Towards Corrective Deep Imitation Learning in Data Intensive Environments

Helping robots to learn faster by leveraging human knowledge

Master Thesis

Irene López Bosque

Delft University of Technology

TU Delft

Towards Corrective Deep Imitation Learning in Data Intensive Environments

Helping robots to learn faster by leveraging human knowledge

by

Irene López Bosque

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday November 25, 2021 at 02:00 PM.

Student number:	5051487	
Project duration:	January, 2021 – November, 2021	
Thesis committee:	Dr. ir. J. Kober,	TU Delft, supervisor
	Dr. C. Celemin Paez,	TU Delft, supervisor
	Ir. R. Pérez Dattari,	TU Delft, supervisor
	Dr. W. Pan,	TU Delft, reader

Cover Image: ”Two robot arms” by David Levêque. Source: Unsplash

TU Delft

Abstract

Interactive imitation learning refers to learning methods where a human teacher interacts with an agent during the learning process providing feedback to improve its behaviour. This type of learning may be preferable with respect to reinforcement learning techniques when dealing with real-world problems. This fact is especially true in the case of robotic applications where reinforcement learning may be unfeasible as there are long training times and reward functions can be hard to shape/compute.

The present thesis focuses on interactive learning with corrective feedback and, in particular, in the framework Deep Corrective Advice Communicated by Humans (D-COACH), which has successfully shown to be advantageous in terms of training time and data efficiency. D-COACH, a supervised learning method whose policy is represented by an artificial neural network, incorporates a replay buffer where samples of states and corresponding labels gathered by the agent's policy from human feedback are stored and replayed. However, this causes conflicts between the data in the buffer because samples collected by older versions of the policy may be contradictory and could deteriorate the performance of the current policy. In order to reduce this issue, the current implementation of D-COACH uses a first-in-first-out buffer with limited size, as the older the sample is, the more likely it is to deteriorate the performance of the learner. Nonetheless, this limitation propitiates catastrophic forgetting, an inherent tendency of neural networks to forget what they have already learnt, and that can be mitigated by replaying information gathered during all the stages of the problem. Therefore, D-COACH suffers from a trade-off between reducing conflicting data and avoiding catastrophic forgetting. The fact that D-COACH limits the size of its buffer automatically restricts the types of problems that it can solve, given that, if the problem is too complex (i.e. it requires large amounts of data), it simply will not be able to remember everything.

If we want to utilise a buffer to train data intensive tasks with corrective feedback, a new method is needed to solve the problem of using information gathered by older versions of the policy. We propose an improved version of D-COACH, which we call Batch Deep COACH (BD-COACH, pronounced "be the coach"). BD-COACH incorporates a human model module that learns the feedback from the teacher and that can be employed to make corrections gathered by older versions of the policy still useful for batch updating the current version of the policy.

To compare the performance of BD-COACH with respect to D-COACH, simulated experiments with three different problems were done using the open-source Meta-World benchmark, which is based on MuJoCo and OpenAI gym. Moreover, to validate the proposed method in a real setup, two planar manipulation tasks were solved using a seven degrees of freedom KUKA robot arm.

Furthermore, we present an analysis between on-policy and off-policy methods both in the fields of reinforcement learning and in imitation learning. We believe there is an interesting simile between this classification and the problem of correctly implementing a replay buffer when learning from corrective feedback.

Imagination is the discovering faculty, pre-eminently. It is that which penetrates into the unseen worlds around us, the worlds of Science.

— Ada Lovelace

Acknowledgements

Delft, November 8, 2021

It was an event of chance that I ended up attending the course *Knowledge-Based Control Systems* of professor Jens Kober. I remember doubting until the last minute to enroll in his class and I could not be happier to have decided to take it. This course was my first contact with machine learning, a world until then unknown to me and which I wish to continue exploring. I am profoundly grateful to my daily supervisors Carlos Celemin and Rodrigo Pérez-Dattari, for their invaluable insight, interesting discussions and corrective feedback. Immense gratitude goes towards my parents and sister because, without their support, this work would not have been possible. My final thanks go to Lidia for accompanying me every day despite the kilometers in between.

Contents

1 Introduction

We can envision our lives in a not too distant future where robots and intelligent agents make our existence easier. From kitchen robots to automated factories, the demand for technology that is able to quickly adapt to changing environments does not stop growing. This necessity of fast adaptation to new situations makes hard-coding control strategies less suitable and requires other types of flexible control approaches [1]. Reinforcement learning (RL) is one of these flexible approaches where an intelligent agent tries to learn how to perform a task by maximizing a sum of rewards in a trial and error manner [2]. RL techniques have been applied in successful cases such as [3], [4], [5], however, these examples tend to happen in simulated environments with very specific learning tasks. This fact greatly differs from real-world situations such in robotics where, in order to learn a good policy, a robot agent requires a huge amount of data, for which it will have to perform thousands of trials [6]. This is very costly both in terms of time and the probable physical damages caused to the robot and its surroundings while learning [7]. Furthermore, many real problems are easier to demonstrate than to design a reward function for applying reinforcement learning [8]. In fact, the incorporation of human knowledge in the learning process results in dramatically more efficient methods compared to autonomous learning techniques [9]. Imitation learning (IL) is the name of the field that leverages the knowledge of a teacher to improve the learning process. The simplest form of IL is known as behavioural cloning (BC) where an expert provides initial demonstrations of the desired task and then, an agent tries to imitate that behaviour via supervised learning. Behavioural cloning has two main drawbacks: First, it requires demonstrations from an expert teacher which limits the possibilities of who can train the agent. And secondly, it suffers from covariate shift, a distribution mismatch problem that initiates at the moment the agent deviates from the expert trajectory causing a cascade of errors that will probably make the agent fail the task.

Interactive imitation learning (IIL) is a branch of imitation learning [10] that deals with the aforementioned issues by allowing a teacher to help the agent learn *during* its training, see Figure 1.1. The human can communicate his/her knowledge using different types of feedback, such as demonstrations when requested by the robot [11], evaluations that can be scalar values [7] or preferences [12], corrections [13], and so forth. In this work, we focus on corrections which gives name to the branch of IIL called corrective imitation learning (CIL). In CIL frameworks the human teacher sends corrections informing the agent whether the value of a taken action should be increased or decreased [14].

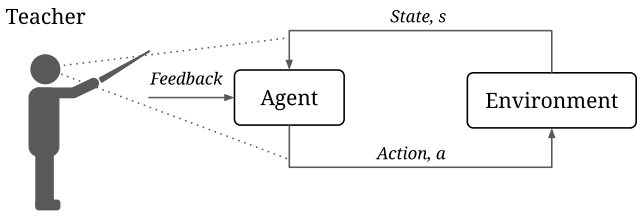


Figure 1.1: Interactive Imitation Learning

The goal of this master thesis is to create an extension of Deep COrrective Advice Communicated by Humans, D-COACH [15], a CIL algorithm designed for non-expert humans that uses an artificial neural network as a function approximator for its policy. This extension focuses on the improvement of a key component in several deep learning algorithms in sequential decision-making problems, experience replay (ER) [3], a technique that leverages past experiences stored in a replay buffer. ER endows algorithms with two main advantages, these being a higher data efficiency and the ability to train with uncorrelated data [16]. These benefits are especially useful when policies are approximated with artificial neural networks (ANNs). Collected past experiences can be reused multiple times and the ANN gets more robust against locally overfitting to the most recent trajectories, a phenomenon known as *catastrophic forgetting*. Note that we refer to ER as *corrections replay* since, in this work, we replay old corrective feedback. In RL, only off-policy frameworks, those that can update their current policies with data gathered by different policies, are able to correctly employ ER [3]. As this is related to the problem we want to solve, in this work, we explore and analyze the terms on-policy and off-policy in imitation learning.

D-COACH already incorporates corrections replay for data-efficiency purposes. However, the way it is implemented, causes ambiguities to arise between the data inside the buffer. This forces to limit the size of the buffer which works under the assumption that the data stored in the replay buffer is still valid for the current version of the policy, even if it was collected by an older version of the policy. As the size of the replay buffer starts to increase, this assumption does not hold anymore and the training of the policy will most likely fail, limiting therefore, the types of problems that D-COACH can address. If we want to utilise a buffer to train data intensive tasks with corrective feedback, a new method is needed to solve the problem of using corrections gathered by older versions of the policy. To address this challenge, we propose Batch Deep COACH (BD-COACH, pronounced "be the coach"). BD-COACH incorporates a human model module that learns the feedback from the teacher and that can be employed to make corrections gathered by older versions of the policy still useful for batch updating the current version of the policy.

1.1. Objectives

The aim of this master thesis can be divided in the following objectives:

- Define and implement a framework that, by using deep neural network policies, allows robots to learn data intensive tasks by replaying old corrective feedback given by a human teacher from a replay buffer.
- Validate the proposed method in simulated experiments.
- Validate the proposed method using a real robotic platform.

1.2. Outline

This work first presents the theoretical framework and relevant work in Chapter 2. Chapter 3 follows with the presentation of the proposed algorithm BD-COACH which is validated with the experiments, both in simulation and in a real setup explained in Chapter 4. Chapter 5 shows the results obtained with the new method and finally, the report ends in Chapter 6 with the relevant conclusions.

2

Background and Related Work

This chapter provides an overview of the theoretical framework and the relevant literature required for this master thesis. In Section 2.1, the basics of reinforcement learning are presented, a theoretical background that is necessary for later better understanding imitation learning and the terms on-policy and off-policy. In Section 2.3, after explaining what is imitation learning, several IL algorithms are classified according to our proposed definitions of on-policy and off-policy imitation learning. Lastly, Section 2.4 focuses on D-COACH, the corrective imitation learning algorithm that will be the base of our proposed new framework.

2.1. Reinforcement Learning

Reinforcement learning is a branch of machine learning where an agent interacts with an environment in a trial-and-error manner with the objective of finding which of the actions it can take will maximize a numerical reward signal [2]. Reinforcement learning makes use of Markov decision processes (MDP) a framework for sequential decision making problems defined by the tuple of sets $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, see Figure 2.1. At time step t, the agent finds itself at state $s_t \in \mathcal{S}$ and performs an action $a_t \in \mathcal{A}(s)$ on the environment following a policy $\pi : \mathcal{S} \to \mathcal{A}$. Then, and as a result of the transition function $\mathcal{P}$, the environment evolves to the next state s_{t+1} where the agent receives a reward $r_{t+1} \in \mathcal{R}$.

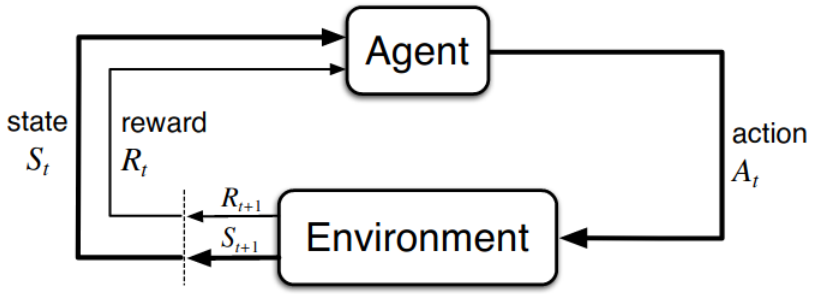


Figure 2.1: Interaction between the agent and the environment in reinforcement learning [2].

MDPs satisfy the *Markov property* that says that for a sequence of states $s_0, s_1, ..., s_t$, at any time t, the next state s_{t+1} only depends on the current state s_t [17]. A state, s, is a sufficient summary of what is going on in the environment and when it is possible to access all this sufficient information, the environment is said to be *fully observable*. At every time step, the agent perceives an *observation* that is a consequence of the state of the environment at that moment. Sometimes, just from the available observation, it is not possible to extract the state underneath and then, the environment is said to be *partially observable*. The goal of the agent is to select the

actions that will maximize the expected discounted return G_t after k time steps, see Equation (2.1).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

In previous Equation (2.1), γ is the *discount rate*, a parameter in the range $0 \leq \gamma \leq 1$ that represents the value that future rewards have at the current time step t. At each time step, the environment sends a *reward signal* to the agent that indicates how well the agent is performing in the *immediate* sense. On the other hand, the *value* of a state indicates how desirable is a state in the *long-run*. In other words, the value of a state, $v_\pi(s)$, is the expected discounted return G_t that an agent will receive over the future if it starts interacting at that state s and continues following its policy π. This is captured by the *value function* shown in Equation (2.2):

$$v_\pi(s) = E_\pi[G_t \mid S_t = s] \tag{2.2}$$

Similarly, the *action-value function* or *Q-function*, denoted as q_π and shown in Equation (2.3), tells us how good it is for the agent to take an action at a given state while following a policy π. The output of the Q-function for any given state-action pair is called the *Q-value*.

$$q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a] \tag{2.3}$$

When a reinforcement learning agent starts learning in an unknown environment it has to *explore* new states to discover where the best rewards are. At the same time it has to *exploit* what it already knows to choose the best actions. This is known as the *exploration-exploitation* dilemma and it is a key challenge in reinforcement learning [2].

2.1.1. On-policy and Off-policy Reinforcement Learning

As presented in Chapter 1, the objective of this thesis is to develop an extension of the D-COACH method that is able to learn by replaying old corrections. We consider that there is a simile between this challenge and the concepts on-policy and off-policy in reinforcement learning given that, "when learning by experience replay, it is necessary to learn off-policy" [3].

According to Sutton and Barto, off-policy methods use two policies: They evaluate or improve a policy, the target policy, while using a different policy to generate the data, the behavior policy. In this case, we say that learning is from data "off" the target policy, and the overall process is termed off-policy learning [2]. On the other hand, on-policy methods use a single policy, the same policy that is evaluated or improved, is the one used to generate behavior. To better understand the terms, we are going to briefly explore two popular RL algorithms, Q-Learning [18] and SARSA [19] to understand what makes the former off-policy and the later on-policy.

