FEDERAL UNIVERSITY OF SANTA CATARINA
TECHNOLOGICAL CENTER
GRADUATE PROGRAM IN AUTOMATION AND SYSTEMS ENGINEERING

Irving Giovani Bronzatti Petrazzini

Imitation Learning for Autonomous Driving: Disagreement-Regularization and Behavior Cloning with Beta Distribution

Florianópolis
2023

Irving Giovani Bronzatti Petrazzini

Imitation Learning for Autonomous Driving: Disagreement-Regularization and Behavior Cloning with Beta Distribution

Thesis submitted to the Graduate Program in Automation and Systems Engineering from the Federal University of Santa Catarina for obtaining the Master Degree in Systems and Automation Engineering.
Supervisor: Prof. Eric Aislan Antonelo, PhD.

Florianópolis
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Petrazzini, Irving Giovani Bronzatti
Imitation Learning for Autonomous Driving: Disagreement Regularization and Behavior Cloning with Beta Distribution / Irving Giovani Bronzatti Petrazzini ; orientador, Eric Aislan Antonelo, 2023.
111 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2023.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Aprendizado por Imitação. 3. Aprendizado por Reforço. 4. Aprendizado por Imitação com Regularização de Desacordo. 5. Condução Autônoma. I. Antonelo, Eric Aislan . II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

Irving Giovani Bronzatti Petrazzini

Imitation Learning for Autonomous Driving: Disagreement-Regularization and Behavior Cloning with Beta Distribution

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Kianté Brantley, Dr.
Cornell University

Prof. Paulo Fernando Ferreira Rosa, Dr.
Instituto Militar de Engenharia

Prof. Rodrigo Clemente Thom de Souza, Dr.
Universidade Federal do Paraná

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de Mestre em Engenharia de Automação e Sistemas.

Documento assinado digitalmente
Julio Elias Normey Rico
Data: 10/10/2023 15:55:37-0300
CPF: ***.840.859-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Júlio Elias Normey Rico, Dr.
Coordenação do Programa de
Pós-Graduação em Automação e Sistemas

Documento assinado digitalmente
Eric Aislan Antonelo
Data: 10/10/2023 15:26:52-0300
CPF: ***.163.409-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eric Aislan Antonelo, Dr.
Orientador

Florianópolis, 25 de setembro de 2023.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my dissertation advisor, Prof. Eric Aislan Antonelo, PhD, for his unwavering guidance, support, and invaluable mentorship throughout the course of my master's dissertation. His expertise, patience, and commitment to excellence have played a crucial role in shaping my research journey.

I would also like to extend my heartfelt thanks to my family, including my parents Idelmir and Rejane, and my brother Igor, for their constant encouragement, love, and understanding. Their steadfast support has been my pillar of strength, enabling me to pursue my academic aspirations with dedication and enthusiasm.

Finally, I wish to acknowledge the numerous authors, researchers, and institutions whose work and resources have enriched my research and expanded my knowledge base.

This dissertation would not have been possible without the support and encouragement of all these individuals, and for that, I am sincerely grateful.

ABSTRACT

Autonomous driving is a challenging problem, since its environment has an open-ended nature with unexpected, critical events that can take place. Imitation Learning (IL) approaches have become dominant for end-to-end autonomous driving not only in academia but also in companies which provide autonomous driving services. In this approach, an expert generates trajectories of observation-action pairs, demonstrating the desired behavior to a computational learning agent. Behavior cloning is the simplest form of IL, where a neural network is trained offline and only once before it is deployed in the environment. Other approaches are interactive, providing an online learning through trial and error in the environment. In this work, we explore one of such approaches: the Disagreement-Regularized Imitation Learning (DRIL), which leverages an ensemble of policies trained to overfit the expert set through behavior cloning. The disagreement in the ensemble, which can be calculated by the variance of policies, indicates if a given state is distant from the states seen by the expert. This can be used to derive a reward signal, facilitating a closed-loop training approach. This work elaborates on different ways of employing DRIL, specially in the autonomous driving scenario, characterized by both high-dimensional observation spaces, such as images, and continuous action spaces. By employing a method analogous to early-stopping, DRIL has demonstrated superior performance compared to results reported by other imitation learning methods in a top-down racing environment. Finally, experiments have shown that a policy trained with behavior cloning alone in that environment and modeling a Beta distribution instead of the standard Gaussian one has shown to offer a competitive alternative in addition to a faster training process.

Keywords: Imitation Learning. Autonomous Driving. Disagreement-Regularized Imitation Learning. Reinforcement Learning.

RESUMO

A condução autônoma de veículos é um problema desafiador, pois seu ambiente possui uma natureza aberta com eventos inesperados e críticos que podem ocorrer. Abordagens de Aprendizado por Imitação (IL) tornaram-se dominantes para a condução autônoma de ponta a ponta, não apenas na academia, mas também em empresas que fornecem serviços de condução autônoma. Nesta abordagem, um especialista gera trajetórias de pares observação-ação, demonstrando o comportamento desejado a um agente aprendiz. A clonagem comportamental é a forma mais simples de IL, onde uma rede neural é treinada "offline" e apenas uma vez antes de interagir com o ambiente. Outras abordagens são interativas, proporcionando um aprendizado online por tentativa e erro no ambiente. Neste trabalho, exploramos uma dessas abordagens: o Aprendizado por Imitação com Regularização por Desacordo (DRIL), que utiliza um conjunto de políticas treinadas para sobreajustar o conjunto de especialistas por meio da clonagem comportamental. O desacordo no conjunto, que pode ser calculado pela variância das políticas, indica se um certo estado está distante dos estados visitados pelo especialista. Isso pode ser usado para derivar um sinal de recompensa, permitindo uma abordagem de treinamento em ciclo fechado. Este trabalho elabora diferentes maneiras de empregar o DRIL, especialmente no cenário de condução autônoma, caracterizado por espaços de observação de alta dimensão, como imagens, e espaços contínuos de ação. Ao empregar um método análogo à interrupção precoce ("early-stopping"), o DRIL demonstrou um desempenho superior em comparação com os resultados relatados por outras abordagens de aprendizado por imitação em um simulador de carro autônomo de vista superior. Finalmente, experimentos mostraram que uma política estocástica treinada naquele ambiente apenas com a clonagem comportamental utilizando uma distribuição Beta, em vez da Gaussiana padrão, demonstrou oferecer uma alternativa competitiva, além de um processo de treinamento mais rápido.

Palavras-chave: Aprendizado por Imitação. Condução Autônoma. Aprendizado por Imitação com Regularização de Desacordo. Aprendizado por Reforço.

RESUMO EXPANDIDO

Introdução

A Inteligência Artificial (IA) testemunhou avanços notáveis nos últimos anos, transformando inúmeros domínios e indústrias por meio de sua capacidade de replicar funções cognitivas semelhantes às humanas. Entre seus inúmeros subcampos, a aprendizagem por imitação surgiu como um paradigma fundamental, permitindo que as máquinas adquiram comportamentos intricados imitando demonstrações de especialistas. Posteriormente, a aprendizagem por reforço ganhou destaque ao superar o desempenho humano em certas tarefas de tomada de decisão.

Objetivos

Nesta dissertação, mergulhamos em um método conhecido como Aprendizagem por Imitação com Regularização de Desacordo (DRIL). Este método tem como objetivo superar algumas das limitações tanto da Aprendizagem por Imitação quanto da Aprendizagem por Reforço, sintetizando suas abordagens. Aplicamos este método a uma versão simplificada de um problema do mundo real: a Condução Autônoma.

Metodologia

A metodologia aplicada foi dividida em quatro estágios. No primeiro estágio, reproduzimos os resultados de (Brantley; Sun; Henaff, 2020), utilizando Atari Breakout como ambiente discreto e Lunar Lander Continuous como ambiente de controle robótico. No Breakout, o algoritmo DRIL é capaz de reproduzir níveis de desempenho de especialistas com apenas uma demonstração de especialista. No ambiente Lunar Lander Continuous, o DRIL apresenta desempenho semelhante a clonagem comportamental (BC), com ambas as políticas ultrapassando o nível mínimo de especialistas. No segundo estágio, aplicamos DRIL à condução autônoma no ambiente CarRacing-v0. Este cenário apresenta características especificas e não contempladas por (Brantley; Sun; Henaff, 2020), como entradas de imagens concatenadas e ações contínuas, exigindo uma combinação das estruturas de políticas do Atari e do Controle Robótico. Demonstrações de especialistas são vitais, mas nenhuma estava disponível para o CarRacing-v0. Portanto, exploramos a criação de um especialista usando *Proximal Policy Optimization* (PPO). Apesar das trajetórias fornecidas por um agente de ponta, a avaliação revelou que as políticas treinadas com DRIL têm desempenho inferior às treinadas com BC.No terceiro estágio, apresentamos um algoritmo para DRIL com interrupção precoce e conduzimos experimentos no ambiente CarRacing. Os resultados mostram que a estratégia de interrupção precoce melhora significativamente o desempenho em comparação com simples BC, especialmente ao usar uma política estocástica. No entanto, o processo de treinamento sem interrupção precoce não supera consistentemente o BC. Armazenar os parâmetros da política no pico de desempenho durante o treinamento (DRIL-peak) ajuda a mitigar o problema do esquecimento catastrófico

durante o treinamento e forneceu os melhores resultados entre todos os experimentos. No quarto estágio, introduzimos o algoritmo DRIL-Beta, enfatizando o uso da distribuição Beta no loop do PPO do DRIL. Neste estágio endereçamos as limitações da distribuição Gaussiana no PPO fazendo modificações na estrutura da política, isto é, incluindo duas distribuições Beta independentes. Também abordamos o BC com a distribuição Beta, como parte da composição do DRIL e para fins de comparação.

Resultados e Discussão
Em nosso estudo, avaliamos três métodos de aprendizado por imitação (BC, DRIL-peak e DRIL-final) com variações em distribuições Gaussianas e Beta, aplicados a conjuntos de dados de especialistas agrupados em conjuntos de 1 e 20 no cenário de condução autônoma no CarRacing. Os resultados mostraram que o desempenho variou entre os métodos e conjuntos de dados, destacando a influência da escolha do método e do tamanho do conjunto de dados no aprendizado por imitação na condução autônoma. O melhor desempenho foi alcançado pela política estocástica DRIL-peak com distribuição Gaussiana obteve as pontuações mais altas em conjuntos de dados com 20 trajetórias (802$\pm$197). Por fim, uma política BC estocástica com distribuição Beta (794$\pm$227) é proposta como uma alternativa competitiva à solução ótima encontrada anteriormente usando o DRIL-peak.

Considerações Finais
Esta dissertação contribui com a literatura ao expandir o algoritmo DRIL para um espaço de observação de pixels com um espaço de ação contínuas e introduzir uma abordagem de 'parada antecipada' durante o treinamento (DRIL-peak). Além disso, para gerar trajetórias ótimas, desenvolvemos uma política estocástica treinada com PPO usando uma distribuição Beta, alcançando resultados no estado da arte na resolução do ambiente CarRacing-v0. Em pesquisas futuras, vislumbramos várias promissoras oportunidades para expandir a aplicação do DRIL no campo de direção autônoma. Isso inclui aplicar o DRIL em simuladores avançados de direção autônoma, como CARLA; explorar várias arquiteturas de redes neurais artificiais, como aquelas com mecanismos de atenção, para aprimorar a eficácia do DRIL; experimentar o DRIL em conjunto com métodos de aprendizado por reforço mais robustos; testar um conjunto em que cada política é treinada como uma política Beta em vez de uma política Gaussiana; avaliar abordagens alternativas para detectar estados que se desviam das distribuições de especialistas, como usar um conjunto de políticas com arquiteturas diversas.

Palavras-chave: Aprendizado por Imitação. Condução Autônoma. Aprendizado por Imitação com Regularização de Desacordo. Aprendizado por Reforço.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Artificial Intelligence (AI) has witnessed remarkable advancements in recent years, transforming numerous domains and industries through its ability to replicate human-like cognitive functions. Among its myriad subfields, imitation learning has emerged as a pivotal paradigm, enabling machines to acquire intricate behaviors by imitating expert demonstrations. Later on, reinforcement learning has made headlines by surpassing human-level performance in certain decision-making tasks.

In this dissertation, we delve into a method known as Disagreement-Regularized Imitation Learning. This method aims to overcome some of the limitations of both Imitation Learning and Reinforcement Learning by synthesizing their approaches. We applied this method to a simplified version of a real-world problem: Autonomous Driving.

1.1 MOTIVATION

In 2014, the Society of Automotive Engineers (SAE) introduced a classification comprising six levels that delineate motor vehicle driving automation systems (SAE, 2014). These levels signify the extent to which autonomous systems engage in the dynamic driving task on a continuous basis, as per their designated level. To illustrate, a Level 0 system offers no automation, demanding the driver to take charge of all actions. In contrast, a Level 5 system is fully capable of driving the vehicle across all conditions, obviating the need for driver intervention. For the purposes of this dissertation, the term "autonomous driving" explicitly refers to the Level 5 system as defined by SAE.

Long before the formal definition and classification of autonomous driving, researchers have been striving to automate the process of driving. Research in autonomous driving primarily follows two main paradigms: modular pipelines and end-to-end driving. Modular pipelines divide the driving task into sub-tasks, including perception, maneuver planning, and control. In contrast, end-to-end driving approaches aim to learn a direct mapping from input raw sensor data to vehicle control signals (Xiao et al., 2019). In this thesis, we will primarily concentrate on the latter approach.

One end-to-end approach for autonomous driving is to record driving trajectories generated by an expert driver and train a policy with supervised learning to imitate the behavior of the expert. This approach, named Behavior Cloning, was first attempted by (Pomerleau, 1989). However, Behavior Cloning faces a key problem known as covariate shift, as described by (Ross, Stephane; Bagnell, D., 2010).

Covariate shift refers to the deviation from the distribution of states covered by the expert, which can lead the agent into unknown states and result in actions that may not be appropriate for such states.

Another idea following the end-to-end principle is reinforcement learning. In this class of methods, an agent interacts with the environment initially by taking random

actions. For each action, the agent receives a reward designed to encourage it to behave like a prudent human driver. However, two significant drawbacks become apparent early on in the application of reinforcement learning to autonomous driving: First, there is a substantial requirement for a large number of interactions and trial-and-error attempts with the environment. In the context of driving, errors can have very high real-world costs. Second, creating a reward function that effectively guides the agent toward the desired behavior has proven to be a challenging task (Knox et al., 2022).

In an effort to address the challenges associated with developing reward functions for reinforcement learning and mitigating the covariate shift issue in behavior cloning, (Brantley; Sun; Henaff, 2020) introduced a novel approach known as Disagreement-Regularized Imitation Learning (DRIL). This method derives a reward signal from an ensemble of policies trained using behavior cloning. The underlying idea is that these policies tend to produce similar actions in states present in the training dataset and diverge in unfamiliar states. By leveraging this agreement and disagreement among policies as a reward signal, a new policy is trained through a combination of reinforcement learning and behavior cloning, which are interleaved during the training process. We illustrate this idea in Figure 1 which depicts both an open-loop and closed-loop imitation learning training methods.

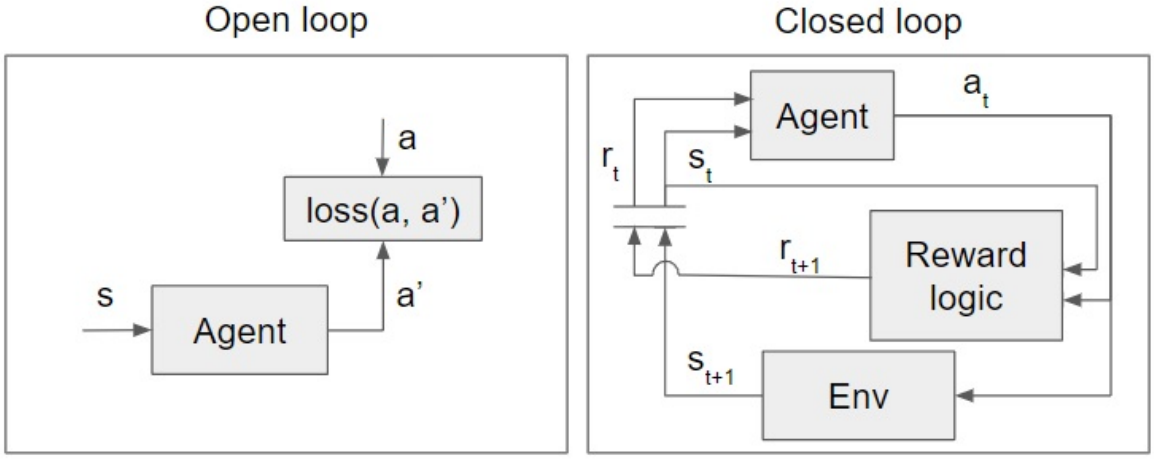

Figure 1 – Left: This illustration depicts an open-loop training method, such as BC or GAIL. For a given state-action pair (s, a) from the expert demonstrations, the agent receives the state s and takes an action a'. A loss function compares a' with a, and its output is utilized to adjust the agent's parameters so that $a' = a$. Right: In contrast, closed-loop training methods, as employed by DRIL and PWIL, involve an agent interacting with the environment (Env), where each action a_t leads the agent to a new state s_{t+1} and a certain logic provides a reward r_{t+1}. This reward is then employed to adjust the agent's parameters following a chosen reinforcement learning algorithm.

The training of Behavior Cloning and other imitation learning methods, such as Generative Adversarial Imitation Learning (Ho; Ermon, 2016), could be compared to

an open-loop control system. It is widely recognized in control theory literature that closed-loop control systems present better performance compared to open-loop systems. In that sense, DRIL could be seen as a closed-loop alternative, with respect to the training loop, to the plain behavior cloning technique. A similar approach was also tried in Primal Wasserstein Imitation Learning (Dadashi et al., 2021), in which authors used the Wasserstein distance between agent's current state and expert dataset sample to derive a reward signal.

1.2 HYPOTHESIS

DRIL proposes an evolution over the standard behavior cloning algorithms by adding a feedback loop based on the disagreement of an ensemble of policies. According to the original work, DRIL outperforms behavior cloning for discrete action spaces with image observations and matches BC for continuous action spaces with low-dimensional observation spaces.

Considering the autonomous driving problem modeled as a continuous action space with an image observation space, we have noted in our preliminary work that behavior cloning is very limited in reproducing expert performance.

Our hypothesis is: **DRIL will outperform BC in the autonomous vehicle environment**.

1.3 GENERAL OBJECTIVE

In order to evaluate the performance of the DRIL algorithm in an autonomous vehicle setting, we propose to adapt the algorithm to an environment with an image-based state and a continuous action space, configuration not yet explored in the literature. Table 1 shows a schematic comparison between previous work and our work, highlighting the scope extension.

Table 1 – Environment configuration in previous work and in our work.

Environment	Atari	Robotic Control[1]	**Autonomous Vehicles**
$\|\mathcal{S}\|$	Image	Low-dimensional	**Image**
$\|\mathcal{A}\|$	Discrete	Continuous	**Continuous**
	Previous Work [2]		**Our work**

1.4 SPECIFIC OBJECTIVES

Our research program is broken down into the following steps:

[1] Physics control environments using Box2D library have been proposed as part of OpenAI Gym by (Brockman et al., 2016) and Multi-Joint dynamics with Contact (Mujoco) have been proposed by (Todorov; Erez; Tassa, 2012)

[2] As proposed by (Brantley; Sun; Henaff, 2020)

1. Reproduce previous results obtained with DRIL
2. Develop an expert model for CarRacing[3], an autonomous vehicles simulator available as part of OpenAI gym (Brockman et al., 2016)
3. Adapt DRIL for the autonomous vehicle setting
4. Implement changes to DRIL that might improve performance, such as making the policy output to model a Beta distribution

1.5 PUBLICATIONS

The following publications are a result of this work:

- The article "Proximal Policy Optimization with Continuous Bounded Action Space via the Beta Distribution" (Petrazzini; Antonelo, Eric A., 2021) has been published at IEEE Symposium Series on Computational Intelligence 2021;
- The article "Disagreement-Regularized Imitation Learning with Early-Stopping" is under elaboration.

1.6 DOCUMENT OUTLINE

The remainder of this dissertation has been structured as follows:

Chapter 2 of the dissertation covers key methodologies used in the research: artificial neural networks (ANNs), Behavior Cloning (BC) and reinforcement learning, emphasizing Proximal Policy Optimization (PPO) and addresses modeling bounded action spaces with both infinite and finite support probability distributions. Subsequently, it delves into Disagreement-Regularized Imitation Learning (DRIL) intuition and formal presentation.

Chapter 3 of the text provides a review of the literature on Imitation Learning, Reinforcement Learning, DRIL and CarRacing. In the Imitation Learning section, it discusses Behavior Cloning, one of the earliest methods for autonomous vehicles. In the section dedicated to DRIL, the text mentions the methodology's initial publication and its citation history, underlining its relevance in the field. The subsection on CarRacing briefly summarizes studies conducted on reinforcement learning and imitation learning within the CarRacing-v0 environment.

Chapter 4 focuses on reproducing the results of the DRIL paper, using Atari Breakout as the discrete environment and Lunar Lander Continuous as the robotic control environment. In Breakout, the DRIL algorithm is capable of reproducing expert performance levels with just a single expert demonstration. In the Lunar Lander Continuous environment DRIL parallels BC performance with both policies surpassing the expert threshold.

[3] Our experiments were run on CarRacing-v0, made available at `https://www.gymlibrary.dev/environments/box2d/car_racing/` by OpenAI, accessed August 31, 2023

Chapter 5 applies DRIL to autonomous driving in the CarRacing-v0 environment. It deals with the unique characteristics of this setting, such as concatenated image inputs and continuous actions, requiring an amalgamation of Atari and Robotic Control policy structures. Expert demonstrations are vital, but none were available for CarRacing-v0. Hence, it delves into the crafting of an expert using Proximal Policy Optimization with Beta distribution (PPO-Beta). Despite trajectories provided by an state-of-the-art agent, evaluation revealed that DRIL-trained policies perform below those trained with BC

Chapter 6 presents an algorithm for DRIL with early-stopping and conducts experiments in the CarRacing environment. The results show that the early-stopping strategy significantly improves performance compared to pure Behavior Cloning (BC), especially when using a stochastic policy. However, the training process without early-stopping does not consistently outperform BC. Storing the policy parameters at the peak performance during training (DRIL-peak) helps mitigate the problem of catastrophic forgetting during training and provided the best results among all experiments.

Chapter 7 introduces the DRIL-Beta algorithm, emphasizing the use Beta distribution in DRIL's PPO loop. It addresses the limitations of the Gaussian distribution in the original PPO and outlines the policy structure modifications, including two independent Beta distributions. The chapter also covers BC with the Beta distribution, as part of the DRIL composition and for comparison purposes. Despite initial policy deterioration during DRIL training, stochastic DRIL-peak ultimately outperforms BC when applied to a dataset containing 1 expert trajectory both in deterministic and stochastic mode, showcasing the benefits of adopting the Beta distribution in DRIL. Finally, a stochastic BC policy with Beta distribution is proposed as a competitive alternative to the optimal solution found in previous chapter using DRIL-peak

Chapter 8 chapter provides a discussion of the study's results and their implications, with reference to the presented data and comparison with related research.

Chapter 9 concludes enumerating key findings and proposes future works.

Fig. 2 presents a diagram with arrows representing the interdependence of main ideal.

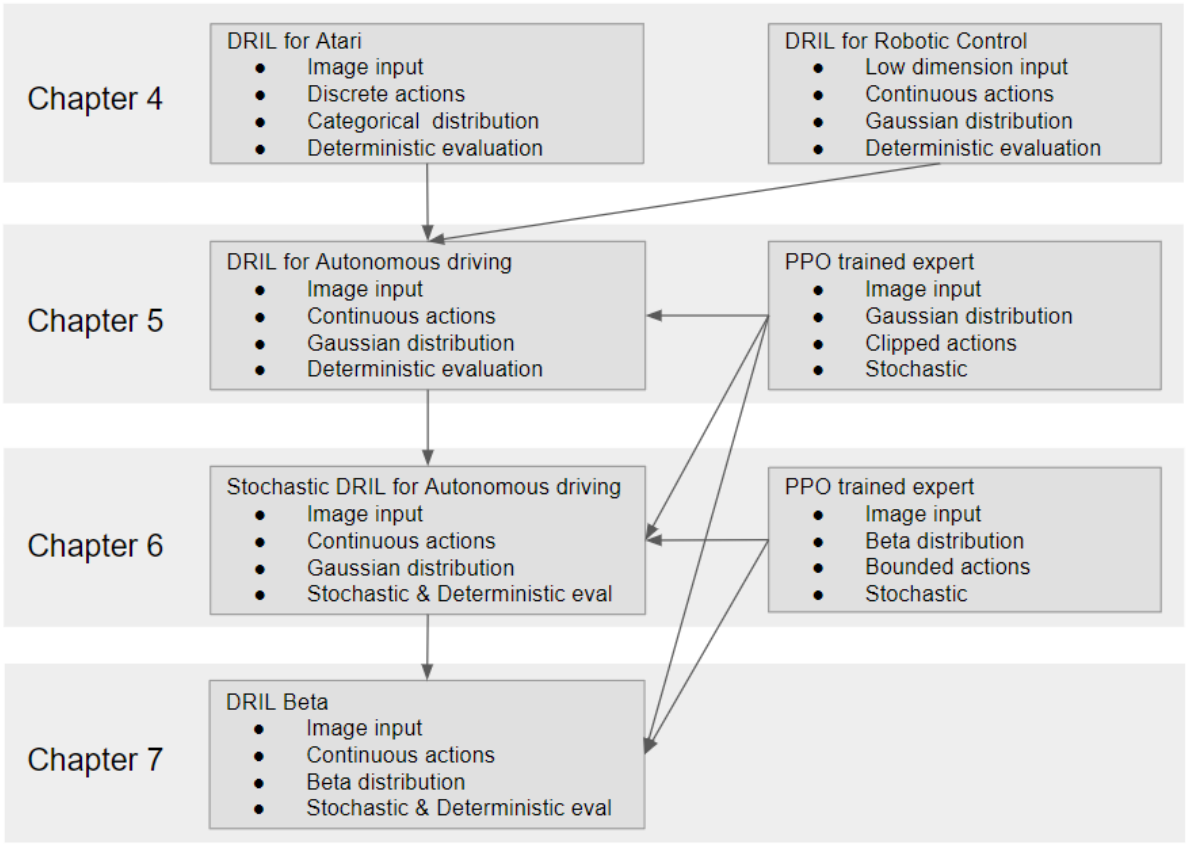

Figure 2 – Dissertation structure.

2 BACKGROUND

In this section, we review the main methodologies that were employed throughout this dissertation, in particular the Disagreement-Regularized Imitation Learning method in Sections 2.4 and 2.5.

2.1 ARTIFICIAL NEURAL NETWORKS

Artificial neural networks (ANNs) are widely used as non-linear function approximators. The appropriateness of ANNs is guaranteed by the universal approximation theorem, which states that a feedforward network with a linear output layer and at least one hidden layer with any squashing activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired nonzero amount of error, provided that the network is given enough hidden units (Hornik; Stinchcombe; White, 1989).

For the purpose of the present work, we will consider that any continuous function on a closed and bounded subset of R^n is Borel measurable and, therefore, can be approximated by a neural network (Goodfellow; Bengio; Courville, 2016).

2.1.1 Neuron Model

The basic unit of an ANN is called a perceptron. In its generic formulation, the perceptron receives n inputs. Each input i is multiplied by a certain weight w_i. The sum is then added to a constant, denoted as b.

$$z = \sum_{i=1}^{n} w_i * x_i + b \tag{1}$$

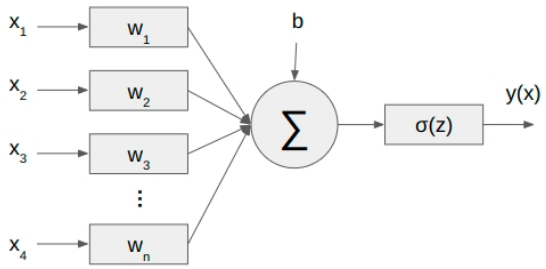

Figure 3 – Generic Neuron.

The weighted sum z is then passed through a non-linearity, which typically takes the form of a sigmoid, hyperbolic tangent, softplux, or rectified linear unit, as shown in Figure 4, producing an output y.

$$\hat{y} = \sigma(z) \tag{2}$$

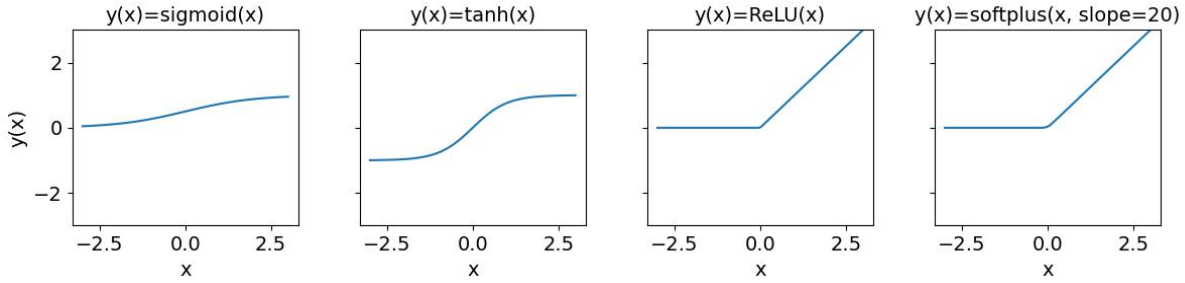

Figure 4 – Common activation functions used in artificial neurons. From left to right: Sigmoid function, hyperbolic tangent and Rectified Linear Unit and Softplus.

2.1.2 The Multi-Layer Perceptron

To fulfill the conditions necessary for an Artificial Neural Network (ANN) to become a universal approximator, it is required to include at least one hidden layer. In Figure 5, we illustrate a generic Multi-Layer Perceptron (MLP) with 4 inputs, 2 hidden layers, and a final layer with 2 outputs.

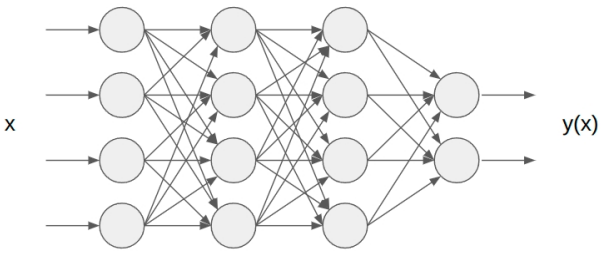

Figure 5 – Generic Multi-Layer Perceptron.

2.1.3 Gradient Descent

After having built an MLP, it is necessary to adjust it's parameters in order to make it behave as an universal approximator. Suppose we want to approximate a certain function $f(.)$, to that we have Y = f(X) using the generic MLP presented on Fig. 5.

As the first step, we could be described our generic MLP by the following equations:

$$a^{(1)} = g^{(1)}(W^{(1)}(X) + b^{(1)}) \tag{3}$$
$$a^{(2)} = g^{(2)}(W_{(2)}(a^{(1)}) + b^{(2)}) \tag{4}$$
$$a^{(3)} = g^{(3)}(W_{(3)}(a^{(2)}) + b^{(3)}) \tag{5}$$
$$\hat{Y} = g^{(4)}(W^{(4)}(a^{(3)}) + b^{(4)}) \tag{6}$$

The weights of each arrow connecting the input X to the first layer of neurons are represented by a matrix $W^{(1)}$. Each neuron has a bias term and those biases for the first layer of neurons are grouped in a vector $b^{(1)}$. Each neuron is followed by a non-linear activation function $g^{(l)}$. The same structure repeats itself for the following layers.

The gradient descent is calculated by the following steps:

- Initialize all Weights $W^{(l)}$ and biases $b^{(l)}$ for all layers l with small random values or predefined values;
- Forward propagation step: After multiplying the input X by the matrix $W^{(1)}$ and adding the bias term, this weighted input goes through an activation function $g^{(1)}$ and generates the activation of the first layer $a^{(1)}$. This activation will serve as input for the next layers and the process will repeat until the final output $\hat{Y}$ is calculated;
- Compute the loss function ($\mathcal{L}$) based on the model's predictions and the true target values: $\mathcal{L} = \mathcal{L}(\hat{Y}, Y)$ where $\hat{Y}$ is the output of the final layer and Y is the true target value;
- Compute the gradients of the loss with respect to the model's parameters $\nabla_{W^{(l)}}\mathcal{L}$ using the chain rule. This involves calculating the error at each layer and propagating it backward through the network:
  - Compute the error at the output layer: $\delta^{(L)} = \nabla_{a^{(L)}}\mathcal{L}$
  - Compute the errors at hidden layers using the chain rule: $\delta^{(l)} = (W^{(l+1)})^T\delta^{(l+1)} * a'^{(l)}$
  - Calculate the gradients for each layer: $\nabla_{W^{(l)}}\mathcal{L} = \delta^{(l)}(a^{(l-1)})^T$
- Update the model's parameters using the computed gradients and a learning rate (η) to control the step size:

$$W^{(l)} = W^{(l)} - \alpha\nabla_{W^{(l)}}\mathcal{L} \tag{7}$$

2.2 BEHAVIOR CLONING

Consider an expert that interacts with the environment in discrete time steps in a finite episode ended at time step T. At each time step t, the agent receives a state s_t and takes an action a_t. At each interaction, the pair (s_t, a_t) is recorded, generating a set of state-action pairs $D^k = \{(s_t, a_t)|t = 0, ..., T-1\}$ that represent the k^{th} expert demonstration. Multiple expert demonstrations can be united to form a demonstration for policies training. In this case, we can represent the union of N different trajectories D^k as $D = D^0 \cup D^1 \cup ... \cup D^{N-1}$

Should we be able to derive a policy $\pi_w(s)$ that, for each state s_t, mimics the action a_t taken by the expert, we would have developed an agent that performs the task as well as the expert.

