IJCNN 2019. International Joint Conference on Neural Networks. Budapest, Hungary. 14-19 July 2019

A Human-Like Agent Based on a Hybrid of Reinforcement and Imitation Learning

Rousslan Fernand Julien Dossa*, Xinyu Lian*, Hirokazu Nomoto†, Takashi Matsubara* and Kuniaki Uehara*
*Graduate School of System Informatics, Kobe University, Hyogo, Japan
Email: {doss@ai.cs., rensy94@ai.cs., matsubara@phoenix., uehara@}kobe-u.ac.jp
†EQUOS RESEARCH Co., Ltd, Tokyo, Japan
Email: i19341_nomoto@aisin-aw.co.jp

***Abstract*—Reinforcement learning (RL) builds an effective agent that handles tasks in complex and uncertain environments by maximizing future reward. However, the efficiency is insufficient for practical use such as game AI and autonomous driving. An effective but selfish agent conflicts with other humans, and hence the demand of a human-like behavior arises. Imitation learning (IL) has been employed to train an agent to mimic the actions of expert behaviors provided as training data. However, IL tends to build an agent limited in performance by the expert skill, and even worse, the agent exhibits an inconsistent behavior since IL is not goal-oriented. In this paper, we propose a training scheme by mixing RL and IL for both discrete and continuous action space problems. The proposed scheme builds an agent that achieves a performance higher than an agent trained by only IL and exhibits a more human-like behavior than agents trained by RL or IL, validated by human sensitivity.**

***Index Terms*—Reinforcement Learning, Imitation Learning, Human-Like Behavior, Game AI, Autonomous Driving**

I. Introduction

Reinforcement learning (RL), where an agent learns a policy by interacting with an environment, has achieved impressive progress in many areas recently, such as Go [1], autonomous driving [2]–[5], and video games [6], [7]. Since the RL model is trained to maximize future reward, an RL agent tends to be efficient. However, high efficiency is not the only important factor in practical use. In a game, an RL non-player character (NPC) instantiated as an opponent may be too strong to be defeated, resulting in the player being quickly frustrated and not enjoying the game. Similarly, a solely efficiency oriented RL autonomous vehicle is likely to inconvenience the surrounding cars and pedestrians, for example by taking abrupt turns, or more generally by exhibiting a selfish, dangerous behavior humanly hard to predict. Hence, an approach to build a human-like agent is indeed desirable. On the other hand, imitation learning (IL) trains an agent to learn a policy from training data provided by human experts [8], [9]. Therefore, we can expect an agent trained by IL to behave in a human-like way. However, as IL is not in a goal-oriented way but is limited to the demonstration data provided by the human expert, the performance of an IL agent is likely to be capped to the former's.

This study was supported by JST-Mirai Program Grant Number JPMJMI18B4, Japan, and partially supported by EQUOS RESEARCH Co., Ltd.

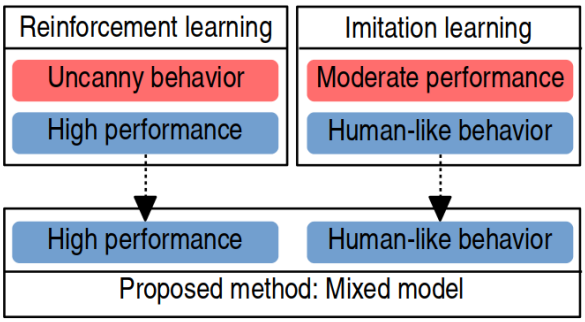

Fig. 1: Building a hybrid model between Reinforcement and Imitation learning

In this paper, we aim to produce an agent behaving in a human-like way while retaining high performance. An intuitive overview is presented in the Fig. 1. We propose a hybrid model based on RL and IL. We demonstrated the performance level of an RL agent and tendency of human expert's behavior could be transferred via distillation and IL to a student agent. We applied our approach to an original game and an Atari game Gopher as environments of discrete action spaces. We also applied our approach to the TORCS racing simulator [10] as an environment of a continuous action space, with an aim of demonstrating its potential in real life applications such as autonomous driving, where a human-like agent is highly desirable. A performance test and a sensitivity test (like Turing test) in double-blind fashion demonstrated that our approach built agents that achieve better performances and exhibit more human-like behaviors compared to RL and IL agents.