In SARSA the agent starts at state s_t and chooses an action a_t according to its policy $\pi(s_t, a_t)$. Then, it observes the reward r_t and the next state s_{t+1} where it chooses the next action a_{t+1} again according to its policy $\pi(s_{t+1}, a_{t+1})$. At this point it has all the required information to apply the update rule showed in Equation (2.4).

$$Q^{\text{new}}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[R + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \tag{2.4}$$

Q-learning is similar to SARSA with the difference that for the *update step*, it uses a *greedy* policy that chooses the actions that yields the highest estimated Q-value, see Equation (2.5)). This is the reason why it is called off-policy, because, for the update, it ignores the policy π that was used to take action a_t at state s_t.

$$Q^{\text{new}}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[R + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \tag{2.5}$$

The fact that Q-learning uses a policy for generating behaviour different from the greedy policy with which it updates the Q-values is the key to understand why Q-learning can use experience replay, see Section 2.1.3.

2.1.2. Online and Offline Reinforcement Learning

Reinforcement learning algorithms can also be classified according to how data is collected. Traditional RL algorithms are online frameworks where the agent iteratively interacts in its environment, collecting experience to update its policy. Online methods can be on-policy or off-policy depending on whether the data is exclusively gathered by the current policy or not, see the previous section. The online approach works well in simulated environments however, for real-world settings, online learning is impractical because the agent still needs to collect a diverse and large dataset.

Offline reinforcement learning, addresses the aforementioned problem. The key idea is that using only previously collected data, the agent has to learn the best possible policy without additional online data collection [20]. With this offline framework, it is possible to apply RL to real-world domains like robotics where the agent, the robot, could easily get damaged while collecting data iteratively in an online manner. The downside of offline learning resides in collecting a suitable dataset for training, which for many real-world applications, can be very expensive to acquire [21].

2.1.3. Experience Replay

Experience Replay is a replay memory technique in which transitions $< s_t, a_t, r_{t+1}, s_{t+1} >$ gathered by an agent at every time step t are stored into a memory or *replay buffer*. The idea of experience replay is to randomly sample mini-batches of experience from the buffer to update the policy.

Experience replay provides several benefits. First, it is an efficient way of taking advantage of previously collected experience by replaying it multiple times [16]. This is particularly important in the case of artificial neural networks because it increases their stability by preserving old knowledge and thus reducing catastrophic forgetting [22]. Furthermore, experience replay provides uncorrelated data to train the neural network, which helps it to generalize and to minimize overfitting to the most recent trajectories.

To be able to use this experience replay technique, it is necessary to have an off-policy algorithm. Off-policy RL methods continue estimating the optimal Q-value, Q^*, even if the policy that they follow changes from π_1 to π_2. However, an on-policy method following a policy π_1 will yield a Q-value Q_{π_1} and if the policy changes to π_2, it will yield Q_{π_2}. The optimal Q-value in this last situation will not be guaranteed. It is important to remark that even if the experiences were collected with a single policy π, because the policy evolves over time, the same policy at time step t, π_t, is not equal to that same policy in a later time step, π_N. They are considered experiences gathered by *different policies* and therefore only off-policy methods are applicable.

2.2. Function Approximation with Artificial Neural Networks

The novel method presented in this thesis uses an artificial neural network (ANN) as a non-linear function approximator of the policy π that the agent follows. ANNs are computational models inspired by biological neural systems and formed by interconnected processing elements. In particular, for this thesis, we will focus on fully-connected feedforward neural networks (FNN) where information flows in only one direction without going through any loop. In FNNs, the processing elements are called artificial neurons and they are arranged in consecutive structures called layers. There are three types of layers, the input layer, the hidden layer(s) and the output layer, see right of Figure 2.2. This hierarchy of layers makes it possible to model complex relationships among data as multiple levels of representation are learnt [23].

An artificial neuron, see left of Figure 2.2, is the simplest element of an FNN. Similar to a biological neuron, an artificial neuron has input connections through which it receives external stimuli, the input data x [24]. The neuron computes a weighted sum of the input data where the weights w are the parameters of the model that have to be adjusted in order for the model to learn. Furthermore, there is an additional input connection to the neuron, the parameter bias b that also is added to the weighted sum, $wx + b$. The final element of the artificial neuron is the activation function f that takes the previous weighted sum as input and distorts it by adding non-linear deformations, $f(wx + b)$.

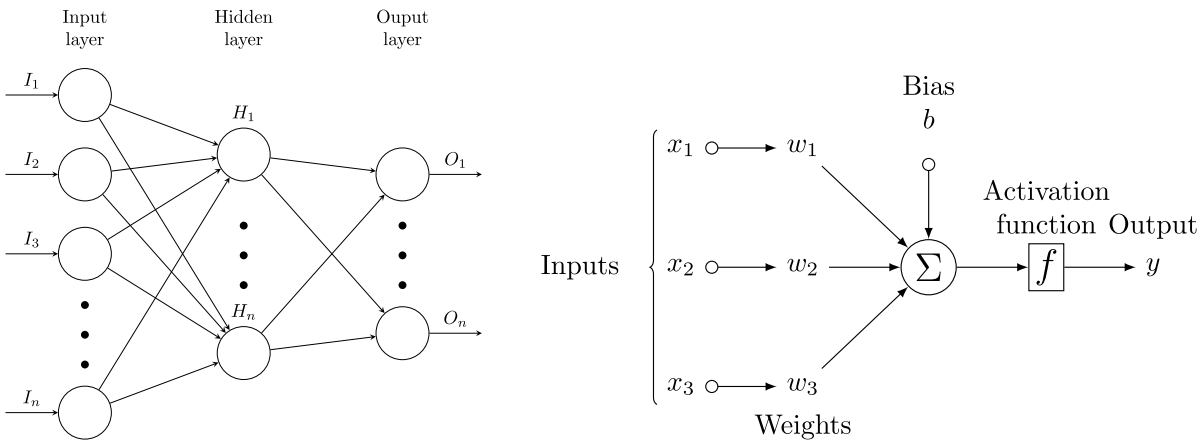


Figure 2.2: On the left, a simplification of the layers of fully-connected feedforward artificial neural network where all the neurons from one layer are connected to every neuron in the next layer. On the right, the main components of a single artificial neuron.

2.3. Imitation Learning

Imitation learning is a problem in machine learning where an agent learns a task leveraging the knowledge of a teacher which can be a human or a more experienced agent [25]. Imitation learning is more useful with respect to reinforcement learning when it is easier for a teacher to demonstrate or provide feedback in order for the agent to learn rather than to specify a reward function that would lead to the desired behaviour. The simplest imitation learning algorithm is called *behavioural cloning* where a teacher provides some initial demonstrations and the agent tries to learn a policy via supervised learning.

2.3.1. Interactive Imitation Learning

Interactive imitation learning (IIL) is a branch of imitation learning [10] where human teachers can help intelligent agents to learn *during* their training, see Figure 1.1. There are different forms in which the human can communicate his/her knowledge to facilitate the learning process. One way is for the human to demonstrate the task when the robot request it [11] for example by teleoperating the robot or by using kinesthetic teaching. Another type of feedback are evaluations. Evaluative feedback in the form of scalar values (e.g., the framework Training an Agent Manually via Evaluative Reinforcement, TAMER [7]) inform the agent how good or bad was the action taken. Evaluative feedback can also be provided as preferences between demonstrated trajectories [12]. Here the human is presented with several executions of a policy, and he/she has to decide which one is better according to the goal of the task. Then, a reward function that explains the decisions of the human is found, and by applying RL, the agent learns how to perform the task. This kind of evaluative feedback is easier to implement than to define a reward function and allows a faster convergence than pure autonomous learning. However, the informativeness of this kind of evaluative feedback is still limited [26], and one way to improve it is to use corrections.

Feedback as relative corrections (e.g., [14]) gives name to *corrective imitation learning (CIL)*, a branch of IIL. Corrective imitation learning improves the informativeness of evaluative feedback, by allowing the teacher to inform the agent whether the value of a taken action should be increased or decreased [14] and it requires less exploration compared to evaluative feedback [26].

2.3.2. On-Policy and Off-Policy Imitation Learning

Subsection 2.1.1 explained the meaning of the terms on-policy and off-policy in the field of reinforcement learning. When researching the literature it is clear that, for RL, the definition provided by Sutton and Barto in 1998 is the most used. However, things are not that obvious when talking about imitation learning. According to the paper *An Algorithmic Perspective on Imitation Learning* [17] the first mention to the terms on-policy and off-policy regarding imitation learning is due to [27]. These definitions are:

"In Off-Policy imitation learning, an agent observes demonstrations from a supervisor and tries to recover the

behavior via supervised learning, an example of off-policy IL is behavioral cloning. On-policy imitation learning methods sample trajectories from the agent's current distribution and update the model based on the data received. A common on-policy algorithm is DAgger."

Similar definitions are found in [28]:

"On-policy imitation learning involves executing the current agent's policy π_{θ_i} in the environment allowing it to make errors and observe new states and then soliciting feedback from a supervisor on the visited states to update π_{θ_i}. This is in contrast to off-policy imitation learning algorithms where policy learning is performed entirely on states from the supervisor's trajectory distribution."

We consider that it is necessary to redefine the terms on-policy IL and off-policy IL in order for their meaning to be closer to the well-accepted definitions in reinforcement learning. Therefore our reinterpretation of the definitions is:

- **On-policy imitation learning**: "On-policy IL methods sample trajectories using the current agent's policy and use that data to update that same policy."
- **Off-policy imitation learning**: "Off-policy IL methods sample trajectories using any policy and use that data to update the current agent's policy."

For the case of off-policy imitation learning, *any policy* could be a combination of different policies including the one of the agent, as it happens in off-policy RL.

In [27], [29] and [17] the first example given for an on-policy imitation learning algorithm is DAgger (Dataset Aggregation) [30]. However, we consider important to make a clarification regarding this classification. The general principle of DAgger is to use a policy to collect a dataset at each iteration and then it trains the next policy under the aggregate of all collected datasets [30]. The policy used to collect the dataset depends on whether it is the simple version or the general version of DAgger, both represented in Algorithm 1. During the first iteration, both versions initialize the policy as the expert's policy π^* (line 2). The difference is that, for the next iterations, the general version implements a modified policy $\pi_i = \beta_i\pi^* + (1-\beta_i)\hat{\pi}_i$ (line 8) to better leverage the knowledge of the expert. Depending on $\beta = [0, 1]$, the expert policy π^*, is able to collect part of the next dataset by itself. If $\beta = 1$, the expert has full control when collecting the data, opposite to $\beta = 0$ which means that all the data is gathered by the current agent's policy. Therefore, we assume that the authors refer to the simple version of DAgger when classifying it as on-policy IL. But, even if referring to the simple version of DAgger, there is another reason to consider it off-policy. By its very nature, DAgger learns from a buffer, in other words, it learns from information gathered by older versions of the policy that are different from the current version.

Algorithm 1 DAgger

1: **Initialize** $\mathcal{D} \leftarrow 0$
2: **Initialize** $\hat{\pi}_1 \leftarrow \pi^*$
3: **for** $i = 1$ **to** N **do**
4: **if** Simple version of DAgger **then**
5: Let $\pi_i = \hat{\pi}_i$
6: **end if**
7: **if** General version of DAgger **then**
8: Let $\pi_i = \beta_i\pi^* + (1 - \beta_i)\hat{\pi}_i$
9: **end if**
10: Sample T-step trajectories using π_i
11: Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by π_i and actions given by expert
12: Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
13: Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$
14: **end for**
15: Return best $\hat{\pi}_i$ on validation

2.3.3. Online and Offline Imitation Learning

As it happens in RL, see Section 2.1.2, imitation learning algorithms can also be classified as online or offline frameworks. In offline imitation learning, the agent learns by imitating a demonstrator without additional online environment interactions unlike in the case of online IL [31]. This distinction between online and offline IL, which is similar to RL, justify our proposed definitions of on-policy IL and off-policy IL given that definitions for these terms found in the literature could be confused with the terms online and offline imitation learning.

To conclude this subsection, we provide Table 2.1 with a classification of several imitation learning algorithms according to the proposed definitions of the terms on-policy and off-policy IL as well as the terms online and offline IL. A list with a brief description of every method that appears in the table can be found in Annex A.

	On-policy Imitation Learning	**Off-policy Imitation Learning**
Online IL	TAMER [7] COACH [13] SHIV [32] I-SABL [33] Convergent Actor-Critic by Humans [34] Deep TAMER [35] D-COACH [15] DQN-TAMER [36] LOKI [37] AOR [29] ABLUF [38] EIL [39] Learning from human preferences [12]	DAgger [30] DAgger by coaching [40] Advise [41] AggreVaTe [42] Safe DAgger [43] DropoutDAgger [44] AggreVaTeD [45] Hierarchically Guided DAgger [46] BAgger [47] Cycle of Learning [48] Retrospective DAgger [49] SAIL [50] HG-DAgger [51] EnsembleDAgger [52] DAgger Replay Buffer [53] FRESH [54] FIRE [55] IWR [56] ThriftyDAgger [57]
Offline IL	-	Behavioural Cloning [58] DART [59] ValueDICE [60] SQIL [61]

Table 2.1: Classification of Imitation Learning algorithms

2.4. Corrective Imitation Learning

As explained before, the goal of this master thesis is to develop a novel corrective imitation learning algorithm that is able to learn by replaying past corrections saved in a memory buffer. This process is very similar to the experience replay technique presented in Section 2.1.3 and to emphasise the fact that we are replaying corrections we refer to this technique as *corrections replay* [62].

The new framework that we propose in Chapter 3 is based on the D-COACH algorithm which, in turn, derives from the COACH algorithm; both methods are presented next.

