Augmented Behavioral Cloning from Observation

Juarez Monteiro*[1], Nathan Gavenski†[1], Roger Granada*, Felipe Meneguzzi‡ and Rodrigo Barros‡
School of Technology, Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga, 6681, 90619-900, Porto Alegre, RS, Brazil
*{juarez.santos, roger.granada}@acad.pucrs.br, †nathan.gavenski@edu.pucrs.br
‡{felipe.meneguzzi, rodrigo.barros}@pucrs.br

***Abstract*—Imitation from observation is a computational technique that teaches an agent on how to mimic the behavior of an expert by observing only the sequence of states from the expert demonstrations. Recent approaches learn the inverse dynamics of the environment and an imitation policy by interleaving epochs of both models while changing the demonstration data. However, such approaches often get stuck into sub-optimal solutions that are distant from the expert, limiting their imitation effectiveness. We address this problem with a novel approach that overcomes the problem of reaching bad local minima by exploring: (i) a self-attention mechanism that better captures global features of the states; and (ii) a sampling strategy that regulates the observations that are used for learning. We show empirically that our approach outperforms the state-of-the-art approaches in four different environments by a large margin.**

***Index Terms*—Imitation Learning, Behavioral Cloning, Learning from Demonstration, Deep Learning**

I. Introduction

Humans can learn how to perform certain activities by observing other humans. This ability of imitating allows humans to transfer the knowledge from demonstrations to the task at hand, despite differences in environment or objects used in the demonstration [2]. For example, one can learn how to cook by watching videos online, even if the stove and pans are different from the ones in the video. The advance in technology and the rising demand for intelligent applications have increased the need for artificial agents that are capable of imitating a human demonstrator. Research on imitation learning is motivated by the ease with which humans transfer their knowledge through demonstration rather than articulating it in a way that the interested learner may understand [5].

Imitation learning, also referred to as *learning from demonstration* (LfD), refers to the task of artificial autonomous agent acquiring skills or behaviors from an expert by learning from its demonstrations [6, 7]. A natural way of imparting knowledge by an expert is to provide demonstrations for the desired behavior that the learner then emulates [3]. Unlike humans that can learn without having direct access to the actions executed in a demonstration [1], classical approaches of LfD use labeled actions in order to imitate the expert behavior. Such an assumption is restrictive and unrealistic since usually, we do not have direct access to the label of the action that is being performed by the expert.

[1]These authors contributed equally to the work.

978-1-4799-7492-4/15/$31.00

Recent approaches perform *imitation from observation* (IfO) [8, 9], which uses only the sequence of state observations from the expert. Such approaches learn two models: the inverse dynamics of the environment (Inverse Dynamics Model, IDM) and an imitation policy model (PM). Current approaches learn both models iteratively from samples based on each other, *i.e.*, the IDM uses demonstrations generated with a specific policy from PM to update its model, and then the PM is updated using the new outcomes from the updated IDM. Using iterations during the learning process allows the policy model to approximate the distribution of actions used by the expert, which improves the imitation process. However, performing IfO using this type of iteration has the drawback of overfitting the policy demonstrations, primarily in the first iterations, and sometimes, causing some of the actions to be ignored altogether during learning of the PM due errors in the IDM.

To deal with this problem in IfO, we design an architecture that uses attention models and a sampling mechanism to regulate the observations that feed the inverse dynamics model, preventing the models from reaching undesirable local minima. We name our proposed approach *Augmented Behavior Cloning from Observations* (ABCO). It learns a model with the inverse dynamics of the environment in order to infer actions from state changes, and a policy model to mimic the expert via behavior cloning. ABCO substantially improves sample efficiency and the quality of the imitation policy model over traditional behavior cloning by exploiting attention mechanisms (Section III-E) within both the inverse dynamic model (Section III-A) and the policy model (Section III-B) and a sampling strategy (Section III-D) that regulates the observations that will feed the inverse dynamic model. Experiments (Section IV) show that by using either low-dimensional state spaces or raw images as input, ABCO outperforms the main IfO algorithms regarding both *Performance* and *Average Episodic Reward*.

II. Problem Formulation

We formulate the problem of imitation learning within the Markov Decision Process (MDP) framework. An MDP is a quintuple $M = \{S, A, T, r, \gamma\}$ [10], where S denotes the set of states in the environment, A corresponds to the set of possible actions, T is the transition model $P(s_{t+1} \mid s_t, a)$, *i.e.*, a function to determine the probability of the agent transitioning from state s_t to s_{t+1} with $s_i \in S$ after taking action $a \in A$ at time t; r is a function that determines the immediate reward for

taking a specific action in a given state, and γ is the discount factor. The solution for an MDP is a policy $\pi(a \mid s)$ that specifies the probability distribution over actions for an agent taking action a in a state s_t when following policy π that imitates the expert behavior.