II. Related Work

A. Reinforcement Learning and Deep Q-Networks

Let us introduce the reinforcement learning (RL) through the framework of Markov decision

978-1-7281-2009-6/$31.00 paper N-20026.pdf

processes (MDP). An MDP is defined by a 5-tuple $< \mathcal{S}, \mathcal{A}, P_a, R_a, \gamma >$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ the set of actions, $P_a(s, s') = P_r(s_{t+1} = s' | s_t = s, a_t = a)$ the probability that state s will transition to state s' due to action a on time-step t, $R_a(s, s')$ is the reward received after transition from state s to the state s', and γ $(0 < \gamma \leq 1)$ is the discount factor that represents the importance of future rewards compared to the present rewards. During training, on every time-step t, the agent obverses a state $s_t \in \mathcal{S}$ and performs an action $a_t \in \mathcal{A}$. In response, the environment returns the corresponding reward r_t and next state s_{t+1}.

In discrete action space cases, Deep Q-Network (DQN) algorithm [6] has proven itself by achieving superhuman performance on numerous experiments. From an arbitrary state s_t at time-step t, the agent is trained to predict a future return $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$,where T is the terminal time-step. The agent then takes an action that maximizes R_t according to the prediction. The optimal action-value function is defined as $Q^*(s_t, a_t) = \max_\pi \mathbb{E}[R_t | s = s_t, a = a_t, \pi]$, to which applying the Bellman equation yields:

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathcal{E}}[r_t + \gamma \max_{a_t} Q^*(s_{t+1}, a_{t+1}) | s_t, a_t].$$

During training, the agent estimates R_t by the action-value function $Q(\phi(s), a)$, where ϕ is a preprocessed function that produces the fixed length representation of s, and updates it iteratively as Algorithm 1, the $Q(\phi(s), a)$ will finally converge to the optimal action-value function $Q_i \to Q^*$ as $i \to \infty$ [11]. The complete process is presented in Algorithm 1

Algorithm 1 Deep Q-learning with Experience Replay

Initialize the replay memory $\mathcal{D}$ to capacity N
Initialize the action-value function Q with random weights
for episode$= 1, M$ **do**
  Initialize the sequence $s_1 = \{x_1\}$ and the preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe the reward r_t and the resulting image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j \text{ for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi(s_j + 1), a'; \theta) \text{ otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

B. Deep Deterministic Policy Gradient

We redefine the action space as $\mathcal{A} \subset \mathbb{R}^N$ to be continuous, and set the goal to obtain a policy π which maximizes the expected return from the start distribution $J = \mathbb{E}_{r_i, s_i \sim \mathcal{E}, a_i \sim \pi}[R_1]$. While DQN achieves strong performance over a high dimensional state space, it was proven to be limited to low dimension discrete action spaces. The Deep Deterministic Policy Gradient (DDPG) method [12] is based on the Deterministic Policy Gradient (DPG) method [13], which defines the policy of the RL agent as an actor function $\mu(s|\theta^\mu)$ parametrized by θ^μ and mapping from a state s_t to a corresponding action a_t, and a critic function $Q(s_t, a_t)$ to approximate the value of a state-action pair (s_t, a_t). The actor update is then performed accordingly to the *policy gradient* as defined by David Silver et al. [13].

By introducing deep and non-linear approximators, the update of the critic function $Q(s, a|\theta^Q)$ becomes prone to divergence. In response, the DDPG method leverages 'soft' target updates by creating copies $\mu'(s|\theta^{\mu'})$ and $Q'(s, a|\theta^{Q'})$ to be used to compute the target values. These target networks however are updated by slowly tracking original networks: $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ with $\tau << 1$ to increase the stability of the training.

Given that a large action space requires much more exploration to converge to the optimal solution, an exploration policy $\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}$ with an added noise sampled from a noise process $\mathcal{N}$ which is chosen depending on the nature of the task or the environment [14] is adopted. The full procedure is detailed in Algorithm 2.

C. Imitation Learning

We assume the policy followed by the expert players during demonstrations as the optimal policy π^*, and the agent learns a policy π which approximates the optimal policy π^*. A traditional approach is to train an agent by supervised learning. The expert human players provide trajectories that consist of sequences of state-action pairs as the training data follows an optimal policy π^*. The agent's policy is trained to predict actions based on the provided states and is thereby able to imitate the behavior of the expert.

D. Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning (GAIL) [15] is a method based on both Inverse Reinforcement Learning (IRL) and classical RL. IRL first fits a cost function c from a family of function $\mathcal{C}$ assigning low cost to the expert policy π_E and high cost to the other policies π:

$$\max_{c \in \mathcal{C}} \left(\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_\pi[c(s, a)] \right) - \mathbb{E}_{\pi_E}[c(s, a)]$$

where Π is the set of all stationary stochastic policies mapping from $\mathcal{S}$ to $\mathcal{A}$ and $H(\pi) \triangleq \mathbb{E}_\pi[-log\pi(a|s)]$ is the γ-discounted causal entropy of the policy π [15]. On the

Algorithm 2 Deep Deterministic Policy Gradient algorithm

Randomly initialize the critic network $Q(s,a|\theta^Q)$ and the actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ respectively
Initialize the target networks Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$ respectively
Initialize the replay buffer R
for episode= $1, M$ **do**
  Initialize a random noise process $\mathcal{N}$ for action exploration
  Obtain the initial observation state s_1
  for $t = 1, T$ **do**
    Select an action $a_t = \mu(s|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute the action a_t and observe the corresponding reward r_t and new state s_{t+1}
    Store the transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss:
    $$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$
    Update the actor policy using the sampled policy gradient:
    $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\nabla_\alpha Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$
    Update the target networks:
    $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
    $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
  end for
end for

Algorithm 3 Generative Adversarial Imitation Learning

Input: Expert trajectories $\tau_E \sim \pi_E$, initial policy and Discriminator parameters θ_0, w_0
for $i = 0, 1, 2, ...$ **do**
  Sample trajectories $\tau_i \sim \pi_{\theta_i}$
  Update Discriminator parameters from w_i to w_{i+1} with the gradient
  $\hat{\mathbb{E}}_{\tau_i}[\nabla_w log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w log(1 - D_w(s,a))]$
  Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $log(D_{w+1}(s,a))$.
  Specifically, take a KL-constrained natural gradient step with
    $\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta log\pi_\theta(a|s)Q(s,a)] - \lambda\nabla_\theta H(\pi_\theta)$,
  where $Q(\overline{s}, \overline{a}) = \hat{\mathbb{E}}_{\tau_i}[log(D_{w_{i+1}}(s,a))|s_0 = \overline{s}, a_0 = \overline{a}]$
end for

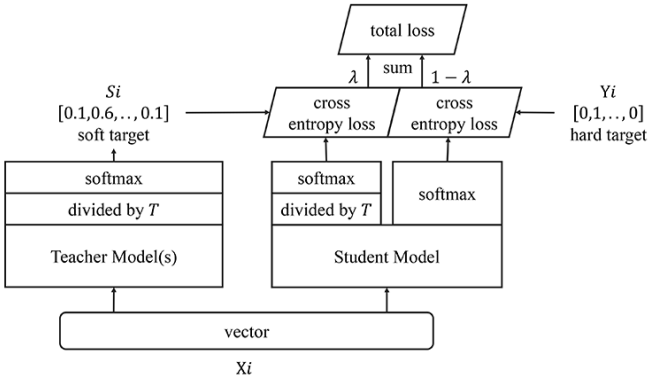

Fig. 2: Procedure of knowledge distillation.

other hand, classical RL finds the optimal policy from the cost function c extracted by IRL:

$$RL(c) = \underset{\pi\in\Pi}{\operatorname{argmin}} -H(\pi) + \mathbb{E}_\pi[c(s,a)]$$

corresponding to the expert policy π_E. To bypass the intermediate IRL step, J. Ho and S. Ermon intoduced the concept of occupancy measure ρ_π of a policy π, which can be interpreted as the distribution of state-action pairs that an agent encounters when navigating the environment by following said policy [15]. A function approximator $D_w : \mathcal{S} \times \mathcal{A} \rightarrow (0,1)$ is used to discriminate between the occupancy measure ρ_π of the student policy and the expert ρ_{π_E} by minimizing the sum of their respective expectations over generated trajectories:

$$\mathbb{E}_\pi[log(D_w(s,a))] + \mathbb{E}_{\pi_E}[log(1 - D_w(s,a))] - \lambda H(\pi), \lambda \geq 0$$

until D_w can no longer distinguish between the π and π_E, meaning that the student has learned the expert's policy. Algorithm 3 recapitulates the complete procedure.

