# Active learning from demonstrations

**Roman Castagné**
Master MVA
Ecole des Ponts Paristech
roman.castagne@gmail.com

**Louis Bouvier**
Master MVA
Ecole des Ponts Paristech
louis.bouvier@eleves.enpc.fr

## Abstract

In Reinforcement Learning (RL), an agent shapes a policy to manage a task based on its interaction with its environment. Learning may be slow, starting with a transition phase of experiencing bad rewards, which can be prohibitive for many real-world applications. Active Learning from demonstrations aims at tackling this issue, taking advantage of the knowledge of some experts during training. But querying might be expensive and demonstrations sub-optimal. In this context, we design an algorithm that combines a model with several heads to query demonstrations smartly and learn from RL signals, and a smaller model that focuses on the potentially sub-optimal expert to inform the former using reward shaping during an initial phase. We consider applications in both a tabular setting and a continuous state space.

## 1 Introduction

Learning from demonstrations has been an active field in Reinforcement Learning (RL), Machine Learning and Robotics over the past few years. Indeed, it has been shown that not only demonstrations can speed-up learning, but also enable the agent to exceed the performance of the (human or agent) demonstration provider, as it was the case with the famous AlphaGo algorithm [13] in 2016. Being able to leverage demonstrations can be crucial in real-world applications, for instance robotics [3], where the action and state spaces are usually continuous and where RL signals (current reward and transition) may be inaccessible. We emphasize existing approaches to this setting in section 2.1. Having the opportunity to get RL signals from the environment paves the way for the combination of information drawn from the demonstrations and from the interactions of the agent with its environment in the learning process (section 2.2). The active learning from demonstrations framework we consider corresponds to this dual information too, but it supposes a decision by the agent regarding the demonstrations. Given a certain criterion on the current learning state and a demonstration budget, the agent decides whether to query a new demonstration or not. In section 2.3, we highlight some recent paths in this direction. The main questions we address are:

- How can we define the agent's criterion to ask for new demonstrations online ?
- What can be the form of the demonstrations and how can we benefit from them ?

We base our approach on the recent work of Wang et al. 2019 [16] and Chen et al. 2018 [5] using uncertainty as a metric to request demonstrations. For our experiments, we consider both the tabular-setting of Cliffwalk and the continuous state space of Cartpole. In section 3, we set the theoretical background and define our algorithm. Section 4 highlights the experimental setting. We criticise our results and compare our approach to baselines in section 5.

## 2 Existing approaches to learn from demonstrations

### 2.1 Learning merely from demonstrations

When a direct access to rewards and transitions is unavailable, a wide spectrum of approaches have cropped-up to take demonstrations into account. Some of them used imitation learning techniques to try to get the policy behind the observed demonstrations, as addressed for the driving task [18] without considering the RL Markov Decision Process (MDP). Others used Inverse Reinforcement Learning to model the demonstration provider's reward function (actually a candidate reward function satisfying some additional criteria since this problem is ill-posed as is) and then extract a policy from this latter, or directly used generative adversarial imitation learning [8]. Another example of learning merely from demonstrations is DAGGER [12], where an aggregated data-set was built with a constant assistance of the demonstration provider.

### 2.2 Combining demonstrations and RL signals

In a different context, some ideas have been emphasized to incorporate the demonstrations' information into the MDP and to combine it with RL signals through policy shaping [2], reward shaping [1], or including a policy improvement bias towards demonstration [4]. Similar ideas added an imitation loss [7] or constraints for the optimization problem of Approximate Policy Iteration [9] in order to bias the learning towards demonstrations. Given the fact that the latter might be sub-optimal, few or insufficient to properly model a policy, some research paths have been considered to incorporate them into the exploration process with confidence analysis like the CHAT algorithm [15].

### 2.3 Actively querying demonstrations

More specifically, the field of active learning from demonstrations in RL has been explored, motivated by the following assessments: querying demonstrations may be expensive, and focusing on critical states can foster learning efficiency. As we mentioned above, the criterion to query demonstrations is crucial. An idea that seems natural and significantly studied in literature is to use demonstration budget in states or regions where the uncertainty (over the current policy, the current $Q$-function or the demonstration provider's modeled policy for instance) is high. In this context, Bootstrapped DQN and Noisy Networks were suggested to estimate recent states' uncertainty during learning and query demonstrations in states where uncertainty is high [5]. In a similar way, DROP [16] is a framework to build confidence metrics online and to query demonstrations in regions where confidence is low. Some other uncertainty measures have been studied. [14] used leverage and discrepancy in the linear regression framework to account for states that have been poorly explored and for those leading to high model errors respectively.