2.4.1. COACH: Corrective Advice Communicated by Humans

The method Corrective Advice Communicated by Humans, COACH [13], is a CIL framework designed for non-expert humans teachers where the person supervising the learning agent, provides occasional corrections when the agent behaves wrongly. This corrective feedback h is a binary signal that indicates the direction in which the executed action $a = \pi_\theta(s)$, should be modified. The parameters of the policy $\pi_\theta(s)$, θ are updated using a stochastic gradient descent (SGD) strategy in a supervised learning manner where the cost function $J(\theta)$ is the mean squared error between the applied and the desired action [63]. Equation (2.6) shows the general update rule of the parameters θ:

$$\theta \leftarrow \theta - \alpha \cdot \nabla_\theta J(\theta) \tag{2.6}$$

COACH works under the assumption that teachers are non-experts and that therefore, they are just able to provide a correction trend that tells the sign of the policy error but not the error itself. To compute the exact magnitude of the error, COACH incorporates a hyperparameter e that needs to be defined beforehand, resulting in $\text{error}_t = h_t \cdot e$ [13]. The error needs to be defined as a function of the parameters in order to compute the gradient in the parameter space of the policy. Thus, the error can also be described as the difference between the desired action generated with the teacher's feedback, $a_t^{\text{target}} = a_t + \text{error}_t$, and the current output of the policy, $a_t = \pi_\theta(o_t)$:

$$\text{error}_t(\theta) = a_t^{\text{target}} - a_t \tag{2.7}$$

Equation (2.8) shows the final update rule of COACH obtained from Equations (2.6), (2.7) and the derivative of the mean squared error:

$$\theta \leftarrow \theta + \alpha \cdot \text{error}_t \cdot \nabla_\theta \pi_\theta \tag{2.8}$$

Where $\nabla_\theta \pi_\theta$ is the gradient of the policy, $\frac{\partial \pi}{\partial \theta}$, w.r.t the parameters θ. Algorithm 2 shows the pseudocode of the basic COACH version:

Algorithm 2 Basic COACH

$e \leftarrow$ error magnitude
$\alpha \leftarrow$ learning rate
while *true* **do**
  $s \leftarrow$ getStateVec()
  execute action $a_t = \pi_\theta(s_t)$
  feedback human corrective advice h_t
  if h_t is not **0 then**
    $\text{error}_t = h_t \cdot e$
    $a_{\text{target}(t)} = a_t + \text{error}_t$
    update π using SGD with pair (s_t, a_t^{target})
  end if
end while

2.4.2. D-COACH: Deep COACH

Deep COACH, D-COACH [15], is the "deep" version of the COACH algorithm in the sense that it uses an artificial neural network to represent the policy of the agent; Algorithm 3 shows the pseudocode of D-COACH when learning from low-dimensional state space, and whose essence is very similar to COACH. The current version of D-COACH implements the corrections replay technique to be more data-efficient. During learning, tuples of old corrections, $(s_t, a_t^{\text{target}})$, are stored in a memory buffer $\mathcal{B}$ and then they are replayed to update the current policy of the agent. However, the way that D-COACH implements the replaying of corrections has

limitations. In this case, the replay buffer $\mathcal{B}$ works by assuming that *recent* feedback is still being valid to update the most recent version of the policy. Due to this assumption, the size of the buffer that D-COACH implements, needs to be drastically reduced, otherwise old corrections could update the policy in undesired directions of the policy's parameter space. On the other hand, a very small replay buffer will provoke an overfitting of the policy to data generated in the most recent trajectories, which limits the current version of D-COACH to work with low data intensive problems. It is necessary to mention that D-COACH [15] as well as [62] and [64] which are based on D-COACH, employ state representation learning (SRL) strategies for learning a low-dimensional representation of the state, whose raw version consist on high-dimensional image states. This reduction of dimensionality makes the problem require less data and therefore it is easier to solve.

Algorithm 3 Deep COACH

1: **Require:** error magnitude e, buffer update interval b
2: **Init:** $\mathcal{B} = [\quad]$ # *initialize memory buffer*
3: **for** t = 1,2,... **do**
4: **observe** state s_t
5: **execute** action $a_t = \pi_\theta(s_t)$
6: **feedback** human corrective advice h_t
7: **if** h_t is not **0 then**
8: $\text{error}_t = h_t \cdot e$
9: $a_{\text{target}(t)} = a_t + \text{error}_t$
10: **update** π using SGD with pair $(s_t, a_t^{\text{target}})$
11: **update** π using SGD with a mini-batch sampled from $\mathcal{B}$
12: **append** $(s_t, a_t^{\text{target}})$ to $\mathcal{B}$
13: **end if**
14: **if** mod(t, b) is 0 **then**
15: **update** π_θ using SGD with a mini-batch sampled from $\mathcal{B}$
16: **end if**
17: **end for**

Figure 2.3 shows a graphical representation of the D-COACH algorithm. The red loop indicates a single update of the policy, which happens when the teacher provides a correction (line 10). On the other hand, the green loop represents a batch update in which by replaying corrections from the buffer, the policy is updated every time step that the teacher provides feedback (line 11) as well as every b time steps (line 15).

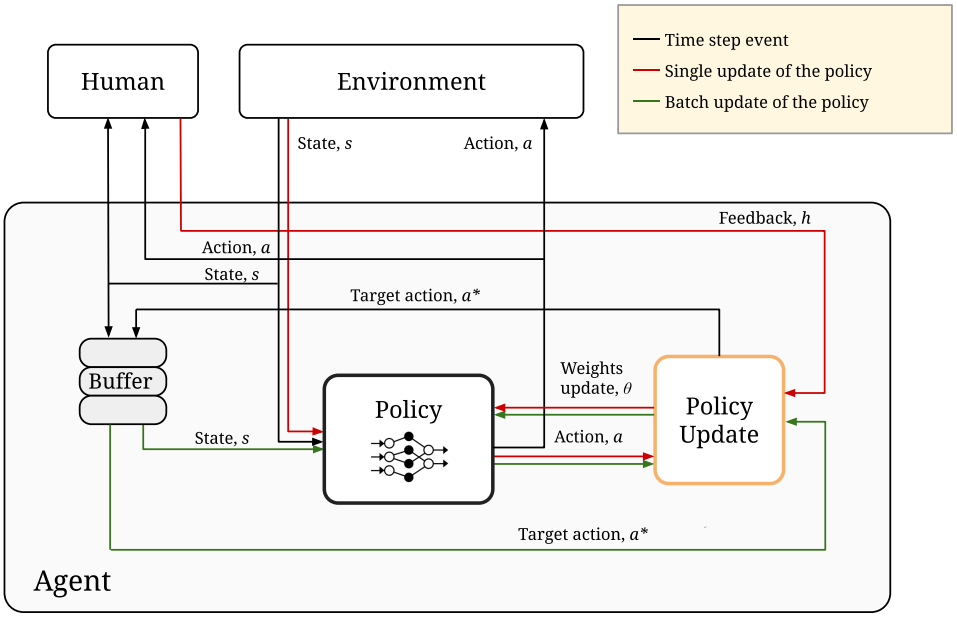


Figure 2.3: D-COACH agent

3

Batch Deep COACH (BD-COACH)

In this chapter we address the question of how a corrective deep imitation learning algorithm can truly leverage old collected corrections. The current version of the CIL algorithm D-COACH is limited to problems that do not require large amounts of data as its replay buffer needs to be kept small. A new method is therefore needed to maximise the full potential of the replay buffer which will allow to train for tasks that are more data demanding. In this chapter we present the method Batch Deep COACH (BD-COACH) an extension of D-COACH that is able to perform policy dependent updates by incorporating a human model module that learns the feedback from the teacher.

Section 3.1 focuses on the difference between D-COACH and BD-COACH.

Section 3.2 explains the modules of the BD-COACH framework.

Finally, section 3.3 gathers the conclusions about the new proposed method.

3.1. Difference between D-COACH and BD-COACH

To mitigate catastrophic forgetting and overfitting, D-COACH makes use of a replay buffer whose size is kept small to avoid old contradictory corrections to be conveyed to the current version of the policy for its update. This limitation makes D-COACH perform worse when facing tasks that require a lot of data and thus, it is limited to the type of problems it can address. There are several aspects that determine the complexity of a task and therefore, the amount of data required for training it. On the one hand, there are problems related to the observation itself, such as how complex it is, or its dimensionality; the higher the dimensionality of the observation, the more difficult the problem is. One way to mitigate the problem of high-dimensional observations such as images is to employ state representation learning (SRL) strategies for learning a low dimensional representation of the state, as D-COACH does. The horizon of the task also has a great impact, the longer the horizon, the more data is required. Finally, another important aspect is the complexity of the decision space of the task. Therefore, a task with a combination of a complex and high-dimensional observation space plus a long horizon and a complex observation-action mapping, would be very challenging for D-COACH to learn.

In D-COACH, batch updates are independent of the policy as corrections from the buffer do not depend on what the policy is currently doing at those particular states. This fact makes that feedback gathered by older versions of the policy can deteriorate the performance of the current policy. If we want to correctly implement the corrections replay technique, we need to answer the following question: How can corrections gathered by older versions of the policy and which are stored in a replay buffer still useful for batch updating the current version of the policy? Remark that this question addresses the *batch update* and not the single update which will be done in the same way as in D-COACH. We need corrections that *depend* on the current agent's policy instead of using corrections gathered by older policies, as happens in D-COACH. This is exactly what the human

teacher does and what we want to imitate. The human observes the state and the action of the agent at a particular moment and gives a correction accordingly. We introduce a human teacher learner module in our framework as an artificial neural network that takes pairs of state/actions and outputs the appropriate feedback correction. This module, called the human model, is learned in parallel together with the policy. At the beginning of the training the corrections predicted by the human model will be inadequate, but we expect that, as the training progresses and the human model improves, the predicted corrections will also improve.

We coin this new method Batch Deep COACH (BD-COACH), a CIL framework with policy dependent batch updates. BD-COACH is able to work with higher demanding data tasks thanks to the human model module that learns to predict the feedback that the teacher provides. These predicted corrections depend on the output of the policy at a particular state making them convenient for updating the current version of the policy.

Method	Feedback is:	Batch updates are:
D-COACH	Policy dependent	Policy independent
BD-COACH	Policy dependent	Policy dependent

Table 3.1: Difference between D-COACH and BD-COACH

Table 3.1 shows a summary of the dependencies. Notice that for both methods, corrections are always dependent of the policy as the human teacher provides corrections depending on what the policy is doing. What changes between D-COACH and BD-COACH is the dependency of the corrections during the batch updates.

3.2. Learning Framework

Figure 3.1 depicts the modules that compose the proposed method BD-COACH. The coloured arrows other than black identify the different updates of the policy and the human model. Specifically, the red arrows show the flow of information for a single update of the policy while the green ones are used for representing its batch update. Finally, the blue arrows indicate a batch update of the human model.

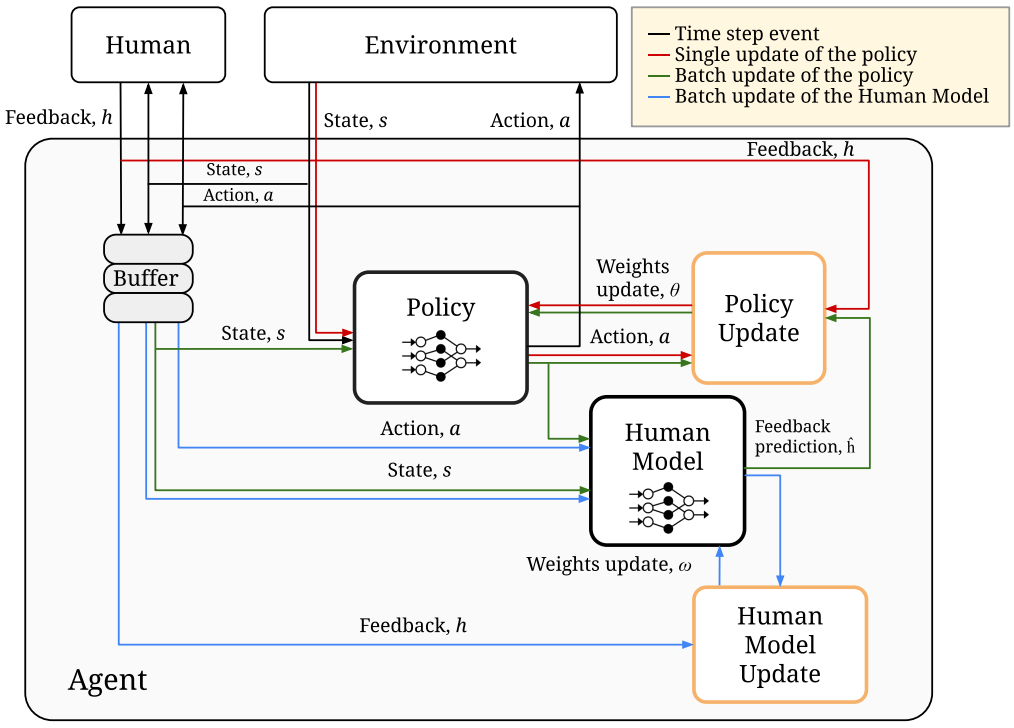


Figure 3.1: Modules of the BD-COACH framework

Policy

The policy $\pi(s)$ is the core of the agent that makes decisions in an environment. This function, represented in BD-COACH by an artificial neural network, observes the state of the environment and emits an action accordingly. The policy is learnt during training with both single and batch updates.

Replay Buffer

The replay buffer $\mathcal{D}$ is an artificial memory used to store the experience gathered by the policy on the time steps when the human teacher provides corrections. This information consists on a 3-tuple formed by the state of the environment at time step t, s_t, the correspondent action from the policy, a_t and the feedback signal from the human, h_t.

Policy Update Module

This module, which is the same as in D-COACH, is in charge of updating the policy with both single updates (red loop of Figure 3.1) and batch updates (green loop of Figure 3.1). The difference lies in the origin of the information that the module takes as input during the batch update. The corrections that are fed to the policy update module depend on the actions taken by the policy for those states. These corrections do not come directly from the replay buffer as in the case of D-COACH but instead are the output of the human model module.

