Université de Montréal

Imitation from Observation using Behavioral Learning

par

Medric B. Djeafea Sonwa

Département de mathématiques et de statistique
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Discipline

Orientation mathématiques appliquées

November 28, 2022

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Imitation from Observation using Behavioral Learning

présenté par

Medric B. Djeafea Sonwa

a été évalué par un jury composé des personnes suivantes :

Irina Rish

(président-rapporteur)

Eugene Belilovsy

(directeur de recherche)

Guillaume Lajoie

(membre du jury)

Résumé

L'Imitation par observation (IPO) est un paradigme d'apprentissage qui consiste à entraîner des agents autonomes dans un processus de décision markovien (PDM) en observant les démonstrations d'un expert et sans avoir accès à ses actions. Ces démonstrations peuvent être des séquences d'états de l'environnement ou des observations visuelles brutes de l'environnement. Bien que le cadre utilisant des états à dimensions réduites ait permis d'obtenir des résultats convaincants avec des approches récentes, l'utilisation d'observations visuelles reste un défi important en IPO. Une des procédures très adoptée pour résoudre le problème d'IPO consiste à apprendre une fonction de récompense à partir des démonstrations, toutefois la nécessité d'analyser l'environnement et l'expert à partir de vidéos pour récompenser l'agent augmente la complexité du problème. Nous abordons ce problème avec une méthode basée sur la représentation des comportements de l'agent dans un espace vectoriel en utilisant des vidéos démonstratives. Notre approche exploite les techniques récentes d'apprentissage contrastif d'images et vidéos et utilise un algorithme de bootstrapping pour entraîner progressivement une fonction d'encodage de trajectoires à partir de la variation du comportement de l'agent. Simultanément, cette fonction récompense l'agent imitateur lors de l'exécution d'un algorithme d'apprentissage par renforcement. Notre méthode utilise un nombre limité de vidéos démonstratives et nous n'avons pas accès à comportement expert. Nos agents imitateurs montrent des performances convaincantes sur un ensemble de tâches de contrôle et démontrent que l'apprentissage d'une fonction de codage du comportement à partir de vidéos permet de construire une fonction de récompense efficace dans un PDM.

Mots clés: Apprentissage par renforcement, Apprentissage par imitation, Imitation par observation, Apprentissage contrastif, Reconnaissance d'actions

Abstract

Imitation from observation (IfO) is a learning paradigm that consists of training autonomous agents in a Markov Decision Process (MDP) by observing an expert's demonstrations and without access to its actions. These demonstrations could be sequences of environment states or raw visual observations of the environment. Although the setting using low-dimensional states has allowed obtaining convincing results with recent approaches, the use of visual observations remains an important challenge in IfO. One of the most common procedures adopted to solve the IfO problem is to learn a reward function from the demonstrations, but the need to understand the environment and the expert's moves through videos to appropriately reward the learning agent increases the complexity of the problem. We approach this problem with a method that focuses on the representation of the agent's behaviors in a latent space using demonstrative videos. Our approach exploits recent techniques of contrastive learning of image and video and uses a bootstrapping algorithm to progressively train a trajectory encoding function from the variation of the agent's policy. Simultaneously, this function rewards the imitating agent through a Reinforcement Learning (RL) algorithm. Our method uses a limited number of demonstrative videos and we do not have access to any expert policy. Our imitating agents in experiments show convincing performances on a set of control tasks and demonstrate that learning a behavior encoding function from videos allows for building an efficient reward function in MDP.

Keywords: Reinforcement learning, Imitation learning, Imitation from observation, Contrastive learning, Action recognition

Contents

List of tables

List of figures

List of acronyms and abbreviations

RL	Reinforcement Learning
IL	Imitation Learning
IfO	Imitation from Observation
IRL	Inverse Reinforcement Learning
BC	Behavioral Cloning
MDP	Markov Decision Process
GAN	Generative Adversarial Network
GAIL	Generative Adversarial Imitation Learning
GAIfO	Generative Adversarial Imitation from Observation
DQN	Deep Q-Network

DDQN	Double Deep Q-Network
DDPG	Deep Deterministic Policy Gradient
CT	Context Translation
CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory

Acknowledgements

I would like to express my sincere gratitude to my supervisors Eugene Belilovsky and Johanna Hansen without whom I would certainly not have been able to complete this work. I am so grateful for your help, support, and especially your patience during the realization of this work. I thank you for your advice, your remarks, and for the time spent analyzing and correcting this work. It helped me a lot to have confidence during my work. I would also like to thank my friends from Mila and the University of Montreal with whom I participated in many projects that served as a basis for defining this thesis. I am thinking in particular of Kavin Patel and Maxime Heuillet with whom I studied approaches of Self-Supervised Learning and Reinforcement Learning through research projects. I would also like to thank you for making my stay in Montreal pleasant despite the cold weather and all the assignments. I want to thank Evelin Fonseca Cruz and Juan Duran with whom I played a lot with robots. Learning to write Reinforcement Learning algorithms for robots with you was the most fun part of this Master. To the Mila Community, I would like to say thank you for what you have built. Thanks to the exchange events and workshops, I was able to discuss my work with many researchers who gave me critiques and suggestions for improvements. To my teachers at Mila and the University of Montreal, Glen Berseth, Aaron Courville, Phillipe Langlais, and Pierre-Luc Bacon, I thank you very much for your advice and teachings. To my Montreal family, Loïs, Yann, Julien, Tristan, Samuel, Larry, Manuel, and all the others, I thank you for being by my side, especially during this last year when I was trying to adapt. You helped me a lot to overcome moments of loneliness. To my brothers, Maurel and Junior Sonwa, my mom Marlyse Sonwa, and to my uncle Charles Tento and his family, I say thank you, for the unconditional support, the encouragement, and the congratulations. You have always believed in me, I am proud to have you as my family. And to my late father, Abraham Sonwa, you left before I had time to complete this project, but I know that from where you are, you are proud of me, you always were. I dedicate this work to you.

Chapter 1

Introduction

1.1. Context: Reinforcement Learning and Imitation from observation

In recent years, Artificial intelligence and Deep learning have allowed the conception of algorithms to solve advanced problems that previously required human analysis. This progress is mainly perceptible in Computer vision with the introduction of the *Convolutional Neural Networks* (CNNs) which have allowed the development of algorithms of recognition [**44, 91, 21**] and detection [**26, 70**] through raw visual data. We also find this progress in *Natural Language Processing* (NLP) with the introduction of *Recurrent Neural Networks* (RNNs) that have been working on interpretation and operations (translation, speech recognition, sentiment analysis, text completion, etc.) [**8, 19**] on human language texts. Among these challenges, one of the tasks that has long interested research is the design of autonomous agents to solve specific tasks. Indeed, with the new skills acquired in recent years in Deep learning and Computer vision, it is natural to ask the question of how to train agents to perform tasks in the real world using physical interactions with the environment around them. This problem is largely studied today by the *Reinforcement Learning* (RL) [**75**]. RL is the branch of Deep learning that studies algorithms to learn an autonomous agent to perform a task by interacting with its environment. Let's consider the well-known problem of designing an autonomous car capable of taking its passengers anywhere without the intervention of a driver. This problem is modeled in RL by the car which is the autonomous agent and the city which is the environment in which the agent acts. In this application, the cameras and sensors placed on the car constitute the agent's sensors and allow it to observe and measure a part of the environment. The movements of the car constitute the actions emitted by the agent in this environment to modify its environment's state. During the execution of a task (observation of the environment and emission of actions), the general approach in RL

consists in providing rewards to the agent indicating how useful each action emitted was for the accomplishment of the task. These rewards are evaluated using a function called *Reward function*, and the agent's goal is to maximize its sum of rewards at the end of each task execution. A high sum of rewards means that the task has been completed. When choosing the action, the agent calls a function called *Policy function* which calculates the optimal action to take for the current state of the system. To date, there are several RL algorithms adapted to each specificity of the environment. These specificities vary according to the environment, the type of interaction, the number of possible actions, etc. The specificity at the level of the environment resides in the agent's capacity to fully or partially observe its environment. Indeed, although it is possible for some tasks to fully observe their environment, and to better predict the effects of their actions, this situation is rarely present in real applications. This problem is studied in particular in *Visual Question Answering* (VQA) [**5, 18, 3**] where we have an agent able to move in an environment and which is asked to answer a question whose answer is obtained after analysis of the environment. The type of interaction contains several parameters such as the atomicity of the actions (the actions can be atomic or composed), the continuous character of the interaction (the emission of the actions can be continuous with the propagation of the effects in all the process) and the sensitivity related to the precision of the actions. This configuration of the environment and interactions is at the heart of many learning algorithms for continuous control tasks [**46, 49, 43**] where the agent, to perfectly execute the task, has to emit at each instant the optimal action in order to maximize the total gain. Very often, in real situations, the task is a mix of a planning problem and a continuous control problem. This case is mainly studied by *Hierarchical Reinforcement Learning* (HRL) algorithms [**59, 52**] which design agents with many planning levels. The agent is composed of an upper layer, which is in charge of planning the non-atomic actions necessary for the realization of the task, and of lower layers which are in charge of executing atomic and continuous actions requiring optimal precision.

The execution process of a continuous control task is modeled in RL by a *Markov Decision Process* (MDP) [**6**] and the classical approach to train an autonomous agent for this task is via the use of a manually designed reward function that the agent will seek to maximize. In this process, the agent seeks for each observation of the environment to select the action that will return the maximum long-term gain. A major problem with this approach is the necessity of the reward function. In practice, in many cases, it is not always easy to design such a function, with the consistency that goes with it. However, for many tasks, it is common to have one person capable of successfully performing the task. One could then ask how to train a second agent to perform the task by observing the movements of an agent executing perfectly the task. It is normal to consider this possibility because imitation is a well-known learning approach used by humans. *Imitation from Observation* (IfO) is a

learning paradigm that consists in using demonstrations of an expert in a task to train a second imitating agent. In the case of an agent capable of obtaining visual observations of the environment, these demonstrations are ultimately videos of the expert performing the task. Contrary to classical RL methods, where we have by definition a reward function that is used to train the agent, in IfO, it is necessary to exploit only the videos showing examples of the execution of the task. Many algorithms have been designed for this type of problem conditioning, but the difficulties related to the training of the agent remain numerous.

1.2. Problems

Imitation from Observation (IfO) [**79, 47**] involves learning a policy function for an agent to solve a task using a set of demonstrations of an expert performing the same task. Distinct from Imitation Learning (IL) [**38, 67, 65, 11**] which uses sequences of expert observation-action pairs as demonstrations, in IfO, the demonstrations are only sequences of observations of the environment by the expert. Depending on the environment and the information available to the agent, an observation can be a description of the environment at a given time (joint angles, distance between objects, direction vector coordinates, velocity, etc.) provided in the form of a low-dimensional state vector of the environment, or a raw visual of the environment subject to further analysis and processing. The advantage of the IfO formalization is that it allows more natural agent learning scenarios, where difficult-to-acquire precise action data is not available. Furthermore, it more accurately approximates the way in which humans imitate experts during learning. The two main methods in the literature that have been used to train IfO problem agents are adversarial methods and reward learning methods [**82**]. The adversarial methods [**80, 81**] adapt ideas from inverse reinforcement learning [**36**], proposing a GAN-like architecture [**27**] where the agent is a trajectory generator associated with a discriminator function that evaluates how similar the state transitions produced are to that of an expert.

These methods have proven their effectiveness in the case of low-dimensional state observations. When dealing with high-dimensional visuals, [**80**] uses a small stack (of size 3) of successive images to evaluate fake and real data. However, when using high-dimensional visual observations instead of low-dimensional state sequences, each video transcribes the variations of the system states without clearly designating the main and critical features that describe the environment. The fact of reducing the problem to a classification of small sequences limits the capacity of the discriminator to analyze and identifies these features describing optimal actions. Being able to correctly understand and represent the system state using environment images is essential to evaluate state transitions. Moreover, this approach limits the understanding of specific behaviors planning their actions over long horizons. On

the other hand, reward learning methods [**40, 47, 12**] are based on the training of a reward function that will be subsequently used to reward an agent while applying a traditional RL algorithm. The main interest of such an approach is to learn a reward function that measures the efficiency of the agent actions based on the expert demonstrations. The different reward learning methods exploit a precise self-supervised learning objective, such as context variation between many executions [**47**], which allows the model to converge to a meaningful and efficient reward function for training the imitating agent.

1.3. Proposal and Contribution

In this thesis, we present *BootIfOL*, a novel reward learning IfO algorithm that relies on the representation of agent behaviors in a latent space using raw pixel-based demonstrations, illustrated in Figure 1.1. Unlike adversarial approaches, we use the entire observation sequences, also called trajectories, with a Long Short-Term Memory (LSTM) neural network [**37**] to keep track of the agent's actions and to represent the induced behavior of the agent at each sequence timestep. This method directly exploits a limited set of visual demonstrations from an expert to train an imitating agent, without having access to the expert policy or actions. Inspired by Berseth *et al.* [**9**], we also train the reward function progressively with new visual trajectories generated by the agent. The trained reward function is based on a trajectory encoding function which represents the trajectories in the latent space. In contrast to [**9**], our method trains the trajectory encoding function on a set of failure trajectories before the agent training starts. This helps to ensure the consistency of the reward function from the beginning of the agent training. We also use Contrastive Multiview Coding [**78**] and Dense Predictive Coding [**32**] methods for self-supervised learning of trajectories. Like [**47, 9**], a reward function is trained by interacting with the environment to encourage the imitating agent's behavior to be similar to the behavior of the expert. The motivation behind this approach is: (1) To learn and estimate, at each timestep, the system state using the agent's visual observations (knowing that the real system state is not accessible); (2) To identify the behavior induced by the agent over a certain period of task execution and make sure that this behavior serves the same purpose as the expert. Our method also demonstrates the interest in a first-round training of the trajectory encoding function in order to provide meaningful rewards to the agent from the beginning of its training, contrary to other similar reward learning methods that bypass this step. Our method successfully solves a range of tasks in the Deepmind Control Suite [**77**] and the Meta-world environment [**89**], approaching the level of sums of rewards per episode obtained by the experts on the same tasks.

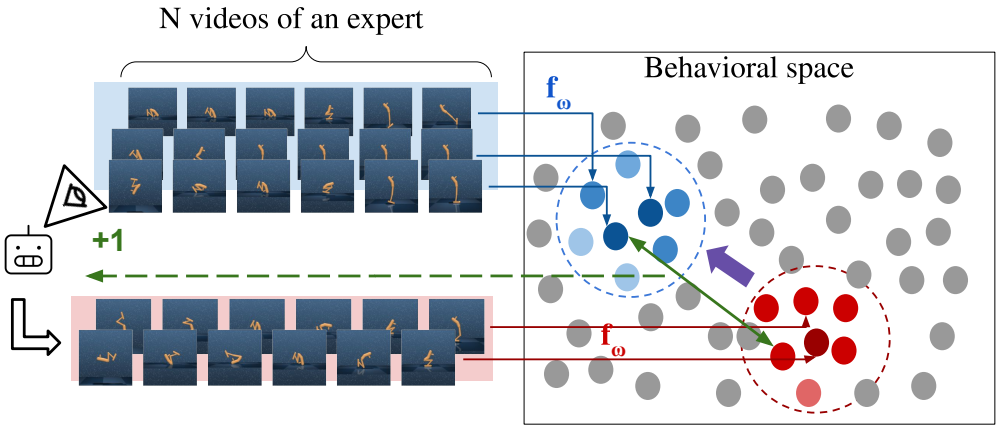


Fig. 1.1. Illustration of our method. An encoder that takes videos of agent trajectories and embeds them in a "behavioral space" is trained using contrastive learning that encourages successful trajectories to be near each other. We use this to encode N expert videos in a region of the behavioral space depicted in blue. The reward function corresponds to the distance of the agent's trajectory to the set of expert trajectories. As the agent progresses, its current trajectories are incorporated as "negative" examples into the contrastive learning in red.

