, \\ & v_{desired} = \frac{g^r}{|g^r|} * v_{max}. \end{aligned} \tag{4.1}$$

Here, $v_{desired}$ is the maximum possible agent velocity in the direction of the goal location. u is the control velocity or the change in velocity of the agent suggested by solution of this problem thus is the optimization variable for this problem. v is the relative velocity of the obstacle i.e., $v = v_0 - (v_r + u)$ and r is the position of the obstacle seen by the agent. v_r is agent velocity. R_A and R_B are robot and obstacle radius respectively. Objective is to find a control velocity u that when added to current robot velocity, gives new robot velocity as close to $v_{desired}$ as possible.This objective should be achieved under the constraint that step direction taken in the direction of r with velocity v should always be smaller than the



(a) (b)

FIGURE 4.2: (a) Difference in x-direction (b) Difference in y-direction

critical distance between the agent and obstacle i.e., $||r||^2 - (R_A + R_B)^2$ to avoid crashes with obstacle.

4.1.3 Results for this environment

In order to legitimize the expression of the stochastic gradient derived in equation (3.16), we plot the difference between the simulated and the analytical gradients by varying the number of samples on which the simulated gradients are calculated. In Fig. 4.2, we observe that the difference in the simulated and analytical gradients reduces as the number of samples increase. We see that the difference in the gradients reduce at a rate of approximately $1/\sqrt{N}$, where N is the number of samples. This plot helps us validate the expression derived in (3.16).

Next we compare various algorithms including Behavioral Cloning (BC), DAGGER, SGD, Adam, RMSProp and STORM, based on some performance metrics for the obstacle avoidance environment. In Fig. 4.3, we plot the success rate achieved by various algorithms by varying the number of trajectories. We observe that the BC algorithm has a success rate of around 70% and it remains constant with the increase in the number of expert trajectories. The SGD algorithm has a success rate of around 82.5% while for the RMSProp algorithm's success rate varies between 72% and 85%. STORM, DAGGER and Adam have very similar performance in terms of success rate and are able to get a success rate of around 90% for large number of expert trajectories.

In Fig. 4.4 we plot the number of crashes for the various algorithms. As expected this plot shows the opposite trend as compared to Fig. 4.3. The best algorithms with the highest

