A Survey of Apprenticeship Learning

Ziqian Bai * ziqian_bai@sfu.ca
Minh Bui * minh_bui_3@sfu.ca
Qiushi Lin * qiushi_lin@sfu.ca
Jiaqi Tan * jiaqit@sfu.ca

Abstract

In this project, we provide a general survey of apprenticeship learning (AL) regarding its theory and learning techniques. The AL aims to learn a new policy from observing the expert in environments without knowledge of reward functions. In this project, we first define the AL problem rigorously and understand it from the game-playing perspective. Based on that, we continue to elaborate on the four most popular AL learning techniques and explain their differences: (I) the projection method [Abbeel and Ng, 2004] (II) the multiplicative weights algorithm [Syed and Schapire, 2007] (III) the Frank-Wolfe algorithm [Zahavy et al., 2020] (IV) the online apprenticeship learning with mirror descent updates [Shani et al., 2022]. Moreover, we implement all these methods over the same benchmarks to compare their performance empirically.

1 Motivation

Recently, reinforcement learning (RL) is becoming more important for many tasks in robotics where model-driven approaches such as control theory are intractable Ge et al. [2012], Chakrabarti [2012], Aghili and Zavlanos [2019]. Typically, the objective of RL is to find a control policy for an agent to follow given a predefined reward function with or without knowledge of the environment.

However, in many real-world settings, it is often very hard to specify a good reward function in order for reinforcement learning algorithms to be applied. Usually, it is more feasible to have a task's demonstration data provided by the experts, and have an algorithm imitate the expert to learn a policy will figure out how to imitate doing the task.

One of such frameworks for imitating is *Apprenticeship Learning* (AL), which aims to learn a policy imitating the expert demonstration, without the need to recover the exact reward function followed by the expert. By learning directly from expert demonstrations, AL can bypass the need for a detailed model of the system, making it more flexible and adaptable to a wide range of tasks Ng and Russell [2000], Ziebart et al. [2008].

This survey reviews four papers in apprenticeship learning. We compare these papers and understand the improvements made over one another. Moreover, we implement the papers and evaluate all of them on several model-free environments to assess their performance and identify the strengths and limitations of each approach. Our goal is to provide a comprehensive and objective assessment of the state of the art in apprenticeship learning, and to identify directions for future work in this field.

2 Related Work

In addition to apprenticeship learning, there are two other approaches to learning a policy by imitating expert demonstrations without the need to recover the expert's exact reward function:

Behavior Cloning (BC) These class of algorithms formulate the imitation learning problem as a standard supervised learning task, in which the input state space (such as images) is mapped to a

*Authors are listed alphabetically. All authors share an equal contribution to the project.

control policy. This approach, which was pioneered by Pomerleau [1988] and further developed by Ross et al. [2011], involves training a machine learning model to predict the expert's actions given a set of input states. While BC algorithms can be effective in some cases Ho et al. [2016], Wang et al. [2017], they suffer from a significant limitation: the resulting control policy may not perform well on input states that are not present in the training dataset, leading to a phenomenon known as generalization error. This limitation can make BC algorithms less robust and less adaptable to new situations.

Inverse Reinforcement Learning (IRL). Rather than simply mimicking the expert's actions, the IRL algorithm attempts to solve the imitation learning problem by first learning the expert's cost function from data and then using reinforcement learning algorithms to derive a control policy Ng and Russell [2000]. This approach allows the learner to adapt to new situations more effectively, as it does not rely on the availability of labeled training data. Additionally, it avoids generalization error as it does not rely on a learned model. However, IRL may not always be able to accurately recover the expert's cost function, which can lead to suboptimal performance Finn et al. [2016]. In contrast, AL does not require the recovery of the expert's cost function, making it a simpler and potentially more effective approach in some cases Abbeel and Ng [2004], Syed et al. [2008].

3 Problem formulation

3.1 Preliminary and Assumptions

Compared to other expert-related reinforcement learning tasks mentioned above, apprenticeship learning aims to solve a special type of infinite-horizon Markov Decision Process without any knowledge of the reward function, MDP\R $M = (\mathcal{S}, \mathcal{A}, \gamma, D, \theta, \phi)$, consisting of finite state sets $\mathcal{S}$ and action sets $\mathcal{A}$, discount factor γ, initial state distribution D, and transition function $\theta(s, a, s\prime) \coloneqq P(s\prime|s, a)$. Instead of the reward function R, we are given a set of the state features $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^k$.

The policy $\pi : S \times A \to \mathbb{R}$ describes the probability of the agent taking the specific action when in a specific state. We further introduce the value function in M as, $V(\pi) \coloneqq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t^* | \pi, \theta, D\right]$, where R^* is the true reward function of the environment. We also define the feature expectation as $\mu(\pi) \coloneqq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t, a_t) | \pi, \theta, D\right]$ w.r.t a certian policy π.

