Machine Learning (2020) 109:1699–1725
https://doi.org/10.1007/s10994-019-05849-4

Active deep Q-learning with demonstration

Si-An Chen[1] · Voot Tangkaratt[2] · Hsuan-Tien Lin[1] · Masashi Sugiyama[2,3]

Received: 26 November 2018 / Revised: 26 June 2019 / Accepted: 26 September 2019 /
Published online: 8 November 2019

Abstract
Reinforcement learning (RL) is a machine learning technique aiming to learn how to take actions in an environment to maximize some kind of reward. Recent research has shown that although the learning efficiency of RL can be improved with expert demonstration, it usually takes considerable efforts to obtain enough demonstration. The efforts prevent training decent RL agents with expert demonstration in practice. In this work, we propose Active Reinforcement Learning with Demonstration, a new framework to streamline RL in terms of demonstration efforts by allowing the RL agent to query for demonstration actively during training. Under the framework, we propose Active deep Q-Network, a novel query strategy based on a classical RL algorithm called deep Q-network (DQN). The proposed algorithm dynamically estimates the uncertainty of recent states and utilizes the queried demonstration data by optimizing a supervised loss in addition to the usual DQN loss. We propose two methods of estimating the uncertainty based on two state-of-the-art DQN models, namely the divergence of bootstrapped DQN and the variance of noisy DQN. The empirical results validate that both methods not only learn faster than other passive expert demonstration methods with the same amount of demonstration and but also reach super-expert level of performance across four different tasks.

Keywords Active learning · Reinforcement learning · Learning from demonstration

---

Editors: Po-Ling Loh, Evimaria Terzi, Antti Ukkonen, Karsten Borgwardt.

✉ Si-An Chen
r05922089@csie.ntu.edu.tw

Voot Tangkaratt
voot.tangkaratt@riken.jp

Hsuan-Tien Lin
htlin@csie.ntu.edu.tw

Masashi Sugiyama
sugi@k.u-tokyo.ac.jp

1 Present Address: CSIE Department, National Taiwan University, Taipei, Taiwan

2 RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

3 Graduate School of Frontier Sciences, The University of Tokyo, Chiba, Japan

1 Introduction

Sequential decision making is a common and important problem in the real world. For instance, to achieve its goal, a robot should produce a sequence of decisions or movements according to its observations over time. A recommender system should decide when and which item or advertisement to display to a customer in a sequential manner. For sequential decision making, reinforcement learning (Sutton and Barto 1998) (RL) has been recognized as an effective framework which learns from interaction with the environment. Thanks to advances in deep learning and computational hardware, deep RL has achieved a number of successes in various fields such as end-to-end policy search for motor control (Watter et al. 2015), deep Q-Networks for playing Atari games (Mnih et al. 2015), and combining RL and tree search to defeat the top human Go expert (Silver et al. 2016). These successes show the power of RL to solve various kinds of sequential decision making and control problems.

In contrast with these successes, deep RL algorithms are notorious for their substantial demands on simulation during training. Typically, these algorithms start from scratch and require millions of data samples to learn a locally optimal policy, which is not a problem if unlimited simulation is available but is infeasible for many real-world applications such as robotic systems. To address this problem, several methods have been proposed to improve learning efficiency by leveraging prior knowledge from human experts. Imitation learning (Schaal 1996), also known as learning from demonstration (LfD), is an attempt to learn the policy of an expert by observing the expert's demonstrations. However, the performance of imitation learning is limited by the expert, since the agent only learns from the expert without regard to rewards given by the environment. Another way is to improve RL by leveraging demonstrations given by the expert and rewards simultaneously. Recently, deep Q-learning from Demonstration (DQfD; Hester et al. 2018) and Policy Optimization with Demonstrations (POfD; Kang et al. 2018) have shown state-of-the-art results on several Atari games by training the agent with an objective that combines the rewards and the expert demonstrations.

Although expert demonstrations improve RL, the efforts made by the expert are not negligible. For instance, it takes a human expert thousands of steps to finish a mere 5 episodes for most Atari games (Hester et al. 2018). The huge efforts make it hard to collect a large number of demonstrations for DQfD in practice. In this paper, we introduce the concept of active learning to make more efficient use of the expert's efforts. In supervised learning, the goal of active learning is to achieve better performance with less labeling effort by interactively querying for new labels from unlabeled data (Settles 2009). In RL, we can also actively ask the expert for a recommended action given the current observed state. Videos of bootstrapped DQN (Osband et al. 2016) have shown that the behavior of different well-performing policies agree at critical points but diverge at other less important states. This suggests that we could save much expert effort by querying only at critical states in contrast to previous methods in which the expert's demonstration is collected for several entire episodes. In other words, we can achieve further improvement in RL with the same number of demonstrations.

In this work, we consider reinforcement learning problems that allow *selecting* human demonstration *during* the learning process. We first propose a new framework called Active Reinforcement Learning with Demonstration (ARLD) for such learning problems. Then we propose Active DQN, which proactively asks for demonstration and leverages the demonstration data. The query criterion should decide when to query—i.e., identifying states where the agent can indeed learn and improve by obtaining the demonstration. We propose two query methods based on uncertainty of Q-value estimation, named divergence and variance, which are derived from two state-of-the-art DQN methods, bootstrapped DQN and noisy DQN, respectively. The uncertainty terms are then thresholded dynamically to form querying decisions.

The dynamic nature allows the two methods to adapt to recent states observed by the agent and can be applied in various kinds of environments without exhaustive parameter tuning. Experimental results show that our method with both uncertainty measurements is effective in four different tasks. In this paper we thus offer three main contributions:

1. We propose a new framework, Active Reinforcement Learning with Demonstration, which is the first work to reduce human effort in RL with demonstration to the best of our knowledge;
2. We propose a novel uncertainty-based query strategy which can be applied toward different tasks and less sensitive to additional parameters;
3. We verify the effectiveness of two DQN uncertainty estimations with promising experiment results.

2 Related work

Imitation learning (Schaal 1996) is a classic technique for learning from human demonstration. Typically, imitation learning uses a supervised approach to imitate an expert's behaviors. DAGGER (Ross et al. 2011), a popular imitation algorithm, requests an action from the expert at each step, and takes an action sampled from a mixed distribution of the agent and the expert. It then aggregates the observed states and demonstrated actions to train the agent iteratively. Deeply AggreVaTeD (Sun et al. 2017) is an extended version of DAGGER which works with deep neural networks and continuous action spaces. However, both require an expert to provide demonstration during the whole training phase. To reduce the demand for human effort, the agent learns actively in active imitation learning (Shon et al. 2007; Judah et al. 2014) by requesting fewer expert demonstrations. The supervised setting of imitation learning make it easier to apply techniques from traditional supervised active learning. However, although imitation learning can lead to no-regret performance in online learning settings, its performance is still limited by the expert given the use of *only* expert demonstration data for learning.

On the contrary, it is possible for Reinforcement Learning (RL) to achieve better performance than the human expert by learning to interact with the environment and maximize the cumulative rewards. In RL, there exist a variety of methods that leverage demonstration to obtain improved performance. For instance, some use expert advice or demonstration to shape rewards in the RL problem (Brys et al. 2015; Suay et al. 2016). Another approach is to ask for demonstration from a given state to another state to improve the exploration (Subramanian et al. 2016). In contrast, the HAT algorithm summarizes the demonstrated knowledge via a decision tree and bootstraps the task with the learned policy to transfer it to the target agent (Taylor et al. 2011). CHAT, an extension of HAT, measures the source policy's confidence to decide whether to take its advice (Wang and Taylor 2017). The main difference between CHAT and our work is that CHAT trains an offline model to learn a source policy from well-collected demonstration by supervised learning and estimate the confidence of this offline policy during RL. In addition, the demonstration is collected without any reward information. In contrast, we estimate the uncertainty of the RL agent directly and ask the expert interactively to collect online demonstration and its rewards.

Table 1 Comparison between different settings

	No expert	Offline/passive learning	Online/active learning
Imitation learning		DAGGER	Active imitation learning
		Deeply AggreVaTeD	
Reinforcement Learning	DQN	DQfD	**ARLD (our work)**
	DDPG	DDPGfD	
	A3C	POfD	
		HAT, CHAT	

Reinforcement Learning with Expert Demonstrations (RLED; Piot et al. 2014) concerns a scenario in which the expert also receives rewards from the environment. In this case, DQfD (Hester et al. 2018), DDPGfD (Vecerik et al. 2017), and POfD (Kang et al. 2018) have shown state-of-the-art results on a variety of tasks by combining the original RL loss with a supervised loss on the expert's demonstrations. Then, the agent simultaneously learns its original objective and the behavior of the expert. In comparison to similar work such as Human Experience Replay (Hosu and Rebedea 2016) and Replay Buffer Spiking (Lipton et al. 2016), RLED methods yield massive acceleration with a relatively small amount of demonstration data. Moreover, experiments show that these methods can also outperform the expert they learn from. In contrast to our work, these works collect demonstration data before training, and the expert must interact with environment by completing the whole episode several times, whereas the proposed method requires the expert to demonstrate only at critical states given the learning progress of the agent.

Most previous works focus on how to improve RL from "passive" demonstration data. To the best of our knowledge, this is the first work to introduce the concept of active learning to leverage demonstration data. The most similar work is active imitation learning (Shon et al. 2007; Judah et al. 2014), which casts imitation learning as a classification problem and conducts active learning on the classification problem. Table 1 compares different settings mentioned above. Note that it is non-trivial to simply combine RL with existing active learning methods for three reasons. First, most mature active learning methods are pool-based (Settles 2009), which cannot be easily adapted to RL due to its streaming nature. Secondly, mature active learning methods are mostly for classification, while estimating the Q-values within RL is closer to regression in nature. The regression-like characteristic makes it hard to connect to the definitions of "decision boundary" that is commonly needed in active learning methods for classification (Dagan and Engelson 1995). Thirdly, most streaming-based active learning methods require calculating the uncertainty through some form of the version space (Mitchell 1982), which cannot easily be done with the deep neural networks within state-of-the-art RL algorithms. The three reasons prevent typical active learning methods to be directly applied in our setting.

3 Background

3.1 Reinforcement learning and deep Q network

The standard reinforcement learning framework consists of an agent interacting with an environment which can be modeled as a Markov decision process (MDP). An MDP is defined by a tuple $M = \langle S, A, R, P, \gamma \rangle$, where S is the state space, A the action space, $R : S \times A \to \mathbb{R}$ the reward function, $P(s'|s, a)$ the transition probability function, and $\gamma \in [0, 1)$ the discount factor. At each step, the agent observes a state $s \in S$ and takes an action $a \in A$ according to a policy π. The policy π can be either deterministic, $\pi : S \to A$, or stochastic, $\pi : S \to P(A)$. On taking an action, the agent receives a reward $R(s, a)$ and reaches a new state s' according to $P(s'|s, a)$. The goal of the agent is to find the policy π which maximizes the discounted accumulative reward $\mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$, where τ denotes the trajectory obtained with π and P.