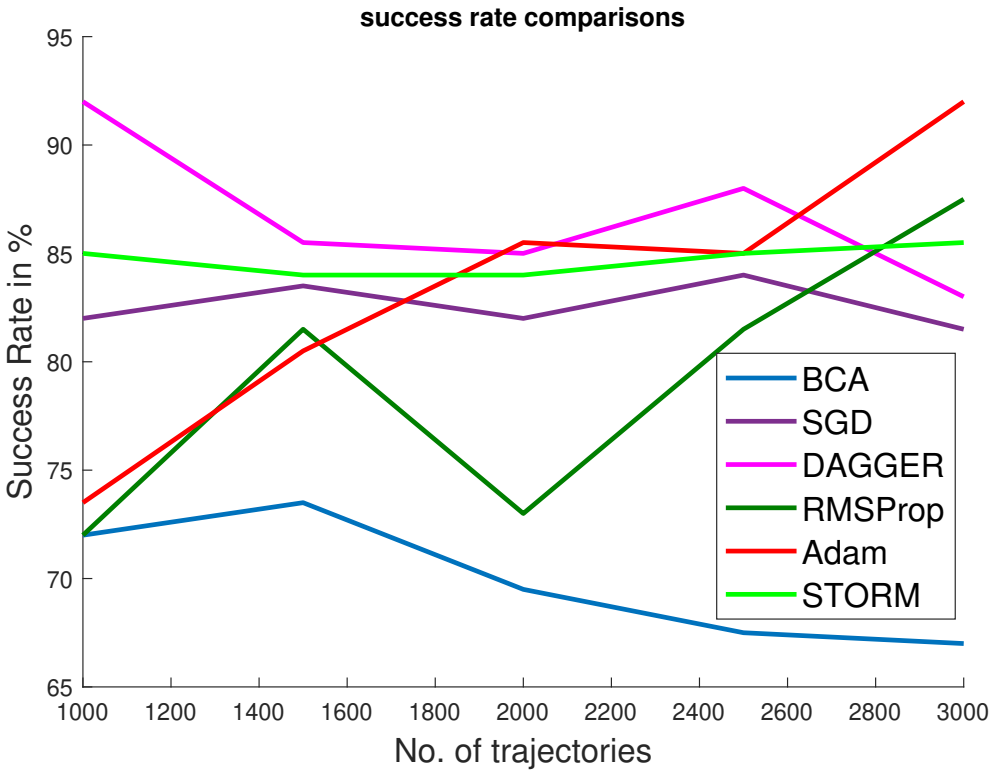


FIGURE 4.3: Success Rate

success rate have the lowest number of crashes. Thus BC has a high number of crashes compared to the other algorithms. The rest of the algorithms are comparable in terms of number of crashes.

Next in Fig. 4.5 we plot the norm of the gradient values in the x and y directions as a function of iterations. As expected the gradient value decreases as the number of iterations increase. We observe that in the y-direction all the algorithms seem comparable since there is not much change in the gradient. But in the x direction we observe that the STORM algorithm converges faster than the other algorithms and it also shows the variance reduction effect. This plot helps us in ensuring that the agent is converging towards a stationary point.

In Fig. 4.6 we plot the loss values in both the x and y directions. We also plot the loss of the DAGGER algorithm in addition to the other algorithms. We observe that for the DAGGER algorithm there is not much change in the loss value. This is because the DAGGER algorithm retrains the model on the entire dataset in each iteration. This makes the loss value very low for all iteration at the cost of computational complexity.