1.4. Working paper

This thesis is based on the working paper called *Imitation from Observation With Bootstrapped Contrastive Learning* [**4**] that was accepted at Neurips 2022 3rd Offline RL Workshop, and currently submitted to the Conference on Computer Vision and Pattern Recognition (CVPR) 2022. In this work, I was the main contributor, and I participated in the development and study of the algorithm, the execution of experiments, the analysis of results, and the reporting of results and interpretations. The code associated to this project is available at `https://github.com/medric49/ifobl`.

1.5. Outline

The subjects covered in this thesis are organized as follows: In Chapter 2, we explain the mathematical foundations of all the approaches we exploit in our work. We present the different properties and algorithms we use in our thesis. In Chapter 3 we discuss the different works that have been realized in Imitation Learning and Imitation from Observation. Knowing the importance of learning and representation of visual data in our work, we also present recent approaches of Self-supervised learning of data and their principles. In Chapter 4, we present and explain our proposal. We explain the motivations behind this approach and the importance of the different parameters. In Chapter 5, we explain and present the results of the different experiments performed on our proposal in order to validate it. We explain the

behavior of this proposal when we vary the defined parameters and we present interpretations. And finally, in Chapter 6, we summarize the results obtained and we approach the possible improvements for future work.

Chapter 2

Background

2.1. Reinforcement Learning

Reinforcement Learning (RL) is the study of algorithms that train agents to make decisions to achieve goals during interactions in an environment. In this setting, we distinguish a system consisting of two entities: the environment and the agent. The environment is the place where events occur and which manifests the variation of the system states. The agent is the entity that observes the variations of states and takes actions to influence future states. When there are several agents that interact simultaneously, we call it a *multi-agent system*. In this project, we will only focus on the case of systems with one agent or *single-agent systems*.

2.1.1. Discrete-time finite-horizon stochastic process

In order to simplify the study of the process of task executions, RL approach formalizes this process as a *discrete-time stochastic process* [**75**]. The process is discretized into a sequence of many steps. During a step, the agent observes the environment which presents the current state of the system. Then, the agent takes a decision and emits an action. This action causes an event that changes the state of the environment and introduces the next step. This process is presented in Figure 2.1. When a task has always reachable final states where no additional action is admissible, we call it *finite-horizon task*.

For a finite horizon task, an instance of the execution process is called *episode*. Let us consider $\mathcal{S} = \{s\}$ the set of system states and $\mathcal{A} = \{a\}$ the set of actions. A finite horizon episode of T steps is described by the sequence $(s_0, a_0, \ldots, s_t, a_t, \ldots, s_T, a_T)$. At step t, the system is in state $s_t \in \mathcal{S}$ and the agent chooses the action $a_t \in \mathcal{A}$. Note that when the agent observes the environment, it does not always observe the whole system. At a time t, the agent measures

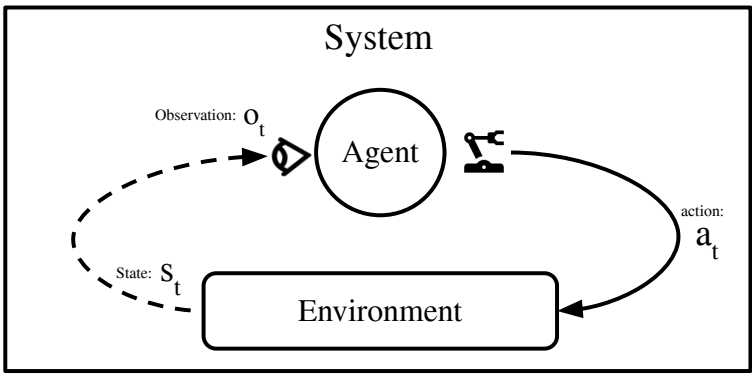


Fig. 2.1. Description of a system in which events and interactions occur. The system is a set consisting of an agent and an environment. The environment shows the state of the system and the agent can observe and modify it with actions.

a set of quantities of the environment called *observation* and represented by o_t. It chooses an action a_t based only on the sequence of observations $(o_0, \ldots, o_t)$. When $\forall\ t,\ o_t = s_t$ we say that the system is *fully observable*. When $\exists\ t \mid o_t \neq s_t$, we say that the environment is *partially observable*. Thus, the sequence observed by the agent is $(o_0, a_0, \ldots, o_T, a_T)$.

2.1.2. Markov decision process

The succession of events and states of the system is dictated by a probability function $\mathbb{P}$ called *dynamics function*, such that at time t, the probability that a state $s_{t+1} \in \mathcal{S}$ is reached leaving s_t is given by $\mathbb{P}(s_{t+1}|s_0, a_0, \ldots, s_t, a_t)$. A discrete-time stochastic process is said to respect the *Markov property* [**6**] when at any time t, the probability of reaching the state s_{t+1} depends only on the state s_t and the action a_t:

$$\forall t\ \mathbb{P}(s_{t+1}|s_0, a_0, \ldots, s_t, a_t) = \mathbb{P}(s_{t+1}|s_t, a_t) \tag{2.1.1}$$

In a discrete-time finite-horizon control process, the agent seeks to execute a specific task by making a sequence of decisions about its actions. In order to help the agent make optimal decisions, RL approach introduces a function R, called *reward function*, which at each new step $t \geq 1$, rewards the agent with a value $r_t = R(s_{t-1}, a_{t-1})$ depending on the previous state and action. When the episode starts, $r_0 = 0$. This reward function tells the agent how optimal its action was for the success of the task at the given time. The agent's objective is thus to maximize the sum of rewards at each episode. In RL formalism, the reward function is provided by the system.

Consider a system consisting of an agent and an environment. An agent-environment interaction process is called *Markov Decision Process* (MDP) [**6, 75**] when it is a discrete-time

stochastic process respecting the Markov property and defined by: (1) $\mathcal{S}$ the set of system states, (2) $\mathcal{A}$ the set of actions executable by the agent, (3) $\mathbb{P}$ the dynamics function of the system, (4) R is the reward function associated with the system. In this study, we assume that the execution processes we work on are finite-horizon MDPs.

2.1.3. Policy function

In an MDP, at each new observation o_t at step t, the agent selects an action a_t. The choice of action is computed using a function called *policy* and denoted π, such that $\pi(a_t|o_t)$ is the probability that the action a_t is optimal to maximize the total gain at step t. When the function π is a probability, we call it *stochastic policy*. When the function π directly returns the optimal action with $a_t = \pi(o_t)$, we speak of *deterministic policy*. In this work, depending on the context, we will use one or the other of these two forms.

Let us consider a finite-horizon MDP of the agent represented by its state-action sequence $(s_0, a_0, \ldots, s_T, a_T)$ also called *trajectory* and noted τ. At the end of this process, the sum of the rewards $\sum_{t=0}^{T} r_t$ obtained by the agent is called *return*. To simplify the writing, we note the return obtained until time t, $G_T = \sum_{t=0}^{t} r_t$. The objective of the agent controlled by the policy π is to maximize its return by selecting appropriate actions for any trajectory τ of the distribution of trajectories $\tau \sim P(s_0, a_0, \ldots, s_T, a_T|\pi)$ conditioned by π. Thus, a policy π is then evaluated by its *expected return* noted $J(\pi)$ and defined by:

$$J(\pi) = \mathop{\mathbb{E}}_{\tau \sim P(\{\tau\}|\pi)} G_T \tag{2.1.2}$$

Being able to measure the efficiency of a policy π, the objective of RL algorithms is then to find the optimal policy π^* from the set of policy functions that maximize the expected return:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} J(\pi) \tag{2.1.3}$$

When the function π is a parametric function of parameter θ and noted π_θ (for example a neural network), the objective is then to find the optimal parameter θ^* such that:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} J(\pi_\theta) \tag{2.1.4}$$

The main challenge in RL is to find algorithms that allow to optimize the policy function of an agent in a reduced computation time. There are two groups of algorithms, model-based RL algorithms, and model-free RL algorithms.

The model-based RL algorithms [**73, 31, 30, 69, 53**] focuses on the learning of functions allowing to estimate the dynamics function $\mathbb{P}$ and the reward function R. Indeed, having

an accurate estimation of these functions would allow the agent to predict the effect of its actions and to select actions that maximize the predicted rewards.

The model-free RL algorithms [**11, 68, 49, 35**] rely on the estimation of the gain value associated to each action and event based on previous experiences and previous gains.

2.1.4. Policy gradient methods

Consider a differentiable policy function π_θ to be optimized to maximize $J(\pi_\theta)$. A well-known approach to optimize θ consists in applying iterations of the *gradient ascent* [**93**] method: $\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta J(\pi_\theta)$ where $\nabla_\theta J(\pi_\theta)$ is the gradient vector of $J(\pi_\theta)$ w.r.t. to θ. The challenge of this method lies on the calculation process of $\nabla_\theta J(\pi_\theta)$. We know that:

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim P(\{\tau\}|\theta)} G_T \tag{2.1.5}$$

$$= \int_{\tau \sim P(\{\tau\}|\theta)} P(\tau|\theta) G_T d\tau \tag{2.1.6}$$

$$\nabla_\theta J(\pi_\theta) = \int_{\tau \sim P(\{\tau\}|\theta)} \nabla_\theta P(\tau|\theta) G_T d\tau \tag{2.1.7}$$

$$= \int_{\tau \sim P(\{\tau\}|\theta)} P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) G_T d\tau \tag{2.1.8}$$

$$= \mathop{\mathbb{E}}_{\tau \sim P(\{\tau\}|\theta)} [\nabla_\theta \log P(\tau|\theta) G_T] \tag{2.1.9}$$

Moreover, consider a trajectory $\tau = (s_0, a_0, \ldots, s_T, a_T)$ realized by the policy π_θ, we have:

$$P(\tau|\theta) = P(s_0, a_0, \ldots, s_T|\theta) \tag{2.1.10}$$

$$= P(s_0) \times \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) \mathbb{P}(s_{t+1}|s_t, a_t) \times \pi_\theta(a_T|s_T) \tag{2.1.11}$$

$$\log P(\tau|\theta) = \log P(s_0) + \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=0}^{T-1} \log \mathbb{P}(s_{t+1}|s_t, a_t) \tag{2.1.12}$$

$$\nabla_\theta \log P(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{2.1.13}$$

Finally,

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim P(\{\tau\}|\theta)} \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) G_T \right] \tag{2.1.14}$$

The result of Equation 2.1.14 allows in practice to evaluate $\nabla_\theta J(\pi_\theta)$ using an estimation over a batch of trajectories. The *REINFORCE* algorithm thus proposes a simple iteration scheme:

(1) Collect N trajectories using the current policy π_{θ_k};

(2) Estimate $\nabla_\theta J(\pi_\theta)$ on the N trajectories by $\nabla_\theta \hat{J}(\pi_\theta)$;
(3) Update θ with $\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta \hat{J}(\pi_\theta)$.

Estimation $\nabla_\theta \hat{J}(\pi_\theta)$ is given by:

$$\nabla_\theta \hat{J}(\pi_\theta) = \frac{1}{N} \sum_{n=1}^{N} G_{n,T} \left(\sum_{t=0}^{T} \nabla_\theta \log \pi_{\theta_k}(a_{n,t}|s_{n,t}) \right) \tag{2.1.15}$$

This algorithm allows to optimize a parametric and differentiable policy function in order to approximate the optimal policy of the task. Nevertheless, it raises a problem. In Equation 2.1.14, the term $_T$ acts as a weight on $\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$. In practice, when we collect N trajectories for an iteration, the gradient will be much more influenced by the trajectories with high returns than by the others.

This situation is not desirable because we would like each step to have a considerable weight in the gradient, as the agent must be able to learn step-by-step and not episode-by-episode. Indeed, some steps could have a large amount of information to provide but will then be shaded by the fact that its episode return is low. To remedy this, $\nabla_\theta \hat{J}(\pi_\theta)$ is rewritten by weighing the steps and not the whole episode:

$$\nabla_\theta \hat{J}(\pi_\theta) = \frac{1}{NT} \sum_{n=1}^{N} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_{\theta_k}(a_{n,t}|s_{n,t}) r_{n,t+1} \approx \mathop{\mathbb{E}}_{\tau \sim P(\{\tau\}|\theta)} [\nabla_\theta \log \pi_\theta(a_t|s_t) r_{t+1}] \tag{2.1.16}$$

Rewriting $\nabla_\theta \hat{J}(\pi_\theta)$ in the form of Equation 2.1.16 allows in practice to process the agent's steps without taking into account the order in their original episodes. Moreover, the steps influence independently the gradient according to the importance of the obtained reward. This ability allows the agent to learn from its mistakes on any episode even when it obtains a low return. Although the use of rewards r_t offers better robustness in the estimation of $\nabla_\theta J(\pi_\theta)$ it nevertheless causes concern about stability. This parameter does not allow the agent to anticipate the repercussions of its actions in the future. Indeed, when the agent selects an action, it first tries to maximize the total return of the episode from the current time. Weighting using only r_t encourages the agent to choose the action that maximizes the reward after one step, which limits its ability to anticipate the future rewards it might obtain. To correct this, a method widely used in practice consists in weighting the terms $\log \pi_\theta(a_t|s_t)$ by $\delta_{t,p} = \sum_{i=1}^{p} \gamma^{i-1} r_{t+i}$ with $\gamma \in [0, 1]$ and p the number of steps in the future. This method has been introduced by the family of algorithms TD(λ)[**74**]. The reason for this value is to inform the agent of the importance of its action in future steps. Using $\delta_{t,p}$ allows the agent to select actions that maximize the return from step t, which allows it to see that actions can have an optimal effect in the future but not in the present. The parameter γ is called *discount factor* and is a hyper-parameter of the system that configures the agent's level of anticipation. Let us consider this time that the episodes generated by the policy

π_θ are concatenated: $(s_{0,0}, a_{0,0}, \ldots, s_{0,T}, a_{0,T}, \ldots, s_{N,0}, a_{N,0}, \ldots, s_{N,T}, a_{N,T})$ and let us use the notation $\bar{t}$ to denote the absolute index of a step $(s_{n,t}, a_{n,t})$ in this sequence of concatenated trajectories. The new estimate of $\nabla_\theta J(\pi_\theta)$ on N steps of the sequence is then:

$$\nabla_\theta \hat{J}(\pi_\theta) = \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta \log \pi_{\theta_k}(a_{\bar{t}_n} | s_{\bar{t}_n}) \delta_{\bar{t}_n, p} \tag{2.1.17}$$

The policy gradient methods [**76, 68, 86**] essentially rely on estimating the gradient of the expected return. We note that in all variations of these methods, it is necessary to collect a set of steps at each iteration that will be used to estimate the new gradient. In order to guarantee the convergence of the method, this collection of steps must always be done using the current policy. This necessity poses a problem because after using these steps they become useless. This way of using the data is called *online setting*, because the data used are continuously controlled by the algorithm, and the data coming from other distributions are not exploitable. Another group of algorithms avoids this problem and are called *offline RL algorithms*. These are mainly the *Q-learning* methods.

2.1.5. Q-learning and Deep Q-networks

Let us consider a policy function π executed by an agent in an MDP. Let $s \in \mathcal{S}$ be a state of this process, let us define the function $V_\pi : \mathcal{S} \to \mathbb{R}$ such that:

$$V_\pi(s) = \mathop{\mathbb{E}}_{P(s_t, a_t, \ldots, s_T, a_T | \pi, s_t = s)} \left[\sum_{i=1}^{T} \gamma^{i-1} r_{t+i} \right] \tag{2.1.18}$$

V_π is called the *value function* associated with π and computes the expected discounted return of the agent from a state s. Let us also define $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ such that:

$$Q_\pi(s, a) = \mathop{\mathbb{E}}_{P(s_t, a_t, \ldots, s_T, a_T | \pi, s_t = s, a_t = a)} \left[\sum_{i=1}^{T} \gamma^{i-1} r_{t+i} \right] \tag{2.1.19}$$

$Q_\pi(s, a)$ is called the q-value function and computes the expected discounted return of the agent from a state s and choosing the action a. The introduction of these two functions allows us to define a new approach of calculating the optimal policy π^*. Consider $Q^* = Q_{\pi^*}$ the q-value function associated to the optimal policy π^*, we have:

$$Q^*(s,a) = \max_\pi Q_\pi(s, a) \tag{2.1.20}$$

Moreover, it is also shown that $\pi^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a)$ [**75**], so finding the optimal q-value Q^* would recover π^*. Let Q_π be a q-value function and V_π a value function associated to a

policy π in the set of policy functions. Consider the following equations:

$$Q_\pi(s,a) = \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)[R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q_\pi(s',a')] \tag{2.1.21}$$

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)[R(s,a) + \gamma V_\pi(s')] \tag{2.1.22}$$

Equations 2.1.21 and 2.1.22 are called *Bellman equations* [**7**] and it is shown that (Q^*, V^*) is the unique solution of these equations [**75, 61**]. Knowing that it is possible to define π^* as a function of Q^*, the problem is reduced to finding an estimate of Q^*, this approach is called *Q-learning*. Several Q-learning algorithms exist to estimate the optimal q-value function of an MDP. When the sets $\mathcal{S}$ and $\mathcal{A}$ are finite sets, the simplest method consists in maintaining a two-dimensional table, where the indices are the couples $(s,a) \in \mathcal{S} \times \mathcal{A}$ and the values are estimates of $Q^*(s,a)$. After initialization of the table $Q = Q_0$, the algorithm loops on 3 operations until the convergence of the table:

(1) Generate one step $(s_t, a_t, r_{t+1}, s_{t+1})$;
(2) Calculate the target value of the q-value function $y = r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q_k(s_{t+1}, a)$;
(3) Update the table $Q_{k+1}(s_t,a_t) \leftarrow (1-\alpha)Q_k(s_t,a_t) + \alpha y$.