For problems with discrete actions, the most popular approach nowadays is the deep Q-network (DQN; Mnih et al. 2015). The key idea of DQN is to learn an approximation of the optimal value function Q^*, which conforms to the Bellman optimality equation

$$Q^*(s, a) = R(s, a) + \gamma \mathop{\mathbb{E}}_{s' \sim P(s'|s,a)} \left[\max_{a' \in A} Q^*(s', a')\right].$$

The optimal policy is then defined by Q^* as $\pi(s) = \text{argmax}_{a' \in A} Q^*(s, a')$. The value-function is approximated by a neural network $Q(s, a; \theta)$ with parameter θ where the parameter is learned by minimizing the temporal difference (TD) loss:

$$\begin{aligned} Q_{target} &= r + \gamma \max_{a' \in A} Q(s', a'; \theta^-) \\ L_{TD}(\theta) &= \mathop{\mathbb{E}}_{(s,a,r,s') \sim D} \left[(Q_{target} - Q(s, a; \theta))^2\right], \end{aligned}$$

where D is a distribution of transitions $(s, a, r = R(s, a), s' \sim P(s'|s, a))$ drawn from a replay buffer of previously observed transitions, and θ^- is the parameter of a separate target network which is copied from θ regularly to stabilize the target Q-values.

3.2 Double deep Q-learning and prioritized experience replay

Double deep Q-learning (van Hasselt et al. 2016) and prioritized experience replay (Schaul et al. 2016) are two common techniques to improve DQN. The double Q-learning update calculates the target value by replacing $\max_{a' \in A} Q(s', a'; \theta^-)$ with $Q(s', \text{argmax}_{a' \in A} Q(s', a'; \theta); \theta^-)$. The modification reduces the overestimation of target value created with the original update rule.

Prioritized experience replay modifies the uniform sampling of replay buffer to weighted sampling, where the probability of sampling each transition is proportional to its priority. The priority of a transition i is $p_i = |\delta_i| + \epsilon$, where δ_i represents the last TD error calculated with this transition and ϵ is a small positive constant that prevent the transitions not being sampled once their error is zero. To deal with the changes of distribution, updates to the network are weighted with importance sampling weights.

These two techniques are widely accepted as the standard for training RL agents. For example, deep Q-learning from Demonstration (DQfD; Hester et al. 2018), which will be discussed in the next section, applied these techniques. We will also apply the techniques in our proposed algorithm, which can be viewed as an "active" extension of DQfD.

3.3 Deep Q-learning from demonstration

Deep Q-learning from Demonstration (DQfD; Hester et al. 2018) is a state-of-the-art method to leverage demonstration data to accelerate the learning process of DQN. The agent is pre-trained on demonstration data to obtain better initial parameters before any interaction with the environment. It keeps demonstration data in a prioritized replay buffer (Schaul et al. 2016) permanently, that means the removal of transition data once the replay buffer is full only happens on the transitions collected by the agent itself. Moreover, a positive constant $\epsilon_d > \epsilon$ is used as the basic priority of demonstration data to increase the probability that they are sampled. DQfD applies a combination of four losses: the typical one-step double Q-learning loss (L_{TD}), N-step double Q-learning loss (L_N), supervised large margin classification loss (L_E), and L2 regularization loss. The overall loss is thus

$$L(\theta) = L_{TD}(\theta) + \lambda_1 L_N(\theta) + \lambda_2 L_E(\theta) + \lambda_3 ||\theta||_2^2.$$

The typical one-step temporal difference loss and N-step temporal difference loss are used to obtain the optimal Q-value conforming to the Bellman equation, where the N-step loss extends the Q_{target} to a more precise form, which combines N-step discounted cumulative reward and optimal Q-value estimation after N steps. The formula is as following:

$$\begin{aligned} L_N(\theta) &= \mathop{\mathbb{E}}_{(s,a,r,s')\sim D}\left[(Q_N - Q(s,a;\theta))^2\right], \\ Q_N &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{N-1} r_{t+N-1} + \max_a \gamma^N Q(s_{t+N}, a). \end{aligned}$$

The large margin classification loss (Piot et al. 2014) is defined as

$$L_E(\theta) = \max_{a \in A}\left[Q(s,a;\theta) + M\,\mathbb{1}\left[a \neq a_E\right]\right] - Q(s, a_E),$$

where a_E represents the action that the expert took in state s, M is a positive constant value which means margin and $\mathbb{1}[\cdot]$ is indicator function. L2 regularization loss is applied to parameters of the network to prevent overfitting on demonstration data. All losses are applied in both pre-training and reinforcement learning phases. At each step, DQfD samples a batch of transition data from the prioritized replay buffer to calculate the total loss. While the sampled data may contain both the demonstration data from the expert and the historical records from the agent, the large margin classification loss is only applied on the demonstration data from the expert.

3.4 Deep exploration via bootstrapped DQN

Exploration is an important issue in RL. E.g., ϵ-greedy is commonly used but it does not exploit any information. Bootstrapped DQN (Osband et al. 2016) is a modification of DQN to improve exploration during training. In practice, the network is built with $K \in \mathbb{N}$ outputs, each representing a Q-value function estimation $Q_k(s, a; \theta)$. Each output head is trained against its own target network $Q_k(s, a; \theta^-)$ and is updated with its own bootstrapped data from the replay buffer. The parameters of each head are initialized independently, while the gradient of each update is normalized with $1/K$. During training, a single head is sampled at the beginning of each epoch, and the agent takes the optimal policy corresponding to the sampled Q-value approximation function for the duration of the episode. This allows the agent to conduct a more consistent exploration as compared to other common dithering strategies such as ϵ-greedy. To keep track of the bootstrapped data for each head, we attach to each transition data in the replay buffer a boolean mask $m \in \{0, 1\}^K$ indicating which heads are privy to this data. The masks are drawn from an identical Bernoulli distribution independently ($m_i \sim Ber(p), \forall i \in 1 \ldots K$). However, their experiments show that the performance of bootstrapped DQN is not influenced by different choices of p, and that all outperform the original DQN. Hence in practice, to increase computational efficiency, we simply share all the data between each head ($p = 1$).

3.5 Noisy networks for exploration

NoisyNet (Fortunato et al. 2017) is an alternative approach to improve the efficiency of exploration in RL, where the parameters in the output layer of a network are perturbed by noise. The noisy parameter θ is a vector of parameters in $Q(s, a; \theta)$, represented by $\theta = \mu + \Sigma \odot \varepsilon$, where $\zeta = (\mu, \Sigma)$ is a set of vectors of learnable parameters, ε is a vector of zero-mean noise sampled from the standard normal distribution, and $\odot$ stands for element-wise multiplication. A noisy linear layer with p inputs and q outputs is then represented by

$$y = (\mu_w + \sigma_w \odot \varepsilon_w)x + \mu_b + \sigma_b \odot \varepsilon_b,$$

where $\mu_w + \sigma_w \odot \varepsilon_w$ and $\mu_b + \sigma_b \odot \varepsilon_b$ replace the weight matrix and bias vector of typical linear regression. The parameters $\mu_w \in \mathbb{R}^{q \times p}$, $\mu_b \in \mathbb{R}^q$, $\sigma_w\, in\, \mathbb{R}^{q \times p}$, $\sigma_b \in \mathbb{R}^q$ are learnable parameters of a layer of the noisy network and $\varepsilon_w \in \mathbb{R}^{q \times p}$, $\varepsilon_b \in \mathbb{R}^q$ are noise variables. The agent samples a new set of random variables ε after each update step to obtain a sample of θ and follows the optimal policy corresponding to the sampled Q-value function. The noise variables of the online network, target network, and online network in double DQN are sampled independently. The loss function for noisy double DQN is defined as:

$$\begin{aligned}
&\bar{L}(\zeta) \\
&\quad = \mathop{\mathbb{E}}_{\varepsilon, \varepsilon', \varepsilon''} \left[\mathop{\mathbb{E}}_{(s,a,r,s') \sim D} [r + \gamma Q(s', a^*, \varepsilon'; \zeta^-) - Q(s, a, \varepsilon; \zeta)]^2 \right] \\
&a^* = \mathop{\mathrm{argmax}}_{a \in A} Q(s', a, \varepsilon''; \zeta),
\end{aligned}$$

where ε, ε', ε'' are noise variables corresponding to the online network, target network, and the online network used to evaluate optimal action in double DQN.

4 Active deep Q-learning with demonstration

In this section, we first describe a new problem setting, then propose an uncertainty-based query strategy to address the problem, after which we introduce two ways to estimate the uncertainty of a deep Q-Network given an observed state with bootstrapped DQN and noisy DQN, separately.

4.1 Problem setup

We proposed a new framework named Active Reinforcement Learning with Demonstration (ARLD) to improve the demonstration efficiency, which is not considered in previous RLED works. In ARLD, we consider the standard RL framework introduced in 3.1. In addition, we assume there is an expert π^+ which performs well on the task we seek to learn and is not required to be optimal nor deterministic. In addition to the usual RL process, the agent also interacts with the expert by querying what action to take upon observing some state, say, at time step t. After querying, we assume that the expert "takes over" the decision of the actions for a few consecutive steps between steps t and $t + k$. The demonstrated actions

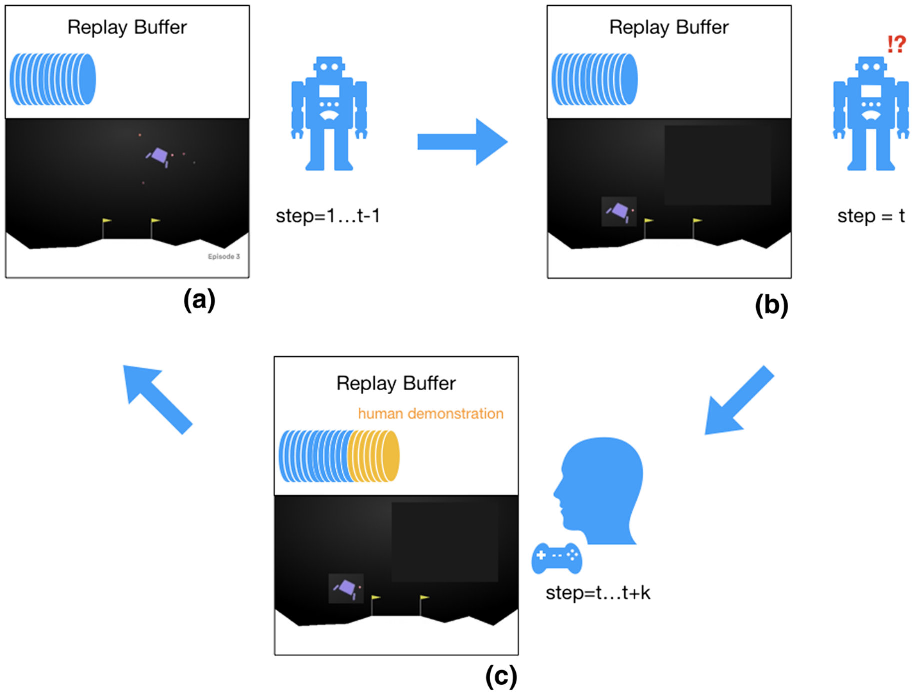


Fig. 1 The process of ARLD: **a** at step 1 to $t-1$, the agent performs usual RL by interacting with the environment; **b** at step t, the agent observes a state that triggers a query to the (human) expert for demonstration; **c** at steps t to $t+k$, the expert replaces the agent and makes consecutive demonstration actions for the agent. Then, after step $t+k$, the agent continues its usual RL from (**a**)

that the expert have taken can then be freely used by the agent for imitation, for instance, as a piece of data in the agent's replay buffer. After the demonstration period, the agent goes back to usual RL. The whole process of ARLD is illustrated in Fig. 1. The main challenge of ARLD is to decide when to query from the expert so that the agent can indeed benefit from the obtained demonstration. As with active learning, our goal is to improve RL by making as few queries as possible. More precisely, given a limited query budget, we seek to enable the agent to learn to solve the task in as few steps as possible. Below, we discuss uncertainty sampling (?), which one of the simplest and most commonly used query frameworks in active supervised learning.