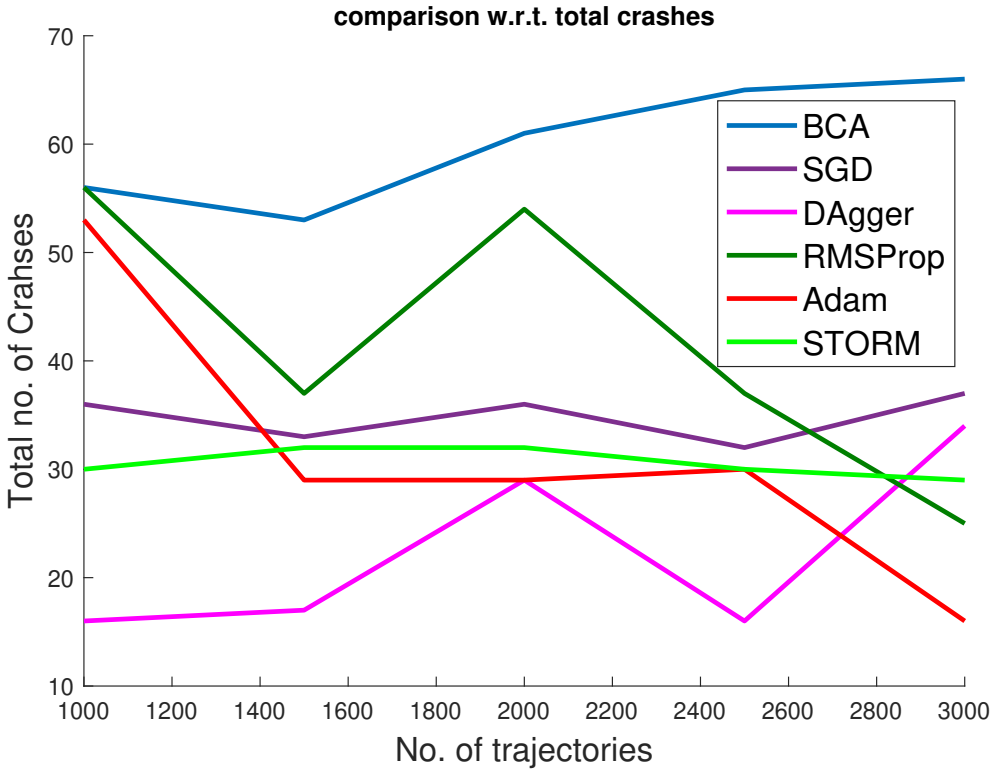
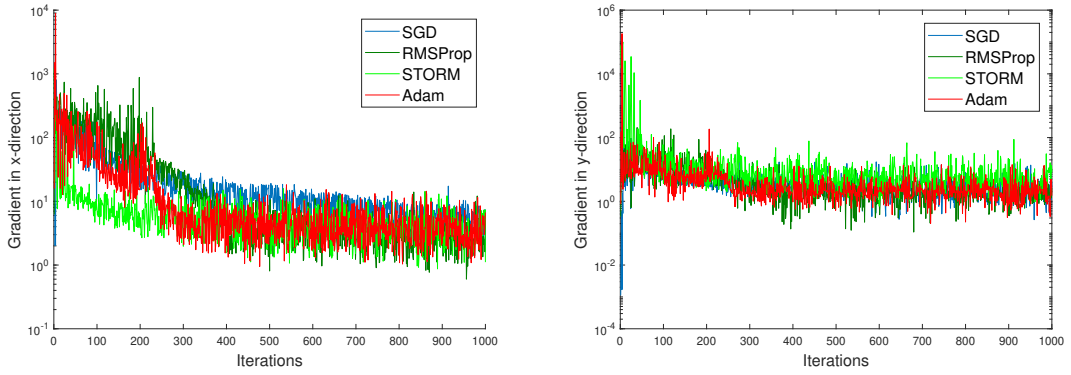
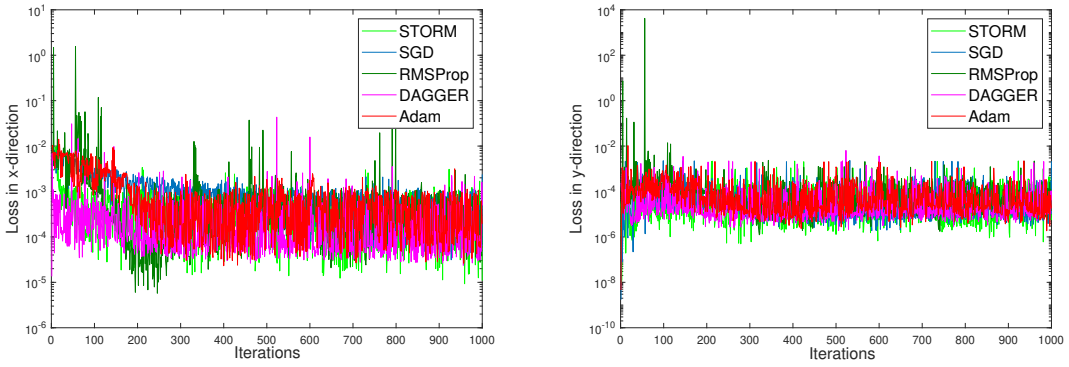


FIGURE 4.4: No. of crashes



(a) (b)

FIGURE 4.5: (a) Gradient in x-direction (b) Gradient in y-direction



(a) (b)

FIGURE 4.6: (a) Loss in x-direction (b) Loss in y-direction

In Table 4.1 we show the number of gradient calls required by the algorithms to converge to a good policy. The DAGGER algorithm requires the most number of gradient calls. The proposed STORM algorithm requires the least number of gradient calls.

Algorithm	No. of gradient calls
STORM	2764 ± 1300
SGD	9450 ± 2886
RMSProp	6411 ± 1880
DAGGER	53704 ± 4233
Adam	10785 ± 3460

TABLE 4.1: No of gradient calls to converge for Obstacle Avoidance.

Algorithm	Runtime in seconds
STORM	561.37
SGD	786.64
RMSProp	594.49
DAGGER	1064.589
Adam	1734.45

TABLE 4.2: Runtime for Obstacle Avoidance.

In Table 4.2, we plot the runtime of the different algorithms. As expected the runtime of the DAGGER algorithm is much higher as compared to the other algorithms. This is because it aggregates the dataset and at each iteration retrains the model on the entire aggregated dataset. Hence as the size of the dataset increases the per iteration complexity of the DAGGER algorithm also increases. The proposed STORM and SGD algorithms have much lower runtime while maintaining the accuracy of the learnt policy. They are also comparable on the basis of the number of expert calls. Thus we have proposed an efficient and scalable algorithm for IL to be applied to complex learning tasks.