Human Model

BD-COACH incorporates a human model, $H(s, a)$, that learns to predict the corrective feedback given by the human teacher for inputs of state-action pairs. This module is inspired in the module with the same name from the original COACH framework [14] where their human model is used to learn an adaptive learning rate which only depends on the states of the environment. The framework Gaussian Process Coach (GPC) [63] also employs a human model that, as our model, does take into account actions in addition to states. The difference is that GPC implements Gaussian processes as function approximators for both its policy and its human model to estimate the uncertainty of states and actions. In the case of BD-COACH, the human model is an artificial neural network that generate labels that are useful for the current version of the policy, see green loop in Figure 3.1.

During the batch update of the policy, a mini-batch of states uniformly sampled from the replay buffer is passed to both the policy and the human model as inputs. To clarify, these mini-batches of states are different from normal mini-batches as for this step, we do not use the actions or corrections from the buffer. For example, if the buffer sampling size is equal to 3, a mini-batch of states could be $\{(s_1), (s_4), (s_9)\}$, whereas a normal mini-batch used to update the human model would be $\{(s_1, a_1, h_1), (s_4, a_4, h_4), (s_9, a_9, h_9)\}$, see next subsection. The policy outputs then the corresponding actions that are fed to the human model. These actions together with the output $\hat{h}$ from the human model, serve to update the weights of the policy in the policy update module. The human model is learnt in parallel together with the policy during the training.

Human Model Update

The human model update module is in charge of updating the weights of the ANN that represents the human model, $H(s, a)$. The human model is updated with tuples of information (s_t, a_t, h_t) stored in the replay buffer, applying like this the corrections replay technique. Specifically, mini-batches of states and actions are fed to the human model, which outputs a mini-batch of $\hat{h}$ predictions. This, together with the correspondent mini-batch of feedback from the buffer, is used to update the weights of the human model.

The human model allows to better observe the problem by associating feedback to (s, a) pairs rather than simply associating it to states as happens in D-COACH and which causes the ambiguities inside the buffer. Even with a large replay buffer, old corrections would not be contradictory to newer ones because they might be corrections for the same states but for different actions.

Complete Algorithm

Algorithm 4 presents the pseudocode of BD-COACH. After setting the required hyper-parameters (line 1), the agent starts interacting with the environment. In the time steps when the teacher provides a correction (line 7), that correction is appended to the buffer $\mathcal{D}$ together with the corresponding state and action. Then, the label a_t^{target} is computed and used for the *single update the policy* (line 11), see red loop of Figure 3.1. Next, the human model is updated with a mini-batch uniformly sampled from the replay buffer $\mathcal{D}$ (line 14) which corresponds to the *batch update of the human model*, see blue loop of Figure 3.1. Then, the human model generates corrections that are used to perform the *batch update of the policy* (line 17) which corresponds to the green loop of Figure 3.1. Lines 14-17 happen not only on the time steps when the human provides a correction but also every b time steps to better leverage the data already collected and minimize overfitting as well as catastrophic forgetting of the neural network.

Algorithm 4 BD-COACH

1: **Require:** error magnitude e, buffer update interval b
2: **Init:** $\mathcal{D} = [\ \ ]$ # *initialize memory buffer*
3: **for** t = 1,2,... **do**
4: **observe** state s_t
5: **execute** action $a_t = \pi(s_t)$
6: **feedback** human corrective advice h_t
7: **if** h_t is not 0 **then**
8: **append** (s_t, a_t, h_t) to $\mathcal{D}$
9: $\text{error}_t = h_t \cdot e$
10: $a_{\text{target}(t)} = a_t + \text{error}_t$
11: **update** $\pi(s_t)$ using SGD with pair (s_t, a_t^{target})
12: **end if**
13: **if** h_t not 0 **or** mod(t, b) is 0 **then**
14: **update** H using SGD with a mini-batch uniformly sampled from $\mathcal{D}$
15: $a = \pi(s)$ # *where s is a mini-batch of states uniformly sampled from $\mathcal{D}$*
16: $\hat{h} = H(s, a)$
17: **update** π using SGD with a and $\hat{h}$
18: **end if**
19: **end for**

3.3. Discussion

This chapter has explained in detail the proposed method BD-COACH. This new framework is similar to the state-of-the-art algorithm D-COACH and it can be used by non-expert human teachers to accelerate the learning of an agent when facing a new task that requires many corrections. Through corrective feedback signals, the teacher modifies the actions taken by the agent if needed. We saw that the main difference between BD-COACH and D-COACH lies in the dependency of the batch updates of the policy. Whereas the batch updates of D-COACH are policy independent, the batch updates in BD-COACH depend on what the newest version of the policy does. This is achieved by incorporating a human model module that learns to predict feedback that is useful for updating the current version of the policy.

4

Experimental Setting

This chapter provides information about the experimental setting for the validation of the proposed method Batch Deep COACH (BD-COACH). Experiments are done both in simulation with a simulated teacher and in a real setup with a human teacher.

4.1. Meta-World Benchmark

To evaluate BD-COACH and compare its performance to the base method D-COACH, we use three simulated environments from the open-source benchmark Meta-World [65]. Meta-World provides 50 standardized manipulation tasks that employ a simulated Sawyer robot arm. All Meta-World tasks are implemented in the MuJoCo physics engine [66], and they are interfaced with OpenAI Gym [67] making these environments easy to use. Even if this benchmark was specifically designed for multi-task and meta-reinforcement learning, it is possible to simply access single goal environments as we do for this work. The three chosen tasks are plate-slide-v2, drawer-open-v2 and button-press-topdown-v2. There are several reasons behind the selection of these tasks. First, for each of them, Meta-World includes an oracle policy that commands the best action for the agent to take at every given time step. The usage of these policies as oracles allows running automated experiments easily, more information in Subsection 4.1.5. Secondly, these tasks use a manipulator similar to the real one that we have available and which we use for the validation in a real setup. Furthermore, from all the tasks that the benchmark provides, we choose three of them that do not involve opening and closing the gripper of the Sawyer's end effector. The reason behind this decision is that we focus on learning continuous actions, but the state of the gripper is either open or close. Finally, the three tasks are quite different between them, which allows generalizing the performance of BD-COACH.

All the tasks in Meta-World need an agent that executes an action in the environment equal to $[\delta x, \delta y, \delta z, g]$. The first three dimensions of the action correspond to the change in position of the end effector in the three Cartesian axes. The last dimension represents the gripper effort that keeps the fingers of the end effector open or close. In our case, for this dimension, the expert policy always commands a constant value keeping the gripper open or close depending on the task. The observation space is a 9 dimensional space formed by the 3D Cartesian positions of the end effector, the object and the goal. In the case of the plate-slide-v2 task, the object refers to the puck, for the drawer-open-v2 task, the object is the handle and finally, in the task button-press-topdown-v2, the object is the button that can be moved vertically.

The Meta-World benchmark defines a success metric as the evaluation criterion for their tasks. This metric, $\|\text{object} - \text{goal}\|_2 < \epsilon$, is based on the euclidean distance between the object position and the goal position where ϵ is a small distance threshold that varies from task to task.

4.1.1. Simulated Experiments

The goal of the simulated experiments is to compare the performance between BD-COACH and D-COACH as a function of the amount of data required to solve a task. To achieve this, we design an experiment where we first use states with relative positions and afterwards, we repeat the experiments changing to absolute positions. The reason behind this design choice is that states formed by relative positions, $s = [[xyz_{\text{end effector}}], [xyz_{\text{object}}], [xyz_{\text{goal}}]]$, make it easier for the robot to generalize as the number of dimensions decreases. Furthermore, relative positions are more informative, and thus the model has to learn fewer correlations. On the other hand, states formed by absolute positions, $s = [[xyz_{\text{object}} - xyz_{\text{end effector}}], [xyz_{\text{goal}} - xyz_{\text{object}}]]$ make the task harder to learn as the number of dimensions increases.

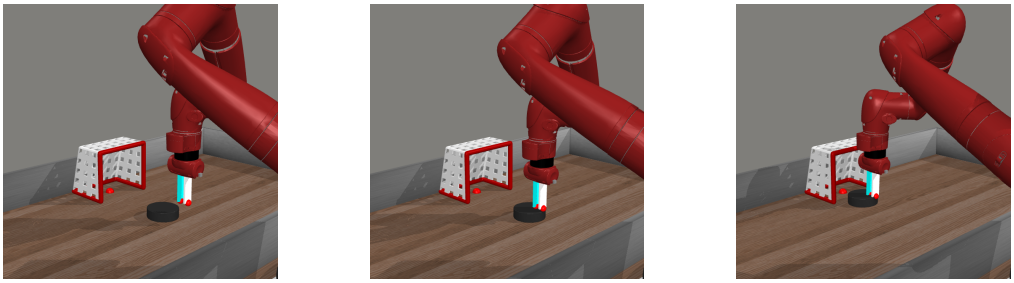
The simulated experiments are therefore divided into two phases. First, with observations formed by relative positions, different values of the parameters error magnitude, e, and replay buffer size, K, are tested to see how robust are BD-COACH and D-COACH to these variations. Specifically, we try three different values of e and three values of K, each time fixing one of them and varying the other to see its effect. For the values of e, we chose 0.01, 0.1 and 1 as these values are within a range previously known to work. The three buffer sizes K are 3000, 15000 and 30000; these sizes correspond to the 10%, 50% and 100% respectively of the average accumulated corrections given by the simulated teacher at the end of the training. Once the previous experiments are done, we choose the best combination of parameters e and K for each method, and then the experiments are repeated with those same values but with observations formed by absolute positions. The parameters used for the policy and the human model are shown in Table 4.1. Trainings are 75000 time steps long which is approximately equivalent to 15 minutes of simulated time as the time step duration of the simulation is 0.0125 seconds.

Parameter	**Value**
Policy hidden layer sizes	(256, 128)
Policy activation of hidden layers	(tanh, tanh)
Policy learning rate	0.001
Human model hidden layer sizes	(512, 512)
Human model activation of hidden layers	(tanh, tanh)
Human model learning rate	0.001
Buffer sampling rate	10
Buffer sampling size	20

Table 4.1: Parameters used for the simulated experiments

4.1.2. Task plate-slide-v2

The task plate-slide-v2, which resembles a hockey movement, can be seen in Figure 4.1. The gripper of the end effector has to approach the puck, press it vertically and then slide the puck to the goal. If the puck goes inside the goal in less than 500 time steps, the episode is considered successful and a fail otherwise. The task starts with the gripper and the object always initiated at the same position whereas the goal is initiated randomly within an area of $0.01m^2$. During all the episode the oracle commands the gripper to remain closed. The success metric that Meta-World defines for this task is $\|\text{object} - \text{goal}\|_2 < 0.07$.



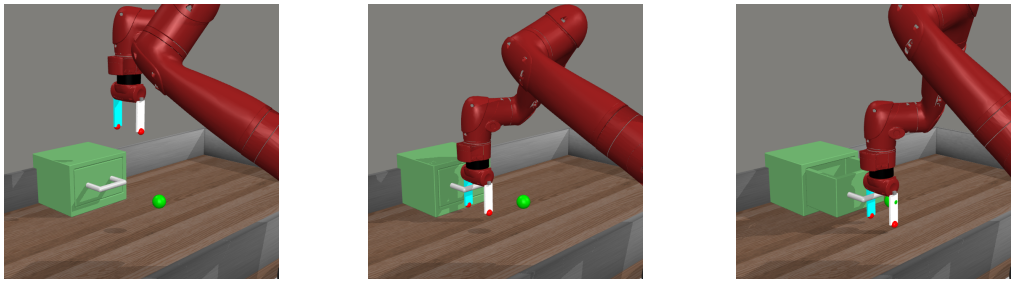
1 - Approach to puck 2 - Press puck vertically 3 - Slide puck to goal

Figure 4.1: Sequence of the plate-slide-v2 task in Meta-World

.

4.1.3. Task drawer-open-v2

The second simulated Meta-World task is called drawer-open-v2, see Figure 4.2. For an episode to be considered a success, the end effector has to approach the handle of the drawer which is always initiated closed, hook to it and pull to open it in less that 500 time steps. For this task, and following the oracle's command, the gripper remains open during all the episodes. The end effector is initiated always at the same position and the drawer can appear at a random position within a length of $0.2m$. The success metric for this task is $\|\text{object} - \text{goal}\|_2 < 0.03$.



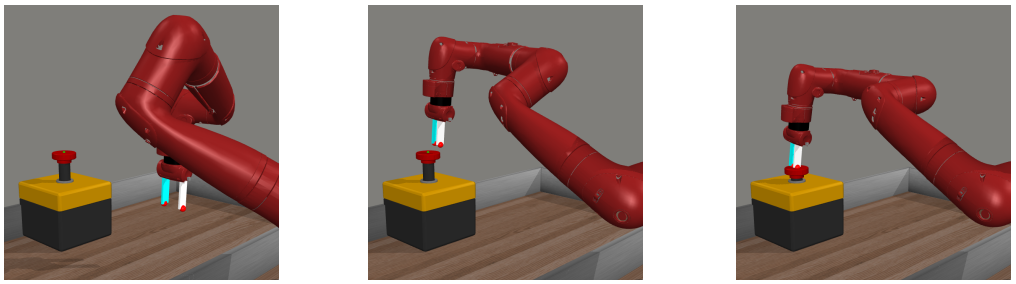
1 - Approach to handle 2 - Hook handle 3 - Open drawer

Figure 4.2: Sequence of the drawer-open-v2 task in Meta-World

.

4.1.4. Task button-press-topdown-v2

The final simulated Meta-World task is called button-press-topdown-v2, see Figure 4.3. For an episode to be considered successful, the end effector has to reach the button, which is placed perpendicular to the surface of the table, and press it vertically in less that 500 time steps. In this task, the gripper remains closed during all the episodes. The end effector is initiated always at the same position whereas the button is initiated randomly within an area of $0.02m^2$. The success metric for this task is $\|\text{object} - \text{goal}\|_2 < 0.02$.



1 - Approach to button 2 - Reach button 3 - Press button vertically

Figure 4.3: Sequence of the button-press-topdown-v2 task in Meta-World

.