4.2 Query strategy with adaptive uncertainty threshold

Uncertainty sampling is one of the simplest and most commonly used query frameworks in active learning (Settles 2009). In this framework, an active learner estimates the uncertainty of a pool of unlabeled instances and submits queries for those it is least certain how to label. It is challenging to apply active learning with deep neural networks, as good deep models typically require large amounts of data. Recent work has shown that uncertainty can be estimated by taking advantage of specialized models such as Bayesian neural networks (Gal et al. 2017). However, to improve RL by requesting an expert demonstration, we require an online query strategy that takes advantage of uncertainty.

In our setting, at each step, before the agent takes an action, it decides whether to query the expert. A naive way to solve this problem is to make the decision with a fixed threshold: the agent asks expert for demonstration once the uncertainty of an observed state exceeds the threshold; otherwise it takes the action which maximizes the estimated Q-value. However, it is difficult to find a proper threshold when the distribution of uncertainty keeps changing; the discrepancy between different tasks also makes this difficult. One proposal is to adjust the threshold with a fixed adjustment factor to work with the online Query by Committee (QBC; Krawczyk and Wozniak 2017), but it is still difficult to choose a good adjustment factor for all tasks, especially when the uncertainty measurement is unbounded and its magnitude unknown.

We propose an adaptive method which enables the agent to adjust its query policy during training time without any prior knowledge of the task. Each time the agent makes a decision, it compares the uncertainty of the current state with that estimated in recent steps. If the current state uncertainty is larger than a given proportion of recent steps, the agent queries the expert for demonstration; otherwise it determines its own action. In this algorithm, we decide whether to ask the expert given the parameters N_r and t_{query}, representing the amount of recent steps we consider and the proportion of recent steps for which the state uncertainty must be higher than the current state uncertainty. In practice, we use a double-ended queue to maintain the uncertainty of recent steps and a balanced binary search tree (BST) to keep these uncertainties in order so that we can make the decision in $O(\log_2 N_r)$ complexity for each step. The pseudocode of the algorithm is provided in Algorithm 1.

The performance of the algorithm depends on the choice of uncertainty estimation. Below, we propose two methods to estimate the uncertainty. One is based on bootstrapped DQN and the other one is based on noisy network.

Algorithm 1 Adaptive Query Strategy with Uncertainty

```
Input: uncertainty $U_t$, reference size $N_r$, proportion threshold $t_{query} \in [0, 1]$, recent uncertainty double-
    ended queue $D$, recent uncertainty BST $B$
Output: asking $\in$ [$TRUE$, $FALSE$], $D$, $B$
 1: $idx \leftarrow$ size of $D \times t_{query}$
 2: $U_{threshold} \leftarrow B[idx]$
 3: if $U_t > U_{threshold}$ then
 4:     asking $\leftarrow$ $TRUE$
 5: else
 6:     asking $\leftarrow$ $FALSE$
 7: end if
 8: if size of D $\geq R_r$ then
 9:     $U_{del} \leftarrow D.pop\_left()$
10:     remove $U_{del}$ from $B$
11: end if
12: add $U_t$ into $D$, $B$
```

4.3 Divergence of bootstrapped DQN

Bootstrapping is a commonly used technique in statistics to estimate a sampling distribution. In bootstrapped DQN (Osband et al. 2016), multiple value function heads $Q_k(s, a; \theta)$ are used to approximate a distribution over Q-values. Each head is initialized with randomness independently. Moreover, each data instance in the replay buffer will be assigned to several heads sampled by Bernoulli distribution. There are several ways to estimate uncertainty with these bootstrapped heads, including calculating the entropy of the voting distribution or averaging the variance of action values predicted by each head. In this work, we consider each head as a distribution and estimate the uncertainty using Jensen-Shannon divergence, which is a well-known method to measure the similarities between multiple distributions.

When considering the divergence measurement among bootstrapped agents, one may concern the entropy of voting distribution by agents. However while the agents in bootstrapped DQN tend to behave differently at less critical states because the Q-values of each action might be close to each other. In this situation, the JS divergence of Q-value distributions is a better measurement than the entropy of the voting distribution. For example, when considering an environment with two actions and two bootstrapped agents, if the two agents predict (0.5, 0.4) and (0.4, 0.5) respectively, we would consider the state is not as critical as another state where the two agents predict (1, 0) and (0, 1). However, with the voting method, both states are considered equally uncertain because at both states the two agents vote for different actions. JS divergence, by contrast, distinguishes the difference between them. Another advantage of JS divergence is that as it is a bounded function, its value is more meaningful and easier to use as a threshold in different environments.

To measure the JS divergence between the bootstrapped heads, we first normalize the Q-values and actions using softmax to obtain a policy distribution. For each head $Q_k(s, a; \theta)$,

$$\pi_k(a|s; \theta) = e^{Q_k(s,a;\theta)} / \sum_{a'} e^{Q_k(s,a';\theta)}.$$

Given this policy distribution, we estimate the uncertainty by calculating the Jensen-Shannon divergence of the policy distribution between each head, yielding

$$U_D = JS(\pi_1, \pi_2, ..., \pi_K) = H\left(\frac{1}{K}\sum_k \pi_k\right) - \frac{1}{K}\sum_k H(\pi_k),$$

where $H(\pi)$ is the Shannon entropy of distribution π and K the number of bootstrapped heads.

4.4 Predictive variance of noisy DQN

For our second estimate of uncertainty, we evaluate the predictive variance of a noisy network. Previous work has shown the effectiveness of estimating uncertainty by the predictive variance of a Bayesian convolution network in classification active learning (Gal et al. 2017). Other works have shown that injecting noise into the parameter space improves the exploration process in deep reinforcement learning (Fortunato et al. 2017; Plappert et al. 2017). Combining these two ideas, we use the noisy network as an exploration policy and estimate uncertainty using its predictive variance. We replace the fully connected layer in the output layer with a noisy fully connected layer. The corresponding noisy output layer can be seen as Bayesian linear regression:

$$\begin{aligned}
&Q(s, a; \theta) = w_a\phi(s) + b_a,\\
&w_a \sim N(\mu^{(w_a)}, \Sigma), \Sigma = \mathrm{diag}((\sigma^{(w_a)})^2)\\
&b_a \sim N(\mu^{(b_a)}, (\sigma^{(b_a)})^2),
\end{aligned}$$

where $\phi(s) \in \mathbb{R}^p$ is input to the last layer. $w_a \in \mathbb{R}^p, b_a \in \mathbb{R}$ represents the variables corresponding to action a, and $\mu^{(w_a)}$, $\sigma^{(w_a)}$, $\mu^{(b_a)}$, and $\sigma^{(b_a)}$ are the parameters actually learned by the model, representing the mean and noise level of w_a and b_a respectively.

Given the posterior distribution of the parameters, we derive the predictive variance as

$$\begin{aligned}\mathrm{Var}[Q(s,a)] &= \mathrm{Var}[w_a\phi(s) + b_a] \\ &= \mathrm{Var}[w_a\phi(s)] + \mathrm{Var}[b_a] \\ &= \phi(s)^T \Sigma\phi(s) + (\sigma^{(b_a)})^2.\end{aligned}$$

The variance of each action measures the lack of confidence with respect to this action. We take the variance of the action with the largest Q-value as our uncertainty:

$$U_V = \mathrm{Var}\left(Q\left(s, \underset{a}{\mathrm{argmax}}\, Q(s,a)\right)\right),$$

which translates to the lack of confidence for the action the agent would take at the step. By querying the states with low confidence, we avoid bad moves leading to task failure and explicitly teach the agent which is the proper action of the state.

5 Experiments

In this section, we describe the environments used for the evaluation as well as the experimental setup. To focus on the effectiveness of each query strategy, we show the experimental results of methods based on the bootstrapped and noisy network separately, after which we present results evaluating the effect of the query proportion threshold of the proposed method.

5.1 Environment details

We use 4 different environments for our evaluation: (1) CartPole-v0, (2) Acrobot-v1 (3) MountainCar-v0 (4) LunarLander-v2. All of them are included in OpenAI Gym (Brockman et al. 2016). Among them Cart-Pole is the simplest task and Lunar Lander is the most complicated one. The target score to mark each task as solved is listed in Table 3. In Sect. 5.10, we also add an experiment with a more complicated task, Pong, as an additional case study. In this section we provide the details of each environments.

5.1.1 Cart-pole

Cart-Pole (Barto et al. 1983) is a well-known environment for reinforcement learning. In this task, a pole is attached to a cart which placed on a one-dimensional frictionless track. The state consists of the position and velocity of the cart and the angle and angular velocity of the pole. The agent can take two actions: applying either left or right force to the cart. The goal is to keep the cart balanced. An episode ends when the pole falls or the cart goes out of bounds. The agent receives +1 reward for each time step until the episode ends. The maximum step of this environment is 200. The task is considered solved when the average reward is greater than or equal to 195 over 100 consecutive trials.

5.1.2 Acrobot

The acrobot (Sutton 1995; Geramifard et al. 2015) is a two-link robotic system with an actuator at the joint between the two links. There are 6 variables describing the sine and cosine of the two rotational joint angles and the joint angular velocities. The action is either applying +1, 0 or -1 torque on the actuator. Initially, the links point downwards against the gravity, and the objective is to swing the end of the lower link up to a given height. The reward of -1 is given at each time step. The episode ends until 500 steps or the goal is reached. There is no specified reward threshold at which the task is considered solved. In our comparison, we set -100 as the threshold of solving this task as most of the methods converge to this score.

