Internal Model from Observations for Reward Shaping

Daiki Kimura[1], **Subhajit Chaudhury**[1], **Ryuki Tachibana**[1], **Sakyasingha Dasgupta**[2*]
[1] IBM Research AI, [2] Neuri Pte Ltd
{daiki, subhajit, ryuki}@jp.ibm.com, sakya@neuri.ai

Abstract

Reinforcement learning methods require careful design involving a reward function to obtain the desired action policy for a given task. In the absence of hand-crafted reward functions, prior work on the topic has proposed several methods for reward estimation by using expert state trajectories and action pairs. However, there are cases where complete or good action information cannot be obtained from expert demonstrations. We propose a novel reinforcement learning method in which the agent learns an internal model of observation on the basis of expert-demonstrated state trajectories to estimate rewards without having to completely learn the dynamics of the external environment from state-action pairs. The internal model is obtained in the form of a predictive model for the given expert state distribution. During reinforcement learning, the agent predicts the reward as a function of the difference between the actual state and the state predicted by the internal model. We conducted multiple experiments in environments of varying complexity, including the Super Mario Bros and Flappy Bird games. We show our method successfully trains good policies directly from expert game-play videos.

Introduction

Reinforcement learning (RL) (Sutton and Barto 1998) enables an agent to learn the desired behavior required to accomplish a given objective, such that the expected return or reward for the agent is maximized over time. Typically, a scalar reward signal is used to guide the agent's behavior so that the agent learns a control policy that maximizes the cumulative scalar reward over trajectories. This type of learning is referred to as *model-free* RL if the agent does not have an apriori model or knowledge of the dynamics of the environment it is acting in. Some notable breakthroughs among the many recent research efforts that incorporate deep models include the deep Q-network (DQN) (Mnih et al. 2015), which approximated a Q-value function used as a deep neural network and trained agents to play Atari games with discrete control, the deep deterministic policy gradient (DDPG) (Lillicrap et al. 2016), which successfully applied deep RL for continuous control agents, and the trust region policy optimization (TRPO) (Schulman et al. 2015), which formulated a method for optimizing control policies with guaranteed monotonic improvement.

In most RL methods, it is critical to choose a well-designed reward function to successfully ensure that the agent learns a good action policy for performing the task. Moreover, there are cases in which the reward function is very sparse or may not be directly available. Humans can often imitate the behavior of their instructors and estimate which actions or environmental states are good for the eventual accomplishment of a task without being provided with a continual reward. For example, young adults initially learn how to write letters by imitating demonstrations provided by their teachers or other adults (experts). Further skills get developed on the basis of exploration around this initial grounding provided by the demonstrations. Taking inspiration from such scenarios, various methods have been proposed, which are collectively known as imitation learning (Ho and Ermon 2016; Duan et al. 2017) or learning from demonstration (Schaal 1997). Inverse reinforcement learning (Ng and Russell 2000; Abbeel and Ng 2004; Wulfmeier, Ondruska, and Posner 2015), behavior cloning (Pomerleau 1991), and curiosity-based exploration (Pathak et al. 2017) are also examples of research in this field. Typically, in all these formulations, expert demonstrations are provided as input.

The majority of such prior work assumes that the demonstrations contain both states and actions $\{(\boldsymbol{s}_0^i, \boldsymbol{a}_0^i), ..., (\boldsymbol{s}_t^i, \boldsymbol{a}_t^i)\}$ and that these can be used to solve the problem of having only a sparse reward or a complete lack thereof. However, there are many cases in real-world environments in which such detailed action information is not readily available. For example, a typical schoolteacher does not tell students the exact amount of force to apply to each of their fingers while they are learning how to write.

As such, in this work, as our primary contribution, we propose a reinforcement learning method in which the agent learns an internal predictive model that is trained on the external environment from state-only trajectories by expert demonstrations. This model is not trained on both the state and action pairs. Hence, during each RL step, it estimates an expected reward value on the basis of the similarity between the actual and predicted state values by the internal model. Therefore, the agent must learn to reward known good states and penalize unknown deviations. We formulate this internal model as a temporal-sequence prediction model that predicts the next state value given the current and past state values

[*]This work originated while working at IBM Research

at every time step. This paper presents experimental results on multiple environments with varying input and output settings for the internal model. In particular, we show that it is possible to learn good policies using an internal model trained by observing only game-playing videos, akin to the way we as humans learn by observing others. Furthermore, we compare the performance of our proposed method regarding the baselines of *hand-crafted* rewards, prior research efforts, and other *baseline* methods for the different environments.