Iteration stops when $\max_{(s,a)}(|Q_{k+1}(s,a) - Q_k(s,a)|) \leq \epsilon$, and the estimate of the optimal policy is $\pi : \pi(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s,a)$. When the set of $\mathcal{S}$ states is a continuous space, [**63, 50**] proposes the method called *Deep Q-Networks* (DQN) which consists in using a parametric and differentiable q-value function Q_θ which returns a vector with $|\mathcal{A}|$ components where each component represents the q-value of the couple (s,a) for a given state $s \in \mathcal{S}$. In this configuration, we introduce the q-value loss function objective to minimize $\mathcal{L}_{\text{DQN}}(\theta)$, defined by:

$$\mathcal{L}_{\text{DQN}}(\theta) = \frac{1}{2}||Q_\theta(s_t,a_t) - y_\theta||^2 = \frac{1}{2}||Q_\theta(s_t,a_t) - (r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q_\theta(s_{t+1}, a))||^2 \tag{2.1.23}$$

Parameter θ_k is updated at each iteration via the instruction:

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta \mathcal{L}_{\text{DQN}}(\theta_k) \tag{2.1.24}$$

Moreover, we introduce a memory $\mathcal{D}$ called *Replay memory* or *Replay buffer* in which are saved the steps $(s_t, a_t, r_{t+1}, s_{t+1})$ generated by the agent. From this set are extracted samples in the form of batches which allow us to evaluate $\mathcal{L}_{\text{DQN}}(\theta)$ and to update θ_k. This algorithm, being an offline RL algorithm, offers the possibility to train the agent with steps generated a long time ago. The main objective here is to train a precise q-value function. In order to improve the stability of this algorithm, the approach proposed by [**51**] consists in introducing a second q-value function $Q_{\bar{\theta}}$ to compute the target value $y_{\bar{\theta}} = r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q_{\bar{\theta}}(s_{t+1}, a)$. The use of $Q_{\bar{\theta}}$ to compute the target value has the effect of limiting the variation of the target value

distribution and thus of limiting the instability of the $\nabla_\theta \mathcal{L}_{\text{DQN}}$. Moreover, the parameter $\bar{\theta}$ is updated regularly with a frequency lower than that of θ_k by the instruction $\bar{\theta} \leftarrow \theta_k$ or by an exponential moving average $\bar{\theta} \leftarrow \rho\bar{\theta} + (1-\rho)\theta_k$.

It is also possible to combine the *TD(λ)* [**72**] approach with the DQN algorithm in order to estimate the target $y_{\bar{\theta}}$ on several steps. Indeed, the current method consists in estimating $y_{\bar{\theta}}$ on the reward value following the current step, but the approach called *multi-step Q-learning* consists in defining the multi-step target $y_{\bar{\theta},p}$ by:

$$y_{\bar{\theta},p} = \sum_{i=1}^{p} \gamma^{i-1} r_{t+i} + \gamma^p \max_{a \in \mathcal{A}} Q_{\bar{\theta}}(s_{t+p}, a) \tag{2.1.25}$$

A fundamental problem with the DQN algorithm and its variations is the term $\max_{a \in \mathcal{A}} Q_{\bar{\theta}}(s_{t+1}, a)$ used to evaluate the target value in Equation 2.1.25. This term has the effect of overestimating the q-value, which in practice is considerably different from the q-value associated with the current policy function. Thus, in order to limit this effect, the approach *Double Deep Q-Networks* [**83**] (DDQN) proposes to replace the target value by:

$$y_{\bar{\theta},\theta_k,p} = \sum_{i=1}^{p} \gamma^{i-1} r_{t+i} + \gamma^p Q_{\bar{\theta}}(s_{t+1}, \operatorname*{argmax}_{a \in \mathcal{A}} Q_{\theta_k}(s_{t+p}, a)) \tag{2.1.26}$$

The interest here is to evaluate the target value using the policy function $\pi_{\theta_k}(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q_{\theta_k}(s)$, which is optimal for the q-value function Q_{θ_k}.

The DQN and DDQN approaches are still limited by the fact that the set $\mathcal{A}$ is finite. We saw in section 2.1.4 that it was possible to parameterize the policy function π and to use it in continuous spaces. The intuition is then to combine the DQN and policy gradient methods in order to eliminate the need to work with a finite set of actions. The *Actor-Critic* algorithms propose a solution to this limitation, and a method to optimize simultaneously a differentiable policy function and q-value function.

2.1.6. Deep Deterministic Policy Gradient

The necessity to have a finite set of actions by the DQN and DDQN poses a problem knowing the actions that our agent can emit could be defined in a continuous space. This difficulty is precisely manifested by the term $\operatorname*{argmax}_{a \in \mathcal{A}} Q_{\theta_k}(s,a)$ in the expression of the DDQN target value defined in Equation 2.1.26. *Deep Deterministic Policy Gradient* (DDPG) approach [**46**] proposes a combination of a policy gradient and a q-learning method to train respectively a parametric policy function called *Actor* and denoted π_ϕ and a parametric q-value function called *Critic* and denoted Q_θ. Q_θ is trained to approximate the q-value of any pair (s, a)

with respect to the policy π_ϕ, then to minimize the objective loss function $\mathcal{L}_{\text{critic}}$ defined by:

$$\mathcal{L}_{\text{critic}}(\theta) = \mathop{\mathbb{E}}_{(s_t,a_t,r_{t+1:t+p},s_{t+p})\in\mathcal{D}} ||Q_\theta(s_t,a_t) - \sum_{i=1}^{p} \gamma^{i-1} r_{t+i} + \gamma^p Q_{\bar{\theta}}(s_{t+p}, \pi_\phi(s_{t+p}))||^2 \quad (2.1.27)$$

π_ϕ is trained to return the action which maximizes $Q_\theta(s,\cdot), \forall s \in \mathcal{S}$, which means to minimize the objective loss function $\mathcal{L}_{\text{actor}}$ defined by:

$$\mathcal{L}_{\text{actor}}(\phi) = - \mathop{\mathbb{E}}_{s\in\mathcal{S}, a\sim\pi_\phi(\mathcal{A}|s))} Q_\theta(s,a) \quad (2.1.28)$$

DDPG is an offline algorithm because of the samples $(s_t, a_t, r_{t+1:t+p}, s_{t+p})$ coming from the replay memory $\mathcal{D}$. We notice in this algorithm that it is possible to use a policy function totally different from π_ϕ to generate the steps. This technique is called *off-policy* and allows to optimize the main policy π_ϕ called *target policy*, with the help of a second policy function called *behavior policy*, and noted b, which is in charge of generating the steps. Using b instead of π_ϕ during the training allows promoting the exploration of the environment and the diversity of trajectories in the replay memory $\mathcal{D}$. The more diverse $\mathcal{D}$ contains trajectories, the more understanding the model has of the system rules and the more accurate Q_θ is. However, if b focuses too much on exploration, $\mathcal{D}$ will not contain enough successful trajectories and the models will not be able to recognize states that lead to success. The conception of b must therefore take into account this trade-off called *exploration-exploitation trade-off* [**75**]. There are several techniques to define the behavior policy b, but one of the most common is to add a noise of decreasing amplitude to the actions returned by π:

$$b(s) = a \sim \mathcal{N}(\pi(s),\epsilon_n); \lim_{n\to+\infty} \epsilon_n \to 0 \quad (2.1.29)$$

with n being the number of training iterations.

DDPG algorithm is one of many other approaches [**53, 49, 68**] to train agents for continuous control tasks with states and actions defined in continuous spaces. The results obtained with this algorithm allow us to deepen the difficulties and explore the cases where this method can be applied. One of the most studied cases to date (and the one that interests us in this work) is the resolution of tasks using only pixel-based visual observations of the environment as inputs. The possibility of solving tasks relying only on visuals is an ultimate goal of RL. In addition to getting as close as possible to the condition in which humans train to perform tasks, this configuration allows to make the process of observation acquisition realistic. In real situations, an agent is more likely to have a raw visual observation of its environment than a feature-based description of it. For this type of environment, in each process, the agent's observations o_t are raw images of the scene observed by the agent. [**88**] proposes the DrQ-v2 algorithm, which is a DDPG-based method that uses raw visual observations of the environment as input data of the agent's policy and applies image augmentation techniques

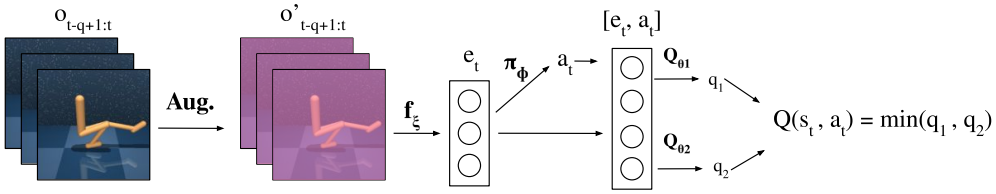


Fig. 2.2. Schema of inference of the algorithm DrQ-v2 [**88**].

on observations to improve the robustness of the policy and value functions. As illustrated at the Figure 2.2, DrQ-v2 simultaneously trains an image encoder function f_ξ, a policy function π_ϕ and two twin q-value functions Q_{θ_1} and Q_{θ_2}:

- f_ξ encodes the q last observations $o_{t-q+1:t}$ of the agent during each step t and returns a low-dimensional encoding e_t.
- π_ϕ is the trained target policy and takes as input an observation encoding e_t.
- Q_{θ_1} and Q_{θ_2} are two q-value functions trained with the loss function $\mathcal{L}_{\text{critic}}$ [**25**]. The final q-value of a pair (e,a) is $\min(Q_{theta_1}(e,a), Q_{theta_2}(e,a))$.

This algorithm relies on the use of a stack of recent observations $(o_{t-q+1}, \ldots, o_t)$ at each time t. This choice allows the agent to measure physical quantities, such as speed and acceleration, that can influence the choice of actions. Moreover, this algorithm proposes the use of data augmentation methods on the observation images received by the image encoder in order to improve the representation of the images in low-dimensional vectors. We present the pseudo-code of DrQ-v2 to the Algorithm 1.

2.2. Imitation learning

Learning autonomous agents in the context of an MDP has shown convincing performances and results are constantly evolving. However, it remains in this configuration some conditions to be respected which are rather strong and difficult to adapt in real situations. Among the constitutive components of an MDP, we have the reward function which provides a signal that indicates to the agent the positive impact of its actions. When learning an agent in an RL algorithm, the reward function is supposed to be acquired but in practice, it is not always possible to have a positive or negative signal function that perfectly describes the effectiveness of each action issued by the agent. In some tasks, it is possible to know what the final states look like but not exactly the optimal paths to reach them. We can also find ourselves in a situation where we know how to start the task but the design of the reward function reflecting the method is difficult. All these situations lead us to conclude that in

Algorithm 1: DrQ-v2

Initialize $\mathcal{D} = \{\}$ `// Replay memory`
Initialize Q_{θ_1}, Q_{θ_2} `// Q-value functions`
Initialize π_ϕ `// Policy function`
Initialize f_ξ `// Image encoding function`
while step $\leq N_\pi$ **do**
  $o_0 \leftarrow$env.reset()
  for $t \in 0, \ldots, T-1$ **do**
    $a_t \leftarrow \pi_\phi(f_\xi(o_{t-q+1:t})) + \epsilon_{\text{step}}$
    $o_{t+1} \leftarrow$env.step(a_t)
    $r_{t+1} \leftarrow$R(o_t, a_t)
    Save $(o_t, a_t, o_{t+1}, r_{t+1})$ into $\mathcal{D}$
    if step mod $N_{\text{update}} = 0$ **then**
      $(o_{i-q+1:i}, a_i, o_{i-q+1+p:i+p}, r_{i+1:i+p}) \leftarrow$sample(D)
      $e_i \leftarrow$f$_\xi$(aug($o_{i-q+1:i}$))
      $e_{i+p} \leftarrow$f$_\xi$(aug($o_{i-q+1+p:i+p}$))
      $a_{i+p} \leftarrow \pi_\phi(e_{i+p}) + \epsilon_{\text{step}}$
      target $\leftarrow \sum_{j=1}^{p} \gamma^{j-1} r_{i+j} + \gamma^p \min(Q_{\bar{\theta}_1}(e_{i+p}, a_{i+p}), Q_{\bar{\theta}_2}(e_{i+p}, a_{i+p}))$
      $\mathcal{L}_{critic} \leftarrow$|| Q$_{\theta_1}(e_i, a_i)$ − target$||^2 + ||Q_{\theta_2}(e_i, a_i) -$ target$||^2$
      Optimize θ_1, θ_2, ξ with $\nabla \mathcal{L}_{critic}$
      $\hat{a}_i \leftarrow \pi_\phi(e_i) + \epsilon_{\text{step}}$
      $\mathcal{L}_{actor} \leftarrow$ -min$(Q_{\theta_1}(e_i, \hat{a}_i), Q_{\theta_2}(e_i, \hat{a}_i))$
      Optimize ϕ with $\nabla \mathcal{L}_{actor}$
      Update $\bar{\theta}_1$, $\bar{\theta}_2$ with θ_1, θ_2 using exponential moving average
    end
  end
end

some conditions the necessity of a reward function is a very strong condition. However, let us consider the fact that we have at our disposal an agent capable of performing the task, without having access to the policy function describing its behavior. This situation is perfectly plausible because, for most tasks, it is the existence of a demonstration that allows us to conclude that the task is feasible. Imitation learning (IL) [**38, 67, 65, 11**] studies the problem of training an agent by imitating the behavior of an expert agent. In this problem, the expert is an agent able to perform the task as many times as desired, but whose action-decision policy is impossible to have. In the IL paradigm, we assume that the information accessible from the expert is its observations o_t (or states s_t in case of fully observable MDP) and a description of its actions a_t chosen at each step t. Thus, for each episode of the expert, we have access to its trajectory $\tau_e = (s_0, a_0, \ldots, s_T, a_T)$ also called *demonstration*. Let us note by π_e the unknown policy of the expert. Knowing that we do not have a reward function, the objective of the IL algorithms is to train the agent policy π using the distribution of expert's demonstrations $\tau_e \sim P(s_0, a_0, \ldots, s_T, a_T | \pi_e)$.

2.2.1. Behavioral cloning

Bevioral cloning [**65, 11**] is an IL approach which learns a policy function by cloning the decision process of an expert. The learned policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is a mapping function trained by supervised learning on the pairs (s_t, a_t) generated by the expert. During training, the agent learns the mapping process between state and action adopted by the expert and generalizes to new states. To train an autonomous car, [**11**] proposes a method of acquiring expert's steps that consists in placing cameras at the front of the car (to obtain a visual of the driver's observation) and sensors measuring the oration of the steering wheel at each instant. The trained policy function is then a CNN translating the raw images into steering wheel orientation inputs. This method has the advantage of exploiting the methodology of supervised learning which simplifies the problem and leaves a variety of choices on the algorithm to use.

