POLITECNICO DI TORINO

Master Degree course in Mechatronic Engineering

Master Degree Thesis

Imitation Learning for Autonomous Highway Merging with Safety Guarantees

Advisor

prof. Andrea Giuseppe Bottino

Candidate

Eleonora D'Alessandro

October 2020

I would like to dedicate this thesis to all those that have contributed, even only in part, to its realization.

My most sincere thanks go to my advisors, Shuo Han and Milos Zefran, that with their constant and diligent support shaped my critical thinking and analysis skills, while guiding this incredibly formative path from the conception of the idea to its realization.

I also want to thank my family, that supported me during my international journey and always believed in my potential. A special dedication goes to my mum, who taught me that tenacity and perseverance can get me everywhere.

I am especially grateful for having a superlative Director of International Programs, Jenna Stephens, whose guidance and advice has been of crucial importance during uncertain times.

Finally, I owe a lot to my friends, especially my Italian friends that joined me in this American adventure. Our mutual support has been a point of reference being so far from home, especially during unexpected and unprecedented circumstances.

EDA

TABLE OF CONTENTS

CHAPTER **PAGE**

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF FIGURES

List of Algorithms

SUMMARY

Thanks to the innovations in Artificial Intelligence, the autonomous driving field experienced an enormous growth in recent years, opening the road to a new safe and efficient way of conceiving transportation. One of the most challenging aspects is designing a driverless car able to safely navigate the highway. A particularly critical maneuver is merging in the highway traffic coming from an on-ramp, which will be the focus of this thesis.

The described task is an highly interactive process, that requires an advanced level of cooperation between drivers. In similar cases, machine learning techniques have demonstrated to be more efficient than manually designed rule-based approaches. In particular, in this work we consider the Imitation Learning (IL) approach, whose objective is to learn how to perform a task by imitating the demonstrations of an expert. The result is a policy that the agent can follow to perform as similar as possible to the expert.

Behavioral Cloning (BC) and Inverse Reinforcement Learning (IRL) are the two main approaches in Imitation Learning. The former uses supervised learning to find a policy that imitates the expert. This method is simple and efficient but it suffers from cascading error. The latter, instead, resorts to methods of solving an MDP when the reward function is unknown. This approach allows to learn a reward function, that constitutes a transferable representation of the desired behavior, but it is computationally more expensive and requires the knowledge of the model dynamics.

The imitation technique proposed in this thesis builds on the Behavioral Cloning approach and

augments it with a safety filter that covers the cascading error issue. This approach preserves the benefits of using a BC-based technique, while overcoming its limitations with an additional controller to ensure safety.

In this work, we explore the state-of-the-art of Imitation Learning, present our method and analyze the results of applying it to the highway merging scenario.

CHAPTER 1

INTRODUCTION

The research in the Autonomous Driving field has experienced an enormous growth in recent years, given the importance of the impact it can have on our society. The National Highway Traffic Safety Administration reports that "the vast majority of traffic crashes are caused by human error. A landmark study by Indiana University (Treat et al, 1979) found that human factors caused or contributed to 93% of the crashes investigated". The perspective of a fully automated car is extremely promising since it has the potential of vastly reducing the number of crashes that happen every year and the relative fatalities, eliminating what constitutes the main cause of car accidents, that is distracted, tired or hazardous human drivers.

Despite the important progresses accomplished in this area, the open challenges are still numerous. The difficulties arise from the fact that the task of driving involves the observance of a number of traffic rules but also various unwritten laws that regulate the interaction between drivers. The large number of vehicles that populate our roads makes the driving operation extremely interactive and based on the cooperation between drivers. An example of particularly interactive driving task is the highway merging maneuver, considered in this thesis. Coming from the highway on-ramp with the intent of joining the traffic in the right-most lane, a vehicle faces a series of challenging operations. The observance of the other vehicles' behavior is crucial for a proper execution of the maneuver as it provides information about important modifications that other cars can operate on their own driving style, such as accelerate, decelerate etc.,

that are representative of their intention of yielding to the merging car or proceeding straight. The entire process is based on understanding other's intention and adopt a responsive behavior. It involves a sequence of decision problems that human drivers, especially experienced ones, can solve extremely easy, but it is remarkably difficult to design an algorithm that can properly and successfully encode this human-based decision process.

The recent diffusion of Artificial Intelligence (AI) and Machine Learning (ML) determined the development of incredibly successfully technologies in the automation industry. Accurately predicting and simulating the behavior of a human driver is a crucial factor for the success of intelligent vehicles. As a sub-field of Machine Learning, Reinforcement Learning (RL) has been used in this sense to achieve important results. Differently from the other types of ML (Supervised and Unsupervised Learning), in RL the learning process is carried out through a trial-and-error procedure, where the training information are evaluations rather than errors.

1.1 Motivation

Although RL techniques have demonstrated remarkable successes in the autonomous driving field, they usually rely on some rewarding system that can properly support the trial-and-error learning by providing rewards or penalties coherently with the specific task. Specifically, they require specifying a cost function that encodes the desired behavior.

However, manually specifying such cost function in a complex application environment often leads to sub-optimal solutions. In fact, a manually designed cost function is unlikely to capture all the aspects of a complex decision-making and planning procedure. Furthermore, this approach is not sustainable in the long term as it requires the continuous intervention of engineers

for improvements and maintenance.

The difficulty of providing specifications that encode the wanted behavior clashes with the ease of performing the same behavior as a human driver, giving rise to the idea of using human driver demonstrations to learn how to perform the task with a learning strategy based on the imitation of human examples. This procedure, called *Imitation Learning*, is the main topic of this thesis, with a particular focus on its application to the highway merging maneuver.

1.2 Overview

The presentation of this thesis is organized as follows. In Chapter 2 we give an introduction to Markov Decision Processes and their use to mathematically formulate Reinforcement Learning problems. In Chapter 3 we describe the problem of Imitation Learning and provide an overview of the state-of-the-art methods. Chapter 4 is dedicated to the analysis of the episodic MDP setting and to a first formulation of the method proposed to ensure safety while performing the imitation operation. Chapter 5 contains the final formulation of the proposed algorithm and the experiment conducted applying the described method to the highway merging scenario. We conclude with final considerations and discussions given in Chapter 6.

CHAPTER 2

MARKOV DECISION PROCESS AND REINFORCEMENT LEARNING

Reinforcement Learning is a Machine Learning paradigm for automated decision-making processes. The fundamental principle of Reinforcement Learning has its roots in a learning theory method, called *Learning by Trial and Error*, according to which learning is the result of the experience acquired by trying a number of different methods and learning from the mistakes. The psychologist Edward Lee Thorndike, who was the initiator of the Trial and Error learning theory, studied the phenomenon in an experiment conducted on cats placed in puzzle boxes and motivated to get out of the box by the smell of the food located outside the box. In the first reiterations of the experiment, the cats tried random movements in the attempt to find a way out. After multiple times, however, the cats needed less and less time to accomplish the task, until they were able to release themselves almost immediately. In his book "Animal Intelligence: Experimental Studies" (1911), Thorndike formulated the *Law of Effect*, also known as *Principle of Reinforcement*, as follows:

"Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will me more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with the situation weakened, so that when it recurs, they will be less likely to occur."

The Thorndike's Law of Effect describes the fundamental principle on the basis of Reinforcement Learning: the learner is more likely to repeat behaviors that previously led to a positive outcome and less likely to repeat behaviors that led to a negative outcome.

In other words, during the learning process, a positive behavior is *reinforced* with a positive compensation, so that the learner is trained to act in the way that is the most beneficial for the accomplishment of the desired task.

A Reinforcement Learning algorithm is the computational approach to learning from interaction. For a formal and rigorous description of the learning from interaction problem, the Reinforcement Learning paradigm strongly relies on the concept of Markov Decision Process (MDP), which provides the mathematical framework for modeling the interaction between the learner and its environment.

Let us introduce the terminology commonly adopted in MDP formulations. The learner is called *agent*: it is the entity entitled to both understand the most beneficial behaviors for the final goal and actively taking decisions to achieve that goal. When the agent executes a decision, it is said to take an *action*. Everything that is outside the agent is defined as *environment*: the interaction between agent and environment is the key element of the learning process. The reinforcement signal that the agent receives when it takes an appropriate action is called *reward*. An action with a higher assigned reward will have a higher probability to be taken again in the future. In this thesis, the terms *reward* and *cost* will be used interchangeably, with the *cost* being the opposite of the reward.

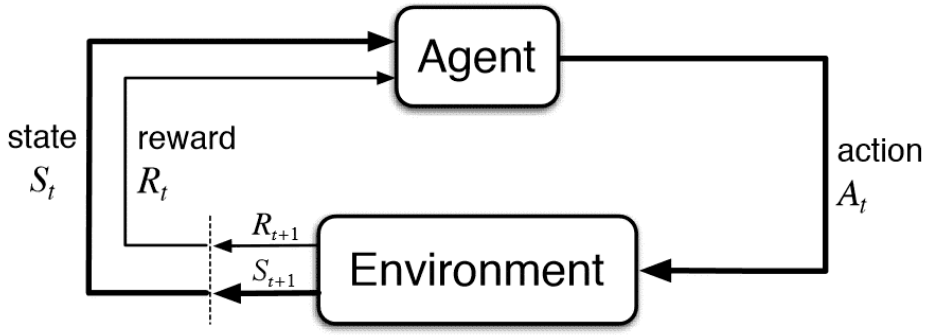


Figure 1: Agent-environment interaction

The interaction agent-environment, illustrated in Figure 1, shapes the learning process. It takes place whenever the agent takes an action, on the basis of the current *state* of the environment (or, more precisely, on the basis of the representation, perceived by the agent, of the current state of the environment). The result of the agent's action is twofold:

- the agent receives an immediate reward based on the quality of its action
- the environment evolves into another state.

In this new situation, the agent takes another action that again results in a reward and a new state of the environment. The iteration of this process gives rise to a sequence of states, actions and rewards, called a *trajectory*.

2.1 Definitions

A Markov Decision Process (MDP) is a formal framing of the problem of an agent learning to perform a goal by interacting with the environment. It provides a mathematical tool to describe the process of the agent learning how to take actions in the environment in order to maximize the total reward.

2.1.1 The Markov Property

In order to properly define a Markov Decision Process, it is necessary to define and analyze its main characteristic, namely the Markov property.
Intuitively, the Markov property refers to the fact that the next state s_{t+1} depends exclusively on the previous state s_t and has no dependency on the past evolution of states preceding s_t. This concept can be formalized as follows.

Definition 2.1.1. A state s_{t+1} is Markov if and only

$$\mathbb{P}(s_{t+1}|s_t) = \mathbb{P}(s_{t+1}|s_1, ..., s_t). \tag{2.1}$$

If the Markov property is valid, the state s_t already captures all the information of the past states that is relevant for the transition to the next state s_{t+1}.

2.1.2 Markov Process

A stochastic process (i.e. collection of random states indexed by time) is said to be a Markov Process when the Markov property holds for all its states. Therefore, a Markov process (or Markov chain) is a memoryless stochastic process.

Definition 2.1.2. A Markov Process is a tuple $\langle \mathcal{S}, P \rangle$:

- $\mathcal{S}$ is a non-empty set of states (*state space*)
- P is a state transition probability function $P : \mathcal{S} \times \mathcal{S} \to [0, 1]$.

The state space $\mathcal{S}$ is the set of all possible system states. Although it can be either finite or infinite, we restrict our attention to finite or countably infinite state spaces for now.
The state transition probability function P defines the system dynamics. $P(s'|s)$, sometimes also denoted as $P_{ss'}$, is the probability of transitioning to state *s'* being in state *s*.

$$P_{ss'} = P(s'|s) = \mathbb{P}(s_{t+1} = s'|s_t = s) \tag{2.2}$$

2.1.3 Markov Reward Process

A Markov Reward Process is a Markov Process where a reward value is assigned to every state. This enables the evaluation of the quality of every step along the trajectory, which is crucial for the decision making process.

Definition 2.1.3. A Markov Reward Process is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:

- $\mathcal{S}$ is a non-empty set of states (*state space*)
- P is a state transition probability function $P : \mathcal{S} \times \mathcal{S} \to [0, 1]$
- R is the reward function $R : \mathcal{S} \to \mathbb{R}$
- $\gamma \in [0, 1]$ is the discount factor.

The reward function $R : \mathcal{S} \rightarrow \mathbb{R}$ is the key element for evaluating the agent performance. $R(s)$ denotes the immediate reward associated with state s and it is given by the expected value of the reward given the current state s.

$$R(s) = \mathbb{E}(R_{t+1}|s_t = s) \tag{2.3}$$

The discount $\gamma \in [0, 1]$ is a weighting factor that determines the relative importance of immediate and future rewards. It plays an important role in the definition of discounted return given in the following.

In many application tasks, instead of maximizing the immediate reward collected at each time step, it is more interesting to optimize the cumulative reward collected along the trajectory. Following this optimization criterion, the agent might prefer not to take the action associated to the maximum possible immediate reward if this choice enables the achievement of a higher future reward.

The most straightforward formalization of this concept is to choose as maximization objective the sum of all the rewards collected along the way. This quantity is called *return*, denoted by G_t and defined as following:

$$G_t = \sum_{k=0}^{T} R_{t+k+1} \tag{2.4}$$

where T is the final time step.

In many cases, however, a reward collected earlier has more importance, for the purpose of the task performance, than the same reward collected many time steps later. In order to give more consideration to earlier collected rewards, the discount factor is introduced to define the *discounted return* as follows:

$$G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{2.5}$$

The discount factor γ determines the importance given to rewards collected at different time steps: a reward collected at time $t+k$ counts as $1/\gamma^{(k-1)}$ of the same reward collected at time t.

For applications that extend over a long time horizon, T can be considered to be infinite. In this case, to prevent the infinite sum in (Equation 2.5) from going to infinite, the discount factor must be chosen within the range $0 \leq \gamma < 1$. When $T < \infty$, instead, γ can also be equal to 1.

For $\gamma = 0$ the agent is said to be "myopic" as it only acts with the purpose of maximize the immediate reward.

For $\gamma = 1$ there is no difference between the same reward collected at different time steps.

2.1.4 Markov Decision Process

A Markov Decision Process is a Markov Reward Process where the next step is determined by an agent actively taking decisions.

Definition 2.1.4. A Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$:

- $\mathcal{S}$ is a non-empty set of states (*state space*)

- $\mathcal{A}$ is a non-empty set of actions (*action space*)
- P is a state transition probability function $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$
- R is the reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- $\gamma \in [0, 1]$ is the discount factor.

The action space $\mathcal{A}$ is the set of all the possible actions the agent can take. We consider either finite or countably infinite action spaces.
The state transition probability $P(s'|s, a)$, sometimes also denoted as $P^a_{ss'}$, is the probability of transitioning to state *s'* being in state s and taking action a.

$$P^a_{ss'} = P(s'|s, a) = \mathbb{P}(s_{t+1} = s'|s_t = s, a_t = a) \tag{2.6}$$

$R(s, a)$ denotes the immediate reward associated with taking action a being in state s and it is given by the expected value of the reward given the current state and action:

$$R(s, a) = \mathbb{E}(R_{t+1}|s_t = s, a_t = a) \tag{2.7}$$

2.2 Policy

The agent's behavior in a given state and time step is defined by the agent's policy, namely the decision-making rule that tells the agent what action to take for every possible configuration. In its more general stochastic version, a policy is a mapping from the perceived state of the environment to the probabilities of selecting every possible action. In particular, indicating

with π the agent's policy, $\pi(a|s)$ is the probability of choosing action $a_t = a$ being in state $s_t = s$ and following policy π.

A deterministic policy is a special case of stochastic policy that directly maps states into actions. In the deterministic setting, indicating with π the agent's policy, $\pi(s)$ is the action the agent will take when it is in state $s_t = s$ and follows policy π.

A policy π can be either stationary (time-invariant) or non-stationary (time-variant): while the former depends only on the current state of the environment, the latter also considers the particular time step along the trajectory. As a deterministic policy is a particular case of stochastic policy, also a stationary policy is a particular case of time-variant policy, as illustrated in Figure 2.

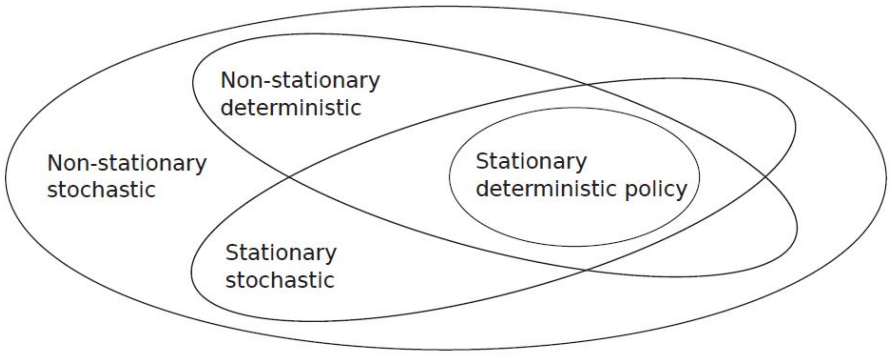


Figure 2: Relationship between different classes of policies

The goal of a Reinforcement Learning algorithm is to find the maps between state and action spaces that maximize the (discounted) return associated with following policy π.