Related Work

In RL, an agent learns a policy $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ that produces good actions from the observation at the time. DQN (Mnih et al. 2015) showed that a Q-value function $q(s_t, a_t)$ can be successfully approximated with a deep neural network. DAQN (Kimura 2018) showed the pre-training by a generative model reduces the number of training iterations. Similarly, actor and critic networks in DDPG can enable continuous control, e.g., in robotic manipulation, by minimizing the distance between the robot end-effector and the target position. Since the success with DDPG, other methods such as TRPO (Schulman et al. 2015) and proximal policy optimization (PPO) (Schulman et al. 2017) have been proposed as further improvements for model-free RL regarding continuous control.

Although RL enables an agent to learn an optimal policy in the absence of supervised training data, in a standard case, it involves the difficult task of *hand-crafting* good reward functions for each environment (Abbeel and Ng 2004). Several approaches have been proposed to work around or tackle this problem. One approach that does not require *hand-crafted* rewards is behavior cloning based on supervised learning instead of RL (Pomerleau 1991). It learns the conditional distribution of actions from given states in a supervised manner. Although it has the advantage of fast convergence (Duan et al. 2017) (as behavior cloning learns a single action from states during each step), it typically results in the compounding of errors in future states.

An alternate approach, inverse reinforcement learning (IRL), was proposed (Ng and Russell 2000). In this work, the authors tried to recover the reward function as the best description of the given expert demonstrations from humans or expert agents using linear programming methods. This was based on the assumption that expert demonstrations are solutions to a Markov decision process (MDP) defined by a hidden reward function (Ng and Russell 2000). It demonstrated successful estimation of the reward function regarding relatively simple environments, such as a grid world and the mountain car problem. Extending (Ng and Russell 2000), entropy-based methods that compute a suitable reward function by maximizing the entropy of the expert demonstrations have been proposed (Ziebart et al. 2008). In another paper (Abbeel and Ng 2004), a method was proposed for recovering the cost function on the basis of expected feature matching between observed policies and agent behavior. Furthermore, the research showed that it is necessary for the agent to imitate the behavior of the expert. Demonstrations have also been used for initializing the value function (Wiewiora 2003).

Recently, there have been some studies that extended such a framework by using deep networks as non-linear function approximators for both the policies and the reward functions (Wulfmeier, Ondruska, and Posner 2015). In another relevant paper (Ho and Ermon 2016), the imitation learning problem was formulated as a two-player competitive game in which a discriminator network tries to distinguish between expert trajectories and agent-generated trajectories. The discriminator is used as a surrogate cost function that guides the agent's behavior to imitate the expert's behavior by updating policy parameters on the basis of TRPO (Schulman et al. 2015). Recent related work also includes model-based imitation learning (Baram et al. 2017) and robust imitation learning (Wang et al. 2017) using generative adversarial networks. It can be argued that our method is similar to the reward shaping method proposed by (Brys et al. 2015) because both methods calculate the similarity of demonstrations as a reward shaping function. However, while their paper dealt only with discrete action tasks, we show a similar approach can be applied to continuous action tasks [1]. Moreover, all the above-mentioned methods rely on both state and action information provided by expert demonstrations.

Another recent line of work aimed at learning useful policies for agents even in the absence of expert demonstrations. In this regard, they trained an RL agent with a combination of intrinsic curiosity-based reward and hand-engineered reward that had a continuous or very sparse scalar signal (Pathak et al. 2017). The curiosity-based reward was designed to have a high value when the agent encountered unseen states and a low value when it was in a state similar to the previously explored states. The paper reported good policies in games such as Super Mario Bros. and Doom, without any expert demonstrations. Here, we also compared our proposed method with the curiosity-based approach and demonstrated better-learned behavior. However, as a limitation, our method assumed that state demonstrations were available as expert data.

Also, there is a work that estimated the reward from linear function (Suay et al. 2016). However, they evaluated by a simple task; specifically, they used 27 discrete state-variables for Mario. In contrast, our method is using the non-linear model.

At the same time as this work, a recent paper (Torabi, Warnell, and Stone 2018) proposed learning policies using a behavior cloning method based on observations only. Unlike that work, here we put primary focus on reward shaping based on an internal model from observation data.

Proposed Method

Problem Statement

We considered a MDP consisting of states $\mathcal{S}$ and actions $\mathcal{A}$, where the reward signal $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ was unknown. An agent acted in accordance with this MDP following the pol-

[1]Please note they used a different Mario game from the one used in this paper.

icy, $\pi(\boldsymbol{a_t}|\boldsymbol{s_t})$. Here, we assumed to have knowledge of a finite set of expert state trajectories, $\tau = \{\boldsymbol{S^0}, ..., \boldsymbol{S^n}\}$, where $S^i = \{\boldsymbol{s^i_0}, ..., \boldsymbol{s^i_m}\}$. These trajectories represented joint angles, raw images, or other environmental states.