## 3 Theoretical setting

### 3.1 General context

We consider the standard Reinforcement Learning setting with an agent interacting with an environment, which can be modeled by a Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, p)$, respectively the state space, the action space, the possibly stochastic reward $r : \mathcal{S} \times \mathcal{A} \to [r_{min}, r_{max}]$ and the transition probability $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ such that $p(.|s, a)$ is a transition probability over the state space given any state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$. Throughout this report we present our results in an infinite horizon setting, involving a discounting factor $\gamma$. We consider both a tabular case where $\mathcal{S}$ and $\mathcal{A}$ are discrete, and a setting where $\mathcal{S}$ is continuous and $\mathcal{A}$ is discrete. Our aim is to find a policy denoted by $\pi^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ which can be seen as a density over the action space $\mathcal{A}$ given a state $s \in \mathcal{S}$, $\pi^*(.|s)$, that maximizes the expected discounted cumulative reward:

$$\pi^* \in \operatorname{argmax}_\pi \sum_t \gamma^t \mathbb{E}_{s_t, a_t \sim \pi(.|s_t)} \left[ r_t \right] \tag{1}$$

In addition to this standard RL setting, we have at our disposal a limited budget $B \in \mathbb{N}$ of demonstrations that we restrict to state-action pairs (that lead to natural transitions): $(s_i^d, a_i^d)_{1 \leq i \leq B}$. This set is not fixed at the beginning of the learning process, only $B$ is. We highlight the fact that given the small value of $B$, allocating those demonstrations online in a way that tends to optimize the learning benefit drawn from their observation is crucial. Besides, we do not assume they reflect an optimal behaviour.

In this context, adding an imitation margin or loss to bias our policy towards demonstrations during the whole training as in [7] for instance does not seem to be appropriate because of their potential sub-optimality, of their online acquisition, and because the resulting optimization task may have divergent losses [6]. Instead, we choose to integrate demonstrations during the learning process following two ideas: merely totting them up in the replay buffer our model is trained on (or observing them in the tabular case), and temporally shaping the reward based on a model of the demonstration policy. We choose to jump on the bandwagon of Active Learning from Demonstrations based on uncertainty estimations. Indeed, as we mentioned above, the crux of the matter is to choose when to query demonstrations and to add them in the transitions data set so as to draw as much benefit as possible from the budget.

Most of the learning methods we consider are based on $Q$-learning, an off-policy algorithm that has shown great success in RL, especially combined with Deep Learning architectures [10].

## 3.2 Jensen-Shannon divergence and uncertainty estimation on the learning agent

We base our algorithm on Bootstrapped DQN [11], adapted for active learning [5]. The main idea of Bootstrapped DQN is to train $K \in \mathbb{N}$ different networks or a single network with $K$ different heads to model the $Q$-function, with $K$ distinct target networks, and pick one of them at each learning episode to follow a greedy exploration. Stochastic gradient descent is used to update the weights of the current network on a batch drawn from the replay buffer, and possibly the weights of the other ones (depending on a Bernoulli realization for instance). The random initialization of the weights, the stochasticity of the gradient descent and the fact that we have different target networks foster a certain diversity among the networks, despite the fact that they are partially or totally trained on the same samples, as discussed in the appendix of [11]. Chen et al. [5] emphasized an idea to estimate the uncertainty of the learning agent in a given state $s \in \mathcal{S}$ with the Jensen-Shannon divergence of the policies $(\pi_k(.|s))_{1\leq k\leq K}$ induced by the different networks $(Q_k)_{1\leq k\leq K}$ of the Bootstrapped DQN as follows:

$$\pi_k(a|s;\theta) = \frac{e^{Q_k(s,a;\theta)}}{\sum\limits_{a'\in\mathcal{A}} e^{Q_k(s,a';\theta)}},\ \forall a \in \mathcal{A} \tag{2}$$

$$JS((\pi_k)_{1\leq k\leq K}; s) = \mathcal{H}(\frac{1}{K}\sum_{k=1}^{K}\pi_k(.|s)) - \frac{1}{K}\sum_{k=1}^{K}\mathcal{H}(\pi_k(.|s)) \tag{3}$$

Where $\theta$ denotes the current weights of the network (or set of networks) and where $\mathcal{H}(p)$ for a discrete density $p : \mathcal{X} \to \mathbb{R}$ is the entropy of $p$ defined as $\mathcal{H}(p) = -\sum_{x\in\mathcal{X}} p(x)log(p(x))$.

In the case of tabular setting, we do not consider the $\theta$ parameters but the current tabular values of the $Q$-functions. The different policies $\pi_k$ for $k \in [K]$ can be derived with the same equation 2 and we can compute in a similar way the Jensen-Shannon divergence with equation 3. The main difference is that since in the tabular $Q$-Learning algorithm updates are deterministic given a transition, we must enforce diversity with a randomized update.

## 3.3 Active queries based on uncertainty

Once we have defined a way to estimate uncertainty at a given state with equation 3, we can derive a simple way to decide whether or not the agent asks for a new demonstration. We follow the principle of [5] keeping track of the last values of the Jensen-Shannon divergence computed with a queue of length $L_q$, and we decide to query a demonstration when the current value of the divergence is greater than a certain percentile $p_r$ of the queue. This process is backed by the following idea: because the $K$ heads or networks emphasize a diversity of policies given the same environment and learning process, asking demonstrations in states that lead to high divergence may enable us to focus on crucial ones (intrinsically hard to model).

