Interactive Learning of Temporal Features for Control
Shaping Policies and State Representations From Human Feedback

Perez-Dattari, Rodrigo; Celemin, Carlos; Franzese, Giovanni; Ruiz-del-Solar, Javier; Kober, Jens

DOI
10.1109/MRA.2020.2983649

Publication date
2020

Document Version
Accepted author manuscript

Published in
IEEE Robotics and Automation Magazine

Interactive Learning of Temporal Features for Control

Shaping Policies and State Representations From Human Feedback

By Rodrigo Pérez-Dattari, Carlos Celemin, Giovanni Franzese, Javier Ruiz-del-Solar, and Jens Kober

Current ongoing industry revolution demands more flexible products, including robots in household environments and medium-scale factories. Such robots should be able to adapt to new conditions and environments and be programmed with ease. As an example, let us suppose that there are robot manipulators working on an industrial production line and that they need to perform a new task. If these robots were hard coded, it could take days to adapt them to the new settings, which would stop production at the factory. Robots that non-expert humans could easily program would speed up the process considerably.

In this regard, we present a framework in which robots are capable of quickly learning new control policies and state representations (SRs) by using occasional corrective human feedback. To achieve this, we focus on robots interactively learning these policies from non-expert humans who act as teachers. We present a neural network (NN) architecture along with an interactive imitation learning (IIL) method that efficiently learns spatiotemporal features and policies from raw high-dimensional observations (raw pixels from an image) for tasks in environments that are not fully temporally observable.

We denominate IIL as a branch of IL, where human teachers provide different kinds of feedback to robots, such as new demonstrations triggered by robot queries [1], corrections [2], preferences [3], reinforcements [4], and so forth. Most IL methods work under the assumption of learning from perfect demonstrations; therefore, they fail when teachers have only partial insights about the task execution. Non-expert teachers could include all users who are neither machine learning/control experts nor skilled enough to fully show the desired behavior of the policy.

Interactive approaches, such as COACH (which is the short form of Corrective Advice Communicated by Humans) [5], and some interactive reinforcement learning (IRL) approaches [4], [6] are intended for non-expert teachers but are not completely deployable for end users. Sequential decision-making learning methods (IL, IIL, IRL, and so forth) rely on good SRs, which simplify the shaping of the policy landscape and provide suitable generalization properties. However, this requirement means that experts on feature engineering must preprocess the states properly before running the learning algorithms.

Digital Object Identifier 10.1109/MRA.2020.2983649
Date of current version: 22 April 2020

1070-9932/20©2020IEEE

The inclusion of deep learning (DL) in IL (given the popularity DL has gained in the field of RL [7]) enables practitioners to skip preprocessing modules for inputting policies since some NN architectures endow the agents with intrinsic feature-extraction capabilities. This has been exhaustively tested in end-to-end settings [7]. In this regard, DL enables non-expert humans to shape policies based only on their feedback.

Nevertheless, in real-world problems, we commonly face tasks wherein the observations do not explain the complete state of the agent, due to the lack of temporal information (e.g., rates of change) or because the agent–environment interaction is non-Markovian (e.g., dealing with occlusions). For these cases, it is necessary to provide memory to the learning policy. Recurrent NNs (RNNs) can learn to model dependencies from past observations and map them to the current outputs. This recurrency has been used in RL and IL, mostly through long short-term memory (LSTM) networks [8].

Therefore, LSTMs are included in our NN architecture to learn temporal features, which contain relevant information from the past. However, DL algorithms require large amounts of data; as a way to tackle this shortcoming, SR learning (SRL) has been used to learn features more efficiently [9], [10]. Considering that real robots and human users have time limitations, as an SRL strategy, a model of the world is learned to obtain SRs that make the policy convergence possible within feasible training time intervals (see Figure 1). The combination of SRL and the teacher's feedback is a powerful strategy for efficient learning of temporal features from raw observations in non-Markovian environments.

The experiments presented in this article show the impact of the proposed architecture in terms of data efficiency and the policy's final performance within the deep COACH (D-COACH) IIL framework [11]. Additionally, the experimental procedure demonstrates that the proposed architecture could even be used with other IL methods, such as data aggregation (DAgger) [12]. The code used in this paper can be found at https://github.com/rperezdattari/Interactive-Learning-of-Temporal-Features-for-Control.

Background and Related Work

Our method combines elements from SRL, IL, and memory in NN models to build a framework that enables non-expert teachers to interactively shape policies in tasks with non-Markovian environments. These elements are introduced in the following.

Dealing With Non-Markovian Environments

There are different reasons why a process could be partially observable. One is when the state describes time-dependent phenomena, but the observation contains only partial information about them. For instance, velocities cannot be estimated from camera images unless observations from different time steps are combined. Other examples of time-dependent phenomena are temporary occlusions and corrupted communication systems between the sensors and the agent.