Since ABCO follows the behavioral cloning from observation (BCO) [8] framework, we are interested in learning an *inverse dynamics model* $\mathcal{M}_a^{s_t,s_{t+1}} = P(a \mid s_t, s_{t+1})$, *i.e.*, the probability distribution of any action a when the agent transitions from state s_t to s_{t+1}. Although we specify the problem as an MDP, the BCO problem is defined without an explicitly-defined reward function, using only *agent-specific states* [11], and having no access to the labels of the actions performed by the expert. Hence, our problem consists in finding an imitation policy π from a set of state-only demonstrations of the expert $D = \{\zeta_1, \zeta_2, \ldots, \zeta_N\}$, where ζ is a state-only trajectory $\{s_0, s_1, \ldots, s_N\}$.

The environment interactions are designed as either pre-demonstrations $\mathcal{I}^{pre}$ or post-demonstrations $\mathcal{I}^{pos}$, where each demonstration contains a set of *interactions* (s_t, a_t, s_{t+1}). Pre-demonstrations set $\mathcal{I}^{pre}$ contains *golden truth* actions, since the agent takes a random action a_t in state s_t of the environment and generates the new state s_{t+1}. Conversely, $\mathcal{I}^{pos}$ contains predicted actions from the model since given some state s_t the model predicts action a_t that tries to mimic the expert behavior and generates the new state s_{t+1}.

III. Augmented Behavioral Cloning from Observation

Behavioral Cloning from Observation (BCO) [8] combines both an *inverse dynamics model* to infer actions in a self-supervised fashion, and a *policy model*, which is a function that tells the agent what to do in each possible state of the environment. The former considers the problem of learning the agent-specific inverse dynamics, and the latter considers the problem of learning an imitation policy from a set of demonstration trajectories. We detail both components, as well as the modifications to this framework we propose in this paper: the *Augmented Behavioral Cloning from Observation* (ABCO) approach.

A. Inverse Dynamics Model

We model the *inverse dynamics model* (IDM) as a neural network responsible for learning the actions that make the agent transition from state s_t to s_{t+1}. In order to learn these actions without supervision, the agent interacts with the environment using a random policy π, generating pairs of states $\mathcal{T}_{\pi_\phi}^{ag} = \{(s_t^{ag}, s_{t+1}^{ag}), \ldots\}$ for agent ag with the corresponding actions $\mathcal{A}_{\pi_\phi} = \{a_t, \ldots\}$. We store pairs of states along with their corresponding action (s_t, a_t, s_{t+1}) as a pre-demonstration ($\mathcal{I}^{pre}$). While randomly transitioning from states in $\mathcal{I}^{pre}$, the model learns the inverse dynamics $\mathcal{M}_\theta$ for the agent by finding parameters θ^* that best describe the actions that occur for achieving the transitions from $\mathcal{T}_{\pi_\phi}^{ag}$. BCO uses the maximum-likelihood estimation (Eq. 1) to find the best parameters, where p_θ is the probability distribution over actions given a pair of states representing a transition. At test time, the IDM uses the learned parameters to predict an action $\hat{a}$ given a state transition (s_t^{ag}, s_{t+1}^{ag}).

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

We augment the original IDM by adding a *Self-Attention* (SA) module [12, 13] (Section III-E), which we use to compensate for the large variation of the samples from $\mathcal{I}^{pre}$ to $\mathcal{I}^{pos}$ in the iterative process. The self-attention forces the IDM to identify what is essential to learn from each state. When using SA with images, it can identify which part of the image representation of the state is essential for predicting the correct action.

B. Policy Model

The *Policy Model* (PM) is responsible for cloning the expert's behavior. Based on the expert demonstrations $D = \{\zeta_1, \zeta_2, \ldots, \zeta_N\}$, where each demonstration comprises pairs of subsequent states $(s_t^e, s_{t+1}^e) \in \mathcal{T}^e$, (A)BCO uses the IDM to compute the distribution over actions $\mathcal{M}_\theta(s_t^e, s_{t+1}^e)$ and predict action $\hat{a}$ that corresponds to the movement made by the expert to change from state s_t to s_{t+1}. With the predicted action (self-supervision), the method builds a set of state-action pairs $\{(s_t^e, \hat{a})\}$ corresponding to the action $\hat{a}$ taken in state s_t. Then this is used to learn the imitation policy π_ϕ that mimics the expert behavior in a supervised fashion.

For behavioral cloning, learning an imitation policy π_ϕ from state-action tuples $\{(s_t^e, \hat{a})\}$ consists of finding parameters ϕ^* for which π_ϕ best matches the provided tuples. Originally, BCO employs maximum-likelihood estimation following the Eq. 2 for finding the best set of parameters ϕ^*. After training the policy network, it performs imitation learning and stores the sequences of states and predicted actions as post-demonstrations ($\mathcal{I}^{pos}$).

$$\phi^* = \arg\max_\phi \prod_{t=0}^{N} \pi_\phi(\hat{a_t} \mid s_t) \tag{2}$$

Compared to the original BCO, we augment the PM by adding a *self-attention* module [12, 13], further detailed in Section III-E. Unlike in the IDM, we use SA to reduce the state changes during each iteration, since the self-attention module focus on small details and differentiate better all classes given by the IDM. The SA also allows the policy to look non-locally at states, helping the model to learn faster for higher dimensional states (*e.g.*, *Maze* and *Acrobot*) with a more gradual success rate in between iterations.