5.1.3 Mountain car

In this task, a car is placed on a one-dimensional track between two steep hills (Moore 1990). The goal is to drive up the hill on the right. At each time step, the agent controls the car by one of the three actions: 1) driving left, 2) driving right, 3) not using engine at all. Two continuous variables are given to describe the position and velocity of the car. Since the engine of the car is not strong enough to accelerate up the hill, the agent must learn to leverage the momentum by driving back and forth. The agent received -1 reward at each time step. The episode ends until 200 steps or the goal is reached. The task defines solving as getting average reward of -110 over 100 consecutive trials.

5.1.4 Lunar lander

The goal of this task is to make the lander lands safely on the landing pad. The state space is described using eight variables: the position and the velocity of the lander described in two-dimension, the angle and the angular velocity of the lander, and whether the leg at each side is contacted to the land. The agent can control the lander by either doing nothing, firing the left orientation engine, firing the main engine, or firing the right orientation engine. Reward for landing on the landing pad is about 100 to 140 points, according to how close the lander is to the landing pad and how close it is to zero velocity. Crashing the lander or making it comes to rest leads to the end of an episode, and will get -100 or +100 points respectively. Firing engine also gets -0.3 points for the main engine and -0.03 points for the engines on both sides. The task is considering solved as getting average reward of 200 over 100 consecutive trials.

5.1.5 Pong

Pong is an Atari game which also included in OpenAI Gym. In this task the player need to control a stick vertically to play Pong with another software agent. We follow the preprocessing step in Mnih et al. (2015) to generate $84 \times 84 \times 4$ images as states. The task ends when one of players get 21 points. Thus the reward of each game is ranging from -21 to 21. The task is considered solved when an agent achieve average reward over 19 in 20 games.

5.2 Experimental setup

We evaluated the proposed method on the four environments described above. For each environment, we evaluated six different methods based on bootstrapped (Osband et al. 2016) or noisy (Fortunato et al. 2017) networks separately. The methods are:

Table 2 Comparison between methods

	Demonstration	Pre-training	Interaction	Query criterion
DQN	No	No	No	No
DQfD	Yes	Yes	No	No
GDQN	Yes	No	Yes	Greedy
BDQN	Yes	No	Yes	Bernoulli
ADQN	Yes	No	Yes	Uncertainty
ADQNP	Yes	Yes	Yes	Uncertainty

1. DQN: Prioritized Double DQN trained without any demonstration
2. DQfD: Deep Q-learning from Demonstration (Hester et al. 2018)
3. GDQN: Greedy query strategy which queries all states until budget is all spent
4. BDQN: Bernoulli query strategy, queries states at fixed probability
5. ADQN: Active DQN, queries states according to proposed query strategy and uncertainty estimation
6. ADQNP: Active DQN with DQfD pretraining. It is pretrained with a pre-collected set of demonstration, and then asks for new demonstration during RL. The demonstration data collected in both stage are kept in replay buffer permanently.

The key differences between the methods are summarized in Table 2. To distinguish whether the method is based on bootstrapped DQN or noisy DQN, in the following we use suffix “-B” to denote methods based on bootstrapped DQN and suffix “-N” to denote methods based on noisy DQN.

We don’t compare the proposed solution with existing active learning methods mainly because most of active learning researches focus on supervised learning and semi-supervised learning. The difference of problem setting make it challenging to apply the traditional active learning methods to RL, as we have described in Sect. 2.

We tuned the the learning rate in $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ and discount factor in $\{0.99, 0.9\}$ for DQN on each environment to ensure reasonable learning progress and then fixed these parameters for all six methods, as listed in Table 4. The network structure applied in all environments was identical: two fully connected hidden layers with 64 neurons followed by another fully connected layer to the Q-Values for each action. The layers all used rectified linear units (ReLU) for non-linearity. We trained the networks using Adam and a ϵ-greedy policy with ϵ annealed linearly from 0.9 to 0.01. We set the parameters of prioritized replay according to Schaul et al. (2016). For bootstrapped DQN, we used 10 bootstrap heads with normalized gradient and shared all the data as in Osband et al. (2016). For the noisy networks, we used factorized noise and followed the initialization and hyperparameter values from Fortunato et al. (2017).

For DQfD, we did not use L2 regularization loss or N-step temporal difference loss, as they brought no benefit to training in our experiments. We set the expert margin $M = 0.8$ as in Hester et al. (2018) and tuned the supervised loss weight λ in $\{10^{-5}, 10^{-4}, 10^{-3}, \ldots, 1\}$. The number of demonstration data and pre-training steps were set to allow DQfD learn a better initial policy than learning from a scratch.

For each query, all ARLD methods receives five consecutive expert demonstrations until the end of the episode. The query threshold t_{query} is tuned in $\{0.05, 0.1, 0.3, 0.5\}$. For ADQNP, the number of demonstration for pretraining is half of DQfD and the query budget

Table 3 Expert's scores on each task. The experiments are repeated 100 times

	Mean score/std	Min. score	Avg. steps	Target score
Cart-Pole	166.77 ± 39.14	93	166.77	195
Acrobot	-128.25 ± 66.86	− 489	128.25	− 100
Mountain Car	-134.0 ± 27.52	− 158	134	−110
Lunar Lander	155.18 ± 55.58	− 16.63	784.92	200

is half of ADQN, resulting in the same number of total demonstrations. All task-specific parameters are all listed in Table 4.

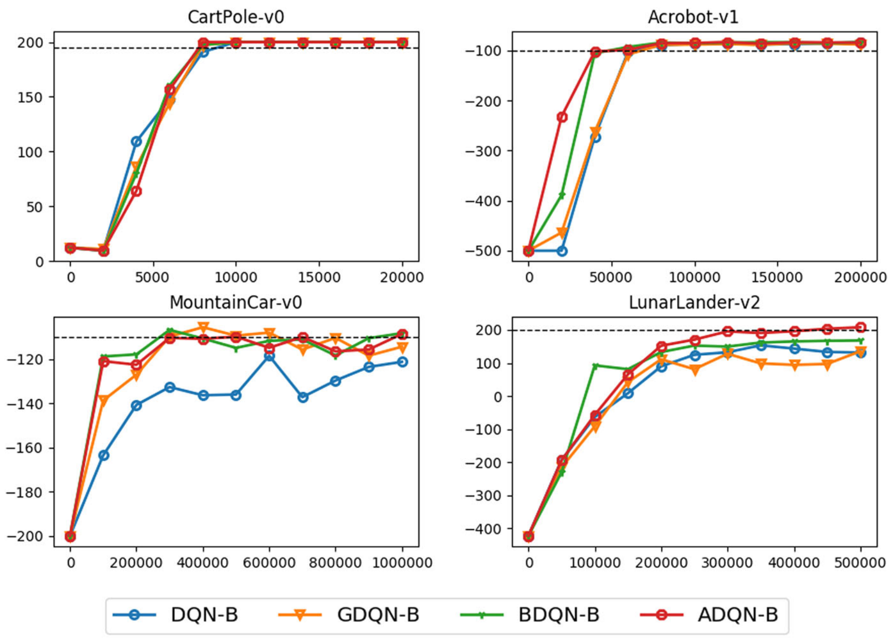
In our experiments, we use artificial experts instead of real human experts to conduct experiments in larger scale. In particular, to obtain an expert for each environment, we saved the prioritized double DQN models during training and evaluated each over 100 episodes. Then we chose as the simulated expert a model that (a) did not perfectly solve the task (b) had reached low performance variance (c) still solved the task before the end of the episode. The reason of choosing with these conditions is to obtain an expert that is as similar to a human one as possible. Condition (a) reflects the fact that a human expert usually does not know the optimal policy to solve the task. Condition (b) represents the consistency in usual human experts. Condition (c) makes the experts sufficiently strong. The choice ensures the experts to be realistic rather than idealistic. These experts were used to collect demonstration data in DQfD and perform interactive demonstration in ADQN. The evaluation statistics of these experts are shown in Table 3. More detailed studies when more extreme artificial experts are considered will be discussed in Sect. 5.6.

All of the experiments were repeated 20 times with different random seeds to obtain more stable results under the nondeterministic ecosystem of RL and the bootstrap method. Figures 3, 4, 5, 6, 7, 8 and 9 show the median of the results over 20 trials. The y-axis indicates the averaged test score, where the test scores in each trial were computed at a fixed frequency by executing 20 test episodes without exploration.

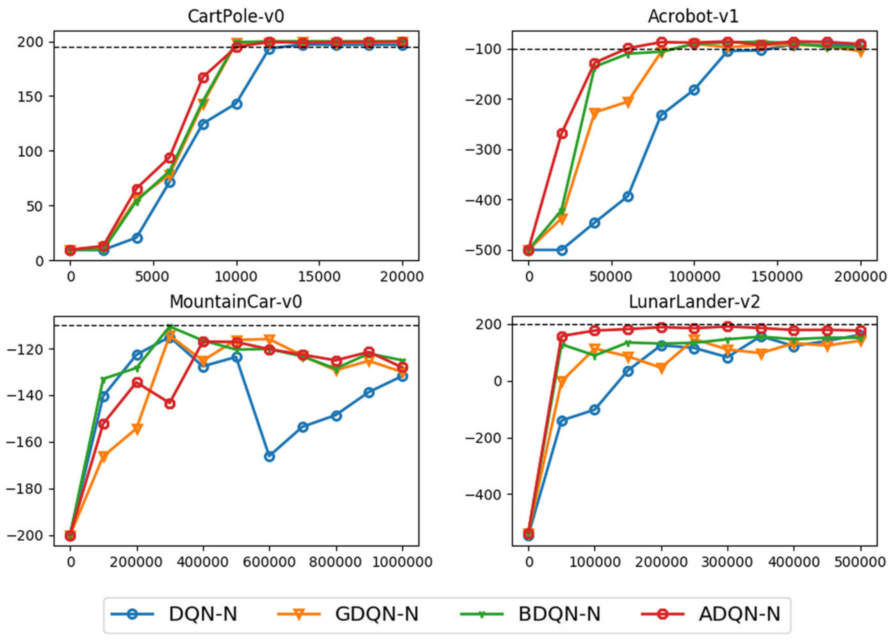
5.3 Comparison between ARLD methods

We first compare the methods without pretraining, i.e., DQN, GDQN, BDQN and ADQN. Table 6 lists the median of number of steps that each method takes to solve the tasks in 20 trials. Among methods based on bootstrapped DQN, the proposed ADQN outperforms other methods in all environments and yields significant improvements in Acrobot, Mountain Car, and Lunar Lander. For methods based on noisy network, ADQN also achieves best performance in three out of four tasks, and improves the learning progress in Acrobot and Lunar Lander dramatically. The strength of ADQN over DQN again confirms the usefulness of interacting with demonstration. Most importantly, the advantage of ADQN over GDQN and BDQN validates that ADQN allows a more effective use of the demonstration efforts by querying strategically at the important moments.