### 3.4 Building a model of the demonstration policy

The main hindrance of an algorithm whereby demonstrated transitions are added to the replay-buffer is that they are merely observed in the same way as transitions during the exploration. Though it seems an efficient way to foster robustness against imperfect or noisy demonstrations, their scarcity may require to give them a predominant position. Therefore, we proceed to another supervised learning task: building a model of the policy behind the demonstrations, given the current data set of demonstrations (states and corresponding chosen actions). To do so, we consider a Gaussian kernel model.

The choice of a kernel method is underpinned by the scarcity of the demonstrations. We can proceed to a Kernel Ridge Regression or to a Kernel Logistic Regression (or Kernel SVM) depending on the dimension of the action space.

### 3.5 Active queries and reward shaping: our approach

As stated above, we decide not to bias our learning agent towards demonstrations with an explicit loss or margin during the whole learning process for reasons mentioned section 3.1. Nonetheless, we think, regarding past results and critics of DQN [5, 6], that demonstrations only added to the replay-buffer would form a "too low signal" to speed up learning. We thus propose a way to find a trade-off between the two approaches: based on the current model of the demonstration policy defined in section 3.4, we proceed to use reward shaping [1] in addition to the observation of the demonstration transition. We temporarily use an updated reward $\hat{r}_t$:

$$\hat{r}_t = r_t + \lambda \Big[ 1 - (p(a|s_t, demo\ model) - \pi(a|s_t))^2 \Big] \tag{4}$$

Intuitively, we will initially prefer actions in accordance with observed and modeled demonstrations. We do not start shaping the reward from the beginning of learning, but at the point when sufficient demonstrations are available to foster a reasonable generalization of their observation.

It has been proven that shaping with potential functions over the state-action pairs $\Phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ does not change the total order over policies if $\Phi$ is stationary [17]. However, active learning from demonstrations induces by definition an online query and update of the information drawn from the demonstrations. Encapsulating part of this latter in a reward can not lead to a stationary potential function. We are aware of this limit and thus stop shaping the reward after a fixed time (for instance twice the time step when the last demonstration was requested).

Given the way we request demonstrations, we hope the shaped reward will give informative indications to the learning agent. Reward shaping seems adapted to settings where the reward function $r$ is poorly informative (sparse for instance, like in Cartpole). Besides, choosing to stop shaping at a given learning step prevents us from being highly limited by a sub-optimal source of demonstrations. We present our approach in Algorithm 1.

## 4 CliffWalk and CartPole settings

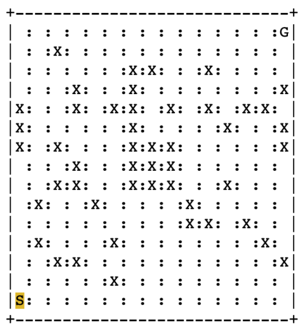
We consider two environments to evaluate the ideas of section 3. As a matter of principle, we start with a simple tabular setting with Cliffwalk, and extend our study to a continuous state space with Cartpole.

CliffWalk is a 2D game in which an agent has to go from a starting point to a finish point while maximising its reward. It can move vertically or horizontally on a 2D grid that consists in plates (reward $-0.1$), pits (reward $-5$), the starting point and the finishing point (reward 5). When the agent reaches the finishing point, the game stops. This corresponds to a tabular setting: the state of the agent is a single integer corresponding to the id of the tile it is on. An example of this environment can be

**Algorithm 1:** Active learning and reward shaping: one episode

```
input : $(Q_k)_{1\leq k\leq K}$, K $Q$-functions parameterized by different weights and having different
        target networks, initial state $s$, budget $B$, current ExpertModel, current divergence
        queue $U\_queue$, current demonstration data set $\mathcal{D}_{dem}$ and replay buffer $\mathcal{D}_{buff}$
output : Updated input
1  Sample: $arm \sim \mathcal{U}([K])$
2  while episode has not ended do
3      if $B > 0$ then                                        // we have a fixed budget
4          $U \leftarrow$ Jensen-Shannon $((\pi_k)_{1\leq k\leq K}; s)$;
5          $U_{threshold} \leftarrow$ Percentile $(U\_queue)$ ;
6          $U\_queue$.push($U$);
7          if $U > U_{threshold}$ then                      // current divergence high
8              $B \leftarrow \max(B-1, 0)$;
9              action $\leftarrow$ Expert query;
10             $\mathcal{D}_{dem} \leftarrow \mathcal{D}_{dem} \cup \{(s, \text{action})\}$;
11         else
12             action $\leftarrow \text{argmax}_{a\in\mathcal{A}} \pi_{arm}(a|s)$;
13         end
14     else
15         action $\leftarrow \text{argmax}_{a\in\mathcal{A}} \pi_{arm}(a|s)$;
16     end
17     take action, observe $s'$, $r$ from environment;
18     Possibly shape: $r \leftarrow shape$(ExpertModel, $s$, action, $s'$, $r$);
19     $\mathcal{D}_{buff} \leftarrow \mathcal{D}_{buff} \cup \{(s, \text{action}, s', r)\}$;
20     update $(Q_k)_{1\leq k\leq K}$ with a batch from $\mathcal{D}_{buff}$ and possibly update target networks;
21     Possibly update ExpertModel from $\mathcal{D}_{dem}$;
22     $s \leftarrow s'$;
23 end
```