The Behavior Cloning technique reduces the sequential decision process to a supervised learning problem, in which the parameters w of a function approximator $\pi_w(s)$ must be optimized to fit the mapping from states to actions, by minimizing a loss function L.

$$\underset{w}{\text{minimize}}\, L(\pi_w(s),a)) \tag{8}$$

2.3 REINFORCEMENT LEARNING

2.3.1 Finite Markov Decision Process

We model our control task as a finite Markov decision process (MDP). An MDP consists of a state space S, an action space A, an initial state s_0, and a reward function $r(s,a) : S \times A$ that emits a scalar value for any transition from state s taking action a. At each time step t, the agent selects an action a_{t+1} according to a policy π, i.e., $a_{t+1} = \pi(s_t)$, such that agent's future expected reward is maximized. A stochastic policy can be described as a probability distribution of taking an action a_{t+1} given a state s_t denoted as $\pi(a|s) : S \rightarrow A$. A deterministic policy can be obtained by taking the expected value of the policy $\pi(a|s)$.

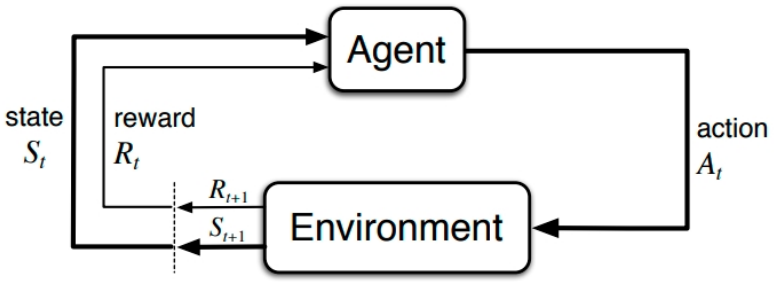

Figure 6 – Markov decision process diagram, as described in (Sutton; Barto, 2018).

2.3.2 Policy Gradient Methods

Value-based reinforcement learning methods first learn to approximate a value function $Q(s,a)$. The policy is obtained by finding the action that maximizes the latter, e.g., $\pi(s) = \arg_a maxQ(s,a)$. On the other hand, policy gradient methods optimize directly an parameterized policy $\pi_\theta(a|s)$ that can model Categorical or Continuous actions for discrete and continuous spaces, respectively.

For a given scalar performance measure $L(\theta) = v_{\pi_\theta}(s_0)$, where v_{π_θ} is the true value function for π_θ, the policy determined by θ, performance is maximized through gradient ascent on L

$$L(\theta) = \int_S \rho^\pi(s) \int_A \pi_\theta(s,a) r(s,a) da ds \tag{9}$$

$$= \mathbb{E}_{s\sim\rho^{\pi},a\sim\pi_{\theta}}[r(s,a)] \tag{10}$$

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla_{\theta}L(\theta_t)} \tag{11}$$

where $\rho^{\pi}(s) = \sum_{t=0}^{\infty}\gamma^t p(s_t = s)$ is the unnormalized discounted state visitation frequency in the limit (Sutton et al., 2000) and α is the learning rate.

2.3.3 Proximal Policy Optimization

Proximal Policy Optimization (Schulman et al., 2017) is one of the most commonly used policy gradient methods. Among the several variants for the performance measures available, we consider the clipped surrogate objective as in (Schulman et al., 2017), as follows:

$$L_t^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}, \text{clip}(r_t(\theta),1-\epsilon,1+\epsilon)\hat{A}_t)\right] \tag{12}$$

where $r_t(\theta)$ is the probability ratio $\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{old}}(a_t|s_t)}}$; θ_{old} denotes the vector of policy parameters before the update; ϵ is a hyperparameter used to clip the probability ratio by $\text{clip}(r_t(\theta),1-\epsilon,1+\epsilon)$, avoiding large policy updates (Schulman et al., 2017); and $\hat{A}_t$ is an estimator of the advantage function at timestep t, which weights the ratio $r_t(\theta)$. Here, $\hat{\mathbb{E}}_t$ denotes an empirical average over a finite set of samples.

The implementation of policy gradient considers a truncated version of the Generalized Advantage Estimator (GAE), as in (Schulman et al., 2015b):

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{13}$$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t), \tag{14}$$

where the policy is run for T timesteps (with T less than the episode size). As commonly used in the literature, γ and λ are discount factor and GAE parameter, respectively. To perform a policy update, each of N (parallel) actors collect T time steps of data. Then we construct the surrogate loss on these NT time steps of data, and optimize it with ADAM algorithm (Kingma; Ba, 2014) with a learning rate α, in mini-batches of size $m \leq NT$ for K epochs. Notice that $V_{\theta}(s)$ in GAE is learned simultaneously in order to reduce the variance of the advantage-function estimator.

Once we use a neural network architecture that shares parameters between the policy and value function, we must use a loss function that combines the policy surrogate and a value function error term. This objective is further augmented by adding an entropy

term to ensure sufficient exploration. Combining these terms, we obtain the following objective, which is (approximately) maximized at each iteration (Schulman et al., 2017):

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right], \tag{15}$$

where S denotes an entropy bonus; L_t^{VF} is the value function (VF) squared-error loss $(V_\theta(s_t) - V_t^{\text{targ}})^2$, with $V_t^{\text{targ}} = r_t + \gamma V_\theta(s_{t+1})$; and c_1, c_2 are coefficients for the VF loss and entropy term, respectively.

2.3.4 Gaussian Distribution

The output of the policy $\pi(s)$ originally proposed parameterizes a Gaussian distribution. The Gaussian distribution is defined by the following probability density function:

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{x - \mu}{\sigma} \right)^2 \right) \tag{16}$$

whose parameters μ and σ are to be estimated by a deep neural network that models a so-called Gaussian policy, i.e., a parameterized policy $\pi_\theta(a|s) \sim \mathcal{N}(\mu, \sigma^2)$.

Therefore, when acting in stochastic mode, the policy samples the distribution whereas in deterministic mode, the policy takes the mean of the distribution, i.e., $\pi(a|s) = \mu$. Since the Gaussian distribution has an infinite support, these actions are clipped to the agent's bounded action space.

2.3.5 Beta Distribution

The Beta distribution has finite support and can be intuitively understood as the probability of success, where $\alpha - 1$ and $\beta - 1$ can be thought of as the counts of successes and failures from the prior knowledge, respectively. For a random variable $x \in [0, 1]$, the Beta probability density function is given by:

$$h(x : \alpha,\beta) = \frac{\Gamma(\alpha\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1 - x)^{\beta-1}, \tag{17}$$

where $\Gamma(.)$ is the Gamma function, which extends the factorial to real numbers. For $\alpha, \beta > 1$, the distribution is uni-modal, as illustrated in Fig. 7. When acting deterministically, the Beta policy outputs $\pi_\theta(a|s) = \alpha/(\alpha + \beta)$. The α, β parameters that define the shape of the function are obtained as outputs of a deep neural network representing the parameterized policy $\pi_\theta(a|s)$.

2.3.6 Bias due to boundary effect

Modeling a bounded action space by a probability distribution with infinite support possibly introduces bias. As a result, the biased gradient imposes additional difficult

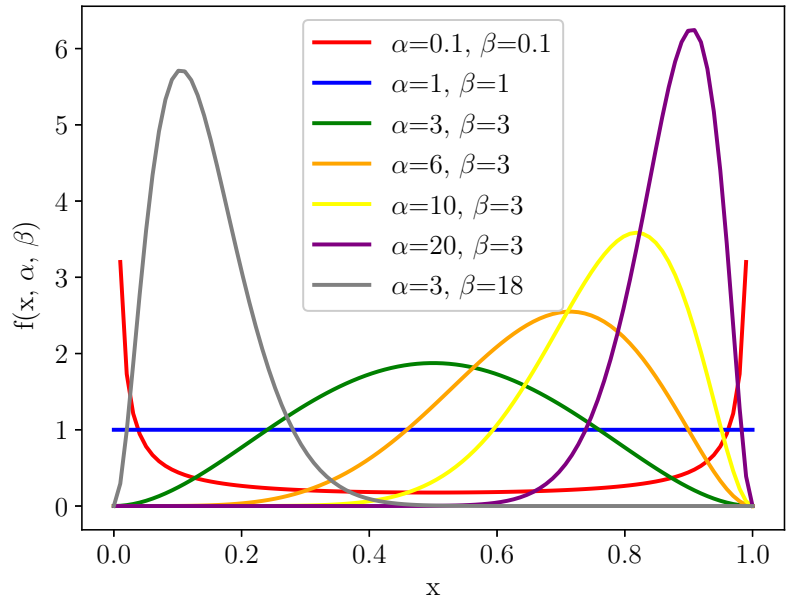

Figure 7 – Beta probability density function for different α,β pairs.

in finding the optimal policy using reinforcement learning. The policy gradient estimator to optimize the parameters θ in ((11)), using Q as the target, can be obtained by differentiating (1), as follows:

$$\nabla_\theta L(\theta_t) = \int_{\mathbb{S}} \rho^\pi(s) \int_{\mathbb{A}} \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) Q^\pi(s|a) dads \tag{18}$$

where $Q^{\pi_\theta}(s,a)$ is a state-action value function for a policy π_θ. Thus, the policy gradient estimator using Q as the target is given by:

$$g_q = \nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a) \tag{19}$$

This gradient is estimated by averaging n samples with a fixed policy π_θ, so that

$$\nabla_\theta L(\theta_t) = \frac{1}{n} \sum_{i=1}^{n} g_q \rightarrow \mathbb{E}[g_q] = \nabla_\theta L(\theta_t) \text{ as } n \rightarrow \infty \tag{20}$$

Let $A = [-h, h]$ be an uni-dimensional action space, with $a \in A$. In the case of an infinite support policy, an action $a \notin A$ is eventually sampled, to which the environments responds as if the action is either h or $-h$. The biased policy gradient estimator would be given by $g'_q = \nabla_\theta \log \pi_\theta(a|s) Q^\pi(s|a')$ in this case. Besides, focusing on the inner integral of ((18)), the bias is computed as follows (also shown in (Chou; Maturana; Scherer, 2017)):

$$
\begin{aligned}
&\mathbb{E}[g'_q] - \nabla_\theta L(\theta) \\
&\quad = \mathbb{E}_s \left[\int_{-\infty}^{\infty} \pi_\theta(a|s) \nabla_\theta \log \pi(a|s) Q^\pi(s,a')da \right] - \nabla_\theta J(\theta) \\
&\quad = \mathbb{E}_s \left[\int_{-\infty}^{-h} \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) [Q^\pi(s,-h) - Q^\pi(s,a)]da \right. \\
&\quad \left. + \int_{h}^{\infty} \pi_\theta(s|a) \nabla_\theta \log \pi_\theta(a|s) [Q^\pi(s,h) - Q^\pi(s,a)]da \right] \qquad (21)
\end{aligned}
$$

These last two integrals evaluate to zero if the policy's distribution support is within the action space A.

2.4 DISAGREEMENT-REGULARIZED IMITATION LEARNING (DRIL): A CONCEPTUAL EXPLANATION

Before we introduce the formal idea of DRIL, we elaborated a conceptual and visual explanation of the main ideas underpinning the novelty presented by the method. After this explanation, we will proceed with a more formal presentation of the method.

2.4.1 Ensemble disagreement

The absence of a naturally defined reward signal in complex tasks, such as autonomous driving, is one of the factors that restrict the application of reinforcement learning methods.

Ensemble disagreement offers an alternative approach to generate a reward signal. The concept involves training E policies, denoted as π_i, on a set of N expert demonstrations, represented as D, using behavior cloning. Each policy π_i is implemented as an artificial neural network (ANN) whose weights are randomly initialized, effectively creating ANNs (policies) that are different from each other from the beginning. Each policy is trained with a random sample (with replacement) of the training set D (Fig. 8a). This helps in creating slightly different policies in the ensemble, after training, which present similar responses for states in the training set, but differing responses for unseen states.

Consequently, considering that all policies in the ensemble were trained, for a given state-action pair (s', a') that is covered by the expert demonstrations in D, all policies π_i should select the same action a' when in state s', exhibiting behavior consistent with that of the expert. Conversely, for a state $\overline{s}$ that is not present in the expert demonstrations, each policy is likely to propose a different action. This is because their response is well defined and in agreement for the training set, but lacks a clear desired outcome for unseen states (usually, they are trained to somewhat overfit in their respective training set). In fact, the further away a state is from the ones found in expert set D, the bigger will be the disagreement in the ensemble (the more different will be the actions given by each policy in the ensemble). This 'disagreement' signal among the policies serves as an indication that

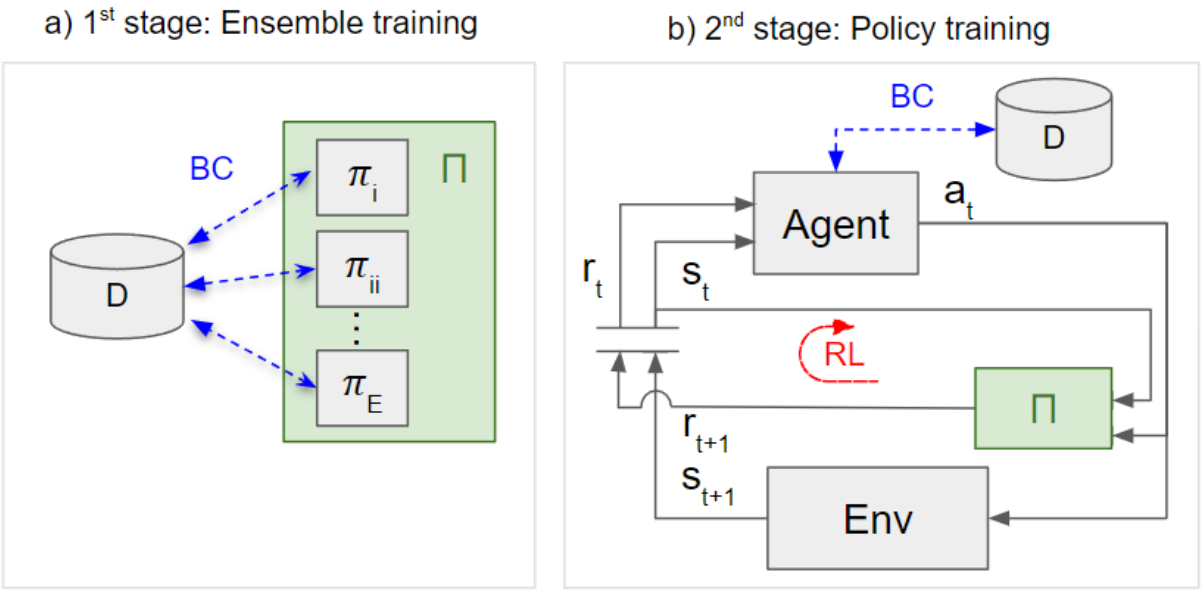

Figure 8 – Left: First stage of DRIL consists of training an ensemble Π with E policies π_i using BC. Right. On the second stage, we initiate a policy π trained with BC, and let this policy interact with the environment (Env). We store the state-action pairs together with the rewards obtained to train the policy π using reinforcement learning. After the RL loop, the policy π is again trained with BC. The training cycle proceeds interleaving RL and BC training.

the state lies outside the distribution of expert states and should be avoided. Thus, this signal can be used as a negative reward, which another policy, now from the actual agent to be trained and deployed in the autonomous vehicle, will use during policy optimization (Fig. 8b).

For instance, let's consider a tabular state space with 3 rows and 4 columns, as illustrated in Figure 9a. In this setup, each state is defined by its row and column numbers (row, col). The agent is capable of moving in four different directions: right ($\rightarrow$), left ($\leftarrow$), up ($\uparrow$), and down ($\downarrow$). The agent begins in the top-left corner, specifically at state (0,0), and its objective is to reach the final destination located in the cell below, at state (1,0), by navigating at the border of the tabular world, in a clockwise movement.

We collected one expert demonstration, as indicated in Figure 9. The expert agent made the following moves: 3 times right ($\rightarrow$), 2 times down ($\downarrow$), 3 times left ($\leftarrow$), and 1 time up ($\uparrow$). We can represent this demonstration as $D = \{$(0,0): $\rightarrow$, (0,1): $\rightarrow$, (0,2): $\rightarrow$, (0,3): $\downarrow$, (1,3): $\downarrow$, (2,3): $\leftarrow$, (2,2): $\leftarrow$, (2,1): $\leftarrow$, (2,0): $\uparrow$ $\}$.

By training an ensemble $\Pi_{E=2}$ with 2 policies, π_1 and π_2, using behavior cloning on the expert demonstrations, we ensure that for each state within the expert demonstration set, the ensemble policies select the same action as observed in the expert behavior (shown in green in Figure 10). However, for states that are not present in the expert demonstrations, the ensemble policies may select arbitrary actions. These arbitrary actions,

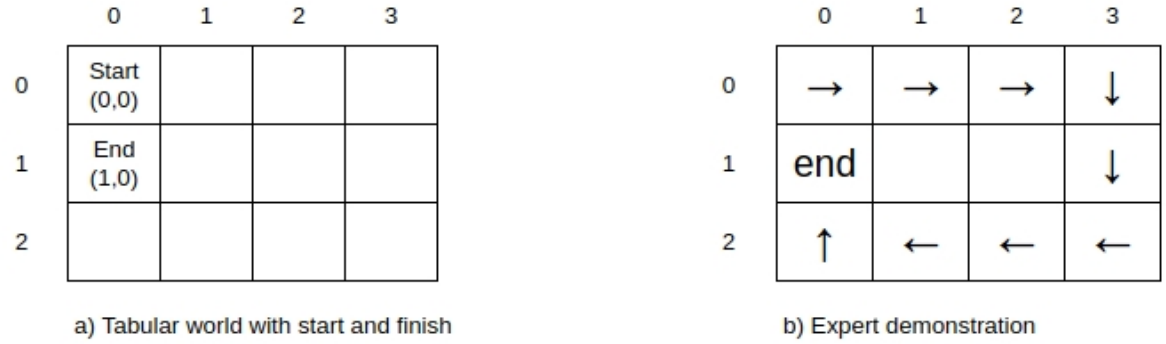

Figure 9 – Tabular environment (a) and expert demonstration (b)

shown in yellow in Figure 10, may differ between the policies. In fact, these arbitrary actions are somewhat random and have a 75

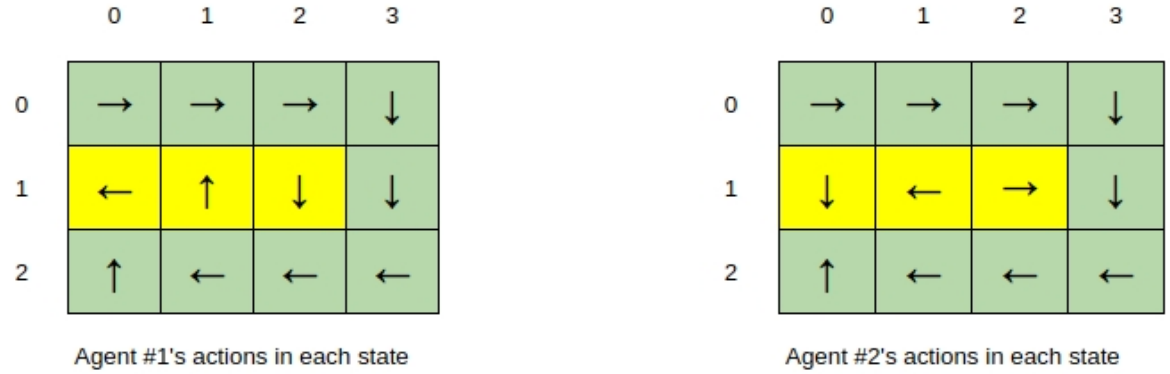

Figure 10 – Policies' actions for each state. States in green are present in the expert demonstration set while states painted in yellow are not.

Thus, for the states shown in green, which are part of the expert demonstration set, both policies agree on the selected action. Therefore, we consider these actions to be in agreement. Similarly, for the states shown in yellow, which are not present in the expert demonstration set, both policies have differing actions. Hence, we consider these actions to be in disagreement. Why do they disagree? As already stated, they were trained from different initial conditions (weights), with a random sample of the expert set, and possibly to the point of overfitting.

With this principle of disagreement in mind, we construct a reward logic using the ensemble. For a state that is part of the expert demonstration, the reward logic outputs a +1 signal. However, for a state that is not observed in the expert demonstration, such as (1,1), the reward logic outputs a -1 signal.

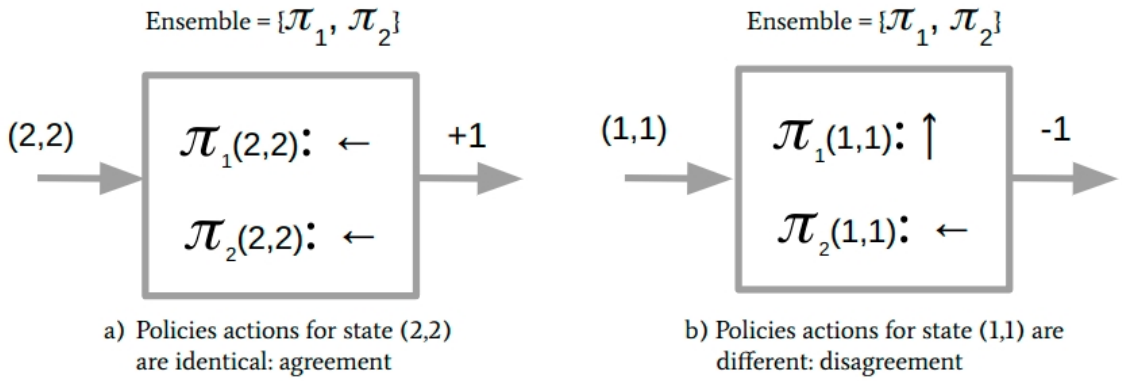

Figure 11 – Ensemble policies response to known states in agreement result in a +1 signal (a) and the same ensemble policies response to unknown states is in disagreement and result in a -1 signal (b)

2.4.2 Using Ensemble signal as a reward for Reinforcement Learning to improve the BC Policy

The second stage of DRIL consists of training a policy π through imitation learning using expert demonstrations D . This policy controls the agent and will interact with the environment, following the standard framework of reinforcement learning. The ensemble was obtained in the first stage of DRIL, where each policy was also trained through imitation learning on the expert demonstrations. This ensemble will provide the reward signal to the agent based on the state the environment transitions to (s_{t+1}). The overall diagram is illustrated in Figure 8.

The interactions between the agent's policy π and the environment, along with the corresponding rewards (derived from the ensemble's disagreement), are recorded and utilized to update the agent's policy weights. This policy update makes the agent's behavior closer to the one seen in the expert dataset, by avoiding states that are outside the expert state distribution (-1 reward) while seeking states that fall within the expert distribution (+1 reward).

2.5 DISAGREEMENT-REGULARIZED IMITATION LEARNING: THE METHOD

Disagreement-Regularized Imitation Learning aims to find a policy π that minimizes the error $J_{DRIL}(\pi)$. This error $J_{DRIL}(\pi)$ is sum of a Behavior Cloning error $J_{BC}(\pi)$ and an uncertainty error $J_U(\pi)$, as shown in Eq. (22):

$$J_{DRIL}(\pi) = J_{BC}(\pi) + J_U(\pi) \tag{22}$$

The behavior cloning error $J_{BC}(\pi)$ is the expected distance between the expert policy's actions π^* and the policy π actions for states s in d_{π^*}, where d_{π^*} is the distribution

of states induced by following the expert policy π^*, as shown in Eq. (23):

$$J_{BC}(\pi) = \mathbb{E}_{s\sim d_{\pi^*}}[\|\pi^*(.|s) - \pi(.|s)\|] \tag{23}$$

While the Behavior Cloning error is computed over the states within the expert demonstrations, the uncertainty cost is computed over distribution d_π of state-action pairs induced by following the current policy π. Thus, we move now to the second part of DRIL cost equation, the uncertainty cost $J_C(\pi)$.

As explained in earlier section on Reinforcement learning, value based methods learn a policy π by letting the policy interact with the environment in episodes of time horizon T. For each state s_τ, the policy takes an action a_τ and receives a reward $R(s_\tau, a_\tau)$. The optimal policy π is the policy that maximizes the value function $Q(s, a)$, defined as the expected sum of all rewards that will be obtained by following policy π, as shown in Eq. (24):

$$Q_{reward}(s,a) = \mathbb{E}[\sum_{\tau=t}^{T} R(s_\tau,a_\tau)|(s_t,a_t), a_\tau \sim \pi] \tag{24}$$

The same results will be obtained if we replace the reward $R(s,a)$ by our uncertainty cost $C_U(s,a)$, so that $R(s,a)$ = -$C_U(s,a)$ and instead of maximizing rewards, we minimize the uncertainty cost.

$$J_U(\pi) = Q_{cost}(s,a) = \mathbb{E}[\sum_{\tau=t}^{T} C(s_\tau,a_\tau)|(s_t,a_t), a_\tau \sim \pi] \tag{25}$$

As we detailed in the conceptual explanation section, we want the uncertainty error to guide the policy π away from states strange to the expert state distribution. Therefore, the cost function $C(s,a)$ is defined as the variance of the ensemble policy actions, after each policy has been individually trained on the expert demonstration data.

The expected value of a discrete random variable X is defined as $\mathbb{E}(X) = \sum_{i=1}^{n} p_i X_i$ where p_i is the probability of random variable X taking the value X_i. The variance of a discrete random value X is defined as $Var(X) = \mathbb{E}(X - \mathbb{E}(X))^2$. Applying this definition the actions of the ensemble, we can estimate the cost function as follows:

$$C_u(s,a) = Var_{\pi\sim\Pi_E}(\pi(a|s)) = \frac{1}{E}\sum_{i=1}^{E}\left(\pi_i(a|s) - \frac{1}{E}\sum_{i=1}^{E}\pi_i(a|s)\right)^2 \tag{26}$$

After having trained each $\pi_i \in \Pi$ with behavior cloning starting from different initial weights and using a different subset of D, we use the expert data D to calculate

$C_U(s,a)$ for all pairs (s,a). We take 98th quantile for the calculated $C_U(s,a)$, and define it as q.

$$C_U^{CLIP}(s,a) = \begin{cases} -1 & \text{if } C_U(s,a) \leq q \\ +1 & \text{otherwise} \end{cases} \tag{27}$$

$$C_U^{CLIP}(s,a) = -r(s,a) \tag{28}$$

Thus, each action policy π takes in the environment on state s_t, the policy will receive a new state s_{t+1} and an ensemble reward $r_{t+1}(s,a)$ (the ensemble reward $r(s,a)$ is the negative of the cost $C_u(s,a)$, as stated in (28)). This ensemble reward, shown on Fig. 8, will encourage the policy to avoid states that are out of expert state distribution.

Formally, the pseudo code for the DRIL algorithm is shown in Algorithm 1 box.

Algorithm 1: Disagreement-Regularized Imitation Learning

Input: Expert demonstration data $D = \{(s_i,a_i)\}_{i=1}^N$
Initialize policy π and policy ensemble $\Pi_E = \{\pi_i, \ldots, \pi_E\}$
for $e \leftarrow 1$ **to** E **do**
  Sample $D_e \sim D$ with replacement, with $|D_e| = |D|$.
  Train π to minimize J_{BC} on $|D_e|$ to convergence.
end
for $i \leftarrow 1$ **to** ... **do**
  Perform one gradient update to minimize $J_{BC}(\pi)$ using a minibatch from D.
  Perform one step of policy gradient to minimize $\mathbb{E}_{s\sim d_\pi, a\sim\pi(.|s)}[C_U^{clip}(s,a)]$.
end

3 RELATED WORKS

3.1 IMITATION LEARNING

Imitation Learning in its most immediate form, Behavior Cloning, was one of the earliest methods applied to the autonomous vehicle problem. In one of the first attempts, (Pomerleau, 1989) explored the concept of end-to-end autonomous driving systems using neural networks. This work also introduced the issue of covariate shift, which has been further investigated by several authors, most notably (Ross, Stephane; Bagnell, D., 2010) and (Ross, Stéphane; Gordon; Bagnell, J. A., 2010). However, covariate shift remains a challenge that needs to be addressed in imitation learning.

An alternative explanation for the limitations of imitation learning suggests that in its formulation, the policies establish a correlation between states and actions, but fail to establish causation (Haan; Jayaraman; Levine, 2019). This misidentification of causality leads to counter-intuitive problems, such as worse performance when the policy is trained with more information. The authors propose to overcome this issue by learning relations between causal graphs and using targeted interventions to effectively achieve an appropriate policy.

More recently, some works have proposed end-to-end autonomous systems using CNN architectures and less than 100 hours of driving data, which primarily consist of front cameras and steering data (Bojarski et al., 2016). The authors claim that with a relatively small ANN comprising 250 thousand parameters, it was possible to build a system capable of autonomously driving 98% of the time, specifically for lane and road following tasks.

Regarding the data aggregation for the supervised learning task, (Zhang, Z. et al., 2021) argued that even though end-to-end autonomous driving approaches often rely on expert demonstrations, humans are not effective coaches for algorithms requiring dense on-policy supervision. In contrast, automated experts with specialized knowledge can efficiently generate on-policy and off-policy demonstrations. However, current automated experts in urban driving rely heavily on manually crafted rules and underperform even in driving simulators. To address these issues, a reinforcement learning expert was trained to map bird's-eye view images to continuous low-level actions. This expert not only achieved superior performance in CARLA (Dosovitskiy et al., 2017) but also served as a more effective coach for imitation learning agents. As a result, a baseline end-to-end agent with monocular camera input achieved expert-level performance, including a 78% success rate in new environments and state-of-the-art performance on challenging routes in the CARLA LeaderBoard.

More recent papers indicate the direction of blending Imitation Learning with reinforcement learning to adequately address safety and reliability concerns. Waymo, a leading autonomous driving company, published an article (Lu et al., 2022) where they

incorporate simplified reward functions to enhance the performance of IL using real-world human-driving data. The reward function encourages the policy to avoid collisions and stay on the road. This approach consistently outperforms pure imitation learning methods, particularly in challenging scenarios.

3.2 DRIL

DRIL methodology was originally published in 2020 and according to Google Scholar[1], as of the writing of this thesis, there were 82 citations of this paper. While most citations are just referencing the work as part of the imitation learning literature, a few actually use the methodology as a benchmark. We discuss below the results obtained by the latter group.

In (Sasaki; Yamashina, 2021) the authors propose a method "Imitation Learning from Noisy Demonstrations" (ILND) to handle sub-optimal expert demonstrations included in the training data set, which the authors denominate "noisy demonstrations". For the experiments, three environments from the Mujoco library (Todorov; Erez; Tassa, 2012) were used: Ant-v2, HalfCheetah-v2 and Hopper-v2. The expert demonstrations were obtained using stochastic policies trained with PPO. The final results were rebased to a standardized range, a procedure that difficults results comparison with other studies. The proposed algorithm was compared with a few other methods, of which, Imitation Learning from imperfect simulations (2IWIL) (Wu et al., 2019) presented the best results. We reproduce in Table 2 the normalized scores and standard deviation shown in the article. DRIL was the best performing algorithm in Ant-v2 and the worst performing in both HalfCheetah-v2 and Hopper-v2.

Table 2 – Normalized scores and standard deviations obtained applying 2IWIL, ILND and DRIL to MUJOCO robotic control tasks.

Environments	2IWIL	DRIL	ILND
Ant-v2	1.042 ±0.021	**1.071 ±0.023**	1.055 ±0.053
HalfCheetah-v2	1.024 ±0.059	0.065 ±0.006	**1.093 ±0.092**
Hopper-v2	**1.223 ±0.135**	0.910 ±0.099	1.003 ±0.045

In *A pragmatic look at deep imitation learning* (Arulkumaran; Lillrank, 2021), authors propose to compare several Imitation learning algorithms. They choose to work with 4 environments from pybullet [2]. Authors use as benchmarks both a PPO trained agent as well as a static dataset containing state-action pairs and rewards for full episodes. This static dataset is primarily intended to provide a more reliable benchmark for those environments and avoid comparison with partially-trained agents. The main problem with this work is that the authors allegedly changed the algorithm so that it does not interleave

[1] https://scholar.google.com/scholar?oi=bibs&hl=en&cites=16471833101337443213

[2] `https://github.com/takuseno/d4rl-pybullet`

BC and RL. In this implementation, the authors only use the RL part of the algorithm. While the authors claim this ablation was necessary in order to allow for a more fair comparison among the algorithms, we found out that this change significantly impairs the algorithm performance.

Table 3 – Systematic study of Imitation Learning methods recommends to consider non-adversarial methods. Below are shown scores obtained by deterministic policies over 5 random seed averaged across 50 consecutive episodes.