Fig. 2 shows the learning curves of ARLD methods based on the bootstrapped or noisy network separately. The results demonstrate that the methods with demonstration not only outperform the original DQN in general, but also achieve higher score than the simulated experts which provide demonstration. Among the methods with demonstration, ADQN is often the most competitive one, especially in the hardest task of Lunar Lander, which was solved by ADQN with fewer steps and a higher score.



(a) Comparison between methods with bootstrapped DQN



(b) Comparison between methods with noisy DQN

Fig. 2 Comparison between different ARLD methods. The upper part is comparison of methods based on bootstrapped DQN. The lower part is comparison of methods based on noisy DQN. Each plot means the learning curve on that task. The x-axis means the number of training step and the y-axis means the evaluate score. The dashed lines indicate the score of solving the task

5.4 Comparison with pretraining methods

Next, we compare DQfD with ADQN to understand the effect of collecting expert demonstration *before* or *during* training. We also design ADQNP as a simple mixture between the two. The results in Table 6 show that ADQN usually solves the tasks with fewer steps than DQfD or ADQNP, except for the simplest task of Cart Pole. ADQNP also often improves over DQfD when the tasks gets harder. The results justify the effectiveness of leveraging expert demonstration *during* training.

For simulating real-world scenario where the demonstrating humans may not always be perfect, our simulated experts are designed to be realistic but imperfect. Then, methods with pretraining need to take additional steps in the beginning to correct the policy learned from the imperfect experts. This situation explains the performance dropping in the beginning of DQfD for Acrobot and Lunar Lander, as shown in Fig 3. ADQN, on the other hand, demonstrates better ability to leverage the imperfect demonstrations to aid RL.

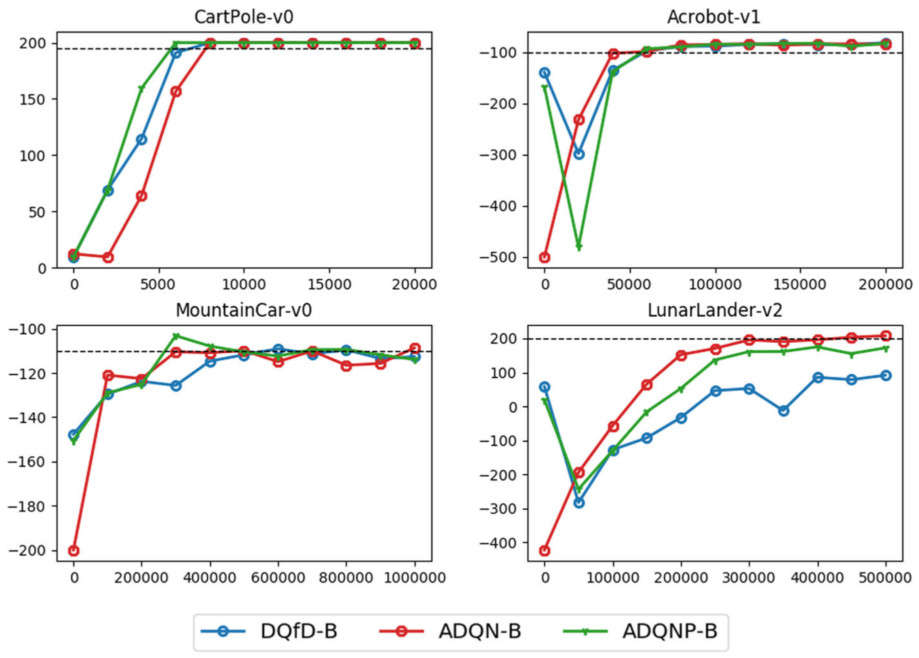
5.5 Effect of query proportion threshold

Active DQN uses two parameters: the number of recent steps for which we compare the uncertainty (N_r) and the proportion threshold that determines whether to make a query given recent steps (t_{query}). Since the uncertainty distribution usually changes smoothly, the value of N_r effects the performance little compared to parameter t_{query}. In Fig. 4 we plot the performance given different values of t_{query}: we observe that in most cases, different choices of t_{query} perform similarly. Moreover, in Table 4 we see that the best values of t_{query} are either 0.1 or 0.3 in Acrobot, Mountain Car, and Lunar Lander; in Cart-Pole, the only exception, the result shows the least variance between choices of t_{query}. Thus, the performance of Active DQN is not sensitive to the parameter t_{query}, and it is easy to choose a value between 0.1 and 0.3 that optimizes performance for all kinds of tasks.

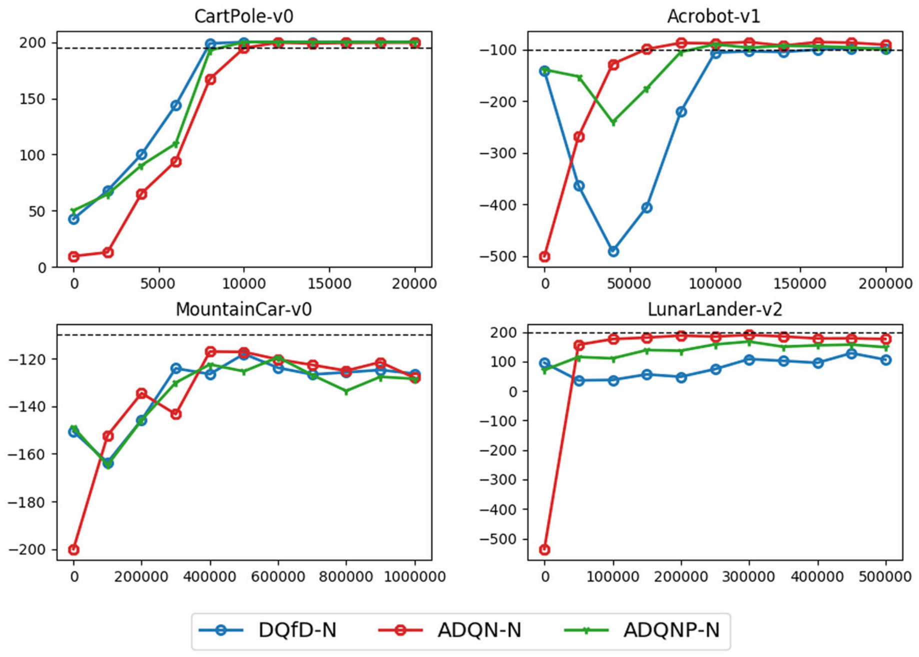
5.6 Effect of expert's quality on ADQN and DQfD

In this subsection, we investigate how different kinds of simulated expert would effect the performance of ADQN and DQfD. We compared three different approaches including the one we used to simulate real human and two other common approaches. First of all, the perfect experts are DQN agents trained on each task until convergence. These well-trained experts can solve their tasks perfectly and efficiently. Second, we obtain weaker experts by applying random noise to the perfect expert. That is, each time an expert is going to make a demonstration, there is a probability the expert will do a random action rather than following the perfect expert's policy. The random behaviors sometimes lead to the end of an episode directly, therefore even though the expert is able to make optimal choices at most of the time, it still might fail to solve the task at the beginning of an episode. Last, as described in Sect. 5.2 and Table 3, we saved the temporary models in the process of training a DQN agent and selected one of them to be a weak expert. Compared to the noisily-acting weak experts, these policy-consistent weak experts act more consistently through an episode, hence their behavior are more similar to a human expert. The performance of simulated experts mentioned above are shown in Table 5.

Figure 5 demonstrates the effect of expert's quality on ADQN and DQfD in Lunar Lander. We experiment with both noisy and bootstrapped network structures along with the three experts mentioned above. The figure shows both DQfD and ADQN can learn efficiently

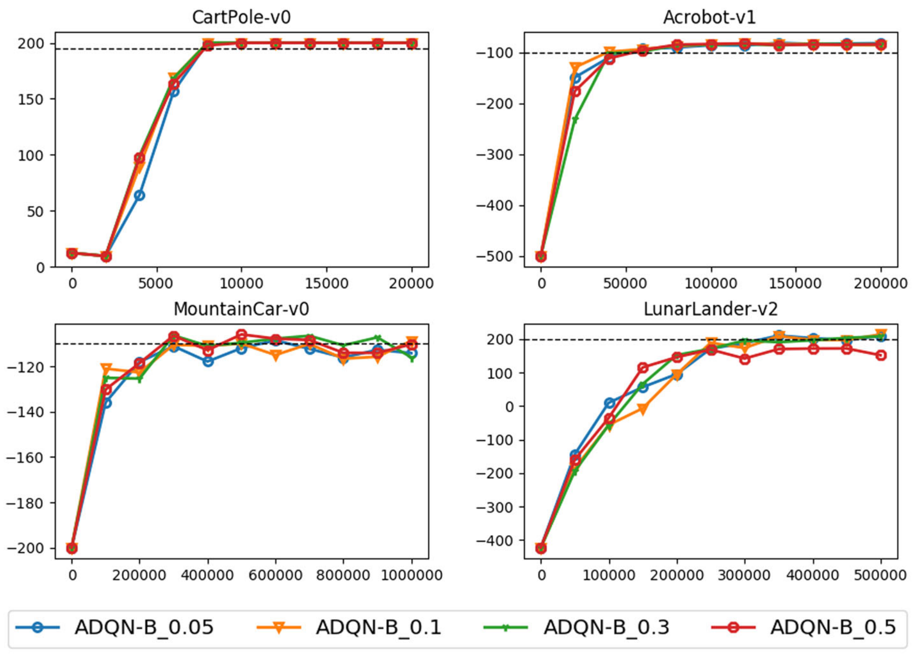


(a) ADQN, DQfD and ADQNP based on bootstrapped DQN

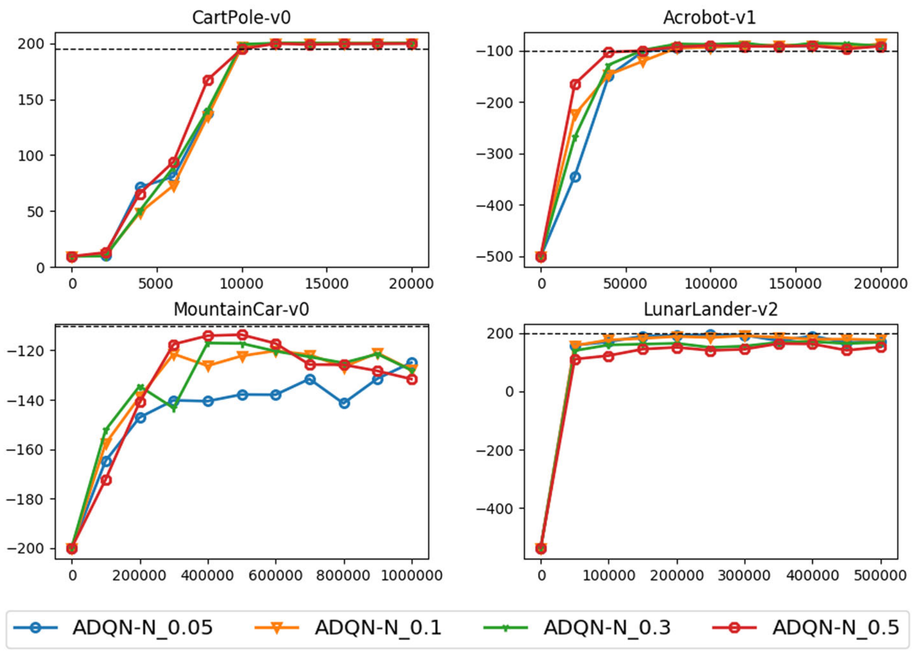


(b) ADQN, DQfD and ADQNP based on noisy DQN

Fig. 3 Comparison between ADQN, DQfD and ADQNP. The dashed lines indicate the score of solving the task