For these environments, the temporal information needs to be implicitly obtained within the policy model. There are

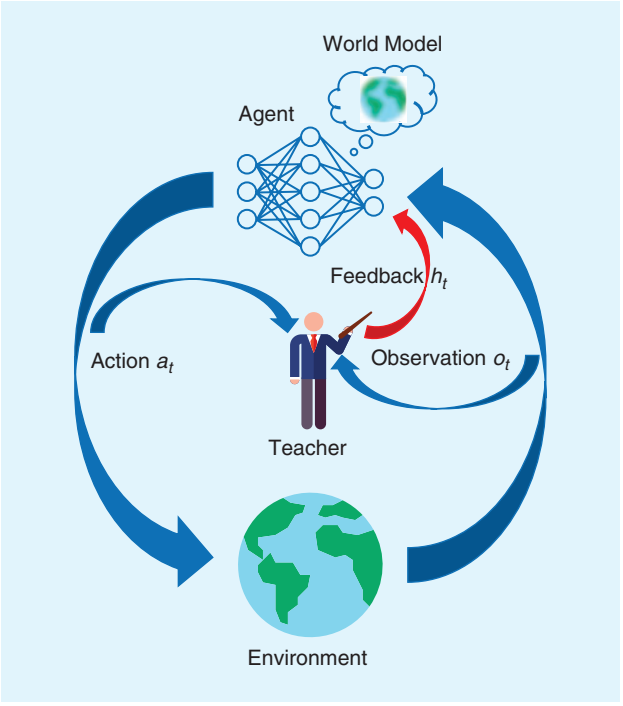

Figure 1. Interactively shaping policies with agents that model the world. (Source: turkkub and Freepik from Flaticon.)

two well-known approaches for adding memory to agents in sequential decision-making problems when using NNs as function approximators:

1) *Observation stacking policies* [7]: stacking the last N observations $(o_t, o_{t-1}, \ldots o_N)$ and using this stack as the input of the network
2) *Recurrent policies* [13]: including RNN layers in the policy architecture.

One of the main issues with observation stacking is that the memory of these models is determined by the number of stacked observations. The overhead increases rapidly for larger sequences in high-dimensional observation problems.

In contrast, RNNs can model information for an arbitrarily long period of time [14]. Also, they do not add input-related overheads because, when these models are evaluated, they use only the last observation. Therefore, RNNs have a lower computational cost than observation stacking. Given the more practical usage of recurrent models and their capability of representing arbitrarily long sequences, in this article, we use RNN-based policies (with LSTM layers) in the proposed NN architecture. Nevertheless, the use of LSTMs has a critical disadvantage since their training is more complex and requires more data, something very problematic when considering human teachers and real systems. We now introduce SRL, which helps to accelerate LSTM convergence.

SRL

In most of the problems faced in robotics, the state s_t, which fully describes the situation of the environment at time step t, is not fully accessible from the robot's observation o_t. As mentioned, in several problems these observations lack the

Algorithm 1: (HG-)DAgger

1: **Require**: demonstrations database $\mathcal{D}$ with initial demonstrations, policy update frequency b
2: **for** t = 1, 2, … **do**
3: **if** mod(t, b) is 0 **then**
4: **update** π_θ from $\mathcal{D}$
5: **observe** state o_t
6: **select** action from agent or expert
7: **execute** action
8: **feedback** provide label a_t^* for o_t, if necessary
9: **aggregate** (o_t, a_t^*) to $\mathcal{D}$

temporal information required in the state description. Moreover, these observations tend to be raw sensor measurements that can be high-dimensional, highly redundant, and ambiguous. A portion of this data may even be irrelevant.

As a consequence, to successfully solve these problems a policy needs to 1) find temporal correlations between several consecutive observations and 2) extract relevant features from observations that are hard to interpret. However, finding relations among these large data structures with the underlying phenomena of the environment while also learning controllers can be extremely inefficient. Therefore, efficiently building controllers on top of raw observations requires learning-informative, low-dimensional SRs [15]. The objective of SRL is to obtain an observer capable of generating such representations.

A compact representation of a state is considered suitable for control if the resulting SR

- is Markovian
- has good generalization to unseen states
- is defined in low-dimensional space (considerably lower than the actual observation dimensionality) [9].

Along with the control objective function (e.g., reward function and imitation cost function), other objective functions can be used for SRL [10]:

- observation reconstruction
- forward model or next observation prediction
- the inverse model
- the reward function
- the value function.

Interactive Learning Methods

This section briefly introduces two approaches for interactively learning from human teachers while agents are executing a task.

Data Aggregation: DAgger and Human-Gated DAgger

DAgger [12] is an IIL algorithm that aims to collect data through online sampling. To achieve this, trajectories are generated by combining the agent's policy π_θ and the expert's policy. The observations o_t and the demonstrator's corresponding actions a_t^* are paired and added to a database $\mathcal{D}$, which is used for training the policy's parameters θ iteratively in a supervised learning manner to asymptotically approach the expert's policy. At the beginning of the learning process, the demonstrator has all the influence over the trajectory made by the agent; then the probability of following the demonstrator's actions decays exponentially.

