ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

NATHAN SCHNEIDER GAVENSKI

SELF-SUPERVISED IMITATION LEARNING FROM OBSERVATION

Porto Alegre
2021

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

SELF-SUPERVISED IMITATION LEARNING FROM OBSERVATION

NATHAN SCHNEIDER GAVENSKI

Master Thesis submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. Rodrigo C. Barros

Porto Alegre
2021

Ficha Catalográfica

G282s Gavenski, Nathan Schneider

Self-supervised imitation learning from observation / Nathan Schneider Gavenski. – 2021.
73 p.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Rodrigo Coelho Barros.

1. Imitation Learning. 2. Behavioral Cloning. 3. Self-supervised learning. I. Barros, Rodrigo Coelho. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Clarissa Jesinska Selbach CRB-10/2051

NATHAN SCHNEIDER GAVENSKI

SELF-SUPERVISED IMITATION LEARNING FROM OBSERVATION

This Master Thesis has been submitted in partial fulfillment of the requirements for the degree of Master in Computer Science, of the Computer Science Graduate Program, School of Technology of the Pontifical Catholic University of Rio Grande do Sul

Sanctioned on 26th March, 2021.

COMMITTEE MEMBERS:

I dedicate this work to Malgorzata Maniszewski Gavenski, who inspired me to come back to study and always be better.

“Concordia res parvae crescunt.”
(Sallust)

ACKNOWLEDGMENTS

I would like to thank the Pontifícia Universidade Católica do Rio Grande do Sul and its employees for allowing me to enter this program and aiding me through this amazing experience. I would also like to thank Motorola and CAPES for their financial support, without which I would not be able to complete the master's degree.

I would like to thank my advisor Rodrigo Coelho Barros, who helped me with all my ideas, even the crazy ones, and showed me the ropes for how to be a better researcher. I also like to thank Alessandra Dutra, Dilnei Venturini, and Guilherme Wiedenhoft for encouraging me to pursue a master's degree during my graduation's final semester. A special thank you to my friends and colleagues at MALTA lab for helping me with my research, late-night work deliveries, and all the other moments we shared. Another special thank you to my friends, who understood my absence or weeks late responses.

Most of all, I would like to thank my parents Aleksander Gavenski and Siana Schneider, that did not measure any efforts for helping me, inspiring me, and supporting me. A very special thank you to my girlfriend Amanda, who stood by my side and helped me alleviate these trying times. To my family, I would like to thank you for all the love and help during this journey. Without any of you, this work would not be possible.

SELF-SUPERVISED IMITATION LEARNING FROM OBSERVATION

RESUMO

Os seres humanos têm a capacidade de aprender através da observação. O equivalente computacional deste aprendizado se chama clonagem de comportamento, uma técnica de aprendizado por imitação na qual um agente estuda o comportamento de um especialista. Abordagens recentes trabalham no uso de dados não rotulados com representações fidedignas dos estados, decodificando as informações observadas em ações de maneira auto-supervisionada. No entanto, ainda existem vários problemas a serem resolvidos, incluindo problemas de mínimos locais e dependência de vetores de estados. Nesta dissertação, apresentamos três novos métodos de aprendizado por imitação: *Augmented Behavioral Cloning from Observation*, *Imitating Unknown Policies via Exploration*, e *Combined Reinforcement and Imitation Learning*, que têm por objetivo resolver os problemas de decaimento de aprendizado durante o processo iterativo, de falta de políticas não-exploratórias, e de fraca eficiência de amostragem durante o treinamento dos agentes. Os resultados de Augmented Behavioral Cloning from Observations mostram que um mecanismo de amostragem pode criar ciclos de aprendizagem iterativos mais apropriados. Já os experimentos com Imitating Unknown Policies via Exploration ressaltam que um mecanismo de exploração pode alcançar resultados superiores do especialista e bater o estado da arte. Por fim, a análise do *framework* de Combined Reinforcement and Imitation Learning, mostra que adicionar um mecanismo de aprendizagem por reforço pode criar políticas mais eficientes e chegar a resultados semelhantes ao segundo método, mas com muito menos amostras. O segundo e o terceiro métodos oferecem diferentes *trade-offs* entre desempenho e eficiência, dependendo da dificuldade de aquisição de amostras especializadas.

Palavras-Chave: Aprendizado por Imitação, Clonagem de Comportamento, Aprendizado Auto-supervisionado.

SELF-SUPERVISED IMITATION LEARNING FROM OBSERVATION

ABSTRACT

Humans have the ability to learn through observation. The computational equivalent of learning by observation is behavioral cloning, an imitation learning technique that teaches an agent how to behave through expert demonstrations. Recent approaches work towards making use of unlabeled data with fully-observable snapshots of the states, decoding the observed information into actions in a self-supervised fashion. However, there are several problems still left to be addressed, including the many times the iterative learning scheme gets stuck into bad local minima. In this work, we propose three different methods, Augmented Behavioral Cloning from Observation, Imitating Unknown Policies via Exploration, and Combined Reinforcement and Imitation Learning, which aim to solve the problems of the decaying learning process, nonexplorative policies, and sample efficiency during the iterative process. The results from Augmented Behavioral Cloning from Observations show that a sampling mechanism can create more appropriate iterative learning cycles, while Imitating Unknown Policies via Exploration results convey that an exploration strategy can achieve results even better than the expert, reaching the state-of-the-art of the task. Lastly, the Combined Reinforcement and Imitation Learning framework shows that adding a reinforcement learning method within the imitation learning framework can create more efficient policies and reach similar results to the second method with fewer samples. Both the second and the third methods offer distinct trade-offs between performance and efficiency, depending on the difficulty of acquiring expert samples.

Keywords: Imitation Learning, Behavioral Cloning, Self-supervised learning.

LIST OF FIGURES

2.1 Example architecture of a feed-forward neural network [4]. 22

4.1 Example of frames for the four environments. 36

5.1 IDM predictions of the expert examples through time. 43

5.2 L_2 distance for the average of each action per iteration normalized by the expert and random samples in the 5×5 mazes. 43

5.3 Heatmap visualization of the gradient filters activating for the maze environment. The first row shows the input image, while the second row shows the gradient activation in the first self-attention module. 44

5.4 Expert demonstrations executing a 5x5 configuration of Gym-Maze. Below the state-image we represent the number of experts that visited that state. The blue line represents the path chosen by ABCO. 45

5.5 Visualization of the mean reward distribution over 10 runs during the training of each learning algorithm. 48

5.6 Visualization of the average reward through time over 10 runs for all environments. The dotted lines are the thresholds defined in Section 4.2. 50

6.1 Percentage of choices in which the self-decaying exploration rate does not select the MAP estimation. At the beginning of training, when the certainty of the PM is lower, IUPE benefits from greater exploration, which naturally decays as training proceeds. 56

7.1 Combined Reinforcement and Imitation Learning framework: (i) set $\mathcal{I}^{pre}$, created by using π_ϕ, as $\mathcal{I}^s$; (ii) use $\mathcal{I}^s$ to learn the inverse dynamics of the environment; (iii) label the approximated expert action, $\hat{A}$, responsible for the state transitions in the expert samples, $\mathcal{T}^e$, with IDM; (iv) use $\mathcal{T}^e$ and $\hat{A}$ to train the policy, π_ϕ in an IL manner; (v) use π_ϕ in the environment to create new state transition samples and uses the RL approach to further learn from its experience; and (vi) use the IUPE sampling mechanism, from Section 6.1, to create a new dataset for IDM. CRIL is an iterative process that cycles from step (ii) to step (vi) until no further improvement is noticed. 60

LIST OF TABLES

5.1 Ablation study considering the 2 main components of ABCO: *attention* and *sampling*. Data from the 5×5 maze environment. 44

5.2 *Performance* ($\mathbb{P}$) and *Average Episode Reward* (AER) for a supervised model (BC), BCO, ILPO, and our approach ABCO using OpenAI Gym environments. 46

5.3 Timesteps results for ABCO, DQN and TRPO algorithms in the CartPole environment using the threshold of the densest point. All results in the table are on $e5$ scale. 47

5.4 Timesteps results for ABCO, DQN and TRPO algorithms in the Acrobot environment using the threshold of the densest point. All results in the table are on $e5$ scale. 49

5.5 Timesteps results for ABCO, DQN and TRPO algorithms in the Mountaincar environment using the threshold of the densest point. All results in the table are on $e5$ scale. 49

5.6 Amount of timesteps needed to reach the goal in Cartpole, Acrobot and MountainCar environments. All results in the table are on $e5$ scale. 51

6.1 Ablation study considering IUPE's 3 main components: *attention*, *sampling*, and *exploration* in the 10×10 maze environment. 57

6.2 *Performance* and *Average Episodic Reward* for our approaches and related work using the OpenAI Gym environments. 58

7.1 *Performance* and *Average Episodic Reward* for CRIL using $\gamma = 0.99$, start $\epsilon = 1.0$, final $\epsilon = 0.01$, ϵ decay $= 500$, and replay buffer size $= 5,000$. 62

7.2 *Performance* and *Average Episodic Reward* for CRIL and IUPE using $10,000$ samples and one expert episode for all three environments. 63

7.3 *Performance* and *Average Episodic Reward* for DQN, IUPE, and CRIL for all three environments using only one expert episode. 64

LIST OF ALGORITHMS

3.1 Behavioral Cloning Observation α [57]. 28
3.2 Imitating Latent Policies from Observation [10]. 29
3.3 Generative adversarial imitation learning [20]. 32
5.1 Augmented Behavioral Cloning from Observation. 39
6.1 Imitating Unknown Policies via Exploration. 55

LIST OF ABBREVIATIONS

ABCO. – Augmented Behavioral Cloning from Observation

AER. – Average Episodic Reward

ANN. – Artificial Neural Network

BCE. – Binary Cross Entropy

BCO. – Behavioral Cloning from Observation

CNN. – Convolutional Neural Network

CRIL. – Combined Reinforcement and Imitation Learning

ConvLSTM. – Convolutional Long Short-Term Memory

DQN. – Deep Q-Network

IfO. – Imitation from observation

IL. – Imitation Learning

ILPO. – Imitating Latent Policies from Observation

IUPE. – Imitating Unknown Policies via Exploration

IDM. – Inverse Dynamics Model

GAIL. – Generative Adversarial Imitation Learning

LfD. – Learning from Demonstration

LSTM. – Long Short-Term Memory

MAP. – Maximum a Posteriori

MDP. – Markov Decision Process

ML. – Machine Learning

MLP. – Multilayer Perceptron

P. – Performance

PM. – Policy Model

SA. – Self-Attention

SSL. – Semi-Supervised Learning

TRPO. – Trust Region Policy Optimization

LIST OF SYMBOLS

x – Input features 19

y – Labels 19

$\mathcal{X}$ – Set of input features 19

$\mathcal{Y}$ – Set of labels 19

$\mathcal{P}$ – Pseudo labels 20

ϕ – Learnable parameters 22

J – Neural network cost function 22

u – Unit of a neural network (neuron) 22

l – Layer of a neural network 22

b – Bias term of a neural network 23

φ – Learning rate parameter 23

π_ϕ – Learnable policy 24

$\mathcal{A}$ – Set of actions 24

a – A single action 24

$\mathcal{S}$ – Set of state representations 24

s – A single state representation 24

r – Reward received for porformin an action 25

t – The time step 25

$\mathcal{D}$ – Set of demonstrations 25

$\hat{a}$ – Predicted action 25

π_ξ – Random policy 27

p_θ – Probability distrbution 27

Γ – Sequences of states pairs without labels 27

$\mathcal{I}^{pre}$ – Random policy experiences tuples of state, next state and action 28

$\mathcal{S}_{demo}$ – Unlabeled expert data 28

α – Amount of iteration for a method 28

$\mathcal{I}^{pos}$ – π_ϕ experiences tuples of state, next state and action 28

$\mathcal{Z}$ – Set of latent actions 29

G – Generator network 29

E – Encoder network 29

Δ_t – Difference between states from an environment 29

π_ω – Latent Policy 30

π_ψ – Remapped policy 30
$\mathcal{DR}$ – Discriminator network 31
π_ε – Expert policy 31
$\mathbb{E}$ – Agent's experience from a given environment 31
E – Amount of environments runs 36
F – Amount of foot steps 36
$\mathbb{P}$ – Performance metric 36
$\mathcal{I}_{spl}$ – Sampled set of experiences 40
β – Softmax function 41
N – Number of feature location 41
C – Number of channels 41
n – Self-attention feature map 42
μ – Self-attention learnable variable 42

CONTENTS

1 INTRODUCTION . . . 17
1.1 HYPOTHESIS AND RESEARCH QUESTIONS . . . 18
1.2 DOCUMENT ORGANIZATION . . . 18

2 BACKGROUND . . . 19
2.1 MACHINE LEARNING . . . 19
2.1.1 SUPERVISED LEARNING . . . 19
2.1.2 UNSUPERVISED LEARNING . . . 19
2.1.3 SEMI-SUPERVISED LEARNING . . . 20
2.1.4 SELF-SUPERVISED LEARNING . . . 20
2.1.5 REINFORCEMENT LEARNING . . . 21
2.2 DEEP LEARNING . . . 21
2.2.1 CONVOLUTIONAL NEURAL NETWORKS . . . 23
2.3 IMITATION LEARNING . . . 24
2.3.1 BEHAVIORAL CLONING . . . 25

3 RELATED WORK . . . 27
3.1 BEHAVIORAL CLONING FROM OBSERVATION . . . 27
3.1.1 NON-ITERATIVE . . . 27
3.1.2 ITERATIVE . . . 28
3.2 IMITATING LATENT POLICIES FROM OBSERVATION . . . 29
3.2.1 LATENT FORWARD DYNAMICS . . . 29
3.2.2 ACTION REMAPPING . . . 30
3.3 GENERATIVE ADVERSARIAL IMITATION LEARNING . . . 31
3.4 DEEP Q-NETWORK . . . 32
3.5 TRUST REGION POLICY OPTIMIZATION . . . 33

4 METHODOLOGY . . . 34
4.1 ENVIRONMENTS . . . 35
4.2 EVALUATION CRITERIA . . . 36

5 AUGMENTED BEHAVIORAL CLONING FROM OBSERVATION . . 39
5.1 INVERSE DYNAMICS MODEL AND POLICY MODEL . . . 39

5.2 SAMPLING METHOD 40
5.3 SELF-ATTENTION MODULE 41
5.4 EXPERIMENTAL ANALYSIS 42
5.4.1 ABCO SAMPLING IMPACT 42
5.4.2 SELF-ATTENTION 44
5.4.3 QUANTITATIVE RESULTS 45
5.5 SAMPLE EFFICIENCY OF REINFORCEMENT AND IMITATION LEARNING ALGORITHMS 46
5.5.1 EXPERIMENTAL DESIGN 47
5.5.2 QUANTITATIVE RESULTS 47
5.5.3 PRE-TRAINED ABCO 51
5.6 DISCUSSION 52

6 IMITATING UNKNOWN POLICIES VIA EXPLORATION 54
6.1 SAMPLING METHOD 54
6.2 EXPLORATION 54
6.3 EXPERIMENTAL ANALYSIS 55
6.3.1 EXPLORATION IMPACT 56
6.3.2 QUANTITATIVE RESULTS 57
6.4 DISCUSSION 58

7 COMBINED REINFORCEMENT AND IMITATION LEARNING . . . 59
7.1 LEARNING FROM ITS OWN EXPERIENCES 59
7.2 EXPERIMENTAL ANALYSIS 61
7.3 RESULTS 62
7.3.1 STANDARD DQN VALUES 62
7.3.2 SAMPLE EFFICIENCY 62
7.3.3 QUANTITATIVE RESULTS 63
7.4 DISCUSSION 63

8 CONCLUSIONS 65
8.1 LIMITATIONS 66
8.2 FUTURE WORK 67
8.3 PUBLISHED WORK 68
8.3.1 ON GOING WORK 68

REFERENCES **69**

1. INTRODUCTION

Humans often learn new abilities by observing other humans performing certain activities: we can mirror the behavior of others by observing sequences of events even though we may not have direct access to the underlying actions and intentions that yielded them [45]. For example, we can learn different tasks, such as cooking or drawing, just by watching videos. Our capabilities go beyond merely imitating; we can transfer the knowledge from a demonstration to a given task, despite differences in the environment, body, or objects that constitute the demonstration.

Learning by imitation has been studied in the area of psychology and recently became a prominent field in the area of artificial intelligence [31, 22, 11]. Imitation learning [44], also referred to as *learning from demonstration* (*LfD*), consists of an artificial agent mimicking the behavior of an expert by learning from demonstrations [47, 1]. In LfD, the agent trains to acquire skills or behaviors by observing a teacher solving a problem, *i.e.*, the agent learns a mapping between observations and actions [22, 57]. While LfD is motivated by how humans learn from demonstration, most existing work assumes that the agent has access to the actions (*i.e.*, action labels) performed, which humans do not. This assumption differs from how humans learn since they efficiently decode the observed information into the underlying actions [45]. Using labeled actions to learn a policy often requires the data to be recorded explicitly for imitation learning, limiting the usage of available unlabeled data. For example, consider an expert playing an unknown video game. Although we may not know the action performed by the expert in terms of pushed control buttons, we are still capable of mapping what we have observed into a known video game and understand the executed action.

Recent approaches for overcoming those issues perform *imitation from observation* (IfO). In IfO, one learns policies by using only the sequence of state observations, *i.e.*, without the need to access the demonstrator's actions [30, 57, 2]. In this dissertation, we address the problem of IfO by learning to imitate the behavior of an expert through the use of its state information without any other prior information of the observed actions. We propose three different Self-supervised Imitation Learning from Observation frameworks. The first method augments the BCO [57] framework by employing a new sample method and a self-attention mechanism to reduce problems with bad local minima from the original method. Our second approach further improves the first method's performance by adding a exploration mechanism. Finally, we create a method that combines reinforcement and imitation learning for improving the sample efficiency of our previous approaches.

1.1 Hypothesis and Research Questions

In this work we hypothesize that: (i) a different sampling mechanism can create policies that do not deteriorate during an iterative learning process; (ii) an exploration mechanism can benefit a self-supervised imitation learning framework by creating more diverse samples; and (iii) combining reinforcement and imitation learning approaches can yield more sample-efficient models. To validate these hypotheses, we aim to answer the following research questions:

1. What are the advantages and disadvantages of a goal-aware sampling mechanism in Self-Supervised Imitation Learning?
2. How effective an exploration mechanism is for avoiding local minima?
3. Can we build more sample-efficient approaches by hybridizing imitation learning and reinforcement learning?

1.2 Document Organization

This document is organized as follows. Chapter 2 introduces the learning paradigms, such as self-supervised and imitation learning, and the standard behavioral cloning approach. Chapter 3 presents current methods for imitation and reinforcement learning. Chapter 4 introduces the methodology we follow throughout this work. In Chapters 5, 6, and 7 we discuss our proposed methods and present experimental analyses to evaluate performance. We end this dissertation with our final thoughts, limitations of the proposed methods and future work directions in Chapter 8.