2.3 Value Functions

In order to evaluate a policy, different value functions are needed to assess the quality of being in a particular state and taking a particular action in that state. In this context, the "quality" is defined in terms of expected return.

State-value function

Given a policy π, the state-value function $V^\pi : \mathcal{S} \to \mathbb{R}$ is defined for every state $s \in \mathcal{S}$ and gives a measure of the quality of each state by determining the expected return starting from state s and following policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \middle| s_0 = s \right] = \mathbb{E}_\pi[G_t | s_t = s], \qquad \text{for all } s \in \mathcal{S} \tag{2.8}$$

where $\mathbb{E}_\pi$ denotes the expected value of the random variable in the argument, given that the agent follows policy π.

The state-value function is often simply called value function.

State-action value function

The value function, only considering the overall quality of the state itself, is not sufficient to support the decision process, i.e. does not discriminate between the quality of the possible

actions the agent can take each state. At this purpose, it is useful to introduce the state-action value function, or Q-function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as follows:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \middle| s_0 = s, a_0 = a \right] = \mathbb{E}_\pi[G_t | s_t = s, a_t = a], \text{ for all } s \in \mathcal{S}, a \in \mathcal{A} \tag{2.9}$$

The quantity $Q^\pi(s, a)$ denotes the value of taking action a in state s under policy π and it is defined as the expected return starting from state s, taking action a and following policy π thereafter.

Providing information about the long-term return that comes from taking a particular action in the present state, the state-action value function helps the decision-making process giving an insight about the effect of every action on the overall trajectory.

The value function can be obtained from the state-action value function simply averaging over all the possible actions $a \in \mathcal{A}$:

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s)} [Q^\pi(s, a)] \tag{2.10}$$

2.4 Optimality

Given an MDP, the goal of a Reinforcement Learning algorithm is to find a policy that performs the best among all the possible policies for the same MDP. In order to compare different policies and select the best one, an ordering criterion over policies is needed.

Definition 2.4.1. A policy π is defined to be better or equal to a policy π' if the value function of π is greater than or equal to the value function of π', for all possible states $s \in \mathcal{S}$:

$$\pi \geq \pi' \iff V^{\pi}(s) \geq V^{\pi'}(s) \quad \text{for all } s \in \mathcal{S} \tag{2.11}$$

In an MDP there is always at least one policy that is better or equal to all the other policies for the same MDP, and it is called *optimal policy*, denoted by π^*.
The state-value function associated with the optimal policy is called *optimal state-value function* and it is defined as follows:

Definition 2.4.2. The optimal state-value function $V^* : \mathcal{S} \to \mathbb{R}$ for a given MDP is the one, among all the possible state-value functions, that achieves the maximum possible value for all states $s \in \mathcal{S}$:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all } s \in \mathcal{S} \tag{2.12}$$

Similarly, the *optimal state-action value function* can be defined as follows:

Definition 2.4.3. The optimal state-action value function $Q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for a given MDP is the one, among all the possible state-action value functions, that achieves the maximum possible value for all state-action pairs $s \in \mathcal{S}$:

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) \quad \text{for all } s \in \mathcal{S}, a \in \mathcal{A} \tag{2.13}$$

For a policy to be optimal, it must be better than all the other policies, according to Definition 2.4.1. This is verified when its value is equal to the optimal value V^* in every state:

$$V^{\pi^*}(s) = V^*(s) \quad \text{for all } s \in \mathcal{S} \tag{2.14}$$

The optimal policy may not be unique, but all the optimal policies for a given MDP will share the same optimal value function $V^*(s)$ as well as the same optimal state-action value function $Q^*(s, a)$.

Finally, the optimal policy is the one that maximize the expected cumulative reward $J(\pi)$:

$$J(\pi) = \mathbb{E}_{s\sim d_0}\left[V^{\pi}(s)\right] \tag{2.15}$$

where d_0 denotes the initial state distribution.

$$\pi^* \in \arg\max_{\pi} J(\pi) \tag{2.16}$$

2.5 Solutions for MDPs

The goal of a Reinforcement Learning algorithm is to find the optimal policy π^* for a given MDP, that coincides with solving the MDP. The most efficient way of solving an MDP is by using *dynamic programming (DP)* techniques. The computational advantage of using DP comes from the fact that a complex optimization problem can be solved iteratively by breaking down the complex problem into smaller and simpler subproblems until the solution of the innermost

subproblem is straightforward. In the case of an MDP, Dynamic Programming can be used exploiting the recursive structure that defines the relationship between the value of a state and the value of the previous state.

$$\begin{aligned} V^{\pi}(s_t) &= \mathbb{E}_{\pi}[G_t|s_t = s] \\ &= \mathbb{E}_{\pi}\left[\sum_{t=0}^{T}\gamma^t R(s_t, a_t)\middle| s_0 = s_t\right] \\ &= R(s_t, \pi(s_t)) + \gamma\mathbb{E}_{\pi}\left[\sum_{t=0}^{T}\gamma^t R(s_{t+1}, a_{t+1})\middle| s_0 = s_t\right] \\ &= R(s_t, \pi(s_t)) + \gamma\sum_{s'\in\mathcal{S}} P(s_{t+1}|s_t, \pi(s_t))\mathbb{E}_{\pi}\left[\sum_{t=0}^{T}\gamma^t R(s_{t+1}, a_{t+1})\middle| s_1 = s_{t+1}\right] \\ &= R(s_t, \pi(s_t)) + \gamma\sum_{s'\in\mathcal{S}} P(s_{t+1}|s_t, \pi(s_t))V^{\pi}(s_{t+1}) \end{aligned} \tag{2.17}$$

In general,

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma\sum_{s'\in\mathcal{S}} P(s'|s, \pi(s))V^{\pi}(s') \tag{2.18}$$

(Equation 2.18) is called *Bellman equation*. It express the recursive relationship between $V^{\pi}(s)$ and $V^{\pi}(s')$, where s' is the state immediately following s. Given its recursive structure, it allows the application of dynamic programming algorithms for solving any MDP. All DP methods present the two following components:

- **Policy evaluation**. The evaluation of a policy π refers to the computation of the value function V^{π}. It can be done with the direct method simply solving the linear system of equations given by the Bellman equation, but this approach can be computational

expensive dealing with large state spaces. Often it is more suitable to implement an iterative solution solving, at each iteration k, the following update rule:

$$V^{\pi}_{k+1}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^{\pi}_{k}(s') \tag{2.19}$$

The sequence V^{π}_{k} can be proved to converge to V^{π} as $k \to \infty$

- **Policy improvement**. Policy improvement refers to the process of generating a new policy π' improved with respect to the original policy π by taking the policy that is optimal with respect of the Q-function calculated according to the original policy π:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q^{\pi}(s, a) \quad \text{for all } s \in \mathcal{S} \tag{2.20}$$

Policy evaluation and policy improvement are all it is needed to describe the two widely used algorithms for solving any MDP: *value iteration* and *policy iteration*.

2.5.1 Policy Iteration

Policy iteration algorithm first initializes arbitrarily both value function and policy, foll all possible states, and then applies, at each iteration, the policy evaluation phase followed by the policy improvement phase.

The policy iteration method is detailed in Algorithm 1.

Algorithm 1: Policy iteration

Input: R, γ, P

Output: optimal value function V^* and relative policy π^*

1 Initialize $V_0(s)$ and π_0 arbitrarily for all states $s \in \mathcal{S}$, $k \leftarrow 0$

2 **while** π_k *not converged* **do**

  while V_k*not converged* **do**

    $k+ = 1$

    for *each state* s **do**

      $V_{k+1}(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V_k(s')$

    end

  end

  for *each state* s **do**

    $\pi_{k+1}(s) \leftarrow \text{argmax}_{a \in A}[R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)(V_{k+1}(s'))]$

  end

end

3 Return $\pi^* \leftarrow \pi_k$, $V^* \leftarrow V_k$

After the initialization of $V_0(s)$ and π_0 (step 1), step 2 consists in the iterative application, until convergence to the optimal policy, of the sequence of policy evaluation and policy improvement. Every iteration of step 2 can be shown to perform a strictly positive improvement over the original policy.

In order to reduce the computational cost given by completely solving both policy evaluation and policy improvement at each iteration, one can choose to stop the algorithm when no

significant improvement is achieved between two consecutive iteration, instead of waiting until exact convergence.

2.5.2 Value Iteration

While policy iteration updates the optimal policy at each step, always passing also through the update of the optimal value function, the value function only iterates on the value function itself, until it achieves its optimal value. Only when the optimal value function is calculated, the optimal policy is extracted from it in a single step. The value iteration method is detailed in Algorithm 2.

Algorithm 2: Value iteration

Input: R, γ, P

Output: optimal value function V^* and relative policy π^*

1 Initialize $V_0(s)$ arbitrarily for all states $s \in \mathcal{S}$, $k \leftarrow 0$

2 **while** V_k *not converged* **do**

$k+ = 1$

for *each state* s **do**

$V_{k+1}(s) \leftarrow \max_{a \in A}[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)(V_k(s'))]$

end

end

3 **for** *each state* s **do**

$\pi^*(s) = \text{argmax}_{a \in A}[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)(V_{k+1}(s'))]$

end

4 Return π^*, $V^* \leftarrow V_k$

After the initialization of $V_0(s)$ (step 1), step 2 consists in the computation of the optimal value function iterating over the Bellman equation. Once V^* is calculated, the optimal policy is extracted from it as in step 3 of Algorithm 1.

With respect to Policy iteration, Value iteration is computationally less demanding but it usually requires more iterations.

CHAPTER 3

IMITATION LEARNING

Given an MDP $\mathcal{M} =< \mathcal{S}, \mathcal{A}, \mathsf{P}, \mathsf{R}, \gamma >$, it can be solved exactly with the methods detailed in chapter 2.5. The result will be a policy that is guaranteed to be optimal: following such policy, the expert will perform the given task in the optimal way, that is, maximizing the expected return. In order to guarantee such results, the MDP must be provided with an accurate description of the task to be executed. The element that encode the goal description is the reward function, that gives information about how beneficial it is in the long-term to take a specific action in a specific state.

When the reward function changes, the agent's behavior will change as well. Therefore, it must be specified properly and it must include all the relevant information about every aspect of the task to be executed. In a large variety of applications (e.g. Arari games), the reward can be specified manually and it will suffice to generate the expected behavior. In some cases, a trial-and-error process is needed to find the reward that is the best possible representation of the task.

However, in safety-critical and complex environments, manually design the reward function is not straightforward. Specifically considering the task of driving on the highway, it is not immediate to manually select all the relevant factors that must be captured by the reward. Examples of such factors may be the distance from the surrounding (front, back and side) cars, the distance from the guard-rail, adapting the speed to the traffic situation, observe speed limits,

prefer a specific lane depending on the situation, guarantee safe and smooth maneuvers, etc. Moreover, even succeeding in the task of carefully selecting all the relevant factors, the most difficult and risky part will be to determine the optimal trade-off between the weighting terms of all the different factors.

For these reasons, Imitation Learning techniques have been widely considered in recent literature and has been applied to a large number of fields. The idea behind Imitation Learning is to determine the optimal policy without knowing the reward function but learning how to perform the task by imitating the demonstrations of the specific task performed by an *expert*. Imitation learning is often referred to as *learning from demonstrations* or *apprenticeship learning*.

In other words, the problem solved by Imitation Learning methods are MDP without the reward function $\mathcal{M}\backslash R$, that is a tuple $\langle \mathcal{S}, \mathcal{A}, P, \gamma \rangle$.

In literature there are an extensive collection of proposed techniques that implement different variation of the learning from demonstration framework. The two main categories are *Behavioral Cloning* (BC) and *Inverse Reinforcement Learning* (IRL).

Behavioral Cloning tries to directly compute a policy that performs a mapping from the state space into the action space. Given a collection of expert demonstrations $\mathcal{D} = \{(s_i, a_i)\}_{i=1}^{N}$, a BC algorithm aims to directly learn what is the best action to take in every state with the purpose of imitating the choices of the expert observed in the demonstration dataset. The direct policy learning process can be done applying supervised learning techniques to the demonstrations in $\mathcal{D}$.

Inverse Reinforcement Learning, instead, has the goal of learning a reward function that repre-

sents the best possible explanation of the expert demonstrations, under the assumption that the expert's policy is optimal (or nearly optimal) with respect an unknown reward function. The concept behind IRL methods is that the reward constitutes a portable and exhaustive description of the desired behavior and it can be used, after being recovered through IRL algorithms, in standard RL frameworks to generate the agent's policy that is optimal with respect to the reward recovered from expert demonstrations.

The choice between BC and IRL depends on whether the desired behavior can be better encoded in the learned policy or in the recovered reward function. Ultimately, every of the two approaches have advantages and disadvantages and the choice of the most appropriate one is strictly related to the characteristics of the specific application.

The purpose of this chapter is to give an overview of the current state-of-the-art of the different Imitation Learning techniques present in literature. Both Behavioral Cloning and Inverse Reinforcement Learning will be considered and positive and negative aspect of both the approaches will be highlighted.

3.1 Behavioral Cloning

Behavioral cloning is one of the main approaches to solve the problem of learning from demonstrations. It does not attempt to recover a reward function that would be an exhaustive representation of the task. On the contrary, since the final goal of any imitation learning method is, eventually, to generate a policy that is able to imitate the expert's behavior, BC skips the reward recovery step and directly tries to learn a policy that maps any state of the environment into an action, in a way that is as coherent as possible with the choices performed by the expert.

The input for every BC algorithm is a set of expert demonstrations $\mathcal{D} = \{(s_i, a_i)\}_{i=1}^{N}$. Given a policy parametrization π_θ, the policy parameters $\theta \in \Theta$ can be selected by solving a supervised learning problem with regression methods.

3.1.1 Loss Functions

Given a set of expert examples to imitate, in order to generate a model that can fit the demonstrations and then be used for further predictions, every regression algorithm needs a loss function $\mathcal{L}$ that quantifies the difference between the expert policy and the policy under the learning process.

Given a loss function $\mathcal{L}$, the optimization objective is given by:

$$\arg\min_{\theta} \mathcal{L}(\pi^{exp}, \pi_\theta) \tag{3.1}$$

where π^{exp} represents the expert policy and π_θ the learned one.

It is worth noticing that the goal of the above optimization is not to find the parameters θ corresponding to a policy that is optimal with respect the performance of the specific task. In fact, the goal of this optimization is purely to imitate the expert. The optimal parameters will be the ones whose policy is optimal with respect to the imitation purpose. For this reasons, expert demonstrations that are exhaustively explicative of the desired behavior are needed for the algorithm to succeed.

The loss function $\mathcal{L}$ can be chosen in many different ways. In the following are illustrated the most common and effective choices.

l_2 loss function

The l_2 loss function (or quadratic loss function) is one of the most commonly used when solving a regression problem. The l_2 loss function between two vectors v_1 and v_2 is formulated as

$$l_2(v_1, v_2) = (v_1 - v_2)^T(v_1 - v_2) \tag{3.2}$$

When the l_2 function is used for quantifying the loss in a regression problem, it is also said to solve a *Least Square regression* problem.
In the context of Behavioral Cloning, it can be used to track the difference between the actions taken by the learned policy and the optimal ones, namely the actions chosen by the expert:

$$l_2(a^{exp}, \pi_\theta(s)) = (a^{exp} - \pi_\theta(s))^T(a^{exp} - \pi_\theta(s)) \tag{3.3}$$

l_1 loss function

The l_1 loss function (or absolute loss function) is another commonly used loss representation in regression problems. The l_1 loss function between two vectors v_1 and v_2 is formulated as

$$l_1(v_1, v_2) = \sum_i |v_{1,i} - v_{1,i}| \tag{3.4}$$

When the l_1 function is used for quantifying the loss in a regression problem, it is also said to solve a *least absolute deviation regression* problem.

As the l_2 loss function, it can be used in the context of Behavioral Cloning to track the difference between the actions taken by the learned policy and those taken by the expert.

Kullback-Leibler Divergence

The Kullback-Leibler (KL) divergence, also known as relative entropy, is one of the most diffuse metrics in mathematical statistics and information theory. It measures how a given probability distribution is different from a second reference probability distribution.