4.1.5. Synthesised Feedback

For the simulated experiments, we employed an oracle or synthesized human teacher to provide feedback corrections. Using an oracle removes human factors such as the human teacher providing inconsistent feedback or he/she getting tired which would make comparisons between algorithms unfair. Furthermore, in order to compare performances of different frameworks, it is necessary to run many simulations to obtain a good average of the results which would be completely impractical if the teacher was a real human. Oracles can provide erroneous corrections to better simulate the behaviour of a person but in our case, the oracle gives always the best action for a particular state. For each of the three tasks explained before, we use the high-performance policies that the Meta-World benchmark provides for each of its tasks. The oracle used in this work generates feedback by computing $h = \text{sign}(a_{\text{teacher}} - a_{\text{agent}})$, whereas the decision on whether to provide feedback at each time step is given by the probability $P_h = \alpha \cdot \exp(-\tau \cdot \text{timestep})$, where $\{\alpha \in \mathbb{R} \,|0 \leq \alpha \leq 1\}$; $\{\tau \in \mathbb{R} \,|0 \leq \tau\}$. Furthermore, this binary feedback h is only provided if the difference between the action of the policy and the action of the teacher is larger than a threshold ϵ.

4.2. Experiments with KUKA Robot Arm

In order to validate the new proposed method BD-COACH on a real robotic setup with real human teachers, we devised two tasks involving a KUKA LBR iiwa 7 robot arm pushing a box placed on top of a table. These two tasks are explained in more detail in the following subsections. Several reflecting markers were attached to the box so its pose could be tracked by an OptiTrack motion capture system. The pose, captured by the eight cameras of the available OptiTrack system, consists of the position and orientation of the central point on the box created by the reflecting markers. The human that supervises the learning process conveys the corrections with a joystick. To make the task easier to teach, the human provides corrections just in the 2 axis parallel to the surface of the table while the vertical position of the end effector is kept constant and close to the surface of the table. Prior to performing the real experiments, everything was checked in a simulated Gazebo environment.

A ROS network connects all the required elements for the experiment, these being: The joystick with which corrections are sent, the information with the pose of the box from the OptiTrack and the position of the end effector of the KUKA robot. Specifically, for retrieving the position and sending the commands to the robot arm, the iiwa ROS stack [68] is used. For the control of the arm, we choose the *CustomControllers* from the iiwa stack as it allows the control of the end effector in the Cartesian space. The control command is 6 dimensional where the first 3 values are the orientation of the end effector and the last 3 values are its position. For our purposes, we fix the orientation and the position in the vertical axis. The requested command in the remaining 2 positions is the current position of the end effector plus a delta which is the output of the network. Therefore, the policy outputs 2-dimensional actions representing the relative changes to the positions in the axes parallel to the table.

Regarding the observation that is fed to the policy, it is a 7-dimensional array whose components are: 1. The relative position between the end effector and the box in the x-axis (the axis parallel to the length of table), 2. The relative position between the end effector and the box in the z-axis (the axis parallel to the width of the table), 3. Velocity of the end effector in the x-axis, 4. Velocity of the end effector in the z-axis, 5. Position of the box in the x axis, 6. Position of the box in the z-axis, 7. Orientation of the box (the pitch angle perpendicular to the surface of the table). Finally, a simple metric of success is implemented. This metric has a value of 1 if the box ends within a small radius from the target position and 0 otherwise.

4.2.1. Task kuka-push-box

For the first task called kuka-push-box, the KUKA arm has to push a box placed on one side of the table to a goal position indicated with a green cross. This task, that may seem trivial at first glance, is known as the *pusher-slider system* [69] and it really highlights the importance of reactive robots. It is challenging because it involves an unstable system. Pushing an box in a straight line without a reactive robot is simply impossible to achieve as the box will naturally fall outside the desired straight trajectory. Figure 4.4 shows this problem where a constant velocity is commanded to the end effector but, because the robot does not react to the misalignments, the box keeps deviating from the desired straight trajectory.

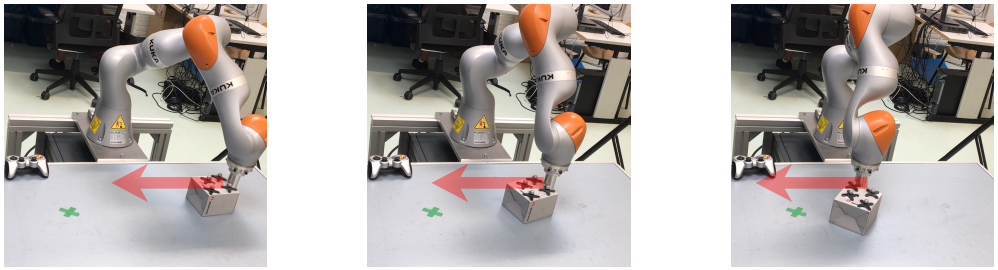
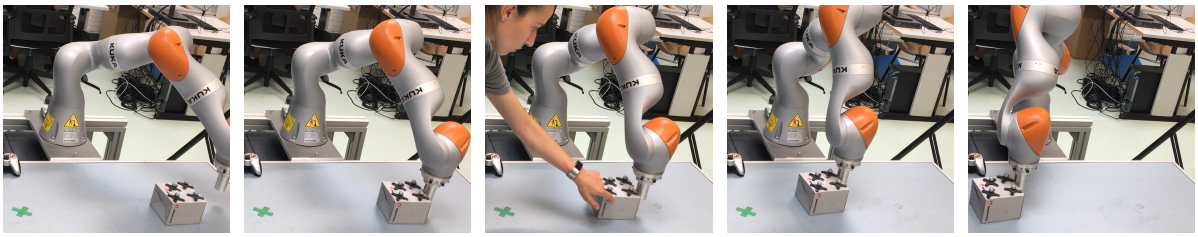


Figure 4.4: By applying a constant velocity (red arrow), the box gets easily misaligned.

Figure 4.5 shows a sequence of the task push-box once the BD-COACH has learnt it. The box is initialized at random positions and disturbances are manually introduced. The agent is able to successfully reach the goal by learning a continuous back and forth motion that makes it possible to quickly correct the misalignments of the box when it starts changing its orientation.



1 - Start position | 2 - Approach to box | 3 - Introduce disturbance | 4 - Correct disturbance | 5 - End position

Figure 4.5: Sequence of the task kuka-push-box

.

4.2.2. Task kuka-park-box

The second task also involves pushing a box but this time the sequence of movements is more complex. To achieve the goal the robot has to push in two different axes. First, it has to push the box until a point where it faces the two brown boxes (third frame of Figure 4.6). Then, the end effector moves around the corner of the pink box and starts pushing from the other side until the box is fit between the 2 walls. Figure 4.6 shows the sequence of movements for this task.

1 - Start position | 2 - Push box | 3 - Center box | 4 - Move around corner | 5- Push box between walls | 6 - End position

Figure 4.6: Sequence of the task kuka-park-box

.

5

Results

This chapter provides the results of the performed experiments. The first section focuses on the results obtained in a simulated environment where three tasks from the Meta-World benchmark [65] are taught. Here, the contribution of BD-COACH is measured by comparing its performance to that of D-COACH for different conditions of replay buffer size K, error magnitude e and type of state (absolute or relative positions). Finally, the results of two planar manipulation tasks are displayed to validate BD-COACH on a real setup.

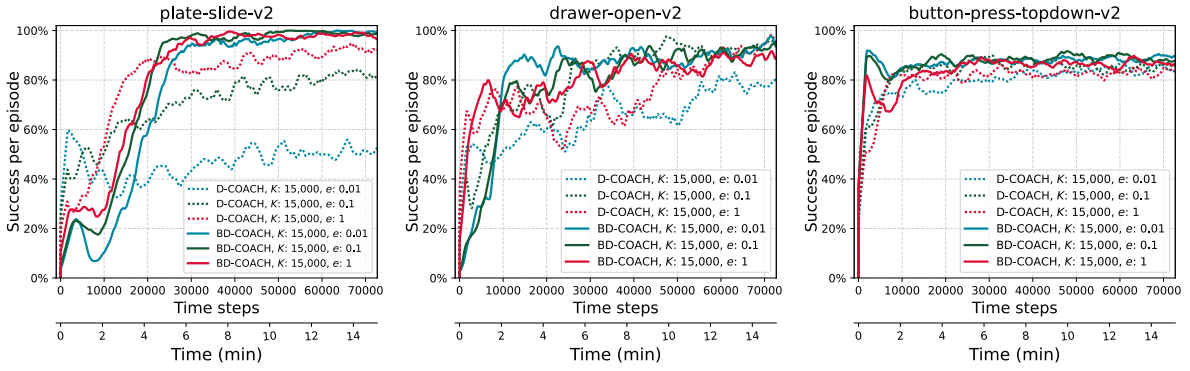
5.1. Results of Simulated Tasks

To evaluate the contribution of BD-COACH with respect to the baseline method D-COACH, we did experiments with a simulated teacher on the Meta-World tasks plate-slide-v2, drawer-open-v2 and button-press-topdown-v2 presented in Chapter 4. All the plots shown below are the result of taking the average of running 20 trainings where each policy is evaluated five times. Figures 5.1 and 5.2 correspond to the first phase of the simulated experiments where using observations formed of relative positions, we analyze the influence of the parameters buffer size K and error magnitude e. Then, in the second phase of the experiments, we compare the best performances of BD-COACH and D-COACH with respect to the same conditions of e and K that resulted in those performances, but this time with observations formed of absolute positions. These results are shown in Figure 5.3.

In the first phase of the experiments, the hyper-parameter e has a higher impact on the performance of D-COACH than K, which is especially true for the task plate-slide-v2. The dotted plots of Figure 5.1a clearly show that a smaller error magnitude e leads to the worst success rate, which is a known problem in D-COACH. As corrections get smaller with a decreasing error magnitude, the teacher needs to provide more corrections to achieve the desired behaviour, which causes the buffer to fill faster. For the tasks drawer-open-v2 and button-press-topdown-v2, this relationship between success rate and the value of e is not as clear, but still, we can see that D-COACH performs worse with $e = 0.01$.

On the other hand, BD-COACH achieves similar success rates with similar growth slopes for the three tasks independently of e. This behaviour is expected, as with BD-COACH, the corrections kept inside the buffer are unambiguous and therefore, a small error magnitude e is not a problem for this new method. In the case of plate-slide-v2, a small e causes a slightly slow growth at the beginning but afterwards, it achieves the same convergence of almost 100% as the other values of e. This could be caused by the corrections being too small and therefore needing more feedback to learn the task. The same happens in drawer-open-v2 where for BD-COACH, the highest e is beneficial at the beginning but afterwards the three values achieve similar success rates. Furthermore, something interesting to remark about this task is the behaviour of the oracle whose plot can be seen in Figure B.1b of Annex B. This task is the only one that shows that BD-COACH is more efficient with the feedback as its slope decreases faster. This fact indicates that on certain time steps, the difference between the agent's actions and the oracle's actions is smaller than the oracle's threshold ϵ and therefore it does not require feedback for those

time steps. For the button-press-topdown-v2 task is harder to extract conclusions because the three values of e result in very similar success percentages for BD-COACH. Finally, these experiments provide a good starting point for selecting the value of e for the real setup experiments which will be within the range $[0.1, 1]$.

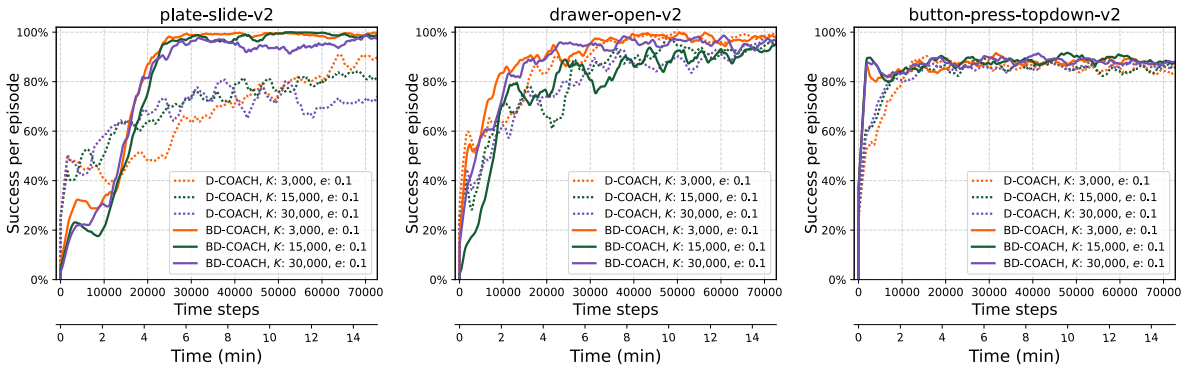


(a) plate-slide-v2 (b) drawer-open-v2 (c) button-press-topdown-v2

Figure 5.1: Results of success rates obtained for a fixed buffer size K and different values of the error magnitude e using relative positions. Simulated teacher: $P_h : \alpha = 0.9; \tau = 0.000015$

Variation in buffer size K, see Figure 5.2, has less pronounced effects than the previous variation of the error magnitude e. For D-COACH, the three K values in plate-slide-v2 present similar slopes and success heights, but it is true that at the end of the training, the smaller buffer gets the highest success rate, followed by the medium buffer and finally the biggest one. This behaviour is also displayed in drawer-open-v2, although less clearly, around time step 50000. This fact fits with the need of D-COACH to require a small buffer to minimize the contradictions between the saved data. Finally, for button-press-topdown-v2, we cannot appreciate the effects of K and D-COACH achieves similar but slightly lower results than BD-COACH.

On the other hand, when analyzing BD-COACH, we cannot find a clear relationship between K and success rate that is true for every task. For example, in plate-slide-v2, the largest buffer has a negative effect between time steps 40000 and 60000 but then it recovers and goes almost to 100% of success rate. However, for drawer-open-v2, the medium buffer is the one that presents the slowest growth slope, which keeps growing until the end of the training. In the case of button-press-topdown-v2, the three buffer sizes behave very similarly.