2. BACKGROUND

2.1 Machine Learning

Machine Learning (ML) is the scientific study of how machines can learn based upon experience. Mitchell [33] defines a learning problem more broadly as follows:

Definition 2.1.1. A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with E.

To perform a given task T, a system learns to approximate a function that represents the unknown real data distribution from the dataset provided. A dataset comprises instances, each consisting of a set of attributes and possibly an associated label. The dataset usually is divided into three parts: training, validation, and test sets. The training set serves as the learnable data for fitting the function one needs to approximate. The validation set is used to measure P and allow for model selection and hyperparameter optimization. And finally, the test set serves as a proxy for unseen data, allowing the estimation of the true generalization error, and thus it is a set never used during the learning phase [55].

Traditionally, machine learning divides its learning algorithms into three categories: supervised, unsupervised, and reinforcement learning. However, with recent advances in the area, other paradigms have emerged, such as semi-supervised and self-supervised learning.

2.1.1 Supervised Learning

The supervised learning paradigm is based on models that learn from tuples that consist of features x (*e.g.* pixels, features, embeddings, attributes), and labels y (*e.g.* classes, targets) [15]. Roughly speaking, the learning algorithm learns to associate some input with some output, given a set of examples. During the learning phase, a model approximates a function $f(x) : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ is the set of inputs $\{x_1, x_2 \ldots, x_n\}$, and $\mathcal{Y}$ the set of possible outcomes in classification problems $\{y_1, y_2, \ldots, y_n\}$. The approximated function is then extrapolated to infer the output variable of unseen input samples.

2.1.2 Unsupervised Learning

Unsupervised learning algorithms make use of the data alone, without any kind of supervision, in order to learn useful properties of the structure of the dataset at hand [15]. In contrast to supervised learning, which usually makes use of human-labeled data, unsupervised allows for modeling of probability densities over inputs [19]. For deep learning applications, we usually want to learn the entire probability distribution that generated a dataset, whether explicitly or implicitly. Other cases of unsupervised learning algorithms perform different tasks, like clustering, which consists of dividing the dataset into previously-unknown classes (clusters) of similar examples.

2.1.3 Semi-Supervised Learning

The semi-supervised learning (SSL) paradigm is halfway between supervised and unsupervised learning [15]. In addition to unlabeled data, the algorithm is provided with some supervision information, but not necessarily for all examples. The most common case for SSL is when not all targets associated with $\mathcal{X}$ are known. For those cases, the data set $\mathcal{X} = (x_i)_{i \in [n]}$ can be divided into two parts: examples $\mathcal{X}_l := (x_1, \dots, x_l)$ for which labels $\mathcal{Y}_l := (y_1, \dots, y_l)$ are provided, and those $\mathcal{X}_u := (x_{l+1}, \dots, x_{l+u})$, whose labels are not known [40].

In SSL, there may be cases in which we have known constraints, such as "these examples belong (or do not belong) to the same label". In this case, semi-supervised learning is considered as unsupervised learning guided by supervision and provides must-link and cannot-link constraints for data clustering. A must-link constraint indicates that both examples in the pair should be placed in the same cluster, while a cannot-link restriction means that two examples should belong to different groups. Typically, the constraints are "soft", that is, a cluster that violates them is undesirable but not prohibited.

Most approaches envision SSL as supervised learning with additional information on the distribution of the examples, and thus more in line with the supervised learning original goal: to predict a target value for a given example x_i.

Engelen and Hoos [58] state that most SSL methods base themselves into three different assumptions: (i) the smoothness assumption, which states that if two examples x_1 and x_2 are close in the feature space, their label counterpart is probably the same; (ii) the low-density assumption, which states that the decision boundary of a classifier does not pass through high-density areas of the feature space; and (iii) the manifold assumption, which states that samples located on the same low-dimensional feature space have the same labels.

2.1.4 Self-supervised Learning

While semi-supervised learning is halfway between supervised and unsupervised learning, the self-supervised learning paradigm is much more similar to supervised learning. Compared to supervised learning methods, self-supervised learning trains with data x_i along with a pseudo label $\mathcal{P}_i$, where $\mathcal{P}_i$ is an automatically generated label for a pre-defined pretext task without involving any human annotation [23]. As long as the pseudo labels $\mathcal{P}$ are automatically generated without involving human annotations, then the methods belong to the self-supervised learning paradigm.

2.1.5 Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward [21]. RL differs from supervised approaches in not needing labeled input/output pairs, and not needing sub-optimal actions to be explicitly corrected. Instead, the focus of this approach is on finding a balance between exploration and exploitation [24]. The most common environment is typically stated in the form of a Markov Decision Process (MDP). Formulating the problem as an MDP assumes the agent directly observes the current environmental state. When we compare the agent's performance to that of an agent that acts optimally, the difference in performance gives rise to the notion of *regret*. Hence, if an agent is to act optimally, the agent must reason about the long-term consequences of its actions, albeit the immediate reward associated with this might be negative. Thus, reinforcement learning is well-suited to problems that include a long-term, short-term reward trade-off.

We can divide RL into two main approaches: model-free and model-based algorithms [54]. Model-free algorithms focus on estimating what the optimal policy is (the agent behavior) to choose actions. Model-based approaches are interested in learning the model of the environment, enabling the algorithm to plan a sequence of actions that will maximize the total reward. Another significant differentiation is off-policy and on-policy reinforcement learning [54]. An off-policy is independent of the agent's actions, and it learns the optimal policy regardless of the agent's motivation, *e.g.* a replay buffer in which all states are appended so eventually the reward from the goal backward into the estimate Q for other states near the goal. Furthermore, an on-policy method attempts to evaluate or improve the policy by its decision process, *e.g.* by performing a policy gradient update after each episode.

2.2 Deep Learning

Conventional machine learning techniques are limited in processing natural data in their raw form. Constructing a machine learning system requires careful engineering and domain expertise to design feature extractors capable of transform the raw data into consumable representations or feature vectors for the learning subsystem. Representation learning is a set of methods and techniques that allow processing raw data and to automatically discover the most suitable representation for the underlying learning task. Deep learning methods fall within the representation learning framework, with multiple levels of representation obtained by composing simple but non-linear modules that transform the raw data into a higher and more abstract level [27].

Deep learning methods currently hold the state-of-the-art in many unstructured data processing tasks such as image recognition [61], speech recognition [17], machine translation [9], and question answering [63]. Furthermore, deep learning methods are also the weapon of choice for generative applications such as image stylization [29], musical composition [42], text generation [18], and image generation [52].

Artificial Neural Networks (ANNs) are the main component of the deep learning area, being vaguely inspired by the biological neural network that constitutes animal brains [32]. An ANN consists of a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons, which receives, processes, and transmits it forward along the network. The connection's signal is a representation of the aggregated input processed by some non-linear function. Neurons and connections typically have a corresponding weight that increases or decreases the strength of the signal as learning proceeds.

The most traditional ANN architecture is the feed-forward neural network, known as the multilayer perceptron (MLP). The goal of the MLP is to approximate some function $f(x, \phi) : \mathcal{X} \to \mathcal{Y}$ — where ϕ are the randomly-initialized learnable parameters — given a cost (loss) function $J(\phi)$ [15]. In Figure 2.1, we present an example of an MLP consisting of four layers. The first layer is the input layer, which connects to the original raw data, and the final layer is the output layer, which provides the network desired output. All remaining layers are called hidden layers.

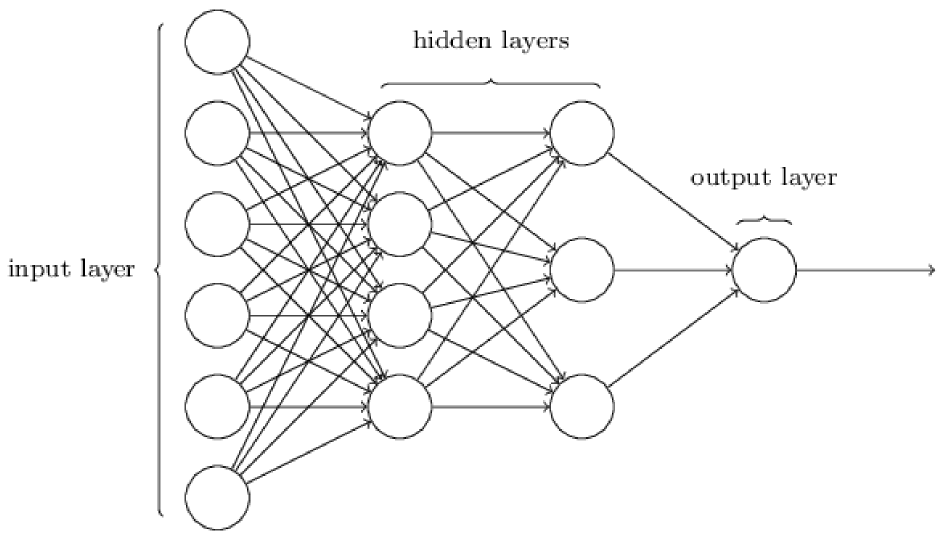


Figure 2.1: Example architecture of a feed-forward neural network [4].

Formally, the output of unit u in layer l is given by a functional transformation calculated via a linear combination [5]:

$$j_u^l = b_u^l + \sum_k w_{uk}^l x_k^{l-1}, \tag{2.1}$$

where b is the bias term for a given input x. Next, the unit's activation is given by applying a differentiable non-linear activation function on the output:

$$z_u = h(j_u), \tag{2.2}$$

where h is any differentiable non-linear activation function such as sigmoid, hyperbolic tangent ($\tanh$), or rectified linear units. MLPs need these non-linear functions due to the fact that multiple layers of linear units still produce only linear functions [33]. By combining the results obtained in a layer and using them sequentially, we are capable of approximating complex non-linear functions. The process of feeding inputs forward in an MLP is called forward propagation. Considering that MLPs are non-convex optimization problems due to the added non-linearities, the parameters ϕ cannot be analytically optimized, and a learning method based on an iterative process is necessary. Often, the iterative optimization process is based on computing the gradient of a loss function using backpropagation [15]. Backpropagation involves computing the gradients of $J(\phi)$ through the layers of the network by applying the chain rule of partial derivatives until the network parameters stop updating or reach a specific stopping criterion.

Given the objective of minimizing a loss function, the model needs to update its parameters ϕ in the opposite direction of the gradient of the objective function $\nabla_\phi J(\phi)$. Since the gradient information indicates the direction towards where it should update its parameters, an additional hyperparameter φ, called learning rate, is defined to control the magnitude of the resulting gradient value:

$$\phi = \phi - \varphi \cdot \nabla_\phi J(\phi). \tag{2.3}$$

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [28] are a specific kind of neural network for processing data that has a known, grid-like topology [15]. Although CNNs have become particularly famous for its image processing abilities [27], they are not specific for images. Time-series data, which are often represented as 1D grid of samples at regular time intervals, are also one of the possible uses for convolutional neural networks. Its name indicates that the network employs a mathematical operation called convolution, which incorporates linear transformations in the grid-view of the input. The convolutional weight matrix, also called Kernel, slides over all the input generating a grid-like output, and the non-linear activation function is then performed element-wise.

CNNs can learn in a supervised fashion how to (i) extract features from its input via local receptive fields of previous layers, forcing it to extract local features; (ii) map features that come from neurons that are constrained to share the same set of synaptic weights, thus reducing the number of free parameters when compared to fully-connected ANNs, and creating feature maps that are shift-invariant; and (iii) subsample the convolutional outputs via pooling layers, which reduces the output feature maps, reducing the sensitivity of the feature map output to shifts and other forms of distortion.

2.3 Imitation Learning

The imitation learning paradigm refers to an agent's acquisition of skills or behaviors by observing a teacher performing a given task [22]. The goal of imitation learning is to learn a policy π_ϕ that maps states to actions, $\pi_\phi : \mathcal{S} \rightarrow \mathcal{A}$, where $\mathcal{S}$ is the superset for states in a teacher trajectory $\{s_1, s_2, ... s_n\}, s_i \in \mathcal{S}$, and $\mathcal{A}$ is the set of actions for the given states $\{a_1, a_2, ... a_n\}, a_i \in \mathcal{A}$ [26]. However, learning a direct function between state and action is often not enough to achieve the desired behavior. The task performed by the learner may slightly vary from the demonstrated due to changes in the environment, obstacles, or targets. Therefore, imitation learning frequently involves another step that requires the learner to perform the learned action and re-optimize the learned policy according to its performance over the task. We formally define the problem of imitation learning following Schaal *et al.*'s definitions [48].

Definition 2.3.1. The process of imitation learning is one by which an agent uses instances of performed actions to learn a policy that solves a given task.

Definition 2.3.2. Imitation learning defines an agent as an entity that autonomously interacts within an environment towards achieving or optimizing a goal [39]. An agent can be a software robot; it receives information from the environment by sensing or communication and acts upon the environment using a set of actuators.

Definition 2.3.3. A policy is a function that maps states (a description of the agent, such as pose, positions, and velocities of various parts of the skeleton, and its relevant surrounding) to an action. It is what the agent uses to decide which action to execute when presented with a situation.

Although imitation learning often uses demonstration as instances of how to complete a given task, experiences such as the agent's sub-optimal trajectories can be used as well. The difference between the two types of training instances is that demonstrations provide the optimal action to a given state, allowing a supervised learning approach. At the same time, experiences show the performed action, which may not be optimal, but also provides the reward (or cost) of performing that action given the current state. More formally, we define demonstrations and experiences as follows:

Definition 2.3.4. A demonstration consists of a pair of input and output (x, y), where x is a vector of features describing the state at that particular instant and y is the action performed by the demonstrator.

Definition 2.3.5. An experience is presented as a tuple (s_t, a, r, s_{t+1}) where s_t is the state, a is the action taken at state s_t, r is the reward received for performing action a and s_{t+1} is the new state resulting from that action.

Imitation learning focuses on demonstrations due to the learner's lack of necessity of learning a cost function optimized by a teacher. It can minimize the difference between teacher and student in a supervised fashion. The learnable policy pi_ϕ from a set of demonstration is defined as $\pi(s_t, t, \phi)$, where s_t is the feature vector, t is the time step, and ϕ is the set of policy learnable parameters. Although t specifies an instance of input and output, it is also input to the policy π as a separate parameter. A policy that uses parameter t is non-stationary, while one that neglects it is stationary.

The advantage of stationary policies is the ability to learn tasks where the horizon is vast or unknown [46], while non-stationary policies are more naturally suited to learn motor trajectories. However, these policies are difficult to adapt to unseen scenarios and changes in the parameters of the task [48]. Given the temporal characteristics of these policies, at one point the trajectory can result in compounded errors as the agent continues to perform the remainder actions.

2.3.1 Behavioral Cloning

Behavioral Cloning is often referred to as the approach that directly maps from the state to the control input $\pi_\phi : \mathcal{S} \to \mathcal{A}$, as seen in Section 2.3. More formally, from a set of demonstrations $\mathcal{D} = \{(s_t, a_t), (s_{t+1}, a_{t+1}), \dots, (s_{t+n}, a_{t+n})\}$ an agent learns policy π_ϕ following $\hat{a}_t = \pi(s_t, t, \phi)$, where $\hat{a}$ is the predicted action, s is the state representation, t is the time step, and ϕ is the set of policy learnable parameters.

Hence, behavioral cloning is a reduction from imitation learning into a classification problem. It can be solvable by supervised and self-supervised approaches, where its output can be a single-label target or a multi-label target. A significant drawback of this reduction is the necessity of $(\mathcal{S}, \mathcal{A})$ pairs. In this work, we first focus on self-supervised learning techniques to reduce the necessity of labeled pairs by automatically creating its labels and removing the need for annotated datasets.

Traditionally, the behavioral cloning area divides itself into two categories: model-free and model-based methods. Model-free behavioral cloning methods learn a policy that reproduces the teacher's behavior without approximating the system dynamics nor recovering the reward function. Considering that model-free methods do not require learning the system's dynamics, it often does not need iterative learning and is more straightforward when compared to model-based methods. However, in trajectory learning, model-free methods do not ensure that the resulting trajectory is feasible in a given system, thus being hard to apply model-free techniques to underactuated systems in which the set of reachable states is limited [41]. Contrary to model-free, model-based methods learn a policy using information about the system dynamics. By learning forward dynamics, it is possible to plan a feasible trajectory close to the expert's behavior even if a system is underactuated [48]. Nevertheless, learning a forward model is a non-trivial and usually time-consuming problem.

3. RELATED WORK

In this chapter, we describe the related work in Imitation Learning. In Section 3.1, we first introduce Behavioral Cloning from Observation (BCO) [57], an iterative learning method based on the Learning from Observation paradigm. In Section 3.2, we explain the Imitating Latent Policies from Observation (ILPO) [10], an approach that infers latent policies directly from state observations. In Section 3.3, we describe the Generative Adversarial Imitation Learning (GAIL) [20], an adversarial learning strategy for Imitation Learning. In Section 3.4, we detail the Deep Q-Network [49], a Reinforcement Learning approach we use as a baseline. Finally, in Section 3.5 we present the Trust Region Policy Optimization [50], a model-free Reinforcement Learning method also used as baseline in our experiments.

3.1 Behavioral Cloning from Observation

BCO [57] combines an inverse dynamics model (IDM) with a model responsible for learning a stationary imitation policy, the Policy Model (PM). Torabi *et al.* propose two different forms of learning to imitate with BCO: (i) a non-iterative method that achieves significant results for environments with smaller state representations, albeit with limited learning capabilities (BCO(0)); (ii) an iterative method called BCO(α) that surpasses previous state-of-the-art learning algorithms.

3.1.1 Non-iterative

BCO allows the agent to learn its agent-specific IDM by looking at pairs of states from a random policy π_ξ previously sampled, denoted as $\mathcal{I}^{pre}$, and classifying the action responsible for the state transition ($\mathcal{A}_{\pi_\phi}$). The IDM uses the maximum-likelihood estimation (Equation 3.1) to find the best parameters:

$$\theta^* = \arg\max_\theta \prod_{t=0}^{|\mathcal{I}^{pre}|} p_\theta(a_t \mid s_t^{\pi_\phi}, s_{t+1}^{\pi_\phi}), \tag{3.1}$$

where p_θ is the probability distribution over actions given a pair of states representing a transition.

Next, it transforms the expert sequence of states into pairs (s_t and s_{t+1}, denoted as Γ_{demo}), so it can be free from domain and temporal specific information, and forward them to the IDM so that it can create pseudo-labels $\hat{a}$ for all expert pairs of states, denoted as Γ_{demo}) for the unlabeled expert data $\mathcal{S}_{demo}$. Finally, it uses Γ_{demo} and $\mathcal{S}_{demo}$ to learn the imitation policy in a behavioral cloning fashion by maximizing the maximum-likelihood estimation of an action (Equation 3.2) given a single state:

$$\phi^* = \arg\max_{\phi} \prod_{t=0}^{N} \pi_\phi(\hat{a}_t \mid s_t). \tag{3.2}$$

The non-iterative BCO heavily depends on the IDM capability of generalizing its classification power during inference. When the model is not able to create a manifold that is sparse enough to comprehend the transition of states, it can generate $\hat{a}$ labels that the policy will not learn or underfit into a smaller subset of actions, making the agent underperform. BCO(0) is also sensitive to the $\mathcal{I}^{pre}$ creation. When accompanied by the random samples dataset that is not close to the expert data, the IDM might never close the gap between what it perceives from $\mathcal{I}^{pre}$ to $\mathcal{S}_{demo}$. We believe that this weakness of the non-iterative method is why its performance decays as the dimensions of the state representation grows. The possible state representation branching factor for environments with higher dimensions makes it difficult for $\mathcal{I}^{pre}$ to be representative enough for IDM and for the policy to properly learn the expert trajectory.