C. Iterated Behavioral Cloning from Observation

Torabi *et al.* [8] extend the BCO algorithm using the post-demonstration environment interaction to improve both the IDM and the imitation policy. The improvement, named BCO(α), where α represents a user specified hyperparameter to control the number of post-demonstration interactions, works as follows. After learning the imitation policy,

Algorithm 1 ABCO(α)

Initialize the model $\mathcal{M}_\theta$ as a random approximator
Initialize the policy π_ϕ with random weights
Generate $\mathcal{I}^{pre}$ using policy π_ϕ
Generate state transitions $\mathcal{T}^e$ from demonstrations D
Set $\mathcal{I}^s = \mathcal{I}^{pre}$
for $i \leftarrow 0$ to α **do**
    Improve $\mathcal{M}_\theta$ by TRAINIDM($\mathcal{I}^s$)
    Use $\mathcal{M}_\theta$ with $\mathcal{T}^e$ to predict actions $\hat{A}$
    Improve π_ϕ by behavioralCloning($\mathcal{T}^e$, $\hat{A}$)
    for $e \leftarrow 1$ to $|E|$ **do**
        Use π_ϕ to solve environment e
        Append samples $\mathcal{I}^{pos} \leftarrow (s_t, \hat{a}_t, s_{t+1})$
        if π_ϕ at goal g **then**
            Append $v_e \leftarrow 1$
        else
            Append $v_e \leftarrow 0$
        end if
    end for
    Set $\mathcal{I}^s$ = SAMPLING($\mathcal{I}^{pre}$, $\mathcal{I}^{pos}$, $P(g \mid E)$, v_e)
end for

the agent executes the environment to acquire new state-action sequences as post-demonstrations ($\mathcal{I}^{pos}$). These post-demonstrations are then employed to update the IDM, and further on, the imitation policy itself.

The problem of the iterated BCO is that it only uses the set of post-demonstrations to re-train the IDM. Thus, for those cases in which the policy still does not have good enough predictive performance, the generated set of post-demonstrations will contain misleading actions for specific pairs of states. Those erroneous actions tend to degrade the predictive performance of the IDM, which leads to degrading the predictions of the policy in a negative feedback loop.

In order to deal with these problems, we create ABCO(α) that iteratively improves over ABCO via a sampling method that weights how much the IDM should learn from pre-demonstrations ($\mathcal{I}^{pre}$) and post-demonstrations ($\mathcal{I}^{pos}$). Algorithm 1 summarizes the ABCO(α) training process, where TRAINIDM($\mathcal{I}^s$) refers to using $\mathcal{I}^s$ to find a θ^* that best explains the transitions in the demonstration $\mathcal{I}^s$ as in Eq. 1.

Unlike BCO(0), which uses only $\mathcal{I}^{pre}$ to train the IDM, ABCO(α) updates the training data ($\mathcal{I}^s$) in every iteration. As ABCO(α) does not have any post-demonstration data in the first iteration, $\mathcal{I}^s$ receives all data from $\mathcal{I}^{pre}$. From the second iteration onwards, $\mathcal{I}^s$ receives the concatenation of a sample ($\mathcal{I}^{pos}_{spl}$) from $\mathcal{I}^{pos}$ and a sample ($\mathcal{I}^{pre}_{spl}$) from $\mathcal{I}^{pre}$ using a win-loss probability according to the agent capability of achieving the goal for each environment.

D. Sampling

For every iteration, our sampling strategy creates new training data $\mathcal{I}^s$ containing a set of post-demonstrations $\mathcal{I}^{pos}_{spl}$ and a set of pre-demonstrations $\mathcal{I}^{pre}_{spl}$. In order to obtain the sample from post-demonstrations ($\mathcal{I}^{pos}_{spl}$), we first select the distribution of actions given a run E in an environment and the current policy $P(A \mid E; \mathcal{I}^{pos})$. We consider only successful runs from $\mathcal{I}^{pos}$, *i.e.*, only state-action sequences in which the agent was able to achieve the environmental goal. Note that this goal might be a specific state (i.e. such that the last transition in $\mathcal{I}^{pos}$ is in a set of specified states), or avoiding an undesirable state for a fixed number of transitions. We infer these goal states from the type of expert demonstration we receive.We represent it as v_e in Eq. 3, where v_e is set to 1 if the agent achieves the environmental goal and zero otherwise, and E is the set of runs in an environment.

$$P(A \mid E; \mathcal{I}^{pos}) = \frac{\sum_{e \in E} v_e \cdot P(A \mid e)}{|E|} \tag{3}$$