For working in real-world systems with humans as demonstrators, a variation of DAgger, human-gated DAgger (HG-DAgger) [2], was introduced. In this approach, the demonstrator is not expected to give labels across every action of the agent but only in places where she/he considers that the agent's policy needs improvement. Only these labels are aggregated to the database and used for updating the policy. Additionally, every time feedback is given by the human, the policy will follow the provided action. As a safety measure, in HG-Dagger, the uncertainty of the policy across the observation space is estimated; that element is omitted in this article. Algorithm 1 shows the general structure of DAgger and HG-DAgger.

D-COACH

In this framework, humans shape policies, giving occasional corrective feedback for actions executed by the agents [11]. The human indicates agent actions that she/he considers to be erroneous through a binary signal h_t, the direction in which the action should be modified. This feedback is used to generate an error signal for updating the policy parameters θ. It is performed in a supervised learning manner, with the cost function J and using the mean squared error and stochastic gradient descent. Hence, the update rule is

$$\theta \leftarrow \theta - \alpha \cdot \nabla_\theta J(\theta). \tag{1}$$

The feedback given by the human indicates only the sign of the policy error. Its magnitude is supposed to be unknown since the algorithm works under the assumption that the user is non-expert; therefore, she/he does not know the magnitude of the proper action. Instead, the error magnitude is defined as the hyperparameter e that must be defined before starting the learning process. Thus, the policy $error_t$ is defined by $h_t \cdot e$.

To compute a gradient in the parameter space of the policy, the error needs to be a function of θ. This is achieved by observing that

$$error_t(\theta) = a_t^{\text{target}} - \pi_\theta(o_t), \tag{2}$$

where a_t^{target} is the incremental objective generated by the feedback of the human $a_t^{\text{target}} = a_t + error_t$, and a_t is the current output of the policy π_θ. From (1), (2), and the derivative of the mean squared error, we can get the COACH update step:

$$\theta \leftarrow \theta + \alpha \cdot error_t \cdot \nabla_\theta \pi_\theta. \tag{3}$$

To be more data efficient and avoid locally overfitting to the most recent corrections, D-COACH has a memory buffer that stores the tuple $(o_t, a_t^{\text{target}})$ and replays this information during learning. Additionally, when working in problems with high-dimensional observations, an autoencoding cost is

incorporated in D-COACH as an observation reconstruction SRL strategy. In the D-COACH pseudocode (Algorithm 2), this SRL step is omitted. D-COACH learns everything from scratch through only one interactive phase, unlike other deep interactive RL approaches [4], [6], which split the learning process into two sequential learning phases: first, recording samples of the environment for training a dimensionality reduction model (e.g., an autoencoder) and, second, using that model for the input of the policy network during the actual interactive learning process.

Learning Temporal Features Based on Interactive Teaching and World Modeling

In this section, the SRL NN architecture is described along with the interactive algorithm for policy shaping.

Network Architecture for Extracting Temporal Features

When approaching problems that lack temporal information in the observations, the most common solution is to model the policy with RNNs, as discussed in the "Dealing With Non-Markovian Environments" section; therefore, we propose to shape policies that are built on top of RNNs, with occasional human feedback. In this article, we use the terms *world model* and *transition model* interchangeably.

IIL methods can take advantage of SRL for training with other objective functions by 1) making use of all of the experience collected in every time step and 2) boosting the process of finding compact Markovian embeddings. We propose a neural architecture separated into two parts: 1) the transition model and 2) the policy. The transition model is in charge of learning the dynamics of the environment in a supervised manner, using samples collected by the agent. The policy part is shaped using only corrective feedback. Figure 2 shows this architecture.

Learning to predict the next observation o_{t+1} forces a Markovian SR. This has been successfully applied in RL [16]. RNNs can encode information from past observations in their hidden state h_t^{LSTM}. Thus, the objective of the first part of the NN is to learn $\mathcal{M}(o_t, a_t, h_{t-1}^{\text{LSTM}}) = \tilde{o}_{t+1}$, which, as a consequence, learns to embed past observations in h_t^{LSTM}. Additionally, when the observations are high-dimensional (raw images), the agents also need to learn to compress spatial information. To achieve this, a common approach is to compress this information in the latent space of an autoencoder.

For the first part of the architecture, we propose a combination of an autoencoder with an LSTM to compute the transition function, i.e., predicting the next high-dimensional observation. A detailed diagram of this architecture can be seen in Figure 3. In the second part of the architecture, the policy takes as input a representation of the state $\hat{s}_t$, which is generated inside the transition model network. This representation is obtained at the output of a fully connected layer (FC3) that combines the information of h_{t-1}^{LSTM} with the encoder compression of the current observation $e(o_t)$. This is achieved by adding a skipping connection between the output of the encoder and that of the LSTM.