3.1.2 Iterative

Torabi *et al.* [57] extend BCO by creating an iterative process called BCO(α), where α represents a hyperparameter to control the number of post-demonstration iterations. Instead of only depending on the $\mathcal{I}^{pre}$, BCO(α) executes its policy in the environment to acquire new state-action sequences as post-demonstrations ($\mathcal{I}^{pos}$). These post-demonstrations are then employed to update the IDM, and further on, the imitation policy itself. By using $\mathcal{I}^{pos}$, IDM is capable of receiving newly-observed state-action sequences, which can further boost BCO's performance. We present BCO(α) in the Algorithm 3.1.

Algorithm 3.1 Behavioral Cloning Observation α [57].

```
1: procedure BCO($\mathcal{I}^{pre}$)
2:     Initialize the model $\mathcal{M}_\theta$ as random approximator
3:     Set $\pi_\phi$ to be a random policy
4:     Set $I = |\mathcal{I}^{pre}|$
5:     while policy improvement do
6:         for time-step t=1 to $I$ do
7:             Generate samples $(s^a_t, s^a_{t+1})$ and $a_t$ using $\pi_\phi$
8:             Append samples $\Gamma^a_{\pi_\phi} \leftarrow (s^a_t, s^a_{t+1})$, $\mathcal{A}_{\pi_\phi} \leftarrow a_t$
9:         end for
10:        Improve $\mathcal{M}_\theta$ by modelLearning($\Gamma^a_{\pi_\phi}$, $\mathcal{A}_{\pi_\phi}$)
11:        Generate set of agent-specific state transitions $\Gamma^a_{demo}$ from the demonstrated state trajectories $D_{demo}$
12:        Use $\mathcal{M}_\theta$ with $\Gamma^a_{demo}$ to approximate $\tilde{\mathcal{A}}_{demo}$
13:        Improve $\pi_\phi$ by behavioralCloning($S_{demo}$, $\tilde{\mathcal{A}}_{demo}$)
14:        Set $I = \alpha|\mathcal{I}^{pre}|$
15:    end while
16: end procedure
```

We believe that Torabi *et al.* [57] added this iterative process so it could solve the aforementioned problems. However, by only employing $\mathcal{I}^{pos}$ to fine-tune the IDM, BCO(α) ends up suffering from another problem. For cases in which the policy still does not present good enough predictive performance, the generated set of post-demonstrations will contain misleading actions for specific pairs of states. Those erroneous actions tend to degrade the predictive performance of the IDM, which leads to the degradation of the policy predictions in a negative-feedback loop.

3.2 Imitating Latent Policies from Observation

ILPO [10] takes a different approach to learn how to mimic the action from an expert. It combines two different models to predict the next state of an expert, together with the most probable latent action for that transition. A latent action is defined as one or more actions that cause a transition of states. This definition is imperative for ILPO since not all actions are a latent action, *e.g.*, if an agent performs the action of moving left and bumps into a wall, this stationary transition may appear to be another type of action.

Edwards *et al.* [10] also consider that for all environments, there are a set of known actions $\mathcal{A}$ and a set of latent actions $\mathcal{Z}$, $\{z_1 ... z_{\mathcal{A}}\} \in \mathcal{Z}$. However, since not all stationary transitions may be directly linked to the original action, ILPO empirically assumes that $|\mathcal{Z}| \neq |\mathcal{A}|$. We present ILPO in Algorithm 3.2.

Algorithm 3.2 Imitating Latent Policies from Observation [10].

```
1: procedure ILPO($s_0^*, s_1^*, \ldots, s_N^*$)
2:     *Step 1: Learning latent policies*
3:     for $k \leftarrow 0 \ldots \#Epochs$ do
4:         for $i \leftarrow 0 \ldots N-1$ do                ▷ (Omitting batching for clarity)
5:             Train latent dynamics parameters $\theta \leftarrow \theta - \nabla_\theta \min_z \| G_\theta(E_{p\theta}(s_i^*), z) - s_{i+1}^* \|_2^2$
6:             Train latent policy parameters $\omega \leftarrow \omega - \nabla_\omega \| \sum_z \pi_\omega(z|s_i^*) G_\theta(E_{p\theta}(s_i^*), z) - s_{i+1}^*) \|_2^2$
7:         end for
8:     end for

9:     *Step 2: Action remapping*
10:    Observe state $s_0$
11:    for $t \leftarrow 0 \ldots \#Interactions$ do
12:        Choose latent action $z_t \leftarrow arg\,max_z \pi_\omega(z|E_{a\psi}(s_t))$
13:        Take $\epsilon$-greedy action $a_t \leftarrow arg\,max_a \pi_\psi(a|z_t, E_{a\psi}(s_t))$
14:        Observe state $s_{t+1}$
15:        Infer closest latent action $z_t = arg\,min_z \| E_{p\theta}(s_{t+1}) - E_{p\theta}(G_\theta(E_{p\theta}(s_t), z)) \|_2$
16:        Train action remapping parameters $\psi \leftarrow \psi + \nabla_\psi \log \frac{\pi_\psi(a_t|z_t, E_{a\psi}(s_t))}{\sum_a \pi_\psi(a|z_t, E_{a\psi}(s_t))}$
17:    end for
18: end procedure
```

3.2.1 Latent Forward Dynamics

ILPO's first model is responsible for learning the Latent Forward Dynamics. It is a generative model $G_\theta\left(E_p\left(s_t\right), z\right)$ that tries to create s_{t+1} given a prior s_t and a latent action z, where E_p is an embedding that is trained concurrently. The Latent Forward Dynamics model predicts the difference between states $\Delta_t = s_{t+1} - s_t$, rather than the absolute next state, and computes $s_{t+1} = s_t + \Delta_t$.

However, when learning to predict the forward dynamics, the generator might predict the mean overall transitions over the desired one. Considering ILPO does not have action information, as a way to condition its states generation the Latent Forward Dynamics model trains to make a prediction based on each of the latent actions $z \in Z$, $f(s_{t+1}|s_t, z)$, rather than $f(s_{t+1}|s_t, a)$. To train the generator, Edwards *et al.* [10] compute the loss as:

$$\mathcal{L}_{\min} = \min_z \left\| \Delta_t - G_\theta \left(E_p(s_t), z \right) \right\|^2 . \tag{3.3}$$

ILPO's policy learns by computing the expectation of the generated prediction under the probability that given a state s_t, a latent transition z will be observed in the expert data, *e.g.*, the expected next state, as:

$$\begin{aligned} \widehat{s}_{t+1} &= \mathbb{E}_{\pi\omega} \left[s_{t+1} | s_t \right] \\ &= \sum_z \pi_\omega (z|s_t)\, G_\theta \left(E_p(s_t), z \right) . \end{aligned} \tag{3.4}$$

The policy objective is to minimize the loss in Equation 3.5. For this problem, the predictions are fixed in order to make predictions that yield the most likely next state and it does not affect the generator output.

$$\mathcal{L}_{exp} = \left\| s_{t+1} - \widehat{s}_{t+1} \right\|^2 \tag{3.5}$$

The network is trained by summing both losses presented in Equation 3.3 and 3.5. ILPO avoids the need of creating an inverse dynamics model by introducing the latent forward dynamics component. However, by conditioning the generative model with a latent action z, it introduces the necessity of domain expertise. Without prior knowledge of $|\mathcal{Z}|$, ILPO needs to be executed multiple times with different vector sizes. Even though the latent forward dynamics is executed offline, it becomes costly for environments with higher $|\mathcal{A}|$.

3.2.2 Action Remapping

ILPO's second model is responsible for learning how to map the actions it previously discovered into the real action space. For such, ILPO collects tuples of $\{s_t, a_t, s_{t+1}\}$ with a random policy or with an iterative process of the remapped policy π_ψ.

While collecting experiences in the agent's environment, ILPO first identifies the latent action that corresponds to the environmental state transition, and then uses the environmental action taken as a label to train $\pi_\psi(a_t|z_t, E_a(s_t))$. For environments with low-dimensional states, the authors use the L_2 distance between the generator and the real next state, presented in Equation 3.6, while for those of higher dimensions, *e.g.* visual domains, it uses the L_2 distance between the encodings E_p, presented in Equation 3.7.

$$z_t = \arg\min \left\| s_{t+1} - G_\theta \left(E_p \left(s_t \right), z \right) \right\|_2 \tag{3.6}$$

$$z_t = \arg\min_z \left\| E_p \left(s_{t+1} \right) - E_p \left(G_\theta \left(E_p \left(s_t \right), z \right) \right) \right\|_2 \tag{3.7}$$

By obtaining the latent action z_t most closely corresponding to the environmental action a_t, the network learns by using the cross-entropy loss in a classification manner.

3.3 Generative Adversarial Imitation Learning

GAIL [20] takes an adversarial [16] approach to learn how to imitate the actions of an expert. It consists of two different models, a generator and a discriminator. Both models use the same architecture of neural networks, two hidden layers of 100 units each with $\tanh$ nonlinearities in between. The generator model is responsible for creating a policy, π_ϕ, that is as close to the unknown expert policy π_ε as possible. The discriminator $\mathcal{DR}$, on the other hand, needs to learn how to discriminate π_ϕ states from π_ε ones. In other words, GAIL uses an *Inverse Reinforcement Learning* approach, which tries to explain the expert behavior but does not directly tell the learner how to act.

The loss function for GAIL tries to minimize the distance of the experiences of π_ϕ from π_ε by trying to maximize $\mathcal{DR}$ capability of distinguishing both policies experiences $\mathbb{E}$. Furthermore, the authors force the generator to change since the discriminator will try to find a way to differentiate expert and π_ϕ regularly. The experiences from both π_ϕ and π_ε consist of an agent's trajectory without the actions since the expert's actions are unknown, and different environments might have the same trajectories with different actions. GAIL's loss function can be summarized in Equation 3.8:

$$\max_{\pi_\phi} \min_{\mathcal{DR}} -\mathbb{E}\pi_\phi[\log \mathcal{DR}(s)] - \mathbb{E}\pi_\varepsilon[\log(1 - \mathcal{DR}(s))]. \tag{3.8}$$

Both models training happens in an iterative supervised manner by collecting $\mathbb{E}_{\pi_\phi}$ and the original $\mathbb{E}_{\pi_\varepsilon}$. Next, when $\mathcal{D}_\mathcal{R}$ and the generator converge, the best policy is found by optimizing $-\log \mathcal{D}_\mathcal{R}(s)$ and uses Trust Region Policy Optimization (TRPO) [50] as a way to minimize the policy update due to noise in policy gradients.

GAIL is ultimately quite efficient in terms of expert data. However, it is not particularly efficient in terms of environment interactions during training, being comparable to the number of interactions needed for TRPO to train the expert policies from a reinforcement signal. Ho and Ermon [20] claim that GAIL results might be further improved by previously initializing a policy in a behavioral cloning approach. However, since GAIL's policy is model-free, for systems that do not maintain a consistent feature representation, the trajectory computed by the model might be distant from the teacher's path. In such approaches, a deviation in a trajectory often causes a degrading feedback loop that will not achieve an ideal performance [26]. We present GAIL's pseudocode in Algorithm 3.3.

Algorithm 3.3 Generative adversarial imitation learning [20].

```
1: procedure GAIL($\Gamma_\varepsilon \sim \pi_\varepsilon$, $\pi_\phi$, $\mathcal{DR}$)
2:     for $i = 0, 1, 2, \ldots$ do
3:         Sample trajectories $\Gamma_i \sim \pi_{\phi_i}$
4:         Update the discriminator parameters from $w_i$ to $w_{i+1}$ with the gradient
               $\hat{\mathbb{E}}_{\Gamma_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\Gamma_\varepsilon}[\nabla_w \log(1 - D_w(s,a))]$    (3.9)
5:         Take a policy step from $\phi_i$ to $\phi_{i+1}$, using the TRPO rule with cost function
           $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with
               $\hat{\mathbb{E}}_{\Gamma_i}\left[\nabla_\phi \log \pi_\phi(a|s) Q(s,a)\right] - \lambda \nabla_\phi H(\pi_\phi)$,
               where $Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\Gamma_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}]$    (3.10)
6:     end for
7: end procedure
```

3.4 Deep Q-Network

Deep Q-Network [49] (DQN) uses a deep convolutional network with hierarchical layers to approximate the optimal Q-function showed in Equation 3.11, where the value of state s, given an action a is defined by the expected sum of all rewards from the current state with the discount factor γ, following policy π.

$$\begin{aligned} Q^*(s,a) = &\max_\pi \mathbb{E}\left[\ r_t + \gamma r_{t+1} + \right. \\ &\left. \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi\ \right] \end{aligned} \tag{3.11}$$

DQN implements an experience replay mechanism that stores a set of observations from the agent to update the Q-function with random samples. This process solves the correlation issue between sequences of observations and smoothing changes in the data distribution. In order to compute the target values to update the Q-function, another network is used and updated iteratively, reducing the correlation between the action-value and target value. Equation 3.12 presents the loss function used by DQN to update the Q-Learning network, where θ_i are the parameters of the network, (s, a, r, s') is a mini-batch of random experience replay, γ is the discount factor of the agent's horizon, and θ_i^- are the target network parameters.

$$L = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q\left(s', a'\right) - Q(s, a)\right)^2\right] \tag{3.12}$$

For each episode, the algorithm has a probability of ϵ to select a random action or use the action-value function. The chosen action is sent to the emulator, which returns a reward and the next state. The method then stores the state-images returned by the environment using pre-processing methods, the reward in $\mathcal{D}$, and then sampling random mini-batch of transitions to update the Q-Learning network with gradient descent. Finally, the target network updates after a constant number of timesteps.

3.5 Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) [50] is a model-free method that optimizes a policy by gradient descent in restricted but safe step sizes. For every policy improvement interaction, the gradient computation is limited by a trust region that guarantee monotonic improvement, using the Minorize-Maximization algorithm to compute the optimal point inside the region.

Equation 3.13 shows how the policy is updated, where a_t and s_t are the action and state in time step i, π is the current policy and π_{old} is the old policy. Each improvement is applied if the Kullback-Leibler divergence (a measure of how different two distributions are) of the old policy and the new one is smaller or equal than δ, which represents a trust-region of step size.

$$\begin{aligned} \underset{\theta}{\text{maximize}}\ & \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] \\ \text{subject to}\ & \hat{\mathbb{E}}_t\left[KL[\pi_{\theta_{old}}(\cdot|s)||\pi_\theta(\cdot|s)]\right] \leq \delta \end{aligned} \tag{3.13}$$

The simple policy gradient methods have challenges with how it updates the policy gradient, *i.e.*, step sizes too large can generate a significant error, making the policy take wrong actions. Therefore, to use policy gradient methods to compare with IL but without losing the essential idea of the method, we choose TRPO since the only significant difference is in the policy update.

4. METHODOLOGY

Expert observations can be easier to come by than labeled experiences. We believe that by creating a method that minimizes the need for labeled data while maintaining an offline training phase is essential for using Imitation Learning models in realistic scenarios. To address such an issue, Torabi *et al.* [57] created BCO [57], a method capable of creating an inverse dynamics model, which is capable of understanding the transition from states and classifying the most likely responsible action. Nevertheless, learning dynamics models online can require a large amount of data, especially in high-dimensional problems. BCO also requires that all possible actions be mapped *a priori*, something that can fall short in more complex environments. Edwards *et al.* [10] created ILPO as a way to address the BCO problems: a method that does not need to know action labels to make an initial hypothesis of the policy. By learning a latent policy, ILPO creates a process that can be done offline and then uses a limited number of interactions with the environment to learn an action-remapping network that associates the true actions with the latent policy identified ones. However, ILPO falls short since the number of latent actions still needs to be defined *a priori*, and a second model still needs to be trained from scratch.

After identifying those gaps in the related work, we first propose the creation of an imitation learning method that introduces a new sampling mechanism into BCO's framework as a way to solve the vanishing action problem. In a second moment, we propose the inclusion of a stochastic mechanism into the BCO's framework as a way to avoid bad local minima created by the maximum-likelihood sample. We hypothesize that, even though the stochastic mechanism will initially deviate the agent from the expert trajectory, later it will help the method by performing a depth search into the possible states.

Finally, we face the problem of efficiency regarding the number of required expert samples. Sample Efficiency is an essential aspect in Reinforcement and Imitation Learning algorithms. It represents the amount of experience the agent needs to achieve a specific performance when deliberating in an environment trying to solve a specific task. In other words, it means how much knowledge the agent can learn from each interaction of the learning process. As the environment gets more complex in real-world applications, more an agent needs to learn. However, paired with an inefficient algorithm, one might find oneself training an algorithm for an extensive period or stuck in exploring states that drive the agent to local minima rewards.

Consider the example of training an agent to fly a plane. Piloting an aircraft is an incredibly complex task involving the airship control and other agents' knowledge to fly safely to a destination. Usually, an agent will first train in a simulator in order not to cause any possible accidents. However, when training an agent in a simulated environment, one should consider that the domain shift from simulation to reality might require additional training with a real aircraft. Moreover, suppose an inefficient learning algorithm is chosen for performing the task. In that case, the cost of teaching such an algorithm, e.g., GPU time and energy, might be higher than its benefits, hindering the algorithm's usage.

Another critical point might be the branching factor of possible states and the time required to generate sufficient samples for each state. When considering a task, one needs to evaluate how many possible states there are in an environment and how long it takes to get into a particular state. Returning to the same example as before, the end state will take each flight's traveling time. It is possible to engineer the environment to only create the desired states a fixed number of times, so the expert (or the agent) can perform the actions to recreate the landing. Nevertheless, as the branching factor grows, the more challenging it gets to engineer such an approach. Thus, it is vital for an algorithm to obtain the maximum amount of knowledge from each sample or require fewer samples.

For addressing the sample-efficiency issue in IL, we propose the creation of a new method that combines reinforcement and imitation learning. We hypothesize that this new method will reduce the number of samples needed for a policy to learn. By combining both learning methods, we believe that the new method might achieve similar results with fewer expert samples and create a more viable framework regarding the onerous task of recording experts for various domains.