E. Distillation

Knowledge distillation is an approach that trains a student model based on teacher model(s), where instead of a one-hot label commonly used in traditional supervised learning, the teacher model trains the student model by the model output which is called *soft target* [16]. Compared to the one-hot label, the values of each dimension of the soft target include more information. With a visual input of a cat, of course, the probability of it being classified as "cat" by the model would be the greatest, and the probability of "dog" would be greater than that of "carrot", which matches with the fact a cat is visually more similar to a dog than it is to a carrot. The procedure of knowledge distillation is presented in Fig. 2. Not only does this approach achieves a higher performance model with less training data, it is also useful to compress large models. The reference [17] has extended this approach to reinforcement learning area, which is named *policy distillation*. They successfully trained the student model that achieves higher performance by extracting the policy of the teacher model. Furthermore, by distilling policies of teacher models trained for different tasks, they demonstrated policy distillation can be applied to multi-task problems.

III. Proposed Method

Recall that our purpose is to build an agent performing similarly to a human expert while retaining the high performance of an RL expert model. This problem can be separated in the two following subproblems: (1) building an agent that has the human-level performance, and (2) building an agent that selects actions similarly to a human expert. As these two subproblems have been tackled by RL and IL respectively, we consider our work as a multi-task learning problem. In this paper, we propose a method for mixing RL and IL by policy distillation for discrete action space case and by adversarial imitation learning for the continuous action space case.

In general, we express the objective function of IL as $\mathcal{L}_{\pi^*}(\pi)$, where π^* is a teacher policy to be imitated by the student policy π. Then, the proposed objective function is

$$\mathcal{L}_{mix}(\pi) = \alpha\mathcal{L}_{\pi_{RL}}(\pi) + (1-\alpha)\mathcal{L}_{\pi_{HE}}(\pi),$$

where π_{RL} is an optimal policy built by RL and π_{HE} is a policy of a human expert (HE), and α is a trade-off coefficient between the RL policy and human expert.

In the case of discrete action space, we define the objective function of IL as the following cross-entropy loss

$$\mathcal{L}_{\pi^*}(\pi) = \mathbb{E}_s\left[-\textstyle\sum_a \pi^*(a|s)\log\pi(a|s)\right],$$

following previous studies on IL and distillation [16], [17]. Since the policy π_{HE} of a human expert is not mathematically modeled but given as empirical demonstrations, we cannot obtain soft labels of the policy for distillation. Fortunately, the study on policy distillation improved the performance by additionally using hard labels during the training, namely computing a weighted average of the hard and soft labels [17]. Following this, we used human expert demonstrations π_{HE} as hard labels and the policy $\pi_{RL}^{(T)}$ of a DQN model [6] for soft labels with a temperature T while a typical policy of a DQN model employs $T \to 0.0$ (i.e., provides hard labels). Then, the final objective function is

$$\begin{aligned}\mathcal{L}_{mix}(\pi) =& \alpha\,\mathbb{E}_{\pi_{RL}}\left[-\textstyle\sum_a \pi_{\mathrm{RL}}^{(T)}(a|s)\log\pi(a|s)\right] \\ &+ (1-\alpha)\,\mathbb{E}_{\pi_{HE}}\left[-\textstyle\sum_a \pi_{HE}(a|s)\log\pi(a|s)\right],\end{aligned}$$

A conceptual diagram of the entire procedure is shown in Fig. 3.