Interactive Algorithm for Policy and World-Model Learning

Algorithm 3 presents the pseudocode of the SRL strategy. The hidden state of the LSTM is denoted as h^{LSTM} and the human corrective feedback as h. In every time step, a buffer $\mathcal{D}$ stores the samples of the transitions with sequences of length τ (line 5). The agent executes an action based on its last observation and the current hidden state of the LSTM (line 6). This hidden state is updated using its previous value and the most recent observation and action (line 7). Line 8 captures the occasional feedback of the teacher, which could be a relative correction when using D-COACH or a corrective demonstration when using HG-DAgger. Also, depending on the learning algorithm, the policy is updated in different ways (line 9). $\mathcal{D}$ replays past transitions of the environment to update the transition function model (line 11). This is done following the bootstrapped random updates [13] strategy. This model is updated every d time steps.

Algorithm 2: D-COACH

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

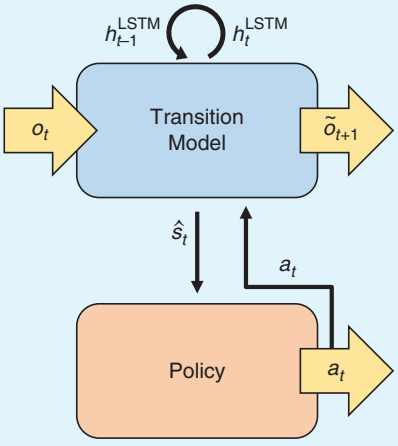

Figure 2. The general structure of the transition model and policy.

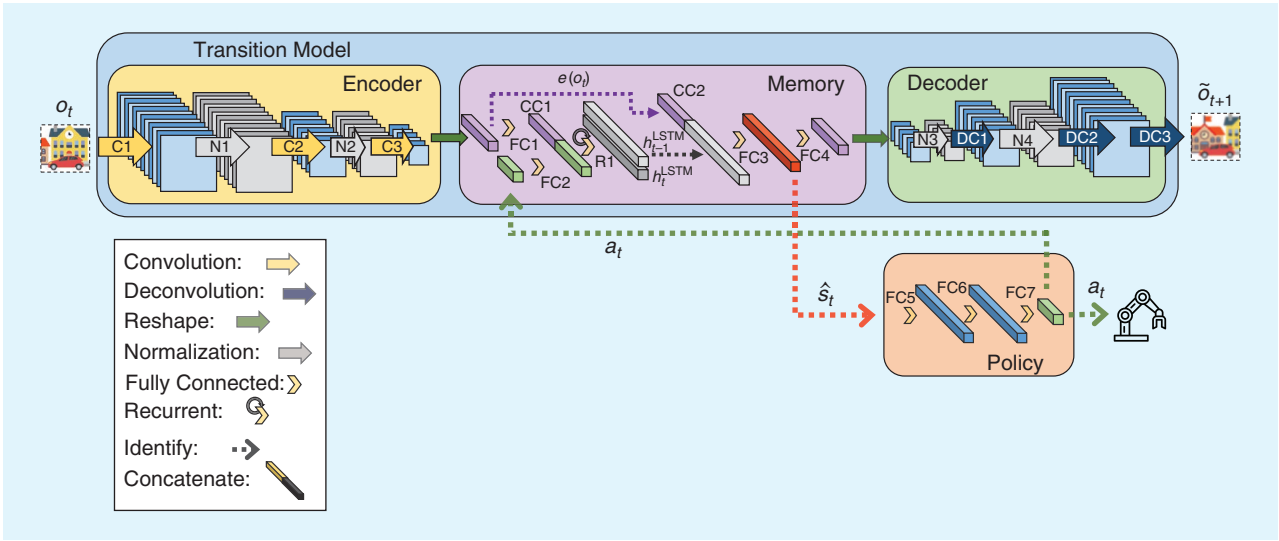

Figure 3. The proposed NN architecture. Convolutional and recurrent (LSTM) layers are included in the transition model for learning of spatiotemporal SRs. The estimated state $\hat{s}_t$ is used as input to the policy, which is a fully-connected NN. C: convolution; N: normalization; FC: fully connected; CC: concatenate; R: recurrent; DC: deconvolution. (Source: Vectors Market, Smashicons, Freepik, and Smalllikeart from Flaticon.)