(a) Active DQN based on bootstrapped DQN



(b) Active DQN based on noisy DQN

Fig. 4 Comparison between different query thresholds among {0.05, 0.1, 0.3, 0.5}

Table 4 Task specific parameters

	Cart-pole	Acrobot	Mountain Car	Lunar Lander
Discount factor	0.9	0.99	0.99	0.99
Learning rate	0.0001	0.0001	0.001	0.001
Training steps	20000	200000	500000	500000
# of demo/budget	200	100	500	3000
Memory size	10000	100000	100000	100000
Pre-training steps	10000	10000	10000	30000
DQfD λ	0.00001	1	1	1
ADQN-B t_{query}	0.05	0.3	0.1	0.3
ADQN-N t_{query}	0.5	0.3	0.3	0.1

Table 5 Mean scores of different type of expert

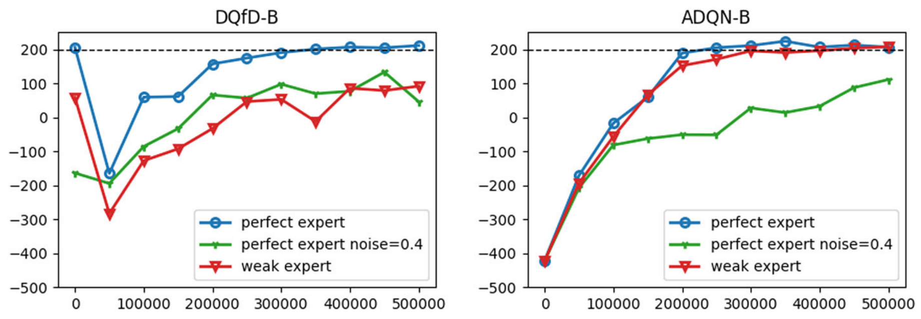
	Cart-Pole	Acrobot	Mountain Car	Lunar Lander
Perfect expert	200.0 ± 0	-76.52 ± 15.12	-102.08 ± 4.78	231.42 ± 16.92
Perfect expert +40% Noise	131.92 ± 63.96	-128.08 ± 24.63	-187.56 ± 12.27	-43.32 ± 59.34
Weak expert	166.77 ± 39.14	-128.25 ± 66.86	-134.0 ± 27.52	155.18 ± 55.58
Target score	195	-100	-110	200

and solve the task within few steps with an perfect expert. On the contrary, both of their performances suffer from the noisily-weak expert. They not only learned slower than learning with perfect experts, but also converge at a worse score. However, while working with the policy-consistent weak expert, though DQfD still perform poorly, ADQN converge at higher scores which are close to ones achieved by working with perfect experts. As a result, we found that ADQN is able to take the advantage of policy-consistent weak experts, which are similar to human experts.

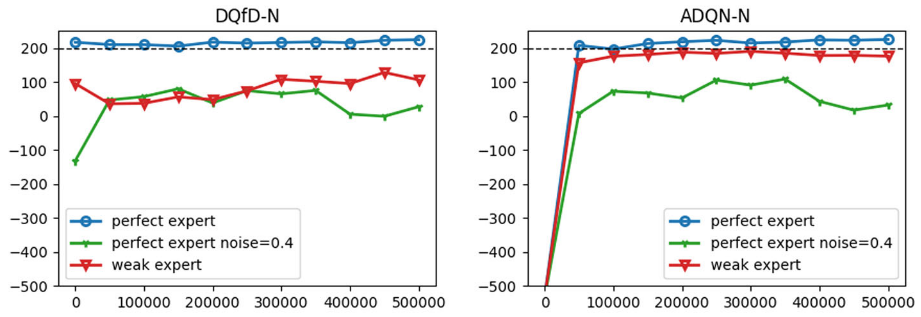
5.7 Comparison between ADQN and DQfD with different query budgets

In previous results, we have shown that RL efficiency can be improved by actively asking for expert demonstration. The results were obtained with a fixed query budget of 3000. In this section, we investigate how different query budgets affect the performance of the "passive" DQfD and the "active" ADQN. Figure 6 shows the learning curves of DQfD and ADQN in Lunar Lander with different query budgets. For DQfD, not surprisingly, a bigger querying budget allows the algorithm to start with more demonstration data, which in term leads to better initial performance. Nevertheless, the performance of DQfD seems to be somewhat grounded to the sub-optimal policy represented by the imperfect demonstration data.

For ADQN, the only case where it performs worse than DQfD is when using noisy network with a really small query budget of 250, which is possibly caused by the sampling bias (Settles 2009) that makes the demonstration data less representative of the expert's policy. For other cases of different query budgets, especially those with a sufficiently large query budget, the valuable demonstration data queried from ADQN soon allows the algorithm to catch up and outperform DQfD significantly in subsequent updates. The results again justify the benefits of actively asking for expert demonstration.



(a) Effect of expert's quality on DQfD-B and ADQN-B

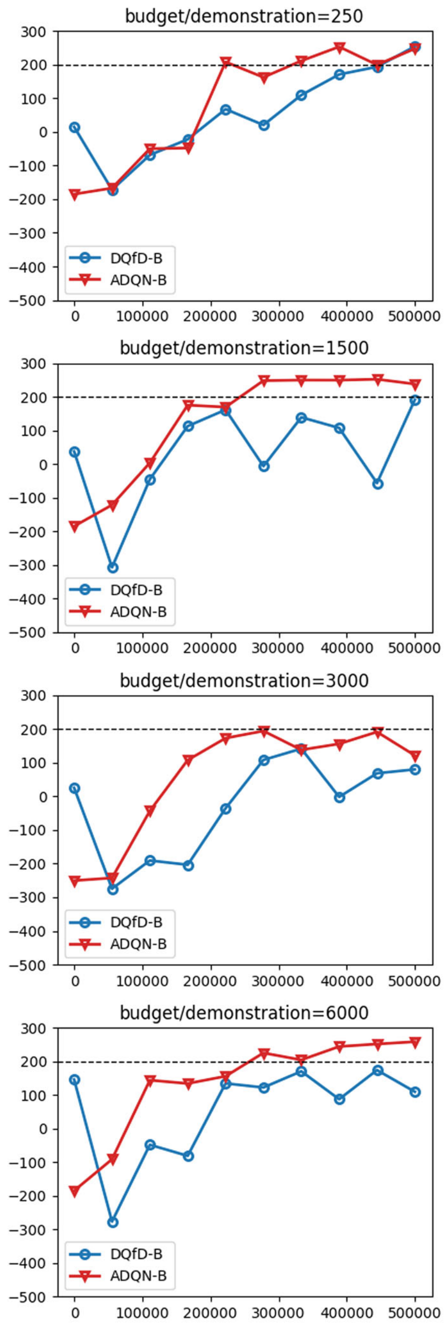


(b) Effect of expert's quality on DQfD-N and ADQN-N

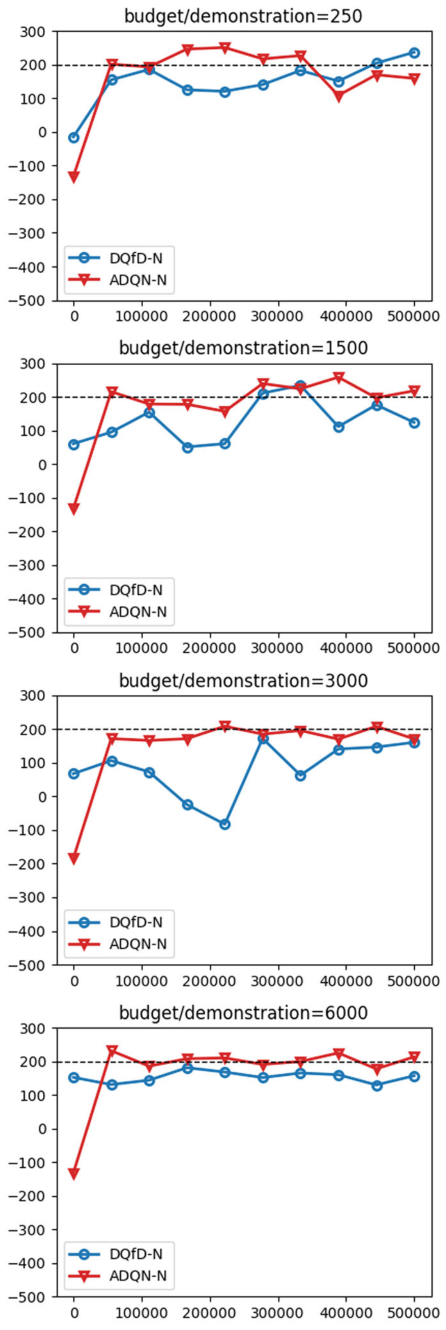
Fig. 5 Comparison between perfect expert, prefect expert with 40% random action, and weak expert

5.8 The evolution of uncertainty measurement

In this subsection, we try to analyse how frequently the agent would ask expert problems and how the uncertainty of the agent evolves during the learning process. Figure 7 shows the number of demonstration each agent has asked during the learning process. In this experiment, we set the query budget as 3000 for all approaches. We compare ADQN with GDQN and BDQN, where GDQN always ask for demonstration until budget running out, and BDQN ask for demonstration with a probability of the query threshold at each step. The setting of query budget and query threshold for each environment are according to Table 4. Apparently, GDQN is always the first one who runs out the budget. In Cart-Pole, Acrobot, and Lunar Lander, BDQN and ADQN both ask for demonstrations in stable frequency, and ADQN usually ask more frequently. On the other hand, for Mountain Car, ADQN asks slower than BDQN, which means there is a period where uncertainty is non-increasing in the beginning of training. Figure 8 compares the uncertainty curves between ADQN and its original DQN counterpart. Although one may expect that the neural network may learn to decrease the uncertainty to obtain the deterministic optimal sollution. Empirical research (Fortunato et al. 2017) has shown the variance of noisy DQN is not really strictly decreasing, and the agent does not necessarily evolve towards a deterministic solution. An interesting observation in our experiment is that though the intervention of expert benefits RL, it does not make the agent converge sooner. Instead, in most of the case the uncertainty of ADQN is higher than the original DQN.