4.2 CartPole Environment

4.2.1 Environment Description

This is the classic control environment of OpenGym with discrete action space is shown in Fig. 4.7. An inert joint connects a pole to a cart, which rolls along a smooth track. The pole must be upright on the cart, and the pole must be balanced by pushing in opposite directions. Since the objective is to maintain the pole's vertical position for as long as possible, a reward of +1 is awarded for each step completed, including the final termination step. The maximum award amount is 500 for an episode. We use PPO as expert for this environment.

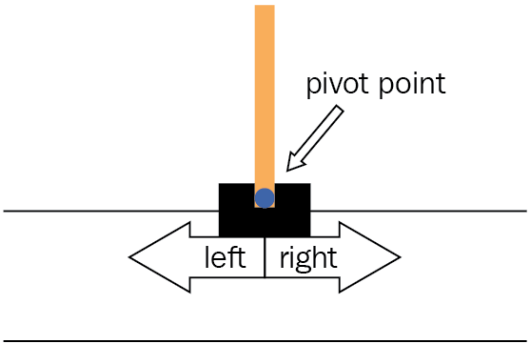


FIGURE 4.7: Cart Pole Environment

4.2.2 CartPole Results

In Fig. 4.8, we plot the loss as the function of number of gradient calls. The loss reduces as the number of gradient calls increase. We observe that the proposed algorithms converge much faster than the DAGGER algorithm. Although the STORM algorithm takes the most number of gradient calls as compared to Adam and RMSProp to converge but the variance reduced effect is observable in the plot.

In Fig. 4.9 we compare the number of crashes. In order to get this plot we reset the environment after every crash till the agent is able to complete the entire 500 steps of the episode. Again the number of crashes of the proposed algorithms goes to zero faster than the DAGGER algorithm. The other three algorithms STORM, Adam and RMSProp are comparable in terms of the number of crashes. Among these three we again observe that the variance of the STORM algorithm seems to be the lowest.