In the continuous action space case, we employed the GAIL method introduced in Section II-D. The GAIL requires empirical trajectories $\tau^* \sim \pi^*$ obtained from a teacher policy π^* for training. The objective function of the discriminator to be maximized by D_w and to be minimized by a student policy π is:

$$\mathcal{L}_{\pi^*}(\pi) = \mathbb{E}_{\tau\sim\pi}[\log(D_w(s,a))] + \mathbb{E}_{\tau^*\sim\pi^*}[\log(1-D_w(s,a))],$$

where τ denotes trajectories $\tau \sim \pi$ sampled from a student policy π. With a human expert and an RL agent respectively providing trajectories $\tau_{HE} \sim \pi_{HE}$ and $\tau_{RL} \sim \pi_{RL}$, the hybrid loss function can be written as:

$$\begin{aligned}\mathcal{L}_{mix}(\pi) =& \mathbb{E}_{\tau\sim\pi}[\log(D_w(s,a))] \\ &+ \alpha\,\mathbb{E}_{\tau_{RL}\sim\pi_{RL}}[\log(1-D_w(s,a))] \\ &+ (1-\alpha)\,\mathbb{E}_{\tau_{HE}\sim\pi_{HE}}[\log(1-D_w(s,a))],\end{aligned} \tag{1}$$

Intuitively, having the discriminator trained to recognize an *intermediate* hybrid policy between the human and RL expert respective policies, the student model — which is trained to learn a policy to fool the discriminator to classify its actions as being the expert's — is expected to converge to said hybrid policy, thereby exhibiting behaviors inherited from both experts. A conceptual diagram of the entire procedure is shown in Fig. 4.

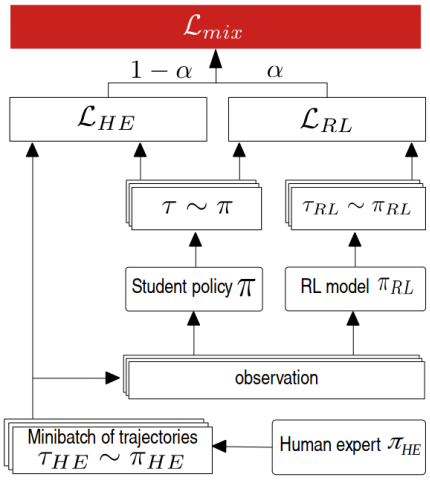

Fig. 3: Building an agent based on the hybrid loss : discrete action space case

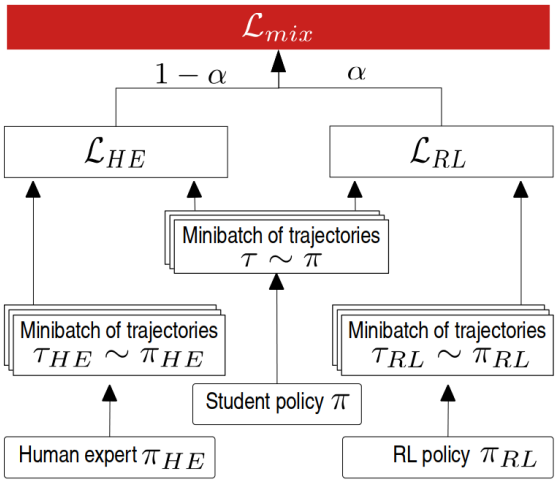

Fig. 4: Building an agent based on the hybrid loss: continuous action space case

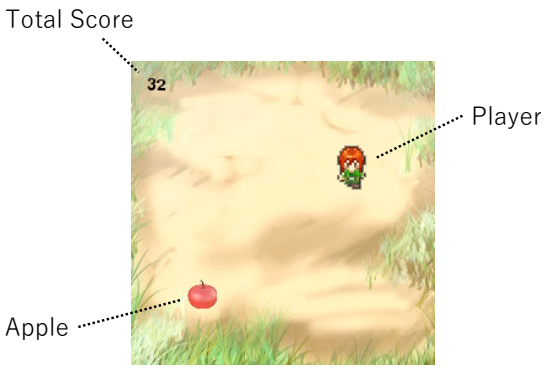

Fig. 5: A screenshot of the *Apple Game*.

Fig. 6: A screenshot of *Gopher*.

IV. Experiments

A. Apple Game

We first applied our approach to an original game called *apple game*, whose screenshot is provided in Fig. 5. In this game, at any time-step, there is one and only one apple in the environment, which disappears after being taken by the player or after a certain period of time has passed, then reappears randomly somewhere else. The goal of the agent is to take the apple to score one point each. The action space $\mathcal{A}$ contains moves in 8 directions and rest, that is, $\mathcal{A} = [(-1,-1),(0,-1),...,(1,1)]$, $|\mathcal{A}| = 9$. Hence, the action space is discrete.