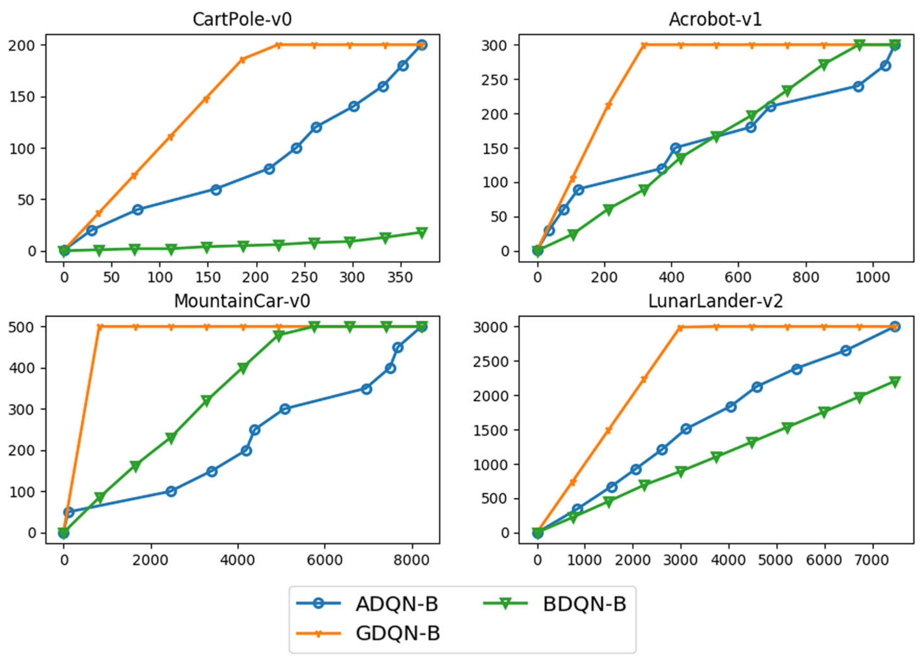


(a) Bootstrapped DQfD and ADQN

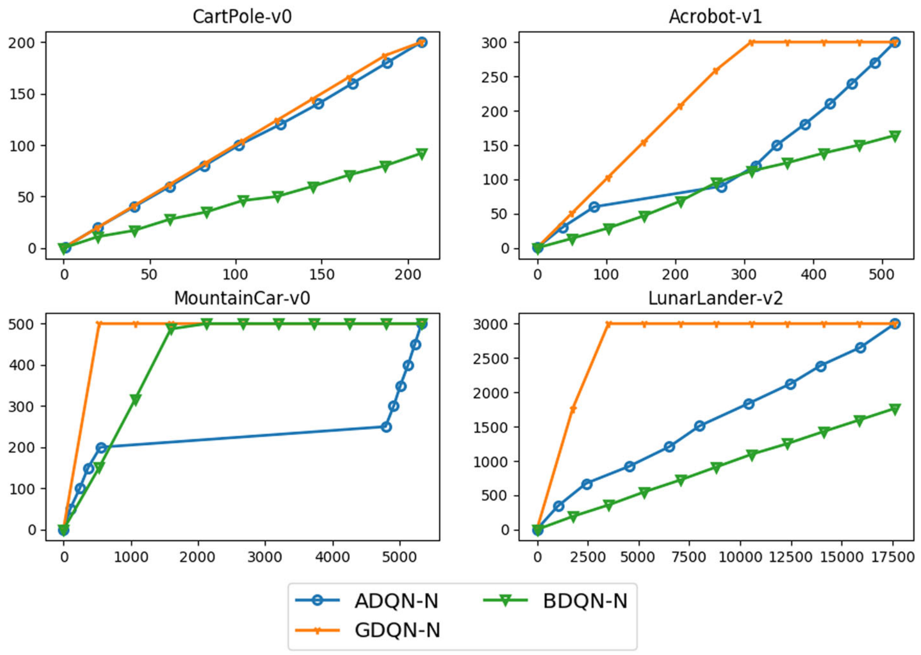


(b) Noisy DQfD and ADQN

Fig. 6 Comparison of the learning curves between DQfD and ADQN with different number of demonstration data or query budgets

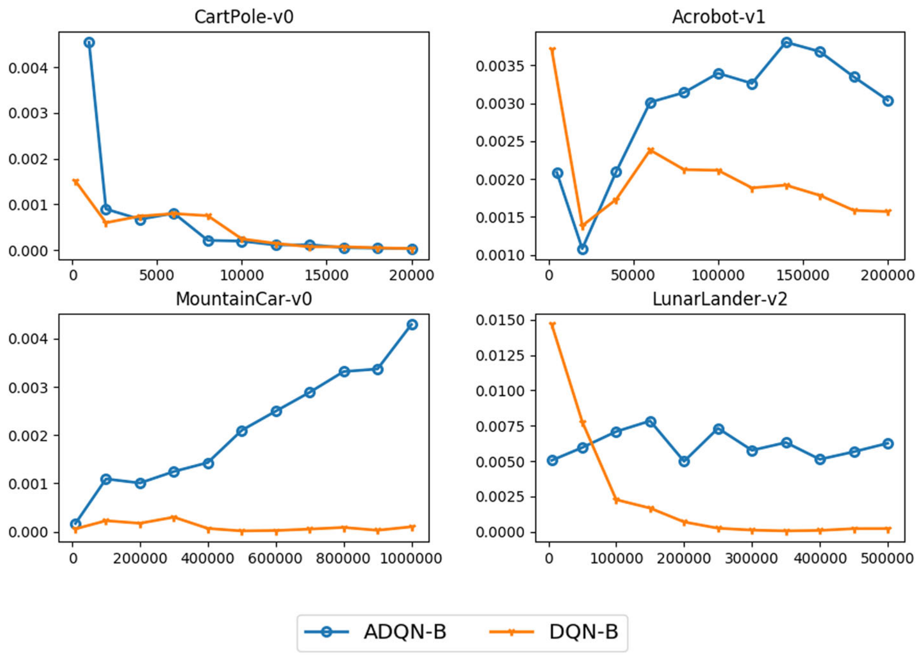


(a) Bootstrapped DQN based methods: GDQN-B, BDQN-B, ADQN-B

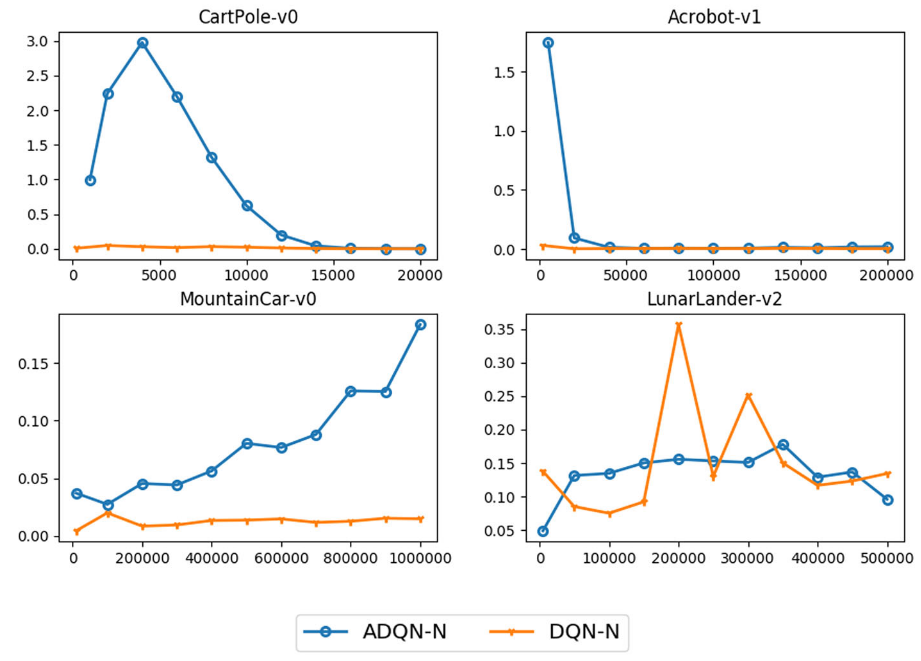


(b) Noisy DQN based methods: GDQN-N, BDQN-N, ADQN-N

Fig. 7 Number of demonstration asked during learning process



(a) JS divergence of DQN-B and ADQN-B



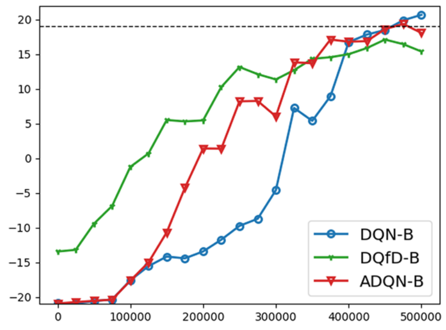
(b) Predictive variance of DQN-N and ADQN-N

Fig. 8 The evolution of uncertainty measurement during learning process

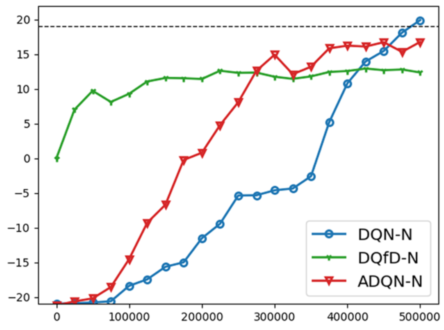
Table 6 The upper table shows the median number of step to solve the task by methods based on bootstrapped DQN

	Cart-Pole	Acrobot	Mountain Car	Lunar Lander
Bootstrapped DQN				
DQN-B	8000	57000	210000	260000
GDQN-B	8000	45000	170000	310000
BDQN-B	7500	31000	100000	217500
ADQN-B	7000	**25000**	**85000**	**205000**
DQfD-B	7500	38000	140000	500000
ADQNP-B	**6000**	49000	135000	267500
Noisy DQN				
DQN-N	13000	114000	190000	355000
GDQN-N	10000	73000	295000	500000
BDQN-N	10000	55000	**170000**	160000
ADQN-N	9500	**8000**	230000	**47500**
DQfD-N	**8000**	95000	300000	402500
ADQNP-N	**8000**	75000	255000	212500

The lower table shows the median number of step to solve the task by methods based on noisy DQN. We use a line to separate DQfD and ADQNP from the other methods to indicate that DQfD and ADQNP are pretrained. The bold numbers indicate the best performance on that task



(a) Bootstrapped DQN



(b) Noisy DQN

Fig. 9 The learning curves of DQN, DQfD and ADQN in Atari Pong

5.9 The choice of bootstrapped DQN or Noisy DQN

In the previous sections, we focus on how the proposed ARLD framework improves the learning process and demonstration efficiency compared to the original DQfD or RL without demonstration. Table 6 shows that bootstrapped DQN and noisy DQN both have their pros and cons on different tasks. While the original bootstrapped DQN (DQN-B) works better then noisy DQN (DQN-N) on more tasks (Cart-Pole, Acrobot, Lunar Lander), their ARLD counterpart (ADQN-B and ADQN-N) both outperform the other on two tasks. That is to say, there is no clear winner between bootstrapped DQN and noisy DQN with regard to performance. However, as for computation efficiency, noisy DQN is clearly better than bootstrapped DQN because of having fewer parameters. Furthermore, from Table 6 we can see that noisy DQN get more significant improvements when coupled with ARLD. The efficiency and improvement aspects suggest that noisy DQN is currently a better choice for ARLD.