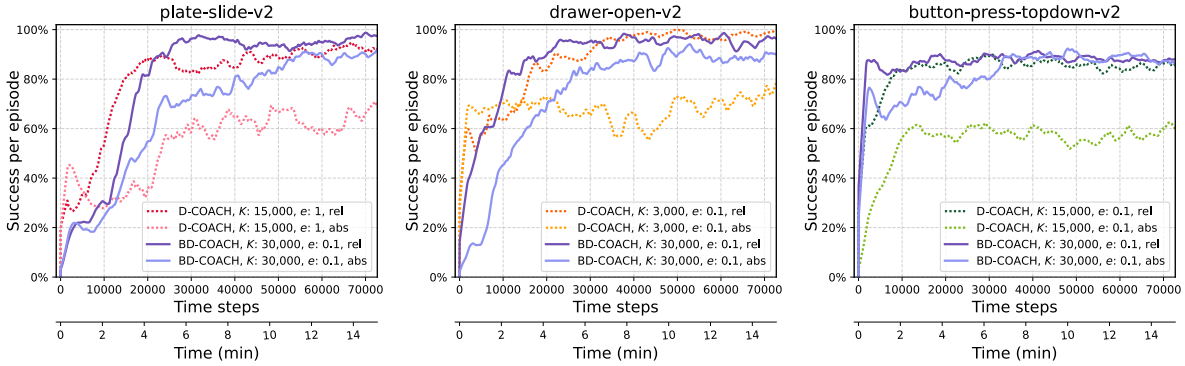


(a) plate-slide-v2 (b) drawer-open-v2 (c) button-press-topdown-v2

Figure 5.2: Results of success rates obtained for a fixed value of the magnitude error e and different values of buffer sizes K using relative positions. Simulated teacher: $P_h : \alpha = 0.9; \tau = 0.000015$

We can best appreciate the contribution of BD-COACH in the second phase of the experiments, where BD-COACH far outperforms D-COACH. Figure 5.3 compares the effect of using relative positions in the observation versus absolute positions, which increases the dimensionality of the observation. For the tasks plate-slide-v2 and drawer-open-v2, Figures 5.3a and 5.3b respectively, BD-COACH is not able to reach the rates of success obtained

with relative positions getting around a 5% less of success. Still, BD-COACH performs better than D-COACH for which the three tasks result in a decrease of success around 20%. This behaviour was expected due to the increase in the dimensionality of the state-space. A higher dimensionality causes the agents to need more feedback to correct their actions in all the possible states, which causes the buffer to fill faster and, in the case of D-COACH, with contradictory data.



(a) plate-slide-v2 (b) drawer-open-v2 (c) button-press-topdown-v2

Figure 5.3: Results of success rates obtained when comparing the best performance of each method using relative positions, against their performances when using absolute positions for the same conditions of K and e. Simulated teacher: $P_h : \alpha = 0.9; \tau = 0.000015$

5.2. Results of Validation in Real System

To validate the proposed BD-COACH algorithm in a real system, we taught the two planar manipulation tasks described in Sections 4.2.2 and 4.2.1 with a human teacher who provides the necessary corrections with a joystick. In this subsection, we show the results obtained for both tasks and because the objective is simply to validate BD-COACH, we do not compare with other methods. A video of the learnt behavior is available at: `https://youtu.be/aWCvxShtHk4`.

5.2.1. Results for Task kuka-push-box

Figure 5.4 shows the average success per episode of task kuka-push-box achieved with a BD-COACH agent after learning for 60 minutes where 165 episodes were performed. Prior to the final training, some trials are done to adjust the error magnitude e, which is found to work best with a value of 0.5. The size of the replay buffer, K, is set to 10000, which never gets completely full because the human sends around 7300 correction signals. The learning is evaluated every five episodes, and every episode was, on average, 20 seconds long. At minute 23 approximately, the slope of the curve starts increasing until minute 35 when the success percentage reaches an average of 80%.

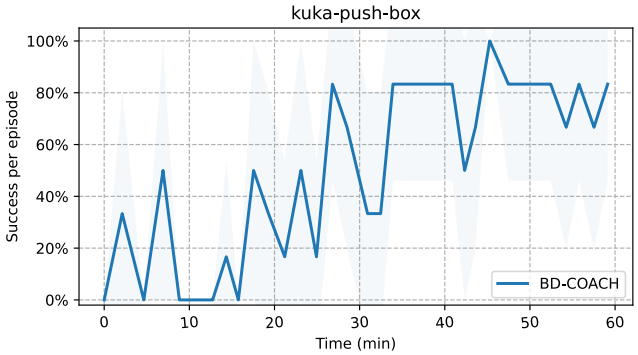


Figure 5.4: Results of the task kuka-push-box with a human teacher.

5.2.2. Results for Task kuka-park-box

The results for kuka-park-box in Figure 5.5 show the average success per episode of BD-COACH after a training of 40 minutes with a human teacher where 65 episodes were performed. Again, prior to the final training, the value of the error magnitude is adjusted, in this case, to $e = 0.3$. The teacher sends an average of 3400 corrections which never fill the buffer whose K is set to 10000. The learning is evaluated every five episodes, and every episode is on average 36 seconds long. Around minute 25 the performance increases drastically. This point in time corresponds with the moment that the BD-COACH agent learns to go around the corner (see the third frame of Figure 4.6). Prior to that moment, all episodes logically fail. In episode 37, the performance reaches its maximum as all the following episodes are successful.

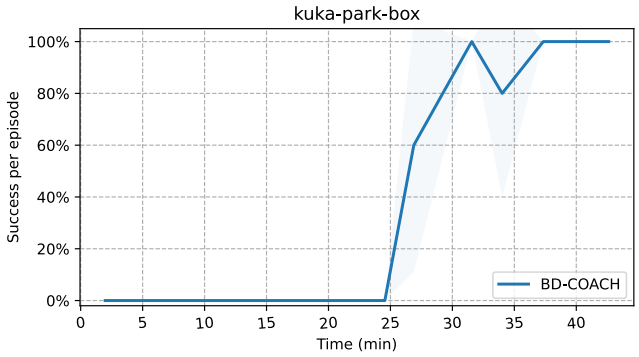


Figure 5.5: Results of the task kuka-park-box with a human teacher.

6

Conclusion

Corrective imitation learning (CIL) encompasses interactive imitation learning frameworks that use corrective feedback to improve the behaviour of an agent during its training. CIL methods have the advantage that can be employed by non-expert teachers as is the case of the framework Deep COrrective Advice Communicated by Humans, D-COACH [15]. D-COACH uses an artificial neural network as a function approximator for the policy of its agent and applies the corrections replay technique to minimize the negative effect of catastrophic forgetting and overfitting. However, due to how this technique is implemented, the past corrections that are replayed during the batch updates are ambiguous because they do not depend on what the policy is doing and can be detrimental to the performance of the policy. This fact forces D-COACH to reduce the size of its buffer which results in a limitation of the type of problems that it can solve. If tasks are too complex, e.g., long-horizon tasks or tasks with high dimensionality, D-COACH is not able to learn them.

To address this issue, we present Batch Deep COACH (BD-COACH). This new method preserves the structure of D-COACH but implements a new module, the human model, that learns to predict the feedback given by the human. By using this human model, the past corrections that are replayed during the batch updates of the policy now depend on the actions of the current policy. This fact makes BD-COACH able to really take advantage of the corrections replay technique.

D-COACH has the inconvenience that, prior to an experiment, it is necessary to adjust the error magnitude e depending on the task. As shown in the results of Chapter 5, variations of e do not affect BD-COACH which facilitates experiments. Regarding the size of the buffer K, which is another important parameter to adjust in D-COACH, it also does not seem to affect the performance of BD-COACH, proving that there are no ambiguities in the data inside. The last and most interesting conclusion is that, when the task requires more data to train, Figure 5.3, BD-COACH shows much better performance than D-COACH.

However, BD-COACH has its limitations. Computationally speaking, it is more expensive than D-COACH as two artificial neural networks are learnt in parallel. With BD-COACH, performance is minimally affected at the beginning of the training when the human model has not learnt yet to predict suitable feedback. This can be appreciated in the results for the task plate-slide-v2. Finally, regarding data efficiency, BD-COACH is faster than RL techniques as we are leveraging human knowledge, but when compared to D-COACH, it does not show an improvement in this aspect. It is true that for the task drawer-open-v2, the feedback plots for BD-COACH in Annex B show an oracle that is less requested but, for the rest of the tasks, BD-COACH is as data efficient as D-COACH. This fact implies that, if the problem is too complex, it could take a long time to train.

We can conclude by saying that BD-COACH solves the challenge of D-COACH of making corrections gathered by older versions of the policy still useful for batch updating the current version of the policy.

Future Recommendations

This work could be extended in future research in the following directions:

- **More exhaustive experiments**. BD-COACH has successfully demonstrated in simulation to be able to benefit from the corrections replay technique. However, it would be necessary to run exhaustive experiments with more human participants to take into account human factors that we have not considered and really prove the benefits of BD-COACH.

- **Use images as observations**. BD-COACH has been validated only with observations formed by Cartesian positions. It would be very interesting to see if it is able to keep its performance when the policy is fed with observations form by images.

- **Validation with longer-horizon tasks**. Finally, more complex tasks could be taught to BD-COACH to see to what extent it can leverage the most from the replay buffer.

References

[1] L. Nardi and C. Stachniss, "Experience-based path planning for mobile robots exploiting user preferences," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 1170–1176. DOI: 10.1109/IROS.2016.7759197.

[2] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st. Cambridge, MA, USA: MIT Press, 1998, ISBN: 0262193981.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013, NIPS Deep Learning Workshop 2013. [Online]. Available: http://arxiv.org/abs/1312.5602.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, ISSN: 0028-0836. DOI: 10.1038/nature16961.

[5] O. M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020. DOI: 10.1177/0278364919887447. eprint: https://doi.org/10.1177/0278364919887447. [Online]. Available: https://doi.org/10.1177/0278364919887447.

[6] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013. DOI: 10.1177/0278364913495721.

[7] W. B. Knox and P. Stone, "Interactively shaping agents via human reinforcement: The tamer framework," in *Proceedings of the Fifth International Conference on Knowledge Capture*, ser. K-CAP '09, Redondo Beach, California, USA: Association for Computing Machinery, 2009, pp. 9–16, ISBN: 9781605586588. DOI: 10.1145/1597735.1597738.

[8] I. Kostrikov, O. Nachum, and J. Tompson, "Imitation learning via off-policy distribution matching," *arXiv preprint:1912.05032*, 2019.

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

[10] R. Hoque, A. Balakrishna, C. Putterman, M. Luo, D. S. Brown, D. Seita, B. Thananjeyan, E. Novoseller, and K. Goldberg, "Lazydagger: Reducing context switching in interactive imitation learning," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021, pp. 502–509. DOI: 10.1109/CASE49439.2021.9551469.

[11] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *Journal of Artificial Intelligence Research*, vol. 34, pp. 1–25, 2009.

[12] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei, "Reward learning from human preferences and demonstrations in atari," in *NeurIPS*, 2018.

[13] C. Celemin and J. Ruiz-del-Solar, "Coach: Learning continuous actions from corrective advice communicated by humans," in *2015 International Conference on Advanced Robotics (ICAR)*, 2015, pp. 581–586. DOI: 10.1109/ICAR.2015.7251514.

[14] C. Celemin and J. Ruiz-Del-Solar, "An interactive framework for learning continuous actions policies based on corrective feedback," *J. Intell. Robotics Syst.*, vol. 95, no. 1, pp. 77–97, Jul. 2019, ISSN: 0921-0296. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1007/s10846-018-0839-z.

[15] R. Pérez-Dattari, C. Celemin, J. Ruiz-del-Solar, and J. Kober, "Interactive learning with corrective feedback for policies based on deep neural networks," in *International Symposium on Experimental Robotics*, Springer, 2018, pp. 353–363.

[16] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *arXiv preprint:1712.01275*, 2018.

[17] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, *An Algorithmic Perspective on Imitation Learning*. 2018. DOI: 10.1561/2300000053.

[18] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

[19] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Cambridge University Engineering Department, CUED/F-INFENG/TR 166, Sep. 1994. [Online]. Available: ftp://svr-ftp.eng.cam.ac.uk/reports/rummery_tr166.ps.Z.

[20] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[21] Q. Wu, H. Wu, X. Zhou, M. Tan, Y. Xu, Y. Yan, and T. Hao, "Online transfer learning with multiple homogeneous or heterogeneous sources," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 7, pp. 1494–1507, 2017. DOI: 10.1109/TKDE.2017.2685597.

[22] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, "Experience replay for continual learning," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/fa7cdfad1a5aaf8370ebeda47a1ff1c3-Paper.pdf.

[23] L. Deng and D. Yu, "Deep learning: Methods and applications," *Foundations and trends in signal processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[24] D. Graupe, *Principles Of Artificial Neural Networks (3rd Edition)*, ser. Advanced Series In Circuits And Systems. World Scientific Publishing Company, 2013, ISBN: 9789814522755. [Online]. Available: https://books.google.es/books?id=Zz27CgAAQBAJ.

[25] F. Torabi, G. Warnell, and P. Stone, "Recent advances in imitation learning from observation," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2019, pp. 6325–6331. [Online]. Available: https://doi.org/10.24963/ijcai.2019/882.

[26] A. Najar and M. Chetouani, "Reinforcement learning with human advice: A survey," *Frontiers in Robotics and AI*, vol. 8, p. 74, 2021, ISSN: 2296-9144. [Online]. Available: https://www.frontiersin.org/article/10.3389/frobt.2021.584075.

[27] M. Laskey, J. Lee, W. Y.-S. Hsieh, R. Liaw, J. Mahler, R. Fox, and K. Goldberg, "Iterative noise injection for scalable imitation learning," *arXiv preprint:1703.09327v1*, 2017.

[28] A. Balakrishna, B. Thananjeyan, J. Lee, F. Li, A. Zahed, J. E. Gonzalez, and K. Goldberg, "On-policy robot imitation learning from a converging supervisor," in *Proceedings of the Conference on Robot Learning*, vol. 100, PMLR, 30 Oct–01 Nov 2020, pp. 24–41. [Online]. Available: https://proceedings.mlr.press/v100/balakrishna20a.html.