Given an expert policy π_E, the goal of AL is to find an ϵ-optimal policy π^*, meaning $|V(\pi) - V(\pi_E)| \leq \epsilon$. In practice, expert policies may not be queried in an online fashion. Instead, we collect m independent trajectories in M with the length H: $(s_0^i, \cdots, s_H^i)$. We can then calculate the expert's feature expectations as $\mu_E = \frac{1}{m} \sum_{i=0}^{m} \sum_{t=0}^{H} \gamma^t \phi(s_i^t, a^t)$.

We make two basic assumptions : (i) the reward is linear w.r.t the features, meaning $R(s, a) = w \cdot \phi(s, a)$ and (ii) $\|w\|_1 \leq 1$. Under these assumptions, we can say that we only need to find a policy whose feature expectation matches the expert's feature expectation, since

$$|V(\pi) - V(\pi_E)| = |w \cdot \mu(\pi) - w \cdot \mu_E| \leq \|w\|_2 \|\mu(\pi) - \mu_E\|_2 \leq \epsilon. \tag{1}$$

3.2 Apprenticeship Learning as Min-Max Optimization

As we assume the reward function is unknown, AL is required to find a policy that is ϵ-optimal for all possible rewards. That is equivalent to saying that the policy we find should be ϵ-optimal in the "worst" reward function. Another way to interpret this is to consider this as a min-max optimization problem:

$$v^* = \max_{\pi} \min_{R} \left[V(\pi|R) - V(\pi_E|R)\right] \tag{2}$$

The "min player" attempts to find the reward function for the worst-case scenario, whereas the "max player" tries to approximate the best policy under the given reward. As long as the policy can be ϵ-optimal in the worst-case reward, it will be ϵ-optimal for all possible reward functions. It is more clear with this perspective that AL is different from reward-given RL tasks (e.g., imitation learning and inverse reinforcement learning) which are merely minimization problems.

4 Methodology

4.1 Apprenticeship Learning via Inverse Reinforcement Learning

In this work, Abbeel and Ng [2004] introduces the concept of Apprenticeship Learning (AL) via Inverse Reinforcement Learning (IRL), a framework for imitating expert demonstrations using IRL. This approach combines the benefits of both apprenticeship learning and IRL to learn a control policy that can adapt to new situations. It first learns the expert's reward function from data using IRL, and then uses an Markov Decision Processes (MDP) solver to derive a control policy that imitates the expert's actions. The authors also introduced the projection algorithm (Algorithm 1) as a practical method for implementing the AL via IRL framework. This algorithm has been shown to be effective in a variety of settings (e.g. Atari game, car control, etc).

4.1.1 The Projection (PROJ) Algorithm

In order to find a ϵ-optimal policy π^* whose feature expectations is close to the expert expectations as stated in Eq. (1), the authors propose the Projection algorithm as follows:

Algorithm 1 The projection method

1: **Input**: expert's feature expectation μ_E, number of iterations T
2: **Initialize**: choose any $\pi^{(0)}$, set $\bar{\mu}^{(0)} = \mu\left(\pi^{(0)}\right)$
3: **for** $t = 1, \ldots, T$ **do**
4: Set $w^{(t)} = \mu_E - \bar{\mu}^{(t-1)}$
5: If $||\mu_E - \bar{\mu}^{(t-1)}||_2 \leq \epsilon$, then terminate.
6: Compute $\pi^{(t)} = \pi^{(w^{(t)})^*}, \mu^{(t)} = \mu\left(\pi^{(t)}\right)$
7: $\alpha^{(t)} = \frac{\left(\mu^{(t)} - \bar{\mu}^{(t-1)}\right) \cdot \left(\mu_E - \bar{\mu}^{(t-1)}\right)}{\left(\mu^{(t)} - \bar{\mu}^{(t-1)}\right) \cdot \left(\mu^{(t)} - \bar{\mu}^{(t-1)}\right)}$ (This step calculates the projection of μ_E onto the line between $\mu_{(i-2)}$ and $\mu_{(i-1)}$, , as illustrated in Fig. 1.)
8: $\bar{\mu}^{(t)} = \bar{\mu}^{(t-1)} + \alpha^{(t)}\left(\mu^{(t)} - \bar{\mu}^{(t-1)}\right)$
9: **end for**
10: **return** $\pi^{(T)}$

In the first iteration, the initial mixed policy's feature expectation $\bar{\mu}^{(0)}$ is set to $\mu^{(0)}$ (line 2), the initial weight $w^{(1)}$ is set to $\mu_E - \mu^{(0)}$ (line 4), and the policy π_1 calculated by the reinforcement learning algorithms (e.g. policy gradient Sammut et al. [1992], value iteration Bellman [1957], etc) is based on the reward $R = w^{(1)} \cdot \phi$ (line 6). Then the Projection algorithm operates by iteratively projecting the learned reward function onto the space of feasible reward functions (line 7), and then using reinforcement learning algorithms, value iteration Bellman [1957], etc) to learn a control policy that maximizes the expected reward under the projected reward function (line 6). This process is repeated until convergence (line 5), resulting in a control policy that imitates the expert's actions (line 10).