4.1 Environments

For all experiments that will be performed in this dissertation, we make use of four environments from OpenAI Gym [6]: *CartPole-v1*, *MountainCar-v1*, *Acrobot-v1*, and *Gym-Maze*. Each environment is described below and illustrated in Figure 4.1.

i) CartPole-v1 is an environment where an agent pulls a car sideways intending to sustain a pole vertically upward as long as possible. The environment has a discrete action space composed of *left* or *right*, while the state space has 4 dimensions: *cart position*, *cart velocity*, *pole angle*, and *pole velocity at tip*. Barto *et al.* [3] define solving CartPole as getting an average reward of 195 over 100 consecutive trials. This environment is considered the easiest to solve with a vector-based approach. We choose CartPole-v1 since it is a base environment for all Imitation Learning approaches.

ii) MountainCar-v0 environment consists of a car in a one-dimensional track, positioned between two "mountains". The state-space has two dimensions, the respective car coordinates (x, y), and the action space consists of 3 possible strengths to move the car (-1, 0, or 1). To achieve the goal in this environment, the car has to acquire the required momentum and reach a flag placed on the second mountain top. Moore [38] defines solving MountainCar as getting an average reward of -110 over 100 consecutive trials. We consider this environment as the second most simple. Its difficulty lies in learning the "no action" movement and not overusing it since it will heavily impact the final reward. We choose this environment for its continuous-movement nature and also for being a baseline for all Imitation Learning approaches.

iii) Acrobot-v1 is an environment that includes two joints and two links, where the joint between the two links is actuated. Initially, the links are hanging downwards, and the goal is to swing the end of the lower link up to a given height. The state space contains 6 dimensions: $\{\cos\theta_1, \sin\theta_1, \cos\theta_2, \sin\theta_2, \theta_1, \theta_2\}$, and the action space consists of the 3 possible forces. Sutton [53] first described Acrobot in 1996 and later Geramifard *et al.* [14] improved it, which is the version we use. Acrobot-v1 is an unsolved environment, *i.e.*, it does not have a specified reward threshold at which it is considered solved. We consider Acrobot-v1 the most complex of all environments with a discrete action space and vector-based state. This environment is also a baseline for all Imitation Learning approaches.

iv) Gym-Maze is a 2D maze environment where an agent (the blue dot in Figure 4.1) must find the shortest path from the start (the blue square in the top left corner) to the goal (the red square in the bottom right corner) [7]. Each maze can have a different set of walls configuration, and three different sizes, 3×3, 5×5, and 10×10. An agent is allowed to walk towards any wall. The agent has a discrete action space composed of N, S, W, and E, and the state space consisting of rendered images of the maze. We consider Gym-Maze the simplest image-base state and a good baseline for Imitation Learning of multiple possible solutions with different paths. We choose Gym-Maze as our baseline when considering a state represented only by images.

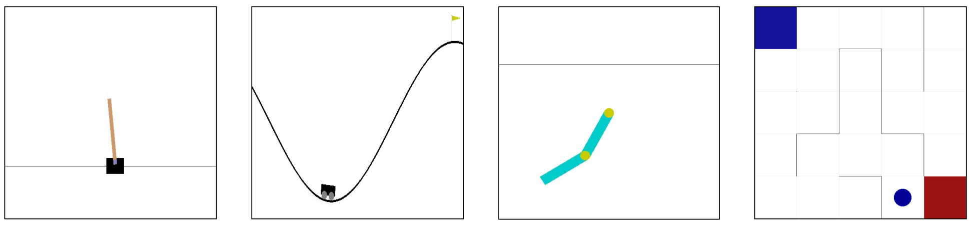

(i) CartPole-v1 (ii) MountainCar-v0 (iii) Acrobot-v1 (iv) Gym-Maze

Figure 4.1: Example of frames for the four environments.

4.2 Evaluation Criteria

To measure the quality of our results, we will use two different quantitative measures. The first one is the *Average Episodic Reward* (AER), a standard measure used to identify how well the generated policy performs in a specific environment. It consists of the average value of 100 runs for each episode in a given environment (*e.g.* running 100 different mazes in the Gym-Maze environment and calculating the average performance, or running 100 consecutive episodes for the CartPole problem and calculating the average reward). We present AER in Equation 4.1, where F_i is the total number of steps, and E is the total number of environments.

$$AER = \frac{\sum_{i=1}^{E} \sum_{j=1}^{F_i} \pi_{\phi}(e_{ij})}{E} \tag{4.1}$$

The AER value is an ideal measure to understand how well the expert did a task and consequently understand how difficult it is to imitate the expert behavior. However, by only looking at AER, we might be led to believe that our policy is doing an excellent job when, in reality, the environment might be too simple to solve. We will also use *Performance* ($\mathbb{P}$) as a second measure to understand how well our agent is performing when compared to an expert π_{ε} and a random policy π_{ξ}. $\mathbb{P}$ is the average reward for each episode scaled from 0 to 1, where zero is the random policy reward, and 1 is the expert's. We present $\mathbb{P}$ in Equation 4.2. A model can achieve scores lower than zero if it performs worst than π_{ξ} and higher than 1 if it performs better than π_{ε}.

$$\mathbb{P} = \frac{\sum_{i=1}^{E} \frac{\pi_{\phi}(e_i) - \pi_{\xi}(e_i)}{\pi_{\varepsilon}(e_i) - \pi_{\xi}(e_i)}}{E} \tag{4.2}$$

However, both metrics do not carry imitation meaning in their calculation. When analyzing both Equations 4.1 and 4.2, it becomes clear that we measure the agent's mimicking capability via the environment reward. By only measuring an agent's final reward and not its trajectory, both measures give importance to the objective instead of the path taken, depreciating model-free approaches.

Furthermore, we want to understand how an iterative method stands in terms of efficiency compared to other reinforcement learning methods. However, some environments do not have a specific goal to be considered solved, *i.e.* Acrobot, while others may have too-simple goals, *i.e.*, Cartpole's 195 mean reward. To address this issue, we define dynamic thresholds based on the algorithm's results for efficiency comparison. We believe that by creating a dynamic goal that each algorithm has to reach, it enables a fair comparison between each of the compared methods, considering that IL algorithms are limited by their expert's performance, and RL approaches can be inefficient in earlier iterations.

The following reasoning defines each threshold. We assume that each run follows an unknown function, since each run outcomes different rewards. However, each unknown function tends to reach similar values due to similar results of different agents trained by the same method. Thus, we cluster all runs for each approach on each environment and estimate each kernel's density following:

$$\rho_K(y) = \sum_{i=1}^{N} K\left(y - x_i; h\right), \tag{4.3}$$

where K represents a positive function controlled by a bandwidth parameter h. We fit each kernel in a Gaussian distribution considering the rewards for each environment to be continuous and that the reward values behave as a normal distribution.

Finally, we compare how many timesteps each algorithm needs to reach each threshold, giving us a sense of how efficient each algorithm is when compared to one another. We understand that setting a dynamic goal for each algorithm creates scenarios where none perform adequately, and considering that as the reward gets to the maximum value the difficulty grows, which makes the difference in efficiency more visible. However, we believe that by using the expert's reward to compare the overall results of each algorithm, we have a better sense of the performance of each algorithm.

5. AUGMENTED BEHAVIORAL CLONING FROM OBSERVATION

Inspired by BCO and in light of the gaps we discovered when trying to reproduce their work (see Section 3.1), we created an improved version of Torabi *et al.* [57] algorithm. We call our approach Augmented Behavioral Cloning from Observation (ABCO), where we further improve BCO's framework by adding: (i) an improved sampling method that regulates the observations that will feed the inverse dynamics model; and (ii) a self-attention mechanism module within both models. ABCO tackles the problem of $\mathcal{I}^{pos}$ not being a good representation of the state transitions. We believe our approach can help us understand how an iterative method can benefit by dynamically building its dataset of experience to fine-tune itself further.

ABCO is capable of outperforming all state-of-the-art approaches based on behavior cloning, either over low-dimensional state spaces or over raw images. The ABCO pseudocode is presented in Algorithm 5.1. ABCO was first published in the *International Joint Conference on Neural Networks* (IJCNN 2020) [37], and its results are presented in Section 5.4.

Algorithm 5.1 Augmented Behavioral Cloning from Observation.

```
 1: Initialize the model $\mathcal{M}_\theta$ as a random approximator
 2: Initialize the policy $\pi_\phi$ with random weights
 3: Generate $\mathcal{I}^{pre}$ using policy $\pi_\phi$
 4: Generate state transitions $\mathcal{T}^e$ from demonstrations $D$
 5: Set $\mathcal{I}^s = \mathcal{I}^{pre}$
 6: for $i \leftarrow 0$ to $\alpha$ do
 7:     Improve $\mathcal{M}_\theta$ by TRAINIDM($\mathcal{I}^s$)
 8:     Use $\mathcal{M}_\theta$ with $\mathcal{T}^e$ to predict actions $\hat{A}$
 9:     Improve $\pi_\phi$ by behavioralCloning($\mathcal{T}^e$, $\hat{A}$)
10:     for $e \leftarrow 1$ to $|E|$ do
11:         Use $\pi_\phi$ to solve environment $e$
12:         Append samples $\mathcal{I}^{pos} \leftarrow (s_t, \hat{a}_t, s_{t+1})$
13:         if $\pi_\phi$ at goal $g$ then
14:             Append $v_e \leftarrow 1$
15:         else
16:             Append $v_e \leftarrow 0$
17:         end if
18:     end for
19:     Set $\mathcal{I}^s$ = SAMPLING($\mathcal{I}^{pre}$, $\mathcal{I}^{pos}$, $P(g \mid E)$, $v_e$)
20: end for
```

5.1 Inverse Dynamics Model and Policy Model

Compared to the original BCO, we augment IDM and PM by adding a *self-attention* (SA) module [60, 62], further detailed in Section 5.3. For the inverse dynamics model, we believe it would compensate for the significant variation of samples from $\mathcal{I}^{pre}$ to $\mathcal{I}^{pos}$ in the iterative process. The self-attention module forces the IDM to identify what is essential to learn from each state. When using SA with images, it can identify which part of the image representation of the state is essential for predicting the correct action.

Unlike in the IDM, we use self-attention in the Policy Model to reduce the state changes during each iteration, since the self-attention module focuses on small details and differentiate better all classes given by the IDM. The SA module also allows the policy to look non-locally at states, helping the model to learn faster for higher dimensional states (*e.g.*, *Maze* and *Acrobot*) with a more gradual success rate in between iterations.

5.2 Sampling method

For every iteration, our sampling strategy creates new training data $\mathcal{I}^{s}$ containing a set of post-demonstrations $\mathcal{I}^{pos}_{spl}$ and a set of pre-demonstrations $\mathcal{I}^{pre}_{spl}$. In order to obtain the sample from post-demonstrations ($\mathcal{I}^{pos}_{spl}$), we first select the distribution of actions given a run E in an environment and the current policy $P(\mathcal{A} \mid E; \mathcal{I}^{pos})$. We consider only successful runs from $\mathcal{I}^{pos}$, *i.e.*, only state-action sequences in which the agent was able to achieve the environmental goal. Note that this goal might be a specific state (*i.e.*, such that the last transition in $\mathcal{I}^{pos}$ is in a set of specified states), or avoiding an undesirable state for a fixed number of transitions. We infer these goal states from the type of expert demonstration we receive. We represent it as v_e in Equation 5.1, where v_e is set to 1 if the agent achieves the environmental goal and zero otherwise, and E is the set of runs in an environment.

$$P(A|E; \mathcal{I}^{pos}) = \frac{\sum_{e=1}^{|E|} v_e \times P(A|e)}{|E|} \tag{5.1}$$

The intuition of using the post-demonstration only for successful runs is that if a policy is unable to achieve the environmental goal, then the post-demonstration alone does not close the gap between what the model previously learned with $\mathcal{I}^{pre}$ and what the expert performs in the environment. Using only successful runs also gives us a more accurate distribution of the expert since we are only using those distributions that achieved the goal instead of the random distribution that consisted of a balanced dataset. By not adding unsuccessful runs to the training dataset, we solve the problem in which BCO(α) degrades the performance in both models. With the distribution of actions from winning executions, we select the sample $\mathcal{I}^{pos}_{spl}$ from those runs, according to the win probability $P(g \mid E)$, *i.e.*, the probability of achieving a goal in an environment, as shown in Equation 5.2.

$$\mathcal{I}^{pos}_{spl} = (P(g \mid E) \times P(\mathcal{A} \mid E, \mathcal{I}^{pos})) \sim \mathcal{I}^{pos} \tag{5.2}$$

The sample from pre-demonstrations has an equivalent size of the post-demonstrations. Thus, to create a sample from pre-demonstrations $\mathcal{I}^{pre}_{spl}$, we use the loss probability with a distribution of actions in pre-demonstrations, denoted by $P(\mathcal{A} \mid \mathcal{I}^{pre})$, as demonstrated in Equation 5.3.

$$\mathcal{I}^{pre}_{spl} = ((1 - P(g \mid E)) \times P(\mathcal{A} \mid \mathcal{I}^{pre})) \sim \mathcal{I}^{pre} \tag{5.3}$$

Complementing the training dataset with random demonstrations offers two main advantages. First, it helps the model avoid overfitting from the policy demonstrations. Second, in the early iterations when the policy generates only a few successful runs, and the distribution might not be closer to the expert, the training data guarantees exploration by the IDM.

Using a win-loss probability, we induce the training data to be closer to the expert demonstration than to the random data, which boosts the model capability of imitating the expert. In this setting, the more an agent can achieve its goal, the less we want $\mathcal{I}^{s}$ consisting of $\mathcal{I}^{pre}$ and more of $\mathcal{I}^{pos}$. It is important to emphasize that our method is only goal-aware since we consider tuples from successful runs in the sample from post-demonstration, and does not use reward information for learning or optimizing. We do not use rewards because not all environments have a predefined function for it. On the other hand, most agents have a goal that is relatively easy to identify by inspecting the last transition visually, *e.g.*, the *mountaincar* reaching the flag pole, arriving at the final square at a *maze*, *acrobot* reaching the horizontal line, and *cartpole* surviving up to 195 steps, as described in Section 4.1.

5.3 Self-attention Module

Self-Attention [60] is a module that learns global dependencies within the internal representation of a neural network by computing non-local responses as a weighted sum of the features at all positions. It allows the network to focus on specific features that are relevant to the task at each step and learns to correlate global features [12].

ABCO uses the SA module based on the Self-Attention Generative Adversarial Network (SAGAN) [62] since models with it outperforms prior works. In SAGAN the self-attention module computes the key $f(x)$, the query $g(x)$, and the value $h(x)$ given a feature map x, with convolutional filters as $f(x) = W_f x$, $g(x) = W_g x$, and $h(x) = W_h x$. We compute the attention map by performing two different steps. First, we apply Equation 5.4 with the current key f and query g.

$$s_{ij} = f(x_i)^T g(x_j) \tag{5.4}$$

Second, we calculate the softmax function $\beta_{j,i}$ over the attention module to the i^{th} location when synthesizing the j^{th} region. With the attention map β and the values $h(x)$ we compute the self-attention feature maps $a = (a_1, a_2, ..., a_N) \in \mathbb{R}^{C \times N}$, where N is the number of feature locations and C is the number of channels, as illustrated in Equation 5.5.

$$a_j = \upsilon\left(\sum_{i=1}^{N} \beta_{j,i} h(x_i)\right), \upsilon(x_i) = W_\upsilon x_i \tag{5.5}$$

In this equation, W_f, W_g and $W_h \in \mathbb{R}^{\hat{C} \times C}$, and $W_v \in \mathbb{R}^{C \times \hat{C}}$, where $\hat{C}$ is C/k to reduce the number of feature maps. Furthermore, we have the self-attention feature map n, which we weigh by μ, a learnable variable initialized as zero.

The SA module in our method minimizes the impact of the constant changes created by the iterations by weighting all features. The model is capable of overlooking the potential local noise an agent might create and focus on features that are more relevant for the action prediction. It also provides smoother weight updates as a consequence of the weighting of all features. We believe that during early iterations, SA modules will learn with the random policy dataset how to weight each state, and this will later translate into more accurate labeling when $\mathcal{I}^s$ becomes more of $\mathcal{I}^{pos}$ than $\mathcal{I}^{pre}$.

5.4 Experimental Analysis

In this section, we discuss the experiments we have performed in order to evaluate ABCO. We perform ablation studies to verify the impact of both ABCO sampling strategy and SA modules. Finally, we compare ABCO with a supervised baseline (Behavioral Cloning, BC), with BCO [57], and with ILPO [10].

5.4.1 ABCO Sampling Impact

In this ablation study, we use only the sampling module to train ABCO(α), *i.e.*, we disable the self-attention modules. We also make use of the sampling method without the reduction of samples from $\mathcal{I}^{pos}$. We hypothesize that sampling from the original random policy dataset helps solving the vanishing actions, as well as closing the gap from the first iteration $\mathcal{I}$ and the expert. The vanishing of actions from the IDM predictions occurs due to the weak policy inference creating an $\mathcal{I}^{pos}$ that does not contain all actions, generating sparse representations that underfit the inverse dynamics model. During the early iterations under these conditions, IDM stops predicting the classes that are the minority in the expert dataset. This misclassification causes the policy to loop between actions, preventing the model from achieving its goal. We compare the distribution of all predictions from the IDM of BCO(α) and ABCO in Figure 5.1, where it shows that our sampling method can better predict all classes due to the artificial growth of our dataset caused by sampling from $\mathcal{I}^{pre}$.

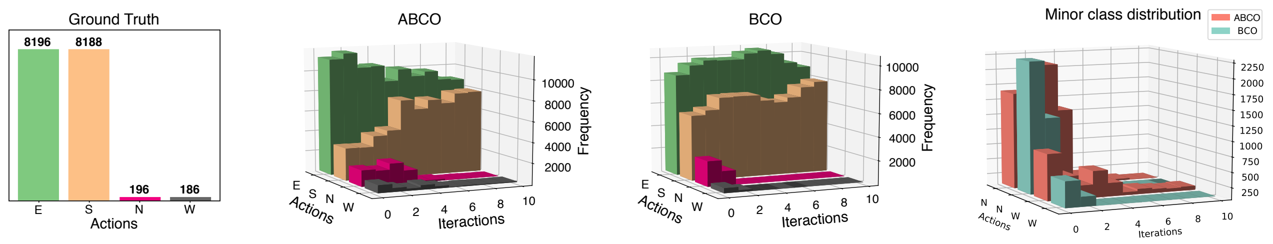


Figure 5.1: IDM predictions of the expert examples through time.

Furthermore, to observe whether the policy can create samples that are closer from the expert than from the random dataset, we calculate the L_2 distances from the average of all maze images from each action during each iteration and normalize them between zero (expert) and 1 ($\mathcal{I}^{pre}$ samples). The results in Figure 5.2 represent how our model learns a policy that creates better $\mathcal{I}$ for the majority classes (*i.e.*, S, and E), and also to the minority classes (*i.e.*, N and W). We assume this difference in approximation of the expert dataset to be due to the minority classes consisting mostly of $\mathcal{I}^{pre}$ since most mazes do not require those actions. By sampling from the random dataset, we force the IDM to balance its labeling and create iterations that are further distant. Still, as the Policy progresses and solves more runs, it approximates and becomes closer to the expert. Since they are closer to the expert, the new samples allow the IDM to fine-tune itself and predict expert labels more precisely.