Given two probability distributions P and Q with density functions p and q absolutely continuous with respect to a base measure dx defined on the domain $\mathcal{X}$, the KL divergence between P and Q is given by

$$D_{KL}(P||Q) = \int_{\mathcal{X}} p(x) \log \frac{p(x)}{q(x)} dx \tag{3.5}$$

Notice that the Kullback-Leibler divergence is not a symmetric operator, meaning that $D_{KL}(P||Q) \neq D_{KL}(Q||P)$.

It can be used to compute the difference between two stochastic policies. In particular, in the context of Behavioral Cloning, the difference between the expert policy and the learned policy can be calculated as

$$D_{KL}(\pi^{exp}||\pi_\theta) = \int_{\mathcal{A}} \pi^{exp}(a|s) \log \frac{\pi^{exp}(a|s)}{\pi_\theta(a|s)} da. \tag{3.6}$$

It can be shown that there exist a strict relationship between KL divergence and expected log likelihood:

$$\begin{aligned} D_{KL}(\pi^{exp}||\pi_\theta) &= \int_{\mathcal{A}} \pi^{exp}(a|s) \log \frac{\pi^{exp}(a|s)}{\pi_\theta(a|s)} da \\ &= \mathbb{E}_{a\sim\pi^{exp}}[\log \pi^{exp}(a|s)] - \mathbb{E}_{a\sim\pi^{exp}}[\log \pi_\theta(a|s)] \end{aligned} \tag{3.7}$$

For the purpose of the optimization over the parameters θ, the term $\mathbb{E}_{a\sim\pi^{exp}}[\log \pi^{exp}(a|s)]$ can be neglected. Therefore, the above equation highlights the fact that minimizing the KL divergence is equivalent to maximizing the expected log likelihood of the given demonstration trajectories.

3.1.2 Advantages and Disadvantages

Behavioral Cloning has the advantage of being an extremely simple and efficient technique since it mainly consists in applying supervised learning to a set of expert's trajectories. Moreover, BC eliminates all the ambiguity issues that come from recovering a reward function, that will be detailed in the next chapter. Since the final goal of any imitation learning algorithm is to learn a policy that allows the agent to perform similarly to the expert, the reward function recovery step can be avoided when the resulting optimal policy is an appropriate description of the expert behavior.

On the other hand, BC also has a number of drawbacks.

First of all, a Behavioral Cloning algorithm requires a large set of demonstrations in order to perform properly. In fact, since the only criterion is the correct imitation of the expert, the examples to be imitated must represent exhaustively every aspect of the task to perform.

Secondly, BC suffers from *cascading error*, meaning that a small error in a early stage that only brings a negligible effect in itself, may cascade over time, generating a bigger error that may

lead the agent to a state that differs from all the states observed in the expert demonstrations. Moreover, if the learner visits a state that has never been visited from the expert, it has no guidance of how to behave in that particular situation because it has no data to imitate. This might lead to a catastrophic scenario depending on the application setting.

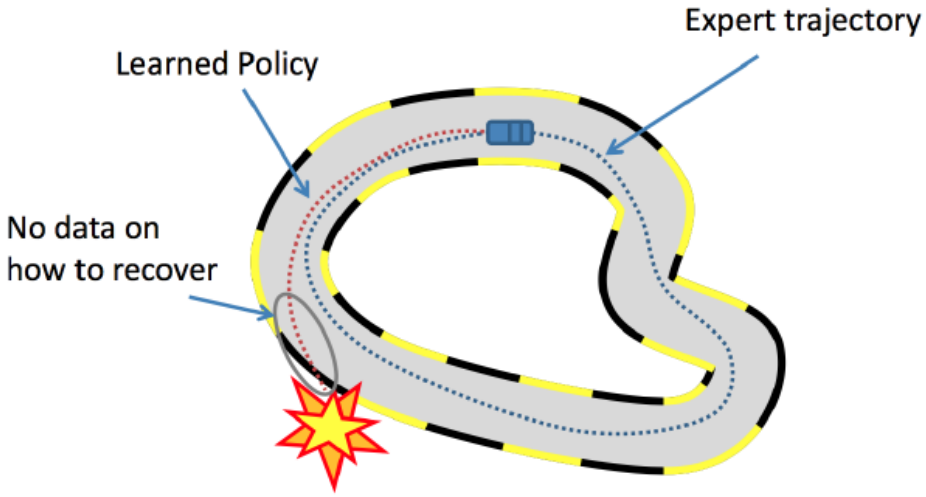


Figure 3: Cascading error that leads to unforeseen states where no expert data are available to recover

An example of the described drawbacks is illustrated in the Figure above. In this example the agent is represented by the car performing the task of driving in the middle of the trail. A small error that makes the car deviate slightly from the expert trajectory can generate a

sequence of cascading errors that add up and drive the car to the side of the track. Since the expert has never visited those corner states, the learner has not information that can exploit to recover and come back to visited and safe states. In such corner states, the behavior is undefined and this can lead, as in this case, to a complete failure.

3.1.3 Linear Programming Formulation

In this chapter we will analyze apprenticeship learning algorithms that frame the imitation problem as a *Linear Programming (LP)* problem. They do not rely on the standard dynamic programming way of solving a known MDP. In fact, it is also possible to formulate a MDP problem as a linear programming problem that can be translated in its dual formulation. Both solving the primal LP or the dual LP lead to the solution of the MDP.

3.1.3.1 Primal and Dual LP

While solving the primal LP or the dual LP both give as result an optimal policy for an MDP, they involve a different formulation of the problem. The following definition of primal and dual LP is inspired to the one presented in Kakade's monograph [2].

Primal LP

The primal LP formulation of an MDP reflects the purpose of the value iteration algorithm. In fact, it tries to maximize the value function, obtaining the optimal value function V^*, and,

as final steps, it derives the policy that achieves the optimal value.

The primal LP formulation is as follows

$$\begin{aligned} &\underset{V \in \mathbb{R}^{|\mathcal{S}|}}{\text{maximize}} \quad \sum_{s} d_0(s) V(s) \\ &\text{subject to} \quad V(s) \geq (1-\gamma) R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \end{aligned} \tag{3.8}$$

where d_0 is the initial state distribution.

The solution of the above LP is the optimal value function, from where the optimal policy can be extracted as in step 3 of Algorithm 2.

Dual LP

In order to derive the dual formulation of the primal LP, it is necessary to introduce the notion of *state-action visitation distribution* (or *occupancy measure*) $\mu^{\pi}_{d_0}$ for a given policy π. It is defined as

$$\mu^{\pi}_{d_0}(s,a) = (1-\gamma) \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s, a_t = a | \pi, d_0) \tag{3.9}$$

The $\mu^{\pi}_{d_0}$ metric provides a measure of the visitation rate of a specific state-action pair (s,a) when following policy π.

It can be shown that a state-action visitation measure $\mu^{\pi}_{d_0}$ verifies the following equation:

$$\sum_{a} \mu^{\pi}_{d_0}(s,a) = (1-\gamma) d_0(s) + \gamma \sum_{s',a'} P(s|s',a') \mu^{\pi}_{d_0}(s',a') \tag{3.10}$$

This allows to formulate the following result.

Theorem 3.1.1 (Puterman, 1994)**.** The state-action polytope given by

$$\mathcal{K} := \{\mu | \mu \geq 0 \text{ and } \sum_{a} \mu(s,a) = (1-\gamma) d_0(s) + \gamma \sum_{s',a'} P(s|s',a')\mu(s',a')\} \tag{3.11}$$

is the set of all feasible state-action distributions, i.e. $\mu \in \mathcal{K}$ if and only if there exists a (stationary) policy π such that $\mu^{\pi} = \mu$.

It is worth noting that the set $\mathcal{K}$ is a convex set.

The conditions that defines the set $\mathcal{K}$ are often referred to as *Bellman flow constraints.*

This allows to formulate the dual LP problem as follows

$$\begin{aligned} \underset{\mu \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}}{\text{maximize}} \quad & \sum_{s,a} \mu(s,a) R(s,a) \\ \text{subject to} \quad & \mu \in \mathcal{K} \end{aligned} \tag{3.12}$$

The solution of the above linear programming problem is the optimal state-action distribution μ^*. As for the primal problem, the next and final step after the solution of the maximization problem is to extract the optimal policy. In this case the optimal policy is simply obtained by marginalization as

$$\pi^*(a|s) = \frac{\mu^*(s,a)}{\sum_{a'} \mu^*(s,a')} \tag{3.13}$$

The above equation specifies the relationship between a policy and its relative occupancy measure. In addiction, this relationship is uniquely determined, as shown in the next theorem.

Theorem 3.1.2 (Syed and Schapire, 2008)**.** Given an occupancy measure μ satisfying the Bellman flow constraints and given a stationary policy $\pi^*(a|s) = \frac{\mu^*(s,a)}{\sum_{a'} \mu^*(s,a')}$, μ is the occupancy measure of π.

Conversely, given a stationary policy π such that μ is its occupancy measure, $\pi^*(a|s) = \frac{\mu^*(s,a)}{\sum_{a'} \mu^*(s,a')}$ holds and μ satisfies the Bellman flow constraints.

An alternative way of extracting the optimal policy π^* from μ is to calculate it as

$$\pi^*(a|s) = \arg\max_a \mu^*(s, a) \tag{3.14}$$

If the optimal policy is unique, then optimal policies calculated in the two different ways gives the same identical result.

3.1.3.2 Distribution Matching

The method proposed by Kakade et al. [2] gives an alternative formulation of apprenticeship algorithms where the imitation goal is achieved by matching the state-action visitation distribution of expert and learner, namely $\mu^{\pi^{exp}}$ and μ^{π}.

The first consideration is that, as imitating the expert policy is the final goal, π^{exp} is obviously unknown. For this reason, $\mu^{\pi^{exp}}$ cannot be calculated exactly but it can be estimated from the examples trajectories performed by the expert. Given a dataset of m expert trajectories $\mathcal{D} = \{s_0^{(i)}, a_0^{(i)}, ...\}_{i=1}^{m}$, the empirical estimate $\hat{\mu}^{\pi^{exp}}$ of the expert occupancy measure $\mu^{\pi^{exp}}$ is given by

$$\hat{\mu}^{\pi^{exp}}(s, a) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \mathbb{1}(s_t^{(i)} = s, a_t^{(i)} = a) \quad \forall (s, a) \in \mathcal{D} \tag{3.15}$$

Given a larger set of demonstrations $\mathcal{D}$ the empirical estimate $\hat{\mu}^{\pi^{exp}}$ will be a better approximation of the true expert occupancy measure $\mu^{\pi^{exp}}$. It is reasonable to assume that the empirical estimate has an ϵ error in the following sense

$$\|\mu^{\pi^{exp}} - \hat{\mu}^{\pi^{exp}}\|_1 \leq \epsilon \tag{3.16}$$

The optimization problem is given by

$$\begin{aligned} &\underset{\mu \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}}{\text{minimize}} \quad |\mu(s,a) - \hat{\mu}^{\pi^{exp}}(s,a)| \\ &\text{subject to} \quad \mu \in \mathcal{K} \end{aligned} \tag{3.17}$$

The above optimization problem is a convex problem as it tries to minimize a convex objective function subject to convex constraints.

From the optimal occupancy measure μ^{DM} that comes from the solution of the optimization problem, the relative optimal policy can be derived by marginalization as

$$\pi^{DM}(a|s) = \frac{\mu^{DM}(s,a)}{\sum_{a'} \mu^{DM}(s,a')} \tag{3.18}$$

The theoretical guarantees of the proposed algorithm are given by the following theorem.

Theorem 3.1.3 (Kakade et al., 2019)**.** Given that the estimation error of $\hat{\mu}^{\pi^{exp}}$ is bounded by ϵ (i.e. $\|\mu^{\pi^{exp}} - \hat{\mu}^{\pi^{exp}}\|_1 \leq \epsilon$), then the following inequality holds

$$|V^{\pi^{DM}}(d_0) - V^{\pi^{exp}}(d_0)| \leq 2\epsilon \tag{3.19}$$

Proof: Since $\mu^{\pi^{exp}}$ is a feasible point of problem (3.18), it has an objective value of ϵ. Therefore, any optimal solution has a lower objective value, meaning that

$$\|\mu^{DM} - \hat{\mu}^{\pi^{exp}}\|_1 \leq \epsilon \tag{3.20}$$

The following considerations complete the proof.

$$\begin{aligned} |V^{\pi^{DM}}(d_0) - V\pi^{exp}(d_0)| &= \left| \sum_{s,a} \mu^{DM}(s,a)R(s,a) - \sum_{s,a} \mu^{\pi^{exp}(s,a)}R(s,a) \right| \\ &= \left| \sum_{s,a} \left(\mu^{DM}(s,a) - \mu^{\pi^{exp}}(s,a) \right) R(s,a) \right| \\ &\leq \|\mu^{DM} - \mu^{\pi^{exp}}\|_1 \\ &\leq \|\mu^{DM} - \hat{\mu}^{\pi^{exp}}\|_1 + \|\hat{\mu}^{\pi^{exp}} - \mu^{\pi^{exp}}\|_1 \\ &\leq 2\epsilon \end{aligned} \tag{3.21}$$

Using the value functions as metric of policy similarity is a reliable and portable way of comparing the two policies. Guaranteeing the similarity between value functions assures the closeness of the respective policies.

3.1.3.3 LPAL

The *Linear Programming Apprenticeship Learning (LPAL)* algorithm proposed by Syed and Schapire [40], as the Distribution Matching, exploit a Linear Programming formulation of the problem. LPAL uses a concept widely diffused in Imitation Learning literature according to which the unknown reward function can be expressed as linear combination of known features

$\phi_1, ..., \phi_k$ that constitute the basis functions of the reward. In this setting, the reward corresponding to the state-action pair (s, a) is given by

$$R(s, a) = \sum_i w_i \phi_i(s, a) \tag{3.22}$$

where the weights $w_1, ..., w_k$ realize the trade-off between the different features.

LPAL algorithm, in particular, considers the reward function to be a convex combination of the given features, meaning that the weights w_i are subject to the constraints $w_i \geq 0$ and $\sum_i w_i = 1$.

The authors define for each basis function of the reward a corresponding *basis value function* given by

$$V_i^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \phi_i(s_t, a_t) \middle| s_0 = s \right] \tag{3.23}$$

The key idea of the proposed algorithm is to find, among all the feasible occupancy measures in the feasibility set $\mathcal{K}$, the one whose policy has the relative value function that is the closest as possible to the expert's value function. The described problem is framed as a Linear Programming optimization problem in the LPAL method, detailed in Algorithm 3.

Algorithm 3: Linear Programming Aprenticeship Learning (LPAL)

Input: $\mathcal{S}, \mathcal{A}, P, d_0, \gamma, \phi_1 ... \phi_k, \mathcal{D}$

Output: optimal imitation policy π^{LPAL}

1 Compute an empirical estimate $\hat{V}_i^{\pi^{exp}}$ of the expert basis value functions using the set of demonstrations $\mathcal{D}$

2 Find a solution (β^*, μ^*) to the following LP

$$\begin{aligned} &\underset{\beta,\mu}{\text{maximize}} \quad \beta \\ &\text{subject to} \\ &\beta \leq \sum_{s,a} \phi_i(s,a)\mu(s,a) - \hat{V}_i^{\pi^{exp}} \\ &\sum_a \mu(s,a) = d_0 + \gamma \sum_{s',a'} \mu(s',a')P(s|s',a') \\ &\mu(s,a) \geq 0 \end{aligned} \tag{3.24}$$

3 Return the policy π^{LPAL} computed from μ^* as

$$\pi^{LPAL}(s,a) = \frac{\mu^*(s,a)}{\sum_a \mu^*(s,a)} \tag{3.25}$$

3.2 Inverse Reinforcement Learning

Inverse Reinforcement Learning (IRL), also called Inverse Optimal Control (IOC), is the second main category of Imitation Learning. Its goal is to recover the reward function that best explains the expert behavior. Under the assumption that the expert is optimal with respect to an unknown reward function, IRL methods try to find the reward that the expert is optimizing in the given demonstrations.

A comparison between Reinforcement Learning and Inverse Reinforcement Learning is give in Figure 4.