seen in figure 1. Transitions are stochastic and rewards deterministic here. We emphasize that several strategies lead to the goal state but some of them are sub-optimal in terms of expected accumulated discounted reward. More precisely, we consider a $15 \times 15$ grid, a discounting factor $\gamma = 0.99$ and a stochastic transition with a probability of success $p_{success} = 0.9$ (the agent effectively moves according to the chosen action $a$ with probability $p_{success}$ and follows a random transition among the 4 possible ones otherwise).



```
+-----------------------------+
| : : : : : : : : : : : : : :G|
| : :X: : : : : : : : : : : : |
| : : : : : :X:X: : :X: : : : |
| : : :X: : :X: : : : : : : :X|
|X: : :X: :X:X: :X: :X: :X:X: |
|X: : : : : :X: : : : :X: : :X|
|X: :X: : : :X:X:X: : : : : :X|
| : : :X: : :X:X:X: : : : : : |
| : :X:X: : :X:X:X: : :X: : : |
| :X: : :X: : : : :X: : : : : |
| : : : : : : : : :X:X: :X: : |
| :X: : : :X: : : : : : : :X: |
| : :X:X: : : : : : : : : : :X|
| : : : : :X: : : : : : : : : |
|S: : : : : : : : : : : : : : |
+-----------------------------+
```

Figure 1: Screenshot of the CliffWalk environment. Each X is a pit. The agent starts from the tile S and has to reach the tile G to end the episode.

CartPole is a classic control problem in which an agent has to balance vertically a pole that can rotate without friction around a cart that moves only horizontally. The episode ends when the pole falls with too much angle on one side or the other, when the cart reaches the end of the screen, or when the episode reaches a time limit of 1000 steps. The state the environment returns is a vector of size 4, with elements corresponding to the cart position, its velocity, the pole angle as well as the velocity of the tip of the pole. An example of the environment taken at a specific step is shown in figure 2. CartPole environment is implemented through OpenAI gym's library[1].

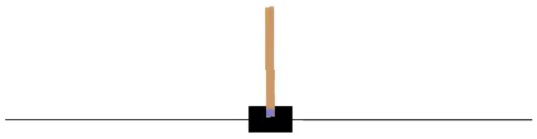


Figure 2: Screenshot of the CartPole environment. The agent controls the cart horizontally on the line.

We argue that those two environments, corresponding to two very different settings (tabular and continuous case), enable us to evaluate our algorithms in two important frameworks of reinforcement learning.

## 5 Results and Discussion

In this section, we present the experimental results of our algorithms. We also expose our training details and hyper-parameters in order to foster reproducibility of our results and criticise them.

### 5.1 Exploiting expert demonstrations

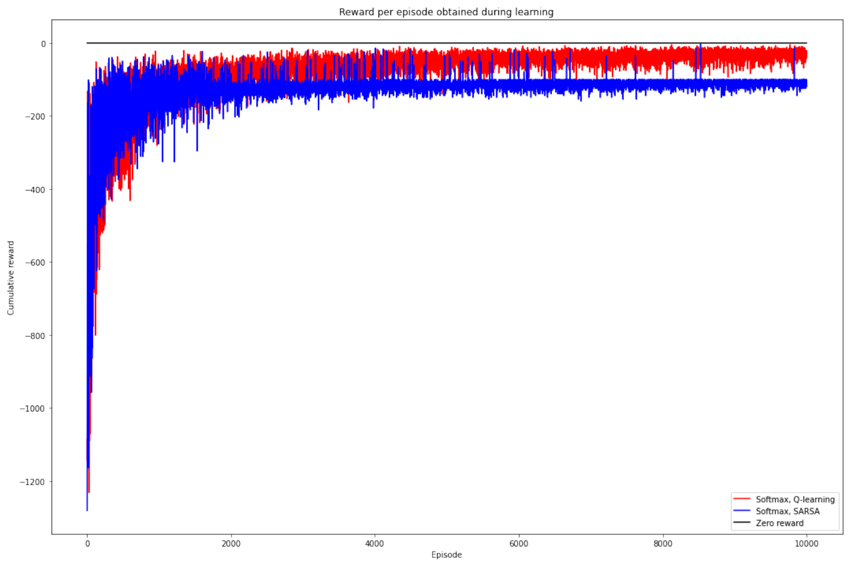
#### 5.1.1 CliffWalk

To provide demonstrations, we train two agents, one with $Q$-Learning algorithm and one with SARSA, both of them using softmax exploration (see the top of figure 3). We see that the "path" found by SARSA to reach the goal has a lower cumulative reward than the one found by $Q$-Learning. We can then use the policies given by different stages of the two algorithms to provide demonstrations with several levels of expected cumulative reward.