Algorithm 3: Online temporal feature learning

Require: Policy update algorithm π^{update}, training sequence length τ, model update rate d
Init: $\mathcal{D} = []$
for t = 1, 2, … **do**
  observe observation o_t
  append $(o_{t-1}, \ldots, o_{t-\tau}, a_{t-1}, \ldots, a_{t-\tau}, o_t)$ to $\mathcal{D}$
  execute action $a_t = \pi_\theta(o_t, h_{t-1}^{\text{LSTM}})$
  compute h_t^{LSTM} from $\mathcal{M}(o_t, a_t, h_{t-1}^{\text{LSTM}})$
  feedback human feedback h_t
  call $\pi^{\text{update}}(o_t, a_t, h_t)$
  if mod(t, d) is 0 **then**
    update $\mathcal{M}$ using SGD with minibatches of sequences sampled from $\mathcal{D}$

Experiments and Results

In this section, we present experiments for validating the proposed NN architecture and the interactive training algorithm. To obtain a thorough evaluation, different experiments are carried out to compare and measure the performance (the return, i.e., the sum of the rewards) of the proposed components. Initially, the network architecture based on SRL is evaluated in an ablation study aiming to quantify the data efficiency improvement added by the network architecture's different components. Then, using the proposed architecture, D-COACH is compared with HG-DAgger using simulated tasks and simulated teachers (oracles). The third set of experiments is performed with human teachers in simulated environments, again comparing different learning methods. Finally, a fourth set of validation experiments is conducted in real systems with human teachers. Most of the results are presented in this article; however, some are in the supplementary material, which can be found in IEEE *Xplore* along with more detailed information about the experiments.

Two real and three simulated environments with different complexity levels are used, all employing raw images as observations. The simulated environments are mountain–car, swing-up pendulum, and car racing, the implementations of which are taken from OpenAI Gym [17]. These simulations provide rendered image frames as observations of the environment. The frames visually describe the position of the system but not its velocity, which is necessary to control the system. The experiments on the real physical systems consist of a swing-up pendulum and a setup for picking oranges on a conveyor belt with a three degrees of freedom (3 DoF) robot arm. The metrics used for the comparisons are the achieved final policy performance and the speed of convergence, which is very relevant when dealing with real systems and human teachers. A video showing most of these experiments can be found at https://youtu.be/4kWGfNdm21A.

Ablation Study

In this ablation study, the architecture of the network is the independent variable. Three independent comparisons were carried out using DAgger, HG-DAgger, and D-COACH. The training sessions were run using a simulated teacher to avoid any influence from human factors. Three different architectures were tested for learning the policy from an oracle. The structure of the networks is introduced in the following:

1) *Full network*: the proposed architecture
2) *Memoryless SRL (M-Less SRL)*: similar to the full network but without using recurrence between the encoder and decoder (the autoencoder is trained using the reconstruction error of the observation)

3) *Direct policy learning (DPL)*: the same architecture as in the full network but without using SRL, i.e., not training the transition model (the encoding, recurrent layers, and policy are trained using only the cost of the policy).

The ablation study is done on a modified version of the car racing environment. Normally, this environment provides an upper view of a car on a race track. In this case, we occluded the bottom half of the observation such that the agent was not able to precisely know its position on the track. This position can be estimated if past observations are taken into account. As a consequence, this is an appropriate setting for making a comparison of different NN architectures. Table 1 gives the various performances obtained by the learning algorithms when modifying the structure of the network. The results show a normalized averaged return through 10 repetitions for each experiment, in which five evaluations were carried out for each of the repetitions.

As expected, DAgger with the full architecture obtained the best performance, and, given that it received new samples every time step, it was robust against changes in the architecture, even when it did not have memory. On the other hand, D-COACH was very sensitive to changes in the architecture, especially the DPL architecture. This shows how the full model is able to enhance the performance of the agents in problems where temporal information is required. It even makes D-COACH perform almost as well as DAgger, despite the fact that the former does not require constant and perfect teacher feedback. Finally, HG-DAgger was more robust than D-COACH in the DPL case, but its performance with the full model was not as good.

Table 1. A comparison of the performance (return) of different learning methods in the car racing problem.

	Full	M-Less SRL	DPL
D-COACH	0.97	0.76	0.68
DAgger	1	0.87	0.96
HG-DAgger	0.89	0.69	0.9

The returns were normalized with respect to the best performance (DAgger full).

Simulated Tasks With Simulated Teachers

In the second set of experiments, we performed a comparison among the DAgger, HG-DAgger, and D-COACH algorithms using the proposed full network architecture. To keep the experiments free of human-factor effects, the teaching process was, once again, performed with simulated teachers. The methods were tested in the mountain–car (in the supplementary material) and swing-up pendulum simulated problems. A mean of the return obtained through 20 repetitions is presented for these experiments, along with the maximum and minimum values of these distributions.

Swing-Up Pendulum

In the case of the swing-up pendulum, the results are very different for both DAgger agents (see Figure 4), which have a higher rate of improvement than D-COACH during the first minutes, when the policy is learning the swinging behavior. Since the swinging part requires large actions, the improvement with D-COACH is slower. However, once the policy is able to swing the pendulum up, the second part of the task is to keep the balance in the upright position, which requires fine actions. It is at this point that learning becomes easier for the D-COACH agent, which obtains a constant and faster improvement than the HG-DAgger agent, even reaching a higher performance. In Figure 4, the expected performance upper bound is indicated by a black dashed line, which is the return obtained by the simulated teacher. The purple dashed line shows the performance of a random policy, which is the expected lower bound.