From each expert, we collected gameplay a total of 16,000 frames, which were randomly sampled into training and test sets of size 14,500 and 1,500 respectively during the training, to avoid overfitting and reliably test the accuracy of the model. We trained a DQN model for comparison and to serve as the RL teacher. The student model was trained using Adam optimizer [18] with a learning rate of 10^{-4} and dropout ratio of 0.5 [19]. We set the temperature $T = 0.1$ for distillation and the trade-off coefficient $\alpha = 0.93$.

Table I: Hyperparameters used for DQN training on *Apple game* and *Gopher*

Hyperparameter	**Apple**	**Gopher**
Total frames T_{play}	$8 \cdot 10^6$	$1.001 \cdot 10^7$
Optimizer	Adam	
Learning rate	10^{-6}	10^{-4}
Batch size	32	
Update frequency	4	
Loss	MSE	
Replay memory	$2 \cdot 10^5$	10^6
Observed frame	$4 \cdot 10^5$	10^4
Discount rate	0.9	0.99
Initial exploration ε_{T_0}	1.0	
Final exploration $\varepsilon_{T_{exploration}}$	0.1	0.01
Final exploration frame $T_{exploration}$	$4 \cdot 10^6$	10^7
Frame skip T_{skip}	4	
History length $L_{history}$	4	
Image size $S \times S$	84×84	

Table II: Training hyperparameters for hybrid agent on *Apple game* and *Gopher*

Hyperparameter	**Apple game**	**Gopher**
Total frames T_{play}	$2 \cdot 10^5$	
Optimizer	Adam	
Learning rate	10^{-4}	
Dropout rate	0.5	
Temperature τ	0.1	
average weight α	0.93	0.8

B. Atari 2600 Game: Gopher

We also applied our method to an Atari 2600 game called *Gopher*, which is slightly more complicated than the *Apple game*. The goal is to prevent the "gopher" — a mole-like creature — from digging its way out of the ground and steal the carrots under the protection of the player by moving laterally and filling up the hole as the gopher dig them. A screenshot of the game is provided in Fig. 6. The training data provided by the human and RL experts consisted of 55,000 frames each, which were also randomly sampled in training and test sets of size 50,000 and 5,000 respectively. We trained the student model following the same method used for the *apple game* except for the trade-off coefficient $\alpha = 0.8$.

C. Torcs

The *Torcs* racing car simulator [10] is a well known environment used for autonomous driving AI research (see [20], [21]). A screenshot of the game is shown in Fig. 7. For this experiment, we used the Gym Torcs environment [22] as a basis. The agent observations consisted of 65 low-level features, i.e. sensors keeping track of the distance between the car and the edges of the track or the opponents, the current speed and acceleration etc., with a bi-dimensional continuous action space to control the steering and the throttling of the car, while using the raced distance as the reward function. The RL agent was based on the DDPG implementation in the OpenAI Baselines [23], which we adapted to the autonomous driving problem and trained using the hyperparameters as specified in Table III.

Fig. 7: A screenshot of Torcs.

Table III: Hyperparameters used for DDPG training on *Torcs*

Hyperparameter	**Value**
Total timesteps T_{play}	10^7
Optimizer	Adam
Critic learning rate	10^{-4}
Actor learning rate	10^{-3}
Epochs	100
Epoch cycles	20
Train steps	50
Rollout steps	100
Replay memory	10^6
Discount rate γ	0.99
Target network update τ	10^{-2}

Table IV: Hyperparameters used for GAIL and Hybrid model training on *Torcs*

Hyperparameter	**GAIL**	**Hybrid**
Episode count	220	
Transition per episode	3600	
Total transitions	$7.92 \cdot 10^5$	
Optimizer	Adam	
Discriminator learning rate	10^{-3}	
Emtropy coefficient λ	10^{-3}	
KL constraint	10^{-2}	
Trade-off coefficient α	None	10^{-3}
Training timesteps	$5 \cdot 10^6$	$7.5 \cdot 10^6$

Furthermore, we upgraded the racing car simulator Torcs to simulate a situation where the human decision factor would stand out more, namely by generating obstacles in the form of stationary bots and added player demonstration recording support, which we used to record 220 episodes of 60 seconds of gameplay, totaling 792,000 transitions.