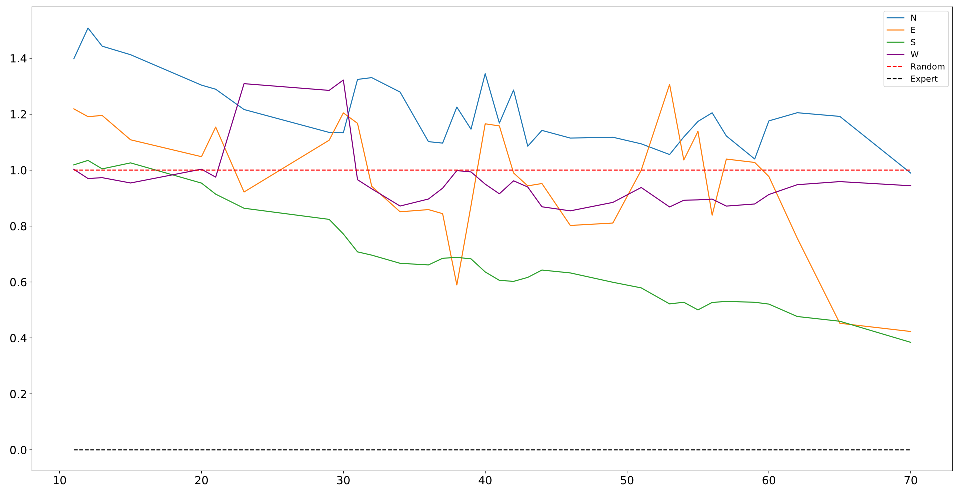


Figure 5.2: L_2 distance for the average of each action per iteration normalized by the expert and random samples in the 5×5 mazes.

We also believe that not all interactions following a sub-optimal policy are relevant to IDM's learning. If our hypothesis that a sub-optimal policy might create samples that harm the IDM ability to label the expert samples correctly holds, then the values for AER and P would be lower than those of the sampling method from Section 5.2. In our experiment, we use a Resnet without attention modules and we create $\mathcal{I}^s$ with all $\mathcal{I}^{pos}$ and the same ratio used in the original sampling method for all $\mathcal{I}^{pre}$. Using this approach, we observe that when employing all interactions from the policy to create the new dataset, the model achieves lower AER and P as expected, as shown in Table 5.1.

Table 5.1: Ablation study considering the 2 main components of ABCO: *attention* and *sampling*. Data from the 5×5 maze environment.

Model	$\mathbb{P}$	AER
BCO [57]	-0.112	-0.941
Attention	-0.415	-0.940
Partial Sampling	0.717	0.716
Whole Sampling	0.628	0.676
ABCO (Attention + Partial Sampling)	0.960	0.932
ABCO (Attention + Whole Sampling)	0.759	0.755

We conclude that the new sampling method alone can boost the learning experience by allowing the IDM to receive a more balanced dataset. Still, when accompanied by the self-attention modules, it further improves the generalization from the model by learning to weigh each sample accordingly and further boosting ABCO's performance.

5.4.2 Self-Attention

In this second ablation study, we trained ABCO using only the self-attention module. We observe in that case that the accuracy of our models was higher than the original method. This behavior can be explained in Figure 5.3, where we used the Grad-CAM [51] technique to visualize the self-attention gradient activations given an image in a trained policy. By observing the gradient activations, we infer that the self-attention modules help the model focusing at the agent, while still being able to pay attention to walls nearby. Furthermore, activation areas are wider when the agent is walking through open passages than between corridors. This behavior resembles human vision and is exemplified in the first and third frames, where the agent is between two walls, while in the other frames the agent has a broader view. However, high accuracy does not represent an excellent performance, since without the sampling method, some actions might not occur in further iterations. With high accuracy and samples that do

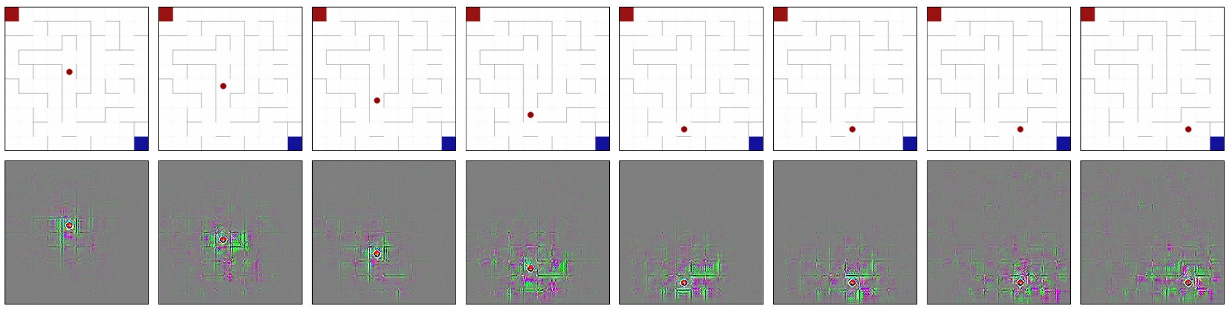


Figure 5.3: Heatmap visualization of the gradient filters activating for the maze environment. The first row shows the input image, while the second row shows the gradient activation in the first self-attention module.

not represent the action from the expert as they should, IDM stops predicting the minority action, creating a sub-set from all possible actions, and the policy learns the new subset of real actions. This behavior results in the policy not performing the less frequent actions that are needed to solve different environments during the inference phase (*e.g.*, N and W on mazes, or not performing actions during acrobot), as we discussed in Section 5.2.

We observe that even when weighing the features, IDM is still capable of predicting the most common path. When feeding the Policy with ten different solutions for each maze, the agent mimics the most common path, as shown in Figure 5.4. Nevertheless, when the first iterations still samples all classes, the model takes more transition samples to reach the results from Table 5.1. Although self-attention alone achieves results similar to BCO(α), when combined with the sampling method it provides a significant impact on the final results.

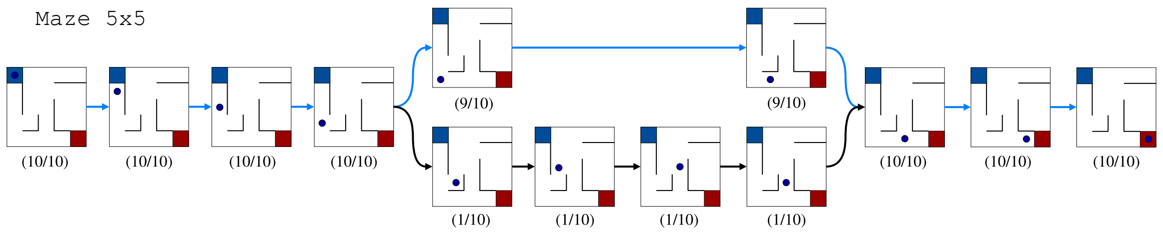


Figure 5.4: Expert demonstrations executing a 5x5 configuration of Gym-Maze. Below the state-image we represent the number of experts that visited that state. The blue line represents the path chosen by ABCO.

5.4.3 Quantitative Results

Table 5.2 shows that our method is on par or surpasses the state-of-the-art approaches in all the environments. The overall results confirm that the attention module and our sampling strategy can improve the imitation process. All approaches achieved the maximum score for CartPole in both AER and $\mathbb{P}$, showing that this problem is easy to learn. Although our model achieved the best $\mathbb{P}$ and AER scores in the Acrobot environment, the state-of-the-art algorithms presented similar results with $\mathbb{P} \approx 1.00$ and $AER = -85.300$. ABCO achieving better results for AER means that it can solve the problem using fewer frames. However, both models present similar imitation capabilities since all models achieved $\mathbb{P} \approx 1.00$. It is important to note that even though ABCO does not use labeled data, it was capable of achieving better results than the supervised approach (BC), which uses labels for actions. For MountainCar, we observe a large difference in terms of $\mathbb{P}$, with our model achieving $\mathbb{P} = 1.289$ and presenting a difference of ≈ 0.34 to BCO, which is the second-highest result.

Although in all the Maze environments we achieved the highest scores for $\mathbb{P}$, we can observe that as the complexity of the environment increases, our performance decreases. Nevertheless, we can see in the results that ABCO is less affected than BCO as the complexity increases, since $\mathbb{P}$ for our results in Mazes 3×3, 5×5 and 10×10 are 1.159, 0.960 and

Table 5.2: *Performance* ($\mathbb{P}$) and *Average Episode Reward* (*AER*) for a supervised model (BC), BCO, ILPO, and our approach ABCO using OpenAI Gym environments.

Model	Metric	CartPole	Acrobot	MountainCar	Maze 3×3	Maze 5×5	Maze 10×10
BC	$\mathbb{P}$	**1.000**	1.071	1.560	−1.207	−0.921	−0.470
	AER	**500.000**	−83.590	−117.720	0.180	−0.507	−1.000
BCO	$\mathbb{P}$	**1.000**	0.980	0.948	0.883	−0.112	−0.416
	AER	**500.000**	−117.600	−150.00	**0.927**	0.104	−0.941
ILPO	$\mathbb{P}$	**1.000**	1.067	0.626	−1.711	−0.398	0.257
	AER	**500.000**	−85.300	−167.00	−0.026	−0.059	−0.020
ABCO(α)	$\mathbb{P}$	**1.000**	**1.086**	**1.289**	**1.159**	**0.960**	**0.860**
	AER	**500.000**	**−77.900**	**−132.30**	0.908	**0.932**	**0.784**

0.860 respectively, while BCO obtained 0.883, −0.112 and −0.416. In terms of *AER*, ABCO was only outperformed by BCO in Maze 3×3 by ≈ 0.02, where Torabi *et al.* [57] achieved *AER* = 0.927. Comparing the results of ABCO with ILPO, we observe that ILPO increases $\mathbb{P}$ as the maze increases in size, but it is still much lower than ABCO for the 10×10 maze. We believe that this discrepancy happens for two reasons. First, ABCO contains an attention module that increases its capability to focus on essential features through non-visited state spaces. Second, ILPO does not consider a full image of the scenario since it uses crop mechanisms and internal manipulations with the state images. Using a partial observation from the environment means that the approach cannot receive essential features from the images (*e.g.* the initial state, the goal state, the agent, *etc*). On the other hand, as the maze increases, ILPO receives more local information through the crops, increasing its $\mathbb{P}$.

5.5 Sample Efficiency of Reinforcement and Imitation Learning Algorithms

In this section, we explore the sample efficiency from ABCO and other reinforcement learning baselines. The goal we want to achieve with these experiments is to understand how ABCO compares to other baseline methods (in particular, with RL) regarding the necessity of expert samples. When designing the ABCO experiments, we stumbled upon a critic hyperparameter: the correct size of the samples needed from an expert. If we use fewer samples, the policy model may never learn how to act within an environment, and such behavior can be directly correlated to the branching factor of the environment or the lack of robustness of the model. Consequently, if we need to create a larger number of samples, perhaps we will need to create an expert agent capable of generating such an amount of samples. That, however, makes IL unfeasible, since we would be teaching it to behave like another automatic approach (like an RL method).

5.5.1 Experimental Design

Reinforcement Learning has several algorithms with different techniques to solve the reward maximization problem. Running all algorithms presented in the literature for comparison would be time-consuming and redundant since several algorithms use equivalent mechanisms to solve the problem. Therefore, we use the algorithms described in Sections 3.4 and 3.5 to compare with ABCO in terms of sample efficiency. We believe that such algorithms represent a vast horizon of techniques used in Model-free Reinforcement Learning.

More speficially, we employ DQN [35] and TRPO [50] as baselines for the three discrete action-space environments. All algorithms are executed 10 times for $1,000$ epochs in each selected environment, training $5,000$ timesteps per epoch. For each epoch, we store the training timesteps, the mean of the obtained rewards, and the standard deviation from the 10 runs.

Recall that ABCO [37] learns from the expert and random policy behaviors from a set of observations. Therefore, to achieve a fair comparison between the learning paradigms, we divide the $5,000$ samples into two groups, resulting in $2,500$ timesteps from each to train the model.

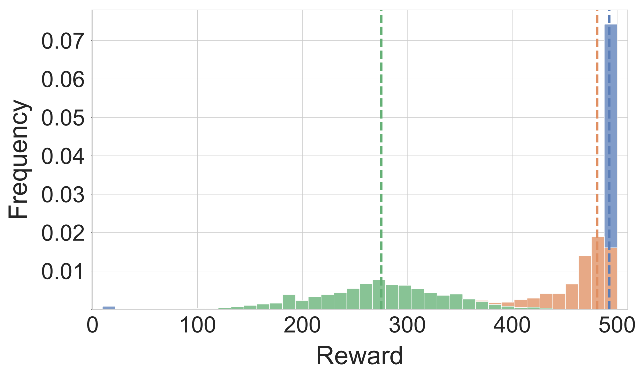
5.5.2 Quantitative Results

We first analyze how fast each algorithm can achieve a predetermined threshold, as explained in Section 5.5.1. All thresholds are shown in Figure 5.5. We further analyze each approach by comparing each algorithm's final reward in order to understand whether time was a real constraint.

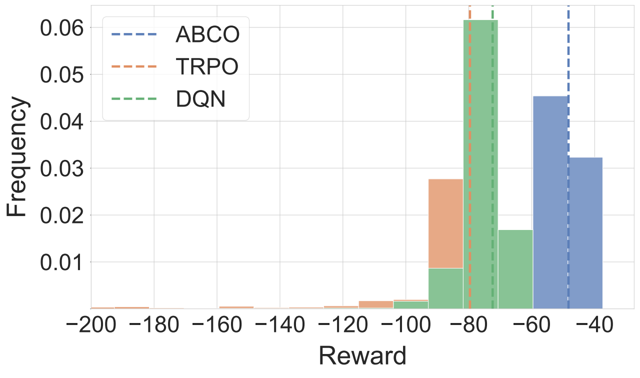
CartPole: This environment is considered the simplest domain among all used in this dissertation. The thresholds for this environment are 275, 481, and 492, defined by DQN, TRPO, and ABCO, respectively. When comparing each algorithm in Table 5.3, we can see that DQN, despite having the smallest threshold, achieves both ABCO's and TRPO's thresholds much faster on average. Those numbers indicate that DQN is an algorithm with a much higher exploration rate than TRPO and ABCO, since the latter does not have any exploration mechanism. How-

Table 5.3: Timesteps results for ABCO, DQN and TRPO algorithms in the CartPole environment using the threshold of the densest point. All results in the table are on $e5$ scale.

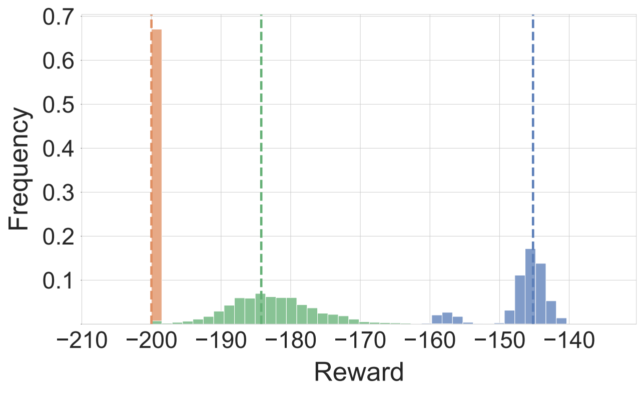
Threshold	**275 (DQN)**	**481 (TRPO)**	**492 (ABCO)**
ABCO	1.07 ± 0.28	1.66 ± 0.32	2.74 ± 2.69
DQN	0.52 ± 0.40	$\mathbf{1,02 \pm 0.31}$	$\mathbf{1.05 \pm 0.33}$
TRPO	$\mathbf{0.27 \pm 0.32}$	1.21 ± 1.68	1.39 ± 1.65



(a) CartPole



(b) Acrobot



(c) MountainCar

Figure 5.5: Visualization of the mean reward distribution over 10 runs during the training of each learning algorithm.

ever, when comparing with TRPO in its threshold, we see that DQN needs $52,000$ samples to reach a mean reward of 275, while TRPO only needs $27,500$. We conclude that DQN, despite being efficient in the number of samples required to reach above-average rewards, is inefficient in maintaining high results. TRPO results are not distant from DQN when comparing both in terms of sample efficiency. Nevertheless, its threshold shows that it maintains higher values than DQN since the densest point for the learning function was 481. Finally, ABCO when compared to both algorithms is the most inefficient learning method. We believe that those numbers are because of ABCO's double model structure. Considering that ABCO's IDM is highly responsible for giving the approximated labels to its policy, we expect to see a higher number of samples needed to achieve the same result than DQN and TRPO. However, ABCO's threshold shows that its results sustained better rewards than the other algorithms.

Acrobot: The thresholds for this environment are -79, -78, and -72 for TRPO, ABCO, and DQN, presented in Table 5.4. ABCO's performance in these experiments is similar to Cartpole's results regarding its inefficiency in learning policies that yield good average rewards. In the Acrobot environment, ABCO needs more samples than previously due to the complexity of the domain, thus creating the need for additional analysis on the matter of policy efficiency without the IDM learning involved, or with a larger amount of samples during training. Nevertheless, when comparing the thresholds of all algorithms, we can see that ABCO is closer to DQN, meaning that after learning the proper expert function, the algorithm is capable of maintaining its performance for a long time. DQN's results show that the exploration mechanism is beneficial for this environment. However, we believe that although the exploration might select a suboptimal action, the Acrobot environment is more forgiving with single non-ideal actions than Cartpole. DQN reaches each threshold faster with a lower deviation than its counterparts. For this environment, TRPO's results are further away from DQN, showing that for more complex domains, it is not as efficient as DQN.

Table 5.4: Timesteps results for ABCO, DQN and TRPO algorithms in the Acrobot environment using the threshold of the densest point. All results in the table are on $e5$ scale.

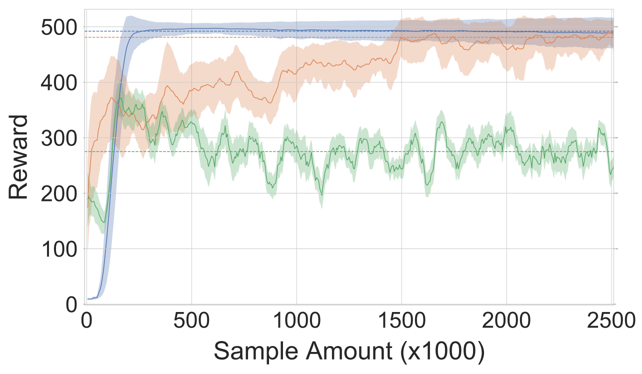
Threshold	**-79 (TRPO)**	**-78 (ABCO)**	**-72 (DQN)**
ABCO	5.62 ± 5.62	8.17 ± 5.61	24.3 ± 12.6
DQN	$\mathbf{0.30 \pm 0.30}$	$\mathbf{1.41 \pm 0.53}$	$\mathbf{2.82 \pm 0.97}$
TRPO	1.41 ± 1.41	4.38 ± 1.41	7.01 ± 1.96

Mountaincar: Even though Mountaincar-v0 has a simple goal of driving a car to the top of a mountain, the action *do nothing* and the necessity of building momentum to achieve its goal make this environment difficult for on-policy models such as TRPO. However, this environment becomes easier to solve for algorithms that possess off-policy mechanisms, *e.g.*, replay buffers such as DQN. The thresholds are, thus, as we expected. TRPO produces the lowest possible value, -200, -185 for DQN, and -146 for ABCO. When analyzing its densest point, we can expect DQN to underperform when compared to ABCO. Nevertheless, when looking into the maximum reward achieved by each approach, DQN achieves -99 against ABCO's -110.33. These results show that although DQN has a more difficult time reaching consistently higher rewards, it reaches them more efficiently. When analyzing TRPO's runs separately, we observe that the algorithm has difficulty reaching the goal, with its maximum reward being -190. Hence, for an environment such as MountainCar, which heavily penalizes highly-exploratory algorithms, TRPO cannot appropriately optimize its loss.