Since the reward signal was unknown, our primary goal was to find a reward signal that enabled the agent to learn a policy, π, that could maximize the likelihood of these sets of expert trajectories, τ. In this paper, we assumed that the reward signal could be inferred entirely on the basis of the information of the current and following states, $r : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$. More formally, we wanted to find a reward function that maximized the following objective:

$$r^* = \arg\max_r \mathbb{E}_{p(\boldsymbol{s_{t+1}}|\boldsymbol{s_t})} r(\boldsymbol{s_{t+1}}|\boldsymbol{s_t}), \qquad (1)$$

where $r(\boldsymbol{s_{t+1}}|\boldsymbol{s_t})$ is the reward function of the next state based on the current state and $p(\boldsymbol{s_{t+1}}|\boldsymbol{s_t})$ is the transition probability. We hypothesized that maximizing the likelihood of the next step prediction in Eq. 1 would result in increasing future rewards. This is because the likelihood was based on the similarity of current state values with the demonstrations obtained using the expert agent, which inherently chooses actions that would maximize their expected future reward. As such, we assumed the agent maximized the reward when it took the action that changed to a similar step value with given states from the expert.

Training the Internal Model

Let $\tau = \{s^i_t\}_{i=1:M, t=1:N}$ be the expert states obtained by the expert agent, where M is the number of demonstration episodes and N is the number of steps within each episode. We trained the internal model to predict reward signals on the basis of the expert state trajectories, τ, which in turn were used to guide a reinforcement learning algorithm and learn a suitable policy.

A straightforward idea (*baseline*) for an internal model is to use a generative model of the state value, $\boldsymbol{s^i_t}$, to understand the τ. The model trains a distribution of the state values, from which a predicted reward can be estimated on the basis of a similarity between the reconstructed state value and the actual experienced state value. This method limits exploration to the states that have been demonstrated by experts and enables learning a policy in a way that closely matches that of the expert. However, the temporal order of states is ignored or not readily accounted for, which is problematic because the temporal order of the next state in the sequence is important for estimating the state transition probability function.

Therefore, our proposed method uses a recurrent neural network (RNN)-based temporal-sequence model as an internal model that can be trained to predict the next state value given current and previous states on the basis of the expert trajectories. Such RNN temporal-sequence prediction models have been used successfully in the past as internal forward models in the context of grammar learning and robot behavior prediction (Bakker 2002; Dasgupta et al. 2015). Here, we trained a deep temporal sequence prediction model as the internal model by using the given state values, $\boldsymbol{s^i_t}$, and the next state values, $\boldsymbol{s^i_{t+1}}$, from the expert demonstration trajectories, τ. The model was trained to maximize the likelihood of the next state, such that the objective function for the model was

$$\boldsymbol{\theta^*} = \arg\min_{\boldsymbol{\theta}} \Big[-\sum_{i=1}^{M}\sum_{t=1}^{N} \log p(\boldsymbol{s^i_{t+1}}|\boldsymbol{s^i_t}; \boldsymbol{\theta}) \Big], \qquad (2)$$

where $\boldsymbol{\theta^*}$ represents the optimal parameters of the internal model. We also assumed the probability of the next state given the previous state value, $p(\boldsymbol{s^i_{t+1}}|\boldsymbol{s^i_t}; \boldsymbol{\theta})$, to be a Gaussian distribution. As such, the objective function could be seen as minimizing the mean square error, $\|\boldsymbol{s^i_{t+1}} - \boldsymbol{\theta}(\boldsymbol{s^i_t})\|_2$, between the actual next state, $\boldsymbol{s^i_{t+1}}$, and the predicted next state, $\boldsymbol{\theta}(\boldsymbol{s^i_t})$.

Reinforcement Learning

During the reinforcement learning, the method predicts a reward value with the trained internal model. The value is estimated as a function of the similarity between an actual next state value, $\boldsymbol{s_{t+1}}$, and the predicted next state value, $\boldsymbol{\theta}(\boldsymbol{s_t})$, given the current state value, $\boldsymbol{s_t}$. Thus the reward function is formulated as

$$r_t = -\psi\Big(\|\boldsymbol{s_{t+1}} - \boldsymbol{\theta}(\boldsymbol{s_t})\|\Big), \qquad (3)$$

where ψ is a function that reshapes the reward structure. In this paper, we experimented with a normal linear function, a hyperbolic tangent function, and a Gaussian function as the ψ function. In this formulation, if the current state was similar to the predicted state value, the estimated reward value was high. However, if the current state was not similar to the predicted state, the reward value was low. Moreover, as the reward value was estimated at each time step, this approach could predict dense rewards even regarding problems in which the original *hand-crafted* reward had a sparse structure.

Algorithm 1 explains the flow of the method. The RL procedure is shown as part of a generic RL pipeline and can be implemented with most on- or off-policy RL algorithms. In this paper, we used DDPG and DQN RL algorithms.