Simulated Tasks With Human Teachers

The previous experiments give insights into how the policy architectures and/or the learning methods perform when imitating an oracle. Most IL methods are intended for learning from any source of expert demonstrations. The source does not necessarily have to be a human; it can be any type of agent. However, the focus of this article is on learning from non-expert human teachers, who are complex to model and simulate. Therefore, conclusions have to be based on results that also include validation with real users.

Experiments with the mountain–car (in the supplementary material) and swing-up pendulum were run with eight human teachers. In this case, the classical DAgger approach was not employed since, as discussed in the "Interactive Learning Methods" section, it is not specifically designed for human users. Instead, HG-DAgger was validated.

Swing-Up Pendulum

This task is relatively simple from a control theory point of view. Nevertheless, it is quite challenging for humans to teleoperate the pendulum due to its fast dynamics. Indeed, participants were not able to successfully teleoperate the agent; therefore, unlike the mountain–car task, we could consider the participants as non-experts in the undertaking.

Figure 5 displays the results of this experiment, which are similar to the ones presented in Figure 4. At the beginning, D-COACH has a slower improvement when learning to swing up; however, it learns faster than HG-DAgger when the policy needs to learn the accurate task of balancing the pendulum. For users, it is more intuitive and easier to improve the balancing with the relative corrections of D-COACH than with the perfect corrective demonstrations of HG-DAgger, as users do not need to know the right action but, rather, just the direction of the correction. Unlike the performance of the simulated teacher depicted in Figure 4, the plot in Figure 5 shows the performance of the best human teacher teleoperating the pendulum with the same interface used for the

teaching process. It can be seen that using both agents facilitated obtaining policies that outperformed the non-expert human teachers. All policies trained with D-COACH were able to balance the pendulum, whereas, with HG-DAgger, the success rate was half as high. Additionally, after the experiment, the participants were queried about which learning strategy they preferred. Seven out of eight expressed a preference for D-COACH.

Validation on Physical Systems With Human Teachers

The previous experiments performed comparison studies of the NN architectures and learning methods under controlled conditions in simulated environments. In this section, D-COACH is validated with human teachers and real systems through two different tasks: 1) a real swing-up pendulum and 2) a fruit-classifier robot arm. The real swing-up pendulum is a very complex system for a human to teleoperate. Its dynamics are faster than that of the OpenAI Gym simulated one used in the previous experiments. The supplementary material provides more details of this environment along with the learning curve of the agents trained by the participants of this validation experiment. Those results as well as the video show that non-expert teachers can manage to teach good policies.

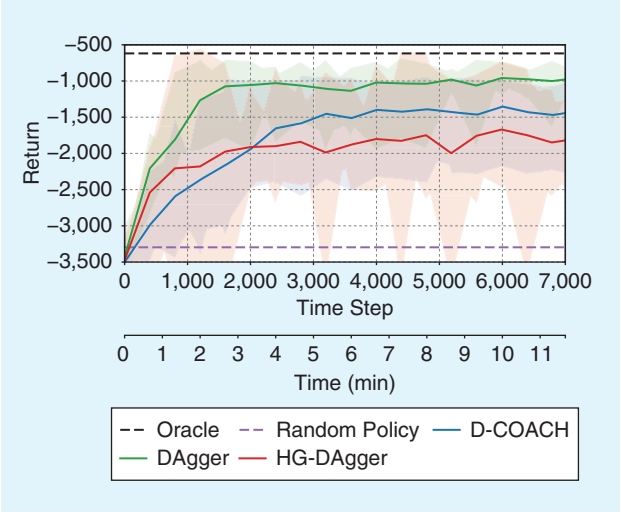

Figure 4. The D-COACH and HG-DAgger comparison in the swing-up pendulum problem using a simulated teacher.

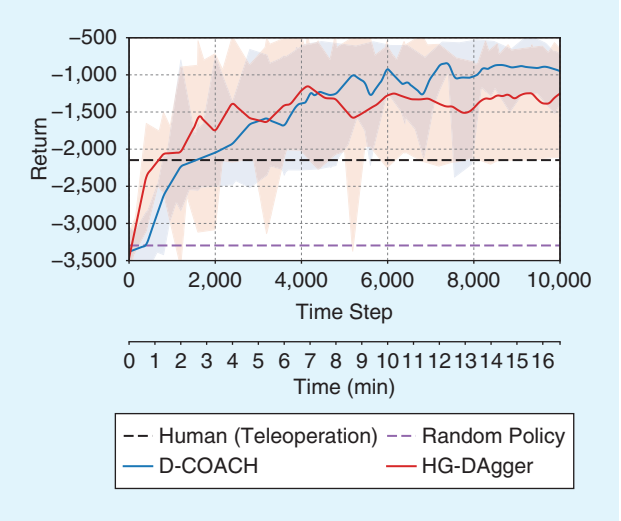

Figure 5. The simulated swing-up pendulum learning curve with human teachers.