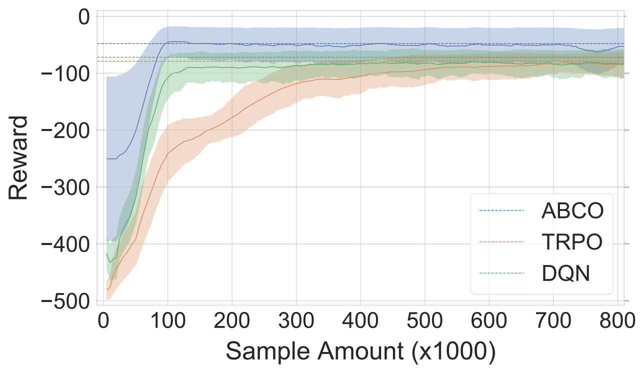
Table 5.5: Timesteps results for ABCO, DQN and TRPO algorithms in the Mountaincar environment using the threshold of the densest point. All results in the table are on $e5$ scale.

Threshold	**-200 (TRPO)**	**-185 (DQN)**	**-146 (ABCO)**
ABCO	0.05 ± 0.0	6.070 ± 2.466	6.900 ± 2.515
DQN	0.05 ± 0.0	$\mathbf{0.740 \pm 2.154}$	$\mathbf{1.190 \pm 0.852}$
TRPO	0.05 ± 0.0	*Did not reach*	*Did not reach*

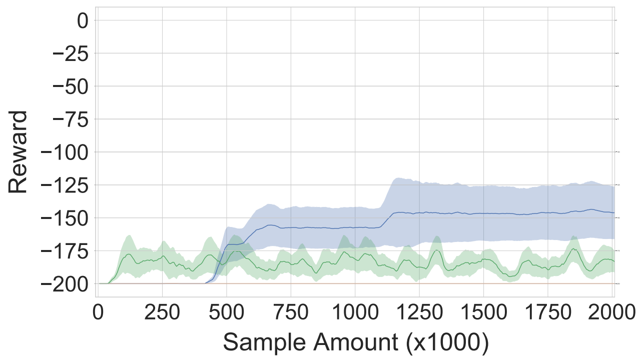
Overall: When analyzing the thresholds for each algorithm, depicted in Figure 5.5, we conclude that the IL algorithm was capable of staying on par with the RL algorithms in terms of performance. However, it proves to be sample-inefficient in each environment we test it. ABCO defines two different neural networks, IDM and PM, which need to learn the domain-specific actions and the unknown expert function. This learning method creates a scenario that is hard to be more efficient than the RL algorithms in this work. Furthermore, for harder environments, ABCO seems to require more samples, or maybe the network topology should be increased, since its results for the Acrobot environment are drastically lower than the others, while in the MountainCar environment the timesteps needed are 8.2 times higher than DQN's. This conclusion is possible to see in Figure 5.6, where ABCO takes longer to start increasing its reward than the RL algorithms.



(a) CartPole



(b) Acrobot



(c) MountainCar

Figure 5.6: Visualization of the average reward through time over 10 runs for all environments. The dotted lines are the thresholds defined in Section 4.2.

DQN is more efficient in environments with some degree of flexibility to errors. At the same time, even though TRPO is closer to DQN in the CartPole experiments, it is not as efficient in the Acrobot and MountainCar environments, where algorithms with less exploration are rewarded. With those results in mind, we question whether training ABCO's IDM with the random samples until no further improvement is achieved could improve its sample efficiency. We perform these experiments next, and we call such an approach ABCO*.

5.5.3 Pre-trained ABCO

In the previous experiments, we hinder ABCO's performance by training both models from scratch with smaller $\mathcal{T}^{pre}$ sizes. In this section, we experiment with ABCO policy only, by giving it a head start with a pre-trained IDM. The assumption is that since the first model learns from a set of random samples, it would be easier to gather $\mathcal{T}^{pre}$ experiences than expert's, and it is more efficient to train offline until there is no further improvement than online like the RL algorithms do. For the IDM training, we use $10,000$ random samples gathered before each training round, and we optimize this model using Adam [25]. Afterwards, we make use of the pre-trained model in ABCO as described in Section 5. We hypothesize that since ABCO's learning phase is offline (the online phase is simply a way to gather new samples for its first model), the training time would be shorter and more efficient. The expert samples continue the same in terms of size as in all prior experiments. We use the same thresholds for a better comparison of the difference between the pre-trained model (ABCO*), the online trained model (ABCO), and the best algorithm for that domain.

We observe that ABCO* is, in fact, more efficient than its prior model and the RL algorithm DQN. Table 5.6 shows that it maintains a lower standard deviation in all three Cartpole's thresholds and a significantly smaller sample size for reaching each objective. Granting that this domain is considered the simplest of the four domains, the Acrobot experiment presents a similar result. For the first two thresholds, ABCO* needs more than four times fewer samples to reach each objective. However, as the threshold gets closer from the expert's reward, -70 for the Acrobot domain, it is harder for it to reach it. ABCO* needs 3.28 times more samples than DQN to reach the -72 mark, showing that it is efficient to learn in a first stage, but inefficient at reaching higher rewards.

Table 5.6: Amount of timesteps needed to reach the goal in Cartpole, Acrobot and MountainCar environments. All results in the table are on $e5$ scale.

	Cartpole		**Acrobot**		**MountainCar**	
Threshold	275 (DQN)	492 (ABCO)	-78 (ABCO)	-72 (DQN)	-185 (DQN)	-145 (ABCO)
ABCO	1.07	2.74	8.17	24.300	6.070	6.900
ABCO*	**0.108**	**0.190**	**0.323**	9.26	1.805	1.935
DQN	0.520	1.05	1.41	**2.82**	**0.740**	**1.190**

With those results in mind, we hypothesize that combining both RL and IL approaches could be beneficial in terms of performance and sample efficiency. The offline training, together with the simplicity of generating random samples from a domain, could be useful when combined with the exploration and external resources that RL usually provides. We believe that by mixing both approaches we could achieve far better results in terms of samples needed and final achieved reward. This is further evaluated in Chapter 7.

5.6 Discussion

ABCO overall results show that the sampling strategy for the policy is detrimental for the iterative process created by Torabi *et al.* [57]. For cases where a policy performs predominantly one action, appending all samples from π_ϕ into $\mathcal{I}^s$ creates the issue of vanishing actions, as shown in Section 5.4.1. This problem can be from an underfitted IDM, that labels all state transitions as the same action, or an underfitted policy, performing the same action. When we append these samples to an already existing balanced dataset, the new samples can unbalance the distribution and force the IDM to overpredict one action over the others, creating a vicious cycle, where each epoch degrades the performance of the policy. However, when using a perfectly balance dataset, the IDM will predict all actions as being equally likely, which is not the case for a vast number of environments, e.g., MountainCar's *do nothing* action. Thus, having a sampling mechanism that diminishes the probability of unlikely actions with no supervision while not vanishing them is of great importance.

ABCO's sampling mechanism combined with the iterative process creates a constant shift into $\mathcal{I}^s$ leading to more drastic weight updates for the IDM. Furthermore, since IDM constantly changes the label from the expert samples, the policy model can receive the same state twice, between epochs, with different labels. This behavior can cause some overfitting for the policy model. The approach of using self-attention modules drastically reduced this behavior in our method by letting the model weigh the representations of the states and maintain a more stable classification process.

Finally, we experimented with the efficiency of ABCO when compared to standard reinforcement learning baselines. ABCO does not need to have access to an environment during its learning process, although without the iterative process its results do not reach values closer to the expert, as shown by Torabi *et al.* [57] work. Thus a trade-off could be used to understand when using an imitation learning method is preferable. We found out with our experiments that mechanisms that allow a model to experiment in the environment, as well as pretraining the IDM, have a beneficial impact on the efficiency of the IL method.

Nonetheless, the exploration mechanism can damage results, *e.g.*, DQN and TRPO in Section 5.5.2. We hypothesize that using an exploration mechanism that can diminish its occurrence with time, such as the case for DQN, would be ideal for an IL method. ABCO iterative process creates a perfect framework for an exploration mechanism given that it only uses expert samples into IDM, not creating cases where a poor policy could degrade its performance. Furthermore, merging reinforcement and imitation learning methods may result in better efficiency, allowing for fewer expert samples. As mentioned in Section 5.5.2, to achieve such a combination we believe an off-policy method should be used to reduce the complexity of the model. Having a policy that heavily depends on its current experience, without any replay buffers, could create cases where the model does not achieve any improvement in the earlier epochs, where ABCO lacks in efficiency the most.

6. IMITATING UNKNOWN POLICIES VIA EXPLORATION

Imitating Unknown Policies via Exploration (IUPE) follows the BCO [57] framework, further augmenting it with two strategies for avoiding local minima, *sampling* and *exploration*, and with self-attention modules (see Section 5.3) for improving the learning of global features and, hence, generalization.

IUPE tackles the problem of ABCO and BCO models using maximum-likelihood estimation in the beginning of training, when they are not sure of the state transition function and domain, respectively. IUPE helps us understanding how an exploration mechanism can impact the imitation learning paradigm and how the distances from $\mathcal{I}^{pre}$ and $\mathcal{I}^{pos}$ behave in an iterative process.

IUPE is capable of outperforming ABCO, and consequently, all state-of-the-art approaches based on behavior cloning. It was published in the British Machine Vision Conference (BMVC 2020) [13].

6.1 Sampling Method

IUPE uses a sampling strategy very similar to ABCO. It samples from the post-demonstration by only using runs that achieved the environmental goal, presented in Equation 5.1. However, after sampling the successful runs, IUPE uses all runs from $\mathcal{I}^{pos}$ instead of resampling according to the win-probability.

Afterwards, it samples from the pre-demonstrations $\mathcal{I}^{pre}_{spl}$ with the inverse probability of the post-demonstrations, *i.e.*, the loss probability distribution given by $1 - P(\mathcal{A}|E;\mathcal{I}^{pos})$. In a nutshell, the samples (observations) that comprise the novel post-demonstration dataset are sampled proportionally to $P(\mathcal{A}|E;\mathcal{I}^{pos})$ for winning executions, and the dataset is filled with the pre-demonstrations proportional to the number of losses. Algorithm 6.1 details IUPE and its sampling strategy.

IUPE, like ABCO, is only goal-aware and does not use any reward information for learning or optimizing the models. We do not want to use rewards since not all environments provide intuitive reward functions for solving the problem.

6.2 Exploration

The original behavioral cloning framework uses the *maximum a posteriori* (MAP) estimation, *i.e.*, it predicts the action with the most significant probability given by the model for a pair of states, both in its original version and its α-iteration version. By using MAP

Algorithm 6.1 Imitating Unknown Policies via Exploration.

```
1: Initialize model $\mathcal{M}_\theta$ as a random approximator
2: Initialize policy $\pi_\phi$ with random weights
3: Generate $\mathcal{I}^{pre}$ using policy $\pi_\phi$
4: Generate state transitions $\mathcal{T}^e$ from demonstrations $D$
5: Set $\mathcal{I}^s = \mathcal{I}^{pre}$
6: Let $\alpha$ be the number of improvement cycles
7: for $i \leftarrow 0$ to $\alpha$ do
8:     Improve $\mathcal{M}_\theta$ by trainIDM($\mathcal{I}^s$)
9:     Use $\mathcal{M}_\theta$ with $\mathcal{T}^e$ to predict actions $\hat{A}$
10:    Improve $\pi_\phi$ by behavioralCloning($\mathcal{T}^e$, $\hat{A}$)
11:    for $e \leftarrow 1$ to $|E|$ do
12:        Use $\pi_\phi$ to solve environment $e$
13:        Append samples $\mathcal{I}^{pos} \leftarrow (s_t, \hat{a}_t, s_{t+1})$
14:        if $\pi_\phi$ at goal $g$ then
15:            Append $v_e \leftarrow 1$
16:        else
17:            Append $v_e \leftarrow 0$
18:        end if
19:    end for
20:    Set $\mathcal{I}^s$ = sampling($\mathcal{I}^{pre}$, $\mathcal{I}^{pos}$, $P(g|E)$, $v_e$)
21: end for
```

predictions, we identified several cases in which the model is still relatively unsure about the correct action, especially in earlier iterations, leading to undesired local minima.

We borrow a simple solution from the area of language modeling to avoid such a problem, which is to sample the actions from the softmax distribution of both models (IDM during the expert labeling and PM during the execution of the environment). By not using the MAP estimation, we create a stochastic policy capable of further exploration in early iterations considering the model uncertainty. We show that by creating those samples we allow the IDM to converge in fewer iterations, since the sampling method guarantees a more sparse dataset consisting of $\mathcal{I}^{pre}$ and $\mathcal{I}^{pos}$, and the stochastic policy guarantees more exploration of the search space for properly achieving the environment goal.

Furthermore, in dynamic environments, the stochastic policy contributes to reaching the goal when deterministic behavior would not. This difference is vital to avoiding local minima during iterations. If the model was not capable of sampling a sub-optimal action during its training phase, the agent actions would resume for the most common action in the expert samples. By sampling the most frequent action, the policy is susceptible to looping between states, *e.g.*, choosing left and right interchangeably.

6.3 Experimental Analysis

In this section, we present the experimental analysis to validate the performance of IUPE. We review how the exploration method can help the imitation learning algorithm and how we can apply such a mechanism with a self-decaying exploration rate using the network's certainty on a classification.

6.3.1 Exploration Impact

Stochastic learning algorithms usually decay their exploration rate throughout iterations, since they assume that the agent progressively learns the optimal solution through time [36]. If the decaying rate is too low, the agent might stick with the first solution it finds, and if it is too high, it might spend too much time exploring sub-optimal states. By using the softmax distribution of actions, we create an exploration mechanism that naturally decays progressively as the neural network learns to separate the feature space (and hence no need for a decay hyperparameter to be tuned). The self-decaying exploration rate can be seen in Figure 6.1.

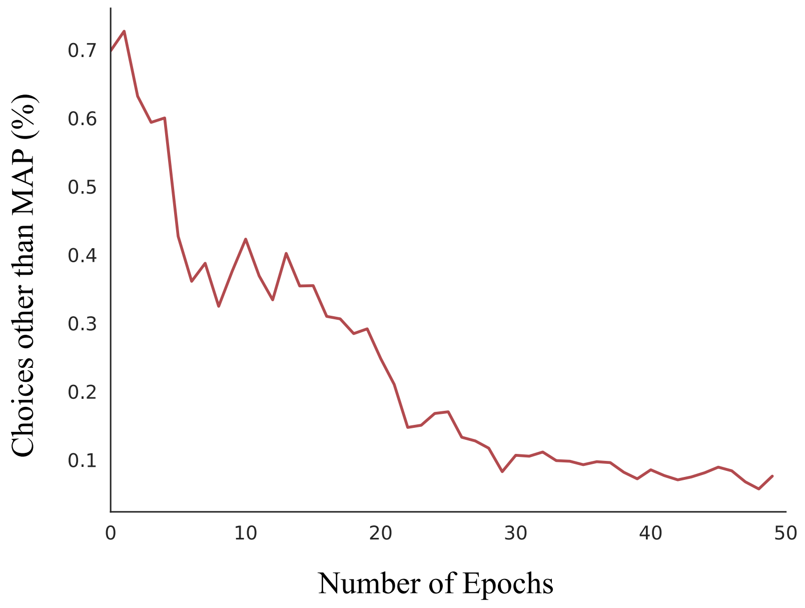


Figure 6.1: Percentage of choices in which the self-decaying exploration rate does not select the MAP estimation. At the beginning of training, when the certainty of the PM is lower, IUPE benefits from greater exploration, which naturally decays as training proceeds.

To understand whether IUPE benefits from this exploration mechanism, we can check the results in Table 6.1, which show the impact of exploration over the baseline (BCO), and also together with the remaining features (attention and sampling). By using this mechanism alone, we solve the problem of looping in-between actions, and even though the PM may reach the goal through non-ideal paths, it allows the generation of samples that are closer to the expert than of those created randomly.

The exploration mechanism, when paired with the sampling method, achieves results similar to IUPE (full method with the three strategies). This outcome is due to the increased stochasticity that helps in breaking action loops. While using exploration with SA results in a model with higher per-action accuracy, we perceive a decrease in both $\mathbb{P}$ and AER. We believe that this happens due to the action vanishing problem caused by the absence of the sampling method. When all mechanisms are combined, we can see that the SA modules do not impact

Table 6.1: Ablation study considering IUPE's 3 main components: *attention*, *sampling*, and *exploration* in the 10×10 maze environment.

Model	$\mathbb{P}$	*AER*
BCO	-0.416	-0.941
Attention	-0.415	-0.940
Sampling	0.534	0.348
Exploration	0.734	0.605
Attention + Sampling	0.367	0.088
Attention + Exploration	-0.407	-0.921
Sampling + Exploration	0.943	0.901
Attention + Sampling + Exploration (IUPE)	**1.000**	**0.981**

the model negatively anymore, but the opposite — they slightly increase both $\mathbb{P}$ and *AER*. With the exploration mechanism supporting the model and preventing it from getting stuck in-between states, IUPE can fine-tune itself earlier due to the non-optimal actions being closer to the expert. By adding the sampling mechanism, responsible for balancing $\mathcal{I}^s$, IUPE achieves the best results of this analysis.

6.3.2 Quantitative Results

As shown in Table 6.2, both methods outperform the state-of-the-art approaches for all environments but *CartPole*, where it provides the same results as the baselines. Results in CartPole are expected since it contains only four dimensions to represent the state, and the actions are easily separated. In Acrobot, IUPE, ABCO, and ILPO achieve similar $\mathbb{P}$ ($\mathbb{P} \approx 1.00$), showing that both approaches have similar imitation abilities. However, IUPE achieves $AER = -78.100$ while ILPO obtains -85.300, which means that IUPE can solve the environment with an advantage of ≈ 10 frames in advance than ILPO. For MountainCar, IUPE achieves $\mathbb{P} = 1.314$, which is the best result when compared with the baselines in this environment, a difference of ≈ 0.37 from the best second result (BCO). *AER* values for MountainCar are also much better for IUPE: -130.70, which is 19.3 better than BCO. For the Maze environments, IUPE reaches top $\mathbb{P}$ values for all types of Mazes that were tested. In terms of *AER*, IUPE was only matched by BCO in Maze 3×3, where both models achieve $AER = 0.927$. In the same environment, we can see that IUPE is virtually not affected as the complexity of the Maze increases, whereas the other approaches become incapable of learning those environments. ABCO can learn better policies due to the new sampling method, while IUPE learns from its exploration mechanism. IUPE stochasticity gives the model the ability to better explore the environment, avoiding local minima and reaching the goal at the end. Compared with the *Expert*, IUPE can achieve similar or better metric values for most environments.