2.2.2. Inverse reinforcement learning

Inverse reinforcement learning (IRL) [**1, 45, 55**] proposes to rebuild a reward function that appropriately rewards the agent as it progresses through the task. Once this function is reconstructed, it is used to train the agent to perform the task using an IRL algorithm. The main challenge of this approach is to deduce this function using the demonstrations provided by the expert. [**1**] proposes an IRL method based on the approximation of a parametric function to an optimal reward function of the expert policy. Indeed, let us consider an expert policy function π_e and a target reward function R_e such that π_e maximizes J_{R_e}:

$$J_R(\pi) = \mathop{\mathbb{E}}_{\tau \sim P(\{\tau\}|\pi)} \sum_t \gamma^t R(s_t, a_t) \tag{2.2.1}$$

The objective of this method is to approximate simultaneously π_e and R_e by respectively a policy π and a reward function R. We assume to have a feature extractor function $f : \mathcal{S} \rightarrow [0,1]^k$ which from a state s extracts the only features allowing to measure a reward. These features are normalized on $[0,1]$. We also assume that the explored reward functions $\mathcal{R}$ are limited to functions $R : \mathcal{S} \rightarrow [0,1]$ associated to a parameter $\omega \in \mathbb{R}^k$, $||\omega||_1 \leq 1$, such that $R(s) = \omega \cdot f(s)$. Here, the reward functions are parametric and take as input only the current state of the system and return a reward in $[0,1]$. This configuration of the problem allows us to rewrite J_R of Equation 2.2.1 by J_ω:

$$J_\omega(\pi) = \mathop{\mathbb{E}}_{\tau \sim P(\{\tau\}|\pi)} \sum_t \gamma^t \omega \cdot f(s_t) \tag{2.2.2}$$

$$= \omega \cdot \mathop{\mathbb{E}}_{\tau \sim P(\{\tau\}|\pi)} \sum_t \gamma^t f(s_t) \tag{2.2.3}$$

$$= \omega \cdot \mu(\pi) \tag{2.2.4}$$

$$\mu(\pi) = \mathop{\mathbb{E}}_{\tau \sim P(\{\tau\}|\pi)} \sum_t \gamma^t f(s_t) \tag{2.2.5}$$

$\mu(\pi)$ is called *feature expectations*. Our objective is then to minimize $||\mu(\pi_e) - \mu(\pi)||$.

Let us consider an RL algorithm that allows us to find (or estimate) the optimal policy function of an MDP defined by a reward function R_ω (for example via a DQN or a Policy gradient algorithm seen in Section 2.1.4). Let us note by $L : \{\omega\} \to \{\pi\}$, the operator which associates to each ω the optimal policy function associated to R_ω by the RL algorithm. Thus, we look for ω and π such that:

$$\omega = \underset{\omega}{\operatorname{argmin}} ||\mu(\pi_e) - \mu(L(\omega))|| \tag{2.2.6}$$

$$\pi = L(\omega) \tag{2.2.7}$$

To obtain optimal ω and π, [**1**] in their algorithm, computes in a loop these two values through Equations 2.2.6 and 2.2.7 until the convergence of π. This algorithm inspires other methods to train IL agents with fewer constraints related to assumptions.

2.3. Imitation from observation with learned reward function

Imitation learning is a problem in which we want to train an agent to perform a task using the demonstrations of an expert performing the same task. The particularity of this problem is that we have (or we are able to have) a set of demonstrations or trajectories O_e of the expert containing at each step the state of the expert, and the action performed by the expert: $O_e = (s_1, a_1, s_2, a_2, ..., s_T, a_T)$. The second particularity in this configuration is that we do not have a reward function to help reward the agent's actions. The agent will have to learn to catch the expert's behavior using its distribution of trajectories $O \sim p(\{O_e\})$. In Imitation from Observation problems, the agent no longer has access to the actions performed by the expert in its trajectories. The reward engineering approach consists of exploiting a manually designed or trained reward function that will help to reward the agent. Although the pretext task motivating the learning of the reward function differs, one of the most popular practices consists in evaluating at each step t the distance $d(s_{1:t}, s_{e,1:t})$ between the agent's trajectory and the expert's one [**71, 47, 9**]. The agent is then rewarded by $r_t = -d(s_{1:t}, s_{e,1:t})$. The

intuition behind this practice is to require the agent to follow a state path close to the expert's (predicted) state. We use this approach in our work.

2.4. Representation learning

2.4.1. Image representation with contrastive learning on multiple views

A key problem when learning high-dimensional data, such as images, is the representation of these data in low-dimensional vectors encoding the amount of information needed to identify the idea transcribed by the data. In a supervised classification of images with a CNN, this problem is easily solved by the fact that the CNN identifies more easily features that match the same label. However, in many cases, including ours, the images have neither labels nor descriptions. Self-supervised learning is a method for learning unlabeled data that relies on identifying pseudo-classes or sub-distributions appearing in the unlabeled dataset in order to represent samples in a target feature vector space.

Among the different algorithms of self-supervised learning of images [**10, 14, 23**], contrastive learning methods [**57, 90**] using views of images have allowed obtaining surprising performances in recent advances. Let us consider a dataset $\mathcal{D} = \{o^i\}_{1\leq i\leq N}$ of images or observations on which we want to learn an image encoder function g_θ. Contrastive Multi-view Coding (CMC) [**78**] is a contrastive learning method whose principle is to transform each image $o \sim p(\mathcal{D})$ into M views $(v_1, v_2, ..., v_M)$ using a transformation that preserves the semantics of the image. Consider an image o, these views $(v_1, v_2, ..., v_M)$ can be luminosity variations, patches, luminance and chrominance components. In the case of videos, a view of a video frame can be one of the closest frames of this frame. CMC objective is to train M encoding functions $\{g_{\theta_i}\}_{1\leq i\leq M}$ for each view type to return similar encodings. In this work, we will focus on the case $M = 2$, but [**78**] generalizes this method to $M \geq 2$ views. $\{g_{\theta_i}\}_{1\leq i\leq M}$ are neural networks and are trained on pairs of views. We can write $\mathcal{D} = \{o^i\}_{1\leq i\leq N} = \{(v_1^i, v_2^i)\}_{1\leq i\leq N}$, and we consider the distributions $x \sim p(\{v_1^i, v_2^i\})$ of positive pairs and $y \sim p(\{v_1^i, v_2^j\}_{i\neq j})$ of negative pairs. Consider $h_k^i = g_k(v_k^i)$, the learning objective is then to minimize the contrastive loss function $\mathcal{L}_S^{1,2}$ given by:

$$\mathcal{L}_S^{1,2} = - \mathop{\mathbb{E}}_{(h_1^0, h_2^0, h_2^1, ..., h_2^n)} \left[\log \frac{\texttt{sim}(h_1^0, h_2^0))}{\texttt{sim}(h_1^0, h_2^0) + \sum\limits_{1\leq i\leq n} \texttt{sim}(h_1^0, h_2^i)} \right] \tag{2.4.1}$$

where $\texttt{sim}(a,b) = \exp(\frac{a^T b}{\tau \|a\| \|b\|})$ and τ is the temperature parameter that controls the value interval of the similarity coefficient. We illustrate the generation of positive and negative

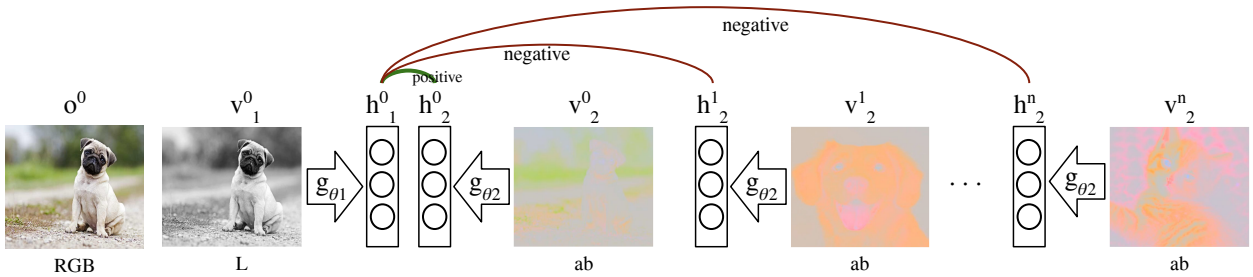


Fig. 2.3. Principle of view generation and creation of positive pairs and negative pairs. n RGB images are collected and translated into the Lab color space. For each image o^i, the view v_1^i is the L component of o^i in the Lab space, and the view v_2^i is the ab component of o^i in the Lab space. A positive pair is a pair of view encodings from the same image. A negative pair is a pair of view encodings from two different images.

pairs in Figure 2.3. When we switch the position of the views 1 and 2, we obtain a new loss function $\mathcal{L}_S^{2,1}$. We can the generalize the contrastive loss function to $\mathcal{L}_S$ by:

$$\mathcal{L}_S(\theta) = \mathcal{L}_S^{1,2}(\theta) + \mathcal{L}_S^{2,1}(\theta) \tag{2.4.2}$$

The final encoding s of an image o is given by $s = g_\theta(o) = [g_{\theta_1}(v_1), g_{\theta_2}(v_2)]$. In this work, we consider the color space transformation $\{R, G, B\} \rightarrow \{L, ab\}$ [**17**] where the component L is the view 1 the component ab is the view 2.

2.4.2. Video representation with next-frame prediction

Learning to represent a sequence of frames [**28, 32**] is also a crucial task in this work and for this, we need to introduce the Dense Predictive Coding (DPC) algorithm [**32**] from which our algorithm for encoding agent trajectories is inspired. DPC proposes the pretext task of predicting the next frame using the sequence of previous frames as input. Let us consider a dataset of videos $\mathcal{D} = \{O^i\}_{1 \leq i \leq N} = \{o^i_{0:T}\}_{1 \leq i \leq N}$. Let us define g_θ as the image encoding function that represents each frame in an image encoding space and f_ω as the sequence encoder function which represents each sequence of image encodings in a trajectory encoding space. We, therefore, need to introduce a function d_ϕ which, from a trajectory encoding z_t of a sequence $s_{0:t}$, predicts the next image encoding $\hat{s}_{t+1}$ of that sequence. The prediction $\hat{s}_{t+1}$ is then used to predict the next image encoding $\hat{s}_{t+2}$:

$$\hat{s}_{t+1} = d_\phi(z_t) = d_\phi(f_\omega(s_1, ..., s_t)) \tag{2.4.3}$$

$$\hat{s}_{t+2} = d_\phi(z_{t+1}) = d_\phi(f_\omega(s_1, ..., s_t, \hat{s}_{t+1})) \tag{2.4.4}$$

This operation is done K times, where K is a parameter representing the length of the future. [**32**] generalizes this method to a spatio-temporal information encoding where each image encoding is a tuple of encodings of different regions of the frame. In this work, we

only explore the case where the image encoding is the representation of the entire image, but we believe that the original method could contribute to better performance. The learning objective is to minimize the loss function $\mathcal{L}_Z$ given by:

$$\mathcal{L}_Z(\theta, \omega, \phi) = - \mathop{\mathbb{E}}_{o_{0:T}} \left[\frac{1}{K} \sum_{1 \leq k \leq K} L_{t+k} \right] \tag{2.4.5}$$

$$L_t = \log \frac{\texttt{sim}(\hat{s}_t, s_t)}{\texttt{sim}(\hat{s}_t, s_t) + \sum\limits_{t' \neq t} \texttt{sim}(\hat{s}_t, s_{t'})} \tag{2.4.6}$$

Indeed, the loss function is also a contrastive loss function where the positive pairs are the pairs $(\hat{s}_t, s_t)$ and the negative pairs are the pairs $(\hat{s}_t, s_{t'})_{t \neq t'}$ in a same video.

Chapter 3

Related work

Imitation from Observation is a sub-domain of Machine Learning that has been extensively studied in several papers and different techniques have emerged and demonstrated optimistic performances on a set of control continuous tasks. In this chapter, we present the main methods that have emerged, and the motivations behind these methods.

3.1. Imitation from observation

Imitation from observation problem originates from *Imitation Learning* (IL) problem [**38, 67, 65, 11**] which consists in training an agent using state-action demonstrations. In IL configuration, the demonstrations provided to the agent also contain the actions performed by the expert. A known method to solve this type of problem is the *Behavioral Cloning* (BC) [**60, 11, 65**], which consists in training a policy function to copy the decision process of the expert represented in the demonstrations. This function is a mapping function from the set of states of the system to the set of possible actions. The policy function is trained using supervised learning algorithms on the expert's demonstration dataset. During the test phase, the policy function is experimented on unencountered states. The supervised learning and a large amount of data favor the generalization to these new states. This approach is notably used by [**11**] for the learning of an autonomous car with the help of visual observations coming from several cameras on the car and of the different movements of the driver. The policy function is a CNN taking as input all the images from the different cameras.

Another IL approach consists in searching and building a reward function from the state-action sequences provided as a demonstration, it is the *Inverse Reinforcement Learning* approach [**1, 45, 55**]. This approach learns a function that efficiently evaluates each state-action pair provided by the agent at each step. The particularity of this approach compared to BC is that the properties of the MDP are exploited and any classical RL algorithm can

be used to train the agent with the learned reward function. [**1**] proposes in particular an algorithm that simultaneously trains a reward function and a policy function. The trained reward function is a differentiable parametric function that is optimized to approximate the supposed initial reward function that the policy expert tries to maximize intuitively. The imitation policy is trained to maximize the learned reward function at each update of this function using a classical RL algorithm. This method remains computationally expensive because each time the reward function is updated, a new optimal policy is learned. An IL method proposed to correct this computational cost using a GAN architecture [**27**].

Generative Adversarial Imitation Learning (GAIL) [**36**] proposes an adversarial learning method where the agent is a trajectory generator associated with a discriminator function evaluating how similar the agent's state transitions are to the transitions performed by the expert. The imitation policy is trained to produce transitions similar to the expert's and the discriminator function is trained to differentiate the expert's state transition (real data) from the agent's state transitions (fake data).

Although the Imitation Learning paradigm allows to train agents on tasks that do not have manually designed reward functions, this configuration of the problem remains quite far from the situation in which humans exploit their ability to imitate experts. The main problem here is the need to have the expert's actions in addition to the observations. This information greatly limits the type of data that can be exploited for learning, especially when only videos are available. We have at our disposal, through the internet, an incomparable set of videos of people performing various tasks. It is essential to find a way to make this data useful by getting rid of the need to have a vector description of the actions associated to each event.

Imitation from observation simplifies the configuration of Imitation learning by imposing the exploitation of state-only expert demonstrations. The initial dataset does not contain any information about the actions performed by the expert and it is impossible to access this information through the expert policy. The agent only has access to observations of the environment, and in the case of visual observations, the demonstrations are videos. Depending on the problem, they can be first-person-view demonstrations or third-person-view demonstrations. Similarly to IRL, one of the approaches to train agents with expert demonstrations is to learn a reward function. We distinguish two groups of approaches, the group of adversarial approaches and the group of reward learning approaches. Adversarial approaches [**80, 81**], consist in using a GAN-like architecture to train the agent (generator) and a discriminator function simultaneously. These approaches use the same logic of GAIL [**36**]. The discriminator function estimates the probability that the state transitions come from the learning agent. Similarly, the learning agent trains itself to fool the discriminator. The probability returned by the discriminator constitutes the agent's reward. [**81**] proposes a variation of this approach which consists in evaluating not a two-step state transition

but a transition on several steps. This variation is motivated by the objective of providing additional information on the expert's behavior through a small sequence of states. Reward learning methods [**40, 47, 12**] consist in searching for a consistent reward function for training the agent from the expert's demonstrations. One of these methods is [**47**] where the authors propose a method for learning a context translation function which, from the initial state of the system, predicts the sequence of next states of the system when the agent will act as an expert. The context translation function takes as input a random demonstration of an expert (the first episode), and the initial state of the system of another episode (the second episode). This function is trained to predict the sequence of next states of the second episode using the first episode and the initial state of the second episode. The objective is to imagine the future states in the second episode if the expert would execute this episode. Once this function is learned, it is then used to reward the learning agent at each new episode. At each step, the agent receives a reward measuring how close it is to the state imagined by the context translation function. Despite the ingenuity of this method, we notice through experiments that the context translation model tends to be imprecise on the intermediate states of the imagined state sequence, yet the intermediate states are used to lead the agent towards the final state.