The strict IL part consisted in imitating a human expert based on recorded data while driving in the same setting as the RL experiment, using the OpenAI [23] implementation of the GAIL method and the hyperparameters value specified in Table IV. Finally, hybrid model, also based on the GAIL implementation but with the discriminator modified according to the Equation 1 described in section III was also trained in the same setting, using $\alpha = 0.5$ to maintain a balance between the two experts. The other hyperparameters value used can be found in Table IV.

D. Evaluation by Sensitivity Test

Besides the performance evaluation of each model, we also conducted a sensitivity test in a double-blind fashion to evaluate the human-likeliness of the agents' behaviors. The test involved 26 participants (23 males and 3 females), their age ranged from 27 to 59, with an average of 44 years old. The participants were employees of EQUOS RESEARCH Co., Ltd, which never had prior exposure to this research's materials or demonstrations, as well as previous studies. They were first introduced to the rules of the games and provided with the opportunity to play the game by themselves to have an approximate idea of the mechanics and how a human would play. For each game (*Apple game*, *Gopher* and *Torcs*), each participant was presented with two 15 seconds (30 seconds for *Torcs*) video of each model playing the game and requested to label it as either human or machine and comment the reasons.

V. Results and Discussion

A. Apple Game

We collected the cumulated rewards over 400 episodes of each agent playing the game. In the *apple game*, the score was the number of apples collected in a 15 seconds interval. Given the simple principle of the game, i.e. moving the player avatar to the spawning apples, the DQN method achieved the highest scores, followed by the proposed method and finally the human agent and its imitation. The detailed results are documented in Table V. As far as human-likeliness is concerned, the human agent came out first. The hybrid model exhibited a human-like behavior more than the RL agent while surpassing the human expert and IL agent in score, followed by the human imitation agent and finally the DQN agent achieved by far the less human-like behavior. Hence, we conclude that the proposed approach balances the human-like behavior and high-performance in this game.

B. Atari 2600 Game: Gopher

From a viewpoint of the performance, the DQN agent achieved the highest score by a large margin, followed by the hybrid agent trained using the proposed method, and finally the human agent and its imitation. The hybrid agent, despite prioritizing the RL label by using $\alpha = 0.8$ to compute the weighted sum of the teachers' labels during the training, only showed an improvement of 3 points over the human agent, and still quite a far way from the DQN agent's score. While the RL agent achieved the best results, only a few participants identified it as a human. In this game, the hybrid agent surpassed the IL agent in both the criteria: game score and human-like behavior. Hence, we conclude that the proposed method built an agent with a human-like behavior by combining the goal-oriented learning scheme of RL and the tendency of human expert's behavior. Surprisingly, the hybrid agent was recognized as a human in the sensitivity test more likely than the

Table V: Results of *Apple Game*

	Game Score				Sensitivity Test
Agent	**Max**	**Min**	**Average**	**Std.**	**Identified as Human (%)**
Human	27	11	18.71	2.86	**64.0**
DQN (RL)	53	15	**36.27**	5.44	8.0
IL	29	3	17.57	4.37	44.0
Proposed method (RL+IL)	35	11	<u>22.02</u>	3.70	<u>54.0</u>

The best and second best results are emphasized by bold fonts and underlines, respectively.

Table VI: Results of *Gopher*

	Game Score				Sensitivity Test
Agent	**Max**	**Min**	**Average**	**Std.**	**Identified as Human (%)**
Human	81	2	23.87	19.81	<u>55.70</u>
DQN (RL)	246	0	**40.30**	36.81	32.69
IL	126	0	23.91	23.79	**59.62**
Proposed method (RL+IL)	132	0	<u>26.05</u>	24.31	**59.62**

The best and second best results are emphasized by bold fonts and underlines, respectively.