In this setting, we compare several algorithms: $Q$-Learning and SARSA learning only based on RL signals, $Q$-Learning and SARSA learning with all budget $B = 100$ transitions observed at the beginning and a tabular version of Bootstrapped DQN [5], as depicted in sections 3.2 and 3.3 (without reward shaping at this stage).

At the bottom of figure 3, we see that merely observing demonstrated transitions at the beginning speeds up learning both for SARSA and $Q$-Learning. Bootstrapped $Q$-Learning tends to be faster than the latter. Nonetheless, varying the parameters of this algorithm, we understand the gain in learning speed is due to the greedy policy the exploration is following rather than to the uncertainty estimation. Indeed, changing the number of models $K$ does not affect the results empirically (neither does the percentile, the queue length or even the quality of the demonstrations on average). Changing the environment, to make it more complex (with a larger grid or a smaller $p_{success}$) leads to very similar results. Demonstrations restricted to transitions in a tabular case where no smoothness or regularity is observable over the state space seem indeed very difficult to leverage, all the more so as they are scarce and thus only observed for a limited portion of the state space. Given this assessment of the combination of a poor structure in the state representation and a small budget $B$ of local information, the tabular case turned out to be a difficult setting in the context of active learning from limited demonstrations. We focus on the second application to test the approach defined in Algorithm 1.

[1]`https://gym.openai.com/envs/CartPole-v0/`



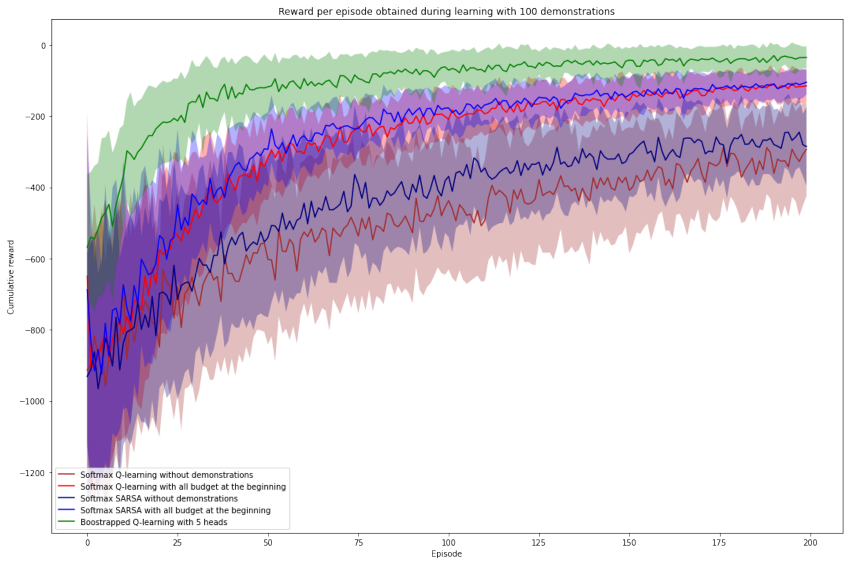


Figure 3: Cumulative reward over learning episodes. At the top: $Q$-Learning and SARSA, learning merely from the RL signals (with a softmax exploration). At the bottom: $Q$-Learning and SARSA, learning merely from the RL signals, $Q$-Learning and SARSA, with all budget $B = 100$ used at the beginning and Bootstrapped $Q$-Learning with active queries

### 5.1.2 Cartpole

We expose in figure 4 the results on the Cartpole environment for a Deep Q Network with all its budget used at the beginning, the Bootstrapped Deep Q Network presented in section 3.2 which uses uncertainty to query the demonstration provider, the same model with an additional learner that models the demonstration using Kernel Ridge Regression as presented in section 3.4, and a similar model using Kernel SVM (both of the latter to shape the reward at the beginning of learning). All models have access to the same budget when training.

We trained our algorithms 50 times and present their expected reward. We observe an important variance, which prevents us from being able to precisely interpret the results. However, we clearly see an improvement of using Bootstrapped DQN over the original DQN model witnessing all demonstrations at the beginning. Indeed, the original DQN model fails at exploiting the demonstrations: after a small surge in total reward at the beginning due to the replay buffer being filled mostly by demonstrations, its performance drops for about 100 episodes until it finally starts learning. Giving this model its budget at the beginning delays the learning rather than helps it. When using improved query mechanics, the model starts increasing its reward sooner, around 50 iterations.