Experiment

We conducted experiments across a range of environments. We prepared four different tasks with varying complexity, namely, controlling a robot arm so that the end-effector reaches a target position, controlling a point agent to move to a target point while avoiding an obstacle, sending commands to a bird agent for the longest flight in the Flappy Bird video game, and controlling the Mario agent to maximize a total travelled distance in the Super Mario Bros. video game. Table 1 summarizes the key differences between the experiments.

Reacher

We considered a two degrees of freedom (2-DoF) robot arm in an x-y plane that has to learn to make the end-

[2]The parameter updates are carried out following the standard procedure of the specific reinforcement learning (on-policy or off-policy) algorithm.

Algorithm 1 Reinforcement Learning with Internal Model

```
1:  procedure TRAINING DEMONSTRATIONS
2:      Given trajectories $\tau$ from expert agent
3:      for $s_t^i, s_{t+1}^i \in \tau$ do
4:          $\theta^* \leftarrow \arg\min_{\theta} \Big[ -\sum_{i,t} \log p(s_{t+1}^i | s_t^i; \theta) \Big]$
5:      end for
6:  end procedure
7:  procedure REINFORCEMENT LEARNING
8:      for $t = 1, 2, 3, \ldots$ do
9:          Observe state $s_t$
10:         Execute action $a_t$, and observe state $s_{t+1}$
11:         $r_t \leftarrow -\psi\Big( \| s_{t+1} - \theta(s_t) \| \Big)$
12:         Update network [2] using $(s_t, a_t, r_t, s_{t+1})$
13:     end for
14: end procedure
```

Table 1: Comparison of different environments.

Environment	Input	Action	RL
Reacher	joint angle	continuous	DDPG
Mover w/ obstacle	pos., dist.[3]	continuous	DDPG
Flappy Bird	image, pos.	discrete	DQN
Super Mario Bros.	image	discrete	A3C

effector reach a target position. The first link of the robot was rigidly connected to the $(0, 0)$ point, and the second link connected to an edge of the first link. It had two joint values: $\boldsymbol{\theta} = (\theta_1, \theta_2)$, $\theta_1 \in (-\infty, +\infty)$ and $\theta_2 \in [-\pi, +\pi]$, and the lengths of the links were 0.1 and 0.11, respectively. The $\boldsymbol{p_2}$ is the end point of the first link, and the $\boldsymbol{p_{ee}}$ is the end-effector position of two links. The joint values and a target position were initialized by random values at the initial step of each episode. Specifically, the x and y of the target position, $\boldsymbol{p_{tgt}}$, were set from a random uniform distribution of $[-0.27, +0.27]$. The applied continuous action value, $\boldsymbol{a_t}$, was used to control the joint angles, such that $\dot{\boldsymbol{\theta}} = 0.05\ \boldsymbol{a_t}$. Each action value was clipped within the range of $[-1, 1]$. The state vector, $\boldsymbol{s_t}$, consisted of the following variables: an absolute end position of the first link ($\boldsymbol{p_2}$), a joint value between the first link and the second link (θ_2), velocities of the joints ($\dot{\theta}_1, \dot{\theta}_2$), and an absolute target position ($\boldsymbol{p_{tgt}}$). We used the roboschool environment with built-in physical dynamics (Brockman et al. 2016; OpenAI 2017) for this experiment, as shown in Figure 1. The robot links are in blue, the green point is the end-effector, and the red point is the target location.

We used the DDPG algorithm (Lillicrap et al. 2016) to train the RL agent. The actor and critic-network had 400, 300 fully connected (FC) neuron layers, respectively. The output from the final layer of the actor was passed through a $\tanh$ activation function while others passed through the ReLU (Nair and Hinton 2010) activation function. The exploration policy was an Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein 1930), the size of replay memory was 1 million steps, and we used the Adam optimizer (Kingma and Ba 2014) for the stochastic gradient updates. The number of steps for each episode was set to 400. All implementations were done using the Keras-rl (Plappert 2016) and Keras (Chollet 2015) libraries. Here, we compared the following reward functions:

Hand-crafted dense reward:

$$r_t = -\|\boldsymbol{p_{ee}} - \boldsymbol{p_{tgt}}\|_2 + r_t^{env} \quad (4)$$

Hand-crafted sparse reward:

$$r_t = -100 \tanh(\|\boldsymbol{p_{ee}} - \boldsymbol{p_{tgt}}\|_2) + r_t^{env} \quad (5)$$

Predictive model (PM, with state-action pair):

$$r_t = -10 \tanh(\|\boldsymbol{s_{t+1}} - \boldsymbol{\theta_{+a}}(\boldsymbol{s_t}, \boldsymbol{a_t})\|_2) + r_t^{env} \quad (6)$$