With the same objective of learning a reward function, [**12**] proposes an approach similar to *Generative Adversarial Imitation from Observation* [**80**] which consists in rewarding the agent at each step according to the probability that the state transition is that of an expert. The method focuses on the estimation of this probability function and the training is done only on the expert's state sequence. Unlike GAIfO, there is no adversarial learning, once the probability function is trained, it is used to reward the agent in a second phase using a classical RL algorithm. The main difficulty of this approach is to estimate probabilities for transition states that do not belong to the distribution of expert states on which the probability function has been trained. To fix this issue, [**12**] proposes to add noise to the states in order to generalize the model to different state distributions and finally to limit the covariate shift problem [**64**] and to favor the out-of-distribution generalization [**54, 42**]. Another approach called ViRL [**9**] proposes to train the learning agent by directly imitating the behavior of an expert in a parallel environment. ViRL configures two environments with similar action properties but with different contexts and physical properties. The goal is to train an agent by observing an expert acting simultaneously in a different universe. ViRL uses a Siamese neural network that encodes the visual demonstration of the agent and the expert. The demonstration encodings of the agent and the expert are evaluated using a triplet loss function. In order to improve the generalization of the sequence encoding functions, ViRL applies augmentation processes on the videos. This video augmentation process allows to generate different positive pairs for the triplet loss function. When the demonstrations are

sequences of high-dimensional visual data, the question of the representation of the visual data in a low-dimensional feature space arises. Using image encodings allows to perform operations and comparisons on vectors encoding the most important features describing each image, and makes possible transfer learning [**58**] to many sub-tasks. In supervised classification, the existence of labels assigned to each image makes it possible to represent images as feature vectors via the correspondence between the distribution of vectors and the target label. However, for the case of visual frames from the demonstrations, we do not generally have labels associated to each frame. Moreover, assigning labels manually to each frame is a tedious and arduous task. This problem is notably the main subject of self-supervised learning which is a branch of machine learning that focuses on the learning and representation of unlabeled data.

3.2. Self-supervised learning of unlabeled data

In recent years, particular attention has been devoted to the self-supervised problem, because of the flexibility it offers in the exploitation of image and video datasets. Indeed, self-supervised learning focuses on the learning and representation of unlabeled data and several methods have been developed for this problem. The first group of methods is based on the resolution of a surrogate problem [**2, 10, 85, 20**] instead of the well-known classification or detection problems. In general, it consists in creating a pretext task from the initial dataset. The success of this pretext task is measured using an objective loss function defined on the unlabeled dataset. As an example, we can take the case of [**92**] which defines a grayscale image colorization task. Considering a dataset of colored images, [**92**] generates a dataset of grayscale versions of the images, and learns a CNN to generate the initial image with color.

Similarly, [**56**] proposes the pretext task of solving a puzzle by ordering different patches extracted from an image. A second group of approaches for this problem is the famous group of Contrastive learning approaches: [**78, 87, 15, 33**]. Considering a set of data with two (or more) samples belonging to the same distribution and others belonging to different distributions, Contrastive learning approach consists in training a model to identify the pair of samples from the same distribution (positive pair) among all the other possible pairs (negative pairs) by minimizing the probability that the negative pairs' items are from the same distribution [**29, 24**]. An example of this approach is *SimCLR* [**14**] which introduces a representation model from different augmented views of the images. [**14**] shows that image augmentation has a positive effect on the representation of images when properly combined with a contrastive loss function. For each image in the dataset, the images are randomly augmented using a stochastic augmentation function, and the augmented versions of the

images constitute positive pairs. A feature extraction function is then trained to extract similar features for each item of the positive pairs.

Adversarial approaches [**22, 23**] are also another group of self-supervised learning that exploit the GAN model [**27**] but by adding an encoder model. In the initial architecture of a GAN, we have a generator model which from a noise distribution generates data belonging to a precise target distribution. This generator model is trained adversarially with a discriminator model that estimates the probability that the generated data are fake or real. *Bidirectional GAN* (BiGAN) [**22**] proposes a modification of this model by adding an encoder model that generates noise vectors from real data. The learning objective is then to distinguish pairs of real noises and fake data from pairs of fake noises and real data. [**22**] shows that at the end of the training, the generator model and the encoder model form a stable data auto-encoder.

Another group of approaches exploits recent advances in language models and word embeddings [**8, 48**] as well as *Recurrent neural networks* (RNN) and *Attention models* [**37, 16, 84**] to encode data of other natures than sequential signals. Concerning image data, *iGPT* [**13**] proposes an image encoding model trained to predict missing pixels on images. [**13**] uses a *GPT-2* decoder model [**62**] to learn the remaining pixels of a partially-masked image and finally predict the missing pixels on the image. The image is transformed to a sequence of pixels (or small patches of pixels) in order to define an auto-regressive and a BERT objective [**19**].

Many other self-supervised learning methods have been developed for learning unlabeled high-dimensional data and the results obtained continue to be convincing. In our work, the learning and representation of images and videos represent one of the main challenges to identify agents' behaviors in demonstrative videos. In the rest of this work, we show how to exploit these methods to learn a reward function precise enough for the definition of a Markov decision process and the training of imitation agents.

Chapter 4

Method

We have presented recent advances in Reinforcement Learning, as well as the specifics of Imitation Learning and Imitation from Observation problems. We have also studied some methods of Self-supervised learning of images and videos. In this chapter, we combine all these knowledges together to explain and study our Imitation from Observation algorithm, and we present the different parameters of this algorithm that have an influence on the final behavior of the agent.

4.1. Overview

We aim to train an agent to perform a task only using a set of videos $\mathcal{D}_e$ of an expert performing the same task many times. Our proposal is based on the trajectory matching objective, which consists of training the agent to have a trajectory similar to that of the expert. In other words, the only information we seek to extract from the expert's videos is its behavior. Our algorithm is divided into two phases, an *Alignment Phase* and an *Interactive Phase*. During the Alignment Phase, we learn a sequence encoding function f_ω, which encodes sequences of frames of an agent trajectory, and an associated distance metric between two sequences of two agent trajectories. We also jointly train an image encoding function g_θ. Note that this image encoding function g_θ can also be replaced by a pre-trained encoder. We discuss the impact of this replacement in Chapter 5. In the Alignment Phase, we only use trajectories coming from two distributions: the distribution of expert's trajectories $O \sim p(\mathcal{D}_e)$, and a distribution of trajectories $O \sim p(\mathcal{D}_a)$ generated by a randomly defined policy function. During the Interactive Phase, we learn the agent policy π using a standard Reinforcement Learning (RL) algorithm [**75**]. To obtain the reward we use the learned distance (trained during the Alignment Phase) between sequence encodings of the expert and the current agent. Critically, we fine-tune the image and sequence encoding functions with additional visual observations from our online interactions with the environment. In this

work, we use DrQ-v2 [**88**] RL algorithm to train the agent policy, π, and its associated q-value function, Q, though our method is agnostic to the choice of the RL algorithm. Specifically, in the Interactive Phase at each new episode of the agent, we sample an expert episode $O_e \sim p(\mathcal{D}_e)$ that acts as a reference expert policy. At each agent step, t, we evaluate the distance between $o_{0:t}$, the agent's trajectory until the step t, and $o_{e,0:t}$ the expert's trajectory until t by $d(o_{0:t}, o_{e,0:t}) = ||f_\omega(g_\theta(o_{0:t})) - f_\omega(g_\theta(o_{e,0:t}))||$. The reward assigned to the agent at this step is finally $r_t = -d(o_{0:t}, o_{e,0:t})$. The function f_ω from the Alignment Phase is trained only on expert trajectories and random trajectories, thus as the policy of the agent improves, the distribution of agent trajectories will change, making the distances produced inconsistent due to the shift of the trajectory distribution. In order to fix this in the Interactive Phase, we continue to update f_ω and g_θ on the new trajectories generated by the agent during the training of π. In Section 5.8, we discuss the necessity of the Alignment Phase before training the agent. The full training process is detailed in Algorithm 2.

4.2. Behavioral learning from videos

4.2.1. Image encoding

Let us consider $\mathcal{D}_e = \{O_e^i\}_{1 \leq i \leq N}$ the set of videos of the expert and $\mathcal{D}_a = \{O_a^i\}_{1 \leq i \leq N}$ a set of videos generated using a random policy function in the environment. We use the datasets $\mathcal{D}_e$ and $\mathcal{D}_a$ to train f_ω and g_θ. g_θ consists of two ConvNets g_{θ_1} and g_{θ_2} for each of the views L and ab. We associate to g_θ a decoder function q_γ which, based on a state $s = g_\theta(o)$, returns an estimate of the initial observation $\hat{o}$. g_θ and q_γ form an auto-encoder architecture. The image learning objective contains three loss terms: (1) $\mathcal{L}_{triplet}$ which enforces image encodings similarity of adjacent frames, (2) $\mathcal{L}_{ae}$ which permits the encoding to be decoded back to an image, and finally (3) we learn a contrastive multiview encoding following[**78**] and using $\mathcal{L}_S$ loss function as explained in Section 2.4.1. The first term $\mathcal{L}_{triplet}$ compares the distance between an anchor image o, a positive image o_p temporally close to the anchor, and a negative image o_n distant from the anchor image in the video:

$$\mathcal{L}_{triplet}(\theta) = ||s - s_p||^2 + \max(\rho - ||s - s_n||^2, 0) \tag{4.2.1}$$

where $s = g_\theta(o)$, $s_p = g_\theta(o_p)$. This term allows to create a distance between frames from the same sequence in order to avoid that frames from the same sequence collapse towards the same encoding vector. In order to ensure consistency and non-degeneracy of the vectors returned by g_θ, we need to make sure that each state vector s can reconstruct the initial image o. The training objective of the autoencoder is then to minimize the loss function $\mathcal{L}_{ae}$

given by:

$$\mathcal{L}_{ae}(\theta, \gamma) = ||o - q_\gamma(g_\theta(o))||^2 \tag{4.2.2}$$

Finally, we incorporate CMC objective to enrich the visual representations following [**78**] and presented in Section 2.4.1. Specifically we consider the color space transformation $\{R, G, B\} \rightarrow \{L, ab\}$ where the component L is the view 1,v_1, processed by g_{θ_1}, and the component ab is the view 2,v_2, processed by g_{θ_2}. The final encoding s of an image o is given by:

$$s = g_\theta(o) = [g_{\theta_1}(v_1), g_{\theta_2}(v_2)] \tag{4.2.3}$$

Following the formulation [**78**], we align the different views using a contrastive learning loss denoted $\mathcal{L}_S$. This process is illustrated in Figure 4.1. Overall the objective function to minimize for the training of the image encoder is $\mathcal{L}_{frame}$ given by:

$$\mathcal{L}_{frame}(\theta, \gamma) = \mathcal{L}_S(\theta) + \mathcal{L}_{triplet}(\theta) + \mathcal{L}_{ae}(\theta, \gamma) \tag{4.2.4}$$

4.2.2. Sequence encoding

The training of the sequence encoder function f_ω requires to consider two objectives: the encoding of the information necessary to predict the next states with the loss function $\mathcal{L}_Z$ described in Equation 2.4.5; the separation of the distributions $O \sim p(\mathcal{D}_e)$ and $O \sim p(\mathcal{D}_a)$. In order bring semantically closer sequences coming from the same distribution, we use another contrastive loss that applies to the sequence encoding vectors z returned by f_ω. Thus for a sequence O belonging to one of the two trajectory distributions, we consider a positive sequence O_p of the same distribution and k negative samples $\{O_{n,i}\}_{1 \leq i \leq k}$ of different distributions. The objective function to be minimized is $\mathcal{L}_O$:

$$\mathcal{L}_O(\theta, \omega) = - \mathop{\mathbb{E}}_{(O, O_p, \{O_{n,i}\})} \left[\log \frac{\texttt{sim}(z, z_p)}{\texttt{sim}(z, z_p) + \sum_i \texttt{sim}(z, z_{n,i})} \right] \tag{4.2.5}$$

$$z = f_\omega(g_\theta(o_0), \ldots, g_\theta(o_T)) \tag{4.2.6}$$

The objective function that we seek to minimize to train the sequence encoder function is $\mathcal{L}_{seq}$ given by:

$$\mathcal{L}_{seq}(\theta, \omega, \phi) = \mathcal{L}_Z(\theta, \omega, \phi) + \mathcal{L}_O(\theta, \omega) \tag{4.2.7}$$

Finally, for a triplet $(O, O_p, \{O_{n,i}\}_{1 \leq i \leq k}$, the frames used to evaluate $\mathcal{L}_{frame}$ come from the sequences of the sample, which allows us to compute the objective loss function $\mathcal{L}$:

$$\mathcal{L}(\theta, \gamma, \omega, \phi) = \mathcal{L}_{frame}(\theta, \gamma) + \mathcal{L}_{seq}(\theta, \omega, \phi) \tag{4.2.8}$$

The complete evaluation scheme of the functions g, f and p is shown in Figure 4.1.

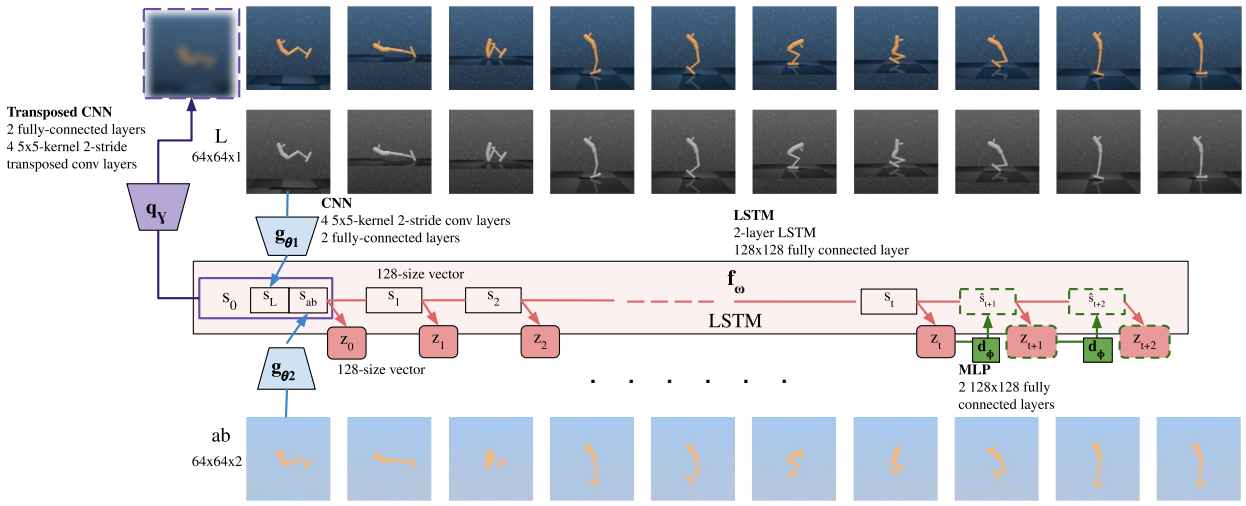


Fig. 4.1. Training architecture of the imitation functions. For each episode, the video is decomposed in the Lab color space, constituting the L and ab views. Each frame is encoded by g_θ and decoded by q_γ. The resulting state sequence $s_0, ..., s_t$ is encoded using the LSTM f_ω to provide the sequence encoding z_t. z_t is then processed by d_ϕ and f_ω to predict future image encodings.