Method	Ant	HalfCheetah	Hopper	Walker2D
PPO	936 ± 551	1299 ± 300	1114 ± 332	536 ± 246
Dataset	571 ± 104	787 ± 104	1078 ± 326	1107 ± 418
BC	629 ± 19	509 ± 186	1006 ± 12	220 ± 24
GAIL	421 ± 183	-863 ± 638	13 ± 1	281 ± 211
AIRL	270 ± 59	24 ± 511	445 ± 205	322 ± 211
FAIRL	499 ± 95	-1411 ± 151	497 ± 322	519 ± 100
GMMIL	591 ± 80	226 ± 546	1193 ± 68	645 ± 67
RED	403 ± 164	-1374 ± 89	641 ± 158	548 ± 124
DRIL	414 ± 109	-1416 ± 48	762 ± 96	591 ± 88

3.3 CARRACING

The CarRacing-v0 environment was launched with the OpenAI Gym library (Brockman et al., 2016). The main objective of the library is to provide a standardized testbed to appropriately benchmark different reinforcement learning algorithms and provide a collection of reference environments. Our literature review revealed works using the CarRacing environment to evaluate both reinforcement learning and imitation learning methods. In this section, we summarize key takeaways from those articles.

3.3.1 Reinforcement Learning

In the paper titled *Optimizing Agent Training with Deep Q-Learning on a Self-Driving Reinforcement Learning Environment* (Rodrigues; Vieira, 2020), researchers implemented discretization to CarRacing's action space and applied Deep Q-learning. After searching for the ideal discretization for the environment and calibrating for different exploration-exploitation ratios, the authors reported a final version of the agent that achieved an average score of 905 ± 24.

In the paper titled *Deep Neuroevolution of Recurrent and Discrete World Models* (Risi; Stanley, Kenneth O, 2019), researchers from Uber AI tested a genetic algorithm named "Deep Neuroevolution" (Such et al., 2018) to train an agent to solve the CarRacing environment. The main idea of the paper was to apply genetic algorithms as a competitive alternative to reinforcement learning methods. With this approach, the authors surpassed the expert-level threshold, achieving an average score of 903 ± 72.

In the paper titled *Weight-Agnostic Neural Networks* (Gaier; Ha, 2019), researchers at Google Brain tested networks that were able to solve certain tasks without weight training. Instead, the network weights shared a single value, and the researchers performed a topological search, increasing the complexity of the network using a method inspired by a genetic algorithm named NEAT (NeuroEvolution of Augmenting Topologies) (Stanley, Kenneth O.; Miikkulainen, 2002). With this approach, the best experiment achieved a performance of 893 ± 74 on the CarRacing environment, slightly below the expert-level performance threshold.

In *Recurrent World Models Facilitate Policy Evolution* (Ha; Schmidhuber, 2018), the world model developed by the researchers could be trained expeditiously through unsupervised learning, enabling it to acquire a compressed spatial and temporal representation of the environment. Utilizing the features derived from this world model as inputs, they were able to train an agent with a remarkably concise and straightforward policy, effectively accomplishing the designated task. Moreover, they successfully conducted full agent training within the confines of a dream environment generated by the world model, with the ability to subsequently transfer the learned policy back into the real environment. This approach was benchmarked on the CarRacing environment, effectively surpassing expert level threshold, with a 906 ± 21 average score.

3.3.2 Imitation Learning

In *Augmenting GAIL with BC for sample efficient imitation learning* (Jena; Liu; Sycara, 2021) researchers proposed a new method that the authors trained an agent with Proximal Policy Optimizations and achieved results in the 740 ± 86, below the expert level stated by the environment specifications. This sub-optimally trained agent, was then used to generate 20 expert trajectories. These trajectories were than used to train four imitation learning techniques, namely: Behavior Cloning (BC), Generative Adversarial Imitation Learning (GAIL), GAIL pre-trained with BC (BC+GAIL), and the proposed Augmented BC-GAIL. The reported results are reproduced on Table 4. BC-GAIL was also used with a hierarchical neural architecture for control of autonomous vehicles in the CARLA simulator in (Couto; Antonelo, Eric Aislan, 2023).

Table 4 – Comparison of Imitation Learning methods trained on 20 trajectories generated by a PPO-trained expert.

Training method	Score
Random	-75.01 ± 4.10
BC	695.36 ± 97.63
GAIL	419.82 ± 198.61
BC+GAIL	594.86 ± 263.12
Augumented GAIL	732.55 ± 45.73
Expert (PPO)	740.42 ± 86.36

4 REPRODUCING DRIL PAPER RESULTS

We began our journey by reproducing the results of DRIL in both discrete and continuous settings. In this section, we present in greater detail the training and evaluation steps from selected environments: Atari Breakout, and LunarLanderContinuous.

4.1 MATERIALS AND METHODS

For this work, we have performed all simulations on a notebook Lenovo Legion with processor Intel Core i7-10750H 2.60GHz x 12 and graphics card NVIDIA GeForce RTX2060/PCie/SSE2 using Linux Ubuntu 18.04.5 LTS and Python 3.7.

4.2 DRIL FOR ATARI

The DRIL algorithm was originally tested on 6 Atari environments from OpenAI Gym (Brockman et al., 2016). These environments have the following configurations in terms of observation states and action spaces.

Table 5 – Atari Environments

Environment	Original	$\|\mathcal{S}\|$	$\|\mathcal{A}\|$
Ms Pacman	210x160x3	4x84x84x1	9
Space Invaders	210x160x3	4x84x84x1	6
Breakout	210x160x3	4x84x84x1	4
Beamrider	210x160x3	4x84x84x1	9
Pong	210x160x3	4x84x84x1	6
Qbert	210x160x3	4x84x84x1	6

In the Breakout environment, the DRIL algorithm is capable of reproducing the expert performance level with just a single expert demonstration. For this reason, we selected this environment to work with.

4.2.1 The Breakout Environment

The Breakout environment (Fig. 12) features a brick wall positioned at the top of the screen. The game begins with the player launching a ball towards the wall. Each time the ball strikes the wall, a brick is destroyed. The player controls a paddle that moves horizontally, rebounding the ball to hit the brick wall. The game ends when the ball falls below the paddle or when all the bricks are destroyed.

The action space consists of 4 discrete actions: move left, move right, do nothing, and fire. The original observation state of the environment is a screen with dimensions of 210 by 160 pixels and has 3 color channels.

To incorporate the ball speed into the observation state, we applied the same preprocessing steps as used by (Mnih et al., 2015): The screen was converted to grayscale,

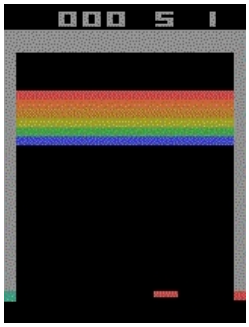

Figure 12 – **Breakout**: Environment

resized to 84x84 pixels, and four sequential screens were concatenated, resulting in an observation with dimensions of 4x84x84x1.

4.2.2 Expert Demonstrations

Expert demonstrations are obtained by utilizing agents trained with Proximal Policy Optimization, which is made available through the Stable Baselines Library (Hill et al., 2018). A single expert demonstration trajectory is defined as one attempt to play the game until its completion. In the case of Breakout, the expert successfully finishes the attempt by destroying all the bricks, thus successfully completing the game.

For training purposes, we collected 20 different trajectories using a random seed of 0. With this seed, we obtained scores of 416, 317, 370, 363, 405, 410, 425, 402, 388, 379, 417, 427, 464, 423, 359, 366, 411, 405, 410, and 303. Subsequently, the paper grouped these trajectories into sets of 1, 3, 5, 10, 15, and 20, where integers represent the number of trajectories in the respective set, and used these sets to train different versions of the policy and assess the sample efficiency of the algorithms. For our purposes, we will reproduce the experiments with 1 and 3 trajectories, as they sufficiently illustrate the results. A score exceeding 300 points is considered sufficient to successfully complete the game and represents an expert-level performance.

4.2.3 Policy Structure

For the behavior cloning step, we instantiate a policy using a 3-layer Convolutional Neural Network (CNN) followed by a two-layer fully connected network. Except for the last layer, all layers are followed by a Rectified Linear Unit (ReLU) activation function. The policy logits[1] are then used to parameterize a discrete 4-action probability distribution. Thus, the last layer has 4 output neurons with soft-max activation function. For the Behavior cloning stage, the critic network is ignored. The critic network is only used during the DRIL stage, when the ensemble signal will be available.

Each action corresponds to the following commands:

[1] Logits represent the output of neurons before any activation function is applied.

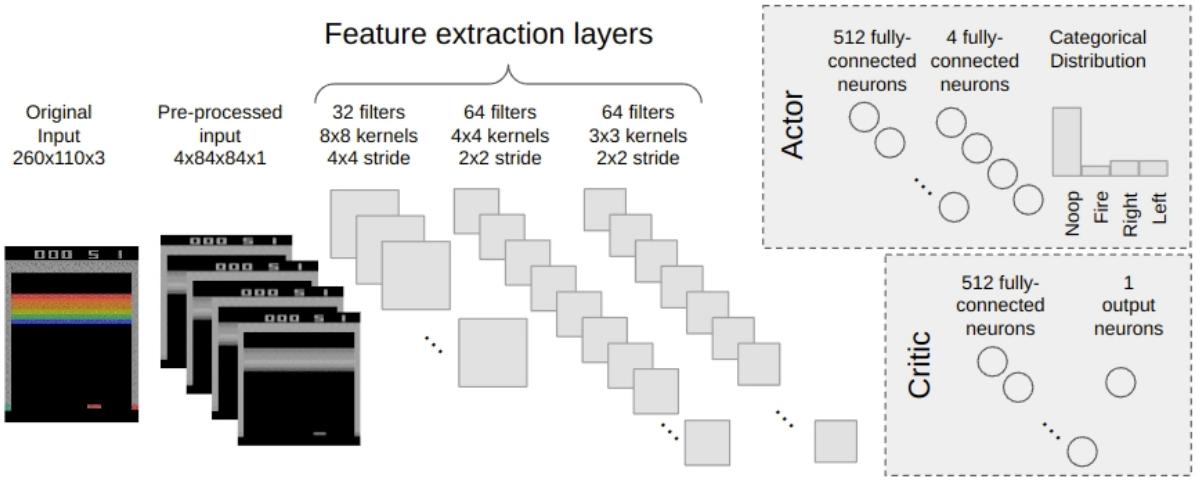

Figure 13 – **Breakout**: Policy structure for the Atari environments. Connection weights between layers are not shown for simplicity. The last convolutional layer is flattened before fedding it to the next fully connected layer (of either the actor or critic).

1. NOOP: No operation
2. FIRE: Starts the task and launches the ball for the first time
3. RIGHT: Move the paddle to the right
4. LEFT: Moves the paddle to the left

4.2.4 Behavior Cloning

We split the expert demonstrations in training and test data following a 80/20 proportion. For the loss calculation, we use the cross entropy between the probability distribution output by the model and the action taken by the expert. We minimize the training loss and measure the test loss in each training epoch, storing the best parameters. After 20 epochs without improvement in test set loss, we stop training and recover the best parameters (with lowest test error).

While the expert achieves a score of 300+ points, indicating successful completion of the game, the policy trained with behavior cloning on a single trajectory only reaches a maximum of 15 points, with an average of 5 points across 100 consecutive trials, as shown in Fig. 14a. This low score indicates that the policy was unable to master the task, as it struggled to move around and make successful ball hits. Likewise, when increasing the training data to include 3 trajectories demonstrated by the expert, the behavior cloning policy shows improved performance, averaging 7 points across 100 consecutive trials (Fig. 14b).

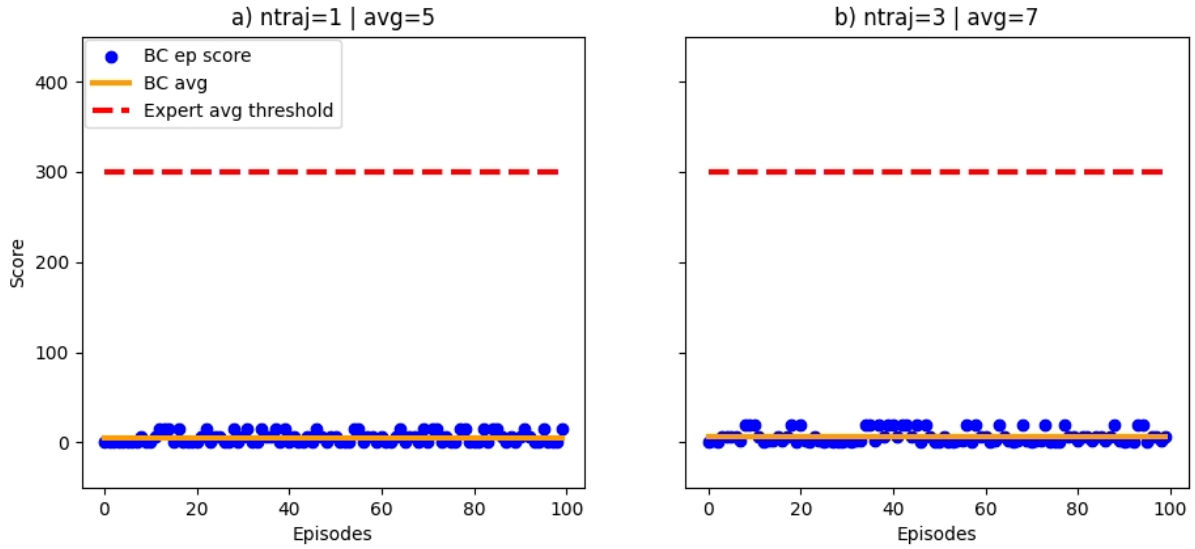

Figure 14 – **Breakout**: Policy performance after training with behavior cloning, with one trajectory (a) and 3 trajectories (b) demonstrated by the expert.

4.2.5 Ensemble Training

The ensemble Π consists of 5 policies with a shared feature extraction stage consisting of three convolutional neural network layers, identical to the structure used in the behavior cloning model. The classification stage is comprised of three layers, with 5 classification networks running in parallel (Fig. 15). Differently from BC, the output of each ensemble policy is a vector with 4 different numbers, one for each action. These numbers will be used to determine the variance threshold for each action over the different policies in the ensemble.

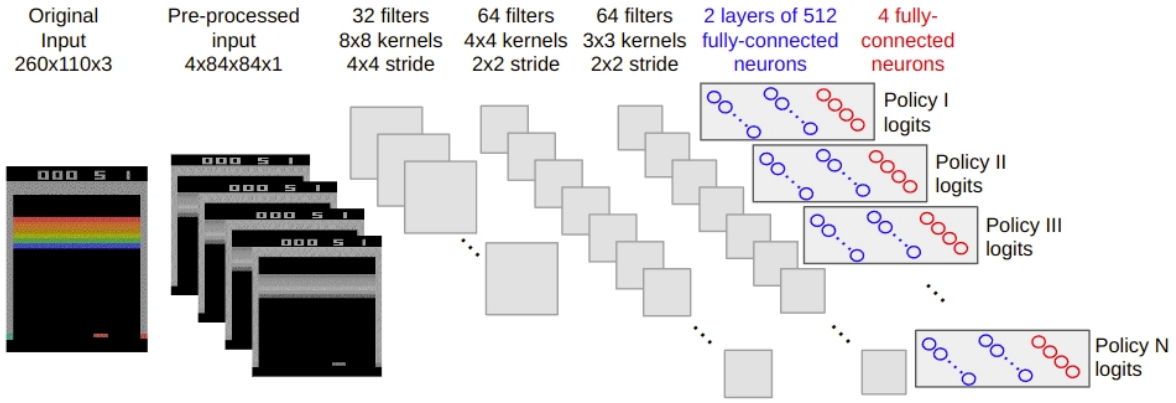

Figure 15 – **Breakout**: Ensemble structure.

These policies are trained for 2,000 iterations using the cross-entropy loss. It is clear that the ensemble has been trained to overfit because we can observe that the training error quickly converges to a low value, while the test error continues to rise (Fig. 16).

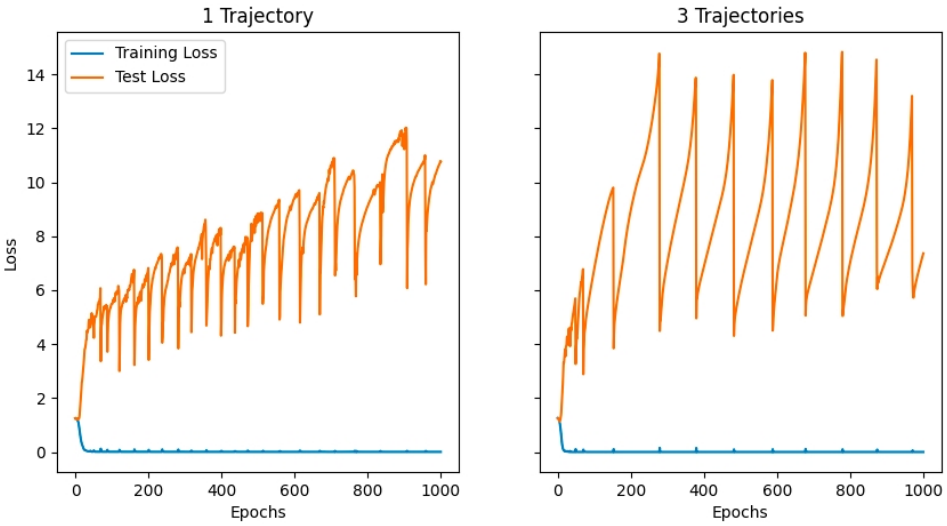

Figure 16 – **Breakout**: Ensemble training.

After having trained each policy in the ensemble, we need to determine the 98th quantile for the variance in each action logit. To illustrate the methodology, we start with an arbitrary state from the training data. The ensemble output for this state is shown on Table 6.

Table 6 – **Breakout**: Ensemble logits for an arbitrary input state.

	NOOP	FIRE	RIGHT	LEFT
Logits policy I	-6.1601	1.6958	-24.5386	-38.3140
Logits policy II	-8.5653	2.9279	-14.2508	-24.1290
Logits policy III	7.0254	-7.9456	-35.4550	-38.2695
Logits policy IV	-6.5959	1.9961	-19.2376	-23.1304
Logits policy V	-12.1714	-7.6832	-21.5352	-25.2002
Variance	53.0532	30.3411	62.5860	60.5056

For each action logit value , we estimate the variance and store it in a dictionary, resulting in the following collection: $Variance = \{NOOP : 53.0532, FIRE : 30.3411, RIGHT : 62.5860, LEFT : 60.5056\}$. We repeat this procedure for all states in the training set. After having calculated the variance for each action in each state, we can determine the threshold for agreement and disagreement based on the training data. For each action logit (NOOP, FIRE, RIGHT and LEFT), the variance across the ensemble policies for all states in the training set is shown in Fig. 17 together with the 98th quantile as a horizontal line.

4.2.6 Disagreement-Regularized Imitation Learning training

Once we have an agent's policy π pretrained with behavior cloning, and an ensemble Π trained, as well as a clear threshold (98th quantile) that will allow us to understand if a given state is known to the training set, we can proceed to the disagreement regularization

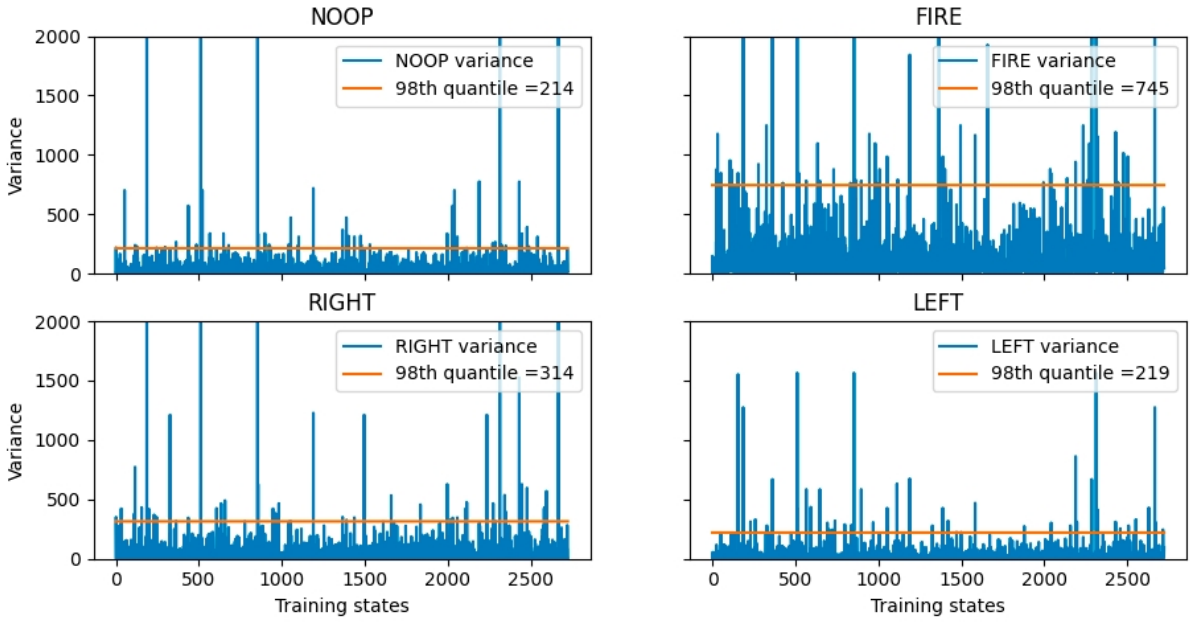

Figure 17 – **Breakout**: Each ensemble policy outputs 4 logits (NOOP, FIRE, RIGHT, LEFT), for a given state s. For instance, an ensemble with E policies will have E logits associated with the NOOP action. We calculate the variance of the NOOP logit of the E policies for every state s within the training data, as shown on the top-left chart. After that, we establish a threshold at the 98th quantile, namely at 214. Therefore, for a given state $s^{'}$ that the agent acts NOOP, if the variance of Ensemble policies NOOP logit is larger than 214, the ensemble will signal a disagreement. We repeat the same procedure for the other three logits (FIRE, RIGHT, LEFT). In this chart, we show the variances for a dataset containing one expert demonstration to illustrate the method.

stage. On the original Atari configuration, the policy π is run on 8 different environments simultaneously. Going forward, we will explain the procedure for one environment, because the additional data gained from the other environments are treated as additional state-action pairs originated by the same policy.

Applying the disagreement regularization, the agent's policy π plays the game, and the state-action pairs are recorded for each step. These trajectories are stored in a rollout (buffer) with 128 steps. For each of these steps, we have a state-action pair that will be fed to the the ensemble. We compare the variance of the ensemble for the each action taken. If the variance of the ensemble is lower than the threshold, it returns a positive reward; otherwise, it returns a negative reward.

For instance, we collected the first 128 steps in a training loop. In step 2, while the agent took a LEFT (Fig. 18, top row), the corresponding ensemble variance evaluated to around 3,000 (Fig. 18, middle row). The threshold for the LEFT action calculated in the previous step and shown on Fig. 17, lower left is 219. Therefore, the variance of ensemble actions is higher then the threshold, resulting in a reward of -1 (Fig. 18, bottom row).

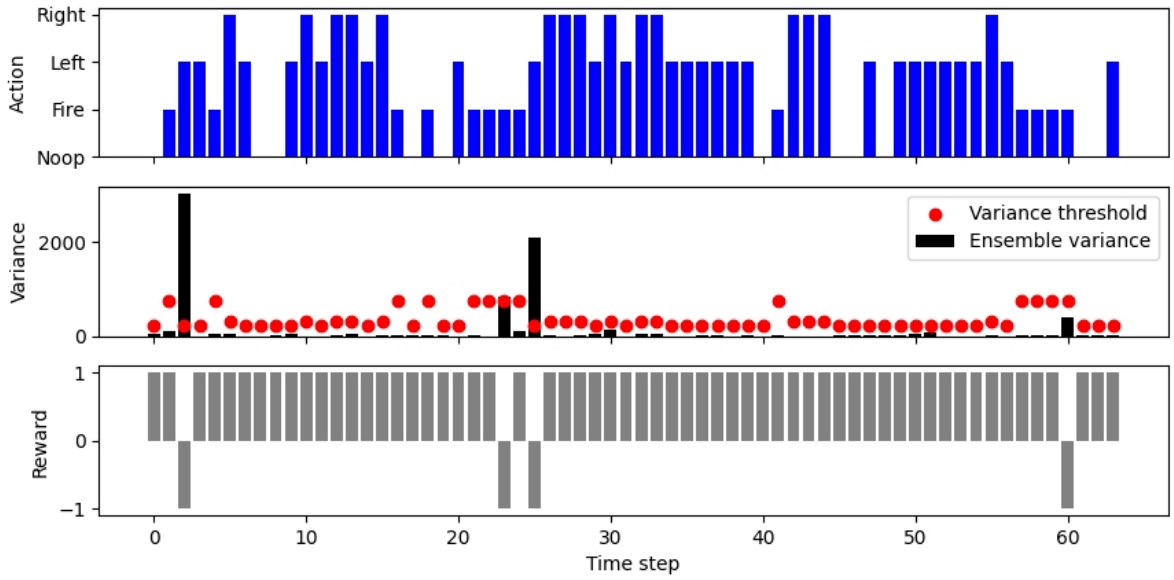

Figure 18 – **Breakout**: Construction of the reward signal based on the ensemble variance for each state-action pair generated during the first 64 steps of DRIL training.

With rewards constructed as described above, we apply one step of the Proximal Policy Optimization method, interleaved with one step of behavior cloning. We observe that during the DRIL procedure, the trained agent's policy π shows significant improvement in performance compared to pure behavior cloning (shaded gray area), as shown in Fig 19. We can observe that the policy is able to surpass the expert threshold (300 points) around 17 million steps both in the case with 1 trajectory (19a) and the case with 3 trajectories (19b).

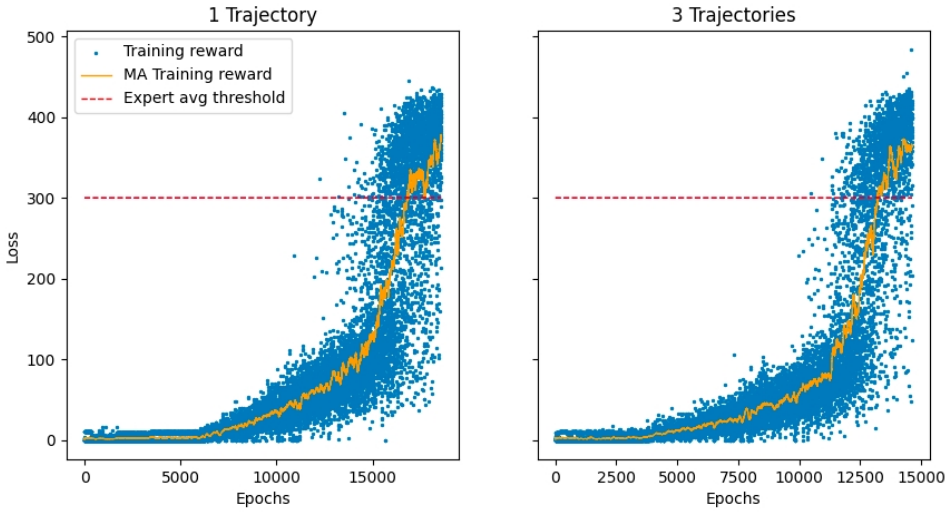

Figure 19 – **Breakout**: DRIL training for 20 million steps. Both policies surpass the expert level (300 points) around 17 million training steps.

4.2.7 Evaluation

For the policy trained with DRIL, the original implementation uses a stochastic policy during training and a deterministic policy for evaluation. The deterministic policy is obtained by sampling the action with the highest probability. Running the policies for 100 consecutive episodes, we can observe that the policy trained with 1 trajectory achieves an average score of 355 points (Fig. 20a), compared to 5 points for the behavior cloning policy. In the case of the training set with 3 trajectories, the performance reaches 339 (Fig. 20b), whereas pure behavior cloning achieves a score of 7. Although the policy trained with 3 demonstrations has a slightly lower average than the policy trained with only 1 trajectory, both surpass the expert level threshold.

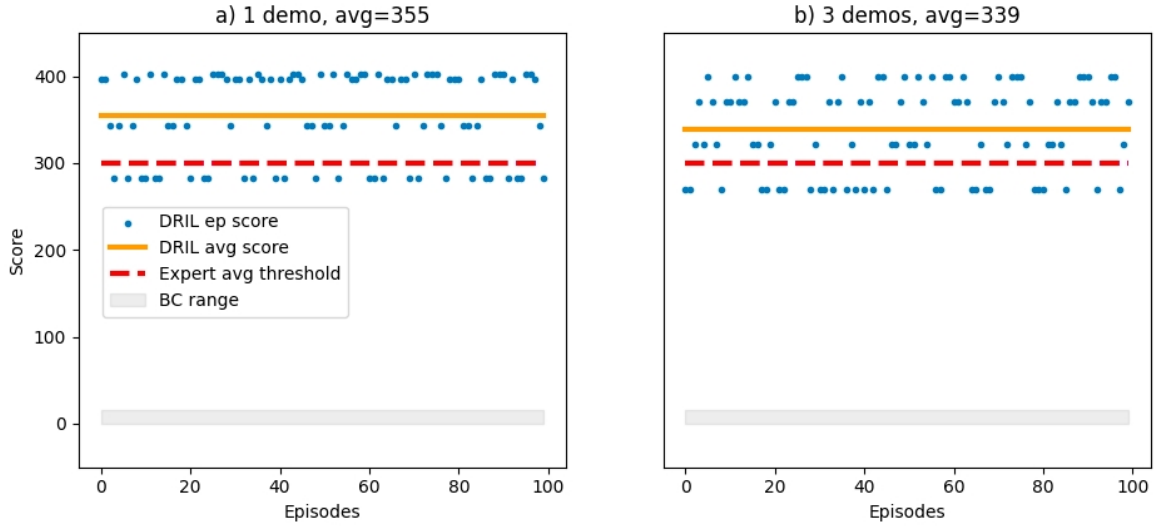

Figure 20 – **Breakout**: DRIL substantially improves results for the Policy trained with only one trajectory.

As observed, adding two expert demonstrations to the training data only slightly increased the BC performance (to 7 points from 5) while decreased the DRIL agent performance (to 339 from 355). The main explanation for this phenomena is a varying degree of expert demonstration optimality. Demonstrations from the expert have different total rewards or scores. While we have curated expert demonstrations so that all of them surpass an expert threshold level, the order of inclusion in the dataset is random and does not assume an increasing level of expertise. For this reason, additional expert demonstrations may decrease the overall score attained by the policy being trained with Imitation Learning. A more in-depth discussion regarding expert demonstration optimality can be found in *Confidence-Aware Imitation Learning from Demonstrations with Varying Optimality* (Zhang, S. et al., 2022).

4.2.8 Summary results

We were able to confirm that the DRIL algorithm was able to improve results over those obtained with pure behavior cloning. Most notably, with only one expert demonstration trajectory, the policy trained with DRIL was able to surpass the expert level threshold.

Table 7 – **Breakout**: Comparison of average scores obtained in 100 consecutive episodes for policies trained with Behavior Cloning and DRIL.

Num trajectories	Expert threshold	Behavior Cloning	DRIL
1	300	5	355
3	300	7	339

4.3 DRIL FOR CONTINUOUS CONTROL

The DRIL algorithm was also tested in 6 continuous control environments. These environments present different configurations in terms of observations states and action spaces, as shown in Table 8. We chose to implement the Lunar Lander Continuous environment for the ease of implementation.

Table 8 – Continuous Control Environments

Environment	$\|\mathcal{S}\|$	$\|\mathcal{A}\|$	Library
AntBulletenv-v0	27	8	Mujoco
HalfCheetahBulletenv-v0	17	6	Mujoco
HopperBulletEnv-v0	11	3	Mujoco
Walker2DBulletEnv-v0	17	6	Mujoco
LunarLanderContinuous-v2	8	2	Box2D
BipedalWalkerHardcore-v2	24	4	Box2D

4.3.1 The LunarLanderContinuous Environment

The LunarLanderContinuous environment (LLC) is proposed by OpenAI Gym (Brockman et al., 2016) as a platform for testing and benchmarking new reinforcement learning algorithms. The environment consists of a lunar module that starts on top of the screan with a random speed and angular velocity. The objective is to fire lateral and bottom engines to land the module smoothly between the two flags on the ground, as shown in Fig. 21.

Environment's state is described by an 8-dimensional vector, as follows:

$$s = (x, y, \dot{x}, \dot{y}, \theta, \dot{\theta}, RL, LL), \tag{29}$$

where: x and y are the distance to the center of the landing pad; $\dot{x}$, $\dot{y}$, are the linear velocity in each axis; and θ and $\dot{\theta}$ are the angular orientation and angular velocity of the

module. Lastly, the two dimensions RL and LL are two sensors that indicate that the module left or right leg have contact with the ground, and for that reason are Boolean variables.

The action space has two dimensions indicating the activation of the main and secondary engine:

$$a = (main\_engine, secondary\_engine) \tag{30}$$

The score system for moving from the top of the screen to the landing pad and settling down is approximately from 100 to 140 points. Deviating from the landing pad results in a deduction of reward. In the event of a crash, an extra -100 points are incurred. Conversely, if the lander comes to a complete stop, it earns an additional +100 points. For each leg in contact with the ground, the lander receives +10 points. Activating the main engine deducts -0.3 points per frame, while firing the side engine deducts -0.03 points per frame. Achieving the objective is worth 200 points, signifying completion of the task.

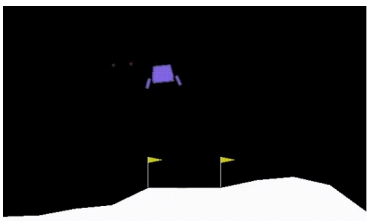

Figure 21 – **LunarLander**: Renderization of the Environment.