The intuition of using the post-demonstration only for successful runs is that if a policy is unable to achieve the environmental goal, then the post-demonstration alone does not close the gap between what the model previously learned with $\mathcal{I}^{pre}$ and what the expert performs in the environment. Using only successful runs also gives us a more accurate distribution of the expert since we are only using those distributions that achieved the goal instead of the random distribution that consisted of a balanced dataset. By not adding unsuccessful runs to the training dataset, we solve the problem in which BCO(α) degrades the performance in both models. With the distribution of actions from winning executions, we select the sample $\mathcal{I}^{pos}_{spl}$ from those runs, according to the win probability $P(g \mid E)$, *i.e.*, the probability of achieving a goal in an environment, as shown in Eq. 4.

$$\mathcal{I}^{pos}_{spl} = (P(g \mid E) \times P(A \mid E, \mathcal{I}^{pos})) \sim \mathcal{I}^{pos} \tag{4}$$

The sample from pre-demonstrations has a complementary size of the post-demonstrations. Thus, to create a sample from pre-demonstrations $\mathcal{I}^{pre}_{spl}$, we use the loss probability with a distribution of actions in pre-demonstrations, denoted by $P(A \mid \mathcal{I}^{pre})$, as demonstrated in Eq. 5.

$$\mathcal{I}^{pre}_{spl} = ((1 - P(g \mid E)) \times P(A \mid \mathcal{I}^{pre})) \sim \mathcal{I}^{pre} \tag{5}$$

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

Using a win-loss probability, we induce the training data to be closer to the expert demonstration than to the random data, which boosts the model capability of imitating the expert. In this setting, the more an agent can achieve its goal, the less we want $\mathcal{I}^s$ consisting of $\mathcal{I}^{pre}$ and more of $\mathcal{I}^{pos}$. It is important to emphasize that our method is only goal-aware since we consider tuples from successful runs in the sample from post-demonstration, and does not use the reward information for

learning or optimizing. We do not use reward because not all environments have a function for it. On the other hand, most agents have a goal that is relatively easy to visually identify by inspecting the last transition, *e.g.* the *mountaincar* reaching the flag pole, arriving at the final square at a *maze*, *acrobot* reaching the horizontal line, and the *cartpole* surviving up to 195 steps, as described in Section IV-A.

E. Self-attention Module

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

ABCO uses the SA module based on the Self-Attention Generative Adversarial Network (SAGAN) [13], since it outperforms prior work in image synthesis. In SAGAN the self-attention module computes the key $f(x)$, the query $g(x)$ and the value $h(x)$, given a feature map x, with convolutional filters with the equations $f(x) = W_f x$, $g(x) = W_g x$ and $h(x) = W_h x$. We compute the attention map by performing two different steps. First, we apply Eq. 6 with the current key f and query g.

$$s_{ij} = f(x_i)^T g(x_j) \tag{6}$$

Second, we calculate the softmax function $\beta_{j,i}$, over the attention module to the i^{th} location when synthesizing the j^{th} region. With the attention map β and the values $h(x)$ we compute the self-attention feature maps $a = (a_1, a_2, ..., a_N) \in \mathbb{R}^{C \times N}$, where N is the number of feature locations and C is the number of channels, as illustrated in Eq. 7.

$$a_j = v\left(\sum_{i=1}^{N} \beta_{j,i} h(x_i)\right), v(x_i) = W_v x_i \tag{7}$$

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

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

IV. Implementation and Experimental Results

In order to test ABCO, we perform experiments using the environments (Section IV-A) of the OpenAI Gym [14] toolkit. We developed two networks comprising vector-based environments and image-based environments (Section IV-B) and evaluated the results in terms of *Average Episodic Reward* (AER) and *Performance* (P) (Section IV-C). We describe our findings and a comparison with the state-of-the-art in Section IV-D.

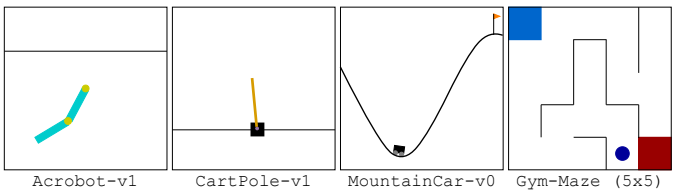

Fig. 1. Example frames of Acrobot-v1, CartPole-v1, MountainCar-v0 and Gym-Maze 5x5 environments.

A. Environments

We perform all experiments using four environments from OpenAI Gym [14]. These environments are separated in vector-based environments (*Acrobot-v1*, *Cart-Pole-v1*, *MountainCar-v0*), and image-base environments (*Gym-Maze* 3×3, 5×5, and 10×10). Each environment is described below and illustrated in Fig. 1.