4.3. Training: Imitation from observation

We use DrQ-v2 algorithm [**88**], a DDPG-based RL algorithm using stack of successive visual observations as input, to train the agent policy π and its associated q-value function Q (see Section 2.1.6). At the end of the Alignment phase, the sequence encoding function f_ω and the image encoding function g_θ are trained on an expert trajectory dataset $\mathcal{D}_e$, and a second trajectory dataset $\mathcal{D}_a$ generated with a randomly defined policy. This training allows to bootstrap the encoding functions on some trajectory distributions in order to provide meaningful rewards from the beginning of the Interactive phase. These encoding functions allow us to define a distance function d which evaluates the similarity between the agent behaviors present in two sequences of observation:

$$d(o_{0:t}, o_{e,0:t}) = ||f_\omega(g_\theta(o_{0:t})) - f_\omega(g_\theta(o_{e,0:t}))|| \tag{4.3.1}$$

Just like [**9, 47**], as illustrated in Figure 4.2, the reward assigned to the agent at step t is finally r_t:

$$r_t = -d(o_{0:t}, o_{e,0:t}) \tag{4.3.2}$$

This reward function implicitly evaluates whether, for each step, the agent tries to reach the same goal as the expert despite the difference in context. At each new training episode of the agent, we sample an exert episode $O_e \sim p(\mathcal{D}_e)$ that will be used to evaluate the trajectory of the agent and compute the rewards.

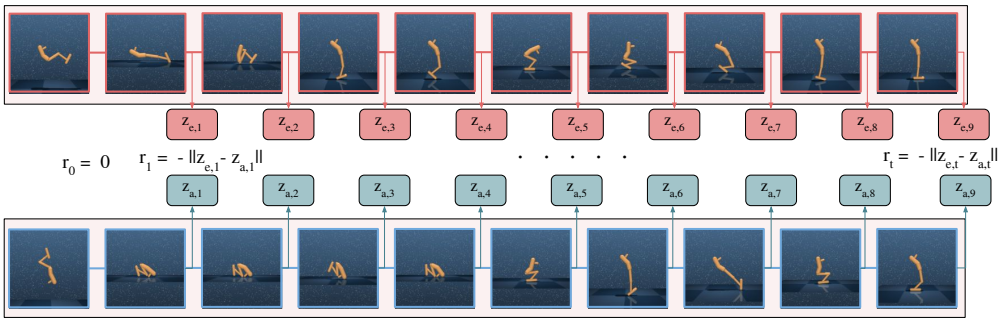


Fig. 4.2. Principle of agent rewarding. At each step, the agent's trajectory is encoded by f_ω and g_θ to produce z_t. This operation is also done with an expert video sampled from the set of expert video at the beginning of the episode. The reward returned to the agent at time t is the euclidean distance between z_t and $z_{e,t}$.

Note that in our method, we do not penalize the agent when s_t is distant from $s_{e,t}$ as does [**9**]. The reason is that the agent and the expert are not forced to have similar observation encodings, but only sequences that reflect the same goal and intention. Two sequences can be visually different but encode the same goal.

Note also that the function f_ω being trained on the distributions $O \sim \mathcal{D}_e$ and $O \sim P(o_0, ..., o_T|\pi_{random})$, it is not consistent for the other trajectories generated by the policy π during the evolution of its parameters. To fix this problem, during the Interactive phase, we train g_θ and f_ω progressively on the new trajectories generated by the agent's policy π during the training of its parameters. During our experiments, we found that the frequency and the number of training step during which the encoding functions are trained has an impact on the learning of π. During the Interactive phase, the enconding functions are updated after every N_{update} in order to create some training delay with respect to π and Q (which are updated at each step). This parameter reduces the over-training of the encoding functions, and prevents the learned distance function from keeping a large gap between the agent trajectories and the expert trajectories. During this same phase, we stop updating the encoding functions after N_{train} training steps. After this number of steps, the parameters θ and ω are frozen and only π and Q continue to be updated with DrQ-v2. Algorithm 2 presents the steps of the method in pseudo-code. The size of the set $\mathcal{D}_a$ of agent trajectories is also limited to a certain number in order to keep the most recent trajectories for the training phase of the agent and the encoders.

4.4. Encoding-based agent training

As presented in Section 2.1.6, DrQ-v2 [**88**] proposes a policy function π and a q-value function Q that share a same image encoding function that we name E (a convolutional neural

Algorithm 2: Imitation from observation with bootstrapped contrastive learning

$D_e = \{O_e^i\}_{1 \leq i \leq N} = \{(o_{e,0}^i, o_{e,1}^i, ..., o_{e,T}^i)\}_{1 \leq i \leq N}$
Initialize f_ω, g_θ, q_γ, d_ϕ
// Alignment Phase: Training f, g
while $k \leq N_{pretrain}$ **do**
  $\{O_e^i\}_{1 \leq i \leq n} \leftarrow sample(D_e)$
  $\{O^i\}_{1 \leq i \leq n} \leftarrow \pi_{random}(env)$
  Eval. $\mathcal{L}(\theta, \gamma, \omega, \phi)$ with $(\{O_e^i\}, \{O^i\})$
  Optimize θ, γ, ω, ϕ
end
// Interactive Phase: Training π, Q, f, g
Initialize $D_a = \{\}$
Initialize $\mathcal{D} = \{\}$, Q, π // Replay buffer, Q-value function, policy
while $step \leq N_\pi$ **do**
  $O_e \leftarrow sample(D_e)$
  $o_0 \leftarrow env.reset()$
  for $t \in 0, .., T-1$ **do**
    $a_t \leftarrow \pi(o_t)$
    $o_{t+1} \leftarrow env.step(a_t)$
    $r_{t+1} \leftarrow -||f(o_{0:t+1}) - f(o_{e,0:t+1})||$
    Save $(o_t, a_t, o_{t+1}, r_{t+1})$ into $\mathcal{D}$
    $(o, a, o', r) \leftarrow sample(\mathcal{D})$
    Using DrQ-v2 update Q,π with (o, a, o', r)
  end
  if $(step \leq N_{train})$ and
  $(step \mod N_{update} = 0)$ **then**
    Save $O = \{o_{0:t+1}\}$ into D_a
    $\{O_e^i\}_{1 \leq i \leq n} \leftarrow sample(D_e)$
    $\{O^i\}_{1 \leq i \leq n} \leftarrow sample(D_a)$
    Eval. $\mathcal{L}(\theta, \gamma, \omega, \phi)$ with $(\{O_e^i\}, \{O^i\})$
    Optimize θ, γ, ω, ϕ
  end
end

netwrok). Our current approach consider g_θ, an image encoding function that is jointly learned with the sequence encoding function. We propose a variation of our algorithm which, instead of training π and Q jointly with E, exploits the image encodings directly returned by g_θ. This choice has the advantage of reducing the computational cost of training E, and of exploiting knowledge transfer from g_θ to pi and Q.

We therefore propose a variation of the Algorithm 2 that uses the image encodings returned by g_θ as observations for the agent. We call this variant of the agent the *encoding-based agent*. Knowing that the stability of the distribution of the observations received by the agent is important, it is primordial that the image encodings keep the same distribution

during the whole training of the agent. For this reason, in this variation of the algorithm training an encoding-based agent, we freeze the θ parameter of g_θ so that this function does not evolve. Knowing that the sequence encoding function f_ω keeps the same utility, it is not necessary to freeze its parameters. During the Interactive phase, only $\mathcal{L}_{seq}$ is evaluated and only the functions f_ω and d_ϕ are updated.

We know that modern RL methods that train agents using low dimensional environment states can achieve surprising performance during task learning. These environment states are usually accurate measures of the conditions of the environment at a given time (joint angles, distances between elements, direction of movements, etc.). With this variation of our algorithm, we get closer to this condition by returning low-dimensional observation encodings to π and Q from which they can train. The main difficulty of this approach is the relatively low accuracy of the encodings we return. It will then be necessary to know if the observation encodings used as environment states are accurate enough to achieve similar or better performances than our initial algorithm. We explore the performances of this algorithm in Chapter 5.

Chapter 5

Experiments and results

We have a clear idea of the IfO approach that we propose in this thesis. In this chapter, we study the behavior of our proposal through the different components that constitute it. We study the different hyper-parameters of our algorithm, while bringing interpretations to results. We evaluate the necessity of the Alignment phase in the algorithm, as well as the contribution of pre-trained Convolutional networks.

5.1. Overview

We evaluate our method using a diverse set of continuous control tasks including 4 tasks (Reacher Hard, Finger Turn Easy, Hopper Stand, and Walker Run) from the DM Control Suite [**77**] and 3 tasks (Button Press, Plate Slide, Drawer Close) from the Meta-world environment [**89**]. The DM Control tasks require our agent to learn to coordinate multiple torque-controlled actuators. The Meta-world end-effector is controlled in a 3DOF task space with a 1DOF parallel jaw gripper modeled on a Sawyer Robot Arm. This environment setting requires our agent to learn to manipulate external objects with complex physical interactions with the world. In these manipulation tasks, we now have to model the robot and object-object interactions, such as in the case of Button Press and Drawer Close where the constrained object joint must be activated along a single axis. Below we describe the network architecture, details of our training procedures, and the results on these two datasets. All episode trajectories begin from randomized starting states of the robot and interactive objects (when applicable).

5.2. Network architectures

Our image encoder function g_θ consists of a pair of convolutional neural networks g_{θ_1} and g_{θ_2}. These models are of similar architectures, but the number of input channels is 1 for

Table 5.1. Architecture of convolutional neural networks g_{θ_1} and g_{theta_2}. X_{in} is the size of the input image. X_{out} is the size of the output matrix of each layer. C is the number of channels of the output matrix at each layer. K is the size of the convolution kernel at each layer. S is the stride of the convolution operation. P is the padding added initially to the input matrix.

Encoder network						
Layer	X_{in}	C	K	S	P	X_{out}
Image	64×64	1,2	-	-	-	-
ConvNet + BatchNorm + LeakyReLU	-	64	5	2	0	30×30
ConvNet + BatchNorm + LeakyReLU	-	128	5	2	0	13×13
ConvNet + BatchNorm + LeakyReLU	-	256	5	2	0	5×5
ConvNet + BatchNorm + LeakyReLU	-	512	5	2	0	1×1
ConvNet + BatchNorm + LeakyReLU	-	512	1	1	0	1×1
ConvNet	-	128	1	1	0	1×1

g_{θ_1} and 2 for g_{θ_2}. We re-use the encoder architecture proposed by [**47**]. The architecture is a succession of four 5×5 stride-2 convolutional layers with 64, 128, 256, and 512 filters. These convolutional layers are followed by two 1×1 convolutional layers of 512 and 128 filters. All layers before the last are followed by BatchNorm [**39**] and a Leaky ReLU [**34**] activation function ($leak = 2$). The image decoding function q_γ has an inverse architecture to the encoder, except that transposed convolutional layers are employed for the last 4 layers. Additional network details are presented in Tables 5.1 and 5.2. The sequence encoder function, f_ω, is a 2-layer LSTM with 128 as input and output sizes. The output sequence is followed by a fully connected layer with input and output size of 128. The next-state predictor is a sequence of two 128×128 fully connected layers with Leaky ReLU activation between the two layers. We employ an image encoding function, g_θ, to learn a latent representation from high-dimensional observations. This function is trained with a self-supervised method that combines different surrogate objectives. However, much modern research [**20, 85, 66**] has shown the benefit of using pre-trained or foundation models trained on a large corpus for downstream tasks. We illustrate the opportunity of using this strategy with our setup by employing EfficientNet [**78**], a convolutional model trained on ImageNet, as a backbone image encoder. In this setting, the function g_θ is composed of the backbone model with the weights frozen followed by a 3-layer multilayer perceptron (MLP) of size $1280 \times 512 \times 128$ with trainable parameters. Though we utilize EfficientNet-B0 as a backbone, there are many reasonable choices for pre-trained encoders.

Table 5.2. Architecture of the image decoding function q_γ based on transposed convolution operations. X_{in} is the size of the input image. X_{out} is the size of the output matrix of each layer. C is the number of channels of the output matrix at each layer. K is the size of the convolution kernel at each layer. S is the stride of the convolution operation. OP is the padding added to the output matrix. The padding added to the input matrix is always 0.

Decoder network						
Layer	X_{in}	C	K	S	OP	X_{out}
Image encoding	1×1	128	-	-	-	-
ConvNet + LeakyReLU	-	512	1	1	0	1×1
ConvNet + BatchNorm + LeakyReLU	-	512	1	1	0	1×1
TransConvNet + BatchNorm + LeakyReLU	-	256	5	2	0	5×5
TransConvNet + BatchNorm + LeakyReLU	-	128	5	2	0	13×13
TransConvNet + BatchNorm + LeakyReLU	-	64	5	2	1	30×30
TransConvNet	-	3	5	2	1	64×64

5.3. Training: Alignment Phase

For each task, we build a dataset of expert episodic trajectories with colored image observations at each timestep by running a trained RL agent and rendering visual images of the environment as needed. We also build a second dataset of trajectories with a randomly defined policy using the same process. In this work, we employ DrQ-v2 agents as our experts for imitation, though any reasonable expert (image-based RL, planning agents, human, or animal experts) could be used since we do not need access to the state of the agent. In the case, where we train an image encoder from scratch (BootIfOL), the demonstration datasets $\mathcal{D}_e$ and $\mathcal{D}_a$, of size $5000 \times 51 \times 64 \times 64$ (trajectory $\times$ timestep $\times$ height $\times$ width), are used as input. In Eff-BootIfOL, where we leverage large-scale pre-training, we utilize smaller datasets of demonstrations for finetuning: $1500 \times 61 \times 224 \times 224$ (trajectory $\times$ timestep $\times$ height $\times$ width).

5.4. Training: Interactive Phase

The parameters of the encoding functions are updated during $N_{\text{train}} = 375K$ training steps of the Interactive Phase. After these training steps, the parameters of the encoding functions are frozen. Our RL agent training is performed over $N_\pi = 1550K$ training steps. We employ DrQ-v2 [**88**], an image-based RL agent based off of the DDPG [**46**] architecture, as our RL policy learner with default hyperparameters. Additional hyperparameters are available in Table 5.3.

Table 5.3. Hyperparameters of Algorithm 2

Parameter	Value
Number of expert trajectories (BootIfOL) (N)	5000
Number of expert trajectories (Eff-BootIfOL) (N)	1500
Number of frames in each trajectory (BootIfOL) (T)	61
Number of frames in each trajectory (Eff-BootIfOL) (T)	51
Size of images (BootIfOL)	64×64
Size of images (Eff-BootIfOL)	224×224
Number epochs during the Alignment Phase ($N_{pretrain}$)	8000
Number of videos per batch (n)	16×2 (pairs)
Total number of training steps during which encoders are trained (N_{train})	375K
Periodicity of encoder parameter update (in training step) (N_{update})	50
Total number of agent training steps (N_π)	1.55 M
Learning rate	1×10^{-4}
Optimizer	Adam [**41**]

5.5. General results

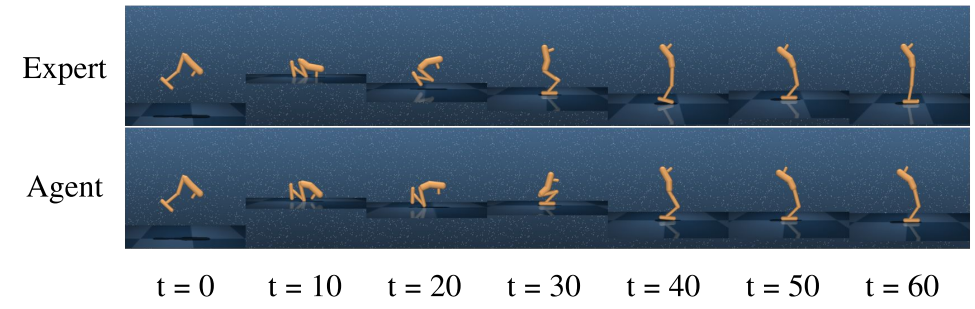
We perform the training on the Reacher Hard, Finger Turn Easy, Hopper Stand, and Walker Run tasks from the Deepmind Control Suite [**77**]. We compare our algorithm to the Context Translation (CT) [**47**] method for IfO by varying the number n of expert trajectory samples used at each training episode to predict the CT agent trajectory. We also baseline against ViRL [**9**] and GAIfO [**80**]. Demonstrated across 4 tasks in DM Control, our method shows strong performance in learning to complete tasks given visual demonstrations as shown in Table 5.4, where GAIfO showed poor performance on all tasks. We also show promising results in complex visual scenes by employing a pre-trained backbone on Meta-world environments as shown in Table 5.5. Average episodic returns over RL training in the Interactive Phase are presented in Figure 5.2. We show visual examples of our agents acting in Hopper Stand and Drawer Close tasks in Figure 5.1. We present the examples for other tasks in Figure 5.3. We observe that in tasks such as Walker Run, where a great degree of coordination between joints is necessary to control the robot, IfO agents struggle to reach even 10% of the expert's reward (see Table 5.4).