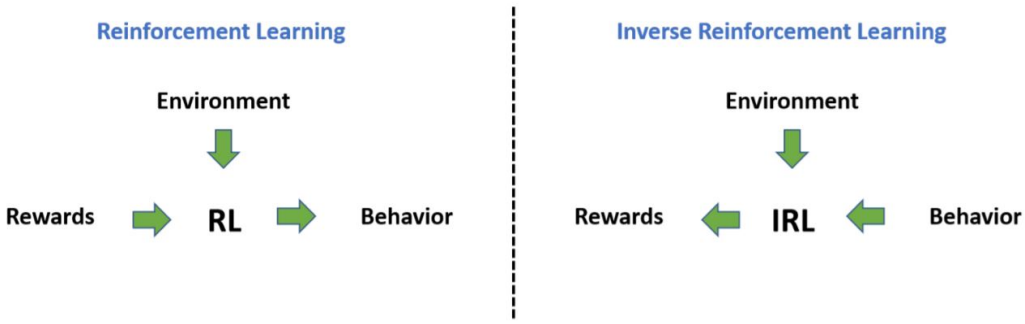


Figure 4: Comparison between RL and IRL

Any IRL algorithm takes as input the set of expert's trajectories and iterates over the following procedure:

- evaluate the current policy against the expert policy
- quantify the difference with a loss function
- update the reward parameters
- update the learned policy solving a RL problem with the updated reward

The performance of the policy corresponding to the current reward can only be assessed by actually computing the policy and comparing its results with the expert examples. Therefore, an RL method is needed in the inner loop of the algorithm.

The particular way of performing the different steps differs depending on the IRL procedure considered.

One of the main difficult of IRL methods is that the problem of finding the reward that best explains the expert behavior is an ill-posed problem.

In fact, its solution suffers from *ambiguity* (first pointed out in [31]). In particular, given the same set of demonstrations, the reward function corresponding to the policy that can explain the example trajectories is not unique. To give a trivial but efficient proof of this ambiguity problem, it is straightforward to see that the the reward function that always gives a zero return can, in principle, explain any demonstrated behavior.

In order to obtain a uniquely defined solution for IRL, many different approaches have been proposed that include an additional optimization criterion (e.g. maximum margin, maximum entropy, etc.).

An overview of the most interesting IRL approaches is proposed in the rest of this chapter.

3.2.1 Maximum Margin Optimization

One of the first formulation of Inverse Reinforcement Learning comes from Abbeel and Ng [1].

In their method, as introduced in chapter 3.1.3.3, the unknown reward function is considered to be a linear combination of known features $\phi_1, ..., \phi_k$ collected in the feature vector $\phi : \mathcal{S} \rightarrow [0, 1]^k$. The reward can be expressed as

$$R(s) = w^T \phi(s) \tag{3.26}$$

where w is the vector containing the weights for each feature.

Differently from LPAL, where the reward function is assumed to be a convex combination of the k basis functions, here the weights are interested from a different constraint that bounds the l_1-norm of the weight vector as $\|w\|_1 \leq 1$.

In this setting, the expected value of a policy π can be expressed as

$$\begin{aligned} \mathbb{E}_{s_0 \sim d_0}\left[V^\pi(s_0)\right] &= \mathbb{E}_{s_0 \sim d_0}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right] \\ &= \mathbb{E}_{s_0 \sim d_0}\left[\sum_{t=0}^{\infty} \gamma^t w^T \phi(s_t)\right] \\ &= w^T \mathbb{E}_{s_0 \sim d_0}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)\right] \end{aligned} \tag{3.27}$$

The authors define the *feature expectation* of a policy π as

$$\nu(\pi) = \mathbb{E}_{s_0 \sim d_0}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)\right] \tag{3.28}$$

With this notation, the value of a policy can be written as

$$\mathbb{E}_{s_0 \sim d_0}\left[V^{\pi}(s_0)\right] = w^{\mathsf{T}}\nu(\pi) \tag{3.29}$$

In order to find a policy whose performance is close to the one of the expert, the proposed method aims to match the feature expectations of the expert $\nu(\pi^{exp})$ and the learner $\nu(\pi)$. In fact, it can be shown that the closeness between feature expectations translates in the closeness between value functions and, therefore, in policy performances.

To prove this, let us assume that the l_2-norm of the difference between expert and learner feature expectations is bounded by ϵ

$$\|\nu(\pi) - \nu(\pi^{exp})\|_2 \leq \epsilon \tag{3.30}$$

The following considerations can be done :

$$\begin{aligned} \left|\mathbb{E}_{\pi^{exp}}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t)\right] - \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t)\right]\right| &= \left|w^{\mathsf{T}}\nu(\pi) - w^{\mathsf{T}}\nu(\pi^{exp})\right| \\ &\leq \|w\|_2\|\nu(\pi) - \nu(\pi^{exp})\|_2 \\ &\leq \epsilon \end{aligned} \tag{3.31}$$

The implementation of the feature matching proposed by the authors, called *max-margin* method, learns a policy such that the difference between such policy and all the other policies found in the previous iteration of the algorithm is maximized. The specific procedure is illustrated in Algorithm 4.

Algorithm 4: max-margin

Input: threshold ϵ,demonstration set $\mathcal{D}$

1 Randomly initialize policy π_0

2 Compute relative feature expectation $\nu^{(0)} = \nu(\pi_0)$

3 Compute the empirical estimate $\hat{\nu}(\pi^{exp})$ of the expert features from the demonstration set $\mathcal{D}$

4 Set i=0

5 **while** $t^{(i)} \leq \epsilon$ **do**

Compute $t^{(i)} = \max_w \min_{j \in \{0...(i-1)\}} w^T(\hat{\nu}(\pi^{exp}) - \nu^{(j)})$

Set $R(s) = (w^{(i)})^T \phi(s)$ and compute the relative optimal policy $\pi^{(i)}$ and feature expectation $\nu(\pi^{(i)})$

Set $i \leftarrow i + 1$

end

6 Return the set of policies $\{\pi^{(i)} : i = 0, ..., n\}$

As many IRL procedure, the max-margin involves the presence of a standard RL subroutine in the inner loop.

Algorithm 4 returns the set of policies calculated at each iteration. To select the optimal one

according to the feature matching principle, the following Quadratic Program can be solved to find the point closest to $\hat{\nu}(\pi^{exp})$ in the convex closure $\mathcal{V}$ of $\nu^{(0)}, ..., \nu^{(n)}$:

$$\begin{aligned} \underset{\nu \in \mathcal{V}}{\text{minimize}} \quad & \|\nu(\pi^{exp}) - \nu\|_2 \\ \text{subject to} \quad & \nu = \sum_i \lambda \nu^{(i)} \\ & \lambda_i \geq 0 \\ & \sum_i \lambda_i = 1 \end{aligned} \tag{3.32}$$

where λ realizes the mixture of feature expectations that is the closest possible to the expert's one.

3.2.2 Maximum Entropy Optimization

In recent literature the maximum entropy principle (introduced by Jaynes, 1957) has attired increasing attention and it has been widely used, in different variations, for solving the ambiguity problem that is intrinsic in every IRL problem. This approach has been introduced by Ziebart et al. in [48] and in the course of the years has been modified and further developed, evolving in a large number of different approaches. In this chapter, the original formulation of the maximum entropy IRL will be reviewed, as well as the most interesting and promising developments that constitute the state-of-the-art.

3.2.2.1 MaxEnt-IRL

The algorithm proposed by Ziebart et al. in [48], called *MaxEnt-IRL*, tries to recover a reward function that explains the expert behavior and it solves the ambiguity problem by

selecting the policy that maximizes the entropy.

The reward of a trajectory τ is considered to be the linear combination, with unknown weighting factors, of the features $\phi(\tau)$ of the trajectory:

$$R(\tau) = w^T \phi(\tau) \tag{3.33}$$

As the max-margin approach, the MaxEnt method focus on matching the expected feature count and chooses, among all the distributions that realize the matching, the one that maximizes the entropy.

The entropy of a distribution over trajectories is given by

$$H(p(\tau)) = \sum_{\tau} p(\tau) \log\Big(\frac{1}{p(\tau)}\Big) = \mathbb{E}_{\tau \sim p}\big[-\log(p(\tau))\big] \tag{3.34}$$

where $p(\tau)$ is the probability of trajectory τ.

The optimization problem proposed for the MaxEnt IRL method is the following:

$$\begin{aligned} \underset{p}{\text{maximize}} \quad & H(p(\tau)) \\ \text{subject to} \quad & \mathbb{E}_{\pi}\big[\phi(\tau)\big] = \mathbb{E}_{\pi^{exp}}\big[\phi(\tau)\big] \\ & \sum_{\tau} p(\tau) = 1 \\ & p(\tau) \geq 0 \end{aligned} \tag{3.35}$$

Since the expert policy is unknown, the term $\mathbb{E}_{\pi^{exp}}\left[\phi(\tau)\right]$ needs to be estimated from the set of expert demonstrations. The empirical estimate can be realized as

$$\mathbb{E}_{\pi^{exp}}\left[\phi(\tau)\right] \simeq \frac{1}{m}\sum_{i=1}^{m}\phi(\tau_i) \tag{3.36}$$

where m is the number of example trajectories.

It turns out that the policy that realizes the feature count matching and, at the same time, maximizes the entropy, is given by the Boltzmann distribution:

$$p(\tau|w) = \frac{1}{Z(w)}exp(w^T\phi(\tau))) \tag{3.37}$$

where Z is the partition function given by

$$Z(w) = \sum_{\tau} exp(w^T\phi(\tau)) \tag{3.38}$$

The intuition behind the Boltzmann formulation of $p(\tau)$ is that trajectories that earn a higher reward are exponentially more likely to be sampled.

In other words, maximizing the entropy of the distribution over trajectories subject to the feature constraint coincides with maximizing the likelihood of the demonstrated trajectories under the Boltzmann distribution. Therefore, the maximization objective is:

$$\mathcal{L}(w) = \sum_{\tau\in\mathcal{D}} logp(\tau|w) \tag{3.39}$$

The gradient of the above maximization function is the difference between the expected expert future counts (or, more precisely, its empirical estimate) and the expected feature count of the learner. The latter one can also be expressed as function of the expected state visitation frequencies D_{s_i}, giving the following gradient:

$$\nabla \mathcal{L}(w) = \mathbb{E}_{\pi^{exp}}[\phi(\tau)] - \sum_{s_i} D_{s_i} \phi(s_i) \tag{3.40}$$

A straightforward solution for the computation of the expected state visitation frequencies D_{s_i} would be to go through the enumeration of all feasible trajectories τ. However, as the number of possible trajectories grows exponentially with the time horizon, this brute-force approach becomes infeasible for almost all practical applications.

A more efficient solution, proposed by Ziebart et al. in [48], is illustrated in Algorithm 5.

Algorithm 5: Computation of the state-visitation frequencies D_{s_i}

Input: $w, \mathcal{S}, \mathcal{A}, P, N$

Output: State visitation frequencies D_{s_i}

Backward Pass

Set $Z_{s_i,0} \leftarrow 1$

for *N iterations* **do**

$Z_{a_{i,j}} = \sum_k P(s_k|s_i, a_{i,j}) exp(R(s_i|w)) Z_{s_k}$

$Z_{s_i} = \sum_{a_{i,j}} Z_{a_{i,j}}$

end

Local action probability computation

$P(a_{i,j}|s_i) = \frac{Z_{a_{i,j}}}{Z_{s_i}}$

Forward Pass

Set $D_{s_i,t} = P(s_i = s_0)$

for $t = 1 : N$ **do**

$D_{s_i,t+1} = \sum_{a_{i,j}} \sum_k D_{s_k,t} P(a_{i,j}|s_i) P(s_k|a_{i,j}, s_i)$

end

Summing frequencies

$D_{s_i} = \sum_t D_{s_i,t}$

Given the visitation frequencies computed as per Algorithm 5, the general scheme of the Max-Ent IRL algorithm can be summarized in Algorithm 6.

Algorithm 6: MaxEnt IRL algorithm

1 Initialize weight vector w randomly

2 Solve the MDP with reward $\mathsf{R}(\cdot|w)$

3 Get state visitation frequency (Algorithm 5)

4 Compute the gradient $\nabla\mathcal{L}(w)$ (Equation 3.40)

5 Update w with one gradient step using $\nabla\mathcal{L}(w)$

6 If not converged : come back to step 2

As many IRL algorithms, also MaxEnt IRL involves fully solving an MDP in the inner loop. This can be computationally expensive as the the dimension of the state and action spaces increase. A number of procedures have been proposed in literature to overcome this issues.
The rest of this chapter is devoted to the analysis of two imitation learning techniques that leverage the similarity between IRL and GANs [21] to apply Deep Learning to the imitation problem.

3.2.2.2 GAIL

Generative Adversarial Imitation Learning (GAIL) algorithm, proposed by Ho and Ermon [23], is not a Inverse Reinforcement Learning method in the strict sense since it does not explicitly attempt to recover the reward function. Nevertheless, it is presented here as it builds on top of the MaxEnt IRL algorithm.
GAIL borrows the conceptual idea behind the GAN principle and adapts it to the problem of learning from demonstrations.

The key underlying concept of the GAN theory (first introduced by Goodfellow et al. [21]) is to use two Neural Networks that, given a set of examples, learn how to artificially generate new samples that look as real as the original ones in the dataset. The first Neural Network plays the role of the *Generator* while the second one serves as *Discriminator*. The Generator net is fed with the examples in the dataset and it outputs new artificial samples. The Discriminator, observing the output of the Discriminator, outputs a value between 0 and 1 that represents the probability of the observed sample being original (coming from the dataset) or artificial (coming from the Generator). During the learning process, both Generator and Discriminator improve. The Generator learns how to generate new samples that look as similar as possible to the ones in the given dataset, with the purpose of making the Discriminator evaluate its outcome as original data. At the same time, the Discriminator improves as it gets better in discriminating between authentic and newly created data. Once the learning process is completed and both the neural networks are optimized, the Discriminator accomplished its scope and only the Generator is kept for getting more data whenever it is needed.

Ho and Ermon adapted this concept for the purpose of imitation learning. In this context, the Generator represents the policy, that constantly generates new trajectories, while the Discriminator plays the role of the reward function and learns how to differentiate between trajectories generated by the expert policy and by the agent's policy.

The authors begin with the following formulation of the MaxEnt IRL objective:

$$\underset{c \in \mathcal{C}}{\text{maximize}} \quad \left(\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_{\pi}\big[c(s,a)\big] \right) - \mathbb{E}_{\pi^{exp}}\big[c(s,a)\big] \tag{3.41}$$

In the above equation, c represent the cost function, namely the negative of the reward function, while $\mathcal{C}$ is the set of all possible cost functions. Π represents the set of all possible stochastic policies and $H(\pi)$ is the causal entropy of policy π, defined as $H(\pi) \triangleq \mathbb{E}_\pi[-\log\pi(a|s)]$. Choosing the largest possible set of cost functions, coinciding with $\mathcal{C} = \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, the IRL can be affected by high variance when a finite dataset is given. To overcome this issue, the authors proposed to incorporate a convex cost function regularizer $\psi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \bar{\mathbb{R}}$ where $\bar{\mathbb{R}}$ denotes the extended set of real numbers $\mathbb{R} \cup \{\infty\}$. The resulting regularized objective becomes:

$$\underset{c\in\mathbb{R}^{\mathcal{S}\times\mathcal{A}}}{\text{maximize}} \quad -\psi(c) + \left(\min_{\pi\in\Pi} -H(\pi) + \mathbb{E}_\pi\big[c(s,a)\big]\right) - \mathbb{E}_{\pi^{exp}}\big[c(s,a)\big] \tag{3.42}$$

Calling $\tilde{c}$ the solution of the optimization problem with objective given in (Equation 3.42), the imitation purpose is absolved when $\tilde{c}$ is used in a RL algorithm to get the final learned policy π. It is demonstrated in the original GAIL paper that this final operation, namely $\pi = RL(IRL_\psi(\pi^{exp})) = RL(\tilde{c})$, is equivalent to

$$RL \circ IRL_\psi(\pi^{exp}) = \arg\min_{\pi\in\Pi} -H(\pi) + \psi^*(\mu^\pi - \mu^{\pi^{exp}}) \tag{3.43}$$

where μ represents the occupancy measure (defined in (Equation 3.9)) and ψ^* represents the convex conjugate of ψ.