Orange Selector With a Robot Arm

This setup consists of a conveyor belt transporting "pears" and "oranges," a 3-DoF robot arm located over the belt, and a red–green–blue (RGB) camera with a view of the belt from above (Figure 6). The image of the camera does not capture the robot arm. The robot has to select oranges with the end effector and avoid pears. The robot does not have any tool, such as a gripper or vacuum gripper, to pick up the oranges. Therefore, in this context, we consider a successful selection of an orange when the end effector intersects the object. The performance of the learning policy is measured using two indices: 1) the orange selection success rate and 2) the pear rejection success rate.

The observations obtained by the camera are from a different region of the conveyor belt than where the robot is acting. Therefore, observations cannot be used to compensate for the robot position in the current time step; rather, they are meaningful for future decisions. In other words, the current action must be based on past observations. Indeed, the delay between the observations and their influence on the actions is roughly 1.5 s. This delay is given by the difference between the time when the object leaves the camera range and the time when it reaches the robot's operating range, which is why this task requires learning temporal features for the policy.

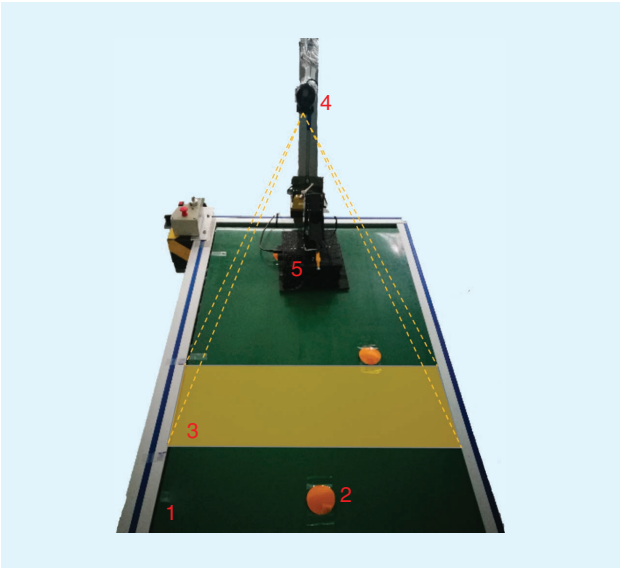

Figure 6. The orange-selector experimental setup. The 1: conveyor belt; 2: orange samples; 3: frame observed by the camera; 4: RGB camera; and 5: 3-DoF robot arm.

The problem is solved by splitting it into two subtasks that are trained separately:

1) *Orange selection*: The robot must intercept the orange coordinate with the end effector exactly when the orange coordinate passes beneath the robot.
2) *Pear rejection*: The robot must classify between oranges and pears, so, when a pear is approaching, the end effector should lift far from the belt plane; otherwise, it should get close.

These two subtasks can be trained sequentially. The orange selection is initially trained through a procedure in which there are some oranges being transported at a fixed position on the belt while some others are placed randomly. This is to avoid overfitting the policy to specific sequences. When the robot is able to track the oranges within its reach, the pear rejection learning starts. For that, pears are placed randomly throughout the sequences of oranges, and the human teacher provides corrections to the robot movement to make the end effector move away from the pears when they are in the operation region of the robot.

Figure 7 depicts the average learning curves for this task after five runs of the teaching process. It is possible to see that the pear rejection subtask is learned within 20 min with 100% success, while the orange selection is a harder subtask that only reaches roughly 80% success after 50 min. Effectively, combining the two subtasks, the performance of the learned policies is given only by the success of the orange selection since the pear rejection was perfectly attained in all runs executed for this experiment.

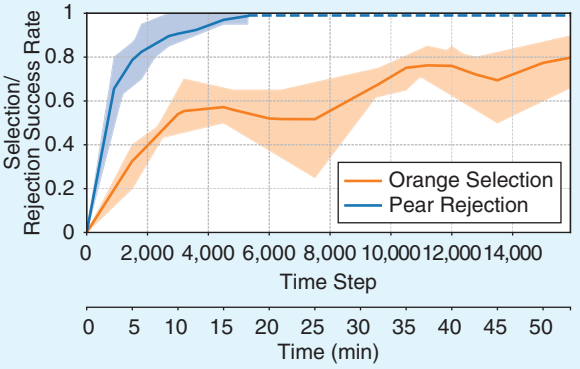

Figure 7. The orange selection/pear rejection learning curve.

Conclusions

This article introduced and validated an SRL strategy for interactively learning policies from human teachers in environments that are not fully temporally observable. Results show that, when meaningful spatiotemporal features are extracted, it is possible to teach complex end-to-end policies to agents using just occasional relative and binary corrective signals. Moreover, these policies can be learned from teachers who are not skilled at executing the task.