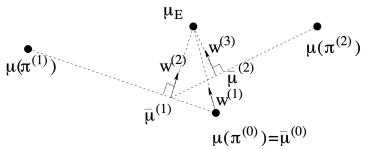

Figure 1: Visualization of the first three iterations of the max-margin algorithm. (Image credit: Abbeel and Ng [2004].)

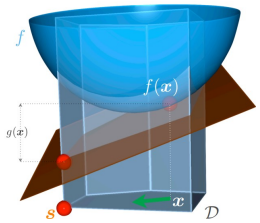

Figure 2: FW algorithm directly searches for a point s in $\mathcal{D}$ that correlates the most with $-\nabla f(x)$. (Image credit: Jaggi [2013])

Main Theoretical Results: As the first work in apprenticeship learning, Abbeel and Ng [2004] proved that the algorithm can terminate with $\mathcal{O}(\frac{k}{(\epsilon(1-\gamma))^2}\log\frac{k}{\epsilon(1-\gamma)})$ iterations, where k is the dimensionality of the feature space and γ is the discount factor. The sample complexity is given

by the formula $\mathcal{O}(\frac{2k}{(\epsilon(1-\gamma))^2}\log\frac{2k}{\delta})$, where ϵ is the distance and δ is the confidence level. Detailed proofs for these statements can be found in the original paper.

4.2 Apprenticeship Learning via Frank-Wolfe

4.2.1 The Frank-Wolfe (FW) algorithm

Typically to minimize a convex function over a convex set $\mathcal{D}$, we use projected gradient descent (PGD) algorithm. PGD first computes the derivative $\nabla f(x_t)$, and then uses that for gradient descent update $z_{t+1} = x_t - \eta\nabla f(x_t)$. We then project this point back to the constrained set to get a feasible solution $x_{t+1} = \text{Proj}_{\mathcal{D}}[x_t - \eta\nabla f(x_t)]$, which can be a very expensive operation for many convex sets and distance metric .

The conditional gradient (CG) method (proposed by Frank and Wolfe in 1956), on the other hand, avoids this projecting operation in their steps. The algorithm goes as follow:

Algorithm 2 Frank-Wolfe algorithm

1: **Input**: A convex set $\mathcal{D}$, a convex function $f(x)$, learning rate γ_t
2: **Initialize**: $x_0 \in \mathcal{D}$
3: **for** $t = 1, \ldots, T$ **do**
4: $\quad s_t = \arg\max_{s\in\mathcal{D}}\langle -\nabla f(x_{t-1}), s\rangle$
5: $\quad$ Set $d_t = s_t - x_{t-1}$
6: $\quad$ Update $x_t = x_{t-1} + \gamma_t d_t$
7: **end for**
8: **return** x_t

Similar to PGD, FW first computes gradient $\nabla f(x_{t-1})$. But instead of using this $\nabla f(x_t)$ for gradient descent update, which can lead the iterate to be outside of $\mathcal{D}$, FW solves a linear optimization problem (line 4 of Algorithm 2). This linear optimization directly search for a point s in $\mathcal{D}$ that correlates the most with the direction of negative gradient $-\nabla f(x_{t-1})$, which minimize the function f. In another words, FW algorithm does not use $\nabla f(x_{t-1})$ to update x_t but rather use it as a guidance to search for a point that minimize the function while staying inside the feasible set $\mathcal{D}$ as illustrated in Figure 2. By that, FW replaces projection operation with solving a linear optimization problem, which can be more computationally desirable in many problems.

Once s_t has been found, the direction vector $d_t = s_t - x_{t-1}$ is then used to update x_t (line 6 of Algorithm 2) using step size γ_t that can be set using linesearch algorithm such as backtracking or exact linesearch. This update is also equivalent to $x_t = (1-\gamma_t)x_{t-1} + \gamma_t s_t$ by substituting d_t, which is a convex combination, and hence x_t is guaranteed to stay inside $\mathcal{D}$.