In the GAIL formulation, the regularizer ψ is chosen as follows:

$$\psi_{GAIL}(c) \triangleq \begin{cases} \mathbb{E}_{\pi^{exp}}\big[g(c(s,a))\big] & \text{if} \quad c < 0 \\ +\infty & \text{otherwise} \end{cases} \tag{3.44}$$

where the function g is defined as

$$g(x) = \begin{cases} -x - \log(1 - e^x) & \text{if} \quad x < 0 \\ +\infty & \text{otherwise} \end{cases} \tag{3.45}$$

It can be shown that, with this choice of the regularizer term, the following equation holds:

$$\psi^*_{GAIL}(\mu^\pi - \mu^{\pi^{exp}}) = \max_{D \in (0,1)^{\mathcal{S}\times\mathcal{A}}} \mathbb{E}_\pi\big[\log D(s,a)\big] + \mathbb{E}_{\pi^{exp}}\big[\log(1 - D(s,a))\big] \tag{3.46}$$

The point D^* that achieves the maximum of the objective in the previous equation is given by

$$D^*(s,a) = \frac{\mu^\pi(s,a)}{\mu^{\pi^{exp}}(s,a) + \mu^\pi(s,a)} \tag{3.47}$$

The following considerations can be done about the quantity $\psi^*_{GAIL}(\mu^\pi - \mu^{\pi^{exp}})$, that constitutes the second term of the objective in (Equation 3.43).

$$\begin{aligned}
\psi^*_{GAIL}(\mu^\pi - \mu^{\pi^{exp}}) &= \max_{D \in (0,1)^{\mathcal{S}\times\mathcal{A}}} \mathbb{E}_\pi\big[\log D(s,a)\big] + \mathbb{E}_{\pi^{exp}}\big[\log(1 - D(s,a))\big] \\
&= \mathbb{E}_\pi\big[\log D^*(s,a)\big] + \mathbb{E}_{\pi^{exp}}\big[\log(1 - D^*(s,a))\big] \\
&= \mathbb{E}_\pi\left[\log \frac{\mu^\pi(s,a)}{\mu^{\pi^{exp}}(s,a) + \mu^\pi(s,a)}\right] + \mathbb{E}_{\pi^{exp}}\left[\log \frac{\mu^{\pi^{exp}}(s,a)}{\mu^{\pi^{exp}}(s,a) + \mu^\pi(s,a)}\right] \\
&= D_{KL}(\mu^\pi \| \mu^\pi + \mu^{\pi^{exp}}) + D_{KL}(\mu^{\pi^{exp}} \| \mu^\pi + \mu^{\pi^{exp}}) \\
&= -\log 4 + D_{KL}(\mu^\pi \| \tfrac{\mu^\pi + \mu^{\pi^{exp}}}{2}) + D_{KL}(\mu^{\pi^{exp}} \| \tfrac{\mu^\pi + \mu^{\pi^{exp}}}{2}) \\
&= -\log 4 + 2D_{JS}(\mu^\pi \| \mu^{\pi^{exp}})
\end{aligned} \tag{3.48}$$

where D_{JS} is the *Jansen-Shannon divergence*.

With these considerations, imitation learning objective can be reformulated as:

$$\underset{\pi\in\Pi}{\text{minimize}} \quad \psi^*_{GAIL}(\mu^\pi - \mu^{\pi^{exp}}) - \lambda H(\pi) = \quad \underset{\pi\in\Pi}{\text{minimize}} \quad D_{JS}(\mu^\pi \| \mu^{\pi^{exp}}) - \lambda H(\pi) \tag{3.49}$$

where $\lambda \geq 0$ is a weighting factor controlling the trade-off between entropy maximization and minimization of the JS divergence between the state-action visitation measures of expert and learner.

For solving the imitation problem, GAIL method finds a saddle point for the following problem, obtained by substituting (Equation 3.46) in (Equation 3.43).

$$\min_{\pi \in \Pi} \max_{D \in (0,1)^{\mathcal{S} \times \mathcal{A}}} \mathbb{E}_{\pi}\big[\log D(s,a)\big] + \mathbb{E}_{\pi^{exp}}\big[\log(1 - D(s,a))\big] - \lambda H(\pi) \tag{3.50}$$

Letting θ and w being respectively the parameters of the policy (Generator) and the reward (Discriminator) and calling $F(\theta, w)$ the objective of the problem in the above equation, the procedure for implementing the GAIL method is detailed in Algorithm 7.

Algorithm 7: GAIL

Input: Demonstration set $\mathcal{D}$

Output: Parameters θ and w of Generator and Discriminator

Initialize arbitrarily policy π and initial parameters θ_0 and w_0

for $t = 0, 1, 2, ...$ **do**

  Update discriminator parameters as $w_{t+1} \leftarrow w_t + \eta \nabla_w F(\theta_t, w_t)$

  Update generator parameters as $\theta_{t+1} \leftarrow \theta_t + \eta \nabla_\theta F(\theta_t, w_t)$

end

3.2.2.3 AIRL

Building on top of GAIL, Fu et al. [18] present a variation of the method, called *Adversatial Inverse Reinforcement Learning* (AIRL), that, as the name suggests, is a IRL algorithm in the

proper sense. In fact, AIRL aims to fully recovering a reward function under the argument that the reward is the most portable representation of the task.

In AIRL, the discriminator is defined as

$$D(s,a) = \frac{exp(f(s,a))}{exp(f(s,a)) + \pi(a|s)} \tag{3.51}$$

The reward, instead, is defined as

$$R(s,a) = logD(s,a) - log(1 - D(s,a)) \tag{3.52}$$

The overall procedure is exposed in Algorithm 8.

Algorithm 8: AIRL

Input: Demonstration set $\mathcal{D}$

Output: Reward function R and relative policy π

1 Initialize arbitrarily policy π and initial parameters θ_0 and w_0

for $t = 0, 1, 2, ...$ **do**

2 Execute π and collect trajectories $\tau_i = (s_0, a_0, ..., s_T, a_T)$

3 Train Discriminator neural network to classify expert data in $\mathcal{D}$ and generated samples τ_i

4 Update reward as per (Equation 3.52) Calculate optimal policy π with respect to the updated reward

end

CHAPTER 4

SAFE IMITATION LEARNING IN THE EPISODIC SETTING

The two main formulations of MDP that are most commonly considered in the Imitation Learning community and, more in general, in the RL literature are the infinite and finite horizon MDP.

- Infinite horizon MDP. The task to be performed extends over an infinite time horizon. In this setting, the agent's goal is to maximize the long-term return giving precedence to maximizing the rewards collected in time steps that are closer in time. In fact, the infinite horizon formulation requires a discounted factor $\gamma < 1$ to be included in the MDP specifications to prevent the value function from diverging. An infinite time horizon formulation of the MDP is suitable for controlling tasks that do not end, as for example the task of controlling the traffic at the intersection.

- Finite horizon MDP. The problem extends over a fixed and predetermined finite time horizon T. The goal of the agent is to maximize the sum of the rewards that it can accumulate up to time T. In this setting, a discount factor is optional. It can be included if the intention is to prioritize rewards collected early in the time line, but the absence of the discount factor (that coincides with the choice $\gamma = 1$) does not cause the value function to diverge, as it will be the sum of a finite number of terms. A finite time hori-

zon formulation is suitable for problems that focus on optimizing the performances over a finite time window, as for example the task of navigating a map moving from a point to another at each time step. In this scenario, the agent will try to optimize that sequence of T moves that maximize the return collected from the initial to the final time instant.

An alternative setting to the previous ones that has not attracted large attention in literature is the *episodic setting*. It involves the presence of a state, called *absorbing* or *terminal* state, that terminates the episode whenever reached. It is, in principle, an extension of the infinite time horizon formulation, as the time window cannot be predefined, but, from a practical point of view, it only extends over a finite number of steps if the terminal state is achieved. In the following, an episodic task will be said to extend over an *indefinite* time horizon. In fact, the terminal time is unknown before running the policy.

As for the finite horizon case, a discount factor γ is optional for the same considerations.

The episodic formulation of the MDP is the optimal mathematical representation of a large number of problems, including the one considered in this work. As a matter of fact, the problem of controlling a car merging in the highway lasts for an indefinite time period because it ends when the car merged the next lane (absorbing state), but the time needed for the completion of the merging maneuver cannot be determined a priori and will differ from case to case.

In the rest of this chapter we will present a mathematical formalization for the episodic setting. Subsequently, a mini-max formulation for the imitation learning problem will be proposed and, finally, it will be tested in an experiment considering discrete state and action spaces.

4.1 Indefinite Time Horizon MDP

The following formalization of the episodic MDP is based on the one given in *Dynamic Programming and Optimal Control* by Dimitri P. Bertsekas [7]. The author refers to the episodic MDP as *Stochastic shortest path problem* (SSPP). The relevant problems for this setting are the ones where reaching the termination state is unavoidable, at least when following an optimal policy. This assumption is formalized as follows.

Assumption 4.1.1 (Bertsekas, 2005)**.** There exists an integer m such that, regardless of the policy used and the initial state, there is a positive probability that the termination state will be reached in no more than m stages, that is, the following holds for all admissible policies π:

$$\max_{i} P(s_m \neq s_{term} | s_0 = i, \pi) < 1 \tag{4.1}$$

In addiction, unitary discount factor is assumed ($\gamma = 1$).

The absorbing state is defined as a special cost-free termination state that, whenever reached, causes the system to remain there indefinitely at zero additional cost. Formally, every state $s_{term} \in \mathcal{S}_{term}$, where $\mathcal{S}_{term}$ is the set of all absorbing states, is subject to the following conditions:

$$\begin{aligned} P(s_{term} | s_{term}, a) &= 1 \quad \forall\, a \in \mathcal{A} \\ R(s_{term}, a) &= 0 \quad \forall\, a \in \mathcal{A} \end{aligned} \tag{4.2}$$

4.2 Problem Setup

In this chapter we present a mini-max formulation for solving the imitation learning problem in the episodic setting. The mini-max framework allows to pose the problem in an adversarial fashion, in a similar way to what is done in the GAIL algorithm, where the generator fights against the discriminator to optimize the policy. The approach proposed is inspired to *Distribution Matching* [2] and *LPAL* [40], presented in chapter 2.

The problem that we want to solve is the following.

$$\begin{aligned} \underset{\mu^\pi}{\text{minimize}} \quad & \max_{c \in \mathcal{C}} \left| J_c(\pi) - J_c(\pi^{exp}) \right| \\ \text{subject to} \quad & \mu^\pi \in \mathcal{K} \end{aligned} \tag{4.3}$$

where:

- μ^π is the state-action visitation measure of the learned policy
- π is the learned policy corresponding to the state-action visitation μ^π and calculated from it by marginalization (as per (Equation 3.13))
- $\mathcal{K} := \{\mu | \mu \geq 0 \text{ and } \sum_a \mu(s,a) = d_0(s) + \gamma \sum_{s',a'} P(s|s',a')\mu(s',a')\}$, with d_0 being the distribution of initial states, is the (convex) set of all feasible state-action visitation measures; in line with the episodic formulation $\gamma = 1$ will be considered
- $\mathcal{C}$ is the set of all possible cost functions

- $J_c(\pi)$ is the long-term cost of policy π, it can be expressed as

$$J_c(\pi) = \mathbb{E}_{s\sim d_0}\left[V^\pi(s)\right] = \sum_{s,a} c(s,a)\mu^\pi(s,a) \tag{4.4}$$

The cost function can be expressed as a linear function of the feature vector $\phi(s,a)$, where the trade-off between different features is realized by the weighting factors in w:

$$-c(s,a) = R(s,a) = w^\intercal \phi(s,a) \tag{4.5}$$

In order to make the problem bounded, the assumption $\|w\| \leq 1$. For the moment we do not specify the type of norm involved in this constraint.

The mini-max problem in Equation 4.3 becomes:

$$\begin{aligned} \underset{\mu^\pi}{\text{minimize}} \quad & \max_{\|w\|\leq 1} \left| \sum_{s,a} w^\intercal \phi(s,a)(\mu^\pi(s,a) - \mu^{\pi^{exp}}(s,a)) \right| \\ \text{subject to} \quad & \mu^\pi \in \mathcal{K} \end{aligned} \tag{4.6}$$

However, $\mu^{\pi^{exp}}$ cannot be calculated exactly since the expert policy is unknown. Therefore, it will be approximated with its empirical estimate $\hat{\mu}^{\pi^{exp}}$ (Equation 3.15).

Moreover, notice that the absolute value can be removed from the problem without affecting its

optimal value μ^* that we are interested in. Those consideration lead to the following problem formulation:

$$\begin{aligned} \underset{\mu^\pi}{\text{minimize}} \quad & \max_{\|w\| \leq 1} w^\intercal \sum_{s,a} \phi(s,a)(\mu^\pi(s,a) - \hat{\mu}^{\pi^{exp}}(s,a)) \\ \text{subject to} \quad & \mu^\pi \in \mathcal{K} \end{aligned} \tag{4.7}$$

Let us focus on the inner maximization problem.

Defining $\beta \triangleq \sum_{s,a} \phi(s,a)(\mu^\pi(s,a) - \hat{\mu}^{\pi^{exp}}(s,a))$, the following observation can be carried out:

$$sup\{w^\intercal \beta \mid \|w\| \leq 1\} = \|\beta\|_* \tag{4.8}$$

where $\|\cdot\|_*$ is the dual norm associated to $\|\cdot\|$.

Here we will choose $\|\cdot\|$ to be the l_∞-norm, whose dual norm is given by the l_1-norm.

Therefore, the original problem simplifies to a minimization problem over μ^π only:

$$\begin{aligned} \underset{\mu^\pi}{\text{minimize}} \quad & \|\sum_{s,a} \phi(s,a)(\mu^\pi(s,a) - \hat{\mu}^{\pi^{exp}}(s,a))\|_1 \\ \text{subject to} \quad & \mu^\pi \in \mathcal{K} \end{aligned} \tag{4.9}$$

Notice that, if the function Φ is chosen to be the Dirac delta function, the above problem is equivalent to the convex optimization problem considered by the Distribution Matching method

(Equation 3.17).

Considering the constraints that define the convex set $\mathcal{K}$:

$$\begin{aligned} \underset{\mu}{\text{minimize}} \quad & \|\sum_{s,a} \phi(s,a)(\mu^{\pi}(s,a) - \hat{\mu}^{\pi^{exp}}(s,a))\|_1 \\ \text{subject to} \quad & \mu \geq 0 \\ & \sum_{a} \mu(s,a) = d_0(s) + \sum_{s',a'} P(s|s',a')\mu(s',a') \end{aligned} \tag{4.10}$$

4.2.1 Safety Guarantees

When the expert demonstrated trajectories are exhaustive enough to give a detailed representation of the task, the procedure of learning from demonstrations returns particularly satisfactory results and, in most applications, it is sufficient for the control problem. However, when it comes to safety critical scenarios, the pure imitation of the expert may not be enough. A typical example of problems where the safety criterion is crucial for the good outcome of the task is given by any control application that involves dealing with human lives, such as the problem of controlling a self-driving car. In this case, a small error can lead to catastrophic results.

In the autonomous driving field, the apprenticeship learning problem has the purpose of imitating the decision-making process that a human realizes when driving. For this reason, the expert demonstrations often come from real-world data collected from human-driven cars. Since different persons can have different driving styles and take different decisions being in the same situation, there is no universal notion of "optimality" when performing a specific driving task.

Thus, collected real-world data can contain demonstrated trajectories that achieve different degrees of "criticality", where the notion of criticality includes safety critical factors such as safety distance from surrounding cars, observance of speed limits etc.

Even supposing that the expert dataset only contains trajectories with a low level of criticality, an algorithm that performs the pure imitation of such data will not have a probability equal to 1 of preventing the controlled car from landing in a critical state.

Therefore, in situations when ensuring safety is an absolute priority, an imitation learning algorithm alone will not suffice. In this work, we will conceive the notion of "safety" as a priority aspect and therefore we do not rely on the expert demonstrations only but we want to enforce safety as an external factor. The resulting algorithm will be guaranteed to observe the enforced safety constraints while performing the imitation task.

In order to properly guarantee safety, we need a definition (and a subsequent formalization) of what is considered to be safe or unsafe. For every application where safety is a determinant feature, it is possible to identify a state, or a set of states $\mathcal{S}_{\texttt{forbidden}}$ that the agent must avoid. As the definition of the set of forbidden states depends on the particular problem, we do not give here any specification about the characterization of the set $\mathcal{S}_{\texttt{forbidden}}$. A practical application example will be given in the next chapter when the specif problem of controlling a merging maneuver will be considered.

We define to be unsafe every state from where the probability of reaching a forbidden state

is greater than a certain threshold δ_{safety}, that can be tuned depending on the specific needs. Thus, the set of all unsafe states $\mathcal{S}_{unsafe}$ is defined as follows:

$$\mathcal{S}_{unsafe} = \{s \in \mathcal{S} \mid P(s'|s,a) \geq \delta_{safety} \quad \forall\, a \in \mathcal{A}, s' \in \mathcal{S}_{forbidden}\} \tag{4.11}$$

The goal of the imitation learning algorithm with safety guarantees is to ensure the observance of safety constraints while imitating the expert. The safety limitations only affect those states that are defined to be unsafe, namely the states $s \in \mathcal{S}_{unsafe}$. For all the other states, the expert imitation alone accomplishes the task. On the other hand, when the agent is in a state $s \in \mathcal{S}_{unsafe}$, it should not act accordingly with the policy π^{IL} that comes from the solution of problem (4.10) (plus marginalization), that performs pure imitation learning. In those states, in fact, the agent should observe a different policy π^{safe} that has reliable safety guarantees. The described method results in a *switching policy* algorithm where the algorithm follows the following policy π:

$$\pi(s) \triangleq \begin{cases} \pi^{safe}(s) & \text{if } s \in \mathcal{S}_{unsafe} \\ \pi^{IL}(s) & \text{otherwise} \end{cases} \tag{4.12}$$

The policy π^{safe} to be followed in the unsafe states can be determined as follows. While the ground truth value of the reward function remains unknown, it is possible to specify an artificial reward function $\bar{R}$ that captures the criticalities of the state space $\mathcal{S}$ while having no information about all the safe states. In other words, $\bar{R}$ only contains information about what states the agent must avoid, but it does not require any knowledge about the relative quality of all states $s \in \mathcal{S}$. Therefore, it can be defined knowing only of the set of forbidden

states $\mathcal{S}_{\text{forbidden}}$. The policy π^{safe} can then be derived solving the MDP with reward $\bar{R}$.