4.3.2 Expert Demonstrations

Expert demonstrations are obtained from using an agent trained with Proximal Policy Optimization and made available at the Stable Baselines repository (Hill et al., 2018). This repository presents a collection of agents that have been trained with reinforcement learning and typically exhibit optimal performance.

We have collected 20 expert trajectories. The scores for these trajectories were: 216, 206, 280, 223, 210, 237, 257, 230, 242, 255, 273, 282, 255, 256, 228, 235, 211, 282, 197 and 237. We grouped these trajectories in sets of 1, 3, 5, 10, 15 and 20 for the purpose of assessing how sample efficient the algorithms are. For the purpose of demonstrating paper results, we will focus on the sets with 1 and 20 trajectories.

4.3.3 Policy Structure

The policy structure receives 8 inputs and outputs a couple of actions. This initial layer is followed by a hidden layer with 64 units and a final layer of 2 units. Subsequently, the outputs of the network are used to parameterize a Gaussian distribution. For the Behavior cloning step, only the actor network is used. The critic network is used only for the DRIL stage, in which we use the PPO algorithm.

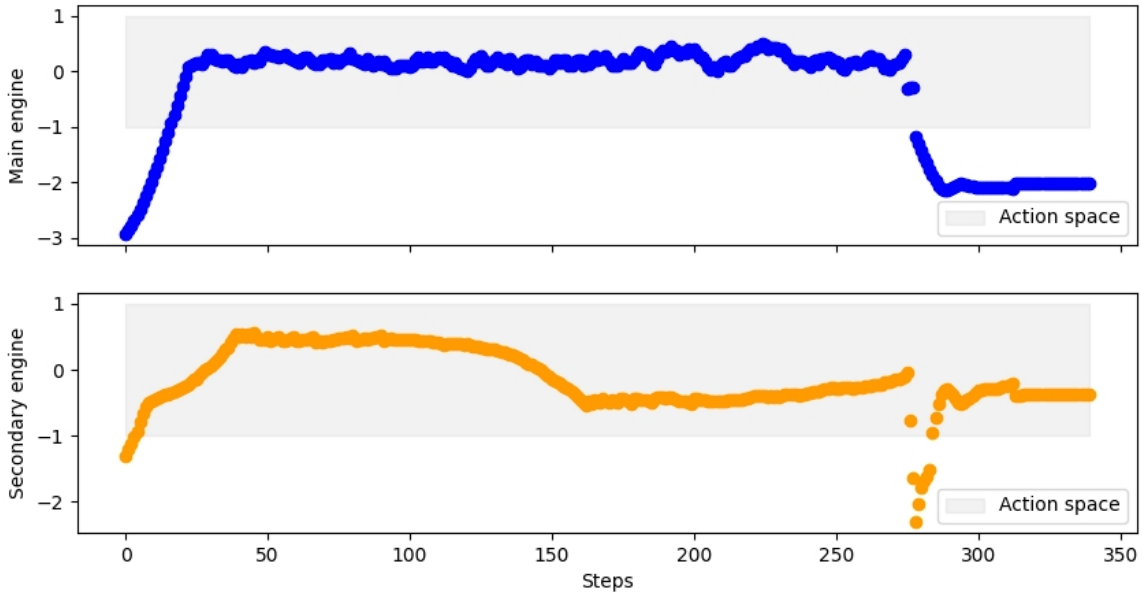

Figure 22 – **LunarLander**: Expert demonstration for one episode. The actions performed by the expert may fall out of action space boundaries, in which case, they are interpreted as the maximum action permitted. Both actions are bounded to the the interval [-1,+1].

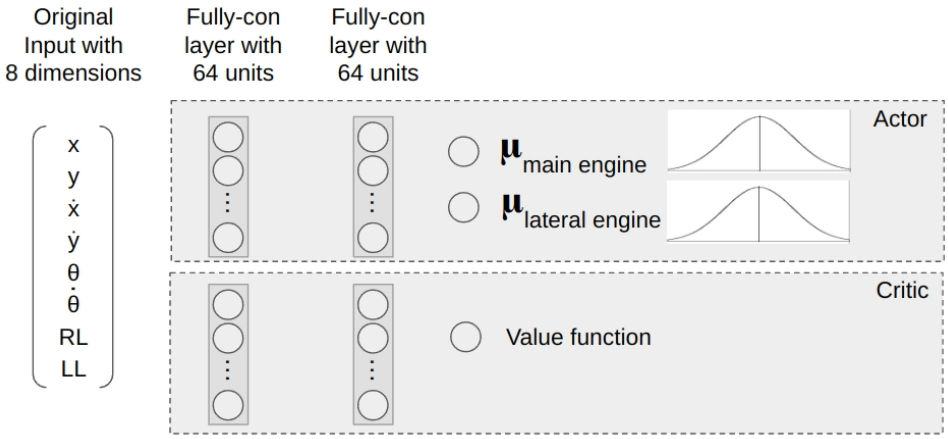

Figure 23 – **LunarLander**: Policy structure for the continuous control environments. The Gaussian distribution is parameterized by the outputs of the neural networks. The mean is taken as the logit from the neurons while the standard deviation is taken from the bias of the same neurons. The bias are implemented to be adaptive.

4.3.4 Behavior Cloning

Following the same general procedure used in the Atari environment, we split the expert demonstrations in training and test data following a 80/20 proportion. For the loss calculation, we use the mean squared error loss between the probability distribution outputted by the model and the action taken by the expert. We minimize the training loss and measure the test loss in each training epoch, storing the best parameters. After

20 epochs without improvement in test set loss, we stop training and recover the best parameters.

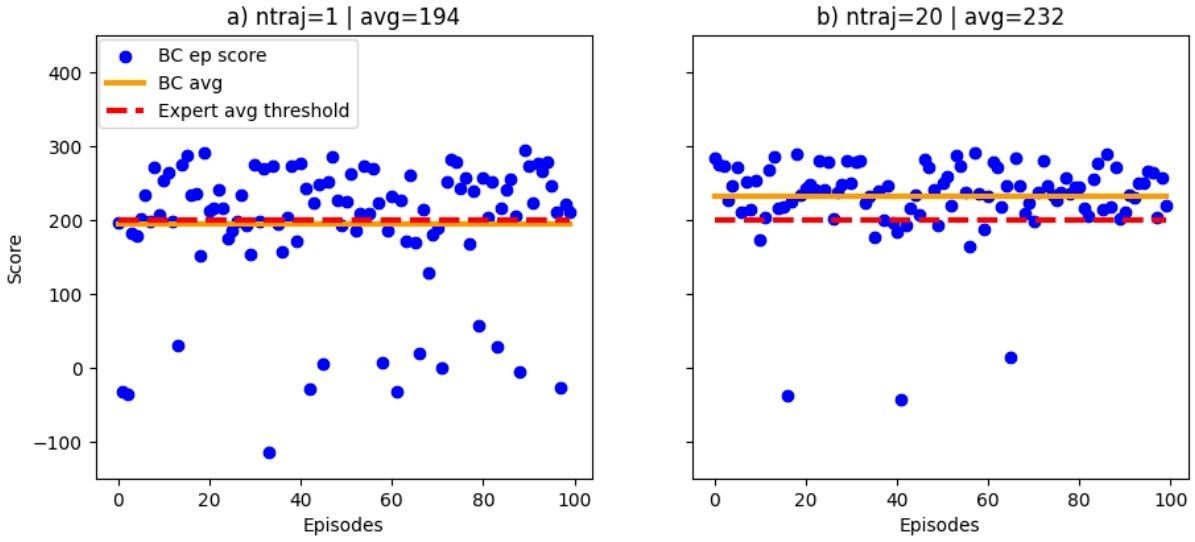

Figure 24 – **LunarLander**: Behaviour cloning policies evaluated over 100 consecutive episodes for policies trained with 1 expert demonstration (a) and with 20 expert demonstrations (b).

After the training, we test our policies for 100 consecutive episode. We show the results on Fig. 24. The policy trained with only one trajectory from expert demonstrations (Fig 24a) slightly misses the expert threshold. However, the policy trained with 20 expert trajectories (Fig 24b) effectively surpasses the expert threshold level. Notably, we observe that the increase in average score is caused by fewer crashes, which result in near zero or negative scores for the episode.

4.3.5 Ensemble Training

For the ensemble training, we train N=5 policies, where each one has an architecture comprising an 8-dimensional input layer, two hidden layers of 512 units each, and one 2-dimensional output layer, as shown in Fig. 25.

The policies are trained in parallel with 1 trajectory and 20 trajectories for 2,000 epochs. We can observe that that the ensemble overfits in both the scenarios with 1 and 20 trajectories, as demonstrated by the validation loss larger than the training loss in Fig. 26.

Consider an arbitrary state-action pair from the expert demonstration (s,a) from the expert demonstration trajectory, given by

$$s = (x, y, \dot{x}, \dot{y}, \theta, \dot{\theta}, RL, LL) \tag{31}$$

$$a = (a_1,a_2) \tag{32}$$

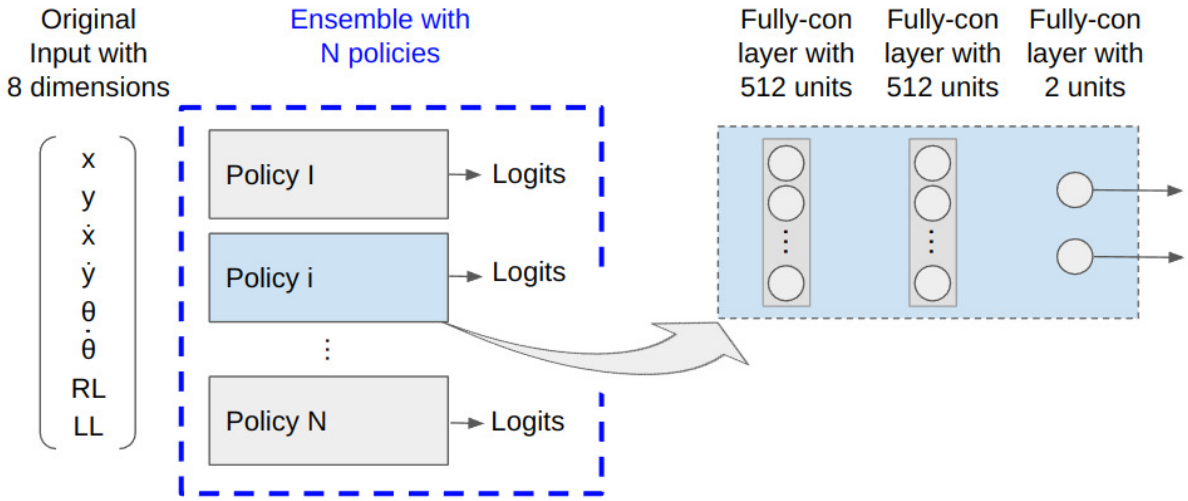

Figure 25 – **LunarLander**: Ensemble structure is comprised of N parallel policies. Each policy consists of two fully connected layers with 512 units and a final layer with 2 units, one for each action dimension.

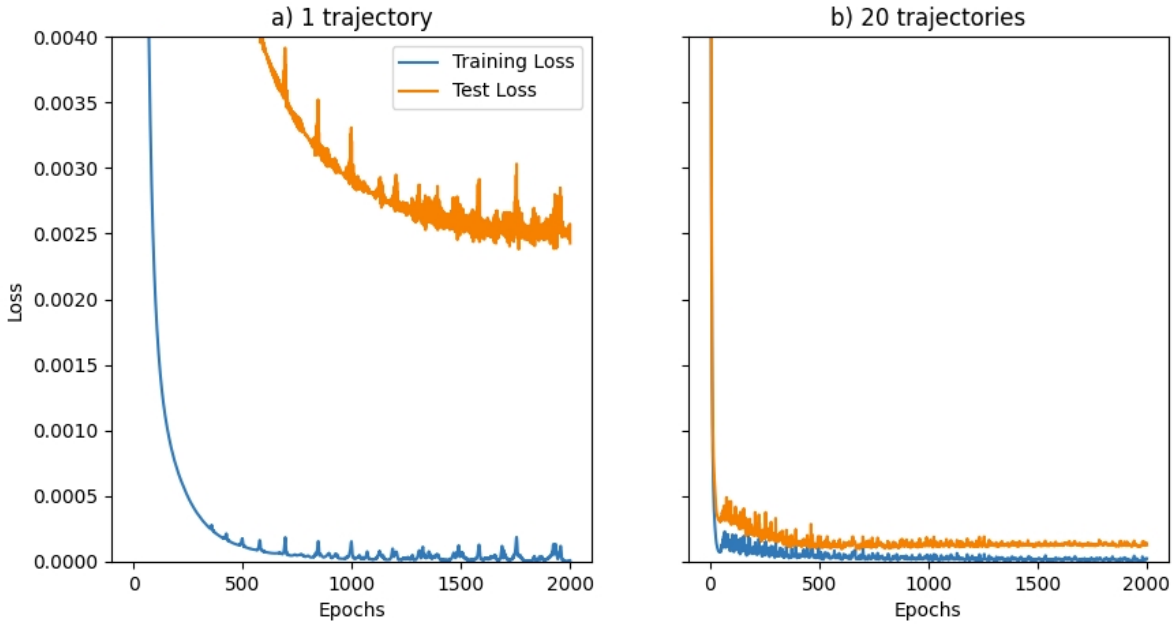

Figure 26 – **LunarLander**: Ensemble training.

.

The ensemble output for the state s is shown on Table 9. Each of the five policies outputs an action. Each action has two dimensions representing the activation of the main and secondary engines, respectively. We can compute the covariance matrix cov of all ensemble actions. The variance is then calculated by Eq. (33)

$$\sigma^2 = a * cov * a' \tag{33}$$

Table 9 – **LunarLander**: Ensemble logits for an arbitrary input state.

Ensemble Policy	Main Engine	Secondary Engine
I	0.4865	0.5586
II	0.4770	0.4287
III	0.3953	0.5147
IV	0.4320	0.2766
V	0.3994	0.5370

Repeating this variance calculation for all states-action pairs in the expert demonstrations, we are able to determine the disagreement threshold, defined as the 98th quantile of the training set variance. The variance for each state-action pair for the cases with 1 and 20 expert demonstrations is shown on Fig 27.

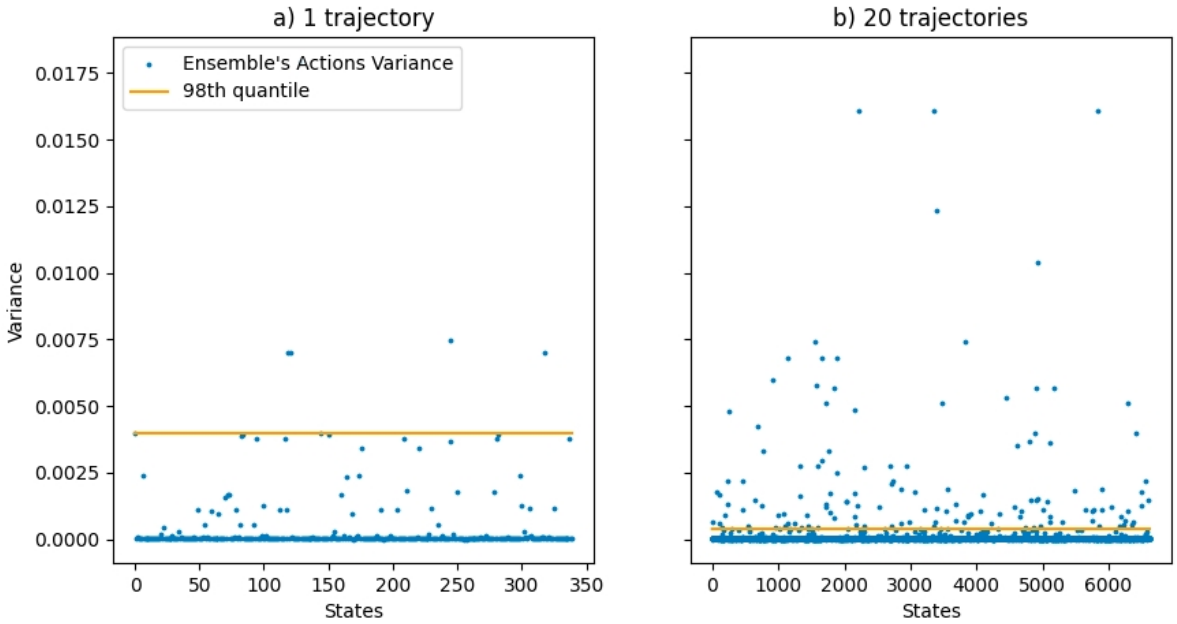

Figure 27 – **LunarLander**: Ensemble action variance threshold for one trajectory of the expert demonstrations. We observe that the variance for 20 trajectories (b) is one order of magnitude lower than the variance for the case with 1 trajectory (a).

.

4.3.6 DRIL training

Following the algorithm described steps, after training a policy π with behavior cloning and an ensemble Π with 5 policies, we can use the disagreement of the ensemble as a reward signal for Proximal Policy Optimization.

In the continuous-control setting, we let the policy π play the game for 2,048 step, following a standard reinforcement learning setting. We record for each step the state-

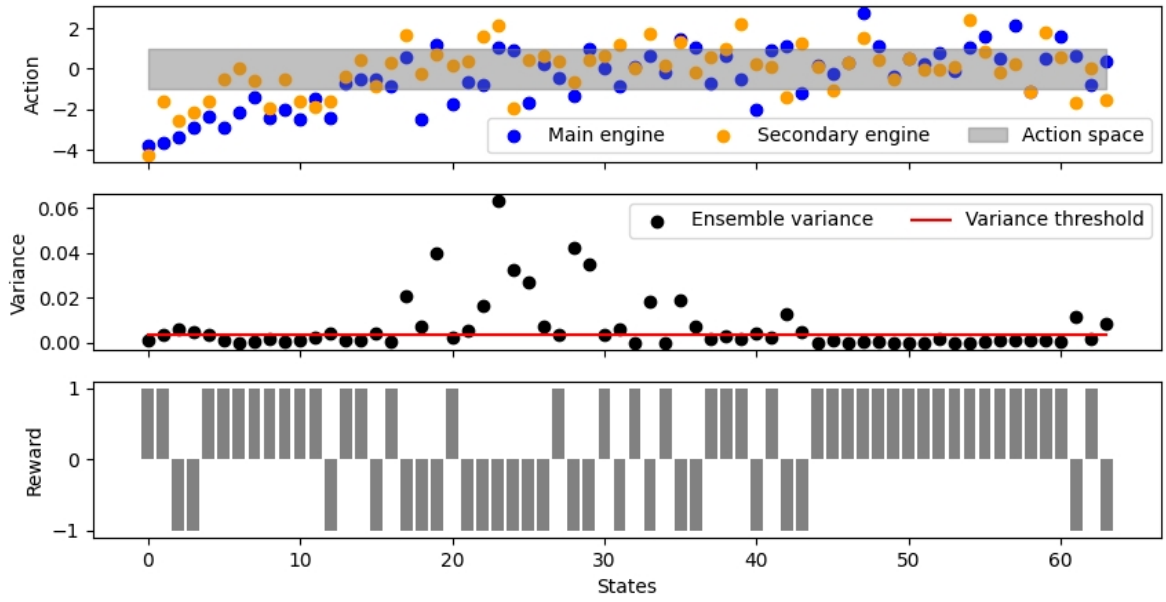

Figure 28 – **LunarLander**: First 128 steps of a game played by the policy π. Each Policy's action on the first row is comprised of two dimensions, firing of main and secondary engines, respectively. The ensemble actions' variance, shown on the second row may be above or below the 98th quantile, which is the threshold for a positive or negative reward. The reward is shown on the last row.

.

action pair along with the uncertainty reward provided by the ensemble. Nonetheless, we still measure the score the policy achieves in each episode.

For each rollout with 2048 steps, we interleave behavior cloning and proximal policy optimization using the ensemble uncertainty rewards. We perform this procedure with our policy being sampled for the actions according to each stage. During this training, our policy π outputs a Gaussian distribution for each state, and the action is taken by sampling this distribution.

Having the policy to perform on this stochastic mode leads to exploration of new states other than the optimal action. For this reason, during the DRIL training, we observe that the average score for the policies is in the 100-200 range, thus, below the expert threshold of 200 points.

4.3.7 DRIL evaluation

After training the policy π for 5 million steps, we proceed to the evaluation stage. In this stage, we take the average of the Gaussian distribution outputted by the policy π. We let the policy play 100 consecutive episodes and store the score achieved in each episode.

Fig. 30 shows the performance of the policies trained with one and 20 expert trajectories. Each blue dot marks the score achieved by the policy in an episode. We can

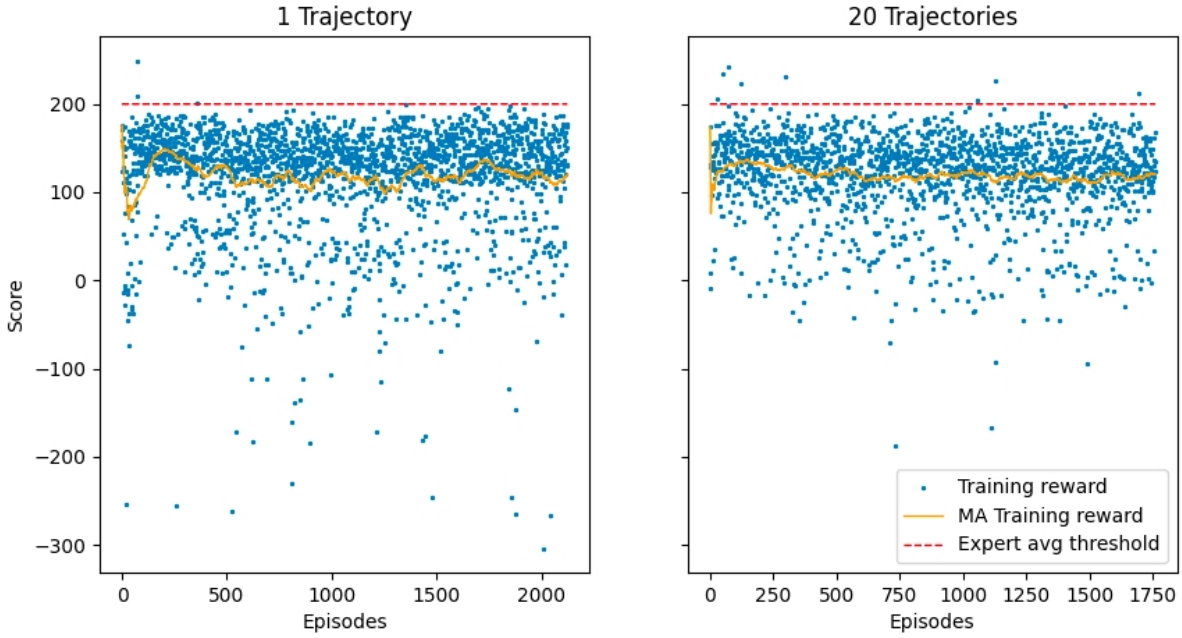

Figure 29 – **LunarLander**: DRIL Training for 2.0 million steps. Some episodes are shorter than others, as a result, there are more episodes in the experiment with 1 expert trajectory (left) than in the experiment with 20 expert trajectories.

observe that for one trajectory, the policy surpasses the expert threshold for the majority of the episodes. This result presents a substantial increase in performance compared with pure Behavior Cloning.

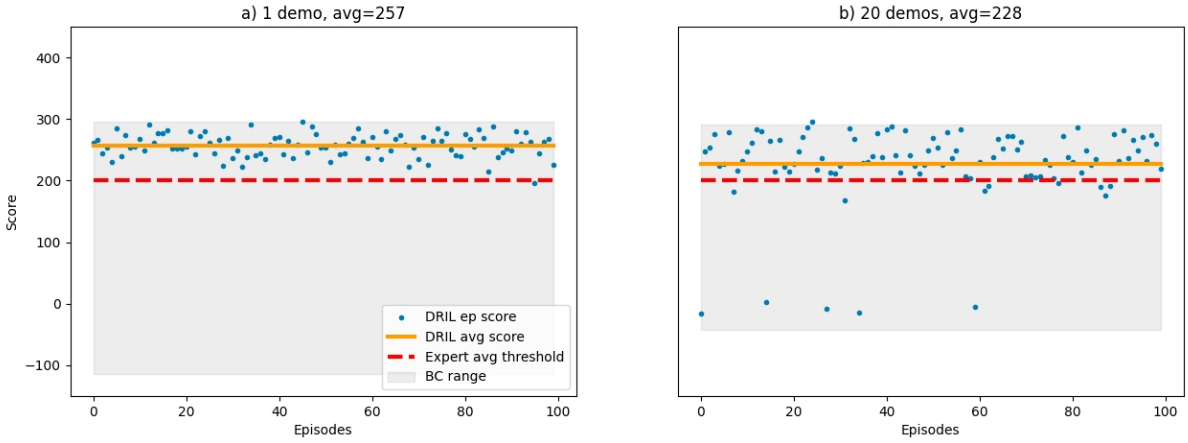

Figure 30 – **LunarLander**: Evaluation of policy trained with DRIL algorithm. We can observe that the vast majority of the episodes exceed the expert threshold demonstrating a significant improvement from Behavior Cloning (shaded area).

.

For the policy trained with 20 expert trajectories, we observe that the policy fails to reach expert level performance in 11 episodes out of 100. There are two possible

explanations for this reduction in performance. First, it is possible that some of the expert demonstrations happened to work well on a very specific landscape. For instance, sometimes the agent lands out out the pad, but the surface around the pad is a slope and the agent slides to the pad. Second, the landscapes of the environment are random and for that reason, the policy trained with 20 trajectories may have been faced with "harder" landing surfaces than the policy trained with only one trajectory.

4.3.8 Summary results

Our experiment results are summarized on Table 10. We were able to confirm that the DRIL algorithm was able to improve results over those obtained with pure behavior cloning for the continuous control example in the case with only one expert demonstration trajectory. The behavior cloning performance, 194 points, increased to 257 points surpassing the 200-point expert threshold.

For the case in which 20 trajectories were used, we could observe that DRIL did not present an improved performance over that of the policy trained with pure behavior cloning. However, we can also note that the reduction in performance was relatively small and the policy trained with DRIL still performed above expert level threshold.

Table 10 – **Lunar Lander**: Comparison of average scores obtained in 100 consecutive episodes for policies trained with Behavior Cloning and DRIL.

Num trajectories	Expert threshold	Behavior Cloning	DRIL
1	200	194	257
20	200	232	228

4.4 CONCLUSION

In this chapter we were able to successfully run the code accompanying (Brantley; Sun; Henaff, 2020) and verify that DRIL surpasses BC in the Atari environments and matches BC in Robotic control tasks, as reported in the article.

5 DRIL FOR AUTONOMOUS DRIVING SETTING

The autonomous driving setting is characterized by: a concatenated image input that combines all state information from the last four timesteps; and by a continuous action space. As originally proposed, the DRIL algorithm has been applied to problems where the observation space is high-dimensional (images) and the actions are discrete (Atari) or where the observation space is low-dimensional and the actions are continuous (Robotic control).

Applying DRIL to the autonomous driving setting required us to merge the structures used in Atari and Robotic Control: From Atari, we took the feature extraction portion of the convolution neural network. From Robotic Control, we adopted the latter portion of the network, including the outputs modeling a Gaussian distribution.

We conducted experiments considering the hyperparameter values used in the Atari experiments as well as the ones employed in the Robotic Control experiments from (Brantley; Sun; Henaff, 2020). We found that the Robotic Control parameters were more suited to our current setting.

The modified DRIL code to include CarRacing and other adaptations presented in the following chapters are available in GitHub[1].

5.1 CARRACING-V0 ENVIRONMENT

OpenAI has provided a suite of environments in the Gym ecosystem (Brockman et al., 2016) to benchmark reinforcement learning algorithms. The CarRacing-v0 environment is one of them and has been selected as our testbed for three main reasons, namely:

1. it is an autonomous driving simulator;
2. it has a simplified version of dynamics allowing for simulations that require less computational resources compared to more complex autonomous driving simulators;
3. and it shares the reinforcement learning API (Application Programming Interface) used in (Brantley; Sun; Henaff, 2020).

In the CarRacing-v0 environment, the agent controls a red car tasked with completing a racing track. The track is divided into N tiles, and the agent receives a reward of 1,000/N for each visited tile. However, the agent loses 0.1 points for each frame, and the game operates at a speed of 50 frames per second. For instance, if the agent completes the track in 750 frames, it concludes the game with a score of 1,000 - 750 * 0.1 = 925 points. An episode is deemed successful if the agent visits all tiles and finishes the track in under 1,000 frames, achieving a score exceeding 900 points. Conversely, the game ends in failure

[1] `https://github.com/igbp/dril_peak`

if the agent deviates significantly from the track (leading to a cliff fall) or if 1,000 frames elapse without visiting all tiles.

The tracks are randomly generated and vary in difficulty, primarily due to the presence of curves. As a result, the standard evaluation for CarRacing agents consists of 100 consecutive episodes. To consider the game solved, an agent must attain an average score exceeding 900 points during this standard evaluation, which we define as the expert threshold. All trajectories used for Behavior Cloning (BC) will have scores surpassing the expert threshold.

The CarRacing-v0 environment boasts an observation space of 96x96 RGB pixels, while its action space comprises three dimensions: direction, throttle, and brake. The direction dimension is continuous, ranging from -1 to +1, while the throttle and brake dimensions are bounded between 0 and 1.

To incorporate speed information into the observation space, we employed the same preprocessing steps utilized in the Atari environment. We converted the 96x96x3 RGB image to grayscale, yielding a 96x96x1 image. Subsequently, we concatenated four consecutive grayscale images. Finally, to align with the convolutional neural network architecture used in successful Atari environments, we resized the frames to 84x84 pixels.

The action space also underwent preprocessing. In the original environment, the agent could simultaneously apply throttle and brake, which we discovered significantly hampers progress. In the real world, vehicles control the throttle and brake using the right foot, rendering simultaneous engagement impossible. To simulate this dynamic in the environment, we merged the throttle and brake into a single dimension. As a result, the first dimension representing direction remained unchanged, with a range of -1 indicating maximum left turn and +1 indicating maximum right turn. The second dimension was re-calibrated to the range of -1 to +1, where -1 represents full brake, +1 represents full throttle, and 0 represents no throttle or brake action.

5.2 EXPERT'S DEMONSTRATIONS

We were unable to locate readily available expert demonstrations for the CarRacing-v0 environment, as was the case for the Atari and continuous control environments. Consequently, we needed to create our own expert agent.

In our initial endeavors to construct this expert agent, we employed the Proximal Policy Optimization (PPO) algorithm as it was originally suggested. However, the performance of the trained policy did not attain the necessary consistency to average 900 points across 100 consecutive trials.

We executed our PPO-trained policy for 100 episodes, and not all episodes reached the expert level. Therefore, for our studies, we exclusively chose the best-performing agent and sampled 20 trajectories in which the agent surpassed the 900-point threshold.

5.3 CONNECTING REWARDS TO PERFORMANCE

In the context of car racing simulations like OpenAI Gym's CarRacing-v0, the scores achieved by an AI agent can offer valuable insights into the car's performance on the track. These scores often indicate how well the agent is tackling the challenges of the race track and making efficient progress toward completing laps. However, certain nuances can lead to a misinformed performance assessment.

For example, in Fig. 31a, the cumulative rewards attained by a policy in a perfect lap are depicted. It's evident that after the initial acceleration, the cumulative rewards, or score, consistently increase until reaching its final peak at 939—well above the expert-level threshold of 900—well before the 1,000-step episode limit.

In some episodes, the agent might miss a few tiles, particularly when navigating a turn in a closed curve. If these missed tiles are situated at the lap's start, the agent could complete the lap and proceed to a second lap, eventually catching up and encountering the missed tile. This is illustrated in Fig. 31b, where the policy managed to surpass the expert level, experienced a slight decrease in the declining line (indicating the agent did not touch new tiles), and ultimately reached the tiles to conclude the episode within 1000 steps. Even during the second lap attempt, the agent might not touch all the previously missed tiles, as demonstrated in Fig. 31c. In this scenario, the agent attained the expert level score but failed to touch all the tiles, resulting in a final score below the expert level.

Lastly, the policy could miss a curve and fail to recover, resulting in a decreasing score and the episode ending at the 1,000-step time limit, as shown in Fig. 31d.

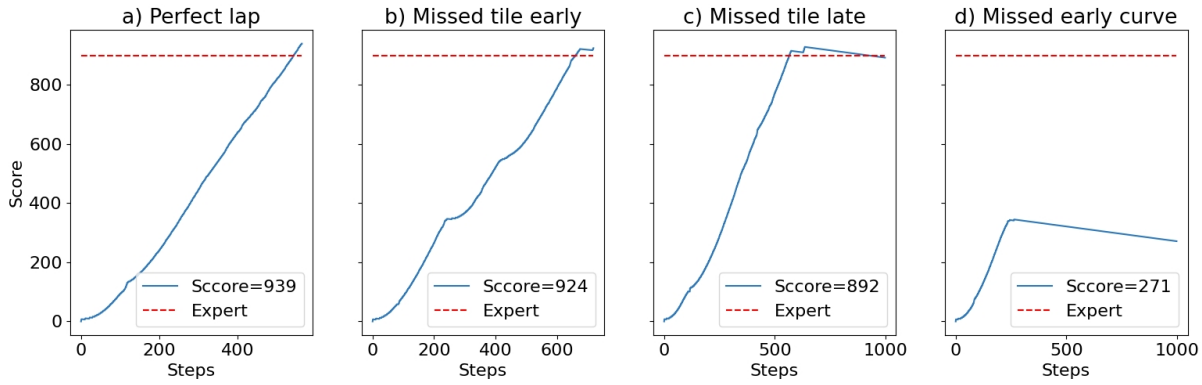

Figure 31 – **CarRarcing**:The tracks are randomly generated, leading to varying difficulty levels. Some tracks feature curves that can be navigated at high speeds, resulting in a monotonically increasing score (a). Other tracks necessitate the policy to slow down, introducing some score progression variance (b). Lastly, the agent might miss a few tiles during a lap, attempt recovery and succeed (c) or never recover (d).