Theorem 1. *Let $f(x)$ be a convex and β-smooth function. Also let $diam_{\mathcal{D}}$ be the diameter of the set $\mathcal{D}$. Then using a step-size $\gamma_t = \frac{2}{t+1}$, Algorithm 2 will computes x_t such that*

$$f(x_t) - f(x^*) \leq \frac{2\beta(diam_{\mathcal{D}}^2)}{t+1}, \quad \textit{for } t \geq 2$$

where x^* is the minimizer of $f(x)$ over $\mathcal{D}$.

The proof of this theorem is available in Jaggi [2013]. In the next section, we will explain how FW algorithm can be applied in the AL context and how the above theorem is relevant to analysing the convergence rate of solving our AL problem.

4.2.2 FW in Solving Apprenticeship Learning

The Projected Algorithm, presented in the previous section, is shown to be an instance of FW algorithm in Zahavy et al. [2020]. To see that, first let's denote the following variable correspondence $x_t = \bar{\mu}_t, s_t = \mu_t$ in the AL context. In the AL context, the feasible set $\mathcal{D}$ is a polytope formed by vertices, each of which is a feature expectation of a policy. Next let's define the function$f(x)$ as follow: $f(x) = \frac{1}{2}||x - \mu_E||^2$, which measures the square distance between current policy's feature expectations and that of the expert. Line 4 of Algorithm 1 is then equivalent to setting the weight

to be negative gradient $w = -\nabla f(x_{t-1}) = \mu_E - \mu_{t-1}$. The step size for each iteration is found through linesearch $\gamma_t = \min_{\gamma_t} f(x + \gamma_t d) = \min_{\gamma_t} ||\bar{\mu}^{(t-1)} + \gamma_t(\mu^{(t)} - \bar{\mu}^{(t-1)}) - \mu_E||^2$, whose solution is equivalent to line 7 of Algorithm 1.

Next we apply theorem 1 to obtain the convergence rate of FW in solving AL. Since $h(x)$ is a 1-smooth, 1-strongly convex function and $diam_{\mathcal{D}} = \sqrt{k}/(1-\gamma)$ Zahavy et al. [2020] (note that γ here means the discount rate), FW with AL will converge at a rate $O(\frac{k}{T(1-\gamma)^2})$. This means that after $O(\frac{k}{\epsilon^2(1-\gamma)^2})$ iterations, the method will find an $\epsilon-$ optimal solution, which is a logarithmic improvement over the projected algorithm.

4.3 Multiplicavite Weights for Apprenticeship Learning

Syed and Schapire [2007] provides a game-theoretic view of the AL problem, which naturally leads to a direct application of the multiplicative-weights algorithm [Freund and Schapire, 1999] for apprenticeship learning (MWAL). This algorithm is computationally faster, but requires the state/action spaces to be relatively small.

Algorithm 3 Multiplicative Weights Algorithm

1: **Input**: expert's feature expectation μ_E, number of iterations T
2: **Initialize**: $W^{(1)}(i) = 1$ for $i = 1, \cdots k$
3: Set $\beta = \left(1 + \sqrt{\frac{2\log(k)}{T}}\right)^{-1}$
4: **for** $t = 1, \ldots, T$ **do**
5: Set $w^{(t)}(i) = \frac{W^{(t)}(i)}{\sum_i W^{(t)}(i)}$
6: Compute $\pi^{(t)} = \pi^{(w^{(t)})^*}$ with respect to $R = w^{(t)} \cdot \phi(s)$
7: Compute an estimate of feature expectation $\mu^{(t)} = \mu\left(\pi^{(t)}\right)$
8: Set $G^t(i) = \frac{(1-\gamma)(\mu^{(t)}(i) - \mu_E(i))}{4}$ for $i = 1, \cdots k$
9: $W^{(t+1)}(i) = W^{(t)}(i) \cdot \beta^{G^t(i)}$ for $i = 1, \cdots k$
10: **end for**
11: **return** $\pi^{(T)}$

4.3.1 Apprenticeship Learning as Two-Player Zero-Sum Games

Syed and Schapire [2007] first models this problem to be a two-play zero-sum game. MWAL first denotes a set of all possible deterministic stationary policies Π in M, whose dimension is $\mathcal{O}(|S|^{|A|})$. Then, it further defines thew game matrix $G \in k \times |\Pi|$ as: $G(i,j) = \mu^j(i) - \mu_E(i)$, where μ^j is the feature expectation of the j-th possible deterministic policy. As the optimal policy can be represented as a combination of mixed deterministic policies, finding the optimal policy becomes assigning different weights to different columns of G. Under the linear reward assumption, by rewriting Eq. 2, we have:

$$v^* = \max_{\psi} \min_{w} [w \cdot \mu(\psi) - w \cdot \mu_E] = \max_{\psi} \min_{w} w^T G \psi \tag{3}$$

where ψ represents a set of mixing weights for all policies in Π. Therefore, the AL is formulated as solving a two-player zero-sum game, although the game matrix is extremely large.