Table 6.2: *Performance* and *Average Episodic Reward* for our approaches and related work using the OpenAI Gym environments.

Models	Metrics	CartPole	Acrobot	MountainCar	Maze 3×3	Maze 5×5	Maze 10×10
Expert	$\mathbb{P}$	1.000	1.000	1.000	1.000	1.000	1.000
	AER	442.628	−110.109	−147.265	0.963	0.970	0.981
Random	$\mathbb{P}$	0.000	0.000	0.000	0.000	0.000	0.000
	AER	18.700	−482.600	−200.000	0.557	0.166	−0.415
BC	$\mathbb{P}$	1.135	1.071	1.560	−1.207	−0.921	−0.470
	AER	500.000	−83.590	−117.720	0.180	−0.507	−1.000
BCO	$\mathbb{P}$	**1.135**	0.980	0.948	0.883	−0.112	−0.416
	AER	**500.000**	−117.600	−150.000	**0.927**	0.104	−0.941
ILPO	$\mathbb{P}$	**1.135**	1.067	0.626	−1.711	−0.398	0.257
	AER	**500.000**	−85.300	−167.000	−0.026	−0.059	−0.020
ABCO	$\mathbb{P}$	**1.135**	1.074	1.289	1.159	0.960	0.860
	AER	**500.000**	−82.500	−132.300	0.908	0.932	0.784
IUPE	$\mathbb{P}$	**1.135**	**1.086**	**1.314**	**1.361**	**1.000**	**1.000**
	AER	**500.000**	**−78.100**	**−130.700**	**0.927**	**0.971**	**0.981**

6.4 Discussion

IUPE achieves state-of-the-art results with its exploration mechanism in low-dimensional or image data. It improves ABCO's results substantially, and the ablations show that the exploration mechanism does indeed play a significant role in its improvement. When using only the exploration mechanism, IUPE achieves the best result over all other mechanisms. The exploration combined with the sampling, originally from the ABCO framework, achieves results near to the desired performance ($\mathbb{P} = 1$). Nevertheless, when combining all mechanisms, IUPE surpasses the expert performance reaching similar overall results than BC, a supervised approach. We believe that even though the self-attention mechanism does not benefit IUPE as much as it does to ABCO, it has an essential role since the dataset shift from the iterative process can produce changes in the action for state transitions causing heavy updates in the policy weights.

A possible adaptation for IUPE is in its exploration mechanism, which is only applicable to discrete domains. Even though there exists reinforcement and/or imitation learning approaches focused on one specific domain, an adaptation of the exploration mechanism can be done to accommodate continuous environments. An exploration mechanism, such as SAFE [8], which does not use the reward as a form to explore the possible actions, could work well within this framework. We believe, however, that after adapting the exploration mechanism for continuous action spaces, the decaying behavior of the exploration values should be maintained.

7. COMBINED REINFORCEMENT AND IMITATION LEARNING

Considering that ML models usually depend on a vast numbers of samples, and expert examples are not easy to create or record due to time and effort constraints, a method that aims to reduce this necessity upon boosting the models' efficiency would be ideal. Combined Reinforcement and Imitation Learning (CRIL) follows the IUPE framework, further augmenting it with an additional strategy. We aim to solve the sample efficiency issue detected in ABCO and IUPE by introducing a reinforcement learning mechanism during the training loop.

CRIL uses all the mechanisms created in the IUPE method. Moreover, during the sampling portion of the training loop, it incorporates the usage of a reinforcement learning method. We believe that such modification can create policy updates that can adjust the current policy's trajectory, improving its efficiency and reducing training time.

7.1 Learning from its Own Experiences

Upon using BCO's training loop, the policy generates new examples to approximate $\mathcal{I}^{pos}$ samples from the expert to learn a function that can determine which action occurs between two states. IUPE further augments this process by weighing the $\mathcal{I}^{pos}$ population using the policy's performance. As the policy reaches more often the goal in the 100 episodes enjoyed between each training loop, the more $\mathcal{I}^{pos}$ comprises policy samples instead of the random samples from $\mathcal{I}^{pre}$. This mechanism helps the model reducing the necessity of creating a vast number of random samples. However, it does not alleviate the necessity of using a vast amount of expert samples for the policy training.

When using toy environments, the creation of a vast number of expert samples is effortless. One can train, or create, an algorithm to simulate how an expert would act in such an environment. Nevertheless, as the environment becomes more complex, the amount of work to create precise expert samples grows. A possible solution for this problem would be to use transfer learning mechanisms when training an agent. It first learns how to act in a simpler environment and afterwards it transfers its knowledge into a more complex scenario. However, upon transferring an agent from one environment to another, one might face several problems, i.e., state differences or action remapping. Therefore, improving the model's efficiency and eliminating the necessity of a vast number of samples is the ideal solution.

RL methods model agents as a MDP, which consists of an agent learning how to navigate in state spaces generally disregarding time as a factor [54]. Such definition is vital for combining RL methods with IUPE, since all agents modeled by IUPE are stationary, as described in Section 2.3. Furthermore, BCO training loop suffers from a significant flaw regarding efficiency. Upon creating the necessity of a policy enjoying an environment e number of times during each loop, it creates a time constraint for the training. During such time, the policy performs each action ignoring how much it did learn, which can eventually generate optimal results, considering there is no real correlation between validation accuracy and environment performance. However, in environments with long lives, *i.e.*, mountaincar, where an agent must perform 500 steps before ending an episode, the first couple of iterations might consume the vast majority of the training time due to poor modeled policies. Hence, we propose to use an RL mechanism at this moment to boost the current IUPE performance.

We alter Algorithm 6.1 to not only append the state transitions $(s_t, \hat{a}_t, s_{t+1})$ into $\mathcal{I}^{pos}$, but also into a replay buffer, which we use to learn with a reinforcement learning algorithm, depending on the environment selected, generating our new method CRIL (Combined Reinforcement and Imitation Learning). A complete visualization of the pipeline in CRIL is shown in Figure 7.1. We believe that any reinforcement learning algorithm can be used and provide a beneficial impact on the training of the policy model. The loss function, however, can create some shortcomings. Upon using an algorithm such as DQN, presented in Section 3.4, we might need to update the loss function to maintain the definition of an IL algorithm that does not use a reward function as a way of learning how to act in an environment.

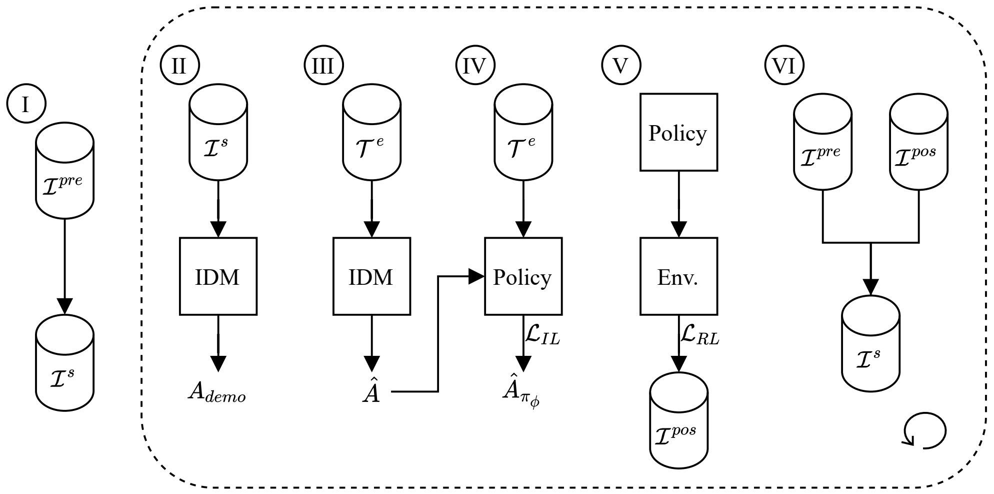


Figure 7.1: Combined Reinforcement and Imitation Learning framework: (i) set $\mathcal{I}^{pre}$, created by using π_ϕ, as $\mathcal{I}^{s}$; (ii) use $\mathcal{I}^{s}$ to learn the inverse dynamics of the environment; (iii) label the approximated expert action, $\hat{A}$, responsible for the state transitions in the expert samples, $\mathcal{T}^{e}$, with IDM; (iv) use $\mathcal{T}^{e}$ and $\hat{A}$ to train the policy, π_ϕ in an IL manner; (v) use π_ϕ in the environment to create new state transition samples and uses the RL approach to further learn from its experience; and (vi) use the IUPE sampling mechanism, from Section 6.1, to create a new dataset for IDM. CRIL is an iterative process that cycles from step (ii) to step (vi) until no further improvement is noticed.

A possible solution for such a problem might be to use a loss function such as GAIL's, where the distance between expert states and policy states is used to create a better discriminator in an inverse reinforcement learning fashion. Furthermore, upon analyzing the results from Section 5.5, we hypothesize that using off-policy methods with some degree of exploration would be more impactful for the framework's efficiency.

Nevertheless, our primary goal with the new method is to understand how both of these learning methods interact when intertwined. Consequently, on the results displayed in Section 7.3, we use the original loss function and reward from the environments, which we understand as potential limitations of this study.

7.2 Experimental Analysis

To validate the performance of CRIL, we use the early version of the DQN [35] algorithm combined with the IUPE framework. We believe that more advanced RL approaches would yield better results but could misrepresent how each learning method can impact the iterative learning process. Moreover, the main goal of these experiments is to understand whether IL can benefit from RL methods considering the results presented in Section 5.5. We do not alter any IUPE configurations during these experiments, using Adam [25] optimizer and the same hyperparameters from Section 6.

Upon combining DQN with IUPE, we create the necessity to tune a few new hyperparameters: replay buffer size, probability of exploration (ϵ), and discount factor (γ); for details of each one of these mechanisms, we direct the reader to Section 3.4. Standard value for γ is 0.99, while ϵ is usually divided into three different values: start, final, and decay — 1.0, 0.01, 500, respectively. For replay buffer sizes, the literature [35, 59, 49] argues that the larger, the better. When searching for the original implementation of DQN, we found that the consensus was $5,000$. During the experiments, we maintain the γ value according to the original work. Nevertheless, considering that we use DQN during an iterative process and the policy learns how to act in an environment outside the RL algorithm, we opt to alter the standard values from ϵ to adapt this algorithm to IUPE's iterative process. If we did not change the ϵ start value, the policy would randomly explore the environment each time, even when $\mathbb{P} = 1$. For the ϵ values, we use the $\mathbb{P}$ values, *i.e.*, as the policy becomes closer to the expert, the less the RL algorithm will explore with random actions. When experimenting with the replay buffer size, we came across a problem we believe to be due to the iterative process from IUPE. When using the standard value, we observe that on environments with a smaller number of steps per run, *e.g.*, Mountaincar (its expert only performs 100 steps on average), bigger replay buffers accumulate all steps and deteriorate the policy. For that reason, we alter the replay buffer size to be the same as $10\times$ the average steps per expert episode.

Considering the results found in Section 5.5 and the efficiency values from RL, we conducted experiments with the same amount of samples from the IUPE experiments in Section 6.3.2, and by feeding the policy only 1 expert episode. For comparison, all IUPE experiments had $10,000$ expert samples, which could represent $1,000$ episodes, while for the experiments with only 1 run, the policy received 500, 99, 103, samples for the CartPole, Mountaincar, and Acrobot, respectively.

Finally, together with the results from IUPE and CRIL, we also present how the DQN performs alone in each environment.

7.3 Results

7.3.1 Standard DQN Values

As shown in Table 7.1, when using the standard values for DQN into the CRIL framework, as the environment grows in difficulty, the algorithm performance deteriorates. Considering that the CartPole environment is less punitive regarding exploration, as mentioned in Section 5.5, we expected that without much tuning, the algorithm would achieve a perfect reward, 500.00. However, since Acrobot and MountainCar require an agent to build momentum to reach its final destination, the standard values achieve lower results. Nevertheless, the original setup of the algorithm achieves $\mathbb{P} = 0.811$ on the Acrobot environment, which we did not expect in advance. We hypothesize that the erratic actions, resulting from the highly exploratory nature of these hyperparameters, created states that the policy otherwise did not see with the expert samples, resulting in a more robust policy. Even though the Acrobot result surpasses our expectations, when comparing with the results from Table 7.3, one can see that this form of the CRIL framework achieves inferior rewards. The MountainCar environment, which heavily penalizes exploratory actions due to opposite movements reducing the car velocity, shows that the standard values do not benefit CRIL.

Table 7.1: *Performance* and *Average Episodic Reward* for CRIL using $\gamma = 0.99$, start $\epsilon = 1.0$, final $\epsilon = 0.01$, ϵ decay $= 500$, and replay buffer size $= 5,000$.

Models	Metrics	CartPole	Acrobot	MountainCar
CRIL*	$\mathbb{P}$	1.00	0.811	0.00
	AER	500.00	-174.90	-200.00

7.3.2 Sample Efficiency

Next, we test how CRIL compares to IUPE in terms of sample efficiency. We run each algorithm with all samples and with only one expert sample. For each environment, we select the first run of each expert. This selection results in superior samples for the experiments with one episode, which can result in more difficult performance to achieve. Table 7.2 shows that IUPE achieves better results when using more samples, while CRIL performs better with only one expert sample. We believe that CRIL fails to reach up IUPE results due to the nature of DQN updating the policy weights based on its own experiences, while IUPE updates are based on a classification problem. During our tests, we clip the gradients from DQN between

-1 and 1 to understand whether less erratic updates can create a smoother update sequence. However, no improvement was found. When using all samples, CRIL achieves rewards closer to IUPE than IUPE from CRIL during the experiments with only one expert sample. The average difference in performance for the experiments with fewer samples is 1.112, while for the experiments with all samples is 0.059. The same can be seen in the AER metric, where the average difference for the fewer samples is 438.607, while for all the samples is 5.367.

Table 7.2: *Performance* and *Average Episodic Reward* for CRIL and IUPE using $10,000$ samples and one expert episode for all three environments.

Models	Metrics	CartPole		Acrobot		MountainCar	
		All episodes	One episode	All episodes	One episode	All episodes	One episode
Expert	$\mathbb{P}$	1.000	1.000	1.000	1.000	1.000	1.000
	AER	442.628	500.00	-110.109	-103.000	-147.265	-100.00
Random	$\mathbb{P}$	0.000	0.000	0.000	0.000	0.000	0.000
	AER	18.700	18.700	-482.600	-482.600	-200	-200
IUPE	$\mathbb{P}$	**1.135**	0.278	**1.086**	0.839	**1.314**	0.180
	AER	**500.000**	152.500	**−78.100**	-164.000	**−130.700**	-182.000
CRIL	$\mathbb{P}$	**1.135**	**1.135**	1.079	**1.026**	1.157	**0.791**
	AER	**500.000**	**500.000**	-80.700	**−93.260**	-139.000	**−120.900**

7.3.3 Quantitative Results

In Table 7.3, we compare both original methods and the CRIL framework. For the sake of comparing both IL methods, we use the IUPE results with only one expert episode. However, DQN does not depend on the number of expert samples since it is an RL algorithm. For these DQN experiments, we use $100,000$ timesteps, which is the same number if we considered each expert sample as a step during the classification problem, and the number of steps performed by DQN during the sampling mechanism. In this scenario, CRIL achieves the best result in each environment. We perceive that DQN outperforms IUPE as expected, considering IUPE's original problem with efficiency and the nature of deep learning problems with dataset sizes. Nevertheless, CRIL still achieves results far superior to DQN. The average performance difference between both algorithms is 0.325, while the AER difference is 133.387. We believe these results show that the RL has a beneficial impact on CRIL's overall results. Notwithstanding, when comparing DQN to IUPE with all samples, in Table 7.2, we see that given the necessary amount of samples, IUPE can outperform the RL approach.

7.4 Discussion

CRIL achieves results that are similar to ABCO and near IUPE's. It uses the IL framework (IUPE) to improve its learning curve with the expert samples and the RL approach

Table 7.3: *Performance* and *Average Episodic Reward* for DQN, IUPE, and CRIL for all three environments using only one expert episode.

Models	Metrics	CartPole	Acrobot	MountainCar
Expert	$\mathbb{P}$	1.00	0.00	0.811
	AER	500.00	-100.00	-103.90
Random	$\mathbb{P}$	0.000	0.000	0.000
	AER	18.700	-482.600	-200.000
DQN	$\mathbb{P}$	0.771	0.976	0.651
	AER	390.000	-111.980	-134.900
IUPE	$\mathbb{P}$	0.278	0.839	0.180
	AER	152.500	-164.000	-182.000
CRIL	$\mathbb{P}$	1.00	1.026	0.791
	AER	**500.00**	$-\mathbf{93.260}$	$-\mathbf{120.900}$

(DQN) to correct its trajectory between different epochs. CRIL maintains the exploration mechanism from IUPE while adopting the replay buffer and discount factor from DQN. The replay buffer allows CRIL to update its weight based on an off-policy approach, which creates long-term memories. Thus, if the policy fails to reach the goal in an epoch, the replay buffer helps the RL approach optimize its weights based on past episodes, avoiding local minima, as seen in Section 5.5.2. The exploration mechanism ensures that the model, while uncertain (with lower confidence), can act in sub-optimal actions to create samples that tend, with time, to approximate the policy and expert trajectories, as seen in Section 6.3.

CRIL offers a good trade-off between efficiency and predictive performance. It accomplishes these results using a substantially smaller sample pool, 20 to 100 times smaller than IUPE, while achieving *AER* and $\mathbb{P}$ closer to the state-of-the-art. Furthermore, CRIL combines the first iteration of the DQN algorithm, as explained in Section 7.2. We believe that combining more advanced approaches can yield better results, though the complexity of adapting the different mechanisms of each algorithm can be challenging.

As future work, we want to investigate such a combination using different RL algorithms to verify how different RL approaches can benefit self-supervised IL methods. Nevertheless, it is important to remember that RL approaches that focus its loss function with a reward term force the IL method to have access to a reward function, which defeats the purpose of IL. We believe the next step for this research is to explore inverse reinforcement learning approaches, considering their ability to optimize a policy without any knowledge of a reward function, or adapting the loss function from RL approaches to support the IL scenario.

8. CONCLUSIONS

In this work, we proposed three different algorithms based on the BCO [57] framework. We first created a method called ABCO, which augments the original BCO framework by introducing: (i) an improved sampling method; and (ii) a self-attention mechanism. The first mechanism weigh how much of the policy samples is used in a new iteration, while the second help minimizing the weights updates that regularly change due to the dataset shift. By introducing these two new mechanisms, we solved the negative feedback loop problem encountered during our experiments. However, ABCO was prone to get deadlock between states, similarly to BCO, *i.e.*, moving back and forward in a maze.