Table VII: Results of *Torcs*

	Game Score ($\times 10^3$)				Sensitivity Test
Agent	**Max**	**Min**	**Average**	**Std.**	**Identified as Human (%)**
Human	48.70	27.68	40.17	3.63	50.00
DDPG (RL)	41.91	39.91	**40.45**	0.43	30.77
IL	39.93	14.88	23.99	1.16	<u>51.92</u>
Proposed method (RL+IL)	40.22	6.71	<u>36.63</u>	1.32	**61.54**

The best and second best results are emphasized by bold fonts and underlines, respectively.

human agent. To shed the light on this trend, we extracted comments tied to the misclassifieda human player demonstrations such as: "the player systematically fills the holes", "there are only a few futile movements" or "the movements are machine-like". It appears the performance of the human expert was underestimated or the level of the human expert too high, especially regarding participants not so familiar with video games.

C. Torcs

The first evaluation phase consisted in measuring the performance of the proposed model and compare it against the human, GAIL human imitation, and the DDPG agent respectively, as documented in Table VII.

While the GAIL method excelled at imitating experts trajectories strictly generated by RL policies (results of [15]), deterministic bots coming with the Torcs racing car simulator, or even the DDPG agent, it turned out to be less effective when applied on a human expert's trajectories, achieving at best around half the score of the latter. We hypothesized it to be a consequence of the human expert's policy being relatively more complex and hard to approximate with standard neural networks structures.

Meanwhile, the hybrid model was able to capture and demonstrate specific behaviors of both the human expert as well as the RL agent, namely a higher speed than the imitated human expert and deceleration during turns respectively. In addition, it was also able to complete full laps, like the human expert and the DDPG agent.

By increasing order of human-likeliness, we have the DDPG agent, which was given away by its high-performance behavior, with comments such as "drives really fast" or "can take turns with high speed". To our surprise, the human agent itself scored lower than what could be expected. For a given human demonstration, diverging opinions were voiced. We hypothesize this is could be the result of personal interpretation of the participants, who may have not expected such a performance demonstrated from the expert. In addition, the human imitation agent which achieved a lower performance than the expert has a better success rate, which seems to match with the previous hypothesis. Finally, the hybrid model which was rated the most human-likely, while nearing the performance of the DDPG agent.

VI. Conclusion

This study proposes a method to build a hybrid agent which behaves in a human-like fashion imitated from a human expert while retaining some of the high performance displayed by pure reinforcement learning agents, ultimately aiming for the best of both worlds. We based our method on state-of-the-art reinforcement and imita-

tion learning algorithms and proposed two variants for discrete and continuous action spaces respectively. We applied said method to an original game and an Atari2600 game for the discrete action space case, and on the Torcs racing car simulator for the continuous action space case. We first evaluated its performance before evaluating its human-likeliness through a sensitivity test. The proposed method successfully exhibited behaviors borrowed from both experts, namely an increase of performance following the reinforcement learning expert as well as a human-like behavior following the human counterpart.

References

[1] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[2] A. E. Sallab *et al.*, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[3] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[4] D. Isele *et al.*, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2034–2039.

[5] B. Vikas, "Deep reinforcement learning approach to autonomous navigation," 2017.

[6] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[7] ——, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

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

[9] J. Ortega *et al.*, "Imitating human playing styles in super mario bros," *Entertainment Computing*, vol. 4, no. 2, pp. 93–104, 2013.

[10] B. Wymann *et al.*, "Torcs: The open racing car simulator," 2015.

[11] R. S. Sutton *et al.*, *Reinforcement learning: An introduction*. MIT press, 1998.

[12] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[13] D. Silver *et al.*, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, pp. I–387–I–395.

[14] M. Plappert *et al.*, "Parameter space noise for exploration," *CoRR*, vol. abs/1706.01905, 2017.

[15] J. Ho and S. Ermon, "Generative adversarial imitation learning," *CoRR*, vol. abs/1606.03476, 2016.

[16] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[17] A. A. Rusu *et al.*, "Policy distillation," *CoRR*, vol. abs/1511.06295, 2015.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, 2014.

[19] N. Srivastava *et al.*, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, 2014.

[20] B. Lau, "Using keras and deep deterministic policy gradient to play torcs," 2016. url: https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html

[21] Y. You, "Torcs for reinforcement learning." url: https://github.com/YurongYou/rlTORCS

[22] N. Yoshida, "Gym torcs," 2016. url: https://github.com/ugo-nama-kun/gym-torcs

[23] P. Dhariwal *et al.*, "Openai baselines," 2017. url: https://github.com/openai/baselines