5.4 POLICY

For the autonomous driving setting, we will design a policy architecture based on the combination of the architectures employed in the Atari and Robotic Control environments (Brantley; Sun; Henaff, 2020). The policy for the Atari environment employs a feature detector composed of convolutional layers, followed by fully connected layers. The output of this network is used as a parameter for a categorical distribution. In our case, we retain the backbone consisting of convolutional layers and fully connected layers, but with an output vector that parameterizes a Gaussian distribution from which continuous actions can be sampled. (Fig. 32). From now on, we call *Gaussian policy* the policy whose outputs parameterize a Gaussian distribution.

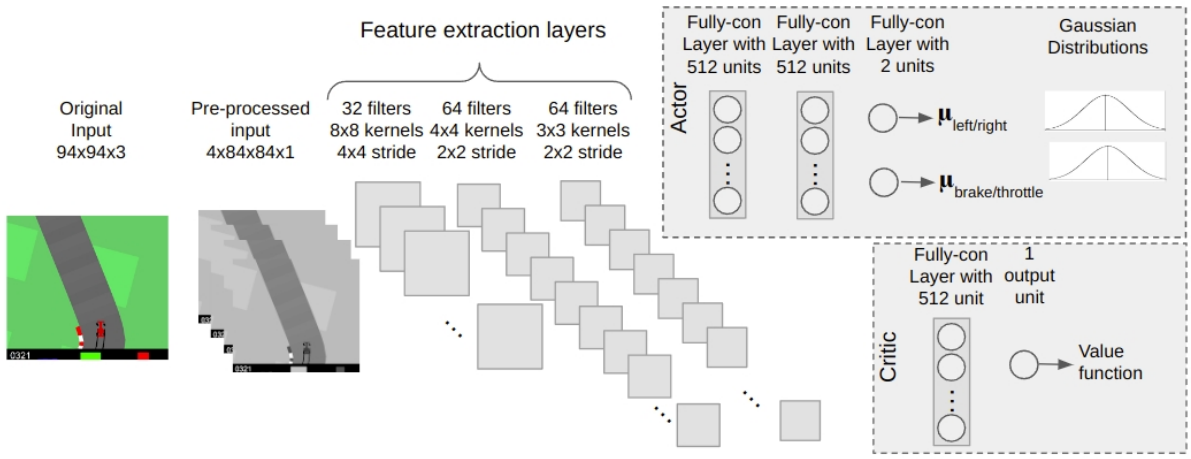

Figure 32 – **CarRarcing**: The policy for the CarRacing environment blends the convolutional feature extraction backbone used in the Atari environment with the Gaussian policy.

5.5 BEHAVIOR CLONING

We will train two behavior cloning policies: one using 1 expert trajectory and another using 20 expert trajectories, and then evaluate the performance for both cases.

Table 11 – Main hyperparameters used in Behavior Cloning.

Parameter	Value	Description
Learning rate	0.00025	Learning rate
Minibatch size	32	Size of each minibatch in the training epoch
Train data split	80%	Split between training and testing data set

The behavior cloning is performed for a maximum of 2,000 epochs using hyperparameters shown on table 11. We train using 80% of the state-action pairs and evaluate the

performance on the validation set (20%) after each training epoch. Whenever the validation error improves, we store the policy parameters as our current optimal BC parameters. If the validation error fails to improve for 20 consecutive training epochs, we stop the training process and retrieve the parameters with the lowest validation error, and consider them the optimal BC parameters.

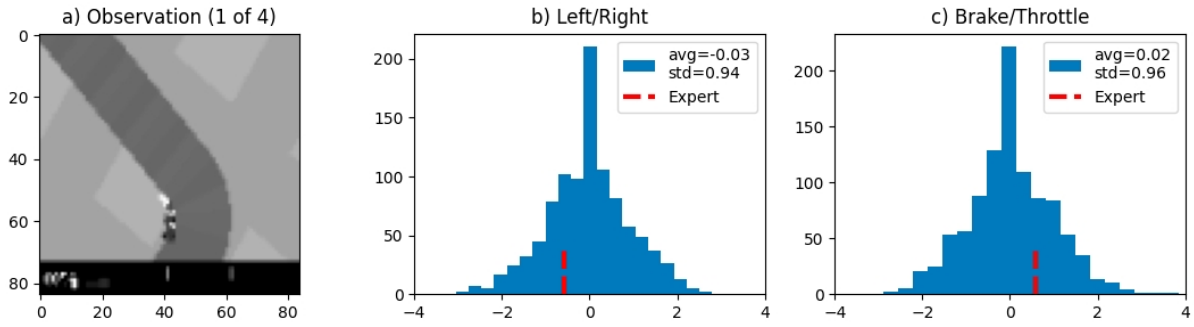

Figure 33 – **CarRarcing**: Policy is initiated to output a $N(0,1)$ distribution. For an arbitrary observation(a) The red line marks what is the expert direction action (b) and Brake/Throttle (c). Before Behavior Cloning training, the mean of the distribution is distant from the expert action (red).

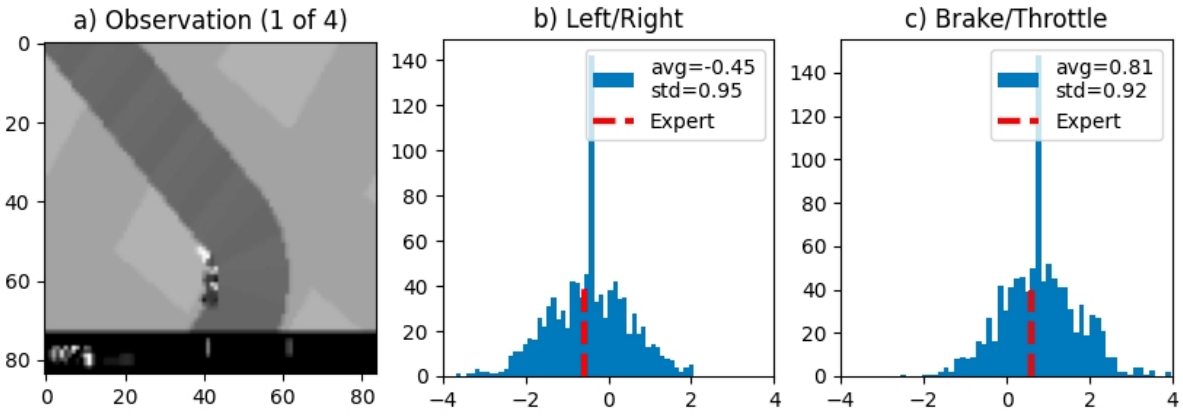

Figure 34 – **CarRarcing**: For a given arbitrary state (a), after Behavior Cloning training, we observe that the Gaussian distribution mean, as parameterized by our policy's outputs, converges for the expert action (red) in (b) and (c).

It is important to note that in this supervised learning problem, we minimize the loss between the mean (μ) of the diagonal normal distribution $N(\mu, \sigma^2)$ and the expert actions in the demonstrations. Since the Gaussian distribution is parameterized by two independent parameters, μ and σ, and behavior cloning (BC) only optimizes the mean, the final distributions retain the same variance as initialized, as observed in Fig. 34. We maintain the structure with the Gaussian distribution because it will be necessary for policy exploration when applying Proximal Policy Optimization with DRIL later on.

Table 12 – **CarRarcing**: Deterministic Policy trained with behavior cloning with 1 and 20 unbounded-action expert trajectories.

Policy mode	1 trajectory	20 trajectories
Deterministic	125±113	171±124

Using the optimal BC parameters, we conducted 100 consecutive runs of the agent. The average scores and standard deviations obtained in the evaluation of BC policies the training sets with 1 and 20 expert demonstrations are shown in Table 12. Analyzing these evaluation scores plotted in A, Fig. A.1a and Fig. A.1b, we can observe that the performance of the policies trained with behavior cloning does not reach the expert level in any trial. We can see that there is a slight improvement, with the average score increasing from 125 to 171, when training with 1 and 20 trajectories, respectively.

5.6 ENSEMBLE TRAINING

The policies in the ensemble have the same structure as the behavior cloning agent, and therefore, we use the same hyperparameters and configuration from the previous section. However, we train each ensemble policy for exactly 2,000 epochs without early stopping as done before, which is sufficient for them to overfit the training data.

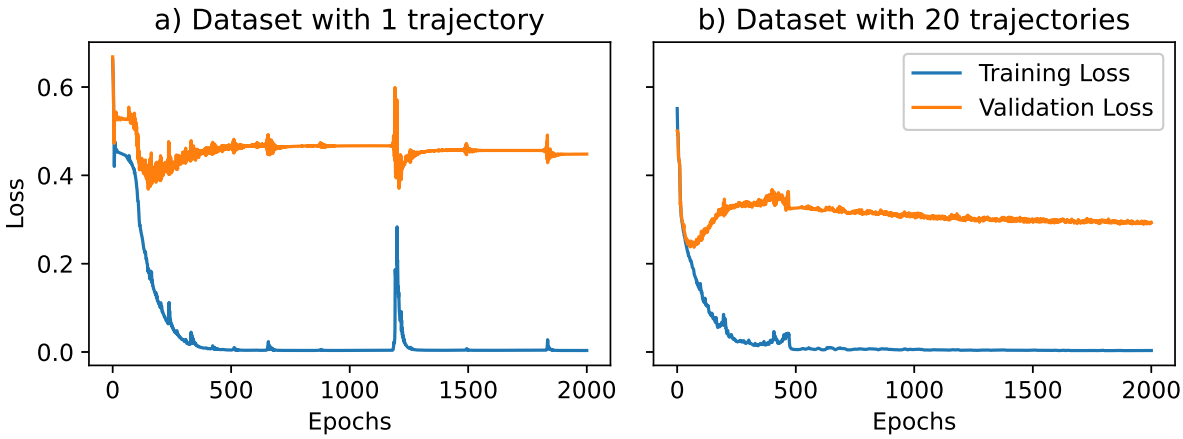

Figure 35 – **CarRarcing**: Ensemble training with 1 and 20 expert trajectories.

This overfitting causes the policies to output disagreeing actions for states that are not present in the expert demonstration data set, as explained in Chapter 2. Fig. 35 illustrates that the training loss approaches zero around 500 training epochs, indicating that our policies can fit the training data relatively well. However, the validation loss stabilizes at values substantially higher than the training loss, indicating that the policies do not generalize to states outside the training set, clearly indicating overfitting of the data.

5.7 DRIL TRAINING

Having trained the behavior cloning model and the ensemble, we proceed with the DRIL training. After testing the hyperparameters values suggested for the Atari and Robotic Control settings, we observed that the Robotic Control hyperparameters values yielded the best results. Therefore, we adopt these hyperparameters values as our starting point for further work. The main hyperparameters along with their values are shown in Table 13.

Table 13 – **CarRarcing**: Summary of hyperparameters used on disagreement regularization stage.

Parameter	Value	Description
Learning rate	3e-4	Learning rate, with linear annealing to zero
EPS	1e-5	Advantage normalization coefficient
gamma	0.99	Temporal discount applied to rewards
use_gae	True	Generalized advantage estimation
gae_lambda	0.95	Generalized advantage estimation
entropy_coef	0	PPO entropy coefficient
value_loss_coef	0.5	PPO value loss coefficient
max_grad_norm	0.5	PPO max grad avoids large steps
num_steps	2048	Number of steps in each rollout
ppo_epoch	10	PPO epochs in a loop with advantages
num_mini_batch	32	Mini-batch used for PPO / BC interleaving

We conducted the disagreement regularization training using robotic control parameters with training sets of 1 and 20 trajectories, and the results are presented in Fig. 36. We observe that the scores of the stochastic policy pre-trained with BC decreases during DRIL training.

To gain a better understanding of the working of the model, we zoom in on the first 1,000 steps of the training, shown on Fig. 37. The policies, which were pre-trained with Behavior Cloning (BC) using datasets containing 1 trajectory (Fig. 37a) and 20 trajectories (Fig. 37b), interacted with the environment during one episode, scoring -14 and 693 points, respectively. As we mentioned before, the Gaussian distribution has infinite support, leading to a reasonable amount of the actions in each episode falling out of the valid range, as we can observe on the first row of Fig. 37.

The policy trained with 1 trajectory encountered a couple of states that were flagged as out-of-distribution by the ensemble, resulting in two negative ensemble rewards. On the other hand, the states visited by the policy trained with 20 trajectories did not trigger a disagreement from the ensemble, indicating that states encountered by the policy trained with 20 trajectories were within the expert state distribution. This behavior is expected because 20 trajectories should cover a larger number of states than 1 trajectory.

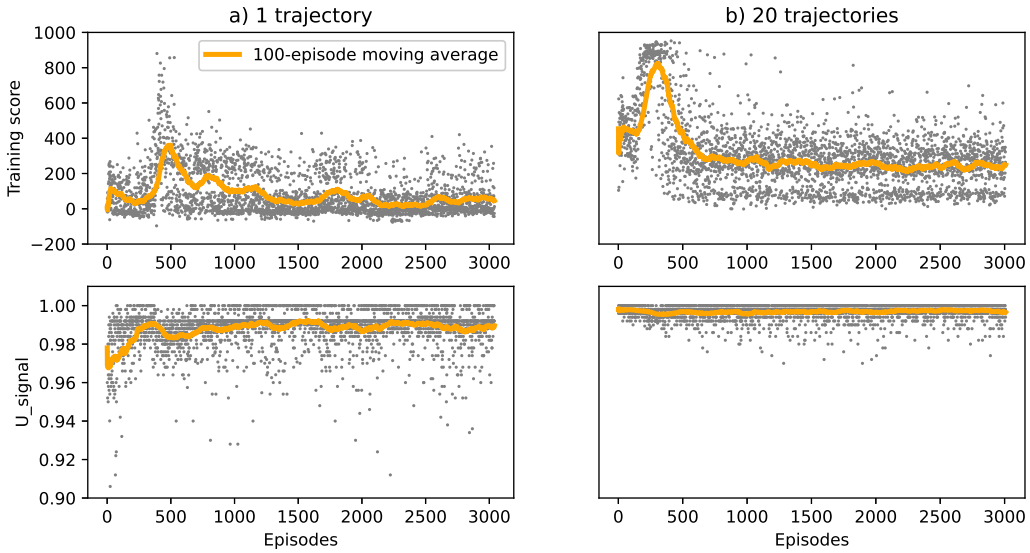

Figure 36 – **CarRarcing**: DRIL training with 1 and 20 expert trajectories with hyperparameters shown on Table 13

5.8 DRIL EVALUATION

While policy training is performed sampling the normal distribution (stochastic mode), the evaluation phase is performed taking the mean of the Gaussian distribution (deterministic mode). We evaluate the final policy obtained after 3 million steps of training on 100 consecutive episodes (DRIL final). We observed that the policy trained with 1 trajectory averaged 41 points, while the policy trained with 20 trajectories averaged 229, as shown on Table 14. Plots of evaluation episode scores are provided in the Appendix A on Fig. A.2. None of policies was able to achieve expert performance level in a single episode.

Table 14 – **CarRarcing**: Comparison of average scores and standard deviations obtained in 100 consecutive episodes for policies trained with BC and DRIL for 3 million steps (DRIL final) using datasets with 1 and 20 expert demonstrations.

		1 trajectories	20 trajectories
Clipped	BC	**125±113**	171±124
Expert	DRIL final	41±80	**229±107**

5.9 CONCLUSION

We can conclude that a policy produced by the DRIL method, specifically for the CarRacing task, when trained in stochastic mode and evaluated in deterministic mode as described in (Brantley; Sun; Henaff, 2020), performs worse than a policy trained solely

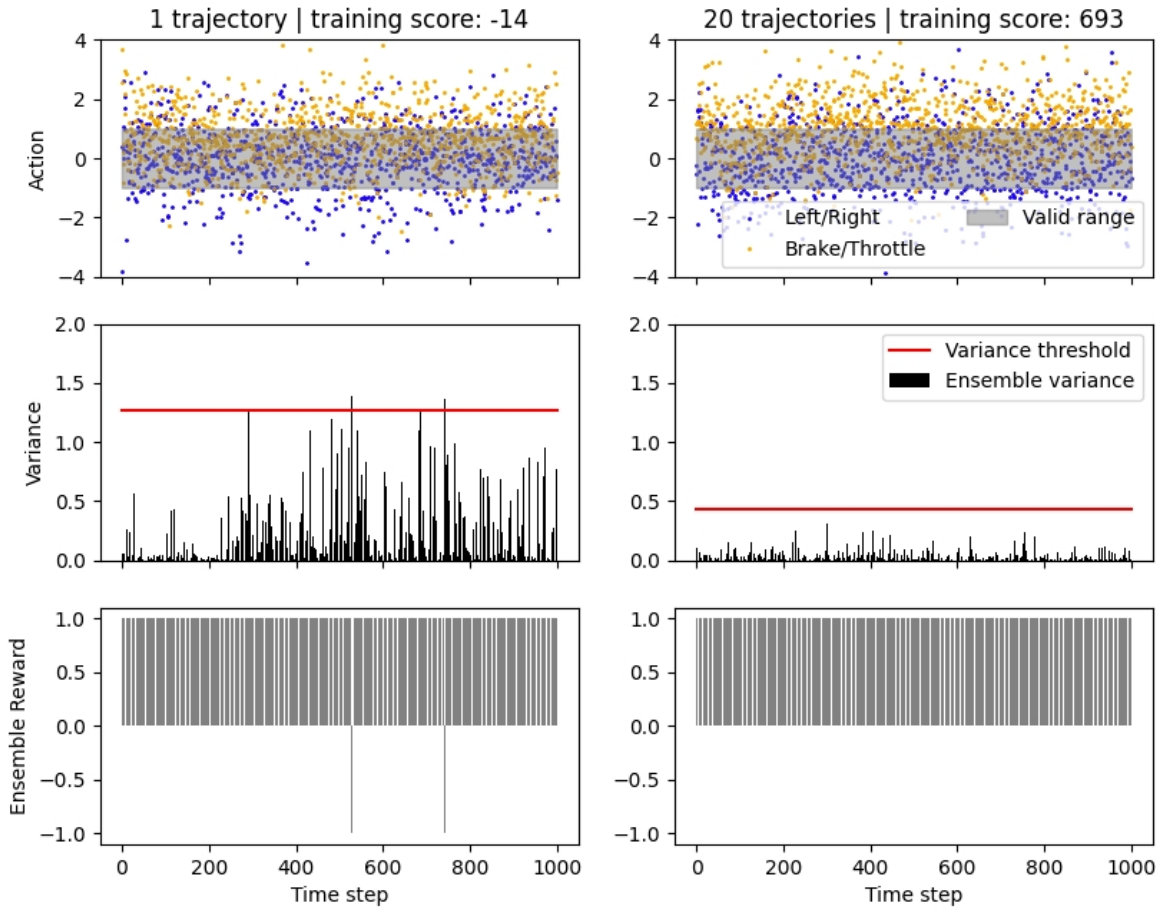

Figure 37 – **CarRarcing**: Ensemble signal during DRIL training of an agent, considering datasets with 1 expert trajectory (left column) and 20 expert trajectories (right column). Each dataset was used in BC training for both ensemble and the agent. Top: actions taken by the agent in small dots along with valid range for actions in grey color. Middle: We can observe that for the one trajectory case, there is a larger presence of disagreement (variance) in the ensemble during the agent's training, when compared to the 20 trajectories case. Bottom: the rewards computed from the ensemble's variance.

with Behavior Cloning for 1 trajectory and slightly outperforms it for 20 trajectories, as shown in Table 14.

Interestingly, in Atari and Robotics control environments, DRIL consistently outperformed or matched the BC performance. This suggests that CarRacing-v0 presents a notably more challenging problem.

6 STOCHASTIC DRIL WITH EARLY-STOPPING

In this section, we analyze the effects three changes to the original DRIL implementation, namely:

1. Selecting the optimal parameters during training, as measured by a 10-episode moving average, instead end-of-training parameters. This idea is analogous to the early-stopping method, commonly employed in supervised learning;
2. Using a stochastic policy instead of a deterministic one;
3. Employing a bounded-action demonstration dataset.

6.1 EARLY-STOPPING

Upon examining the DRIL training scores in Fig. 36, we observe that, following a transient period of fewer than 100 episodes, the stochastic policy's 100-episode moving average reaches its peak between the 350th and 500th training episodes. These peak scores hover around 350 and 800 for the experiments with 1 and 20 trajectories, respectively. This suggests that a stochastic policy may yield better results than its deterministic counterpart.

Algorithm 2: DRIL with Early-Stopping

Input: Expert demonstration data $D = \{(s_i,a_i)\}_{i=1}^{N}$
Initialize policy π, optimal policy π_{opt}, and policy ensemble $\Pi_E = \{\pi_i, \ldots, \pi_E\}$
Initialize maximum score $\phi \leftarrow -\infty$
for $e \leftarrow 1$ **to** E **do**
  Sample $D_e \sim D$ with replacement, with $|D_e| = |D|$.
  Train π to minimize J_{BC} on $|D_e|$ until convergence.
end
for $i \leftarrow 1$ **to** ... **do**
  Perform one gradient update to minimize $J_{BC}(\pi)$ using a minibatch from D.
  Perform one step of policy gradient to minimize $\mathbb{E}_{s\sim d_\pi, a\sim\pi(.|s)}[C_U^{clip}(s,a)]$.
  if *average score of the last 10 episodes* $\geq \phi$ **then**
    Store the parameters of π in π_{opt}
    Update ϕ value to the average score of the last 10 episodes
  end
end
Return: π_{opt}

For this reason, we stored the policy's optimal parameters at peak performance during DRIL training. We defined performance as a 10-episode moving average of the agent's training scores, as it provides a quicker measure of evolving performance than the 100-episode average. A revised version of the DRIL algorithm is provided in Algorithm 2

We ran a policy with peak parameters (DRIL peak) for 100 consecutive episodes and presented the average scores and standard deviations in Table 15, alongside the results obtained for BC and for DRIL using parameters at the end of 3,000 training episodes (DRIL final). A detailed chart with scores obtained for DRIL final and DRIL peak experiments is provided in the Appendix in Fig. A.2. DRIL peak posted significantly superior results in both the 1-trajectory and 20-trajectory datasets.

Table 15 – **CarRarcing**: Average scores and standard deviations for 100 consecutive episodes with BC, DRIL peak and DRIL final. Expert datasets used for these experiments are the same used in Chapter 5. Stochastic DRIL-peak outperforms both BC and DRIL-final by a large margin in both 1 and 20 trajectories datasets.

			1 trajectory	20 trajectories
Clipped Expert	BC	Deterministic	125±113	171±124
		Stochastic	30±67	473±115
	DRIL peak	Deterministic	166±131	423±211
		Stochastic	**322±208**	**802±197**
	DRIL final	Deterministic	41±80	229±107
		Stochastic	39±79	218±108

6.2 BOUNDED-ACTION DEMONSTRATION DATASET

Our PPO algorithm used in the chapter 5 did not achieve expert level. In fact, the policy trained with PPO averaged 897 with a 41-point standard deviation. This performance is right below the "solving" threshold, which requires policies to achieve 900+ in 100 consecutive episodes.

Therefore, to generate a policy that would be able to consistently produce expert level demonstrations, we reviewed the literature for newest approaches. We learned from (Chou; Maturana; Scherer, 2017) that replacing the Gaussian distribution by a Beta distribution led to improved performance for the Trust Region Policy Optimization (TRPO) method (Schulman et al., 2015a), which is a policy gradient method that came before PPO. Following the same approach, we have proposed to extend the Proximal Policy Optimization method by replacing the Gaussian by the Beta distribution in the output layer of the policy. This new development has been applied to the CarRacing environment, generating results better than the state-of-the-art, and published at SSCI (Petrazzini; Antonelo, Eric A., 2021).

We were able to train an agent using a Beta distribution by PPO that performed above the expert level threshold (Fig. 38). Although some of the episodes fell below the 900 threshold, the average of our best agent (agent B2 on Fig. 38) was 913 wit a 26-point standard deviation. A video of the policy's performance can be found on YouTube [1].

[1] https://www.youtube.com/watch?v=KSoXwt77ueY&t=26s

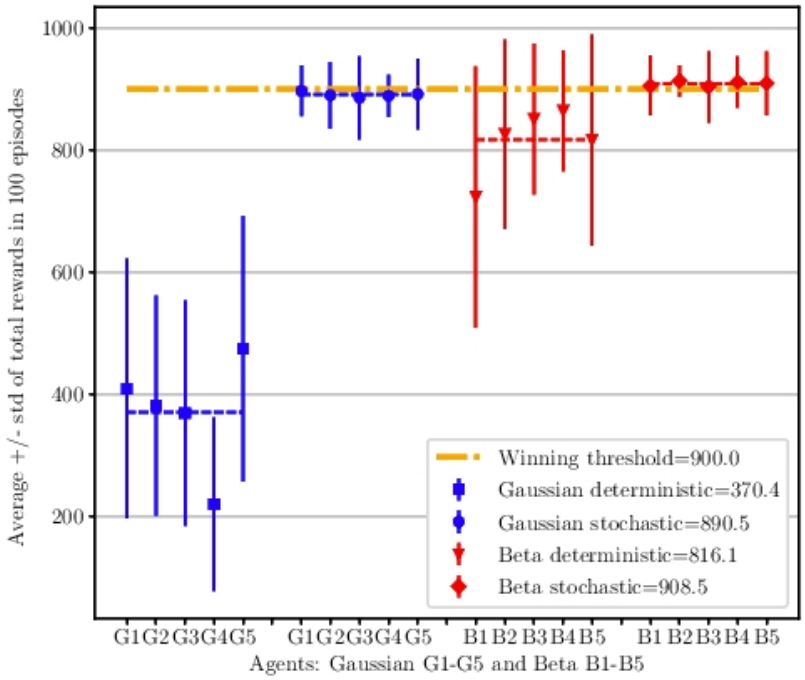

Figure 38 – Results after training Experts' policies purely by RL with PPO, in CarRacing-v0. Policies with the Gaussian distribution performed better when in stochastic mode than when in deterministic mode but failed to reach expert level. Policies with Beta distribution also performed better when in stochastic mode and were able to "solve" the environment.

Details of the training of the expert modeled with a Beta distribution, and used in this chapter onward, can be found in (Petrazzini; Antonelo, Eric A., 2021).

Table 16 – CarRacing-v0 Leaderboard

Method	Average Evaluation Score
PPO with Beta (Ours)	**913 ± 26**
World models (Ha; Schmidhuber, 2018)	906 ± 21
Adapted DQN (Rodrigues; Vieira, 2020)	905 ± 24
Genetic Algorithms (Risi; Stanley, Kenneth O, 2019)	903 ± 72
PPO with Gaussian (Ours)	897 ± 41
Weight Agnostic NN (Gaier; Ha, 2019)	893 ± 74
PPO (Jena; Liu; Sycara, 2020)	740 ± 86
Random agent	-32 ± 6

The OpenAI Gym maintains a leaderboard with self-reported performance of different approaches towards solving their environments. The leaderboards can be found on this website [2]. Our policy presented the best performance measured by average score in 100 consecutive episodes. We show on Table 16 selected works from the Leaderboard that self-reported results and, at the same time, had accompanying publications.

[2] https://github.com/openai/gym/wiki/Leaderboard#carracing-v0

As of August 2023, our proposal was still the state-of-the-art work in terms of performance in the CarRacing benchmark. Our review of all the other works presented in Table 16 has been included in our literature review in Chapter 3.

The code for the article "Proximal Policy Optimization with Continuous Bounded Action Space via Beta distribution" is available in a GitHub repository[3]. We have also made available a video[4] with 100 consecutive runs demonstrating that the expert reaches an average 913 score with standard deviation of 26 points.

Notice that once the agent with the Beta policy was trained by PPO, it was used then to generate expert demonstrations for imitation learning of a second agent in CarRacing using the DRIL algorithm instead. For that, we run expert agent B2 for a few episodes until we were able to collect 20 trajectories in which the expert exceeded the 900-point threshold. These twenty trajectories form the training set of the imitation learning agent. The following sections elaborate on that proposal. Besides, in order to see the characteristics of the data generated by both the Gaussian and Beta policies, Fig. 39 shows the demonstrations originated from different experts in terms of policy actions taken over time, i.e., the ones generated by a Beta policy as well as those produced by a Gaussian policy. In this chapter, we use the former expert, while previous chapter employed the latter by clipping its actions to the valid action space.

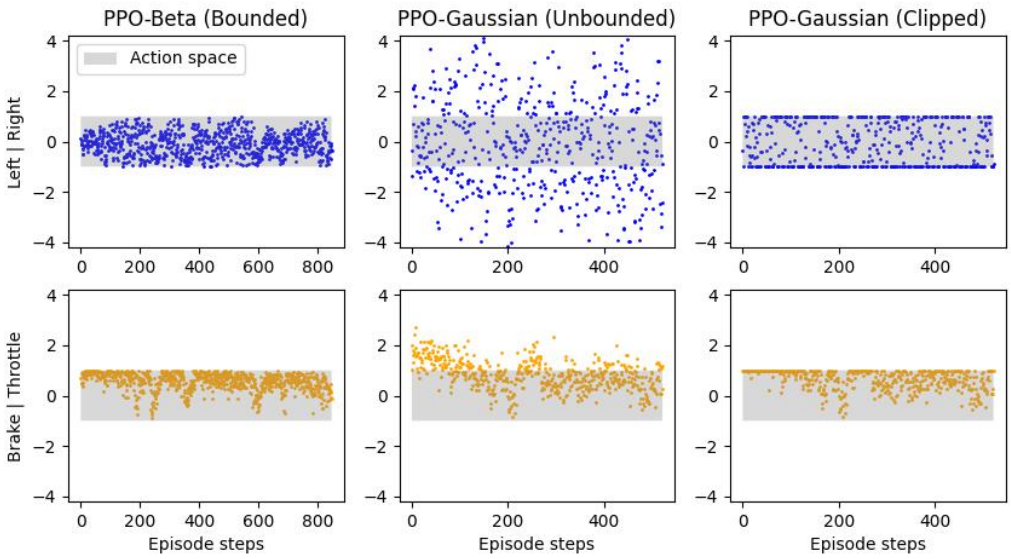

Figure 39 – **CarRacing**: The demonstration generated by the expert policy with the Beta distribution presents actions within the bounded action space (a), whereas that of the expert policy with the Gaussian distribution presents 73% of the actions falling out of the valid action space (b). The clipped Gaussian expert in (c) shows the clipped actions from (b), which are sent to the environment and used with the Behavior Cloning algorithm.

[3] `https://github.com/igbp/SSCI`
[4] `https://www.youtube.com/watch?v=KSoXwt77ueY&t=137s`

6.3 POLICY

The policy's architecture for the DRIL agent used in this chapter is identical to that used in Chapter 5. See Section 5.4 for more details.

6.4 BEHAVIOR CLONING

The same procedure from previous chapter is employed here. The behavior cloning is performed for a maximum of 2,000 epochs. We train using 80% of the state-action pairs generated by a Beta expert policy and evaluate the performance after each training epoch. Whenever the validation error improves, we store the policy parameters. If the validation error fails to improve for 20 consecutive training epochs, we stop the training process and retrieve the policy's parameters with the lowest validation error.

Using these best-performing policy we conducted 100 consecutive runs of the agent. The results obtained for the training sets with 1 and 20 expert demonstrations are shown in Table 17. It can be observed that the performance of the policy trained with behavior cloning does not reach the expert level in any trial when trained with 1 trajectory. However, with 20 trajectories, there is a substantial improvement, with the average score increasing from 50 to 304.

Table 17 – **CarRarcing**: Policy trained with behavior cloning shows improvement with increased training data in both Deterministic and Stochastic mode for bounded action demonstration datasets

Policy mode	1 trajectory	20 trajectories
Deterministic	194±133	617±260
Stochastic	75±47	137±70

6.5 ENSEMBLE TRAINING

The ensemble training done in this chapter employed the same approach described in Chapter 5. The shape of the training and validation loss curves during BC training obtained are essentially the same to those shown in previous chapter on Fig. 35.

6.6 DRIL TRAINING

Having trained the behavior cloning model and the ensemble, we proceed with the DRIL training, using the same hyperparameters values employed in Chapter 5. The training scores obtained in each episode are presented in Fig. 40 together with the Uncertainty Signal (U_{signal}).