4.3.2 Multiplicavite Weights for Apprenticeship Learning (MWAL)

Freund and Schapire [1999] described a multiplicative weights algorithm for finding approximately optimal strategies in games with large or even unknown game matrices. Algorithm 3 describes a direct adoption of the multiplicative weights algorithm to solve the game mentioned above. Line 6 involves solving MDP under the known reward function, for which a huge set of RL algorithms can be applied (e.g., value iteration and policy iteration). Line 7 involves computing an estimate of feature expectations given a certain policy, which is typically done by sampling plenty of trajectories. Importantly, the complexity of both steps scales with the size of the MDP\R, and not with the size of the game matrix G. Line 9 shows the exponential gradient descent step.

Main Theoretical Results: Table 3 shows the comparison of convergence rate and sample complexity of all offline apprenticeship learning methods. Compared to the PROJ and the FW, MWAL improves the convergence rate and the sample complexity to $\mathcal{O}(\log k)$, as proven in Freund and Schapire [1999]. However, the drawback of this algorithm is also obvious that G cannot be too large meaning that the state and action space have to be relatively small. However, in practice, we can limit the dimension of G by only considering a small set of deterministic policies, which has been empirically proven to be effective.

4.4 Online Apprenticeship Learning (OAL)

Mirror Descent (MD) Policy Update: Originally, Mirror Descent (MD) is an iterative framework designed to solve convex optimization problems. The core idea is to minimize the sum of 2 terms at each iteration: (1) a linear approximation of the objective function; (2) a regularization that keeps the updated estimates close to the current one. Formally, we have

$$x_{k+1} \in \underset{x \in C}{\operatorname{argmin}} \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{t_k} B_\psi(x, x_k), \tag{4}$$

where B_ψ is the Bregman divergence defined as $B_\psi(x, x_k) := \psi(x) - \psi(x_k) - \langle \nabla \psi(x_k), x - x_k \rangle$ for regularization, with ψ being the strongly convex potential function and t_k being the step-size. Recent works Shani et al. [2020], Tomar et al. [2021] adapt MD to RL, leading to MD-style policy update steps (referred as "MD policy update" in this report) for solving MDPs with theoretical guarantees.

OAL and Differences to Prior AL Methods: One common step used in prior AL methods is "solve the MDP (i.e. compute the optimal policy) given the current reward or cost per iteration" (i.e. line 6 in Algorithm 1, line 4 in Algorithm 2, line 6 in Algorithm 3). However, "MDP solving" is a challenging problem by itself, and typically relies on complex algorithms running for lots of iterations, which makes prior AL methods computationally expensive and less practical Shani et al. [2022].

To this end, Shani et al. [2022] propose Online Apprenticeship Learning (OAL). The core idea is to replace the "MDP solving" step with a single Mirror-Descent (MD) policy update. The reward / cost update is also replaced by MD accordingly. Since one MD policy update is much cheaper than solving an MDP (e.g. solving an MDP takes numerous MD policy updates), the computational cost of each iteration can be vastly reduced. Discussions regarding the total computational cost can be found at the theoretical results paragraph and experiments Sec. 5. Formally, after plugging in the MD formulation, we have Algorithm 4 Shani et al. [2022].

Algorithm 4 Online Apprenticeship Learning

1: **for** $k = 1, ..., K$ **do**
2: Rollout a trajectory by acting π_k
3: # Evaluation Step
4: Evaluate Q^{π^k} using the current cost c^k. Evaluate $\nabla_c L(\pi^k, c; \pi^E)|_{c=c^k}$
5: # Policy Update
6: Update π^{k+1} by an MD policy update with Q^{π^k}:
7: $\pi_h^{k+1} \in \operatorname{argmin}_\pi \langle Q_h^{\pi_k, c_k}, \pi_h \rangle + t_k^\pi d_{KL}(\pi_h || \pi_h^k)$.
8: # Costs Update
9: Update c^{k+1} by an MD step on $\nabla_c L(\pi^k, c)|_{c=c^k}$:
10: $c^{k+1} \in \operatorname{argmin}_c \langle \nabla_c L(\pi^k, c)|_{c^k}, c \rangle + \frac{t_k^c}{2} \left\| c - c^k \right\|^2$.
11: **end for**

Main Theoretical Results: Shani et al. [2022] define the AL regret similar to Eq. (2) as $\text{Reg}_{AL}(K) := \max_{c \in \mathcal{C}} \sum_{k=1}^K \left[V(\pi_k | c) - V(\pi^E | c) \right]$ but with a finite-horizon MDP and tabular costs, and derive an $O\left(\sqrt{H^4 S^2 A K} + \sqrt{H^3 S A K^2 / N} \right)$ regret bound. The first $O(\sqrt{K})$ term is similar to the optimal regret of solving an MDP. Although it may not lead to a clear conclusion, this gives an encouraging guess that the *total computational cost* of OAL may be connected to the *one iteration cost* of prior AL methods. The second term is a statistical error term depends on the amount of expert data N we have. Importantly, the theoretical results of OAL cannot be directly compared against other discussed

works due to assumption differences. For example, OAL assumes tabular costs while other methods assume linear costs based on state-action features.