The system modeling demonstrations can quickly learn a good policy with only a few training samples. Over 50 runs and seeing 100 demonstrated transitions from an agent that performs a 1000 score on average, it obtains a median score of 180 when evaluated as a policy on its own. Our aim when using a kind of reward shaping as described in section 3.5 is to exploit this signal even when the demonstration signal is not available (in states where no demonstration has been observed or for a fixed period after the budget is exhausted). As can be seen in figure 4, both models of demonstration (with Support Vector Machines and Kernel Ridge Regression) succeed at helping the policy network to get to good levels of reward faster. For instance, using Kernel Ridge Regression, our model reaches an expected reward of 100 after about 70 iterations, while Bootstrapped DQN reaches it at about 110 iterations, and even later for DQN. Empirically, using Kernel Ridge Regression seems to yield even better results than SVM, although the high variance makes interpretation hard.

In order to balance the reward given by the model of demonstrations with the reward from the environment, we use a weight parameter. If it is too low, then the reward corresponds only to the one from the environment, leading to a model similar to Bootstrapped DQN. On the contrary, if it is too high, the main policy modeled depends too much on a possibly biased or poorly trained model. This parameter could also be seen as a way to cope with sub-optimal demonstrations, if the level of the expert is known in advance. We performed grid search to find the optimal value of this weight, 0.75. All the results were obtained with this value.

Finally, to ensure that the gain in performances from Bootstrapped DQN is not only due to its greedy nature as seems to be the case in the tabular setting, we plot the performance of a Bootstrapped DQN with a single head. This model uses a greedy strategy instead of $\epsilon$-greedy for the simple DQN. As can be seen in figure 6 (in the appendix), the model fails to learn from the environment, thus demonstrating the usefulness of using uncertainty in querying demonstrations.

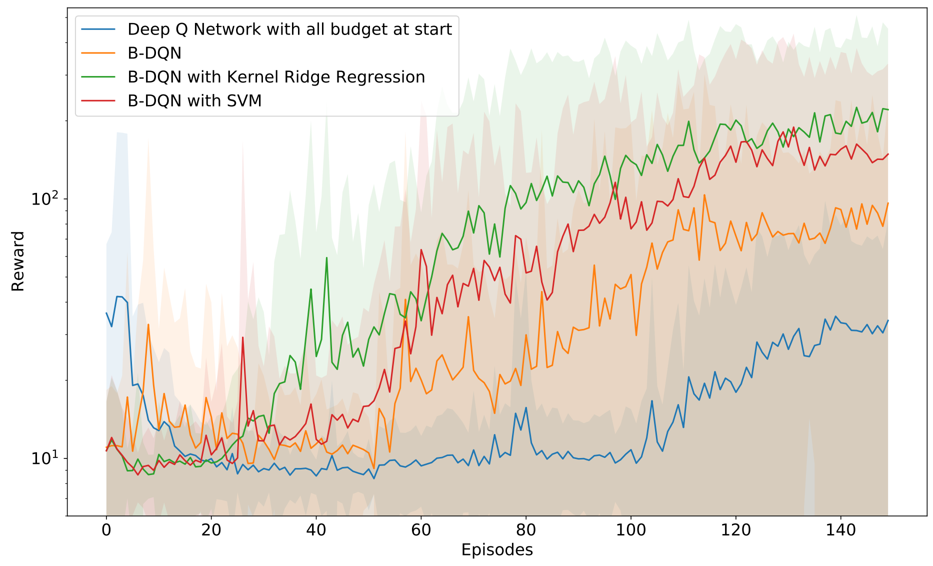


Figure 4: Expected reward when training agents with Deep Q Network (DQN), Bootstrapped DQN (B-DQN), Bootstrapped DQN with a Kernel Ridge regressor or a SVM modelling demonstrations. The lines here correspond to the mean over 50 runs, and the shaded part to the variance. We plot in the log domain for better visualisation.

## 5.2 Behaviour with sub-optimal demonstrations

In order to mimic sub-optimal demonstrations, we train several agents until they reach a fixed expected reward. We choose these rewards to be $50$, $100$, $200$, $500$ and $1000$. We then use these agents (trained policy networks) to provide demonstrations when requested by the agent currently being trained. The learning curve of the model depending on the level of sub-optimality of the demonstration provider can be seen in figure 5.

Due to the high variance, it is hard to interpret this graph precisely. However, we can still notice that our model is relatively robust to sub-optimal demonstration. Indeed, models provided with demonstrations from an expert scoring $200$ or above behave in a similar fashion. However, it seems that taking a demonstration model that has poor performance (getting score below $100$) hurts the learning speed. It is worth noting that the experts were saved when reaching the desired level of rewards, but did not necessarily sustain the same rewards throughout all their training.