[29] J. N. Lee, M. Laskey, A. K. Tanwani, A. Aswani, and K. Goldberg, "Dynamic regret convergence analysis and an adaptive regularization algorithm for on-policy robot imitation learning," *The International Journal of Robotics Research*, vol. 40, no. 10-11, pp. 1284–1305, 2021.

[30] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[31] J. D. Chang, M. Uehara, D. Sreenivas, R. Kidambi, and W. Sun, "Mitigating covariate shift in imitation learning via offline data without great coverage," *arXiv preprint:2106.03207*, 2021.

[32] M. Laskey, S. Staszak, W. Y. Hsieh, J. Mahler, F. T. Pokorny, A. D. Dragan, and K. Goldberg, "Shiv: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 462–469. DOI: 10.1109/ICRA.2016.7487167.

[33] R. Loftin, B. Peng, J. Macglashan, M. L. Littman, M. E. Taylor, J. Huang, and D. L. Roberts, "Learning behaviors via human-delivered discrete feedback: Modeling implicit feedback strategies to speed up learning," *Autonomous Agents and Multi-Agent Systems*, vol. 30, no. 1, pp. 30–59, Jan. 2016, ISSN: 1387-2532. DOI: 10.1007/s10458-015-9283-7.

[34] J. MacGlashan, M. K. Ho, R. Loftin, B. Peng, G. Wang, D. L. Roberts, M. E. Taylor, and M. L. Littman, "Interactive learning from policy-dependent human feedback," in *International Conference on Machine Learning*, PMLR, 2017, pp. 2285–2294.

[35] G. Warnell, N. Waytowich, V. Lawhern, and P. Stone, "Deep tamer: Interactive agent shaping in high-dimensional state spaces," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11485.

[36] R. Arakawa, S. Kobayashi, Y. Unno, Y. Tsuboi, and S.-i. Maeda, "Dqn-tamer: Human-in-the-loop reinforcement learning with intractable feedback," *arXiv preprint:1810.11748*, 2018.

[37] C.-A. Cheng, X. Yan, N. Wagener, and B. Boots, "Fast policy learning through imitation and reinforcement," in *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, AUAI Press, 2018, pp. 845–855. [Online]. Available: http://auai.org/uai2018/proceedings/papers/302.pdf.

[38] X. He, H. Chen, and B. An, "Learning behaviors with uncertain human feedback," in *Conference on Uncertainty in Artificial Intelligence*, PMLR, 2020, pp. 131–140.

[39] J. Spencer, S. Choudhury, M. Barnes, M. Schmittle, M. Chiang, P. Ramadge, and S. Srinivasa, "Learning from Interventions: Human-robot interaction as both explicit and implicit feedback," in *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, Jul. 2020. DOI: 10.15607/RSS.2020.XVI.055.

[40] H. He, H. III, and J. Eisner, "Imitation learning by coaching," *Advances in Neural Information Processing Systems*, vol. 4, pp. 3149–3157, Jan. 2012.

[41] S. Griffith, K. Subramanian, J. Scholz, C. Isbell, and A. Thomaz, "Policy shaping: Integrating human feedback with reinforcement learning," *Advances in Neural Information Processing Systems*, Jan. 2013.

[42] S. Ross and J. A. Bagnell, "Reinforcement and imitation learning via interactive no-regret learning," *arXiv preprint:1406.5979*, 2014.

[43] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end simulated driving," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.

[44] K. Menda, K. Driggs-Campbell, and M. J. Kochenderfer, "Dropoutdagger: A bayesian approach to safe imitation learning," *arXiv preprint:1709.06166*, 2017.

[45] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell, "Deeply aggrevated: Differentiable imitation learning for sequential prediction," in *International Conference on Machine Learning*, PMLR, 2017, pp. 3309–3318.

[46] H. Le, N. Jiang, A. Agarwal, M. Dudík, Y. Yue, and H. Daumé, "Hierarchical imitation and reinforcement learning," in *International conference on machine learning*, PMLR, 2018, pp. 2917–2926.

[47] C. Cronrath, E. Jorge, J. Moberg, M. Jirstrand, and B. Lennartson, "Bagger: A bayesian algorithm for safe and query-efficient imitation learning," in *Machine Learning in Robot Motion Planning–IROS 2018 Workshop*, 2018.

[48] N. R. Waytowich, V. G. Goecks, and V. J. Lawhern, "Cycle-of-learning for autonomous systems from human interaction," *arXiv preprint:1808.09572*, 2018.

[49] J. Song, R. Lanka, A. Zhao, A. Bhatnagar, Y. Yue, and M. Ono, "Learning to search via retrospective imitation," *arXiv preprint:1804.00846*, 2018.

[50] B. Xiong, F. Wang, C. Yu, X. Liu, F. Qiao, Y. Yang, and Q. Wei, "Learning safety-aware policy with imitation learning for context-adaptive navigation," *CEUR Workshop Proceedings*, 2019.

[51] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, "Hg-dagger: Interactive imitation learning with human experts," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 8077–8083.

[52] K. Menda, K. Driggs-Campbell, and M. J. Kochenderfer, "Ensembledagger: A bayesian approach to safe imitation learning," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 5041–5048.

[53] A. Prakash, A. Behl, E. Ohn-Bar, K. Chitta, and A. Geiger, "Exploring data aggregation in policy learning for vision-based urban autonomous driving," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 760–11 770. DOI: 10.1109/CVPR42600.2020.01178.

[54] B. Xiao, Q. Lu, B. Ramasubramanian, A. Clark, L. Bushnell, and R. Poovendran, "Fresh: Interactive reward shaping in high-dimensional state spaces using human feedback," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '20, 2020, pp. 1512–1520, ISBN: 9781450375184.

[55] T. Ablett, F. Marić, and J. Kelly, "Fighting failures with fire: Failure identification to reduce expert burden in intervention-based learning," *arXiv preprint:2007.00245*, 2020.

[56] A. Mandlekar, D. Xu, R. Martín-Martín, Y. Zhu, L. Fei-Fei, and S. Savarese, "Human-in-the-loop imitation learning using remote teleoperation," *arXiv preprint:2012.06733*, 2020.

[57] R. Hoque, A. Balakrishna, E. Novoseller, A. Wilcox, D. S. Brown, and K. Goldberg, "Thriftydagger: Budget-aware novelty and risk gating for interactive imitation learning," *arXiv preprint:2109.08273*, 2021.

[58] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991. DOI: 10.1162/neco.1991.3.1.88.

[59] M. Laskey, J. Lee, R. Fox, A. D. Dragan, and K. Goldberg, "DART: noise injection for robust imitation learning," in *1st Annual Conference on Robot Learning, CoRL 2017*, vol. 78, 2017, pp. 143–156. [Online]. Available: http://proceedings.mlr.press/v78/laskey17a.html.

[60] I. Kostrikov, O. Nachum, and J. Tompson, "Imitation learning via off-policy distribution matching," *arXiv preprint:1912.05032*, 2019.

[61] S. Reddy, A. D. Dragan, and S. Levine, "SQIL: imitation learning via reinforcement learning with sparse rewards," in *8th International Conference on Learning Representations, ICLR 2020*, 2020. [Online]. Available: https://openreview.net/forum?id=S1xKd24twB.

[62] R. Pérez-Dattari, C. Celemin, J. Ruiz-del-Solar, and J. Kober, "Continuous control for high-dimensional state spaces: An interactive learning approach," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 7611–7617. DOI: 10.1109/ICRA.2019.8793675.

[63] D. Wout, J. Scholten, C. Celemin, and J. Kober, "Learning gaussian policies from corrective human feedback," *arXiv preprint:1903.05216*, 2019.

[64] R. Perez-Dattari, C. Celemin, G. Franzese, J. Ruiz-del-Solar, and J. Kober, "Interactive learning of temporal features for control: Shaping policies and state representations from human feedback," *IEEE Robotics Automation Magazine*, vol. 27, no. 2, pp. 46–54, 2020. DOI: 10.1109/MRA.2020.2983649.

[65] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on Robot Learning*, PMLR, 2020, pp. 1094–1100.

[66] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[67] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint:1606.01540*, 2016.

[68] C. Hennersperger, B. Fuerst, S. Virga, O. Zettinig, B. Frisch, T. Neff, and N. Navab, "Towards mri-based autonomous robotic us acquisitions: A first feasibility study," *IEEE transactions on medical imaging*, vol. 36, no. 2, pp. 538–548, 2017.

[69] F. Hogan and A. Rodriguez, "Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics," in *Algorithmic Foundations of Robotics XII*. May 2020, pp. 800–815, ISBN: 978-3-030-43088-7. DOI: 10.1007/978-3-030-43089-4_51.

[70] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, vol. 297, p. 103 500, 2021, ISSN: 0004-3702. DOI: 10.1016/j.artint.2021.103500.

[71] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems*, vol. 30, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf.

[72] M. Laskey, "On and off-policy deep imitation learning for robotics," Ph.D. dissertation, EECS Department, University of California, Berkeley, Aug. 2018. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-108.html.

[73] R. Zhang, F. Torabi, L. Guan, D. H. Ballard, and P. Stone, "Leveraging human guidance for deep reinforcement learning tasks," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2019, pp. 6339–6346. [Online]. Available: https://doi.org/10.24963/ijcai.2019/884.

[74] J. Lee, M. Laskey, A. K. Tanwani, and K. Goldberg, "Stability analysis of on-policy imitation learning algorithms using dynamic regret," in *RSS Workshop on Imitation and Causality*, 2018.

A

List of Imitation Learning algorithms

This annex consists of a list of several imitation learning algorithms that are classified in Table 2.1 in Chapter 2.

Behavioural Cloning

Behavioural cloning (BC) [58], one of the oldest imitation learning frameworks, consists on directly learning a policy, via supervised learning, given a dataset of demonstrations provided by a teacher. The main problem with BC is known as *covariate shift* and it happens because at the moment that the learner agent deviates from the expert trajectory, it cannot recover from that failure and go back to the expert trajectory. This provokes a cascade of errors that keep growing because the agent does not know how to act in those states that have not been visited by the expert [9].

Inverse Reinforcement Learning methods

Inverse Reinforcement Learning (IRL) is two-step approach for learning from humans. First, based on demonstrations provided by the human expert, the agent learns the reward function R that best explains the behaviour of the human. Then, with this reward function, the optimal policy is learnt using RL methods [70].

Learning from human preferences methods

Learning from human preferences, [71], [12], are methods where iteratively, a human decides which of several executions of a policy is better according to the goal of the task. Then, a reward function that explains the decisions of the human is found and with it and applying RL, the agent learns how to perform the task. The user continues deciding between executions, and therefore, the policy gradually improves.

DART

DART (Disturbances for Augmenting Robot Trajectories) [59] is an algorithm close to behavioural cloning [58]. According to its authors, DART is an off-policy imitation learning algorithm because, after the first expert's demonstration, it does not require to query the expert anymore. This method works by injecting noise into the supervisor's policy while demonstrating the task. This makes the agent have to visit a wider region of the state-space where it gets recovery demonstrations, reducing the problem of covariate shift.

SQIL

SQIL (Soft Q Imitation Learning) [61] is a variant of behavioural cloning that incentives the agent to match human demonstrations over a long horizon. SQIL uses reinforcement learning but does not learn a reward function,

instead, when the agent matches a demonstrated action in a demonstrated state, it receives a reward of ”1” and ”0” for every other behaviour.

ValueDICE

ValueDICE (Distribution Correction Estimation) [60] is an off-policy imitation learning [72] that minimize the KL-divergence between the expert distribution and the distribution induced by the agent when interacting with the environment in an off-policy manner.

DAgger

DAgger (Data Aggregation) [30] is one of the most well-known imitation learning algorithms and it was explained in-depth in Section 2.3.2. Many variants of DAgger exist and some of them are presented next.

DAgger by coaching

DAgger by coaching [40] is a version of DAgger [30] where the human teacher executes actions that are within the learner's ability. This means that when the agent is at a state far from the desired state, the teacher will not try to correct directly that difference but instead, it will try to redirect the agent gradually [9].

AggreVaTe

AggreVaTe (Aggregate Values to Imitate) [42] is a version of DAgger [30], that learns to choose actions that minimize the cost-to-go of the expert, [9]. The cost-to-go Q is the cost of executing an action in a state and continuing executing a policy for the next steps. The first iteration is the same as in DAgger, then, for the next iterations, at a time t, the cost-to-go Q of taking an action in a state, is observed and added to the initial dataset D together with the action and the state.

AggreVaTeD

AggreVaTeD [45] is a differentiable version of the AggreVaTe algorithm introduced by [42] that does not perform any data aggregation. It includes two gradient update procedures: An online gradient descent and a natural gradient update similar to exponential gradient descent.

SafeDAgger

SafeDAgger [43] is a version of DAgger [30], that aims to reduce the burden of constantly querying the human teacher by incorporating a safety policy. This safety policy predicts the discrepancy between the teacher and the learner and it is only on states where the discrepancy is high, that the teacher is queried.

DropoutDAgger

DropoutDAgger [44] is a method similar to SafeDAgger [43]. It uses the dropout technique to train the artificial neural network policy applying N random dropout masks. For each random configuration of the network, an action is obtained for the same input observation. Only if its distribution over actions has enough probability mass around the action suggested by the expert, the mean action of the novice is taken, otherwise the action of the expert is chosen. When the distribution of actions has high entropy, it means that that region of the state-space is unfamiliar and therefore, it is safer to choose the action of the expert.

EnsembleDAgger

EnsembleDAgger [52] is an extension of DAgger where the learning agent only acts when two measures are compliant with two preset thresholds: The first measure is the *discrepancy*, which is the same as in SafeDagger

[43]. The discrepancy represents the deviation between the agent and the expert. The second measure is the *doubt* which is the variance that indicates the familiarity of the agent with its current state. The doubt constrains the learner to only act in familiar states.