5.10 Case study: Atari Pong

To further complement the type of tasks in our experiment. We choose Atari Pong as a case study of more complex task with visual state space. We follow the same procedure described in Sect. 5.2 to tune the hyper-parameters and obtain a simulated expert. The values are: discounted factor: 0.99, learning rate: 10^{-4}, training steps: 500000, number of demonstration: 50000, memory size: 150000, pre-training steps: 5000, DQfD λ (learning rate for demonstration data): 0.01, t_{query} (query threshold): 0.5. The simulated expert achieves 11.85 in average with 4.59 standard deviation.

Compared to the 4 tasks we have experimented above, Pong is a more challenging task because of its longer task duration and miss-intolerant rule. For example, it takes the simulated expert 784.92 steps in average to finish an episode in Lunar Lander. In contrast, it takes 2801.45 steps to finish an episode in Pong. The longer horizon of the task makes it harder to catch the critical states with limited number of queries in a game. In addition, as a competitive game, an agent is more likely to receive negative rewards when it makes mistakes. Therefore even following the demonstrations of expert may lead to negative results. This fact makes the balance between learning from expert and learning from agent harder. As a result, we reduce the learning rate of expert demonstration data from 1.0 to 0.01 and find it works best with both DQfD and ADQN.

Figure 9 illustrated the learning curves of DQN, DQfD, and ADQN with both bootstrapped and noisy network. We can see that although DQfD is benefited from pretraining thus performs better at beginning, it is also affected by the imperfect expert so it can not achieve higher score even after 500000 steps of learning. In contrast, both DQN and ADQN achieve scores higher than the expert. This again shows that ADQN is more robust to learn from imperfect demonstration. Furthermore, with the help of demonstration, ADQN is able to learn faster than DQN at most of the time.

6 Conclusion and future work

In this work, we propose a new framework: Active Reinforcement Learning with Demonstration, which improves RL with demonstration more efficiently with regard to human effort. We also proposed a solution called Active DQN, where we use DQfD to leverage demonstration data and propose a novel uncertainty-based query strategy which applies to diverse tasks. We provide two measurements of the uncertainty: the divergence of Bootstrapped DQN, and the predictive variance of Noisy DQN.

Experimental results show that the proposed methods are indeed benefited from expert demonstration. They learn faster than RL without demonstration and achieve higher score than the expert they learn from. Compared to DQfD, the proposed method is more robust to imperfect experts and more flexible to query demonstration online. Furthermore, we show that the method works with a range of query threshold so we don’t need to tune it cautiously and we can also control the query frequency by adjusting the threshold.

As an initial work on Active Reinforcement Learning with Demonstration, the proposed method has achieved promising performance. A possible extension of this work is to apply it on RL algorithms such as DDPG which work on a continuous action space. A more difficult challenge is to work with on-policy methods which require a different way to learn with demonstration. Another natural way of designing query strategies is to evaluate whether the current decisions by the agents might be bad. Somehow the bad decisions are not as easy to define as the uncertain decisions, and they may not always be the most uncertain ones. It is an interesting future direction to design query strategies based on whether the decisions might be bad.

In this work, we focused on large-scale experiments to demonstrate the feasibility of the proposed framework, and thus relied on only simulated experts. The work serves as an initiative to improve the learning efficiency of RL agents through active querying. To achieve even better efficiency, another interesting future direction is to study how RL agents interact with real human experts, where new challenges and issues will emerge. For example, interacting with real human experts will need a well-designed system to switch control between the agent and the human expert seamlessly, and to allow the expert to observe some recent states easily. The real human experts can also be affected by good/bad memories of past demonstration efforts, making the quality of the demonstration data varying in time. Conquering those challenges in the future direction will make it more realistic for human experts to share their knowledge with RL agents.

Acknowledgements MS was supported by KAKENHI 17H00757. SC and HL were partially supported by MOST 107-2628-E-002-008-MY3 of Taiwan.

References

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans Systems, Man, and Cybernetics*, *13*(5), 834–846.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. CoRR abs/1606.01540. arXiv:1606.01540.

Brys, T., Harutyunyan, A., Suay, H.B., Chernova, S., Taylor, M.E., & Nowé, A. (2015). Reinforcement learning from demonstration through shaping. In *IJCAI* AAAI Press, pp. 3352–3358.

Dagan, I., & Engelson, S.P. (1995). Committee-based sampling for training probabilistic classifiers. In *Machine learning, proceedings of the twelfth international conference on machine learning*, Tahoe City, California, USA, July 9–12, 1995, pp. 150–157, https://doi.org/10.1016/b978-1-55860-377-6.50027-x.

Fortunato, M., Azar, M.G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., & Legg, S. (2017). Noisy networks for exploration. CoRR abs/1706.10295. arXiv:1706.10295.

Gal, Y., Islam, R., & Ghahramani, Z. (2017). Deep bayesian active learning with image data. In Precup, D., Teh, Y. W. (eds.) *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, Sydney, NSW, Australia, 6–11 August 2017, PMLR, *Proceedings of Machine Learning Research* (Vol. 70, Pp. 1183–1192). http://proceedings.mlr.press/v70/gal17a.html.

Geramifard, A., Dann, C., Klein, R. H., Dabney, W., & How, J. P. (2015). Rlpy: a value-function-based reinforcement learning framework for education and research. *Journal of Machine Learning Research*, *16*, 1573–1578.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., Dulac-Arnold, G., Agapiou, J., Leibo, J.Z., & Gruslys, A. (2018). Deep Q-learning from demonstrations. In McIlraith SA, Weinberger KQ (eds) *AAAI*, AAAI Press. Retrieved April 6, 2018, from https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16976.

Hosu, I., Rebedea, T. (2016). Playing atari games with deep reinforcement learning and human checkpoint replay. CoRR abs/1607.05077.

Judah, K., Fern, A. P., Dietterich, T. G., et al. (2014). Active lmitation learning: formal and practical reductions to iid learning. *The Journal of Machine Learning Research*, *15*(1), 3925–3963.

Kang, B., Jie, Z., Feng, J. (2018). Policy optimization with demonstrations. In Dy. J,, Krause, A. (Eds.) *Proceedings of the 35th International Conference on Machine Learning, PMLR, Stockholmsmässan, Stockholm Sweden, Proceedings of Machine Learning Research* (Vol. 80, pp. 2469–2478). Retrieved April 3, 2018, from http://proceedings.mlr.press/v80/kang18a.html.

Krawczyk, B., & Wozniak, M. (2017). Online query by committee for active learning from drifting data streams. In *2017 International Joint Conference on Neural Networks, IJCNN 2017*, Anchorage, AK, USA, May 14–19, 2017, IEEE, pp 2120–2127. https://doi.org/10.1109/IJCNN.2017.7966111.

Lipton, Z.C., Gao, J., Li, L., Li, X., Ahmed, F., & Deng, L. (2016). Efficient exploration for dialog policy learning with deep BBQ networks \& replay buffer spiking. CoRR abs/1608.05081

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, *18*(2), 203–226. https://doi.org/10.1016/0004-3702(82)90040-6.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236.

Moore, A.W. (1990). Efficient memory-based learning for robot control. Tech. rep.

Osband, I., Blundell, C., Pritzel, A., Roy, B.V. (2016). Deep exploration via bootstrapped DQN. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (Eds.) *NIPS* (pp. 4026–4034). Retrieved April 7, 2018, from http://papers.nips.cc/paper/6501-deep-exploration-via-bootstrapped-dqn.

Piot, B., Geist, M., Pietquin, O. (2014). Boosted bellman residual minimization handling expert demonstrations. In *ECML/PKDD (2), Springer, Lecture Notes in Computer Science* (vol 8725, pp. 549–564).

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R.Y., Chen, X., Asfour, T., Abbeel, P., Andrychowicz, M. (2017). Parameter space noise for exploration. CoRR abs/1706.01905. arXiv:1706.01905

Ross, S., Gordon, G.J., Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS, JMLR.org, JMLR Proceedings* (Vol. 15, pp. 627–635).

Schaal, S. (1996). Learning from demonstration. In *NIPS* (pp. 1040–1046). MIT Press .

Schaul, T., Quan, J., Antonoglou, I., Silver, D. (2016). Prioritized experience replay. In *International Conference on Learning Representations*. Puerto Rico

Settles, B. (2009). Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison

Shon, A.P., Verma, D., Rao, R.P.N. (2007). Active imitation learning. In *AAAI* (pp. 756–762) AAAI Press.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Suay, H.B., Brys, T., Taylor, M.E., Chernova, S. (2016). Learning from demonstration for shaping through inverse reinforcement learning. In *AAMAS* (pp. 429–437). ACM.

Subramanian, K., Jr CLI, Thomaz, A.L. (2016). Exploration from demonstration for interactive reinforcement learning. In *AAMAS* (pp 447–456). ACM.

Sun, W., Venkatraman, A., Gordon, G. J., Boots, B., & Bagnell, J. A. (2017). Deeply aggrevated: Differentiable imitation learning for sequential prediction. *ICML, PMLR, Proceedings of Machine Learning Research*, *70*, 3309–3318.

Sutton, R.S. (1995). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *NIPS* (pp. 1038–1044). MIT Press

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning–an introduction. Adaptive computation and machine learning*. Cambridge: MIT Press.

Taylor, M.E., Suay, H.B., Chernova, S. (2011). Integrating reinforcement learning with human demonstrations of varying ability. In *AAMAS, IFAAMAS* (pp. 617–624).

van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In Schuurmans D, Wellman MP (eds) *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, February 12–17, 2016 (pp. 2094–2100). Phoenix, AZ: AAAI Press. Retrieved April 11, 2018, from http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389.

Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., & Riedmiller, M.A. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. CoRR abs/1707.08817. arXiv:1707.08817.

Wang, Z., & Taylor, M.E. (2017). Improving reinforcement learning with confidence-based demonstrations. In *IJCAI* ijcai.org (pp 3027–3033).

Watter, M., Springenberg, J.T., Boedecker, J., & Riedmiller, M.A. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In *NIPS* (pp 2746–2754).