We observe that at the beginning of the training, the policy trained with 1 trajectory is able to achieve performance up to 600 points, but it eventually degenerates to a

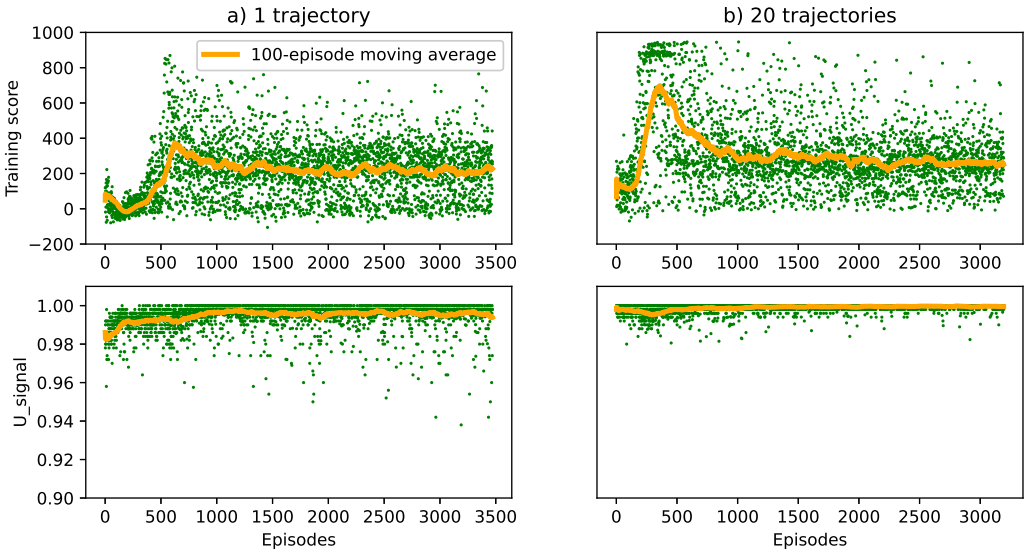

Figure 40 – **CarRarcing**: DRIL training with 1 and 20 expert trajectories with hyperparameters' values originally employed in continuous control experiments. The moving average of the score is shown by a yellow curve.

performance range of -50 to 400 points. Similarly, the policy trained with 20 trajectories initially has episodes scoring above the 900 threshold, but it also degenerates and settles at a lower performance level, scoring around 300 points at the end of training.

The U_{signal} is the average of all ensemble rewards $r(s_t,a_t) = -C_u(s_t,a_t)$ throughout an episode. Thus, for a T-step episode the U_{signal} is calculated by (34). For every state-action pair, the ensemble produces a reward. If the ensemble deems the state-action pair (s,a) to be within the expert distribution, it will return $+1$ and -1 otherwise. Therefore, a U_{signal} close to $+1$ indicates that the ensemble flags most state-action pairs as pertaining to the expert demonstration distribution.

$$U_{signal} = \frac{1}{T-1} \sum_{t=0}^{T-2} r(s_t,a_t) \tag{34}$$

To have a better understanding of the working of the model, we zoom in on the first 1,000 steps of the training, shown in Fig. 41. As we mentioned before, the Gaussian distribution has infinite support, leading to a reasonable amount of the actions in each step falling out of the valid range, as we can observe on the first row of Fig. 41. Even though the expert policy's actions are always within the valid action space, this does not transfer to the imitation learning agent trained by DRIL if the latter uses a Gaussian distribution to model a stochastic policy with continuous actions.

Each CarRacing-v0 episode has a maximum of 1,000 steps. In the episode depicted on Fig 41, the policies trained with 1 and 20 trajectories ended with 64 and 6 points, respectively. The policy with fewer trajectories pointed out a higher number of out-of-

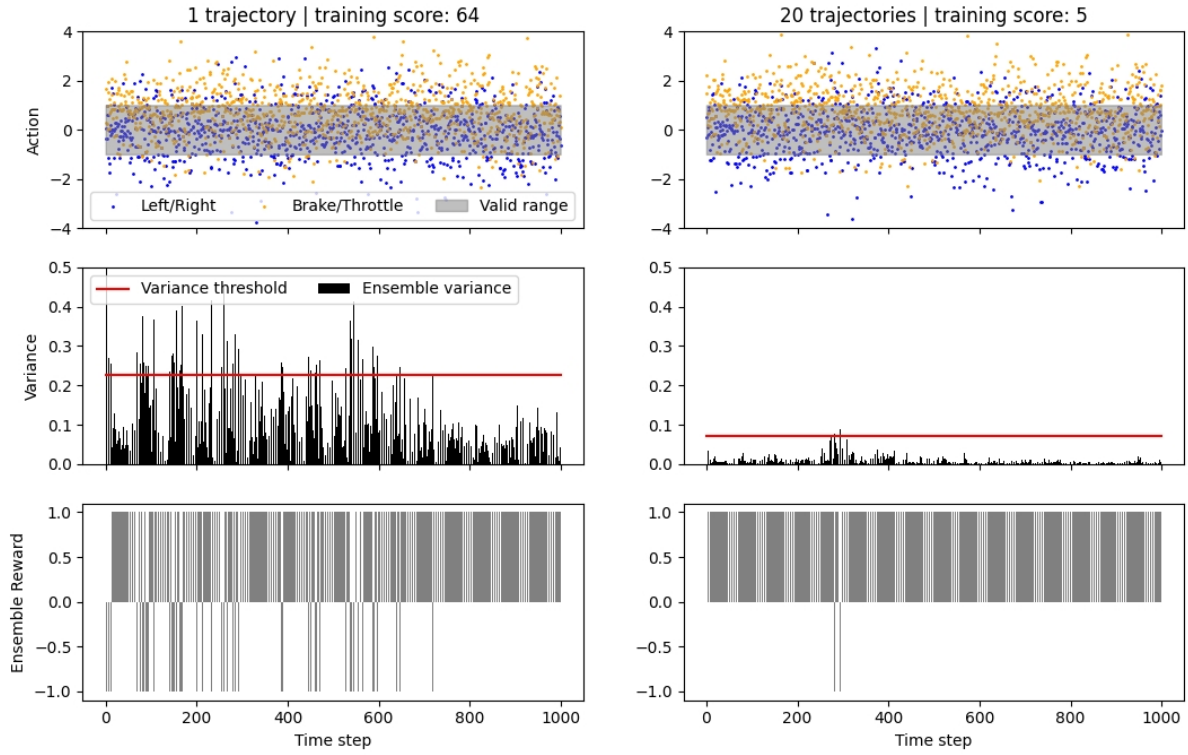

Figure 41 – **CarRarcing**: DRIL training ensemble signal. Top: actions taken by the agent in small dots along with valid range for actions in grey color. Middle: we can observe that for one trajectory (left column), there is a larger presence of disagreement (variance) in the ensemble than for twenty trajectories in the training set (right column). Bottom: the rewards computed from the ensemble's variance.

distribution states, while the ensemble trained with 20 trajectories, almost did not signal any out-of-the-expert-distribution states. This behavior is expected, because there should be more known states to the policy trained with a larger number of states. Nonetheless, as we will see in the next section, the DRIL algorithm still substantially improves the performance of the policy trained with 20 trajectories, in spite of a more sparse negative ensemble reward.

6.7 EVALUATION OF DRIL WITH ACTION-BOUNDED DATASET

Observing the deteriorating performance of the agent's policy throughout the training period, we saved the model at its peak performance (*DRIL peak*) and compared it to the model trained up to the final timestep (3,000) without early stopping (*DRIL final*). These results are shown in Tab. 18.

When trained with only one expert demonstration, the DRIL method in deterministic mode without early stopping allowed for episodes with slightly higher scores than those obtained with pure Behavior Cloning while DRIL with twenty trajectories did

not surpass pure BC (Table 18). On the other hand, by evaluating the model with peak performance in stochastic mode, by early stopping the DRIL training, both the cases of 1 and 20 trajectories in the training set largely outperform their respective pure BC agents in either deterministic or stochastic mode. Evaluation scores are shown in greater detail in Appendix A, Fig. A.3.

Table 18 – **CarRarcing**: Comparison of average scores obtained in 100 consecutive episodes for policies trained with Behavior Cloning, DRIL up to 3,000 episodes and DRIL peak performing policy during training. Experiments used bounded action datasets.

			Gaussian Policy	
			1 trajectory	20 trajectories
Bounded Expert	BC	Deterministic	194±113	617±260
		Stochastic	75±47	137±70
	DRIL peak	Deterministic	235±123	656±266
		Stochastic	**341±246**	**720±289**
	DRIL final	Deterministic	202±128	246±120
		Stochastic	184±130	240±129

6.8 CONCLUSION

In this chapter, we evaluated DRIL in the CarRacing task using demonstrations from a Gaussian policy expert with clipped actions and Beta policy expert with bounded actions. This Beta expert produces actions exactly within the valid range of the action space without having to clip the action values, for they are bounded by construction.

We conclude that DRIL, carried out for approximately 3 million steps only slightly improves policy performance compared to plain BC for datasets with one trajectory and does not improve on average for datasets with 20 expert demonstrations. This happens because the training of a DRIL agent is not able to sustain in a high-performing state, probably due to a catastrophic forgetting situation taking place in the middle of the training process, which can be verified by a increase and subsequent decrease in the agent's score. Other changes such as, lower learning rate, longer rollout, more policies in the ensemble, more units per layer in the ensemble, longer training, did not prevent this catastrophic forgetting effect.

To overcome this undesirable situation, we introduced a novelty consisting of storing the policy parameters obtained at peak performance during training, which is computed as the average of the policy score on the last 10 episodes of training. This avoids forgetting the best policy seen so far in training.

In summary, we have considered the problem of imitation learning of an agent in the CarRacing task, which is a problem with a high-dimensional observation space and a continuous action space. We found that running DRIL for the CarRacing task until

Table 19 – **CarRarcing**: Comparison of average scores obtained in 100 consecutive episodes for policies trained with Behavior Cloning, DRIL up to 3,000 episodes and DRIL peak performing policy during training. Experiments used clipped and bounded action datasets.

			Gaussian Policy	
			1 trajectory	20 trajectories
Clipped Expert	BC	Deterministic	125±113	171±124
		Stochastic	30±67	473±115
	DRIL peak	Deterministic	166±131	423±211
		Stochastic	**322±208**	**802±197**
	DRIL final	Deterministic	41±80	229±107
		Stochastic	39±79	218±108
Bounded Expert	BC	Deterministic	194±113	617±260
		Stochastic	75±47	137±70
	DRIL peak	Deterministic	235±123	656±266
		Stochastic	**341±246**	**720±289**
	DRIL final	Deterministic	202±128	246±120
		Stochastic	184±130	240±129

the training stabilizes (by observing the agent's score), does not considerably outperform behavior cloning (BC). However, by applying an "early-stopping" strategy, we can significantly improve performance compared to BC. This conclusion holds for both clipped and bounded action datasets, generated from Gaussian and Beta distributions, respectively, as evidenced in Table 19. This conclusion is particularly relevant for problems in which behavior cloning alone does not reach expert-level performance.

7 DRIL-BETA

Proximal Policy Optimization as originally proposed utilizes a Gaussian distribution (PPO-Gaussian). We noted in our article (Petrazzini; Antonelo, Eric A., 2021) that an adapted version of the PPO using a Beta distribution (PPO-Beta) achieved better than state-of-the-art results.

These improved results mostly result from the fact that the Gaussian distribution has an infinite support. Having an infinite support causes the output of the policy to constantly fall off the valid action range, thus saturating the actuators.

In this chapter, we will dive into DRIL and BC agents whose policy's outputs parametrizes Beta distributions, since the latter led to state-of-the-art performance in the CarRacing task for pure RL agents trained by PPO (Petrazzini; Antonelo, Eric A., 2021).

7.1 POLICY STRUCTURE

The policy structure was kept as similar as possible to the structure originally proposed with the DRIL algorithm . The feature extraction portion of the ANN is kept identical, i.e., all layers except the output layer. The latter was changed as described next.

The output layer has not anymore only two units representing the means of a Gaussian distribution (for both actions steering and acceleration), since it models now the parameters of a Beta distribution.

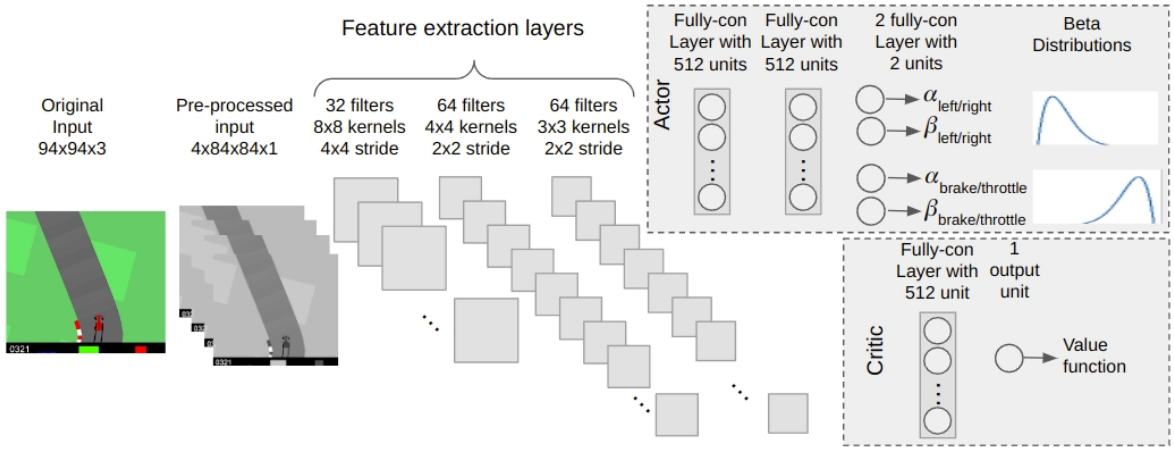

Figure 42 – Policy structure for the DRIL-Beta algorithm. Output layer of the actor now models two Beta distributions for both actions.

Since the Beta distribution requires two positive inputs, α and β, we apply a softplus function with a slope of 20 to the policy's logits. We then proceed to use these resulting positive parameters in the Diagonal Beta distribution, which allows for two

independent Beta distributions, as shown in Fig. 42. Thus, four output units in the Actor network are able to model two Beta distributions, for steering and acceleration.

From now on, we call *Beta policy* the policy whose outputs parameterize a Beta distribution.

7.2 BEHAVIOR CLONING

For behavior cloning, we followed the same procedure used in (Brantley; Sun; Henaff, 2020). We split the expert demonstration state-action pairs randomly in training data and validation data using a 80/20 proportion. After training starts, we progressively save the optimal weights and stop training when the policy fails to improve validation set error for 20 consecutive epochs.

At initialization, policy parameters are randomly set, resulting in a distribution that spans the entire support of the Beta distribution, as illustrated in Fig. 43.

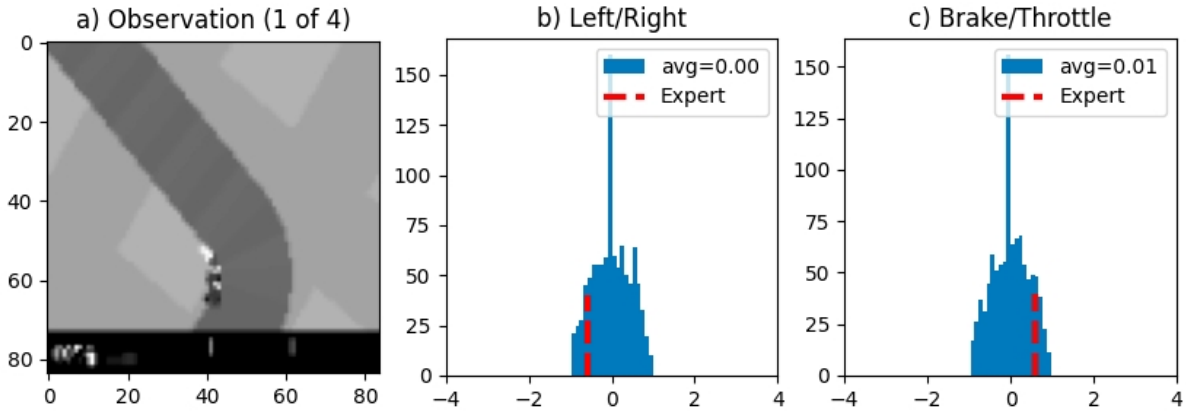

Figure 43 – Randomly initialized Beta policy outputs α and β parameters leading to a Beta distribution covering the entire support ($[-1, +1]$) and concentrated towards the center. In (a), we can see the first snapshopt (out of four) of the environment at a turn received as agent's observation, which remains fixed while the policy is sampled 2,000 times. In (b) and (c), histograms depict each dimension of the sampled actions, while the desired expert action is seen a dashed red vertical line.

For a policy with the Beta distribution, we minimize the loss between the mean of the Beta distribution and the expert actions. Notice that the mean of the Beta distribution is given by $\mu = \alpha/(\alpha + \beta)$ and the variance by $\sigma^2 = \alpha * \beta/((\alpha^2 + \beta^2)(\alpha + \beta + 1))$. As a result, with the adjustment of α and β as outputs of the policy network, the variance of the distribution also changes along the training. This is was not the case for the employed Gaussian policy in the previous chapter, as only its mean parameter was adapted through learning.

After training, we run 100 consecutive episodes to evaluate the performance of each policy. Average scores and standard deviation are shown in Table 20 and a detailed chart

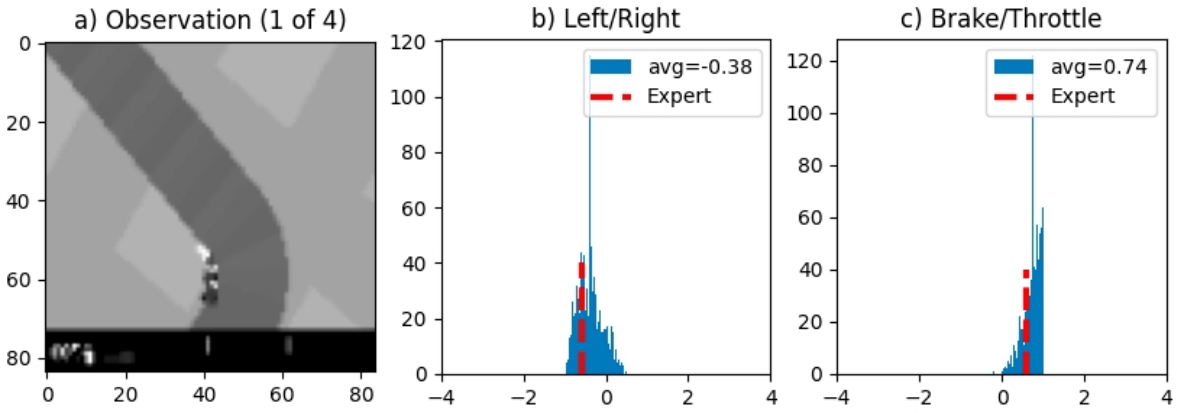

Figure 44 – **CarRacing**: Evolution of the policies after training. Optimizing the loss for the Beta distribution mean also reduces the variance of the distribution.

with individual episode scores is presented in Appendix A, Fig. A.1.

Table 20 – **CarRarcing**: Stochastic, Behavior Cloning Policies with Beta distribution have superior performance for both clipped and bounded-action expert demonstrations.

			Beta Policy	
			1 trajectory	20 trajectories
Clipped Expert	BC	Deterministic	163±67	252±143
		Stochastic	**216±106**	**758±237**
Bounded Expert	BC	Deterministic	147±105	567±239
		Stochastic	**161±156**	**794±227**

7.3 ENSEMBLE TRAINING

It is worth noticing that the ensemble of DRIL-Beta is exactly the same ensemble used in the previous chapter. Thus, only the agent's policy parametrizes a Beta distribution, whereas each policy in the ensemble is a Gaussian policy which minimizes the mean squared error between desired actions and the respective policy network output.

Ensemble policies were trained by behavior cloning with expert demonstrations divided into 80/20 train/validation set. Training was performed for 2,001 epochs for the case of 1 expert trajectory and 20 expert trajectories. Analyzing Fig. 45 with the training error, we can observe that the policies in the ensemble have been trained to overfit, as the validation error is well above the training error. As the ensemble is equivalent to the one used in the previous chapter, the overall shape of the training curves guards a strong similarity with the curves shown in Fig. 35.

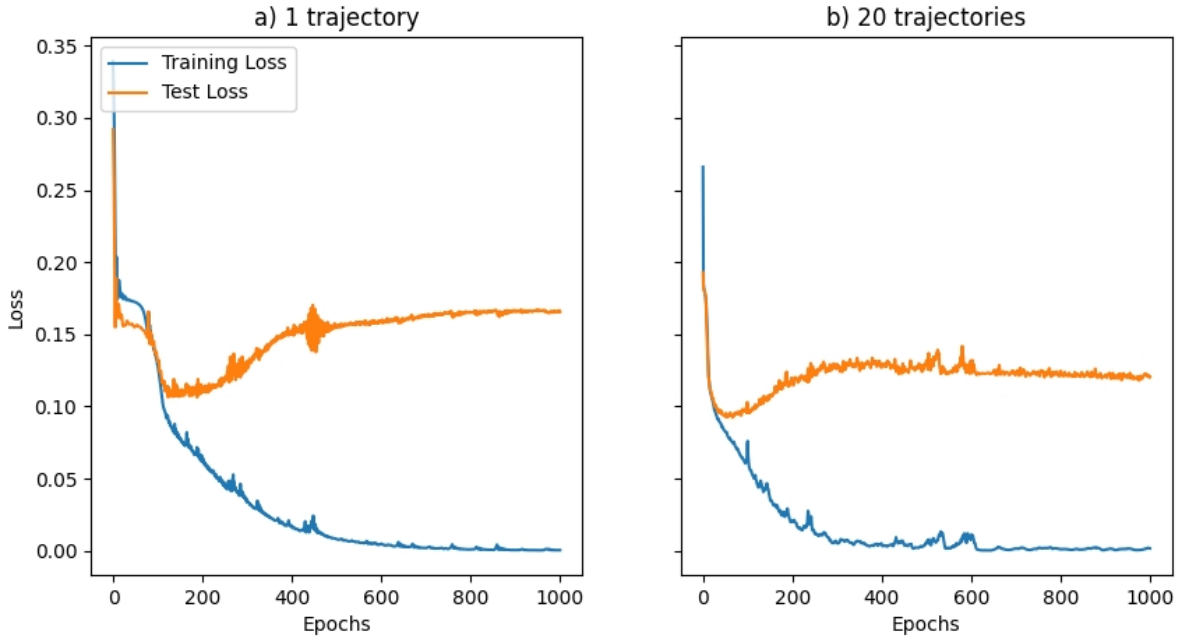

Figure 45 – **CarRacing**: Ensemble are trained to overfit, with test loss consistently above training loss.

7.4 TRAINING FOR THE DRIL-BETA AGENT

Once the ensemble is trained, the agent called DRIL-Beta is pretrained by BC using the complete expert set before it undergoes DRIL training, following the same procedure used in the previous chapter, but now with a Beta policy. After experimenting with the hyperparameters' values suggested for the Atari and robotic control settings, we observed that the robotic control parameters also yielded the best results for the Beta policy. Therefore, we adopted these same hyperparameters values as our starting point for further work.

Before DRIL training starts, as the agent was pretrained by BC, it should have an initial performance in the same level of a BC agent with a Beta policy, which is already relatively high, specially for the case of 20 trajectories in the expert set (Fig. 46). However, the performance of the agent during training by DRIL quickly deteriorates, as shown in the the first row of Fig. 46. On the second row, we plot an average reward for each episode. A mean reward close to 1 indicates that the ensemble policies are not being able to identify states out of the expert distribution. A sparse reward signal presents a significant challenge for the PPO algorithm and we believe this is the main reason behind the policy performance deterioration.

To have a better understanding of the working of the model, we zoom in on the first 1,000 training steps, shown on Fig. 47. On the first row, we observe that all actions fall within the valid range, as expected, since the Beta distribution has a finite support between 0 and 1 that has been mapped into the valid action ranges of the environment,

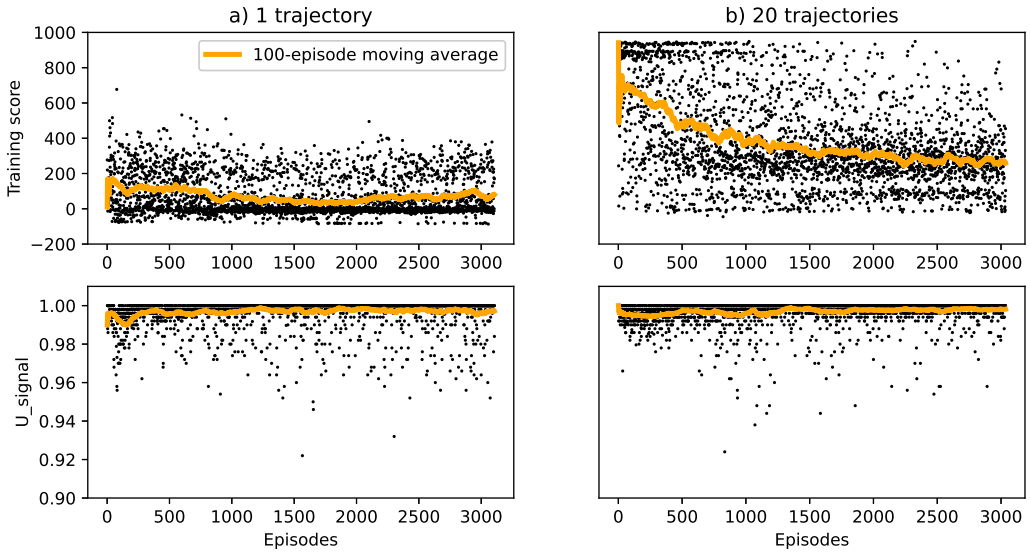

Figure 46 – **CarRacing**: DRIL training for 1 and 20 expert demonstrations with Beta distribution and standard continuous control parameters. Top: training score per episode. Bottom: average reward per episode as per (28). Policy performance decays with training to random values for data set with single demonstration.

which lie in the -1 to 1 range. On the second row, we observe that the variance of the actions do not trespass the variance threshold in both the 1 and 20 trajectories cases. This result is not coherent with the performance of the agent, which reached scores of 402 and 885 in each case, both below the expert threshold level. This means that the vehicle in both cases drove away of the track. However, that was likely not enough for the ensemble to generate a large variance and consequently a negative reward in enough of the timesteps during the run (just few timesteps with negative reward can be seen in Fig. 47). Therefore, in its current version, the ensemble of DRIL is somewhat limited in its ability to identify states that do not belong to the expert dataset (e.g., driving away of the track).

7.5 EVALUATION DRIL-BETA

As the policy performance deteriorates during training of DRIL-Beta, we apply the early-stopping procedure already presented, which gets the best performing policy with highest 10-episode running average score during training (DRIL peak) and compare its results with BC and DRIL after 3 million steps (DRIL final). Average evaluation scores and standard deviations for the clipped and bounded action datasets are presented in Table 21, while detailed charts for all experiments with unbounded and bounded-action datasets are provided in Appendix Fig. A.2 and A.3, respectively.

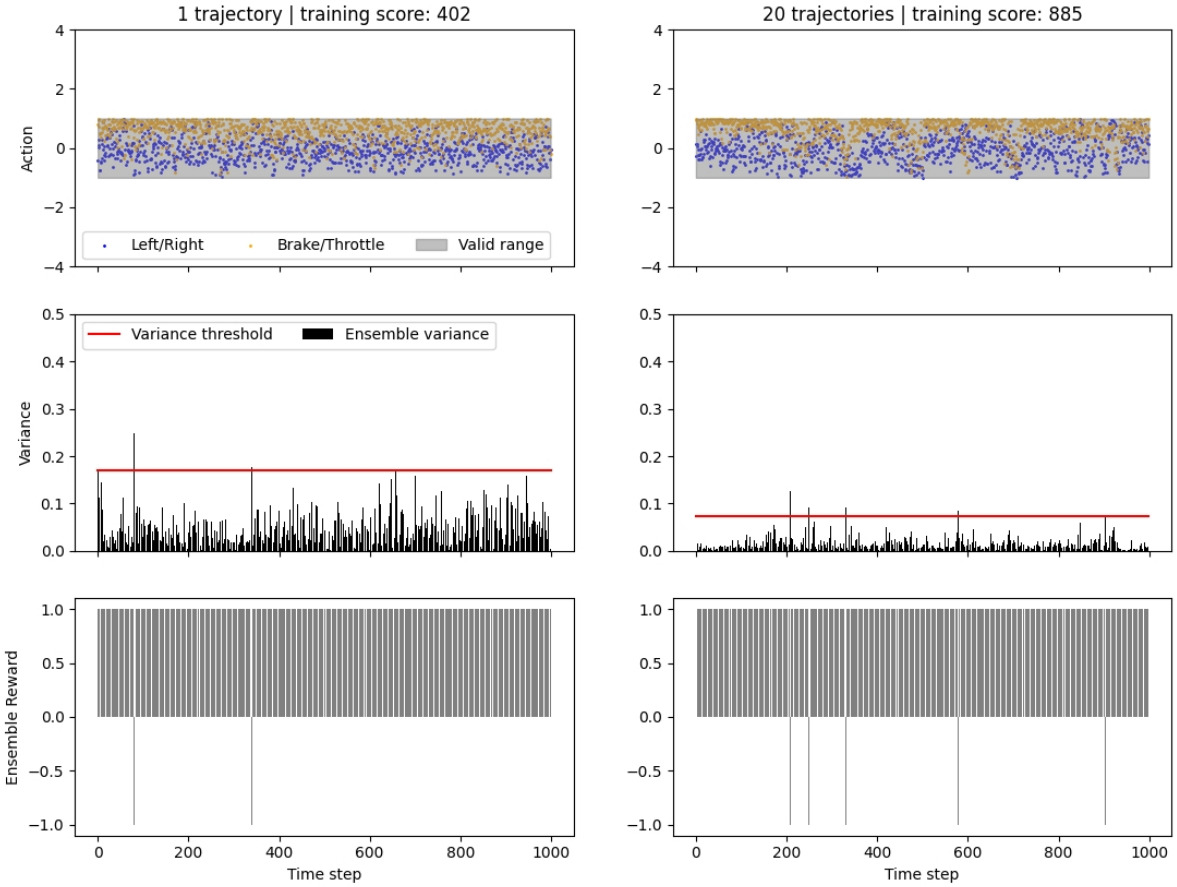

Figure 47 – **CarRarcing**: DRIL training ensemble signal. We can observe that for both 1 and 20 trajectories, there is no presence of negative rewards.

7.6 CONCLUSION

Despite the initial policy deterioration observed during training in DRIL for autonomous racing, the stochastic DRIL-peak policy ultimately manages to slightly outperform behavior cloning when applied to a dataset consisting of just 1 expert trajectory, both in deterministic and stochastic modes, for both clipped and bounded action experts as shown in Table 21.

Behavior cloning proves particularly competitive, achieving values exceeding 750 in stochastic mode. It is worth noting that the DRIL training begins with a highly capable behavior cloning policy, potentially limiting DRIL's ability to make even small improvements, as observed in the case of smaller datasets.

Table 21 – **CarRarcing**: Comparison of average scores obtained in 100 consecutive episodes for deterministic and stochastic Beta policies trained with Behavior Cloning, DRIL peak performing policy during training and DRIL up to 3,000 episodes.

			Beta Policy	
			1 trajectory	20 trajectories
Clipped Expert	BC	Deterministic	163±67	252±143
		Stochastic	216±106	**758±237**
	DRIL peak	Deterministic	229±142	210±115
		Stochastic	**348±230**	572±280
	DRIL final	Deterministic	122±116	265±165
		Stochastic	143±111	241±140
Bounded Expert	BC	Deterministic	147±105	567±239
		Stochastic	161±156	**794±227**
	DRIL peak	Deterministic	195±88	398±225
		Stochastic	**200±184**	738±277
	DRIL final	Deterministic	123±125	275±144
		Stochastic	99±121	267±153

8 DISCUSSION

Our study centered on evaluating three imitation learning methods applied to the CarRacing autonomous driving setting: BC (Behavior Cloning), DRIL-peak, and DRIL-final, each in two variants employing Gaussian and Beta distributions. These methodologies underwent testing on four distinct datasets created by experts trained with PPO, featuring trajectories with actions either clipped and bounded, grouped into sets of 1 and 20. All different combinations of datasets and agents underwent a standardized evaluation encompassing 100 consecutive episodes after training the respective agent. The summary with average scores and standard deviations is presented in Table 22, while detailed training curves and episodic scores can be found in Appendix A (Fig. A.2 to A.4).

Key insights regarding method performance within each dataset are as follows:

- **1 trajectory from a clipped action expert:** The stochastic DRIL-peak policy with Beta distribution achieved the top performance (348±230), closely trailed by its Gaussian counterpart (322±208);
- **20 trajectories from a clipped action expert:** The stochastic DRIL-peak policy with Gaussian distribution secured the highest scores (802±197), followed closely by stochastic BC with the Beta distribution (758±237). As anticipated, augmenting the dataset size from 1 to 20 trajectories generally led to improved performance, except for the deterministic DRIL-peak with Beta distribution;
- **1 trajectory from a bounded action expert:** The stochastic DRIL-peak policy with Gaussian distribution (341±246) significantly outperformed other methods within this data group;
- **20 trajectories from a bounded action expert:** Stochastic BC achieved the top evaluation score (794±227), with stochastic DRIL-peak using the Gaussian distribution as a close contender (720±289). Since DRIL Beta failed to exhibit a peak during DRIL training in this dataset section, the stochastic DRIL peak with Beta is disregarded as it represents a degraded version of the initial pre-trained policy, which posted the best results in this dataset section. As expected, increasing number of trajectories within the dataset from 1 to 20 led to improved performance in all cases.

Notably, the maximum scores achieved through imitation learning with only 1 demonstration from clipped and bounded action experts were very close. These high scores were attained by stochastic DRIL-peak policies, though with different distributions: 348±230 using the Beta distribution and 341±246 using the Gaussian distribution. A similar phenomenon occurred in the case of 20 trajectories, where Stochastic DRIL-peak with Gaussian and Stochastic BC with Beta achieved scores of 802±197 and 794±227, respectively.