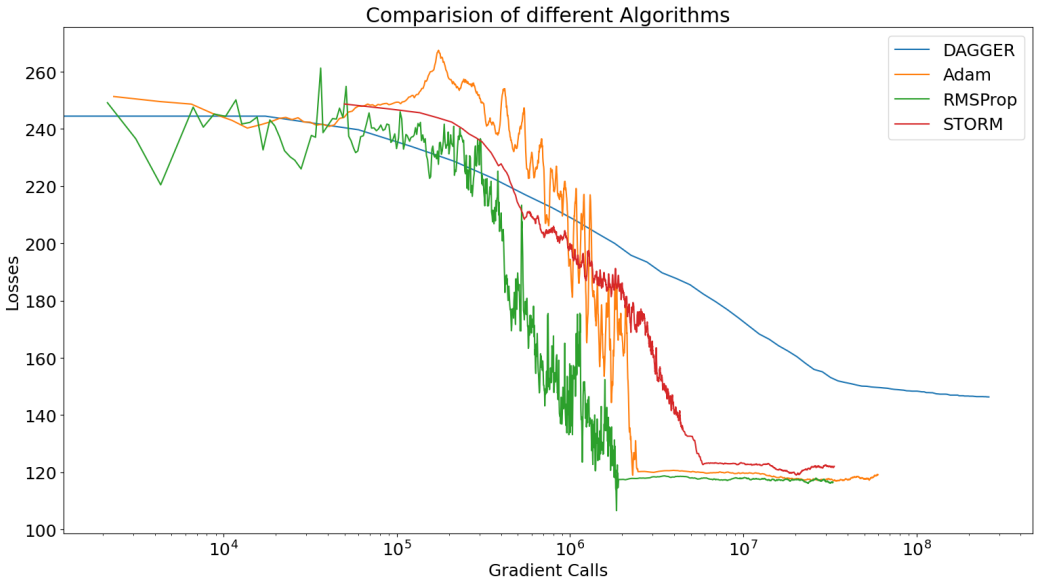


FIGURE 4.8: Loss in CartPole Environment

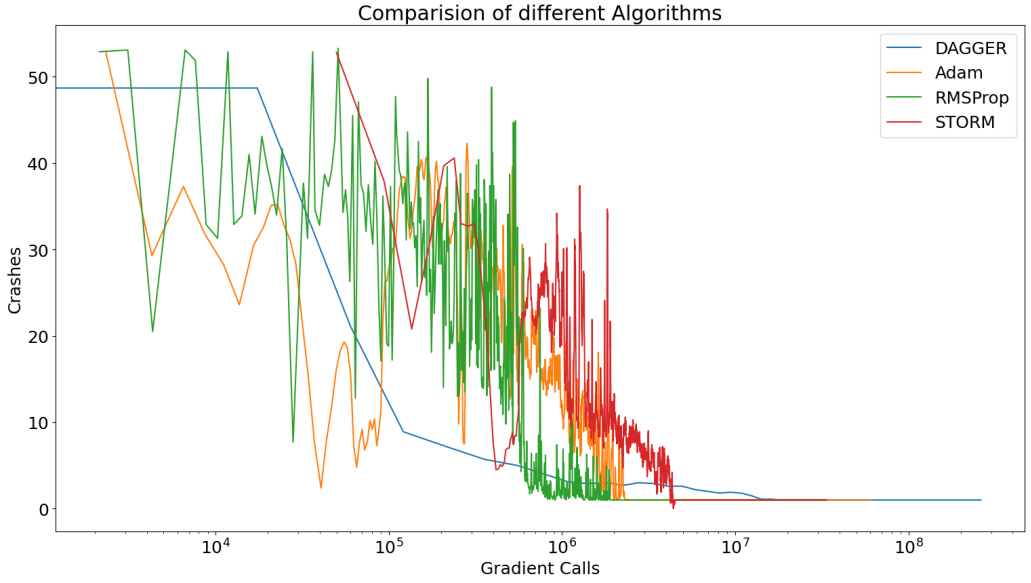


FIGURE 4.9: Crashes in CartPole Environment

In Fig. 4.10 we plot the reward measures. As the number of gradient calls increase the rewards keep on increasing until the maximum episode reward of 500. The DAGGER algorithm takes significantly more number of gradient calls to converge to the maximum episode reward. The trend is similar to the other two plots.