- **Acrobot-v1** is an environment that includes two joints and two links, where the joint between the two links is actuated. Initially, the links are hanging downwards, and the goal is to swing the end of the lower link up to a given height. The state space contains 6 dimensions: $\{\cos\theta_1, \sin\theta_1, \cos\theta_2, \sin\theta_2, \theta_1, \theta_2\}$, and the action space consists of the 3 possible forces. Acrobot-v1 is an unsolved environment, *i.e.*, it does not have a specified reward threshold at which it is considered solved.
- **CartPole-v1** is an environment where an agent pulls a car sideways with the goal of sustaining a pole vertically upward as long as possible. The environment has a discrete action space composed of *left* or *right*, while the state space has 4 dimensions: *cart position*, *cart velocity*, *pole angle*, and *pole velocity at tip*. CartPole-v1 defines solving as getting average reward of 195 over 100 consecutive trials.
- **MountainCar-v0** environment consists of a car in an one-dimensional track, positioned between two "mountains". The state space has 2 dimensions, the respective car coordinates (x, y), and the action space consists of 3 possible strengths to move the car (-1, 0, or 1). To achieve the goal in this environment, the car has to acquire the required momentum from the left mountain to drive up to the mountain on the right. MountainCar defines solving as getting average reward of -110.0 over 100 consecutive trials.
- **Gym-Maze** is a 2D maze environment where an agent (the blue dot in Fig. 1, should find the shortest path from the start (the blue square in the top left corner) to the goal (the red square in the bottom right corner). Each maze can have a different set of walls configuration, and three different sizes, 3×3, 5×5, and 10×10. An agent is allowed to walk towards any wall. The agent has a discrete action space composed of

N, *S*, *W*, and *E*, and the state space consisting of rendered images of the maze.

B. Implementation

We create two different networks to address each type of environment: a network for low-dimensional *vector-based environments*, and a network for high-dimensional *image-based environments*. We developed all models using the *PyTorch* framework, with the Cross Entropy loss function, and the Adam optimizer [15]. We added self-attention [12, 13] modules in both IDM and PM. Below, we describe the details of each network used in our experiments, where FC_d is a fully connected layer containing d dimensions, SA_d is a self-attention layer, and ... indicates the sequence of layers from the original architecture up to the description of the next layer.

• **Vector-based Environments**: $Input_{dims} \rightarrow FC_{12} \rightarrow SA_{12} \rightarrow FC_{12} \rightarrow SA_{12} \rightarrow FC_{12} \rightarrow FC_{12} \rightarrow Output_6$, where *dims* is a vector with twelve and six states for the IDM and PM, respectively.

• **Image-based Environments**: we modified the ResNet [16] architecture by adding two self-attention modules as follows: $Input_{224\times224} \rightarrow \ldots \rightarrow ResBlock_2 \rightarrow SA_{64} \rightarrow \ldots \rightarrow ResBlock_4 \rightarrow SA_{128} \rightarrow \ldots \rightarrow FC_{dims} \rightarrow LeakyRelu \rightarrow Dropout_{0.5} \rightarrow FC_{512} \rightarrow LeakyRelu \rightarrow Dropout_{0.5} \rightarrow Output_4$, where *dims* is a vector with 1024 and 512 features for the IDM and PM, respectively.

C. Metrics

We evaluate our policies with two known metrics in the area: the *Average Episodic Reward* (AER) and the *Performance* (P). AER is a standard metric used to measure how well our generated policy performs in a specific environment. The metric consists of the average value of one hundred runs for each episode in a given environment (*e.g.*running one hundred of different mazes in Gym-Maze environment and calculating the average performance, or running one hundred consecutive episodes for the CartPole problem and calculating the average reward). AER value is an ideal metric to understand how well the expert did a task and consequently understand how difficult it is to imitate the expert behavior.

On the other hand, the *Performance* (P) metric calculates the average reward for each run scaled from zero to one. Zero scores for P represents the reward obtained for a random policy running in a given environment, while the one score represents the reward obtained by the expert policy. It is important to note that each environment has its own value for the minimum and maximum rewards, and thus P is not comparable between different environments. We do not use accuracy to measure the quality of our generated policies since this metric can not guarantee high-quality results for this problem. As mentioned in Section III, we run our problem in a two-phase approach, where in the first phase, our model can quickly propagate the error to the second. Therefore, we cannot use the accuracy as a metric to verify the quality of the generated PM since achieving 100% accuracy with a generated policy using a poor IDM will lead us to lower AER and P.

D. Results

In order to evaluate our approach, we compare our trained models with the state-of-the-art approaches. All models are trained using the same initial set of random pre-demonstrations $\mathcal{I}^{pre}$. Table I shows the results in terms of *Average Episodic Reward* (AER) and *Performance* (P) for our models and the related work: BCO [8] and ILPO [17]. For comparison purposes, we also show the results for *Behavioral Clone* (BC), which is a supervised approach.

Table I shows that our method is equal or surpasses the state-of-the-art approaches in all the environments but the Maze 3×3 where results are similar to BCO. The overall results confirm that the attention module and our sampling strategy can improve the imitation process. All approaches achieved the maximum score for CartPole in both AER and *Performance*, showing that this problem is easy to learn. Although our model achieved the best P and AER scores in the Acrobot environment, the related work presented similar results with $P \approx 1.00$ and $AER = -85.300$. Our model achieving better results for AER metric means that ABCO can solve the problem using fewer frames. However, both models present similar imitation capabilities since all models achieved $P \approx 1.00$. It is important to note that even ABCO not using labeled data, it was able to achieve better results than the BC approach that uses labels for actions. For MountainCar, we observe a large difference in terms of *Performance*, with our model achieving $P = 1.289$ and presenting a difference of ≈ 0.34 to BCO, which is the second-highest result.