5.6. Results of Evaluation on Meta-World

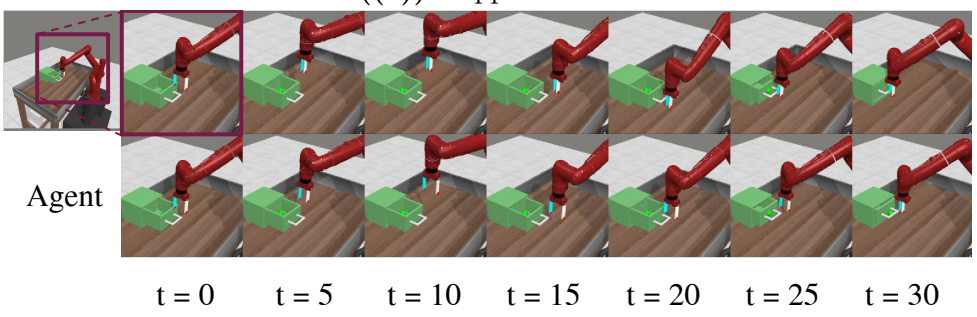
Considering the Meta-world environment [**89**], we study precisely the importance of a pre-trained convolutional network as a feature extraction model. The visual complexity offered by Meta-world tasks allows us to evaluate the relevance of an accurate image encoder during

Table 5.4. Evaluation of the average return over 500-step episodes of agents trained with the Context translation (CT) [**47**] and ViRL [**9**] algorithms. We evaluate the agents on the Reacher Hard, Finger Turn Easy, Hopper Stand, and Walker Run tasks. For 3 of the environments, our approach exceeds the existing methods by a wide margin. For Reacher Hard, we are able to achieve rewards on par with the Expert policy, while our comparison methods completely fail to learn good policies. Though this is a fairly simple control problem (a visual version of inverse kinematics), the distribution of starting states and goals is fairly large compared to other tasks we look at. Our technique of training with failure demonstrations is particularly advantageous in this setting as we see more of the state space.

Agent	Avg. return			
	Reacher Hard	Finger turn easy	Hopper Stand	Walker Run
Expert	850.52 ± 315.70	893.44 ± 219.75	880.93 ± 76.89	789.77 ± 17.25
BootIfOL (ours)	$\mathbf{843.16 \pm 274.84}$	199.76 ± 399.02	$\mathbf{651.33 \pm 362.95}$	$\mathbf{79.50 \pm 1.25}$
ViRL	0.44 ± 1.27	160.2 ± 366.52	516.27 ± 417.21	29.31 ± 24.72
CT (n=10)	97.84 ± 254.52	$\mathbf{238.92 \pm 422.07}$	1.42 ± 3.22	25.15 ± 18.71
CT (n=1)	38.04 ± 127.53	199.48 ± 396.99	1.01 ± 1.71	52.42 ± 34.28
Random	6.64 ± 16.94	92.6 ± 202.68	2.44 ± 5.19	29.93 ± 4.52

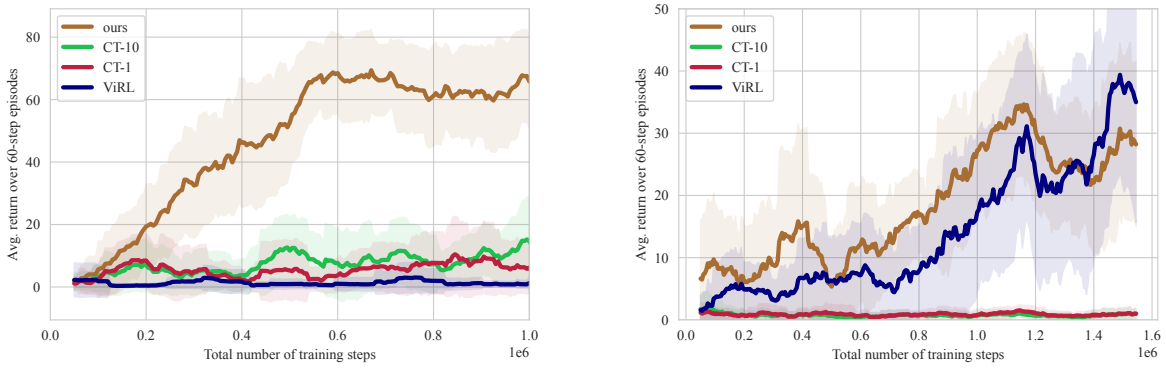


((a)) Hopper Stand



((b)) Drawer Close

Fig. 5.1. Comparison of the actions taken between an expert (top row) policy and imitation agent (bottom) learned using our proposal. We show learned agents in Hopper Stand and Drawer Close with the same initial conditions. Observe that for Hopper Stand the agent behavior of our learned agent is very similar to that of an expert. For Drawer Close although the learned agent takes a different trajectory than the expert(e.g. keeping the gripper wider open) it is able to solve the task.



((a)) Reacher Hard

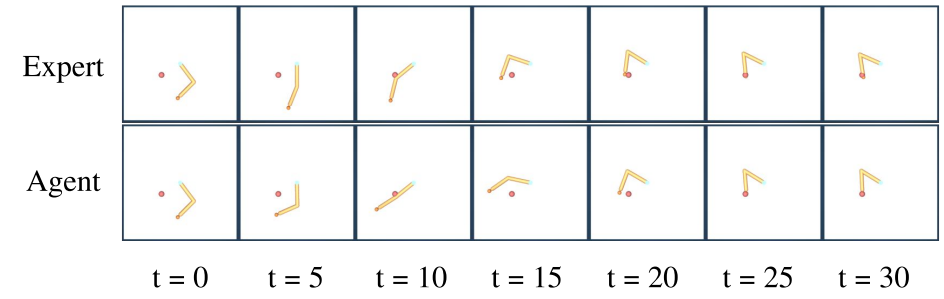
((b)) Hopper Stand

Fig. 5.2. Average returns throughout agent training in Interactive Phase compared to CT and ViRL agents on Reacher Hard and Hopper Stand tasks. We observe that the agent progresses quickly in both cases as compared to other baselines. Although ViRL is able to do well in Hopper Stand, it is unable to tackle all environments (e.g. Reacher Hard).

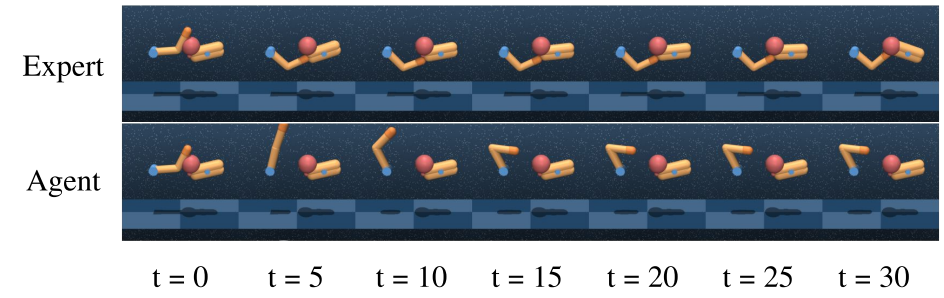
Table 5.5. Evaluation of the average return over 500-step episodes of our agent (*BootIfOL*) where the encoder was trained from scratch and an agent which exploited EfficientNet-B [**66**]0 as a backbone model (*Eff-BootIfOL*). We evaluate these agents on the manipulation tasks: Button Press, Plate Slide, and Drawer Close in the Meta-world simulator [**89**].

Agent	Avg. return		
	Button Press	Plate Slide	Drawer Close
Expert	556.06 ± 6.46	734.56 ± 172.51	4223.65 ± 21.14
Eff-BootIfOL (ours)	$\mathbf{434.39 \pm 139.74}$	340.45 ± 46.56	$\mathbf{3949.90 \pm 681.42}$
BootIfOL (ours)	146.56 ± 17.71	$\mathbf{342.40 \pm 93.08}$	2847.10 ± 1679.56
Random	153.29 ± 51.68	209.12 ± 87.38	409.07 ± 1288.89

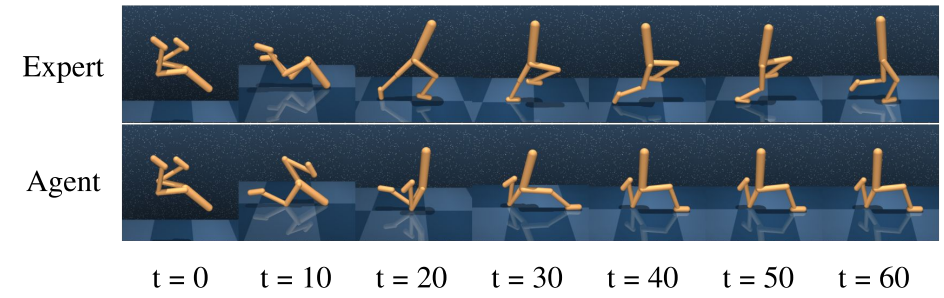
the training of the encoding functions. We evaluate our new agent, trained with a learned distance metric using a pre-trained image encoding, as described in the Network Architectures section (5.2), on the tasks Button Press, Drawer Close, and Plate Slide of the Meta-world environment. In Table 5.5, we compare the average return of this agent with our initial agent. The results in Table 5.5 demonstrate the importance of the image encoding function for the effectiveness of the learned distance metric. In particular, these results show that the success of the reward function learning depends on the level of accuracy in the interpretation of each frame. We observe a particularly weak result concerning the Plate Slide task. We hypothesize that this failure is due to a difficulty related to the sequence encoding function, which has a different role. Despite the accuracy of the feature extractor, the task remains difficult to imitate by our agent.



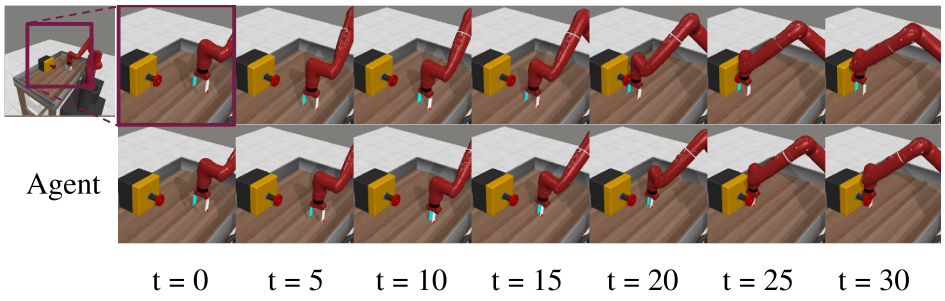
((a)) Reacher Hard



((b)) Finger Turn



((c)) Walker Run



((d)) Button Press

Fig. 5.3. Comparison of actions taken between an expert policy (top row) and an imitation agent policy learned using our proposal. We show these agents acting in the Reacher Hard, Finger Turn, Walker Run and Button Press tasks. Note also that for all these tasks, the expert and the agent are placed in identical initial conditions.

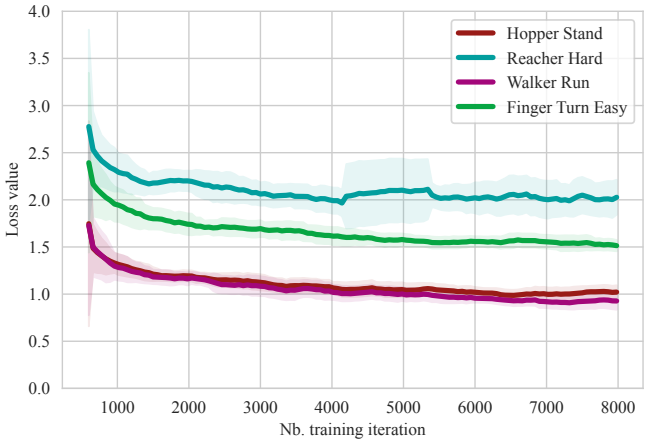


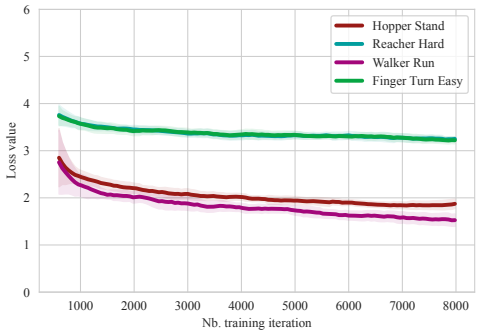
Fig. 5.4. Evolution of the loss term $\mathcal{L}$ during the Alignment phase. We evaluate these values on the trajectories generated from the Hopper Stand, Reacher Hard, Walker Run and Finger Turn Easy tasks.

5.7. Trajectory encoding learning

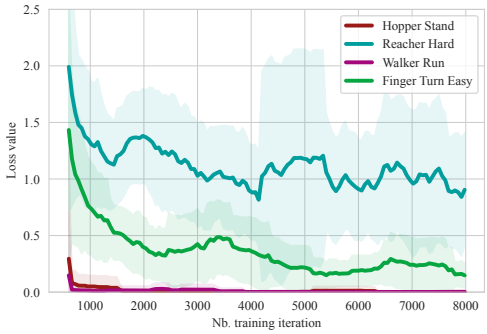
Another essential element to observe in this algorithm is the behavior of the objective loss functions linked to the training of the f_ω and g_θ functions. During the Alignment phase, we aim to train the functions f_ω and g_θ to encode the expert trajectories and the random trajectories. This training is led by the loss terms $\mathcal{L}_Z$, $\mathcal{L}_{seq}$, $\mathcal{L}_{ae}$, $\mathcal{L}_{triplet}$ and $\mathcal{L}_S$, each of which brings a learning constraint, limiting the deviance and collapse of the encoding functions. We present in Figure 5.4 the evolution of the final loss function $\mathcal{L}$ during the Alignment phase estimated on the evaluation dataset.

This evaluation dataset consists of 400 expert videos and 400 random videos. We evaluate this feature on 4 different tasks from the Deepmind control suite [**77**]: Hopper Stand, Reacher Hard, Finger Turn Easy and Walker Run. This objective loss function evolves according to our expectations. We detail the evolution of the intermediary loss terms in Figure 5.5.

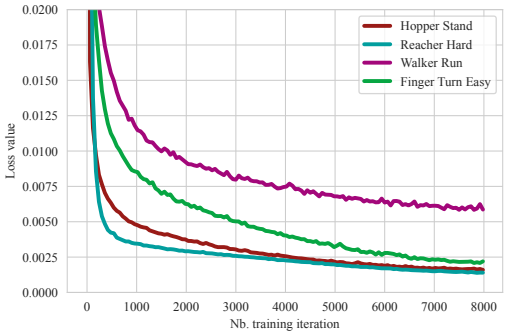
In the Interactive phase, during the training of the imitation agent, the encoding functions are progressively trained on the new trajectories generated by the agent and on the initial expert trajectories. We present in Figure 5.6, the evolution of $\mathcal{L}_{seq}$ function during the Interactive phase. We can see a slight tendency for $\mathcal{L}_{seq}$ to increase with time. This effect is due to the sequence encoding function, which has more and more difficulties to distinguish expert sequences from agent sequences. Knowing that the agent policy π improves progressively with the rewards which are provided to it by the sequence encoding function, π generates trajectories progressively similar to that of the expert. This progress is the cause of the



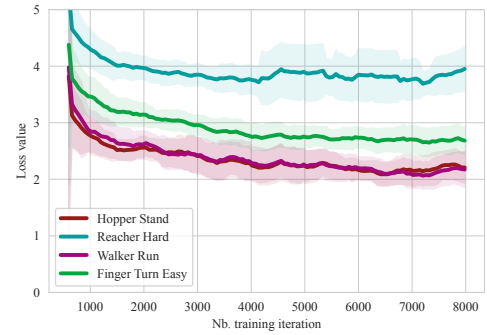
((a)) Evaluation of $\mathcal{L}_Z$



((b)) Evaluation of $\mathcal{L}_{seq}$



((c)) Evaluation of $\mathcal{L}_{ae}$



((d)) Evaluation of $\mathcal{L}_{triplet}$

Fig. 5.5. Evolution of the loss terms $\mathcal{L}_Z$, $\mathcal{L}_{seq}$, $\mathcal{L}_{ae}$ and $\mathcal{L}_{triplet}$ during the Alignment phase, during which the encoding functions g_θ and f_ω are trained over expert trajectories and randomly generated trajectories.

increase of $\mathcal{L}_{seq}$. This observation allows us to conclude the sink of the agent's distribution into that of the expert.