Generative model (GM, *baseline*):

$$r_t = -\tanh(\|\boldsymbol{s_{t+1}} - \boldsymbol{\theta_g}(\boldsymbol{s_{t+1}})\|_2) + r_t^{env} \quad (7)$$

Proposed method:

$$r_t = -10 \tanh(\|\boldsymbol{s_{t+1}} - \boldsymbol{\theta}(\boldsymbol{s_t})\|_2) + r_t^{env} \quad (8)$$

where r_t^{env} is an environment-specific reward, which is the cost for current action, $-\|\boldsymbol{a_t}\|_2$. This regularization was required to find the shortest path to reach the target. The expert demonstrations, τ, had 2000 episode trajectories by running a trained agent. The model, $\boldsymbol{\theta_{+a}}$, used both state-action pairs to estimate the reward function, $\|\boldsymbol{s_{t+1}^i} - \boldsymbol{\theta_{+a}}(\boldsymbol{s_t^i}, \boldsymbol{a_t^i})\|_2$, where $\boldsymbol{s_t^i}$ and $\boldsymbol{a_t^i}$ were obtained from demonstrations. Our proposed internal model did not require such action information $\boldsymbol{a_t^i}$. The proposed method was constructed using long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) as with the temporal sequence model. The model had two 128-unit LSTM layers with $\tanh$ activation and a 40-unit FC layer with ReLU activation. Furthermore, we also compared it with a standard behavior cloning (BC) (Pomerleau 1991) procedure, which used the actor-network directly trained with state-action pairs from expert demonstrations.

Figure 2 shows the performance of the agents. In all cases, using internal-model-based rewards gave better results than

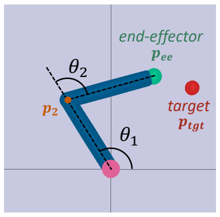

Figure 1: Reacher environment. Objective of agent is to make end-effector *(green)* reach target *(red)*.

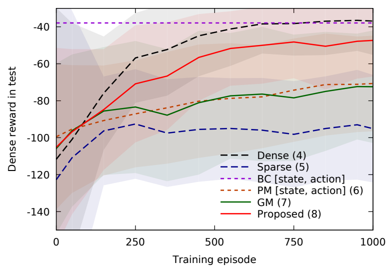

Figure 2: Performance of RL for reacher. Number in brackets corresponds to equation number.

[3]"pos." implies position, and "dist." implies distance.

having sparse rewards. Moreover, the model-based learning curves started from a better initial point compared to the dense reward curve. As observed, our proposed method achieved the best results when compared with all the baseline methods and also nearly achieved the results obtained in the dense reward case. As expected, the GM failed to work well in this complex environment. The PM model with state-action information also performed poorly. However, in comparison, the BC method worked relatively well. This is not surprising and clearly indicates that it is better to use behavior cloning than reward prediction when both state and action information are available from expert demonstrations.

Mover with Obstacle

In this task, we developed a new environment that has position control and an obstacle. The task was to move toward a target position without colliding with the obstacle. Figure 3 illustrates the environment setup. The initial position of the agent, the target position, and the obstacle's position were initialized randomly. The state vector, $\boldsymbol{s_t}$, contained the following variables: the agent's absolute position ($\boldsymbol{p_t}$), the current velocity of the agent ($\boldsymbol{\dot{p}_t}$), the target position ($\boldsymbol{p_{tgt}}$), the obstacle's position ($\boldsymbol{p_{obs}}$), and the relative target and obstacle location regarding the agent ($\boldsymbol{p_t} - \boldsymbol{p_{tgt}}, \boldsymbol{p_t} - \boldsymbol{p_{obs}}$). The RL algorithm used was DDPG (Lillicrap et al. 2016); the actor and critic networks had 64 and 64-unit FC layers, and each layer had a ReLU activation function. The exploration policy was the Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein 1930), the size of the replay memory was $500,000$ and the optimizer was Adam. The number of steps for each episode was set to 500.

Here, we tried predicting a part of the state that is related to a given action, thus taking the relevance into account. In former work (Pathak et al. 2017), the authors predicted the function of the next state, $\phi(\boldsymbol{s_{t+1}})$, rather than predicting the actual value, $\boldsymbol{s_{t+1}}$. In this experiment, we chose the agent position, ($\boldsymbol{p_t}$), as the selected state value. Furthermore, we changed the non-linear function, ψ, to a Gaussian function. This allowed us to compare the robustness of our proposed method when using different non-linear functions. Here, we used the following reward functions:

Hand-crafted dense reward:

$$r_t = -\|\boldsymbol{p_t} - \boldsymbol{p_{tgt}}\|_2 + \|\boldsymbol{p_t} - \boldsymbol{p_{obs}}\|_2 \tag{9}$$

Proposed method (predict next state values):

$$r_t = \exp(-\|\boldsymbol{s_{t+1}} - \boldsymbol{\theta}(\boldsymbol{s_t})\|_2 / 2\sigma_1^2) \tag{10}$$