The proposed design of the artificial reward $\bar{R}$ follows the following rules:

$$\bar{R}(s) = \begin{cases} 0 & \text{if } s \in \mathcal{S}_{\text{term}} \\ -\alpha z & \text{if } s \in \mathcal{S}_{\text{forbidden}} \\ -z & \text{otherwise} \end{cases} \tag{4.13}$$

where z is a a value that will be equally assigned as reward for all the safe state and α is a multiplicative factor that scales up with the grade of forbiddenness.

4.3 Grid-world Experiment

The method proposed in the previous section will be initially tested in a simple environment with discrete state and action spaces, such as a 5x5 Gridworld. The state space is composed by 25 discrete state, while the action state is defined by 5 possible actions available at each state (moving right, moving left, moving up, moving down, staying still).

The top right corner is chosen to be the forbidden state, while the bottom right corner is the absorbing state. An optimal policy would drive the agent towards the absorbing state in the least number of steps, while keeping distance from the forbidden state.

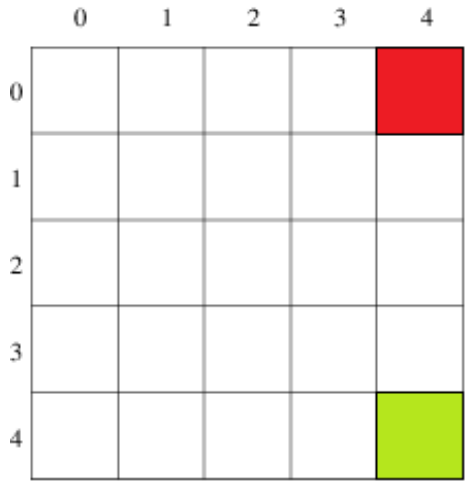


Figure 5: 5x5 Gridworld environment. Red cell: forbidden state, Green cell: Terminal state

The environment is chosen to be stochastic. In particular, taking action a in state s will result in a random move 30% of the times. In this configuration, the probability of reaching the forbidden state is never zero.

To gather the demonstrations a random ground truth reward is generated and a standard MDP is solved with value iteration with respect to it. The resulting policy is then used to generate the expert demonstrations with starting point chosen randomly among the safe states.

In the following are illustrated the ground truth reward and the artificially one generated as described in the previous section to compute the safety policy π^{safe}.

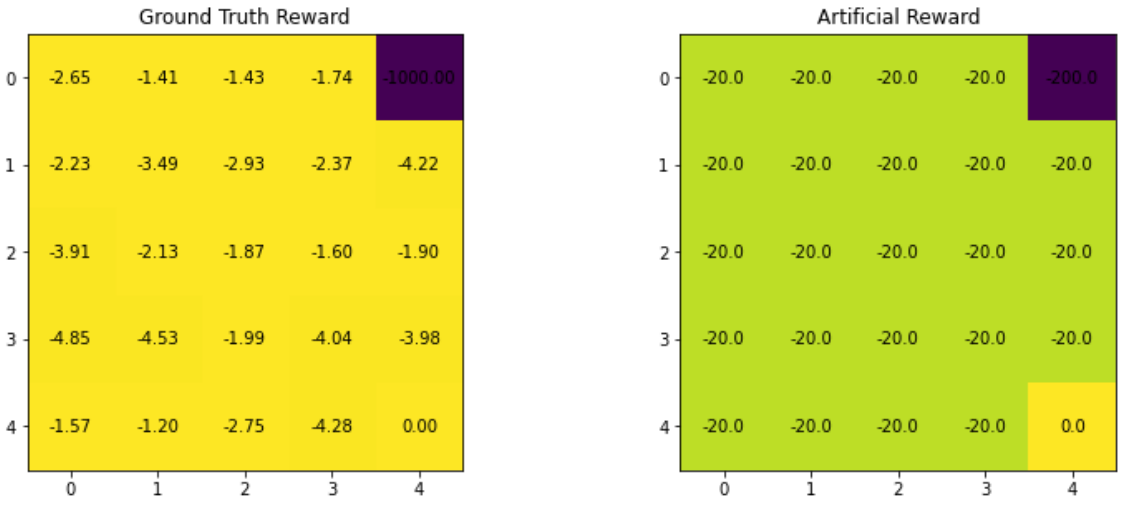


Figure 6: Ground truth and Artificial Rewards

The switched policy algorithm generates the following results:

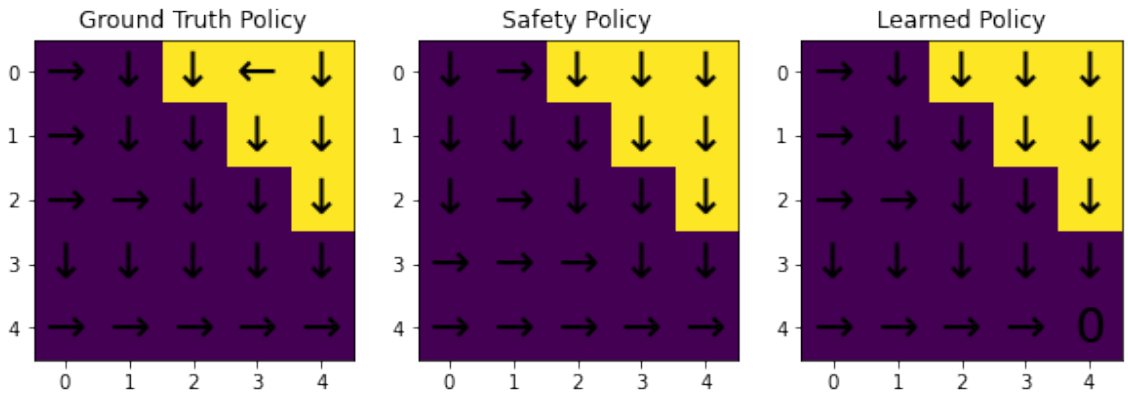


Figure 7: π^{exp}, π^{safe} and π in a grid representation

In the figure, the yellow cells represent the unsafe states, calculated as in (4.11) setting a safety threshold of $\delta_{safety} = 0.05$, meaning that the states that are classified as unsafe are the one from where the agent has a probability greater than 5% of reaching the forbidden state. As the image shows, the agent succeeds in imitating the expert when it is in a safe situation, while it follows what is dictated by the policy π^{safe} when it is detected to be in an unsafe state. Finally, the overall performances are evaluated by comparing the relative error between expert and agent value functions:

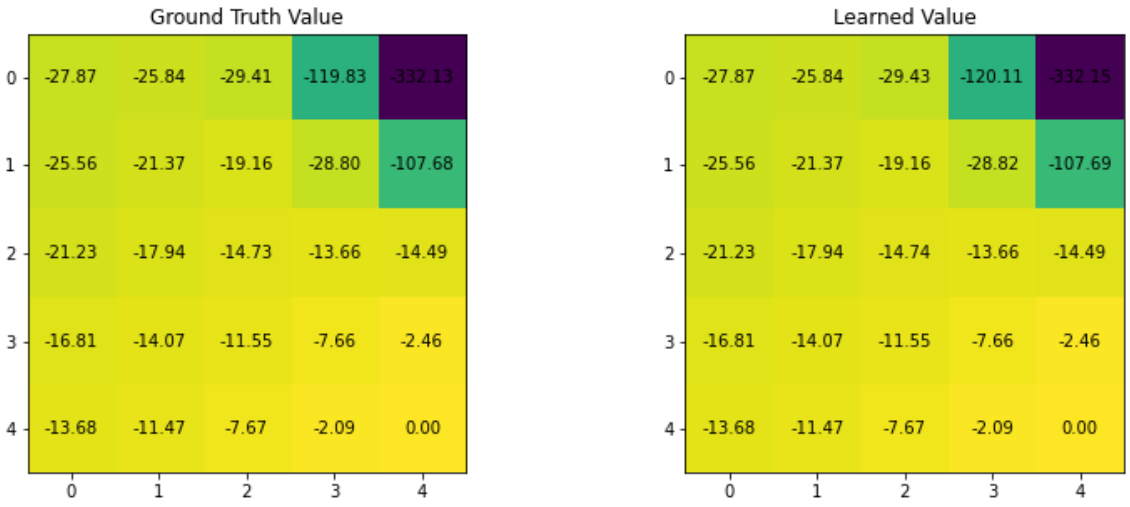


Figure 8: Comparison between expert and agent value functions

As a conclusion, the algorithm outcome is satisfactory as it succeeds in both the purposes of being safe and imitating the expert. As the similarity between value functions points out, the learned policy almost achieve the optimal value, while also observing the safety policy when needed.

Despite the satisfactory performances, this algorithm has some important limitations that prevent its application to more complex scenarios and, in our case, to the autonomous driving

merging task. First of all, it requires the knowledge of the transition dynamics, that is difficult to estimate in many real-world applications. Moreover, the use of the state-action visitation measure, whose matching is the key to the imitation phase of our algorithm, is not suitable when considering continuous state and action spaces. In fact, since the state action pairs $(\mathtt{s}, \mathtt{a})$ would be infinite, the convex problem would require an infinite number of constraints.

The next chapter considers the concrete case of designing an imitation learning algorithm for the task of controlling a self-driving car during its merging on the highway. It will take inspiration from the switching policy algorithm exposed in the present chapter, while proposing a solution to overcome its limitations.

CHAPTER 5

BEHAVIORAL CLONING WITH SAFETY GUARANTEES

The problem faced in this work involves learning from demonstrations the task of controlling a self-driving car in the specific maneuver of joining the highway traffic when coming from the on-ramp. Although this maneuver might be considered straightforward for a human, in reality it involves a sequence of complex operations. As reported by the National Highway Traffic Safety Administration [42], the counts of crashes related to the merging maneuver that happen every year is about 19,000. This data underlines the dangerousness of the operation and the importance of ensuring safety when designing a control system that automates the merging task.

Learning from demonstration examples is an efficient way of generating a control policy without designing an hand-crafted controller, however, this method does not have any safety guarantees, that are crucial in this case and, more in general, in the self-driving area or in any application that may put at risk a human life.

Hence, the scope of the presented method is not just to imitate the expert, but to doing it in the safest way possible. We propose to not rely on the expert demonstrations only for the safety encoding, but to augment the imitation procedure with safety guarantees. In this way, safety specifications can be enforced as a external factors and they can be tuned and adapted in order to realize a suitable trade-off between achieving the goal and do it safely. In fact, while being a priority, safety is not the only factor to consider: the optimal policy would drive the

agent through a sequence of safe states that lead to the efficient completion of the task.

The method proposed for augmenting the imitation learning with safety guarantees takes inspiration both from the Behavioral Cloning procedure and from the switching policy algorithm presented in the previous chapter. While the most concerning drawback of BC is the cascading error issue and the inability to recover from a critical unseen state, the main limitation of the switching policy algorithm presented in Chapter 3 is the impossibility of scaling it to environments where state and action spaces are continuous. The two approaches are combined in the *Behavioral Cloning with Safety Guarantees* approach in a way such that every of the two methods tries to overcome each other's limitations.

5.1 Problem Setup

Behavioral Cloning with Safety Guarantees method considers the two aspects of learning by imitating and ensuring safety, analyzed in the following.

5.1.1 Imitation Learning

The use of Behavioral Cloning as imitation learning algorithm has attracted limited consideration in modern literature due to its intrinsic limitations that prevent its application to challenging tasks. However, not considering the cascading error issue (that will be addressed further on), BC methods also entail significant advantages.

First of all, they allow to completely skip the reward function recovery step, that carries the ambiguity problem. Besides having multiple options for the choice of the reward function that represents an appropriate description of the expert behavior, given a determined reward function, also the choice of the relative optimal policy is not unique.

Moreover, a BC algorithm produces, given a set of demonstrations, a direct mapping between state space and action space, without the need of interacting with the environment. As a result, it does not require the knowledge of the model dynamics, unlike other methods such as distribution matching (Equation 3.11), LPAL (Equation 3.24) and every form of IRL that involves solving a forward RL as inner loop. This is an significant advantage in cases where estimating the model dynamics is not an easy task.

The use of Behavioral Cloning for the imitation phase allows us to overcome the limitations of the occupancy measure matching approach, that prevent its application to environments with continuous state and action spaces and unknown dynamics.

The practical implementation of the BC method for imitation implies using Supervised Learning on the provided dataset. A regression model is needed to learn the mapping between states and actions, given a number of demonstrated mappings realized by an expert. The choice of the appropriate regression model depends on the particular application. Simple models, such as Linear Regression, have the advantage of an easier training, but the low complexity may cause it to be not informative enough for the purpose. On the other hand, complex models, such as Neural Networks, can capture highly non-linear relations between states and actions but they can be difficult to train.

Given a regression model and a set of demonstrations $\mathcal{D} = \{(s_i, a_i)_{i=1}^N\}$, a BC algorithm applies Supervised Learning to find a mapping $\pi^{BC} : \mathcal{S} \rightarrow \mathcal{A}$.

5.1.2 Safety Guarantees

The switching policy method constitutes an effective solution to overcome the cascading error problem that arises from using BC as imitation learning procedure. Switching to a safe policy π^{safe} whenever the agent is detected to be in an unsafe state provides a reliable method for the agent to recover and come back to a safe situation, from where the imitation approach can then be used to complete the task.

Although the cascading error is still possible, in case it drives the agent to unseen and possibly safety-critical circumstances, from where no demonstrated examples are provided for the recovery, the behavior will never be undefined.

$$\pi(s) \triangleq \begin{cases} \pi^{safe}(s) & \text{if } s \in \mathcal{S}_{unsafe} \\ \pi^{BC}(s) & \text{otherwise} \end{cases} \tag{5.1}$$

The design of the policy π^{safe} depends on the specific application and on the specifications of the set $\mathcal{S}_{unsafe}$, that is also task-dependent. It can either determined using the artificial reward procedure specified in chapter 4.2.1 or manually designed. While the former approach is simpler from the design point of view, since it does not require hand-crafting the controlling specifications, the latter provides the possibility of enforcing precise directives whose observance is guaranteed in unsafe circumstances.

5.2 Experiment

In the conducted experiment, the described safe imitation procedure is applied in a real-world problem such as the control of autonomous driving vehicles during a safe merging maneuver. For this purpose, a simulator has been developed for a visual evaluation and assessment of the algorithm performances. The expert demonstrations consist of real data collected at the intersection between the highway and the on-ramp and it contains real merging maneuvers performed by human-driven vehicles, filmed from a top view. Given the complexity of the task, a Neural Network has been selected to represent the mapping $\pi^{\mathrm{BC}} : \mathcal{S} \rightarrow \mathcal{A}$.

The self-driving context gives rise to a variety of dangerous or unsafe situations, but all of them originate from the excessive proximity of two vehicles, that can lead to hazardous maneuvers or, in the worst cases, to an accident. Therefore, selected a safety distance δ_{dist} that should be present between every car and every other car in the merging scenario in order to consider the driving situation to be safe, the set of unsafe states can be specified as:

$$\mathcal{S}_{\mathrm{unsafe}} = \{\mathrm{s} \in \mathcal{S} \mid \mathrm{d}_{\mathrm{min}}(\mathrm{s}) < \delta_{\mathrm{dist}}\} \tag{5.2}$$

where $\mathrm{d}_{\mathrm{min}}(\mathrm{s})$ is the minimum distance, in state s, of the ego (controlled) car from every other vehicle.

The safety policy π^{safe} is designed as an hand-crafted controller, providing the designer with the full control of the safety directives to adopt in critical situations. It acts like a *safey filter* that let the control action suggested by π^{BC} pass unchanged in safe states, while filtering out

the actions taken in unsafe states, applying a filtering operation based on the specifications of π^{safe}.

5.2.1 Simulator

The simulator serves as graphic tool for the visualization of the trajectories realized by the learned policy. It required the extraction of the relevant geometric features from the section of interest of the real highway. Subsequently, the relevant information about all the vehicles in a specific time instant (e.g. position, length, width, etc.) are derived from the dataset and used to plot the cars that populate the merging scene. Although cars that are relevant for the merging purpose are specifically the ones on the on-ramp and on the lane immediately next to it, the full situation, also considering the other lanes, is pictured in the simulator to have an inclusive understanding of the scene. The visualization provided by the simulator in a specific time instant appears as the one illustrated in Figure 9.