The evaluations with the DAgger approaches and D-COACH depict the potential of this kind of architecture to work with different IIL methods, especially those based on occasional feedback, which are intended to reduce the human workload. The comparative results between HG-DAgger and D-COACH with non-expert teachers showed that, with the former, the policy remains biased, with mistaken samples even if the teacher makes sure not to provide more wrong corrections (given that HG-DAgger works with the assumption of expert demonstrations), thus making the policy harder to refine. On the other hand, D-COACH proved to be more robust against mistaken corrections since all non-expert users were able to teach tasks they were not able to demonstrate.

The previously discussed shortcoming of DAgger algorithms opens possibilities for future works intended to study how to deal with databases that have mistaken examples. Another field of study is data-efficient movement generation in animation [19], which, combined with our method, would make it possible to learn (non)periodic movements using spatiotemporal features and IIL. Challenges such as the generation of smooth, precise, and stylistic movements (i.e., dealing with high-frequency details [20]) could be also addressed.

Acknowledgments

This research is funded by the Netherlands Organization for Scientific Research project Cognitive Robots for Flexible Agro-Food Technology, grant P17-01; European Research Council Starting Grant Teaching Robots Interactively, project reference 804907; Chile's National Fund for Scientific and Technological Development project (FONDECYT) 1201170; and Chile's Associative Research Program of the National Research and Development Agency (ANID/PIA) project AFB180004.

References

[1] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *J. Artificial Intell. Res.*, vol. 34, pp. 1–25, Jan. 2009. doi: 10.1613/jair.2584.

[2] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. Kochenderfer, "HG-DAgger: Interactive imitation learning with human experts," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2019, pp. 8077–8083. doi: 10.1109/ICRA.2019.8793698.

[3] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Proc. Advances Neural Information Processing Systems*, 2017, pp. 4299–4307.

[4] W. B. Knox and P. Stone, "Interactively shaping agents via human reinforcement: The TAMER framework," in *Proc. 5th ACM Int. Conf. Knowledge Capture*, 2009, pp. 9–16. doi: 10.1145/1597735.1597738.

[5] C. Celemin and J. Ruiz-del Solar, "An interactive framework for learning continuous actions policies based on corrective feedback," *J. Intell. Robotic Syst.*, vol. 95, pp. 77–97, July 2019. doi: 10.1007/s10846-018-0839-z.

[6] J. MacGlashan et al., "Interactive learning from policy-dependent human feedback," in *Proc. 34th Int. Conf. Machine Learning*, 2017, vol. 70, pp. 2285–2294.

[7] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. doi: 10.1038/nature14236.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

[9] W. Böhmer, J. T. Springenberg, J. Boedecker, M. Riedmiller, and K. Obermayer, "Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations," *KI-Künstliche Intelligenz*, vol. 29, no. 4, pp. 353–362, 2015. doi: 10.1007/s13218-015-0356-1.

[10] T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, and D. Filliat, "State representation learning for control: An overview," *Neural Netw.*, vol. 108, pp. 379–392, Dec. 2018. doi: 10.1016/j.neunet.2018.07.006.

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

[12] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. 14th Int. Conf. Artificial Intelligence and Statistics*, 2011, pp. 627–635.

[13] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Proc. AAAI Fall Symp. Sequential Decision Making for Intelligent Agents (AAAI-SDMIA15)* Arlington, VA, Nov. 2015.

[14] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," in *Proc. 31st AAAI Conf. Artificial Intelligence*, 2017, pp. 2140–2146.

[15] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Proc. Advances Neural Information Processing Systems*, 2018, pp. 2450–2462.

[16] A. Zhang, H. Satija, and J. Pineau, Decoupling dynamics and reward for transfer learning. 2018. [Online]. Available: arXiv:1804.10689

[17] G. Brockman et al., OpenAI gym. 2016. [Online]. Available: arXiv:1606.01540

[18] I. Mason, S. Starke, H. Zhang, H. Bilen, and T. Komura, "Few-shot learning of homogeneous human locomotion styles," *Comput. Graph. Forum*, vol. 37, no. 7, pp. 143–153, 2018. doi: 10.1111/cgf.13555.

[19] H. Zhang, S. Starke, T. Komura, and J. Saito, "Mode-adaptive neural networks for quadruped motion control," *ACM Trans. Graph. (TOG)*, vol. 37, no. 4, pp. 1–11, 2018. doi: 10.1145/3197517.3201366.

Rodrigo Pérez-Dattari, Department of Cognitive Robotics, Delft University of Technology, The Netherlands. Email: r.j.perezdattari@tudelft.nl.

Carlos Celemin, Department of Cognitive Robotics, Delft University of Technology, The Netherlands. Email: c.e.celeminpaez@tudelft.nl.

Giovanni Franzese, Department of Cognitive Robotics, Delft University of Technology, The Netherlands. Email: g.franzese@tudelft.nl.

Javier Ruiz-del-Solar, Department of Electrical Engineering and the Advanced Mining Technology Center, Universidad de Chile, Santiago. Email: jruizd@ing.uchile.cl.

Jens Kober, Department of Cognitive Robotics, Delft University of Technology, The Netherlands. Email: j.kober@tudelft.nl.