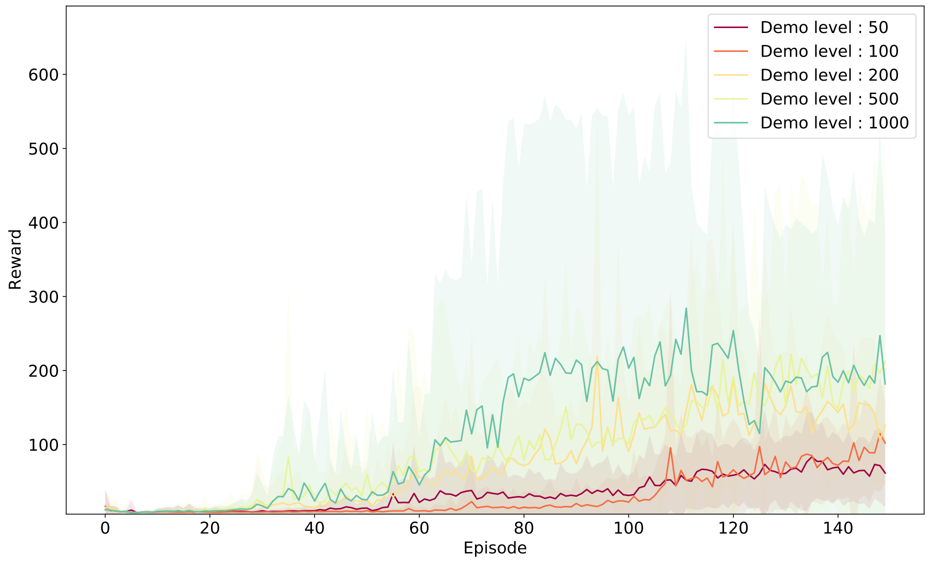


Figure 5: Expected reward when training agents with different levels of sub-optimality. The lines here correspond to the mean over 10 runs, and the shaded part to the variance. We plot in the log domain for better visualisation.

## 5.3 Training details

For CartPole, we use Adam to optimize our model, with a learning rate of $0.001$. We set the size of the replay buffer to $10,000$ transitions, and a batch size of $64$. We explore the state space using an $\epsilon$-greedy strategy, with $\epsilon$ decaying exponentially from $1$ to $0.05$.

We update the target network every $50$ iterations. For Bootstrapped DQN, we found that using $2$ heads gave the best performance. The Kernel Ridge Regression has parameters $\alpha = 0.1$ and $\gamma = 0.1$, and the SVM has parameter $C = 10$. Both models are available in the Scikit-Learn library[2].

## 5.4 Discussion

Two main factors can explain why it is hard to exploit demonstrations in the tabular setting. First, the environment is simple enough to be solved without help from an external entity. Indeed, simply taking a greedy strategy yields very good results. This means that even with minimal or no exploration, an agent can quickly get decent rewards in CliffWalk, thus eliminating the need for a demonstration provider. Secondly, this environment does not possess any global regularity, linking

[2]`https://scikit-learn.org/stable/`

states together. For instance, having two states next to each other does not mean that the optimal action in each state is the same, and does not tell anything about similarities in the rewards. For these reasons, we switched to an environment with a lot more regularity in the state and reward spaces that allows generalising demonstrations to a wider spectrum of states.

In the Cartpole environment, demonstrations given at a specific state might be useful in a different state. For instance, a demonstration given for a specific angle of rotation and speed is useful independently of the position of the cart on the horizontal axis. However, simply giving demonstrations at the beginning, as was demonstrated with DQN, is not informative enough. Indeed, as explained in [6], witnessing a demonstration informs about a good action, but not about traps and low reward decisions. Thus, after having seen a batch of good actions, the agent still has to explore other states to learn a better $Q$-function. This explains why even with demonstrated actions, DQN takes more than 100 episodes before learning a better policy.

Bootstrapped DQN enables to identify states where the divergence between *all heads* is high, similar to an ensemble of models. This way, it allows exploration while speeding the learning in inherently hard states for the $Q$-function. We can then query the demonstration provider only when needed which makes it particularly suitable in the context of active learning. However, the mechanism after the expert has been requested remains the same: the demonstrated transition is added to the replay buffer, and therefore diluted with transitions taken by a sub-optimal $Q$-function. We see here a trade-off: we would like to spend with care and thus relatively slowly our budget to leverage demonstrations as much as possible, but waiting entails a larger dilution in the buffer. Therefore, Bootstrapped DQN takes more than 100 iterations before reaching rewards in the hundreds for an episode.

To address the scarcity of the demonstration provider information, two possible solutions could be proposed. The first simple one is to sample demonstrated transitions more frequently in the replay buffer. The second approach is the one we used, similar to reward shaping using an additional learner. Using Kernel methods, we can make the most out of the small demonstration buffer. Using two complementary models (the kernel-based one with small capacity but capable of learning quickly, and the Deep Q Network with huge capacity but that has a slow learning curve) allows us to gain over the initial Bootstrapped DQN [5]. However, unlike classical reward shaping, we do not have guarantees of convergence. Empirically, adding this reward still improved over Bootstrapped DQN.