Figure 9: Intantaneous visualization of the simulator

5.2.2 Dataset

The expert demonstrations are extracted from the *Next Generation SIMulation (NGSIM) dataset* [15], provided by the Federal Highway Administration (FHWA) division of the U.S. Department of Transportation. Among all the different highways present in the NGSIM dataset, we consider the data relative to the US Highway 101 (US 101), also known as the Hollywood Freeway, located in Los Angeles, CA. The section of interest extends for 640 meters (2100 feet) in length and 5 lanes in width, plus the additional on-ramp lane at Ventura Boulevard that is of particular interest in this study. The area of interest is shown in Figure 10.

The corridor following the on-ramp terminates in the off-ramp at Cahuenga Boulevard. Although the merging lane flows into an off-ramp and does not terminate in the highway right-most lane, it can still be considered as delimited from the merging purpose. In fact, in order to be considered successful, a merging should be completed before reaching the off-ramp.
The traffic in the considered area is monitored by 8 synchronized video cameras mounted at the top of an adjacent building, providing a top view of the scene (Figure 11).

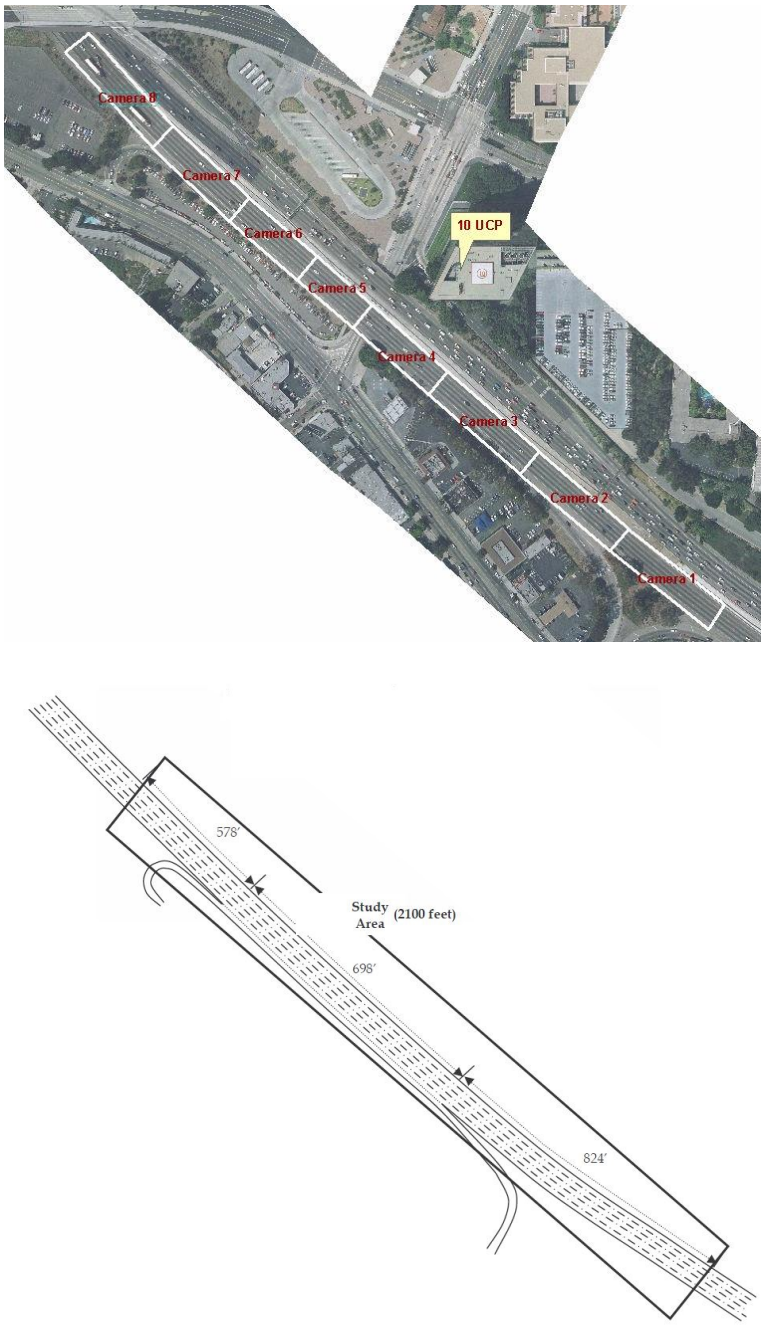


Figure 10: Highway section and camera coverage

Figure 11: Camera mounted on top of a building adjacent to the highway

The dataset corresponds to 45 minutes of recording, organized in 3 sub-datasets containing 15 minutes of recorded data each. The total time studied time period extends from 7.30 a.m. to 8.35 a.m. of June 15th, 2005. The actual dataset contains multiple information, updated each 0.1 seconds, about every vehicle in the scene, such as its position, geometry, instant velocity and acceleration, current lane, type of vehicle (motorcycle, car, truck).

The total count of merging maneuvers performed in the considered time period is 280.

5.2.3 Neural Network Configuration

The Neural Network configuration chosen to represent π^{BC} is a fully-connected 5 layers Neural Network, with 64 nodes in the hidden layers and Exponential Linear Unit (ELU) activation function in all layers other than the output layer. The Mean Squared Error (MSE) has been selected as loss function to optimize and the Adam optimizer has been used. A learning rate of 0.001 allows to achieve the best results. For every state s, the policy π^{BC} outputs a pair $\left(a_x^{BC}(s), a_y^{BC}(s)\right)$ corresponding to the lateral and longitudinal displacements. Positive values of $a_x^{BC}(s)$ and $a_y^{BC}(s)$ result respectively in moving towards the left lane (to merge) and in moving forward.

5.2.4 Safety Filter Design

The safety filter is the key element for ensuring safety and its design is crucial for the good outcome of the learned policy. The following description of the safety controller allowed to optimize the results.

The safety controller operates when an unsafe state, as defined in (Equation 5.2) is detected. Different unsafe situations fall under this general definition, and each of them requires a different control action, that cannot be fixed and predefined but needs to reflect the level of dangerousness of the situation.

A possible implementation is proposed in the following.

$$\pi^{safe}(s) \triangleq \begin{cases} \left(-\alpha \frac{\delta_{emergency}}{d_{min}(s)}, a_y^{BC}(s) \right) & \text{if } d_{min}(s) < \delta_{emergency} \\ \pi^{gap}(s) & \text{otherwise} \end{cases} \tag{5.3}$$

$$\pi^{gap}(s) \triangleq \begin{cases} (0, a_y^{BC}(s)) & \text{if } d_{gap}(s) < \delta_{gap} \\ \pi^{merge}(s) & \text{otherwise} \end{cases} \tag{5.4}$$

$$\pi^{merge}(s) \triangleq \begin{cases} (-m_b \frac{d_{min}(s)}{\delta_{dist}} + q_b)(a_x^{BC}(s), a_y^{BC}(s)) & \text{if } d_{min}(s) = d_{back}(s) \\ (m_f \frac{d_{min}(s)}{\delta_{dist}} + q_f)(a_x^{BC}(s), a_y^{BC}(s)) & \text{if } d_{min}(s) = d_{front}(s) \end{cases} \tag{5.5}$$

where α is an a positive real value; $\delta_{emergency}$ is a threshold indicating the minimum acceptable distance between the ego vehicle and every other vehicle; d_{gap} and δ_{gap} are respectively the current length of the gap where the ego vehicle is trying to merge (illustrated in Figure 12) and its minimum acceptable value that allows to safely complete the maneuver; m_b, m_f, q_b, q_f are parameters that can be adjusted to regulate the strength of the controlling action operated by π^{merge}; d_{front} and d_{back} are the distances between the closest points of ego car and front and back cars respectively. The front and back car are the ones defining the length of the gap where the ego vehicle is trying to merge, as shown in Figure 12.

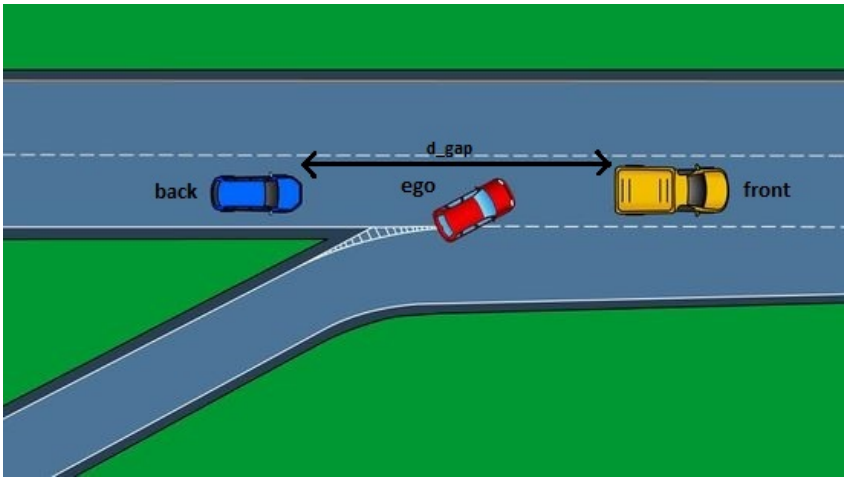


Figure 12: Scheme of the merging scenario

A overall scheme representing the operations performed by the resulting policy is illustrated in Figure 13.

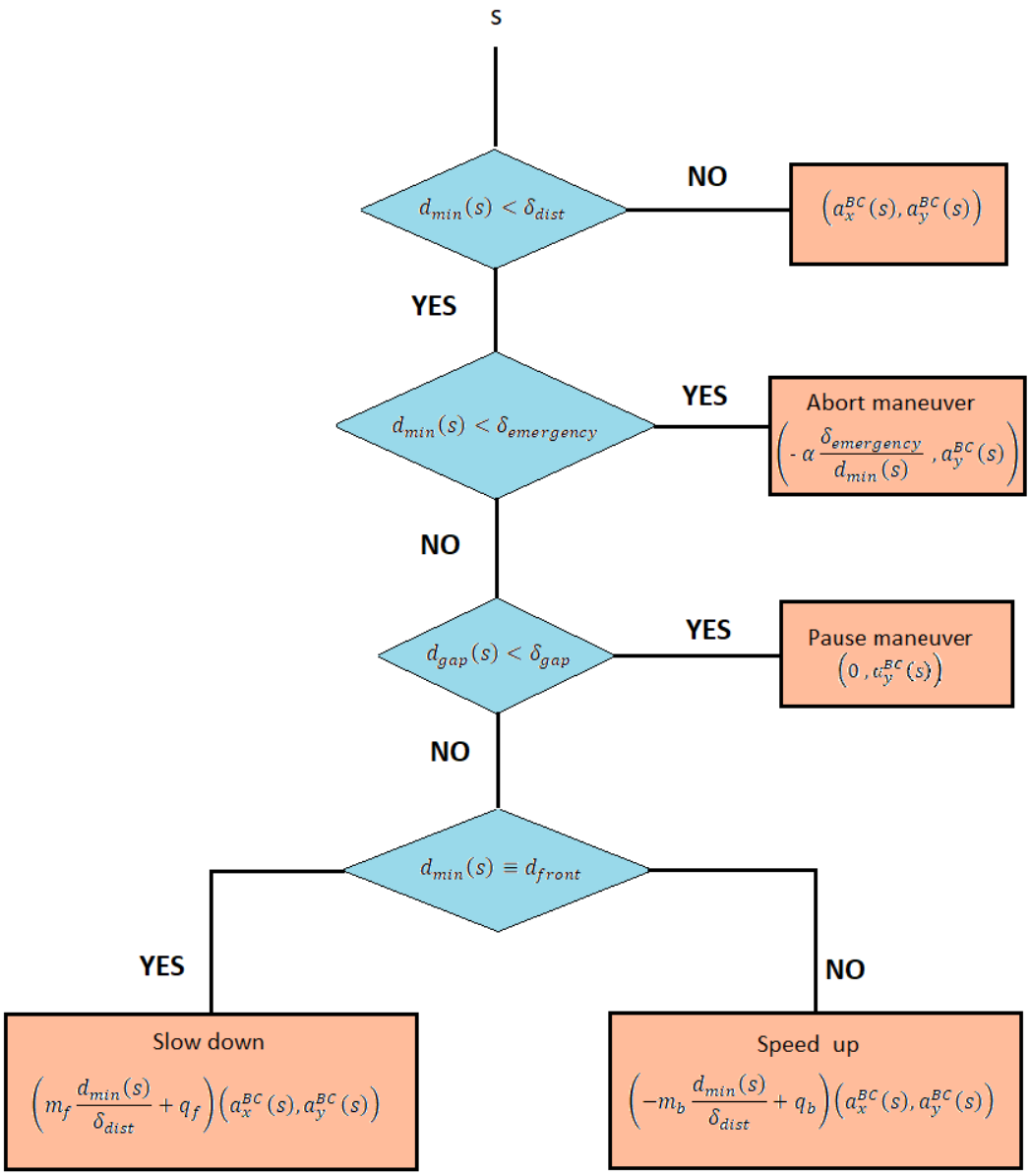


Figure 13: Scheme of the overall policy

The safe controller π^{safe} only activates when a minimum distance d_{min} between the ego car and every other car reaches the threshold δ_{dist}. In such case, s is defined to be in the unsafe set $\mathcal{S}_{unsafe}$ and a hierarchical controller is initiated.

When d_{min} is detected to be below a critical threshold $\delta_{emergency}$, the emergency maneuver is activated. Forcing a negative value of a_x^{safe}, the vehicle is driven away from the left lane, with an action whose strength is inversely proportional to the distance d_{min}, determining the merging maneuver to abort.

When the d_{min} is above the critical value $\delta_{emergency}$, but the length of the gap the car is trying to merge in is detected to be below the allowed threshold δ_{gap}, the maneuver is not aborted but paused preventing the car from further moving laterally. In this situation, if the front and back cars respond positively to the attempt to merge of the ego car, their distance d_{gap} will increase allowing the car to merge safely, otherwise, in absence of their cooperation, the gap will narrow causing the abort of the merging.

Being d_{min} greater than $\delta_{emergency}$ and being the gap large enough to preserve safety, the controlled car is still in an unsafe situation, given than $d_{min}(s) < \delta_{dist}$, which caused π^{safe} to activate. In this circumstance, the attempted merging is causing the ego vehicle to be too close (below the detection threshold δ_{dist}) to either the front or the back car. The actions operated in this situation do not abort or pause the merging, but support it to make it happen in a safer way. In this situation, we check whether $d_{min}(s) = d_{back}(s)$ or $d_{min}(s) = d_{front}(s)$, to understand which one of the two distances activates the safety controller and, based on that, identify the appropriate action to take. The idea is to slow down the maneuver when the

unsafe distance d_{min} that activated the safe policy corresponds to the distance from the front car, while speeding up the operations when it coincides with the distance from the back car. In both cases, the intensity of the acceleration (or deceleration) is determined taking into account the actual value of d_{min} and its closeness to the threshold. When $d_{min} = \delta_{dist}$ the acceleration (or deceleration) is moderate, given by the multiplicative factor $-m_b + q_b$ (or $m_f + q_f$), and it increases as d_{min} becomes smaller.

5.2.5 Results

Running the switching policy algorithm results in satisfactory results. As expected, when the ego vehicle keeps a reasonable safety distance from all the other cars, the merging is fully controlled by the policy π^{BC} parametrized with the weights of the trained Neural Network. This is the case the majority of the times, thanks to the satisfactory performance of the Neural Network training.