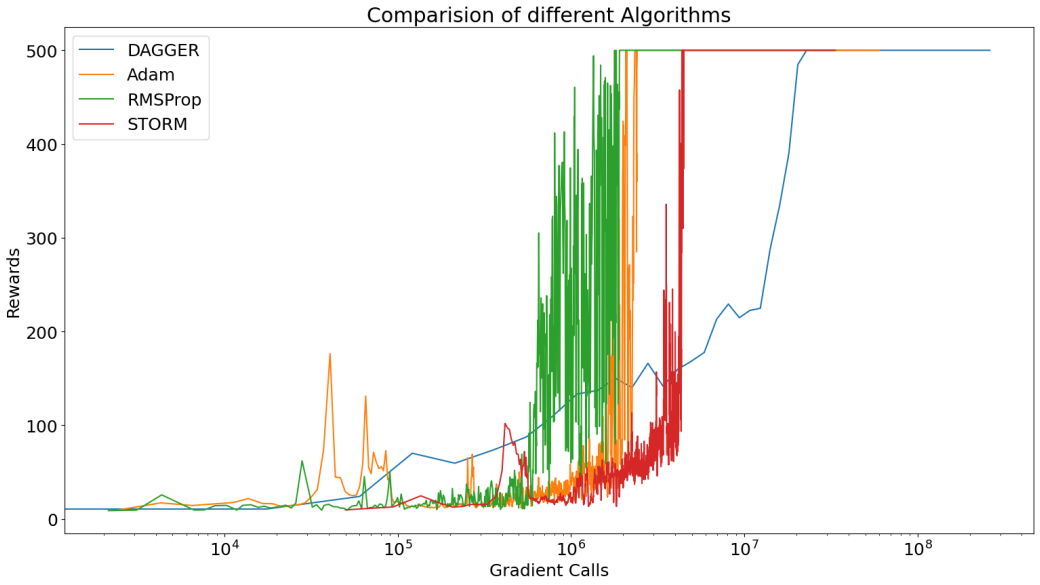


FIGURE 4.10: Rewards in CartPole Environment

In Table 4.3 we plot the number of gradient calls made by each of the algorithms. DAGGER requires the maximum number of gradient calls to converge. In Table 4.4 show the runtime for the various algorithms. The DAGGER algorithm, as expected, has a significantly longer execution time than the other methods. The complexity of the DAGGER algorithm grows with each iteration as the size of the dataset grows. The proposed algorithms have significantly shorter execution times without sacrificing their performance.

Algorithm	No. of gradient calls
STORM	4472049± 11031
RMSProp	1903548 ± 9866
DAGGER	31905600 ± 4263
Adam	2467044 ± 11636

TABLE 4.3: No of gradient calls to converge for CartPole.

Algorithm	Runtime in seconds
STORM	8233.7
RMSProp	3328.4
DAGGER	39340.1
Adam	4717.6

TABLE 4.4: Runtime for CartPole.

Chapter 5

Conclusions and future work

In this thesis we have explored imitation learning from an optimization perspective. We derived a stochastic gradient corresponding to the objective and implemented the SGD, STORM, Adam and RMSProp algorithms for IL. We also compared the performance of these algorithms against one of the most widely used active IL algorithms DAGGER. Using the stochastic optimization techniques along with variance reduction we were able to reduce the computational bottleneck of the current IL algorithms allowing their implementation for more computationally demanding real-world applications. We validated the proposed algorithms for the Obstacle avoidance and the CartPole environments. In addition we looked at IL from the performative prediction viewpoint and showed that the proposed algorithms converge to a performatively optimal point unlike the DAGGER algorithm which converges to a performatively stable point. We also looked at the a non-convex non-smooth optimization algorithm for IL, which helps us understanding the theoretical guarantees when the optimization problem is non-smooth.

As a direction of future work, the algorithms can be implemented for a wide range of environments to understand their effectiveness across different domains. In the non-convex non-smooth optimization we saw the results for just the OGD algorithm. Different types of online leanring algorithms can be implemented along with online-to-non-convex conversion technique.

Bibliography

[1] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 661–668, PMLR, 13–15 May 2010.

[2] 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* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 627–635, PMLR, 11–13 Apr 2011.

[3] A. Attia and S. Dayan, "Global overview of imitation learning," *arXiv preprint arXiv:1801.06503*, 2018.

[4] H. Daumé, J. Langford, and D. Marcu, "Search-based structured prediction," *Machine learning*, vol. 75, pp. 297–325, 2009.

[5] K. Judah, A. P. Fern, and T. G. Dietterich, "Active imitation learning via reduction to iid active learning," in *2012 AAAI Fall Symposium Series*, 2012.

[6] N. Cesa-Bianchi, A. Conconi, and C. Gentile, "On the generalization ability of online learning algorithms," *IEEE Transactions on Information Theory*, vol. 50, no. 9, pp. 2050–2057, 2004.

[7] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Machine Learning*, vol. 69, no. 2-3, pp. 169–192, 2007.

[8] S. Shalev-shwartz and S. M. Kakade, "Mind the duality gap: Logarithmic regret algorithms for online optimization," in *Advances in Neural Information Processing Systems* (D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds.), vol. 21, Curran Associates, Inc., 2008.

[9] H. He, J. Eisner, and H. Daume, "Imitation learning by coaching," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[10] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems* (D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds.), vol. 29, Curran Associates, Inc., 2016.

[11] J. Quinonero-Candela, *Dataset Shift in Machine Learning*. Neural Information Processing series, MIT Press, 2009.

[12] P. L. Bartlett, "Learning with a slowly changing distribution," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, (New York, NY, USA), p. 243–252, Association for Computing Machinery, 1992.