A solution in the same vein as ours would have been to design balls for a certain metric around the states where we requested demonstrations, then propagate the information of those demonstrations to other states. However, we argue that our solution actually resembles this hard-coded process while enabling a softer decision model that is not based on an arbitrary metric of similarity but that learns this latter through kernels.

## 6 Conclusion

This project was an opportunity for us to learn a lot about a recent research topic in RL. After having studied several ideas from the literature and having implemented some baselines, we proposed an algorithm to address the problem of actively learning from scarce and possibly sub-optimal demonstrations. Our algorithm is the result of successive iterations and improvements. We chose to restrict demonstrations to state-action pairs, that constitutes a weak and application-independent hypothesis. In future work, some higher-level forms of demonstrations could be considered such as pointing at symmetries or invariance in a specific problem that would reduce the complexity of the learning task, although this signal might be harder to integrate in an active learning framework.

## References

[1] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé. Reinforcement learning from demonstration through shaping. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[2] T. Cederborg, I. Grover, C. L. Isbell Jr, and A. L. Thomaz. Policy shaping with human teachers. In *IJCAI*, pages 3366–3372, 2015.

[3] C. Celemin, J. Ruiz-del Solar, and J. Kober. A fast hybrid reinforcement learning framework with human corrective feedback. *Autonomous Robots*, 43(5):1173–1186, 2019.

[4] J. Chemali and A. Lazaric. Direct policy iteration with demonstrations. In *IJCAI-24th International Joint Conference on Artificial Intelligence*, 2015.

[5] S.-A. Chen, V. Tangkaratt, H.-T. Lin, and M. Sugiyama. Active deep q-learning with demonstration. *Machine Learning*, pages 1–27, 2019.

[6] Y. Gao, H. Xu, J. Lin, F. Yu, S. Levine, and T. Darrell. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018.

[7] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[8] J. Ho and S. Ermon. Generative adversarial imitation learning. *arXiv preprint arXiv:1606.03476*, 2016.

[9] B. Kim, A.-m. Farahmand, J. Pineau, and D. Precup. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems*, pages 2859–2867, 2013.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[11] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. *arXiv preprint arXiv:1602.04621*, 2016.

[12] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[14] K. Subramanian, C. L. Isbell Jr, and A. L. Thomaz. Exploration from demonstration for interactive reinforcement learning. In *Aamas*, pages 447–456, 2016.

[15] Z. Wang and M. E. Taylor. Improving reinforcement learning with confidence-based demonstrations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 3027–3033. International Joint Conferences on Artificial Intelligence Organization.

[16] Z. Wang and M. E. Taylor. Interactive reinforcement learning with dynamic reuse of prior knowledge from human and agent demonstrations. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 3820–3827. International Joint Conferences on Artificial Intelligence Organization.

[17] E. Wiewiora, G. W. Cottrell, and C. Elkan. Principled methods for advising reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 792–799, 2003.

[18] H. Xu, Y. Gao, F. Yu, and T. Darrell. End-to-end learning of driving models from large-scale video datasets. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3530–3538. IEEE.

# Appendices

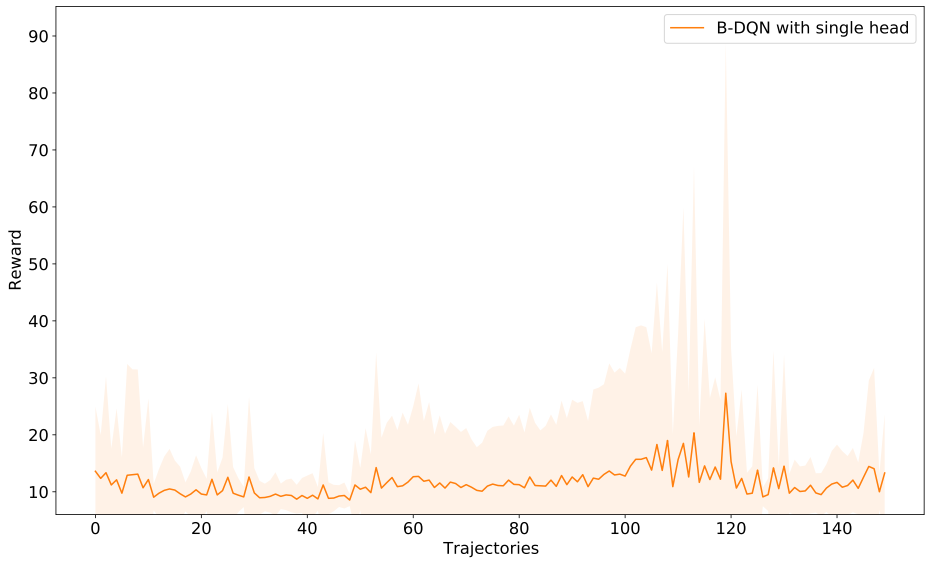


Figure 6: Expected reward when training a Bootstrap DQN with a single head, using a greedy strategy. The lines here correspond to the mean over 20 runs, and the shaded part to the variance.