We then analyzed the efficiency of the ABCO method to understand how it compares to RL methods. The experiments in Section 5.5 show that, without prior knowledge, the ABCO framework is inefficient with fewer expert samples. However, when combining the method with a pre-trained IDM model, ABCO achieved better results in fewer samples.

To address the deadlock issue, we created a second method that further improved ABCO by remodeling its sampling and adding an exploration mechanism, namely IUPE. This exploration mechanism uses the softmax likelihood to select the desired action. As the model improves its predicting capabilities, it is more likely to select the MAP prediction, as shown during the discussion in Section 6.3.1. IUPE achieves state-of-the-art results, as displayed in Section 6.3.2. Nevertheless, the necessity of a vast number of expert samples to learn a function that can perform as well as the expert is a major concern. This is particularly true when considering that the goal of imitation learning applications is creating agents capable of acting in real-life scenarios. The complexity of certain environments may require the recording of hours and hours of an expert performing the desired activity.

To reduce the necessity of a large number of expert data, we created a novel framework called CRIL. It is a method that combines RL and IL to update the policy inside the original BCO framework from the first two proposed approaches. It updates the policy weights using its experiences during the $\mathcal{I}^{pos}$ creation. CRIL achieves similar results to ABCO and IUPE, with fewer expert samples, as shown in Section 7.3. For the RL component, we simply used a traditional version of the DQN method to understand how IUPE interacts when updating the policy weights outside the original training loop. Nevertheless, we hypothesize that one can use more advanced RL methods to achieve even better results.

Even though each method described here presented similar results, with IUPE having the state-of-the-art predictive performance, we believe that each algorithm can be used in distinct situations. In environments with less room for exploration, or that are more punitive, as seen in Section 5.5, ABCO can achieve good results with fewer iterations since it uses the MAP prediction to create new samples for $\mathcal{I}^{pos}$. For environments with more freedom for algorithms that allow for exploration, IUPE can achieve results superior to the expert. Finally, when expert samples are hard to acquire, CRIL can achieve results resembling the expert.

Furthermore, CRIL achieves performances that are not far from the results achieved by the other two IL algorithms.

8.1 Limitations

Each method in this work has its own limitations, even though individually they were created to solve a problem from the original framework. While ABCO and IUPE share the efficiency limitation, ABCO also lacks the capability of exploring possible states to avoid deadlock actions. The efficiency from CRIL, on the other hand, takes a toll on the overall performance of the algorithm. The exploration from IUPE could be used as a hyperparameter to deactivate its mechanism for environments with less room for exploration. Furthermore, better RL algorithms can be employed in this framework to achieve better results while maintaining the efficiency from CRIL.

Other limitations we faced during this work were the computational time regarding each experiment. Some experiments took weeks to finish due to the iterative nature of the framework worked on. For each iteration, the agent has to play e amount of times, which leads to more prolonged experiments. As an environment becomes more complex, the more an agent takes to learn an appropriate function. This deficiency results in an agent that does not perform well and only enjoys an environment until the max number of timesteps. During some experiments, the policy enjoyment can use an early stop mechanism due to the ABCO and IUPE nature of discarding episodes that do not achieve the environmental goal. Nevertheless, CRIL uses each experience to learn how to act, nullifying this possibility.

Another limitation faced during this work was the availability of source code from IL methods, or of well-documented environments. Although they are usually released as papers of important conferences, most of them lack community support, or sometimes even the creator's support. Some environments are created inside OpenAI Gym framework [6], which facilitates when learning how to adapt an agent to learn into a new domain. However, some environments do not follow any framework and lack the documentation necessary to understand the essential components. Moreover, OpenAI Gym environments are RL environments with RL metrics, which (i) create the necessity of implementing IL metrics; and (ii) are mostly vector-state focused. The first problem creates the necessity of implementing newer code and testing whether the code will work on each new environment, while the latter creates problems when adapting vector to image-state.

Finally, we believe that the current metrics for imitation learning are limited regarding the measurement of how well an agent performed. Performance, as well as Average Episodic Reward, uses the final reward of an agent instead of its mimicking capabilities. These metrics allow for the possibility of an agent learning a path different from the expert and achieving the same result. A perfect example of that would be the following. Assume that a maze has two possible paths of the same size, and the agent and the expert each take one of the paths. The final reward would be the same, though the agent did not learn to imitate the expert. Assuming that the goal of IL is to create an agent as good as the expert, this would not be a problem. However, if a domain fails to account for a relevant action, *i.e.*, picking some item inside the maze, the metric would not reflect that. Ergo, both metrics are actually overly reliant on the reward function designed for each environment.

We believe that a new metric that considers the distance from teacher and learner would be better suited to understand how their behavior differ. The cost involving in calculating such a distance depending on the state representation could result in much slower training times, which is something to be considered when thinking of such a metric.

8.2 Future Work

As future work, we believe that adapting IUPE to continuous environments is a natural next step. Considering that the exploring mechanism used in IUPE makes use of the softmax distribution of a model's prediction, it creates the necessity of using another exploration mechanism for continuous action spaces. The literature for RL exploration mechanism is vast [56, 43, 34, 8], and an existing method could be combined into the already existing IUPE framework. However, we believe an adaptation would be necessary since the IUPE framework acts in a self-supervised regression approach. Considering that BCO [57] achieves results far better than GAIL, we hypothesize that this adaption could achieve state-of-the-art performance on continuous action spaces.

Another possible future work would be to explore the different reinforcement learning mechanisms and how they impact the iterative nature of the CRIL framework. We assume that upon using newer algorithms, one might achieve better results. However, it could face higher complexity regarding the multiple optimization functions. This complexity can create more intricate optimization problems with a higher number of hyperparameters to control, and maybe a different approach for combining reinforcement and imitation learning might be necessary. A different venue to research would be to use inverse reinforcement learning to achieve the same results. We note that GAIL already uses an inverse reinforcement learning approach to approximate its policy from the expert, and this might indicate that combining this approach with IL can yield good results.

Finally, we believe that even though an IL metric was not the focus of this work, there might be the necessity of creating a new way to measure the effectiveness of IL algorithms. We consider that the sole goal of IL methods should not be to achieve expert rewards. IL agents should also perform trajectories as close as possible to the expert samples.

8.3 Published Work

• **Gavenski, Nathan** ; Monteiro, Juarez; Granada, Roger; Meneguzzi, Felipe; and Barros, Rodrigo C. Imitating Unknown Policies via Exploration. In Proceedings of the 31st British Machine Vision Conference (BMVC), Manchester, UK, 2020. (Qualis A1)

• Monteiro, Juarez; **Gavenski, Nathan**; Granada, Roger; Meneguzzi, Felipe; and Barros, Rodrigo C. Augmented Behavioral Cloning from Observation. In Proceedings of the 33rd International Joint Conference on Neural Networks (IJCNN), Glasgow, Scotland, 2020. (Qualis A2)

8.3.1 On Going Work

• **Gavenski, Nathan**; Monteiro, Juarez; Granada, Roger; Meneguzzi, Felipe; and Barros, Rodrigo C. Sample Efficiency of Reinforcement and Imitation Learning Algorithms.

• **Gavenski, Nathan**; Monteiro, Juarez; Granada, Roger; Meneguzzi, Felipe; and Barros, Rodrigo C. Combining Reinforcement and Imitation Learning

REFERENCES

[1] Argall, B. D.; Chernova, S.; Veloso, M.; Browning, B. “A survey of robot learning from demonstration”, *Robotics and Autonomous Systems*, vol. 57–5, May 2009, pp. 469–483.

[2] Aytar, Y.; Pfaff, T.; Budden, D.; Paine, T.; Wang, Z.; de Freitas, N. “Playing hard exploration games by watching youtube”. In: International Conference Advances in Neural Information Processing Systems, 2018, pp. 2930–2941.

[3] Barto, A. G.; Sutton, R. S.; Anderson, C. W. “Neuronlike adaptive elements that can solve difficult learning control problems”, *IEEE transactions on systems, man, and cybernetics*, vol. 1–5, Sep 1983, pp. 834–846.

[4] Bian, J.; Tian, D.; Tang, Y.; Tao, D. “Trajectory data classification: A review”, *ACM Transactions on Intelligent Systems and Technology*, vol. 10–4, Aug 2019, pp. 1–34.

[5] Bishop, C. M. “Neural Networks for Pattern Recognition”. Oxford University Press, Inc., 1995, 738p.

[6] Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. “Openai gym”, *arXiv preprint arXiv:1606.01540*, vol. 1, Jun 2016, pp. 1–4.

[7] Chan, M. “gym-maze”. GitHub Repository, Source: https://github.com/MattChanTK/gym-maze, 2021-06-22.

[8] Dalal, G.; Dvijotham, K.; Vecerik, M.; Hester, T.; Paduraru, C.; Tassa, Y. “Safe exploration in continuous action spaces”, *arXiv preprint arXiv:1801.08757*, vol. 1, Jan 2018, pp. 1–9.

[9] Edunov, S.; Ott, M.; Auli, M.; Grangier, D. “Understanding back-translation at scale”. In: International Conference on Empirical Methods in Natural Language Processing, 2018, pp. 489–500.

[10] Edwards, A. D.; Sahni, H.; Schroecker, Y.; Isbell, C. L. “Imitating latent policies from observation”. In: International Conference on Machine Learning, 2019, pp. 1755–1763.

[11] Fang, B.; Jia, S.; Guo, D.; Xu, M.; Wen, S.; Sun, F. “Survey of imitation learning for robotic manipulation”, *International Journal of Intelligent Robotics and Applications*, vol. 3–4, Sep 2019, pp. 362–369.

[12] Gatys, L. A.; Ecker, A. S.; Bethge, M. “Image style transfer using convolutional neural networks”. In: International Conference on Computer Vision and Pattern Recognition, 2016, pp. 2414–2423.

[13] Gavenski, N.; Monteiro, J.; Granada, R.; Meneguzzi, F.; Barros, R. C. "Imitating unknown policies via exploration". In: International British Machine Vision Virtual Conference, 2020, pp. 1–8.

[14] Geramifard, A.; Dann, C.; Klein, R. H.; Dabney, W.; How, J. P. "Rlpy: A value-function-based reinforcement learning framework for education and research", *Journal of Machine Learning Research*, vol. 16–46, Dec 2015, pp. 1573–1578.

[15] Goodfellow, I.; Bengio, Y.; Courville, A. "Deep Learning". MIT Press, 2016, 775p.

[16] Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. "Generative adversarial nets". In: International Conference on Neural Information Processing Systems, 2014, pp. 2672–2680.

[17] Han, K. J.; Prieto, R.; Ma, T. "State-of-the-art speech recognition using multi-stream self-attention with dilated 1d convolutions". In: International Conference on Automatic Speech Recognition and Understanding Workshop, 2019, pp. 54–61.

[18] He, J.; Spokoyny, D.; Neubig, G.; Berg-Kirkpatrick, T. "Lagging inference networks and posterior collapse in variational autoencoders". In: International Conference on Learning Representations, 2018, pp. 1–7.

[19] Hinton, G. E.; Sejnowski, T. J.; Poggio, T. A.; et al.. "Unsupervised learning: foundations of neural computation". MIT press, 1999, 398p.

[20] Ho, J.; Ermon, S. "Generative adversarial imitation learning". In: International Conference on Advances in Neural Information Processing Systems, 2016, pp. 4565–4573.

[21] Hu, J.; Niu, H.; Carrasco, J.; Lennox, B.; Arvin, F. "Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning", *IEEE Transactions on Vehicular Technology*, vol. 69–12, Dec 2020, pp. 14413–14423.

[22] Hussein, A.; Gaber, M. M.; Elyan, E.; Jayne, C. "Imitation learning: A survey of learning methods", *ACM Computing Surveys (CSUR)*, vol. 50–2, Jun 2017, pp. 1–35.

[23] Jing, L.; Tian, Y. "Self-supervised visual feature learning with deep neural networks: A survey", *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 1–01, May 5555, pp. 1–24.

[24] Kaelbling, L. P.; Littman, M. L.; Moore, A. W. "Reinforcement learning: A survey", *Journal of artificial intelligence research*, vol. 4, May 1996, pp. 237–285.

[25] Kingma, D. P.; Ba, J. "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, vol. 1, Jan 2014, pp. 1–15.

[26] Le, H. M.; Yue, Y. “Imitation learning tutorial”. ICML Presentation, Source: https://sites.google.com/view/icml2018-imitation-learning, 2021-06-22.

[27] LeCun, Y.; Bengio, Y.; Hinton, G. “Deep learning”, *Nature*, vol. 521–7553, Nov 2015, pp. 436–444.

[28] LeCun, Y.; Haffner, P.; Bottou, L.; Bengio, Y. “Object recognition with gradient-based learning”. Springer, 1999, 1 ed., vol. 1, chap. 19, pp. 319–345.

[29] Li, Y.; Liu, M.-Y.; Li, X.; Yang, M.-H.; Kautz, J. “A closed-form solution to photorealistic image stylization”. In: European Conference on Computer Vision, 2018, pp. 453–468.

[30] Liu, Y.; Gupta, A.; Abbeel, P.; Levine, S. “Imitation from observation: Learning to imitate behaviors from raw video via context translation”. In: International Conference on Robotics and Automation, 2018, pp. 1118–1125.

[31] Maisto, S. A.; Carey, K. B.; Bradizza, C. M. “Social learning theory.”, *Psychological Theories of Drinking and Alcoholism*, vol. 1, Jul 1999, pp. 106–163.

[32] McCulloch, W. S.; Pitts, W. “A logical calculus of the ideas immanent in nervous activity”, *The bulletin of mathematical biophysics*, vol. 5–4, Dec 1943, pp. 115–133.

[33] Mitchell, T. “Machine Learning”. McGraw-Hill, 1997, 432p.

[34] Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. “Asynchronous methods for deep reinforcement learning”. In: International Conference on Machine Learning, 2016, pp. 1928–1937.

[35] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. “Playing atari with deep reinforcement learning”, *arXiv preprint arXiv:1312.5602*, vol. 1, Dec 2013, pp. 1–9.

[36] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al.. “Human-level control through deep reinforcement learning”, *Nature*, vol. 518–7540, Feb 2015, pp. 529–533.

[37] Monteiro, J.; Gavenski, N.; Granada, R.; Meneguzzi, F.; Barros, R. C. “Augmented behavioral cloning from observation”. In: International Joint Conference on Neural Networks, 2020, pp. 1–8.

[38] Moore, A. W. “Efficient memory-based learning for robot control”, Ph.D. Thesis, University of Cambridge, 1990, 42p.

[39] Norvig, P.; Russel, S. “Artificial Intelligence, A modern approach”. Prentice Hall, 2002, 1132p.

[40] Olivier, C.; Bernhard, S.; Alexander, Z. “Semi-supervised learning”. , IEEE Transactions on Neural Networks, 2006, vol. 20, pp. 542–542.

[41] Osa, T.; Pajarinen, J.; Neumann, G.; Bagnell, J. A.; Abbeel, P.; Peters, J. “An algorithmic perspective on imitation learning”, *Foundations and Trends® in Robotics*, vol. 7–1-2, Nov 2018, pp. 1–179.

[42] Payne, C. “Musenet”. OpenAI Blog about MuseNet, Source: https://openai.com/blog/musenet, 2021-06-22.

[43] Plappert, M.; Houthooft, R.; Dhariwal, P.; Sidor, S.; Chen, R. Y.; Chen, X.; Asfour, T.; Abbeel, P.; Andrychowicz, M. “Parameter space noise for exploration”. In: International Conference on Learning Representations, 2018, pp. 1–6.

[44] Raza, S.; Haider, S.; Williams, M.-A. “Teaching coordinated strategies to soccer robots via imitation”. In: International Conference on Robotics and Biomimetics, 2012, pp. 1434–1439.

[45] Rizzolatti, G.; Sinigaglia, C. “The functional role of the parieto-frontal mirror circuit: Interpretations and misinterpretations”, *Nature Reviews Neuroscience*, vol. 11–4, Mar 2010, pp. 264–274.

[46] Ross, S.; Gordon, G. J.; Bagnell, J. A. “A reduction of imitation learning and structured prediction to no-regret online learning”. In: International Conference on Artificial Intelligence and Statistics, 2011, pp. 627–635.

[47] Schaal, S. “Learning from demonstration”. In: International Conference on Neural Information Processing Systems, 1996, pp. 1040–1046.

[48] Schaal, S.; Ijspeert, A.; Billard, A. “Computational approaches to motor learning by imitation”, *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, vol. 358–1431, Feb 2003, pp. 537–547.

[49] Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. “Prioritized experience replay”, *arXiv preprint arXiv:1511.05952*, vol. 1, Nov 2015, pp. 1–21.

[50] Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. “Trust region policy optimization”. In: International conference on Machine Learning, 2015, pp. 1889–1897.

[51] Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; Batra, D. “Grad-cam: Visual explanations from deep networks via gradient-based localization”. In: International Conference on Computer Vision, 2017, pp. 618–626.

[52] Song, Y.; Ermon, S. “Generative modeling by estimating gradients of the data distribution”. In: International Conference on Advances in Neural Information Processing Systems, 2019, pp. 11895–11907.

[53] Sutton, R. S. “Generalization in reinforcement learning: Successful examples using sparse coarse coding”. In: International Conference on Advances in neural information processing systems, 1996, pp. 1038–1044.

[54] Sutton, R. S.; Barto, A. G. “Reinforcement learning: An introduction”. MIT press, 2018, 552p.

[55] Tan, P.-N.; Steinbach, M.; Kumar, V. “Introduction to data mining”. Pearson Education India, 2016, 864p.

[56] Tassa, Y.; Doron, Y.; Muldal, A.; Erez, T.; Li, Y.; Casas, D. d. L.; Budden, D.; Abdolmaleki, A.; Merel, J.; Lefrancq, A.; et al.. “Deepmind control suite”, *arXiv preprint arXiv:1801.00690*, vol. 1, Jan 2018, pp. 1–24.

[57] Torabi, F.; Warnell, G.; Stone, P. “Behavioral cloning from observation”. In: International Joint Conference on Artificial Intelligence, 2018, pp. 4950–4957.

[58] van Engelen, J. E.; Hoos, H. H. “A survey on semi-supervised learning”, *Machine Learning*, vol. 1, Nov 2019, pp. 1–68.

[59] Van Hasselt, H.; Guez, A.; Silver, D. “Deep reinforcement learning with double q-learning”. In: International Conference on Artificial Intelligence, 2016, pp. 1–17.

[60] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; Polosukhin, I. “Attention is all you need”. In: International Conference on Neural Information Processing Systems, 2017, pp. 5998–6008.

[61] Xie, Q.; Luong, M.-T.; Hovy, E.; Le, Q. V. “Self-training with noisy student improves imagenet classification”. In: International Conference on Computer Vision and Pattern Recognition, 2020, pp. 1–9.

[62] Zhang, H.; Goodfellow, I.; Metaxas, D.; Odena, A. “Self-attention generative adversarial networks”. In: International Conference on Machine Learning, 2019, pp. 7354–7363.

[63] Zhang, Z.; Yang, J.; Zhao, H. “Retrospective reader for machine reading comprehension”, *arXiv preprint arXiv:2001.09694*, vol. 1, Dec 2020, pp. 1–9.