5 Experiments

Despite that theoretical analysis and comparisons have been discussed in Sec. 4, there are still factors that prevent a direct comparison on the actual performance of the surveyed methods, such as the gaps between assumptions and realities, and the assumption differences between methods. Although empirical results are given in corresponding papers (e.g. Abbeel and Ng [2004], Zahavy et al. [2020], Syed and Schapire [2007], Shani et al. [2022]), these experiments are usually conducted on inconsistent tasks for different approaches. In addition, some old school methods such as Abbeel and Ng [2004], Syed and Schapire [2007] did not leverage recent deep Reinforcement Learning (RL) techniques in their experiments, making their results less comparable to more recent approaches like Shani et al. [2022]. To this end, we design experiments to compare the surveyed methods in a more consistent manner. We consider 3 methods to implement and test: PROJ (Abbeel and Ng [2004]), MWAL (Syed and Schapire [2007]), and OAL (Shani et al. [2022]), on 2 classical control tasks: Pendulum (Sec. 5.1) and Mountain Car Continuous (Sec. 5.2). We merge FW (Zahavy et al. [2020]) into PROJ since they are equivalent.

Experiment Setup: We use feature-based linear rewards in all experiments. To ensure consistency among methods, we use Mirror-Descent (MD) policy updates for both (1) each OAL iteration, and (2) the MDP solving step in other methods. We use Mirror Descent Policy Optimization (MDPO) Tomar et al. [2021], a deep RL approach, to implement the MD policy updates. Therefore, the policy is parameterized by a neural network. All implementations are based on Tensorflow and simulators provided by OpenAI Gym Brockman et al. [2016]. The expert is obtained by learning a policy via Soft Actor-Critic (SAC) Haarnoja et al. [2018] on the true rewards given by the simulators. Then, we sample 2k episodes as the expert data.

Metric: We use the mean true rewards of 100 episodes as the performance measurement of a method. The true reward of each episode is obtained by summing up the true reward of every timestep. Since we use MD policy updates for all methods and the reward updates are cheap in general, we use the number of policy updates to approximate the computational cost. Thus, we can visually compare all methods via the plots of "true reward" vs. "number of policy updates". Note that PROJ (Abbeel and Ng [2004]) and MWAL (Syed and Schapire [2007]) require to solve an MDP on each iteration, leading to a inner loop with multiple policy updates. We only compute and plot the true reward for each outer loop iteration, since the inner loops may not directly optimize the function of interest.

5.1 Task: Pendulum

Task Description: As shown in Fig. 5, we have a pendulum with one end attached to a fixed position. The other end is free, and can be rotated by applying torque on it. Given a random starting state with the free end pointing downward (i.e. left side of Fig. 5), the goal is to swing it into an upright position with zero velocity (i.e. right side of Fig. 5). The definitions of states and actions are given in Table 1. The true reward is defined as $R_{gt}(s, a) = -(\theta^2 + 0.1s[2]^2 + 0.001a^2)$ with $s[0] = \cos(\theta)$ and $s[1] = \sin(\theta)$, where θ is the angle of the pendulum between $[-\pi, \pi]$ with 0 corresponding to the upright position. Thus, we have the maximum reward $R^*_{gt} = 0$ when the pendulum perfectly stops at the upright position (i.e. target state).

Feature Definition: For features, we directly define it to be the concatenated states and actions normalized into $[0, 1]$, i.e. $\phi(s, a) = (\hat{s}, \hat{a}) \in [0, 1]^4$.

Results: As shown in Fig. 3, all methods successfully converge at the end and achieve comparable performances to the expert. Although the OAL (Shani et al. [2022]) performance quickly goes up at the beginning, it does not fully reach the expert-level performance but has a minor gap. Although OAL still reaches the expert performance at the end, it taks slightly more policy updates than MWAL (Syed and Schapire [2007]). PROJ (Abbeel and Ng [2004]) follows a similar trend as MWAL, but with more small oscillations around 10k to 20k policy updates. Overall, we consider all methods have roughly similar performances on this Pendulum task. We suspect that the simplicity of Pendulum could make it hard to clearly show the pros and cons of each method.