Although in all the Maze environments we achieved the highest scores for *Performance*, we can observe that as the complexity of the environment increases, our performance decreases. Nevertheless, we can see in the results that ABCO is less affected than BCO as the complexity increases, since the *Performance* for our results in Mazes 3×3, 5×5 and 10×10 are 1.159, 0.960 and 0.860 respectively, while BCO obtained 0.883, -0.112 and -0.416. In terms of AER, ABCO was only outperformed by BCO in Maze 3×3 by ≈ 0.02, where Torabi *et al.* [8] achieved $AER = 0.927$. Comparing the results of ABCO with ILPO, we observe that ILPO increases its *Performance* as the maze increases in size but it is still much lower than ABCO for the 10×10 maze. We believe that this discrepancy happens for two reasons. First, our method contains an attention module (III-E), which increases the capability of ABCO to focus on essential features through non-visited state spaces. Second, ILPO does not consider a full image of the scenario since it uses crop mechanisms and internal manipulations with the state images. Using a partial observation from the environment means that the approach cannot receive essential features from the images (*e.g.* the initial state, the goal state, the agent, *etc*). On the other hand, as the maze increases, ILPO receives more local information through the crops, increasing its *Performance*.

V. Discussion

We perform an ablation study to observe the impact of each component of our architecture. We measure *Performance*

TABLE I

Performance (P) AND *Average Episode Reward* (AER) FOR A SUPERVISED MODEL (BC), THE RELATED WORK (BCO AND ILPO) AND OUR APPROACH (ABCO) USING OPENAI GYM ENVIRONMENTS.

Model	Metric	CartPole	Acrobot	MountainCar	Maze 3×3	Maze 5×5	Maze 10×10
BC	P	**1.000**	1.071	1.560	-1.207	-0.921	-0.470
	AER	**500.000**	-83.590	-117.720	0.180	-0.507	-1.000
BCO[8]	P	**1.000**	0.980	0.948	0.883	-0.112	-0.416
	AER	**500.000**	-117.600	-150.00	**0.927**	0.104	-0.941
ILPO[17]	P	**1.000**	1.067	0.626	-1.711	-0.398	0.257
	AER	**500.000**	-85.300	-167.00	-0.026	-0.059	-0.020
ABCO(α)	P	**1.000**	**1.086**	**1.289**	**1.159**	**0.960**	**0.860**
	AER	**500.000**	**-77.900**	**-132.30**	0.908	**0.932**	**0.784**

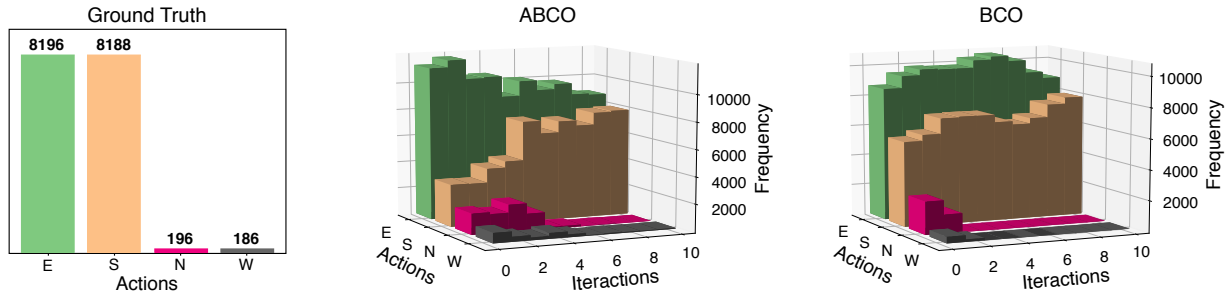

Fig. 2. Inverse Dynamic Model predictions of the expert examples through time. It is possible to see that after the first two iterations, due to BCO's Policy poor performance, IDM stopped predicting "North" and "West" classes, while ABCO, although lower than the ground truth, kept predicting sixty and ten after the fifth iteration. We believe that the vanishing of the action from BCO is due to all examples from the less present classes in $\mathcal{I}^{pos}$ being worse representations that the random ones, making the Inverse Dynamic model stop predicting those classes, due to expert examples being closer, in the feature space, from other classes than their own.

and *AER* when using the only the self-attention mechanism without sampling, when using the sampling strategy without self-attention, and when using the combination of attention with different samplings. We generate all results using the Maze 5×5 environment.