ThriftyDAgger

ThriftyDAgger [57] is a method similar to SafeDagger [43] and EnsembleDAgger [52] in the sense that it switches the control between the agent and the human supervisor depending on the novelty and the risk of a particular state. ThriftyDAgger proposes a novel metric for measuring the riskiness of a state, which captures the likelihood that the agent cannot succeed in converging to the goal state.

SHIV

SHIV (Svm-based reduction in Human InterVention) [32] is a method similar to DAgger [30] that differs from this one in the fact that the human teacher is not queried to provide labels for all the visited states but only when it is risky. The risk is described as distance to a boundary defined by a one-class support vector machine.

Hierarchical Guidance DAgger

Hierarchical guidance [46] is a framework for hierarchical problems where low level sub-tasks can be identified by an expert. The authors present two settings: In the first one, hierarchical imitation learning, the teacher provides high-level feedback and only provides low-level feedback when needed. The second setting is a hybrid imitation–reinforcement learning where the teacher provides only high-level feedback and the learner uses reinforcement learning at the low level.

BAgger

BAgger (Bayesian dataset Aggregation) [47] is an extension of DAgger [30] that aims to increase safety and reduce expert burden by querying the expert only when there is risk of not being able to imitate the expert. BAgger trains a Bayesian neural network or a Gaussian process to predict the expected error between teacher and learner.

SAIL

SAIL (Safety-Aware Imitation Learning) [50] is an extension of DAgger that aims to reduce the number of queries to the expert. Only when the confidence level of the learning agent is lower than a threshold is the expert queried. The expert continues providing labels until the confidence in all states is again higher than a threshold. Then, an uncertainty based approach decides whether or not continue querying the expert.

Retrospective DAgger

Retrospective DAgger [49] is a version of the DAgger [30] algorithm, designed for combinatorial search spaces. The policy learns from its mistakes by learning from retrospective inspections of its own roll-outs.

HG-DAgger

HG-DAgger (Human-Gated Data Aggregation) [51] is a version of DAgger [30] that includes a gating function controlled by the expert. When the expert teacher detects that the agent is at a unsafe region of the state space, the expert takes control and leads the learning agent to a safe region of the state space.

EIL

EIL (Expert Intervention Learning) [39] is an algorithm similar to HG-Dagger, that allows the teacher to take the control of the agent when needed. In HG-Dagger, the labels that the human provides when taking control (explicit

feedback) are used to update the dataset. On the other hand, EIL not only uses this explicit feedback but also the implicit feedback which are those actions that the human does not correct because the agent is already behaving correctly. Finally, EIL also takes into account the timing, meaning when the feedback was provided.

FIRE

FIRE (Failure Identification to Reduce Expert burden) [55] is an algorithm similar to HG-Dagger that incorporates the ability of notifying the expert when the agent is at an unsafe state. FIRE has a failure predictor that classifies expert and non-expert data and predicts when a failure may occur. When this happens, the policy stops and if the human agrees with the prediction, he/she teleoperates the agent back to a safe state.

IWR

IWR (Intervention Weighted Regression) [56] is an algorithm similar to FIRE, that focuses on *bottlenecks* regions, that is, those states where sequences of precise actions are needed. IWR keeps two different datasets, one with data provided by the human when intervening in a trajectory (mostly during bottleneck regions), and another one for the rest of the trajectory when the human does not intervene. During training, the two data sets are equally sampled which re-weights the data distribution. The goal is to reinforce the labels provided by the human during bottleneck regions, whereas the data sampled from the non-intervention dataset keeps the policy close to previous policy iterations [56] .

DA-RB

DA-RB (DAgger Replay Buffer) [53] is an extension of the DAgger algorithm [30] that incorporate an additional replay buffer with the aim of controlling the proportion of data provided by the human and data gathered by the agent. This buffer is said to help the policy to focus on its weaker behaviours.

COACH

COACH (COrrective Advice Communicated by Humans) [13] is an algorithm designed for non-expert humans where feedback is provided as a binary signal interpreted as an increase or decrease of the value of an action. The feedback is immediately used for updating the policy which makes it easy for the teacher to observe the change in the behaviour of the agent and continue providing further corrections. When a sequence of corrections has the same sign, it indicates that the correction to be made has a large magnitude. Opposite, if the signal alternates signs, the teacher is trying to make smaller corrections around a certain state. Furthermore, COACH includes a Human Feedback Model that helps to interpret and adapt the corrections. This algorithm, and more specifically its deep version [15] are the main focus of this master thesis.

D-COACH

D-COACH (Deep COrrective Advice Communicated by Humans) [15] is the deep learning version of the COACH algorithm [13]. It uses artificial neural networks to represent the policy and includes a buffer to replay recent experiences. This algorithm is the starting point of the present master thesis.

Advise

Advise [41], is a policy shaping approach where the human teacher feedback is interpreted as direct policy labels. An example of feedback could be “this is right” or “this is wrong” given an action taken by the agent. Advise also takes into account that the human feedback can be inconsistent and that correct feedback is provided with a probability C. Advise uses this probability and the Bayes rule to represent the human feedback policy [73].

I-SABL

I-SABL (Inferring Strategy-Aware Bayesian Learning) [33], which is most similar to Advise [41] uses expectation-maximization to calculate the best action. With this method, if the learning agent takes an optimal action, the human teacher provides positive feedback, otherwise, the human provides negative feedback [73].

ABLUF

ABLUF (Adaptive Bayesian Learning with Uncertain Feedback) [38] is based on expectation maximization algorithms, and it is similar to I-SABL [33]. However, whereas I-SABL assumes that the expert only provides positive feedback when the agent takes an optimal action, ABLUF models the human feedback as a probability distribution, where the probability of providing positive or negative feedback, increases or decreases with respect to the distance between the action taken and the optimal action.

TAMER

In the TAMER (Training an Agent Manually via Evaluative Reinforcement) framework [7] the teacher is seen as a reward function that maps the actions of the agent to negative, neutral or positive feedback. This kind of feedback is called evaluative feedback because the teacher evaluates how good or bad is the action taken by the agent. This reward function replaces the rewards provided by the environment in a classical reinforcement learning problem [73].

Deep TAMER

Deep TAMER [35] is a version of the TAMER framework [7] where the policy is represented with a deep neural network.

DQN-TAMER

DQN-TAMER [36] is a combination of the TAMER framework [7] with Deep Q-Network (DQN). The original TAMER framework does not take into account the environment reward; DQN-TAMER trains a DQN agent and a TAMER agent, and the final decision policy is a weighted average of the policies from both agents [73].

Convergent Actor-Critic by Humans

Convergent Actor-Critic by Humans [34] is an algorithm inspired in TAMER [7] that differs from this one in that fact that TAMER interprets human feedback as a reward function independent of the agent's current policy [73]. Contrary, Convergent Actor-Critic by Humans assumes that human feedback depends on the agent's current policy and that it should be interpreted as the advantage function that tells how much better or worse when deviating from the agent's current policy.

FRESH

FRESH (Feedback-based REward SHaping) [54] is similar to Deep-TAMER [35] but unlike Deep-TAMER, FRESH takes into account the reward from the environment.

LOKI

LOKI (Locally Optimal search after K-step Imitation) [37] is an algorithm that has two phases: A first imitation learning phase and a second reinforcement learning phase. First, it randomly picks a number within a range and performs that number of online imitation learning steps. Then, in the reinforcement learning phase, it improves the policy with a policy gradient RL method.

AOR

AOR (Adaptive On-Policy Regularization) [74] is an algorithm that uses dynamic regret which measures the performance of a policy at each iteration. Dynamic regret compares the current policy against the best it could be on its distribution with respect to the expert.

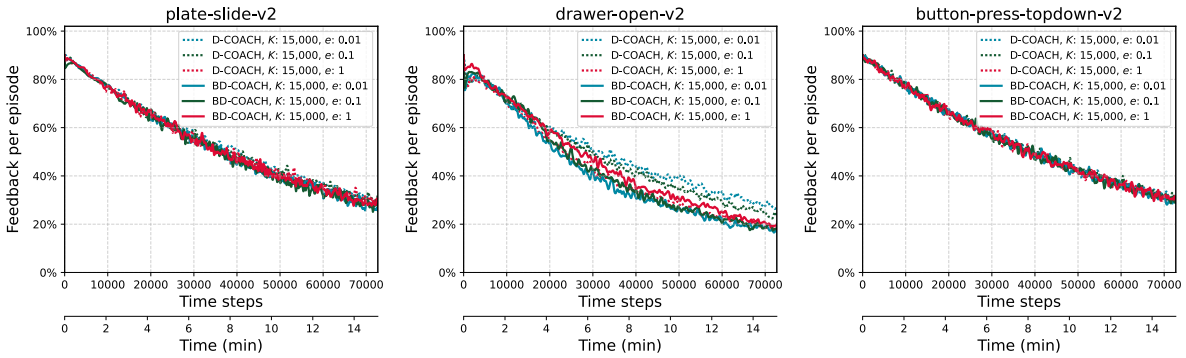
Cycle-of-Learning

Cycle-of-Learning [48] is a framework that allows to switch between multiple types of human interventions when teaching an agent. Human interactions are divided in three categories arranged from more human control to less human control: Learning from human demonstration, learning from human intervention and learning from human evaluation. As the learning task progresses in time and the agent improves, less human intervention is required and therefore the algorithm switches to the following type of intervention technique.

B

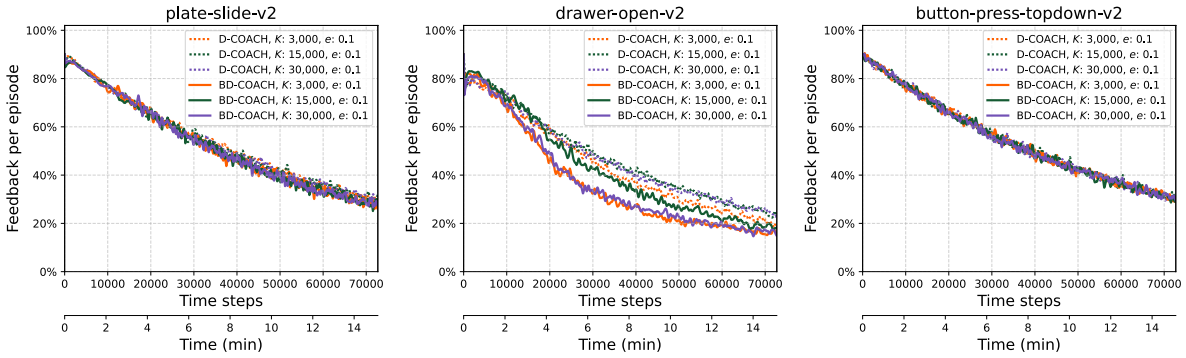
Feedback plots

The following feedback results show the behaviour of the simulated oracles that were implemented to run the simulated experiments for the three tasks plate-slide-v2, drawer-open-v2 and button-press-topdown-v2 explained in Section 4.1.



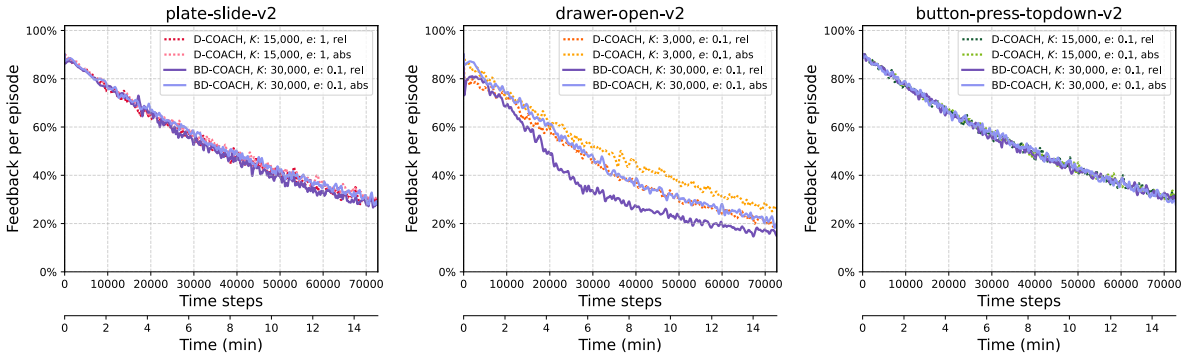
(a) plate-slide-v2 (b) drawer-open-v2 (c) button-press-topdown-v2

Figure B.1: Results of the percentage of time steps per episode that the simulated teacher, $P_h : \alpha = 0.9; \tau = 0.000015$, provides feedback for a fixed buffer size K and different values of e. These feedback plots correspond to the success results of Figure 5.1.



(a) plate-slide-v2 (b) drawer-open-v2 (c) button-press-topdown-v2

Figure B.2: Results of the percentage of time steps per episode that the simulated teacher, $P_h : \alpha = 0.9; \tau = 0.000015$, provides feedback for a fixed error magnitude e and different values of the buffer size K. The observations for all the conditions shown are relative positions. These feedback plots correspond to the success results of Figure 5.2.



(a) plate-slide-v2 (b) drawer-open-v2 (c) button-press-topdown-v2

Figure B.3: Results of the percentage of time steps per episode that the simulated teacher, $P_h : \alpha = 0.9; \tau = 0.000015$, when comparing the best performance of each method using relative positions, against their performances when using absolute positions for the same conditions of K and e. These feedback plots correspond to the success results of Figure 5.3.

C

Glossary

ANN	Artificial Neural Network
BC	Behavioral Cloning
BD-COACH	Batch deep COACH
CIL	Corrective Imitation Learning
COACH	COrrective Advice Communicated by Humans
D-COACH	Deep COACH
DAgger	Dataset Aggregation
ER	Experience Replay
FNN	Feedforward Neural Network
IL	Imitation Learning
IIL	Interactive Imitation Learning
MDP	Markov Decision Process
ML	Machine Learning
SGD	Stochastic Gradient Descent
TAMER	Training an Agent Manually via Evaluative Reinforcement