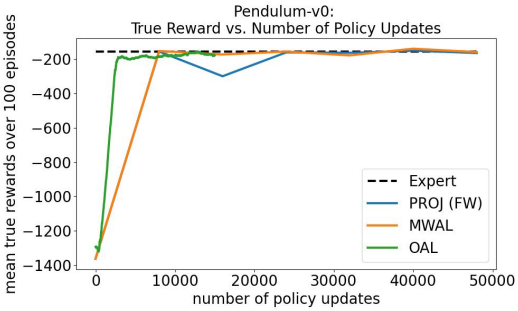

Figure 3: Example starting and target states.

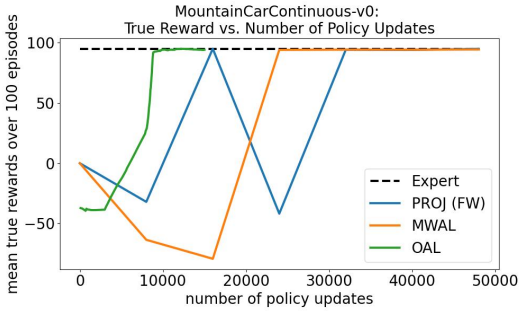

Figure 4: Example starting and target states.

5.2 Task: Mountain Car Continuous

Task Description: As shown in Fig. 6, we have a 1-D valley with the goal (i.e. the flag) at the top of the hill on the right side. At the starting state, a car is randomly put at the bottom of the valley. The objective is to push the car up the hill and reach the goal position. Note that the force is not large enough to directly push the car to the top of the hill. We need to "swing" the car a bit to get enough velocity. The definitions of states and actions are given in Table 2. The true reward is defined as $R_{gt}(s, a) = -0.1a^2 + [100 \text{ if } s[0] \geq 0.6 \text{ (i.e. reach the goal) else } 0]$.

Feature Definition: We first tried the same feature definition as in Pendulum Sec. 5.1, i.e. $\phi(s, a) = (\hat{s}, \hat{a}) \in [0, 1]^3$ where $\hat{s}$ and $\hat{a}$ are states and actions normalized to $[0, 1]$. However, we found that PROJ (Abbeel and Ng [2004]) and MWAL (Syed and Schapire [2007]) failed to learn reasonable policies using this naive definition. Thus, we design a more sophisticated feature for PROJ and MWAL. More specifically, we uniformly split the 2-D state space into 3×3 blocks. We then duplicate the normalized states $\hat{s}$ and actions $\hat{a}$ by 9 times, with each of the replica associated with one block, denoted as $\hat{s}_i$, $\hat{a}_i$ for $i = 1, 2, ..., 9$. When the value of $\hat{s}$ falls in block j, we keep $\hat{s}_j$, $\hat{a}_j$ as their original values, and set all other $\hat{s}_i$, $\hat{a}_i$ to zeros. In this way, we can enable a piece-wise linear reward function, with each block has its own reward weights applied on $(\hat{s}, \hat{a})$. Formally, we have the feature $\hat{\phi}(s, a) = (\hat{s}_1, \hat{a}_1, \hat{s}_2, \hat{a}_2, ..., \hat{s}_9, \hat{a}_9) \in [0, 1]^{27}$ with $(\hat{s}_i, \hat{a}_i) = (\hat{s}, \hat{a})$ if $\hat{s}$ falls in block i otherwise 0.

Results: As shown in Fig. 4, all methods successfully converge at the end and achieve comparable performances to the expert. Note that even PROJ (Abbeel and Ng [2004]) and MWAL (Syed and Schapire [2007]) take only ≤ 4 (outer loop) iterations to converge, they still need more policy updates comparing to OAL (Shani et al. [2022]) due to the need of solving an MDP per (outer loop) iteration. This empirically verifies the OAL's claimed advantage on "reducing the computational complexity of the algorithm" via "averting the need to solve an MDP in each iteration" Shani et al. [2022]. Moreover, OAL works on the naive feature definition $\phi(s, a)$, while PROJ and MWAL don't, indicating that OAL may potentially be less limited by the feature complexity. Also, MWAL takes slightly less iterations than PROJ to stably converge, which is consistent to the theoretical results in Sec. 4.1 and Sec. 4.3.

6 Conclusion and Future Work

In this project, we focus on surveying apprenticeship learning, a specific RL task that aims at learning a policy from an expert in environments without any knowledge of the reward functions. We first define the setting and objective of AL rigorously and then interpret it as a min-max optimization problem. Based on that, we mainly study several major methods that have been proposed in this area. We elaborate on the techniques applied to these methods and their advantages and disadvantages. Empirically, we implement all these methods on the same benchmark to compare their performance and showcase the results.