A. ABCO and Self-attention

To measure the impact on the learning process, we trained ABCO using only the self-attention module. We observe that when using only the self-attention, the accuracy of our models was higher than the original method. However, high accuracy does not represent an excellent performance since, without the sampling method, some action might not occur in further iterations. With high accuracy and samples that do not represent the action from the expert accordingly, IDM stops predicting the minority action, creating a sub-set from all possible actions, and the policy learn the new subset of the real actions. This behavior results in the policy not performing the less frequent actions that are needed to solve different environments during the inference phase (*e.g.*, North and West, on mazes, or not performing actions during acrobot), as we discuss in Section V-B.

We observe that even when weighing the features, IDM is still capable of predicting the most common path. When feeding the Policy with ten different solutions for each maze, the agent mimics the most common path, as shown in Fig. 3. Nevertheless, when the first iterations still sample all classes, the model takes more transition samples to reach the results from Table II. Although self-attention alone achieves results similar to BCO(α), when combined with the sampling method, it has a significant impact on the results.

TABLE II

ABLATION STUDY CONSIDERING THE 2 MAIN COMPONENTS OF ABCO: *attention* AND *sampling* IN THE 5×5 MAZE ENVIRONMENT.

Model	P	AER
BCO [8]	-0.112	-0.941
Attention	-0.415	-0.940
Partial Sampling	0.717	0.716
Whole Sampling	0.628	0.676
ABCO (Attention + Partial Sampling)	0.960	0.932
ABCO (Attention + Whole Sampling)	0.759	0.755

B. ABCO and Sampling

In this experiment, we use only the sampling module to train ABCO(α) by disabling the self-attention module. We use the sampling method without the reduction of samples from $\mathcal{I}^{pos}$. We hypothesize that sampling from the original random

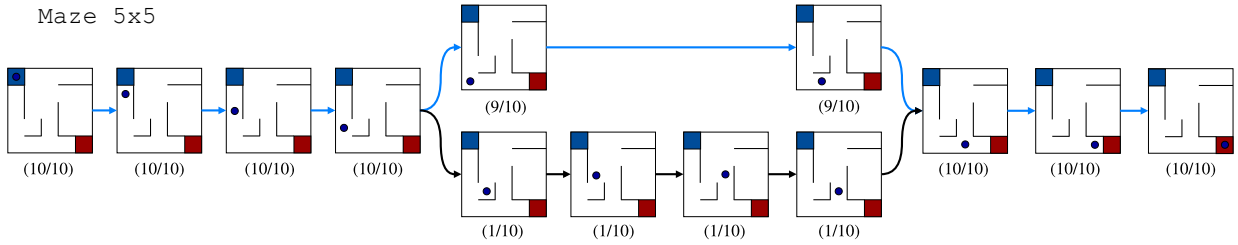

Fig. 3. Expert demonstrations executing a 5x5 configuration of Gym-Maze. Bellow the state-image we represent the number of experts that visited that state. The blue line represents the path chosen by our ABCO agent.

policy dataset helps to solve the vanishing actions, as well as close the difference from the first iteration $\mathcal{I}$ and the expert. The vanishing of actions from the IDM prediction occurs due to the weak policy inference creating a $\mathcal{I}^{pos}$ that does not contain all actions or sparse representations that underfit the inverse dynamic model. during the early iterations under these conditions, IDM stops predicting the classes that are the minority in the expert dataset. This misclassification causes the policy to loop between actions that prevent the model from achieving its goal. We compare the distribution of all predictions from the IDM from the BCO(α) and the ABCO in Fig. 2, where it shows that our sampling method can better predict all classes due to the artificial growth of our dataset caused by sampling from the $\mathcal{I}^{pre}$.

Furthermore, to observe if the policy can create samples that are closer from the expert than the random dataset, we calculate the $L2$ distances from the average of all images from each action during each iteration and normalize them between zero, for the expert, and one, for the $\mathcal{I}^{pre}$ samples. The results in Fig. 4 represent how our model learns a policy that creates better $\mathcal{I}$ for the majority classes (*e.g.*, S and E), and even for the minority classes (*e.g.*, N and W). We assume this difference of the approximation of the expert dataset to be due to the minority classes consisting mostly of the $\mathcal{I}^{pre}$ since most mazes do not require those actions. By sampling from the random dataset, we force our IDM to balance its labeling and create iterations that are further distant. Still, as the Policy progresses and solves more runs, it approximates

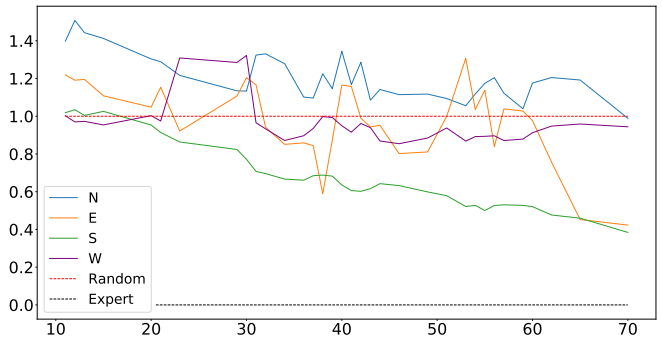

Fig. 4. $L2$ distance for the average of each action for each iteration normalized by the expert and random samples in the 5×5 mazes.

and becomes closer. By being closer to the expert, the new samples allow the IDM to finetune itself and predict expert labels more precisely.