Table 22 – **CarRacing:** Summary results for Beta and Gaussian Policies.f

			1 trajectory		20 trajectories	
			Gaussian	Beta	Gaussian	Beta
Clipped Expert	BC	Deterministic	125±113	163±67	171±124	252±143
		Stochastic	30±67	216±106	473±115	758±237
	DRIL peak	Deterministic	166±131	229±142	423±211	210±115
		Stochastic	322±208	**348±230**	**802±197**	572±280
	DRIL final	Deterministic	41±80	122±116	229±107	265±165
		Stochastic	39±79	143±111	218±108	241±140
Bounded Expert	BC	Deterministic	194±113	147±105	617±260	567±239
		Stochastic	75±47	161±156	137±70	**794±227**
	DRIL peak	Deterministic	235±123	195±88	656±266	398±225
		Stochastic	**341±246**	200±184	720±289	738±277
	DRIL final	Deterministic	202±128	123±125	246±120	275±144
		Stochastic	184±130	99±121	240±129	267±153

Lastly, we present our best results in Table 23 and compare them to scores obtained by other Imitation Learning methods for CarRacing. While the comparison may not be rigorous due to the use of different expert demonstrations in the methods from the literature and our own, it does provide an indication of the range of performance achieved by other works using the CarRacing task as their testbed. In this sense, our results align with recent Imitation Learning methods, including Generative Adversarial Imitation Learning and its variants.

Table 23 – **CarRarcing**: Comparison of our best results with other Imitation learning scores reported by (Jena; Liu; Sycara, 2021)

Algorithm	Score
Random	-75 ± 4
BC	696 ± 98
GAIL	420 ± 199
BC+GAIL	595 ± 263
Augumented GAIL with BC	732 ± 46
DRIL-peak Gaussian Stochastic (Ours)	**802 ± 197**
BC Beta Stochastic (Ours)	**794 ± 227**

9 CONCLUSION AND FUTURE WORK

In this work, we successfully reproduced the results originally obtained in (Brantley; Sun; Henaff, 2020) for selected Atari and Continuous Control environments. Subsequently, we expanded the algorithm to make it applicable to the Autonomous Vehicle setting. This involved merging the Atari and Continuous Control policy structures to create a framework capable of operating in a pixel observation space with a continuous action space. We selected the CarRacing-v0 environment as our testbed for this modified version of the algorithm.

Our initial hypothesis was that DRIL would outperform BC in the autonomous vehicle environment. However, we found that the DRIL algorithm, as originally proposed, did not outperform Behavior Cloning.

For this reason, we introduced several innovations, namely:

1. A stochastic policy that was trained with Proximal Policy Optimization using a Beta distribution, achieving state-of-the-art performance in solving the CarRacing-v0 environment and generating bounded-action expert demonstrations;
2. We achieved superior performance with a stochastic policy trained using DRIL compared to policies trained with Behavior Cloning in three out of four datasets by implementing an 'early-stopping' approach during training. This adaptation has led us to introduce the term 'DRIL-peak' to describe this improved strategy;
3. In developing a version of the DRIL algorithm using a Beta distribution, we found out that a stochastic BC policy with Beta distribution is a competitive alternative to DRIL-peak in stochastic mode.

In future research, we envision several promising avenues for expanding the application of DRIL in the field of autonomous driving. These include:

- Applying DRIL to more advanced autonomous driving simulators, such as CARLA (Dosovitskiy et al., 2017);
- Exploring various artificial neural network (ANN) architectures such as those with attention mechanisms to enhance DRIL's effectiveness;
- Experimenting with DRIL in conjunction with more robust reinforcement learning methods;
- Testing an ensemble where each of its policy is trained as a Beta policy instead of a Gaussian policy;
- Assessing alternative approaches for detecting states that deviate from expert distributions, such as employing an ensemble of policies with diverse architectures.

REFERENCES

ARULKUMARAN, Kai; LILLRANK, Dan Ogawa. A Pragmatic Look at Deep Imitation Learning. **CoRR**, abs/2108.01867, 2021. arXiv: 2108.01867.

BOJARSKI, Mariusz et al. **End to End Learning for Self-Driving Cars**. [S.l.: s.n.], 2016. arXiv: 1604.07316 [cs.CV].

BRANTLEY, Kianté; SUN, Wen; HENAFF, Mikael. Disagreement-Regularized Imitation Learning. In: INTERNATIONAL Conference on Learning Representations. [S.l.: s.n.], 2020.

BROCKMAN, Greg; CHEUNG, Vicki; PETTERSSON, Ludwig; SCHNEIDER, Jonas; SCHULMAN, John; TANG, Jie; ZAREMBA, Wojciech. **OpenAI Gym**. [S.l.: s.n.], 2016. eprint: arXiv:1606.01540.

CHOU, Po-Wei; MATURANA, Daniel; SCHERER, Sebastian. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In: PMLR. INTERNATIONAL conference on machine learning. [S.l.: s.n.], 2017. P. 834–843.

COUTO, Gustavo Claudio Karl; ANTONELO, Eric Aislan. Hierarchical Generative Adversarial Imitation Learning with Mid-level Input Generation for Autonomous Driving on Urban Environments. **arXiv preprint arXiv:2302.04823**, 2023.

DADASHI, Robert; HUSSENOT, Léonard; GEIST, Matthieu; PIETQUIN, Olivier. **Primal Wasserstein Imitation Learning**. [S.l.: s.n.], 2021. arXiv: 2006.04678 [cs.LG].

DOSOVITSKIY, Alexey; ROS, German; CODEVILLA, Felipe; LOPEZ, Antonio; KOLTUN, Vladlen. CARLA: An Open Urban Driving Simulator. In: PROCEEDINGS of the 1st Annual Conference on Robot Learning. [S.l.: s.n.], 2017. P. 1–16.

GAIER, Adam; HA, David. Weight agnostic neural networks. **arXiv preprint arXiv:1906.04358**, 2019.

GOODFELLOW, Ian J.; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. Cambridge, MA, USA: MIT Press, 2016. http://www.deeplearningbook.org.

HA, David; SCHMIDHUBER, Jürgen. **Recurrent World Models Facilitate Policy Evolution**. [S.l.: s.n.], 2018. arXiv: 1809.01999 [cs.LG].

HAAN, Pim de; JAYARAMAN, Dinesh; LEVINE, Sergey. Causal Confusion in Imitation Learning. **CoRR**, abs/1905.11979, 2019. arXiv: 1905.11979.

HILL, Ashley et al. **Stable Baselines**. [S.l.]: GitHub, 2018. `https://github.com/hill-a/stable-baselines`.

HO, Jonathan; ERMON, Stefano. **Generative Adversarial Imitation Learning**. [S.l.: s.n.], 2016. arXiv: `1606.03476 [cs.LG]`.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. **Neural Networks**, v. 2, n. 5, p. 359–366, 1989.

JENA, Rohit; LIU, Changliu; SYCARA, Katia. Augmenting GAIL with BC for sample efficient imitation learning. **arXiv preprint arXiv:2001.07798**, 2020.

JENA, Rohit; LIU, Changliu; SYCARA, Katia. Augmenting GAIL with BC for sample efficient imitation learning. In: KOBER, Jens; RAMOS, Fabio; TOMLIN, Claire (Eds.). **Proceedings of the 2020 Conference on Robot Learning**. [S.l.]: PMLR, 16–18 Nov 2021. v. 155. (Proceedings of Machine Learning Research), p. 80–90.

KINGMA, Diederik P; BA, Jimmy. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

KNOX, W. Bradley; ALLIEVI, Alessandro; BANZHAF, Holger; SCHMITT, Felix; STONE, Peter. **Reward (Mis)design for Autonomous Driving**. [S.l.: s.n.], 2022. arXiv: `2104.13906 [cs.LG]`.

LU, Yiren et al. **Imitation Is Not Enough: Robustifying Imitation with Reinforcement Learning for Challenging Driving Scenarios**. [S.l.: s.n.], 2022. arXiv: `2212.11419 [cs.AI]`.

MNIH, Volodymyr et al. Human-level control through deep reinforcement learning. **nature**, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015.

PETRAZZINI, Irving G. B.; ANTONELO, Eric A. Proximal Policy Optimization with Continuous Bounded Action Space via the Beta Distribution. In: 2021 IEEE Symposium Series on Computational Intelligence (SSCI). [S.l.: s.n.], 2021. P. 1–8.

POMERLEAU, Dean. ALVINN: An Autonomous Land Vehicle In a Neural Network. In: TOURETZKY, D.S. (Ed.). **Proceedings of (NeurIPS) Neural Information Processing Systems**. [S.l.]: Morgan Kaufmann, Dec. 1989. P. 305–313.

RISI, Sebastian; STANLEY, Kenneth O. Deep neuroevolution of recurrent and discrete world models. In: PROCEEDINGS of the Genetic and Evolutionary Computation Conference. [S.l.: s.n.], 2019. P. 456–462.

RODRIGUES, Pedro; VIEIRA, Susana. Optimizing Agent Training with Deep Q-Learning on a Self-Driving Reinforcement Learning Environment. In: IEEE. 2020 IEEE Symposium Series on Computational Intelligence (SSCI). [S.l.: s.n.], 2020. P. 745–752.

ROSS, Stephane; BAGNELL, Drew. Efficient Reductions for Imitation Learning. In: TEH, Yee Whye; TITTERINGTON, Mike (Eds.). **Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics**. Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010. v. 9. (Proceedings of Machine Learning Research), p. 661–668.

ROSS, Stéphane; GORDON, Geoffrey J.; BAGNELL, J. Andrew. No-Regret Reductions for Imitation Learning and Structured Prediction. **CoRR**, abs/1011.0686, 2010. arXiv: 1011.0686.

SAE. **SAE Society of Autonomous Engineers**. [S.l.: s.n.], 2014. https://www.sae.org/blog/sae-j3016-update. Accessed: 2023-08-30.

SASAKI, Fumihiro; YAMASHINA, Ryota. Behavioral Cloning from Noisy Demonstrations. In: 9TH International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. [S.l.]: OpenReview.net, 2021.

SCHULMAN, John; LEVINE, Sergey; ABBEEL, Pieter; JORDAN, Michael; MORITZ, Philipp. Trust region policy optimization. In: PMLR. INTERNATIONAL conference on machine learning. [S.l.: s.n.], 2015. P. 1889–1897.

SCHULMAN, John; MORITZ, Philipp; LEVINE, Sergey; JORDAN, Michael; ABBEEL, Pieter. High-dimensional continuous control using generalized advantage estimation. **arXiv preprint arXiv:1506.02438**, 2015.

SCHULMAN, John; WOLSKI, Filip; DHARIWAL, Prafulla; RADFORD, Alec; KLIMOV, Oleg. Proximal policy optimization algorithms. **arXiv preprint arXiv:1707.06347**, 2017.

STANLEY, Kenneth O.; MIIKKULAINEN, Risto. Evolving Neural Networks Through Augmenting Topologies. **Evolutionary Computation**, v. 10, n. 2, p. 99–127, 2002.

SUCH, Felipe Petroski; MADHAVAN, Vashisht; CONTI, Edoardo; LEHMAN, Joel; STANLEY, Kenneth O.; CLUNE, Jeff. **Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning**. [S.l.: s.n.], 2018. arXiv: 1712.06567 [cs.NE].

SUTTON, Richard S; BARTO, Andrew G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018.

SUTTON, Richard S; MCALLESTER, David A; SINGH, Satinder P; MANSOUR, Yishay. Policy gradient methods for reinforcement learning with function approximation. In: ADVANCES in neural information processing systems. [S.l.: s.n.], 2000. P. 1057–1063.

TODOROV, Emanuel; EREZ, Tom; TASSA, Yuval. MuJoCo: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. [S.l.: s.n.], 2012. P. 5026–5033.

WU, Yueh-Hua; CHAROENPHAKDEE, Nontawat; BAO, Han; TANGKARATT, Voot; SUGIYAMA, Masashi. **Imitation Learning from Imperfect Demonstration**. [S.l.: s.n.], 2019. arXiv: `1901.09387 [cs.LG]`.

XIAO, Yi; CODEVILLA, Felipe; GURRAM, Akhil; URFALIOGLU, Onay; LÓPEZ, Antonio M. Multimodal End-to-End Autonomous Driving. **CoRR**, abs/1906.03199, 2019. arXiv: `1906.03199`.

ZHANG, Songyuan; CAO, Zhangjie; SADIGH, Dorsa; SUI, Yanan. **Confidence-Aware Imitation Learning from Demonstrations with Varying Optimality**. [S.l.: s.n.], 2022. arXiv: `2110.14754 [cs.LG]`.

ZHANG, Zhejun; LINIGER, Alexander; DAI, Dengxin; YU, Fisher; GOOL, Luc Van. **End-to-End Urban Driving by Imitating a Reinforcement Learning Coach**. [S.l.: s.n.], 2021. arXiv: `2108.08265 [cs.CV]`.

Appendix

APPENDIX A – DETAILED EXPERIMENT RESULTS

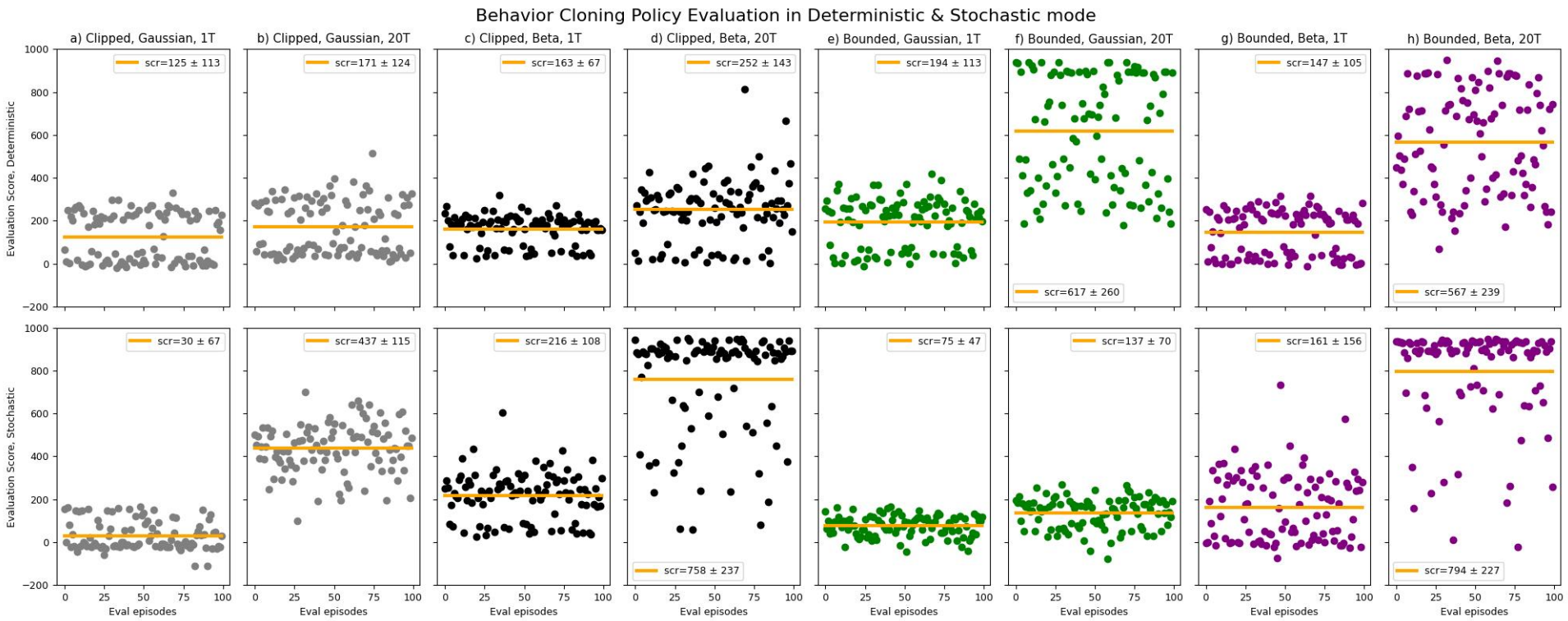

Figure A.1 – **CarRacing:** Policies trained using BC (Behavior Cloning) were assessed over 100 consecutive episodes, both in deterministic and stochastic modes. The results are presented in the first and second rows, respectively, with each dot representing the score achieved in an individual evaluation episode. The orange lines signify the averages, which are accompanied by standard deviations in the legends.The policies are color-coded as follows: Gray for the clipped action dataset with Gaussian BC policy, Black for the clipped action dataset with Beta BC, Green for the bounded action dataset with Gaussian BC, and Purple for the bounded action dataset with Beta BC. Experiments were performed with 1 and 20 trajectories, as indicated on subtitles.

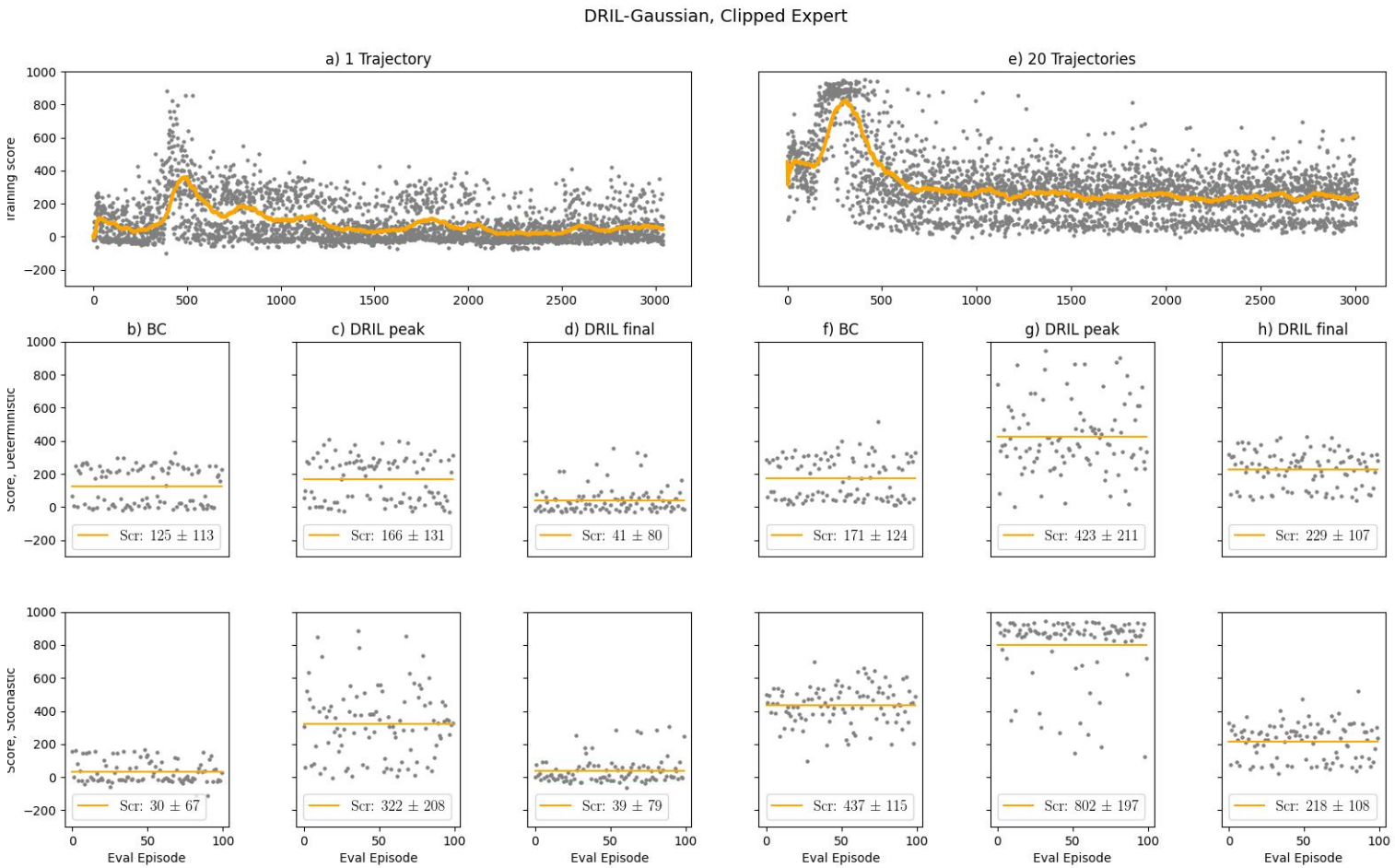

Figure A.2 – **CarRacing:** Training and evaluation results for the DRIL-Gaussian policy trained on a clipped action dataset. a) Training scores for a dataset with 1 trajectory over 3 million steps, resulting in approximately 3,000 episodes. The policy is initially pre-trained with BC, as reiterated in (b) for reference regarding its starting point performance. Training halts at its peak, around the 600th episode. Subsequently, we present the scores of 100 consecutive episodes in (c), featuring both deterministic (top) and stochastic (bottom) modes. To provide a basis for comparison, we conduct the same evaluation with parameters obtained after 3 million steps, and these results are displayed in (d). Charts (e) to (h) show results for the same experiment using a 20-trajectory dataset.

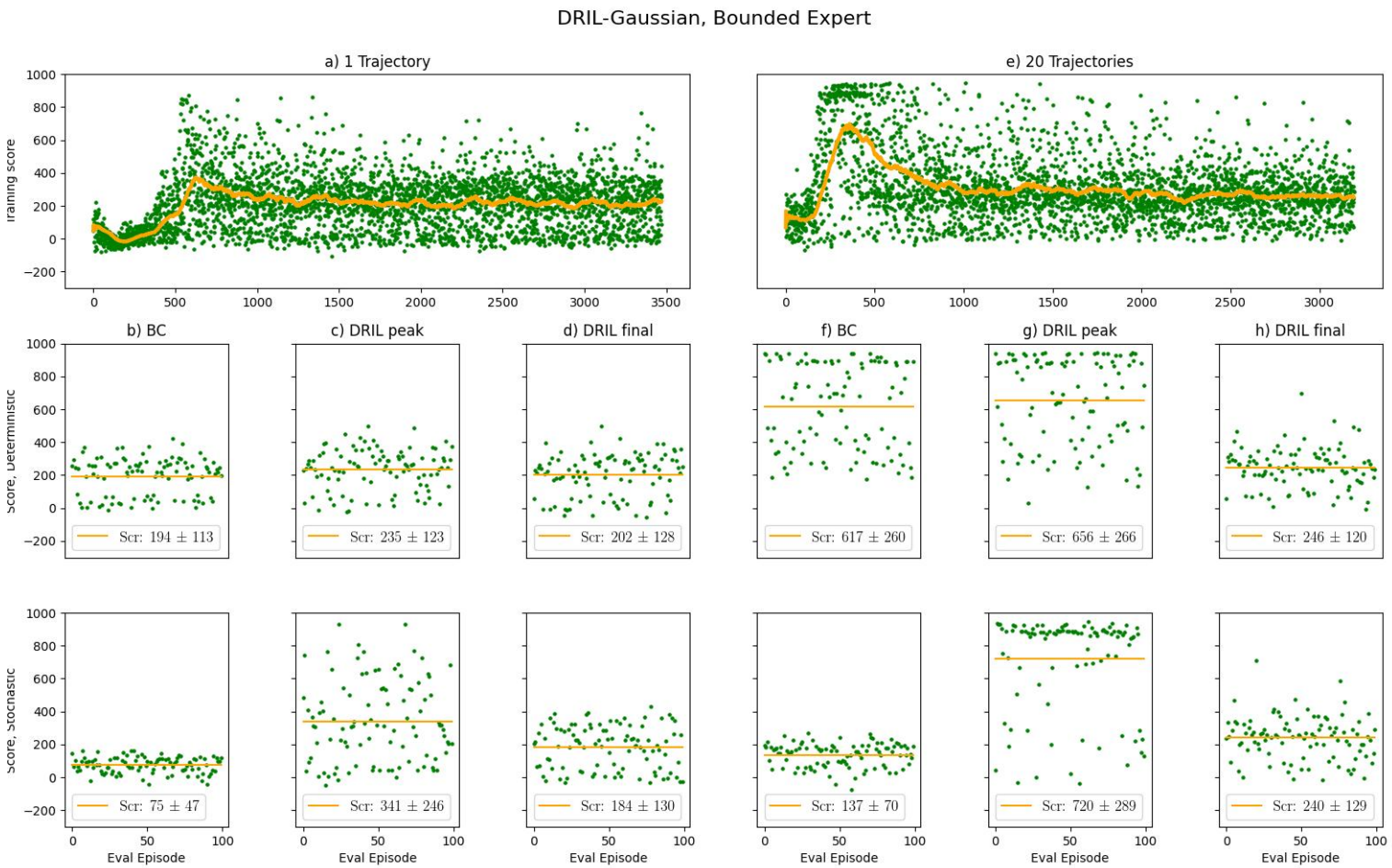

Figure A.3 – **CarRacing:** Training and evaluation results for the DRIL-Gaussian policy trained on a bounded action dataset. a) Training scores for a dataset with 1 trajectory over 3 million steps, resulting in approximately 3,000 episodes. The policy is initially pre-trained with BC, as reiterated in (b) for reference regarding its starting point performance. Training halts at its peak, around the 600th episode. Subsequently, we present the scores of 100 consecutive episodes in (c), featuring both deterministic (top) and stochastic (bottom) modes. To provide a basis for comparison, we conduct the same evaluation with parameters obtained after 3 million steps, and these results are displayed in (d). Charts (e) to (h) show results for the same experiment using a 20-trajectory dataset.

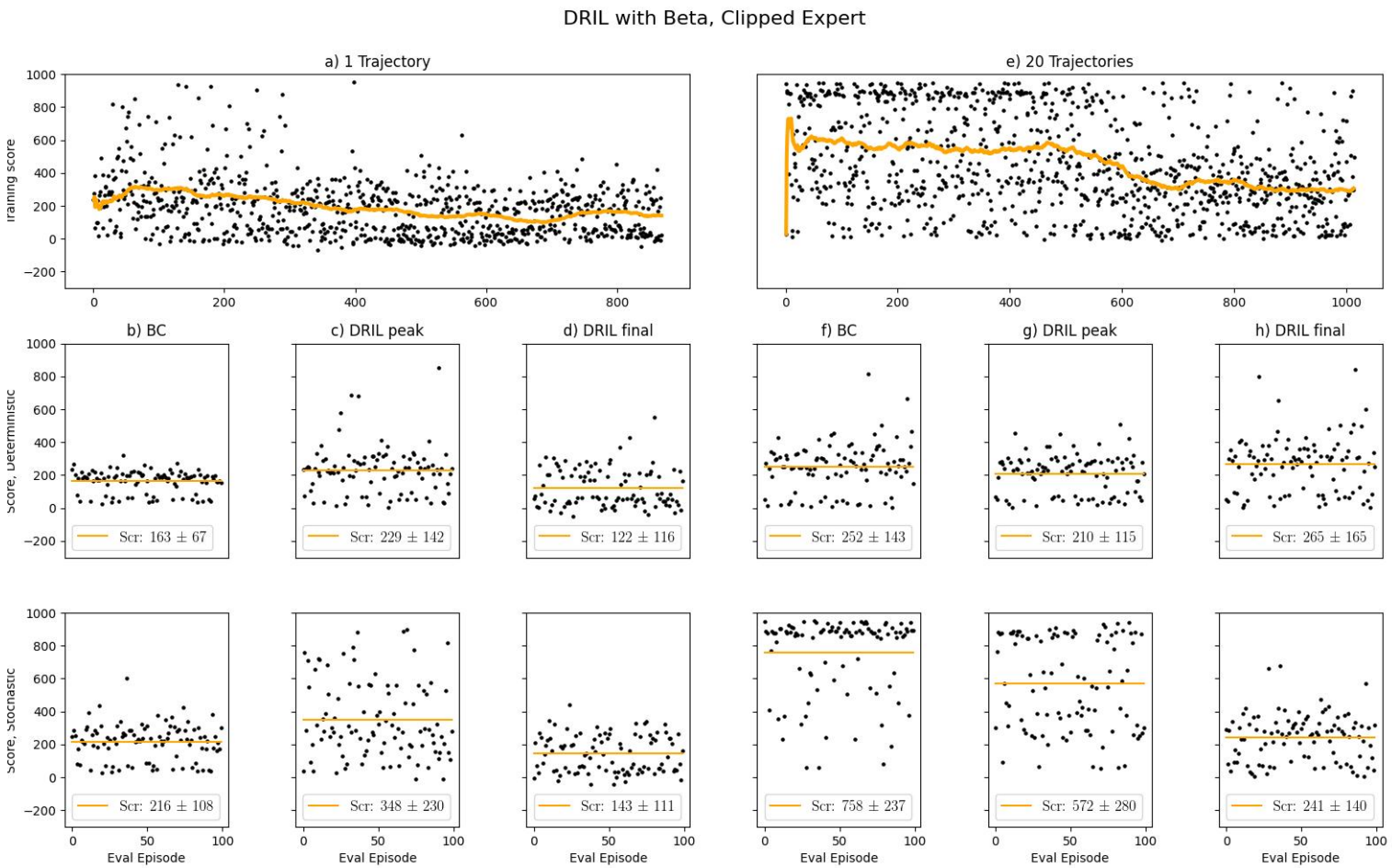

Figure A.4 – **CarRacing:** Training and evaluation results for the DRIL-Beta policy trained on a clipped action dataset. a) Training scores for a dataset with 1 trajectory over 3 million steps was interrupted around 1 million steps because of errors caused by the high amount of actions at the extreme ends of the Beta distribution support. The policy is initially pre-trained with BC, as reiterated in (b) for reference regarding its starting point performance. Training halts at its peak, around the 600th episode. Subsequently, we present the scores of 100 consecutive episodes in (c), featuring both deterministic (top) and stochastic (bottom) modes. To provide a basis for comparison, we conduct the same evaluation with parameters obtained after 3 million steps, and these results are displayed in (d). Charts (e) to (h) show results for the same experiment using a 20-trajectory dataset.

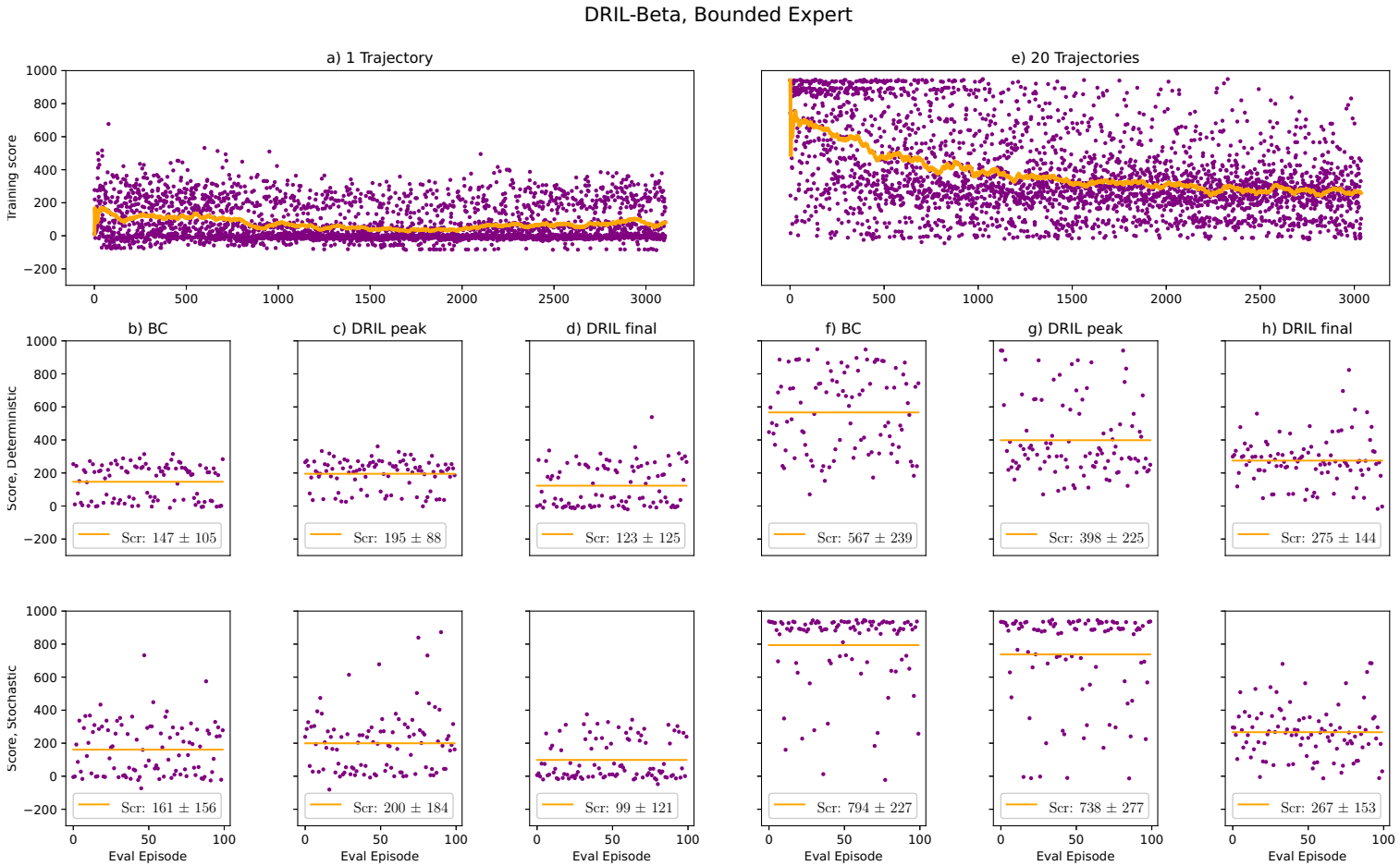

Figure A.5 – **CarRacing:** Training and evaluation results for the DRIL-Beta policy trained on a bounded action dataset. a) Training scores for a dataset with 1 trajectory over 3 million steps, resulting in approximately 3,000 episodes. The policy is initially pre-trained with BC, as reiterated in (b) for reference regarding its starting point performance. Training halts at its peak, around the 600th episode. Subsequently, we present the scores of 100 consecutive episodes in (c), featuring both deterministic (top) and stochastic (bottom) modes. To provide a basis for comparison, we conduct the same evaluation with parameters obtained after 3 million steps, and these results are displayed in (d). Charts (e) to (h) show results for the same experiment using a 20-trajectory dataset.