Proposed method (predict only next agent position):

$$r_t = \exp(-\|\boldsymbol{s'_{t+1}} - \boldsymbol{\theta'}(\boldsymbol{s_t})\|_2 / 2\sigma_2^2), \tag{11}$$

where $\boldsymbol{s'_t}$ is the agent's position, $\boldsymbol{\theta'}$ is an internal network that predicts a selected state $\boldsymbol{\hat{s}'_t}$, σ_1 is 0.005, and σ_2 is 0.002. The dense reward was composed of both the target distance cost and an obstacle distance bonus. The expert trajectories, τ, contained 800 *human-guided* demonstration data with only state values; therefore, behavior cloning could not be directly applied. The internal prediction model once again used an LSTM network that consisted of two 256-unit LSTM layers with ReLU activations.

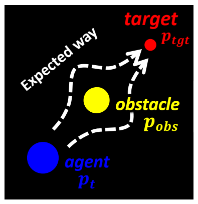

Figure 3: Mover with obstacle. Objective of agent *(blue)* is to move to target *(red)* while avoiding obstacle *(yellow)*.

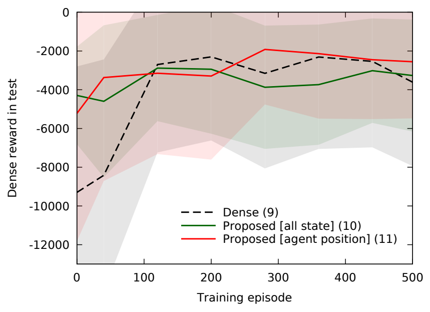

Figure 4: Performance for mover with obstacle. We tested the proposed method under two different conditions.

Figure 4 shows the performance obtained with the different reward settings. As observed, the proposed internal model learned to reach the target faster than the dense reward. Using the agent's position prediction internal model achieved the best performance.

Flappy Bird

In this experiment, we used a re-implementation (Lau 2017) of the "Flappy Bird" game. The objective of this game is to make the agent pass through as many pipes as possible without collision. The control is a single discrete command of whether to flap the bird's wings or not. The RL state value had four consecutive gray frames ($4 \times 80 \times 80$ pixels). A well-trained agent can play for an arbitrary number of steps; however, we set the limit to 1,000 steps for each episode. Each position of the pipe is random. In this case, we used the DQN (Mnih et al. 2015) RL algorithm in which the network had three convolutional and two FC layers. Each layer had ReLU activation, and it used the Adam optimizer and mean-squared loss. The size of replay memory was 2 million steps, the batch size was 256, and all other parameters were fixed following the original implementation (Lau 2017). The update frequency of the deep network was 100 steps. Here, we compared the following rewards:

Hand-crafted reward (the point for game):

$$r_t = \begin{cases} +0.1 & \text{if } \textit{alive} \\ +1 & \text{if } \textit{passes through a pipe} \\ -1 & \text{if } \textit{collides with a pipe} \end{cases} \tag{12}$$

Proposed method (predict next bird position):

$$r_t = \exp(-\|\boldsymbol{s'_{t+1}} - \boldsymbol{\theta'}(\boldsymbol{s_t})\|_2 / 2\sigma^2) \quad , \tag{13}$$

where $\boldsymbol{s'_t}$ is the absolute position of the bird that can be given from the simulator, and σ is 0.02. The absolute position was not in the state value; however, it can be estimated by simple image processing. The internal model, $\boldsymbol{\theta'}$, was constructed using an LSTM network to predict the bird's next position given the image input. The set of expert trajectories, τ, had only 10 episodes obtained from a trained agent available from the Github repository (Lau 2017). In this case, we also

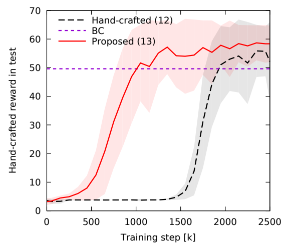

Figure 5: Performance for Flappy Bird (k is 10^3). Proposed method trains 10 episodes.

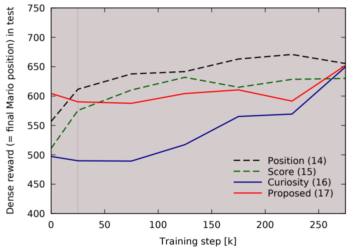

Figure 6: Performance for Super Mario Bros. Proposed method trains only 15 videos without any meta data.

compared the learned agent behavior with that obtained using a behavior cloning method.