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

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

VI. Related Work

Many approaches for imitating from observations have been recently proposed [8, 17, 18, 22]. Ho and Ermon [18] propose a *generative adversarial imitation learning* (GAIL) approach that learns to imitate policies from state-action demonstrations using adversarial training [19].

Edwards *et al.* [17] describes a forward dynamics model, *i.e.*, a mapping from state-action pairs $\{(s_t, a_t)\}$ to the next state $\{s_t+1\}$, called *imitating latent policies from observation* (ILPO). In their two-step approach, the agent first learns a latent policy offline that estimates the probability of a latent action given the current state. Then, in a limited number of steps in the environment, they perform remapping of the actions, associating the latent actions to the corresponding exact actions. This approach is very efficient in terms of interactions needed since most of the process occurs offline.

Torabi *et al.* [8] develop *behavioral cloning from observation* (BCO) to imitate the behavior of an expert in a self-supervised way by observing its states. Their approach contains a model that learns the inverse dynamic of the agent, and a policy model learns which action the agent should use

given a state. In that work, Torabi *et al.* train BCO using only low-dimensional state features.

Using high-dimensional space, Torabi *et al.* [23] explores the fact that agents often have access to their internal states (*i.e.*, *proprioception*). In that approach, the architecture learns policies over proprioceptive state representations and compares the resulting trajectories visually to the demonstration data.

VII. Conclusions and Future Work

In this paper, we developed a novel approach to learn how to imitate the behavior from experts just by observing their states with no prior information about their actions. The pipeline of the architecture includes training two modules iteratively. The *Inverse Dynamics Model*, which is responsible for learning actions given that the agent transitioned from two states; and the *Policy Model* that aims to predict which action the agent has to select given a state in order to imitate the expert. Using four different environments mentioned in Section IV-A, we perform experiments showing that our approach can use low-dimensional or raw image data to learn how to imitate an expert, and achieve better results than the best current methods.

As future work, we aim to evaluate our technique in more challenging domains, such as Continuous Control Tasks, Atari Games, Robotics Goal-based Tasks and others. Such domains, have larger space states when compared with the ones in our evaluation, which makes them harder to imitate. We plan to work with temporal approaches and possibly adversarial techniques in order to search for a model that could be able to imitate and generalize having comparable or better results than the state-of-the-art.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and CAPES/FAPERGS agreement (DOCFIX 04/2018) process number 18/2551-0000500-2. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the graphics cards used for this research.

References

[1] A. Bandura and R. H. Walters, *Social Learning Theory*, 1st ed. Prentice-hall Englewood Cliffs, NJ, 1977.

[2] S. Raza, S. Haider, and M.-A. Williams, "Teaching coordinated strategies to soccer robots via imitation," in *Proceedings of ROBIO 2012*, 2012, pp. 1434–1439.

[3] S. Schaal, "Learning from demonstration," in *Proceedings of NIPS 1996*, 1996, pp. 1040–1046.

[4] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[5] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys*, vol. 50, no. 2, pp. 21:1–21:35, 2017.

[6] G. Rizzolatti and C. Sinigaglia, "The functional role of the parieto-frontal mirror circuit: Interpretations and misinterpretations," *Nature Reviews Neuroscience*, vol. 11, no. 4, pp. 264–274, 2010.

[7] Y. Liu, A. Gupta, P. Abbeel, and S. Levine, "Imitation from observation: Learning to imitate behaviors from raw video via context translation," in *Proceedings of ICRA 2018*, 2018, pp. 1118–1125.

[8] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," in *Proceedings of IJCAI'18*, 2018, pp. 4950–4957.

[9] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 2, no. 4.

[10] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine, "Learning invariant feature spaces to transfer skills with reinforcement learning," in *Proceedings of ICLR 2017*, 2017, pp. 1–14.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of NIPS 2017*, 2017, pp. 5998–6008.

[12] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *Proceedings of ICML 2019*, 2019, pp. 7354–7363.

[13] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2414–2423.

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

[15] A. D. Edwards, H. Sahni, Y. Schroecker, and C. L. Isbell, "Imitating latent policies from observation," in *Proceedings of ICML 2019*, 2019, pp. 1755–1763.

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

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of CVPR 2016*, 2016, pp. 770–778.

[18] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Proceedings of NIPS 2016*, 2016, pp. 4565–4573.

[19] F. Torabi, G. Warnell, and P. Stone, "Generative adversarial imitation from observation," in *I3 Workshop at ICML 2019*, 2019.

[20] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of NIPS'14*, 2014, pp. 2672–2680.

[21] F. Torabi, G. Warnell, and P. Stone, "Imitation learning from video by leveraging proprioception," in *Proceedings of IJCAI'19*, 2019, pp. 3585–3591.