5.8. Ablating the Alignment Phase

In our algorithm, we integrate the Alignment Phase, described in Section 4.1, allowing us to bootstrap the encoding functions f and g on two trajectory distributions: (1) Expert's trajectories, (2) random trajectories. This particularity is not present in most methods such as GAN-like approaches. We hypothesize that providing, from the beginning of the agent's training, significant and consistent rewards offers an important gain on the search for the optimal imitation policy. We evaluate the importance of this step by removing the Alignment Phase in two ways: (1) the encoding functions are trained during 375K total training steps (standard case) in the Interactive Phase; (2) the encoding functions are trained during all the agent's training in the Interactive Phase (similarly to ViRL and GAIfO). We present the

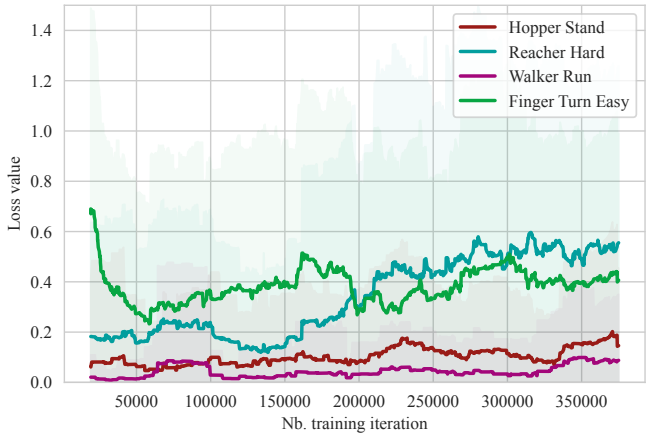


Fig. 5.6. Evaluation of the loss term $\mathcal{L}_{seq}$ during the Interactive phase. This loss is evaluated during N_{train} training steps during the update of f_ω and g_θ. We notice a slight increase of this term, and this is due to the progress of the agent policy function in generating trajectories similar to that of an expert policy.

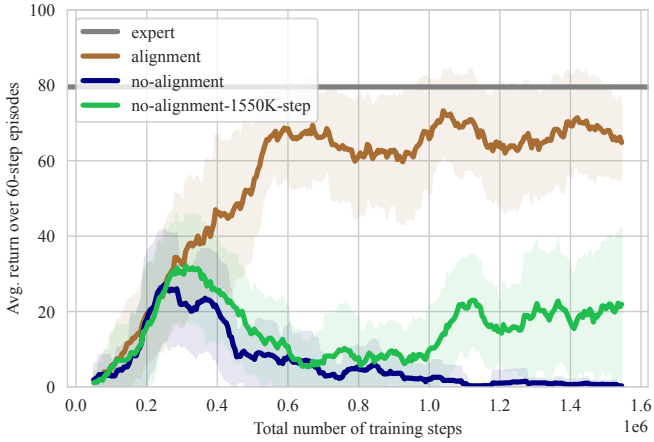


Fig. 5.7. Ablating the effect of the encoding functions' training in the Alignment Phase (described in Section 4.1) on the final performance of the agent on the Reacher Hard task. *alignment* is our initial model with the Alignment Phase executed; *no-alignment* and *no-alignment-1550K-step* are the models without execution of the Alignment Phase. In *no-alignment-1550K-step*, the encoding functions are updated continuously until the end of the agent's training.

results of this experiment in Figure 5.7. We observe that the use of the alignment phase leads to drastically better performance, highlighting this as a critical phase. Although training these for the full interactive phase can lead to some progress, it is not nearly as efficient as the bootstrapped approach that includes the Alignment Phase.

5.9. Ablating the use of a Learned Image-Encoder for RL

To date, in our experiments, the RL agent is learned through a dissociate policy function π and q-value function Q as proposed [**88**]. In this architecture, the policy function π is composed of a CNN E that internally encodes the observation images, followed by an MLP that feeds the image encodings to return actions. E is shared between the policy pi and the associated q value function Q, which feeds encoding-action pairs to return q-values. A natural question is whether we can directly link our learned image encoding function g_θ to pi and Q by dropping the CNN E. The interest of this approach is to transfer the learning of the g to the agent in order to learn directly the extracted features. We know that in modern RL approaches, using low-dimensional state vectors generally improves agent performance and efficiency, because of higher precision in the description of the state. We are interested in learning if our learned observation feature descriptions - optimized with a pixel-based reconstruction loss - provide enough information about the state to enable an RL policy to learn efficient and quality control policies without needing access to either the true simulator state or raw pixels. Specifically, we replace E with an MLP shared between π and Q. The image encodings returned by g are the input data of the MLP E. Note in this experiment we freeze the parameters of g_θ with respect to the RL update and call this ablation the *encoding-based agent*. We present in Figure 5.8 the average return of the original image-based agent and the encoding-based agent on the Reacher Hard task over training. This leads us to believe that the image encodings returned by g present biases that limit the understanding of the environment by the policy, thus preventing this policy from achieving performance similar to agents with access to image-based states directly. We suppose that this bias is because g and f are trained jointly with learning objectives that are not optimized for the policy function.

5.10. Effect of Encoder Training Length in Interactive Phase

In the *Interactive Phase*, we train the trajectory and image encodings for N_{train} steps before switching to only using a standard RL algorithm. We now study how the length of this phase affects the final reward reached by the agent. We evaluated our algorithm with N_{train} set to a period of 0, 100K, 375K, and 1550K total training steps. Updates to encoder parameters happen every $N_{update} = 50$ steps. The results of these experiments on the Reacher Hard task are presented in Figure 5.9. We observe that when the encoding functions are not updated during the Interactive Phase, learning does not proceed, as the rewards provided by the

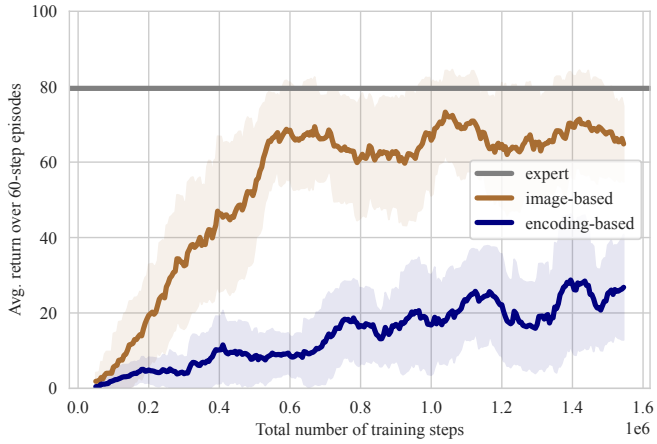


Fig. 5.8. Ablation studies. We show the average return of the agent on the Reacher Hard task over 5 episodes of 60 steps. We evaluate whether we can re-use the image encoding CNN from our imitation function for policy learning (encoding-based) or whether the RL agent should optimize a new image encoding network (image-based). We observe that attempting to use the encodings from BootIfOL directly in the policy network (encoding-based) degrades performance.

Alignment Phase are not sufficient. Increasing the number of N_{train} steps initially has a positive effect on average returns per episode, reaching close to the expert return per episode in the case of $N_{train} = 375$K steps. However, when the encoding functions are not frozen training during the Interactive Phase, the learned reward function becomes brittle due to changing state encoding. Thus we observe that $N_{train} = 1550$K steps eventually led to a collapse in the policy.

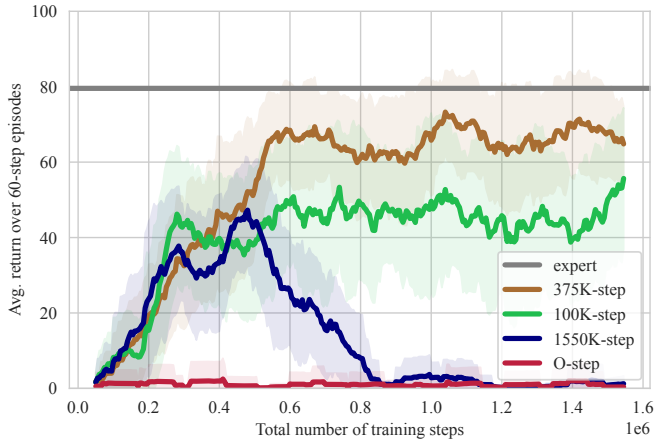


Fig. 5.9. Ablation studies. We show the average return of the agent on the Reacher Hard task over 5 episodes of 60 steps. We evaluate the agent with respect to the training duration of the encoding functions. If they are not trained after the Alignment Phase (*0-step*), the rewards are non-informative. Similarly, if we continue to train them as the agent begins to converge to a strong policy, they can degrade the reward signal.

Chapter 6

Conclusion and future work

We present a new method of Imitation from Observation that relies on the encoding of agent behavior using exemplar demonstrations from visual observations. This method uses self-supervised learning of states and sequences to progressively train a reward function for use by a reinforcement learning agent. We demonstrate the strength of this method on a set of 7 simulated robotic tasks which have access to a limited set of expert demonstrations. We note that our adopted problem framework: imitation from image observations, though important for its potential practical applications, is challenging from both the computational and observational perspectives for tasks that require precise control. In this paper, we explore this challenge by training our agents with access to information from task-specific and pre-trained image encoders and show that the fidelity of the encoding function is critically important for downstream control tasks. Our approach to increase the performance of the learned representation is to keep the data generated by the agent during the Interactive Phase.

During the experiments, we studied the evolution of the loss terms used to train the encoding functions, and we evaluated the final loss term during the Alignment and Interactive phases. At this point, one question that arises is the importance of each loss term. Indeed, in the definition of our objective loss function, all terms have the same importance and influence the final encoding function sequence at the same level. In order to complete this work, we leave in future work the analysis of the importance of the different components of the loss function.

During the Alignment and Interactive phases, we always used the same number of expert sequences and agent sequences. In practice, the number of sequences we use allows us to evaluate the feasibility of the method, and the ideal is to minimize this number. In two groups of experiments, we exploited 2×5000 and 2×1500 sequences, and this variation did

not show significant impacts. We believe that a thorough study of the minimum number of usable trajectories would clarify this point.

In this work, our expert trajectory dataset is composed of a set of trajectories that vary in terms of the initial condition of the agent and the position of objects in the scene. Although this context variation is significant, it remains very optimistic compared to the variations encountered in real applications. For future work, we think it would be important to bring randomness in each episode on several dimensions of the context. These variations can be in the decoration of the environment's background, the camera's angle of view, or the presence of other dynamic objects independent of the agent.

Finally, we have mainly studied in this work, continuous control tasks executed in fully observable environments. We believe that this work will contribute to the development of hierarchical reinforcement learning techniques in partially observable environments. Indeed, in these types of problems, the agent has to plan a set of tasks or reach a certain global goal. Since our proposal mainly studies continuous control tasks, it would be important to study how this could be adapted to more complex architectures, and how to exploit Imitation from Observation in environments that require exploration and discovery beforehand.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[2] Dosovitskiy Alexey, Philipp Fischer, Jost Tobias, Martin Riedmiller Springenberg, and Thomas Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 99, 2015.

[3] Ankesh Anand, Eugene Belilovsky, Kyle Kastner, Hugo Larochelle, and Aaron Courville. Blindfold baselines for embodied qa. *arXiv preprint arXiv:1811.05013*, 2018.

[4] Anonymous. Imitation from observation with bootstrapped contrastive learning. In *3rd Offline RL Workshop: Offline RL as a "Launchpad"*, 2022.

[5] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.

[6] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[7] Richard E Bellman and Stuart E Dreyfus. *Applied dynamic programming*, volume 2050. Princeton university press, 2015.

[8] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.

[9] Glen Berseth, Florian Golemo, and Christopher Pal. Towards learning to imitate from a single video demonstration. *arXiv preprint arXiv:1901.07186*, 2019.

[10] Piotr Bojanowski and Armand Joulin. Unsupervised learning by predicting noise. In *International Conference on Machine Learning*, pages 517–526. PMLR, 2017.

[11] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[12] Wei-Di Chang, Juan Camilo Gamboa Higuera, Scott Fujimoto, David Meger, and Gregory Dudek. Il-flow: Imitation learning from observation using normalizing flows. *arXiv preprint arXiv:2205.09251*, 2022.

[13] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International conference on machine learning*, pages 1691–1703. PMLR, 2020.

[14] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[15] Ting Chen, Calvin Luo, and Lala Li. Intriguing properties of contrastive losses. *Advances in Neural Information Processing Systems*, 34:11834–11845, 2021.

[16] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[17] Christine Connolly and Thomas Fleiss. A study of efficiency and accuracy in the transformation from rgb to cielab color space. *IEEE transactions on image processing*, 6(7):1046–1048, 1997.

[18] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–10, 2018.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[20] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015.

[21] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655. PMLR, 2014.

[22] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[23] Jeff Donahue and Karen Simonyan. Large scale adversarial representation learning. *Advances in neural information processing systems*, 32, 2019.

[24] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. *Advances in neural information processing systems*, 27, 2014.

[25] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[26] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[27] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[28] Daniel Gordon, Kiana Ehsani, Dieter Fox, and Ali Farhadi. Watching the world go by: Representation learning from unlabeled videos. *arXiv preprint arXiv:2003.07990*, 2020.

[29] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[30] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[31] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[32] Tengda Han, Weidi Xie, and Andrew Zisserman. Video representation learning by dense predictive coding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[33] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[35] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[36] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

[37] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[38] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

[39] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[40] Daiki Kimura, Subhajit Chaudhury, Ryuki Tachibana, and Sakyasingha Dasgupta. Internal model from observations for reward shaping. *arXiv preprint arXiv:1806.01267*, 2018.

[41] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[42] Polina Kirichenko, Pavel Izmailov, and Andrew G Wilson. Why normalizing flows fail to detect out-of-distribution data. *Advances in neural information processing systems*, 33:20578–20589, 2020.

[43] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[45] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. *Advances in neural information processing systems*, 24, 2011.

[46] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[47] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1118–1125. IEEE, 2018.

[48] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[49] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[50] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[51] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[52] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[53] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[54] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? *arXiv preprint arXiv:1810.09136*, 2018.

[55] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[56] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European conference on computer vision*, pages 69–84. Springer, 2016.

[57] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[58] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[59] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

[60] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[61] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[62] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[63] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European conference on machine learning*, pages 317–328. Springer, 2005.

[64] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

[65] Stéphane Ross, Geoffrey Gordon, and Drew 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*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[66] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[67] Stefan Schaal. Learning from demonstration. *Advances in neural information processing systems*, 9, 1996.

[68] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[69] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR, 2020.

[70] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[71] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1134–1141. IEEE, 2018.

[72] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[73] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[74] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 8, 1995.

[75] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[76] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[77] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[78] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *European conference on computer vision*, pages 776–794. Springer, 2020.

[79] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

[80] Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*, 2018.

[81] Faraz Torabi, Garrett Warnell, and Peter Stone. Imitation learning from video by leveraging proprioception. *arXiv preprint arXiv:1905.09335*, 2019.

[82] Faraz Torabi, Garrett Warnell, and Peter Stone. Recent advances in imitation learning from observation. *arXiv preprint arXiv:1905.13566*, 2019.

[83] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[84] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[85] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2015.

[86] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[87] Tete Xiao, Xiaolong Wang, Alexei A Efros, and Trevor Darrell. What should not be contrastive in contrastive learning. *arXiv preprint arXiv:2008.05659*, 2020.

[88] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

[89] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019.

[90] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pages 12310–12320. PMLR, 2021.

[91] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[92] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016.

[93] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pages 928–936, 2003.