Figure 5 clearly demonstrates that our proposed method converges faster than *hand-crafted* rewards. This can be ascribed to the fact that the hand-crafted reward only took into account the distance traveled, whereas our internal-model-estimated reward provides information about which absolute transitions are good. The hand-crafted reward of this game was a big positive value when it passed the pipe, otherwise it was a small positive value when the bird was alive. This means a big positive value will be delayed even if the bird chose a good action. Even though it is given each step, the hand-crafted reward does not contain the detailed reward value for each transition. In contrast, our method could estimate the detailed reward by using the similarity of state information for each transition. Furthermore, our method converges significantly better with fewer demonstrations than the baseline BC method, since the reason is the number of demonstrations was small.

Super Mario Bros.

In the final task, we considered a more difficult setting so that we could obtain only raw state information to clarify the benefits of the proposed method. Here, we applied our internal-model-based reward estimator to Nintendo's "Super Mario Bros." game and used a classic Nintendo video game emulator (Paquette 2017) for the environment. In this experiment, we compared our method with a curiosity-based method (Pathak et al. 2017) using their implementation (Pathak 2017). However, we slightly modified the game implementation to always initialize Mario at the starting position rather than at a previously saved checkpoint. The game has a discrete control where an agent (Mario) can make 14 types of action; however, a single action was repeated for six consecutive frames. The state, $\boldsymbol{s_t}$, consisted of sequential input of four 42×42-pixel gray-frame images with skipping every six frames. We used the A3C (Mnih et al. 2016) on-policy RL algorithm to evaluate our model, and played stage "1-1" of the game. The main objective of the agent was to travel as far as possible. We compared the following rewards:

Difference of Mario's position (dense reward):

$$r_t = position_t - position_{t-1} \tag{14}$$

Difference of score (sparse reward):

$$r_t = score_t - score_{t-1} \tag{15}$$

Curiosity (Pathak et al. 2017):

$$r_t = \eta \| \phi(\boldsymbol{s_{t+1}}) - \boldsymbol{\theta_F}(\phi(\boldsymbol{s_t}), \boldsymbol{a_t}) \|_2 \tag{16}$$

Proposed method (predict next frame):

$$r_t = max(0, -\| \boldsymbol{s'_{t+1}} - \boldsymbol{\theta'}(\boldsymbol{s_t}) \| + \zeta) \tag{17}$$

where $position_t$ is Mario's current position value, $score_t$ is a score value, $\boldsymbol{s'_t}$ is the latest frame in $\boldsymbol{s_t}$, and ζ is 0.025. Position, score, and related meta-information could be directly obtained from the emulator. In our proposed method, we took 15 game playing videos, each showing a single episode, from five different expert players and provided the demonstration trajectories, τ. In total, τ consisted of 25 thousand frames without any action or meta-information. We skipped 36 frames to generate $\boldsymbol{s^i_t}$ because people cannot play as fast as an RL agent. We used a three-dimensional convolutional neural network (3D-CNN) (Ji et al. 2013) as the $\boldsymbol{\theta'}$ model. The internal model, $\boldsymbol{\theta'}$, predicted the next frame image given the continuous frames, $\boldsymbol{s_t}$. The 3D-CNN network consisted of four convolutional layers[4] and one final convolutional layer to reconstruct the image. Once again, the proposed method required only videos to train the internal model.

Here, we changed the ψ function to a linear function to evaluate a simple formulation of the proposed method. However, a naïve reward estimate, ($r_t = -\|\boldsymbol{s'_{t+1}} - \boldsymbol{\theta'}(\boldsymbol{s_t})\|_2 + 1$),[5] does not work for this stage of the game. The Mario with the naïve method ends up getting positive rewards even if the agent remains stationary at the initial position (since enemy agents do not appear if Mario does not move). Hence, we applied a threshold, ζ, value to prevent this trivial sub-optimal outcome. ζ was calculated on the basis of the reward value obtained by staying stationary at the initial position.

Figure 6 shows the performance with the different reward functions. The graph presents the mean learning curves across trials. As observed, the agent does not reach the goal every time, even with the *hand-crafted* dense rewards [6]. This behavior was also observed in the original paper for their reward case (Pathak et al. 2017). However, as observed in Figure 6, our proposed method learns relatively faster than the curiosity- and score-based reward methods. Moreover, it was faster to obtain a good policy with the proposed method than with cases using dense rewards.

Comparing with the flappy bird experiment, the position reward represents the goodness for each transition, which means it is a 'dense' reward; the hand-crafted reward in the flappy bird was the delayed reward. We found that the proposed method could generate the predicted dense reward,

[4]Two layers had $(2 \times 5 \times 5)$ kernels, and the next two layers had $(2 \times 3 \times 3)$ kernels. All had 32 filters and (2, 1, 1) stride in every two layers.

[5]The +1 reward was for the terminal condition.

[6]The average position was 650, even with very long training steps, e.g., 3 million.

which is better value than the sparse reward and has potential to become similar to the dense reward, without any reward information.