[13] P. L. Bartlett, S. Ben-David, and S. R. Kulkarni, "Learning changing concepts by exploiting the structure of change," *Machine Learning*, vol. 41, pp. 153–174, 2000.

[14] A. Kuh, T. Petsche, and R. Rivest, "Learning time-varying concepts," *Advances in neural information processing systems*, vol. 3, 1990.

[15] J. Perdomo, T. Zrnic, C. Mendler-Dünner, and M. Hardt, "Performative prediction," in *Proceedings of the 37th International Conference on Machine Learning* (H. D. III and A. Singh, eds.), vol. 119 of *Proceedings of Machine Learning Research*, pp. 7599–7609, PMLR, 13–18 Jul 2020.

[16] C. Mendler-Dünner, J. Perdomo, T. Zrnic, and M. Hardt, "Stochastic optimization for performative prediction," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 4929–4939, Curran Associates, Inc., 2020.

[17] D. Dmitriy and X. Lin, "Stochastic optimization with decision-dependent distributions," *arXiv preprint arXiv:2011.11173*, 2020.

[18] J. Cutler, M. Díaz, and D. Drusvyatskiy, "Stochastic approximation with decision-dependent distributions: asymptotic normality and optimality," *arXiv preprint arXiv:2207.04173*, 2022.

[19] G. Brown, S. Hod, and I. Kalemaj, "Performative prediction in a stateful world," in *International Conference on Artificial Intelligence and Statistics*, pp. 6045–6061, PMLR, 2022.

[20] Z. Izzo, L. Ying, and J. Zou, "How to learn when data reacts to your model: performative gradient descent," in *International Conference on Machine Learning*, pp. 4641–4650, PMLR, 2021.

[21] S. Ghadimi and G. Lan, "Stochastic first-and zeroth-order methods for nonconvex stochastic programming," *SIAM Journal on Optimization*, vol. 23, no. 4, pp. 2341–2368, 2013.

[22] X. Li and F. Orabona, "On the convergence of stochastic gradient descent with adaptive stepsizes," in *The 22nd international conference on artificial intelligence and statistics*, pp. 983–992, PMLR, 2019.

[23] R. Ward, X. Wu, and L. Bottou, "AdaGrad stepsizes: Sharp convergence over nonconvex landscapes," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 6677–6686, PMLR, 09–15 Jun 2019.

[24] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *International Conference on Learning Representations*, 2018.

[25] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[27] T. Tieleman, G. Hinton, *et al.*, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[28] Z. Allen-Zhu and E. Hazan, "Variance reduction for faster non-convex optimization," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 699–707, PMLR, 20–22 Jun 2016.

[29] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. Smola, "Stochastic variance reduction for nonconvex optimization," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 314–323, PMLR, 20–22 Jun 2016.

[30] L. Lei, C. Ju, J. Chen, and M. I. Jordan, "Non-convex finite-sum optimization via scsg methods," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[31] C. Fang, C. J. Li, Z. Lin, and T. Zhang, "Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[32] D. Zhou, P. Xu, and Q. Gu, "Stochastic nested variance reduction for nonconvex optimization," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[33] A. Cutkosky and F. Orabona, "Momentum-based variance reduction in non-convex sgd," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[34] K. Levy, A. Kavis, and V. Cevher, "Storm+: Fully adaptive sgd with recursive momentum for nonconvex optimization," in *Advances in Neural Information Processing Systems* (M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), vol. 34, pp. 20571–20582, Curran Associates, Inc., 2021.

[35] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Foundations and Trends in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[38] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: a system for large-scale machine learning.," in *Osdi*, vol. 16, pp. 265–283, Savannah, GA, USA, 2016.

[39] A. Cutkosky, H. Mehta, and F. Orabona, "Optimal stochastic non-smooth non-convex optimization through online-to-non-convex conversion," *arXiv preprint arXiv:2302.03775*, 2023.

[40] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, p. 928–935, AAAI Press, 2003.

[41] P. S. N. Jyotish, Y. Goel, A. V. S. S. B. Kumar, and K. M. Krishna, "Ivo: Inverse velocity obstacles for real time navigation," *International Journal of Plasticity*, vol. 66, pp. 31–45, 2015.