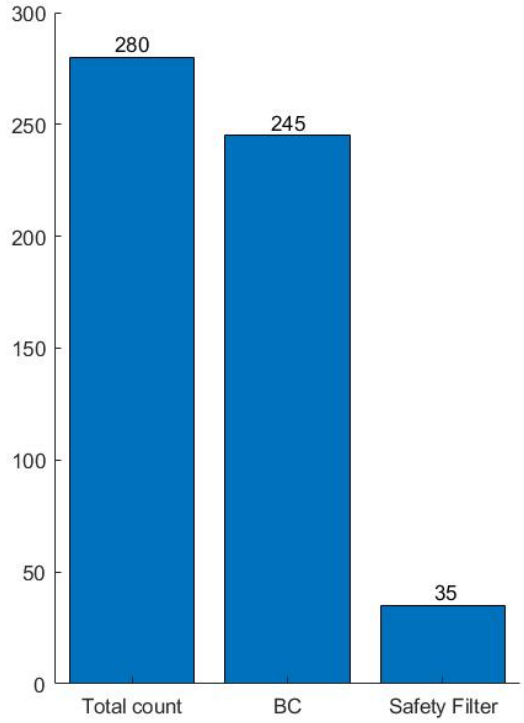
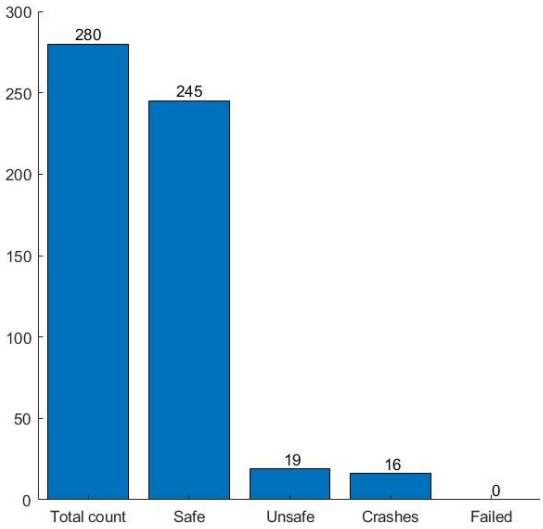


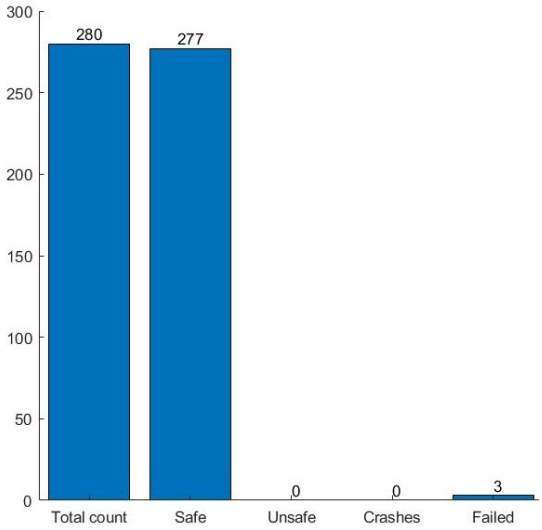
Figure 14: Safety filter activation rate

Over a total of 280 merging maneuvers, the safety filter activates 35 times, leading to an activation percentage of 12.5%.

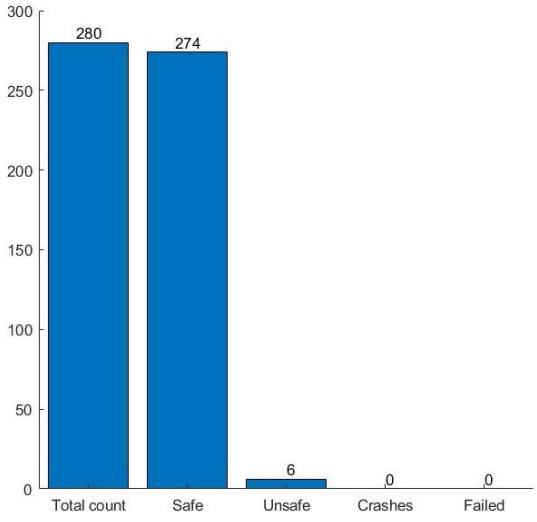
Although the Behavior Cloning alone performs safely and effective the majority of the times, the 35 activations of the safety filter denotes situations where following π^{BC} alone would be unsafe. The safety policy π^{safe} compensates for those situations leading to an overall safe policy, as the data in Figure 15 show.



(a) Behavioral Cloning



(b) With safety augmentation



(c) Expert demonstrations

Figure 15: Comparison of BC only, BC with safety guarantees and expert demonstrations

The augmentation of the Behavioral Cloning approach with safety guarantees results in a substantial improvement of the performances, that do not include any crash or unsafe merging. On the other hand, when including safety guarantees, 3 of the 280 cases result in a failed merging, meaning that the car does not manage to merge before the off-ramp. Those results highlight a propensity of the method for a slightly conservative approach, that prefer not to merge rather than doing it unsafely.

From the safety point of view, our resulting policy performs better than the expert, with 0 unsafe maneuvers against 6 of the expert, that correspond to cases where the human driver adopts a less conservative driving style that an autonomous driving policy is not advisable to have.

In Figure 16 is captured a situation that gives evidence of the safer behavior adopted by the controlled cars with respect to the the actual behavior of the human drivers.

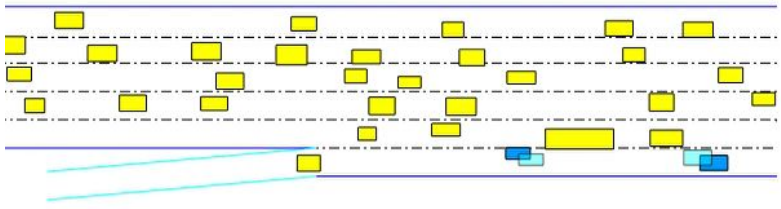


Figure 16: Example of safer behavior of our policy wrt human drivers. Blue cars represent the controlled cars while the light blue shadows represent the actual cars from the dataset

As the Figure illustrates, the controlled cars are able to keep a safer distance from the front and back cars while merging.

CHAPTER 6

CONCLUSIONS

This thesis proposed an alternative approach to the state-of-the-art of Imitation Learning approaches. While IRL methods have received large consideration in modern literature, many IRL algorithm are suitable for simple applications but impossible to scale to complex real-world environment, such as autonomous driving, due to their computational cost. Recovering the reward function, in fact, entails the need of fully solving, for each iteration where a new reward function is found, a "forward" Reinforcement Learning problem, i.e. finding the solution of the MDP for the specific reward. Besides the high demand of computational resources, this operation also requires the knowledge of transition model of the environment, which limits its implementation in settings whose transition dynamics is difficult to estimate.

On the other hand, Behavioral Cloning algorithms overcome those limitations by skipping the step of recovering a reward function. While opening the way to a practical implementation in more complex scenarios, this solution may comport some disadvantages in situations where the transferability is particularly relevant. When the purpose is to develop an imitation learning algorithm that is robust against transfer across different settings, the reward function is the most parsimoniuos representation of the expert behavior. However, this consideration does not constitute a substantial limitation in cases where significant variations in the environment are not present. In those cases, under the assumption of collecting expert data that are satisfactory explanatory of the wanted behavior and contain the highlights of the environment configuration,

a direct mapping from states to actions can be considered as the most suitable representation of the task.

Besides eliminating important disadvantages of IRL methods, Behavioral Cloning brings a series of drawbacks. Essentially, a small error may drive the agent to a state the expert never visited and from where it has no data to assist its recovery. This requires a large amount of expert demonstration to provide sufficient variability to support the learning phase. This aspect prevents its application to safety critical control tasks.

In this work we propose a method to benefit from the BC advantages, such as the low computational efficiency and the scalability to unknown dynamics, while compensating for its disadvantages, such as the cascading error, with an external component that provides reliable safety guarantees. The designed safety filter monitors the level of hazard of the current state and, based on this evaluation, decides the proper operation to actuate to efficiently complete the task while remaining in safe circumstances. When no dangers are detected, the actions dictates by the BC policy are implemented without modifications. Instead, when an unsafe situation arises and the cascading error issue is more likely to lead the agent to catastrophic scenarios, those actions are modified so that to assist the agent in its recovery towards safe states.

We applied the BC method with safety augmentation to the task of guide a self-driving car in a safe merging maneuver all the way from the on-ramp through the merge completion in the right-most lane, described in Chapter 5. Although satisfactory results are achieved in the presented experiment, they also highlight some imperfections that give room for improvements. Two main limitations can be identified:

- Absence of the responsiveness of the other vehicles. The controlled vehicles are only the ones that come from the on-ramp of the highway an have the intent of merging in the lane next to the transit corridor. For all the other cars, no control action is performed. Their position, velocity and acceleration is kept unchanged and taken from the NGSIM dataset. This constitutes a limitation when the vehicle under control decides to take actions differently from what has been done by the human driver controlling the same car. The surrounding cars play an important role in the decision-making process, as their behavior is a manifestation of their intention to yield or not to the merging vehicle. Therefore, the lack of responsiveness of the other vehicles lead to a missing cooperation between drivers, that is fundamental for an interactive driving maneuver.
- Limited expert demonstrations. Although the NGSIM dataset used in this experiment is a useful resource for studying the highway traffic, its focus is on collecting data of merging example specifically. Since the merging maneuver occurs less frequently than regular lane following, this results in a large amount of data for the cars traveling the regular highway lanes and few data for the ones coming from the on-ramp. Having more data would significantly improve the performance of the model obtained by regression and, therefore, would lead to a better imitation policy, that would require switching to the safe policy less frequently.

Overcoming those limitations would lead to better overall performances.

To compensate for the lack of cooperation, the NGSIM dataset could be used only for training, while implementing a lane following algorithm to control the surrounding cars. It would generate

an adjustable traffic situation, where all the vehicles react to the presence of the merging cars. To make up for the second limitation, instead, a more extensive data collection procedure could be carried on, with particular focus on the merging vehicles.

Both suggested improvements and improved implementations of the safety filter are left for future work.

The code of this work is available at `https://github.com/edales3/safe-BC`.

1. Pieter Abbeel and Andrew Y. Ng. *Apprenticeship learning via inverse reinforcement learning.* ICML, 2004.

2. Alekh Agarwal, Nan Jiang and Sham M. Kakade. *Reinforcement Learning: Theory and Algorithms.* 2019.

3. Martin Arjovsky, Soumith Chintala and Leon Bottou. *Wasserstein Generative Adversarial Networks.* ICML, 2017.

4. J. Andrew Bagnell, Andrew Y. Ng and Jeff G. Schneider. *Solving Uncertain Markov Decision Processes.* 2001.

5. Mayank Bansal, Alex Krizhevsky and Abhijit Ogale. *ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst.* CoRR, 2018.

6. Richard Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

7. Dimitri P. Bertsekas. *Dynamic programming and optimal control, volume I.* MIT, Athena Scientific Belmont, Massachusetts, 2005.

8. Dimitri P. Bertsekas. *Dynamic programming and optimal control, volume II.* MIT, Athena Scientific Belmont, Massachusetts, 2007.

9. Lionel Blondé and Alexandros Kalousis. *Sample-Efficient Imitation Learning via Generative Adversarial Nets.* AISTATS, 2019.

10. Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

11. Qi Cai, Mingyi Hong, Yongxin Chen and Zhaoran Wang. *On the Global Convergence of Imitation Learning: A Case for Linear Quadratic Regulator.* CoRR, 2019.

12. Minshuo Chen, Yizhou Wang, Tianyi Liu, Zhuoran Yang, Xingguo Li, Zhaoran Wang and Tuo Zhao. *On Computation and Generalization of Generative Adversarial Imitation Learning.* ArXiv, 2020.

13. Glen Chou, Dmitry Berenson and Necmiye Ozay. *Learning Constraints from Demonstrations.* CoRR, 2018.

14. Maryam Fazel, Rong Ge, Sham M. Kakade and Mehran Mesbahi. *Global Convergence of Policy Gradient Methods for the Linear Quadratic Regulator.* CoRR, 2018.

15. Federal Highway Administration Research and Technology, NGSIM dataset, US Highway 101 Dataset. `https://www.fhwa.dot.gov/publications/research/operations/07030/index.cfm`

16. Chelsea Finn, Paul Christiano, Pieter Abbeel and Sergey Levine. *A Connection between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models.* CoRR, 2016.

17. Chelsea Finn, Sergey Levine and Pieter Abbeel. *Guided cost learning: Deep Inverse Optimal Control via Policy Optimization.* ICML, 2016.

18. Justin Fu, Katie Luo and Sergey Levine. *Learning Robust Rewards with Adversarial Inverse Reinforcement Learning.* ICLR, 2018.

19. Davide Gagliardi and Giovanni Russo. *On a probabilistic approach to synthesize control policies from example datasets.* ArXiv: Optimization and Control, 2020.

20. Hongbo Gao, Guanya Shi, Guotao Xie and Bo Cheng. *Car-following method based on inverse reinforcement learning for autonomous vehicle decision-making.* International Journal of Advanced Robotic Systems, 2018.

21. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville and Yoshua Bengio. *Generative Adversarial Networks.* NIPS, 2014.

22. Peter Henderson, Wei-Di Chang, Pierre-Luc Bacon, David Meger, Joelle Pineau and Doina Precup. *OptionGAN: Learning Joint Reward-Policy Options Using Generative Adversarial Inverse Reinforcement Learning.* AAAI, 2018.

23. Jonathan Ho and Stefano Ermon. *Generative adversarial imitation learning.* NIPS, 2016.

24. Jonathan Ho, Jayesh K. Gupta and Stefano Ermon. *Model-free imitation learning with policy optimization.* ICML, 2016.

25. Seyed Kamyar Seyed Ghasemipour, Shane Gu and Richard Zemel. *Understanding the Relation Between Maximum-Entropy Inverse Reinforcement Learning and Behaviour Cloning.* ICLR, 2019.

26. Liyiming Ke, Sanjiban Choudhury, Matt Barnes, Wen Sun, Gilwoo Lee and Siddhartha Srinivasa. *Imitation Learning as f-Divergence Minimization.* CoRR, 2020.

27. Kee-Eung Kim and Hyun Soo Park. *Imitation Learning via Kernel Mean Embedding.* 2018, Association for the Advancement of Artificial Intelligence (AAAI), 2018.

28. Markus Kuderer, Shilpa Gulati and Wolfram Burgard. *Learning driving styles for autonomous vehicles from demonstration.* 2015 IEEE International Conference On Robotics And Automation (ICRA), 2015.

29. Alex Kuefler, Jeremy Morton, Tim Wheeler and Mykel Kochenderfer. *Imitating Driver Behavior with Generative Adversarial Networks.* 2017 IEEE Intelligent Vehicles Symposium (IV), 2017.

30. Yunzhu Li, Jiaming Song and Stefano Ermon. *InfoGAIL: Interpretable Imitation Learning from Visual Demonstrations.* NIPS, 2017.

31. Andrew Y. Ng and Stuart J. Russell. *Algorithms for Inverse Reinforcement Learning.* ICML, 2000.

32. Arnab Nilim and Laurent El Ghaoui. *Robust Control of Markov Decision Processes with Uncertain Transition Matrices.* Operations Research, 2005.

33. Sebastian Nowozin, Botond Cseke and Ryota Tomioka. *f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization.* ArXiv, 2016.

34. Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, Pieter Abbeel and Jan Peters. *An Algorithmic Perspective on Imitation Learning.* Foundations and Trends in Robotics, 2018.

35. Dean A. Pomerleau. *ALVINN: An Autonomous Land Vehicle in a Neural Network.* NIPS, 19888.

36. Martin L. Puterman. *Markov Decision Processes, Discrete Stochastic Dynamic Programming.* University of British Columbia, 2004.

37. Nathan D. Ratlif, J. Andrew Bagnell and Martin A. Zinkevich. *Maximum Margin Planning.* ICML, 2006.

38. Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn and Sergey Levine. *End-to-End Robotic Reinforcement Learning without Reward Engineering.* CoRR, 2019.

39. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction, volume I.* MIT press Cambridge, 1998.

40. Umar Syed, Michael Bowling and Robert E. Schapire. *Apprenticeship Learning Using Linear Programming.* ICML, 2008.

41. Faraz Torabi, Garrett Warnell and Peter Stone. *Generative Adversarial Imitation from Observation.* ICML, 2019.

42. U.S. Department of Transportation, National Highway Traffic Safety Administration. *Analysis of Lane Change Crashes.* `https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/doths809571.pdf` 2003.

43. Pin Wang, Ching-Yao Chan and Arnaud de La Fortelle. *A Reinforcement Learning Based Approach for Automated Lane Change Maneuvers.* 2018 IEEE Intelligent Vehicles Symposium (IV), 2018.

44. Pin Wang, Dapeng Liu, Jiayu Chen, Hanhan Li and Ching-Yao Chan. *Human-like Decision Making for Autonomous Driving via Adversarial Inverse Reinforcement Learning.* ArXiv: Artificial Intelligence, 2020.

45. Yisong Yue and Hoang M. Le. Imitation Learning, ICML 2018 Tutorial. https://www.youtube.com/watch?v=WjFdD7PDGw0

46. Yufeng Zhang, Qi Cai, Zhuoran Yang and Zhaoran Wang. *Generative Adversarial Imitation Learning with Neural Networks: Global Optimality and Convergence Rate.* ArXiv, 2020.

47. Brian D. Ziebart *Modeling purposeful adaptive behavior with the principle of maximum causal entropy.* PhD thesis, Carnegie Mellon University, 2010.

48. Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell and Anind K. Dey. *Maximum Entropy Inverse Reinforcement Learning.* AAAI, 2008.