Regarding future work for the Mario experiment, we believe using deeper networks as function approximators and higher-resolution input images may improve the performance of the convergence further.

Conclusion

In this paper, we proposed a reinforcement learning method that uses an internal model based on expert-demonstrated state trajectories to predict rewards. This method does not require learning the dynamics of the external environment from state-action pairs. The internal model consisted of a temporal sequence predictive RNN for the given expert state distribution. During RL, the agent calculated the similarity between actual and predicted states, and this value was used to predict the reward. We compared our proposed methods with *hand-crafted* rewards and previous methods in four different environments. Overall, we found that using internal model agents enables the learning of good policies, learning curves that have better initialization, and learning that converges faster than *hand-crafted* reward and sparse reward in most cases. It was also shown that the method could be applied to cases in which the demonstration was obtained directly from videos by person.

However, detailed trends were different for the different environments depending on the complexity of the task. As a current limitation of the method, we found that none of the rewards based on our proposed method were versatile enough to be applicable to every environment without any changes in the reward definition. There is thus room for further improvement, especially regarding modeling the global temporal characteristics of state trajectories. We intend to tackle the problem of generalizing across tasks in future work.

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*.

Bakker, B. 2002. Reinforcement learning with long short-term memory. In *Advances in neural information processing systems*, 1475–1482.

Baram, N.; Anschel, O.; Caspi, I.; and Mannor, S. 2017. End-to-end differentiable adversarial imitation learning. In *International Conference on Machine Learning*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *CoRR* abs/1606.01540.

Brys, T.; Harutyunyan, A.; Suay, H. B.; Chernova, S.; Taylor, M. E.; and Nowé, A. 2015. Reinforcement learning from demonstration through shaping. In *International Conference on Artificial Intelligence*.

Chollet, F. 2015. keras. `https://github.com/fchollet/keras`.

Dasgupta, S.; Goldschmidt, D.; Wörgötter, F.; and Manoonpong, P. 2015. Distributed recurrent neural forward models with synaptic adaptation and cpg-based control for complex behaviors of walking robots. *Frontiers in Neurorobotics*.

Duan, Y.; Andrychowicz, M.; Stadie, B.; Ho, J.; Schneider, J.; Sutskever, I.; Abbeel, P.; and Zaremba, W. 2017. One-shot imitation learning. *arXiv:1703.07326*.

Ho, J., and Ermon, S. 2016. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Comput.* 9(8):1735–1780.

Ji, S.; Xu, W.; Yang, M.; and Yu, K. 2013. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence* 35(1):221–231.

Kimura, D. 2018. Daqn: Deep auto-encoder and q-network. *arXiv preprint arXiv:1806.00630*.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

Lau, B. 2017. Keras-flappybird. `https://github.com/yanpanlau/Keras-FlappyBird`.

Lillicrap, T.; Hunt, J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In *The International Conference on Learning Representations (ICLR)*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*.

Nair, V., and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, 807–814. Omnipress.

Ng, A. Y., and Russell, S. J. 2000. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 663–670.

OpenAI. 2017. Roboschool. `https://github.com/openai/roboschool`.

Paquette, P. 2017. gym-super-mario. `https://github.com/ppaquette/gym-super-mario`.

Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*.

Pathak, D. 2017. noreward-rl. `https://github.com/pathak22/noreward-rl`.

Plappert, M. 2016. keras-rl. `https://github.com/matthiasplappert/keras-rl`.

Pomerleau, D. A. 1991. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation* 3(1):88–97.

Schaal, S. 1997. Learning from demonstration. In *Advances in Neural Information Processing Systems 9*. MIT Press. 1040–1046.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *International Conference on Machine Learning*, 1889–1897.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv:1707.06347*.

Suay, H. B.; Brys, T.; Taylor, M. E.; and Chernova, S. 2016. Learning from demonstration for shaping through inverse reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents &#38; Multiagent Systems*, AAMAS '16, 429–437. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st edition.

Torabi, F.; Warnell, G.; and Stone, P. 2018. Behavioral cloning from observation. In *International Joint Conference on Artificial Intelligence*.

Uhlenbeck, G. E., and Ornstein, L. S. 1930. On the theory of the brownian motion. *Phys. Rev.* 36:823–841.

Wang, Z.; Merel, J.; Reed, S. E.; Wayne, G.; de Freitas, N.; and Heess, N. 2017. Robust imitation of diverse behaviors. abs/1707.02747.

Wiewiora, E. 2003. Potential-based shaping and q-value initialization are equivalent. *J. Artif. Int. Res.* 19(1):205–208.

Wulfmeier, M.; Ondruska, P.; and Posner, I. 2015. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*.

Ziebart, B. D.; Maas, A.; Bagnell, J. A. D.; and Dey, A. 2008. Maximum entropy inverse reinforcement learning. In *AAAI*.