Annex

ANNEX A – EXPERT AGENT TRAINED WITH PPO VIA BETA DISTRIBUTION

Proximal Policy Optimization with Continuous Bounded Action Space via the Beta Distribution

Irving G. B. Petrazzini
Automation and Systems Engineering
Federal University of Santa Catarina
Florianópolis, Santa Catarina, 88040-900
Email: irving.petrazzini@posgrad.ufsc.br

Eric A. Antonelo
Automation and Systems Engineering
Federal University of Santa Catarina
Florianópolis, Santa Catarina, 88040-900
Email: eric.antonelo@ufsc.br

***Abstract*—Reinforcement learning methods for continuous control tasks have evolved in recent years generating a family of policy gradient methods that rely primarily on a Gaussian distribution for modeling a stochastic policy. However, the Gaussian distribution has an infinite support, whereas real world applications usually have a bounded action space. This dissonance causes an estimation bias that can be eliminated if the Beta distribution is used for the policy instead, as it presents a finite support. In this work, we investigate how this Beta policy performs when it is trained by the Proximal Policy Optimization (PPO) algorithm on two continuous control tasks from OpenAI gym. For both tasks, the Beta policy is superior to the Gaussian policy in terms of agent's final expected reward, also showing more stability and faster convergence of the training process. For the CarRacing environment with high-dimensional image input, the agent's success rate was improved by 63% over the Gaussian policy.**

I. Introduction

Deep Reinforcement Learning (RL) has achieved unprecedented results on challenging high-dimensional continuous state-space problems, surpassing human performance in 29 out of 49 Atari 2600 games in [1], for instance. Later, AlphaGO, an agent that combines reinforcement learning and Monte Carlo balanced search tree algorithms with self play was able beat Lee Sedol, a 9th-dan, world champion [2]. In this context, Convolutional Neural Networks (CNNs) [3] serve as function approximators for the Q-value function, since they can efficiently process image inputs and learn useful feature representations from these high-dimensional continuous state-space domains.

Handling discrete action spaces in a deep reinforcement task usually resumes to defining an output layer of a neural network that has the same dimension of the action space. If this space is small, an action can be easily drawn from the distribution yielded by the layer's activation. Otherwise, finding the best action for high-dimensional or continuous action spaces constitutes an expensive optimization process per se, which needs to be run inside another loop, the agent-environment cycle.

Many interesting real-world problems such as control of robotic arms and autonomous cars require a continuous action space. Instead of modeling the state-action Q-value function, model-free continuous control via reinforcement learning is made possible by directly optimizing a policy function which maps states to probability distributions over continuous action spaces. This family of policy gradient methods have undergone important advancements allowing for high-dimensional continuous state spaces (such as images) and continuous action spaces [4]–[7].

To model a stochastic policy, these methods choose the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, whose parameters μ and σ^2 are to be estimated as outputs of a deep neural network. However, many real-world applications have bounded action spaces, usually owing to physical constraints, e.g., by the joints of a humanoid robot or manipulator, and by the accelerator and steering direction of a vehicle. Thus, in these cases, this Gaussian policy, which has infinite support, introduces a estimation bias since it can give a nonzero probability for *actions* outside the valid action space.

Recently, [8] proposes to model the stochastic policy with the Beta distribution, with parameters α and β, such that the resulting policy has a suitably bounded action space, that presents no bias as the previously considered Gaussian distribution. Instead of the mean and variance, now the outputs of the neural network represent the policy parameters α and β. The Beta distribution can be used with any on- or off-policy methods such as Trust Region Policy Optimization (TRPO) [4] and Actor-Critic Experience Replay (ACER) [7].

So far, the Beta distribution has been evaluated only for TRPO and ACER on a variety of problems. Proximal Policy Optimization (PPO), which evolved from TRPO but has a much simpler implementation and a similar performance to ACER, still lacks experimentation with the Beta distribution. This is the first work to report experiments on PPO with the Beta distribution on RL applications with high-dimensional observation spaces. Besides, our investigation focus on two continuous control applications from OpenAI Gym, the Lunar Lander and the Car Racing, both of which were not considered in [8].

The benefits of the approach are better stability and faster

convergence of the training process. Furthermore, because the estimation bias is absent, the final learned Beta policy is superior to the final Gaussian policy. We also report results better than state-of-the-art on the Car Racing problem.

II. Background

A. Markov decision process

We model our continuous control reinforcement learning task as a finite Markov decision process (MDP). An MDP consists of a state space S, an action space A, an initial state s_0, and a reward function $r(s,a) : S \times A$ that emits a scalar value for any transition from state s taking action a. At each time step t, the agent selects an action a_{t+1} according to a policy π, i.e., $a_{t+1} = \pi(s_t)$, such that agent's future expected reward is maximized. A stochastic policy can be described as a probability distribution of taking an action a_{t+1} given a state s_t denoted as $\pi(a|s) : S \to A$. A deterministic policy can be obtained by taking the expected value of the policy $\pi(a|s)$.

B. Policy Gradient Methods

Value-based reinforcement learning methods first learn to approximate a value function $Q(s,a)$. The policy is obtained by finding the action that maximizes the latter, e.g., $\pi(s) = \arg\max_a Q(s,a)$. On the other hand, policy gradient methods optimize directly an parametrized policy $\pi_\theta(a|s)$ that can model Categorical or Continuous actions for discrete and continuous spaces, respectively.

For a given scalar performance measure $L(\theta) = v_{\pi_\theta}(s_0)$, where v_{π_θ} is the true value function for π_θ, the policy determined by θ, performance is maximized through gradient ascent on L

$$L(\theta) = \int_S \rho^\pi(s) \int_A \pi_\theta(s,a) r(s,a) da ds \tag{1}$$

$$= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[r(s,a)] \tag{2}$$

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla_\theta L(\theta_t)} \tag{3}$$

where $\rho^\pi(s) = \sum_{t=0}^{\infty} \gamma^t p(s_t = s)$ is the unnormalized discounted state visitation frequency in the limit [9] and α is the learning rate.

C. Proximal Policy Optimization

Proximal Policy Optimization [5] is one of the most commonly used policy gradient methods. Among the several variants for the performance measures available, we consider the clipped surrogate objective as in [5], as follows:

$$L_t^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{4}$$

where θ_{old} is the vector of policy parameters before the update; $r_t(\theta)$ denotes the probability ratio $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$; ϵ is a hyperparameter used to clip the probability ratio by $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$, avoiding large policy updates [5]; and $\hat{A}_t$ is an estimator of the advantage function at timestep t, which weights the ratio $r_t(\theta)$. Here, $\hat{\mathbb{E}}_t$ denotes an empirical average over a finite set of samples.

The implementation of policy gradient considers a truncated version of the Generalized Advantage Estimator (GAE), as in [10]:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{5}$$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t), \tag{6}$$

where the policy is run for T timesteps (with T less than the episode size). As commonly used in the literature, γ and λ are discount factor and GAE parameter, respectively. To perform a policy update, each of N (parallel) actors collect T timesteps of data. Then we construct the surrogate loss on these NT timesteps of data, and optimize it with ADAM algorithm [11] with a learning rate α, in mini-batches of size $m \leq NT$ for K epochs. Notice that $V_\theta(s)$ in GAE is learned simultaneously in order to reduce the variance of the advantage-function estimator.

Once we use a neural network architecture that shares parameters between the policy and value function, we must use a loss function that combines the policy surrogate and a value function error term. This objective is further augmented by adding an entropy term to ensure sufficient exploration. Combining these terms, we obtain the following objective, which is (approximately) maximized at each iteration [5]:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right], \tag{7}$$

where S denotes an entropy bonus; L_t^{VF} is the value function (VF) squared-error loss $(V_\theta(s_t) - V_t^{\text{targ}})^2$, with $V_t^{\text{targ}} = r_t + \gamma V_\theta(s_{t+1})$; and $c1$, $c2$ are coefficients for the VF loss and entropy term, respectively.

D. Gaussian Distribution

The Gaussian distribution is defined by the following probability density function:

$$f(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{x-\mu}{\sigma} \right)^2 \right) \tag{8}$$

whose parameters μ and σ are to be estimated by a deep neural network that models a so-called Gaussian policy, i.e., a parametrized policy $\pi_\theta(a|s) \sim \mathcal{N}(\mu,\, \sigma^2)$.

Therefore, when acting in stochastic mode, the agent samples the policy whereas in deterministic mode, $\pi(a|s) = \mu$. Since the Gaussian distribution has an infinite support, these sampled actions are clipped to the agent's bounded action space.

E. Beta Distribution

The Beta distribution has finite support and can be intuitively understood as the probability of success, where $\alpha - 1$ and $\beta - 1$ can be thought of as the counts of successes and failures from the prior knowledge, respectively. For a random variable $x \in [0, 1]$, the Beta probability density function is given by:

$$h(x : \alpha, \beta) = \frac{\Gamma(\alpha\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \tag{9}$$

where $\Gamma(.)$ is the Gamma function, which extends the factorial to real numbers. For $\alpha, \beta > 1$, the distribution is uni-modal, as illustrated in Fig. 1. When acting deterministically, the Beta policy outputs $\pi_\theta(a|s) = \alpha/(\alpha+\beta)$. The α, β parameters that define the shape of the function are obtained as outputs of a deep neural network representing the parametrized policy $\pi_\theta(a|s)$.

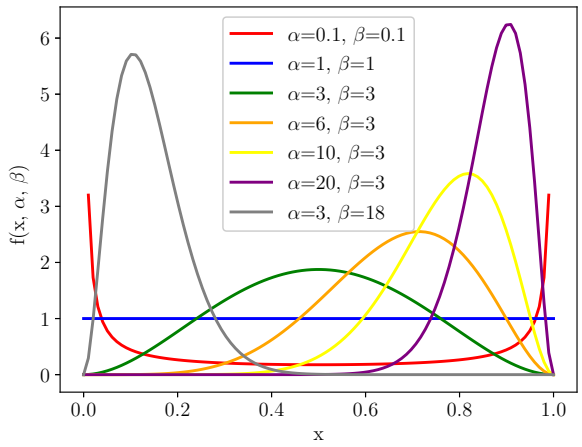

Fig. 1. Beta probability density function for different α, β pairs

F. Bias due to boundary effect

Modeling a bounded action space by a probability distribution with infinite support possibly introduces bias. As a result, the biased gradient imposes additional difficult in finding the optimal policy using reinforcement learning. The policy gradient estimator to optimize the parameters θ in (3), using Q as the target, can be obtained by differentiating (1), as follows:

$$\nabla_\theta L(\theta_t) = \int_{\mathbb{S}} \rho^\pi(s) \int_{\mathbb{A}} \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) Q^\pi(s|a) da ds \tag{10}$$

where $Q^{\pi_\theta}(s, a)$ is a state-action value function for a policy π_θ. Thus, the policy gradient estimator using Q as the target is given by:

$$g_q = \nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \tag{11}$$

This gradient is estimated by averaging n samples with a fixed policy π_θ, so that

$$\nabla_\theta L(\theta_t) = \frac{1}{n} \sum_{i=1}^{n} g_q \to \mathbb{E}[g_q] = \nabla_\theta L(\theta_t) \text{ as } n \to \infty \tag{12}$$

Let $A = [-h, h]$ be an uni-dimensional action space, with $a \in A$. In the case of an infinite support policy, an action $a' \notin A$ is eventually sampled, to which the environments responds as if the action is either h or $-h$. The biased policy gradient estimator would be given by $g'_q = \nabla_\theta \log \pi_\theta(a|s) Q^\pi(s|a')$ in this case. Besides, focusing on the inner integral of (10), the bias is computed as follows (also shown in [8]):

$$\mathbb{E}[g'_q] - \nabla_\theta L(\theta)$$

$$= \mathbb{E}_s \left[\int_{-\infty}^{\infty} \pi_\theta(a|s) \nabla_\theta \log \pi(a|s) Q^\pi(s, a') da \right] - \nabla_\theta J(\theta)$$

$$= \mathbb{E}_s \left[\int_{-\infty}^{-h} \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) [Q^\pi(s, -h) - Q^\pi(s, a)] da + \int_{h}^{\infty} \pi_\theta(s|a) \nabla_\theta \log \pi_\theta(a|s) [Q^\pi(s, h) - Q^\pi(s, a)] da \right] \tag{13}$$

These last two integrals evaluate to zero if the policy's distribution support is within the action space A.

III. Experiments

This section presents results for the proximal policy gradient method (PPO) on two continuous control problems from OpenAi gym [12]: the LunarLanderContinuous-v2 with low-dimensional state space; and the CarRacing-v0 with high-dimensional image input (Table I).

For all architectures, the last two layers output two 2-dimensional real vectors. For the Gaussian distribution, each dimension of the policy outputs its mean μ and its standard deviation σ, whereas for the Beta distribution the network outputs its parameters α and β >1. Here, 1 is added to a softplus layer $log(1 + exp(x))$ to ensure both α and β are larger than 1. Our implementation for PPO was based on a popular reinforcement learning library found in [13].

TABLE I
Environments

Environment	$\|\mathcal{S}\|$	$\|\mathcal{A}\|$
LunarLander	8	2
CarRacing	96x96x3	3

For each environment, we trained five models using different seeds for both the Gaussian and Beta distributions for a fixed number of total time steps. After completing the training, each model was evaluated in 100 consecutive episodes in both stochastic mode (sampling from the distribution) and deterministic mode (using the average of each distribution as the optimal action).

The hyperparameters for PPO can be seen in Table II for both control problems. Notice that the PPO configuration for the Lunar Lander was adapted from the one used for the MuJoCo environment in [5], whereas for the CarRacing, the parameters found in Atari [1] were used as a starting point.

TABLE II
HYPERPARAMETERS FOR TRAINING

	Lunar Lander	CarRacing
Horizon (T)	2048	500
Parallel environments (N)	1	8
Adam step size (lr)	$3 \times 10^{-4} \times \alpha$	$2.5 \times 10^{-4} \times \alpha$
Number of PPO epochs (K)	10	10
Mini-batch size (m)	32	64
Discount (γ)	0.99	0.99
GAE parameter (λ)	0.95	0.95
Clipping parameter (ϵ)	0.2	0.1
Value Function coefficient (c_1)	0.5	0.5
Entropy coefficient (c_2)	0	0.01
Total timesteps	10^6	5×10^6

α is linearly annealed from 1 to 0 over the course of learning

A. *LunarLanderContinuous-v2*

The LunarLanderContinuous-v2 environment simulates the landing of a space module on the moon. The overall objective corresponds to landing the module on the lunar surface delimited by two flags, approaching zero speed at the final step (Fig. 2). It has an unbounded, 8-dimensional observation space and a 2-dimensional action space. The actions are the main engine throttle and the secondary engine throttle, both bounded in the interval $[0, 1]$. The agent loses points for firing up the engines and for crashing (landing at high speed). The simulation is considered solved if the agent manages to score at least 200 points [14].

The agent follows an actor-critic framework, where the actor $\pi_\theta(a|s)$ consists of a neural network made of 3 fully-connected layers of 64 units each, with $\tanh$ activation functions. The output layer has 2 linear neurons to model either the Gaussian or the Beta distribution over the actions. The critic $V_{\theta_v}(s)$ does not share layers with the actor, but has an equivalent architecture of 3 hidden layers with only one output neuron which represents the value function.

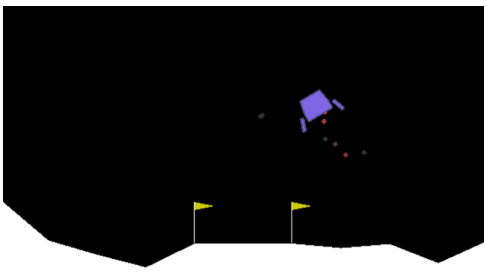

Fig. 2. LunarLanderContinuous-v2 Environment

B. *CarRacing-v0*

The CarRacing-v0 environment [15] simulates an autonomous driving environment in 2D. For each episode, a random track with 12 curves is generated. Each track is comprised of N tiles, with N ranging from 250 to 350. The agent receives 1000/N points for visiting each tile and loses 0.1 point for each frame. The episode ends in one of three situations:

1) Agent visited all tiles
2) Agent does not visit all tiles in 1000 frames
3) Agent gets too far way from the track and falls in the abiss (-100 points added)

Therefore, if the agent visits all tiles in 732 frames, the reward is 1000 - 0.1*732 = 926.8 points. Should the agent miss one or more tiles in its first lap attempt, the episode keeps on until the agent visit missing frame or the time limit is reached. The task is considered to be solved if the agent is able to get an average reward of at least 900 points in 100 consecutive trials (episodes).

The observation space consists of top down images (Fig. 3) of 96x96 pixels and three (RGB) color channels. The latest four image frames were stacked and given as input to the agent's network after rescaling and preprocessing them to gray scale (totalling 84x84x4 input dimensions). The action space has three dimensions: one encodes the steering angle and is bounded in the interval $[-1, +1]$. The other two dimensions encode throttle and brake, both bounded to $[0, 1]$.

For our implementation, throttle and brake have been merged on a single dimension so that on a given step, the agent does not simultaneously accelerates and brakes. We believe this is a more representative structure of real world systems: separated control inputs (throttle/brake) but single activation mechanism (right foot). In practice, one output neuron is responsible for both actions, making the output of the agent to be a two-dimensional vector. With this approach, we were able to make the agent learn effectively, mainly because it does not enter a deadlock state resulting from accelerating and braking at the same time. If we did not follow this approach,

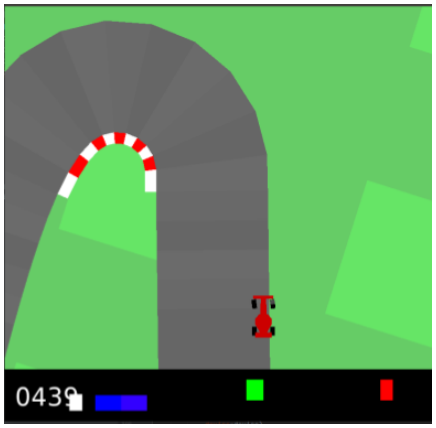

Fig. 3. CarRacing-v0 Environment

learning to control the vehicle would not take place. So far, we were not able to find other work in the literature that takes advantage on the aforementioned approach. Also, notice that we have not changed the original reward signal as some other works might have done.

The actor-critic network resembles that of [1] with respect to the shared encoder base comprised of the first 3 convolutional layers. Instead of connecting directly to the output layer as in [1], the shared base has an additional fully connected (FC) layer with 512 units. The critic $V_{\theta_v}(s)$ specializes further with its exclusive 1 FC layer of 512 units, that connects to a final output. The actor $\pi_\theta(a|s)$ has its own 2 FC layers with 512 units each on top of the shared base. The output layer is equivalent to the one from Section III-A, but its two neurons now refer to the steering angle or acceleration (brake/throttle).

IV. Results and Discussion

A. *LunarLanderContinuous-v2*

For the LunarLanderContinuous-v2 environment, we observe that using a Beta distribution allow for both a faster convergence and higher total reward during training. Five agents were trained with the same hyperparameters and different seeds.

After a million times steps, training is frozen and we evaluated each agent for 100 episodes in deterministic mode (using the mean of the policy's distribution as the action) and in stochastic mode (sampling the policy's distribution).

For the Gaussian distribution, we observe that the performance of the agents hovers around 225.7 and 219.0 points for the deterministic and stochastic policies, respectively. For the Beta distribution, we observe the agents perform at 267.0 (deterministic) and 273.6 points (stochastic). It is worth noting that Agents B4 and B5, which were trained with the Beta distribution, were able to score at least 200 points for all 100 episodes (Fig. 5) whereas the best agent trained with the Gaussian distribution (G3, deterministic policy) was able to score above the 200 points threshold for 92% of the 100 evaluation episodes. We can also observe that the variance of the Gaussian policy is higher than that of the Beta policy, even at the latest training iterations (Fig. 4) or after training ends (Fig. 5).

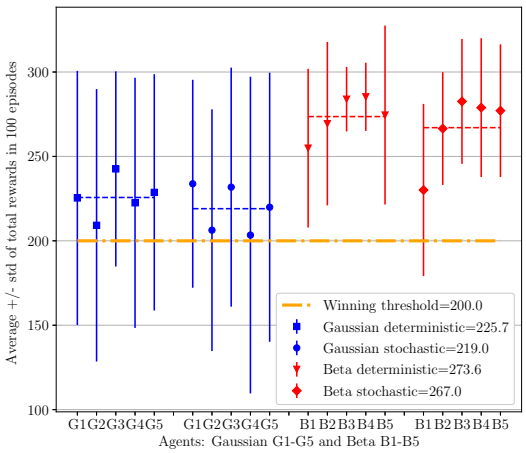

Fig. 5. Lunar Lander results: comparison of the Gaussian policy to the Beta policy in terms of the average rewards obtained by agents for 100 consecutive episodes after training. In blue (red), the average reward and standard deviation for each one of 5 agents using the Gaussian (Beta) policy. Both deterministic and stochastic policies were employed for evaluation. The winning threshold given by the horizontal black line represents the minimum threshold for successful completion of the task. Agents powered by the Beta distribution achieved superior performance and less variance.

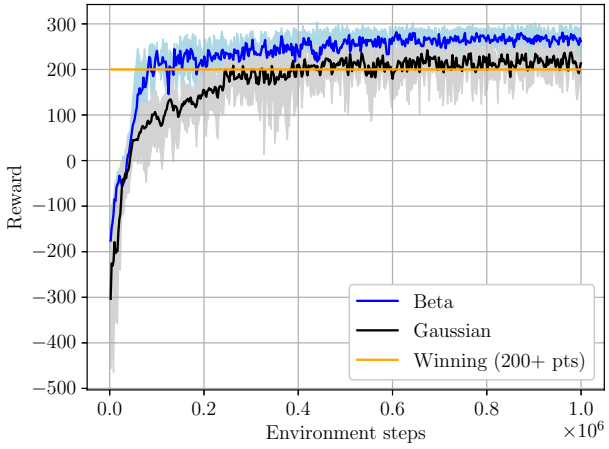

Fig. 4. Average rewards for five agents on the Lunar Lander task trained with Beta or Gaussian distribution for 1 million steps. The solid line represents the mean over a moving window of the previous 10 episodes for these five agents. The shaded area represents the interval between the minimum reward and maximum reward.

B. *CarRacing-v0*

For the CarRacing-v0 environment, the number of agent-environment interactions was fixed to 5 million steps during training. Afterwards, an evaluation of the agent's performance takes place, measured as the average reward in 100 consecutive episodes. The task is solved if this value is at least 900 points. We have observed that agents trained with Beta and Gaussian distributions have a similar convergence rate during training time. In Figure 6, we show the average reward over a moving window of 10 episodes, along the training process. Each policy optimization takes in 500 environment steps across 8 parallel environments.

Using the performance measure for 100 consecutive episodes training, in Fig. 7, we show that the stochastic policy presented better average performance than the deterministic policy for both distributions. For the Gaussian distribution, we observe that the five agents with the deterministic policy fail to follow the track, presenting an average score of 370.4 points that is much lower than the required 900 score points to solve the task. In stochastic mode, the policy presents an improved

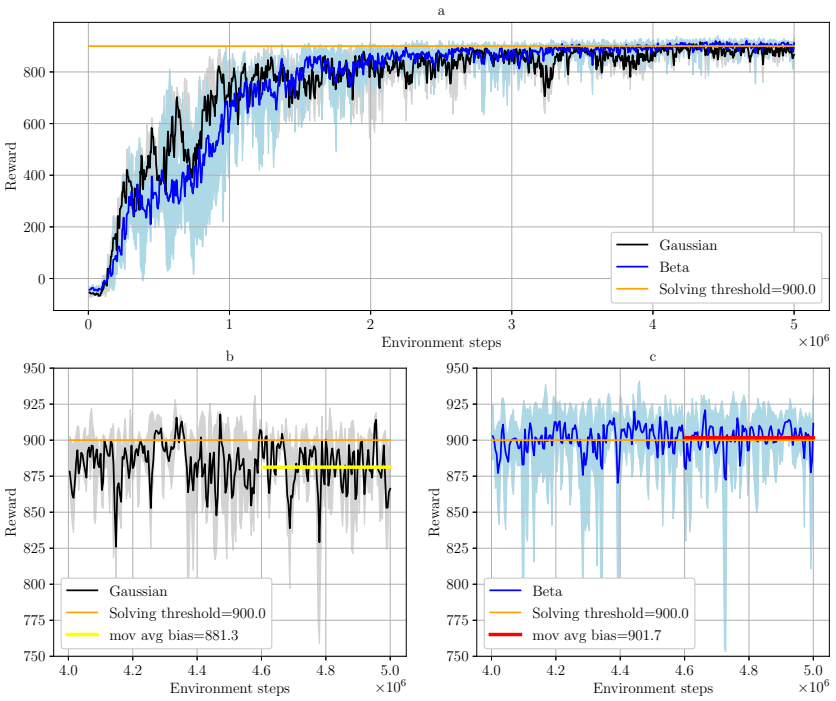

Fig. 6. Average rewards for 5 agents with different seeds for the CarRacing environment, plotted equivalently to Fig. 4. The final performance is shown on the bottom plots at a bigger scale, for agents with Gaussian policy (bottom left) and Beta policy (bottom right).

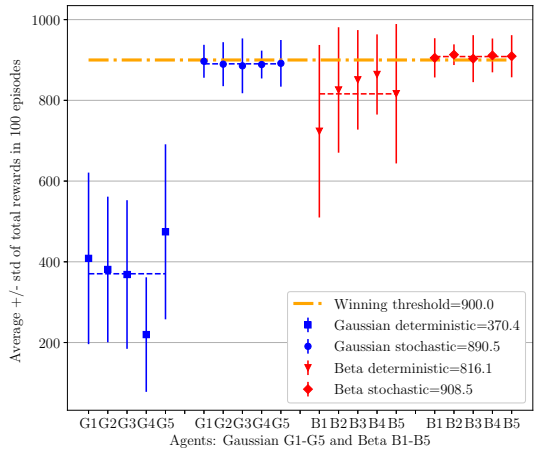

Fig. 7. Car Racing results after training: The evaluation followed the same procedure used for Lunar Lander (plots can be understood as in Fig. 5). For this task, the stochastic policy clearly yielded better performance than the deterministic one.

performance with an average score of 890.5 points, although in 38% of the 100 episodes the agents were not able to pass the winning threshold. For the Beta distribution, the agents' performance with the deterministic policy improves over the Gaussian policy by 320%, with average score of 816.1 points. These agents surpass the winning threshold in 26% of the evaluation episodes. In the stochastic mode, all agents were able to score above the winning threshold in at least 60% of the 100 of episodes played by each agent. All five agents with the Beta policy were able to successfully solve the game since each one of them reached a performance higher 900 points. This was not the case for the stochastic Gaussian policy, where each agent performed less than the threshold of 900 points. The best performing agent, B2, consistently reached scores above the other five agents, and it's the chosen agent to compare our approach with other works in the literature in the next section. Fig. 8 shows the resulting Gaussian and Beta policies at a specific timestep of the simulation, after training, when the car was about to turn left as it can be seen on the image fed to the policy network. The sampled distributions for both policies show that the Gaussian distribution, with its infinite support, falls outside the bounded action space, what is associated with the bias calculated in Section II-F. On the other hand, the Beta distribution fits well within the bounded action space, yielding an unbiased policy gradient estimator.

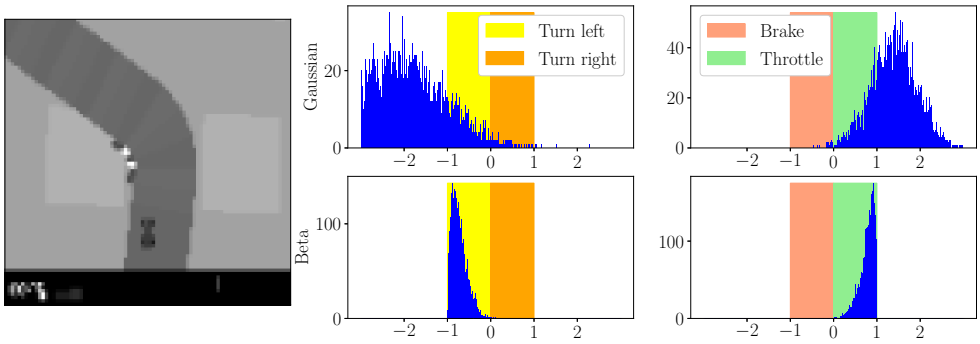

Fig. 8. Illustration of the Gaussian and Beta stochastic policy distributions in relation to the action space of the CarRacing environment. For a fixed observation s (preprocessed image in left plot), we sampled the Gaussian and Beta policies for 5000 actions. For the Gaussian distribution, a significant portion of the actions fall out of the valid direction and brake/throttle range (both $[-1, 1]$), whereas for the Beta distributions, all actions fall within boundaries.

C. Considerations on the CarRacing-v0 environment and other approaches

Simulation environments designed as test beds for reinforcement learning algorithms are primarily used in two ways:

1) To benchmark new algorithms or techniques without focusing particularly on a specific task;
2) To develop methods to solve a specific simulation task or benchmark, such as scoring more than 900 points on average in 100 consecutive runs for the CarRacing-v0 env, in an attempt to beat the other reported results.

Although our primary objective was the former, we emphasize that our work happens to fulfill to the latter as well. OpenAI CarRacing-v0 Leaderboard [16] hosts a series of self-reported scores. We compare our results only to those found in peer-reviewed articles (Table III), since they provide a basis for comparison and discussion.

Among the works that use Car Racing as a test bed, [17] claim to have been the first to solve the problem, using a recurrent world model. Other attempts included Deep Q-Networks with action-space discretization [18] and Genetic algorithms [19]. Other work that uses the Car Racing environment for benchmarking other algorithms are [20] and [21], and have been included for reference.

TABLE III
CARRACING-V0 LEADERBOARD

Method	Average Evaluation Score
PPO with Beta (Ours)	**913 +/- 26**
World models [17]	906 +/- 21
Adapted DQN [18]	905 +/- 24
Genetic Algorithms [19]	903 +/- 72
PPO with Gaussian (Ours)	897 +/- 41
Weight Agnostic NN [20]	893 +/- 74
PPO [21]	740 +/- 86
Random agent	-32 +/- 6

V. CONCLUSIONS

In this study, we observed that agents trained with PPO using a Beta distribution for the stochastic policy presented faster and more stable convergence of the training process (mainly for the Lunar Lander task), while their final performance was significantly superior to those trained with a Gaussian distribution. Thus, the Beta distribution is better able to satisfy the requirements of real-world applications with bounded action spaces, overcoming the estimation bias of the Gaussian policy.

Our results also show that continuous control with bounded action space for challenging car racing with random tracks and a high-dimensionality of the observation space (based on images) is much facilitated when the Beta distribution is employed. In fact, the agent's success in this task is considerably affected by this approach, achieving the best score so far on the CarRacing-v0 Leaderboard among the published work in literature. Finally, the results suggest that the Beta distribution should be a standard choice for those type of tasks.

Originally proposed in [8], the Beta distribution was tested in their work with TRPO/ACER on Atari games, which have high-dimensional observation space, but a discrete action space; and on robotic control tasks with a continuous action space and a low-dimensional observation space. In this work, we proposed to use the Beta distribution with PPO on high-dimensional image inputs and continuous action spaces.

We plan to extend these experiments to other types of reinforcement learning algorithms that are more sample efficient, in an attempt to verify if the Beta distribution transfers to other setups. Besides, experiments with more complex autonomous navigation in urban scenarios could benefit from the faster and more stable convergence as the training of end-to-end models is not a trivial task.

ACKNOWLEDGMENT

The authors would like to thank CAPES/Brazil for the financial sponsorship.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[4] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

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

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[7] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[8] P.-W. Chou, D. Maturana, and S. Scherer, "Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution," in *International conference on machine learning*. PMLR, 2017, pp. 834–843.

[9] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[10] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

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

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[13] I. Kostrikov, "Pytorch implementations of reinforcement learning algorithms," https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

[14] O. Klimov, https://github.com/openai/gym/blob/master/gym/envs/box2d/lunar_lander.py.

[15] ——, https://github.com/openai/gym/blob/master/gym/envs/box2d/car_racing.py.

[16] OpenAI, https://github.com/openai/gym/wiki/Leaderboard#CarRacing-v0.

[17] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," 2018.

[18] P. Rodrigues and S. Vieira, "Optimizing agent training with deep q-learning on a self-driving reinforcement learning environment," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 745–752.

[19] S. Risi and K. O. Stanley, "Deep neuroevolution of recurrent and discrete world models," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 456–462.

[20] A. Gaier and D. Ha, "Weight agnostic neural networks," *arXiv preprint arXiv:1906.04358*, 2019.

[21] R. Jena, C. Liu, and K. Sycara, "Augmenting gail with bc for sample efficient imitation learning," *arXiv preprint arXiv:2001.07798*, 2020.