We can conclude from the experiment part that OAL has the strongest performance. For future work, we could replace the mirror descent with its different variants (e.g., Nesterov's dual averaging or Mirror Prox), to see whether they can improve the performance even better.

References

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

MH Aghili and MJ Zavlanos. Model-free learning and control in robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:169–184, 2019.

Richard Bellman. A dynamic programming approach to the optimal allocation of resources. *Journal of the Operations Research Society of America*, 5(1):61–63, 1957.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Sujit S Chakrabarti. *Model-free control for robotics*. Springer, 2012.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Guided cost learning: Deep inverse optimal control via policy optimization. *arXiv preprint arXiv:1603.00448*, 2016.

Yoav Freund and Robert E Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, 1999.

SS Ge, J Hu, and X Liu. Model-free control for robot manipulators: A survey. *IEEE Transactions on Industrial Electronics*, 59(10):4145–4155, 2012.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

J Ho, E Lee, and S Ermon. Generative adversarial imitation learning. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 2852–2861, 2016.

Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 427–435, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `https://proceedings.mlr.press/v28/jaggi13.html`.

Andrew Y Ng and Stuart Russell. Algorithms for inverse reinforcement learning. *ICML*, 1:657–664, 2000.

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *AISTATS*, volume 15 of *JMLR Proceedings*, pages 627–635. JMLR.org, 2011. URL `http://dblp.uni-trier.de/db/journals/jmlr/jmlrp15.html#RossGB11`.

Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. In *Machine Learning Proceedings 1992*, pages 385–393. Elsevier, 1992.

Lior Shani, Yonathan Efroni, and Shie Mannor. Adaptive trust region policy optimization: Global convergence and faster rates for regularized mdps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5668–5675, 2020.

Lior Shani, Tom Zahavy, and Shie Mannor. Online apprenticeship learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8240–8248, 2022.

Umar Syed and Robert E Schapire. A game-theoretic approach to apprenticeship learning. *Advances in neural information processing systems*, 20, 2007.

Umar Syed, Michael Bowling, and Robert E Schapire. Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning*, pages 1032–1039, 2008.

Manan Tomar, Lior Shani, Yonathan Efroni, and Mohammad Ghavamzadeh. Mirror descent policy optimization. In *International Conference on Learning Representations*, 2021.

Z Wang, K Wahlström, and B Schölkopf. Imitation learning for robust visuomotor control. *arXiv preprint arXiv:1703.07192*, 2017.

Tom Zahavy, Alon Cohen, Haim Kaplan, and Yishay Mansour. Apprenticeship learning via frank-wolfe. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6720–6728, 2020.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. *AAAI*, 8:1433–1438, 2008.

A Author Contributions

- Ziqian Bai: survey the OAL Shani et al. [2022], and take part in all the implementation
- Minh Bui: survey the FW Zahavy et al. [2020], and take part in all the implementation
- Qiushi Lin: survey the MWAL Syed and Schapire [2007], and take part in all the implementation
- Jiaqi Tan: survey the PROJ [Abbeel and Ng, 2004], and take part in all the implementation

B Frank-Wolfe Away step

Frank-Wolfe with away step is a variation of the original FW algorithm. The idea is that, the iterate x_t can always be represented as a sparse convex combination of a subset of vertices of $\mathcal{D}$ (Caratheodory theorem) as $x_t = \sum_{j=1}^{l_t} \alpha_{y_j} y_j$. Using this information, the algorithm reduces weight from y_j that are not good representing the minimizer and hence take the "away-step". At every time step, the algorithm will maintain a list of vertices representation $S^{(t)} = \{y_1, \ldots, y_{l_t}\}$ with $l_t = |S^{(t)}|$. In addition, it also maintain a corresponding list of coefficients $\{\alpha_{y_j}\}_{j=1}^{l_t}$ such that $x_t = \sum_{j=1}^{\ell_t} \alpha_{y_j} y_j$.

Algorithm 5 Frank-Wolfe with Away steps (FWAS)

Input: A convex set $\mathcal{D}$, a convex function $f(x)$, learning rate γ_t
Initialize: $x_0 \in \mathcal{D}$
for $t = 1, \ldots, T$ **do**
  $y_t = \arg\max_{s \in \mathcal{D}} \langle -\nabla f(x_t), s \rangle$
  $d^{FW} = y_t - x_t$
  $z_t = \arg\max_{z \in \mathcal{S}^{(k)}} \langle \nabla f(x_t), z \rangle$
  $d^{AS} = x_t - z_t$
  if $\langle \nabla h(x_t), d^{FW} \rangle < \langle \nabla h(x_t), d^{AS} \rangle$ **then**
    Frank-Wolfe step: $d = d^{FW}$, $\gamma_{\max} = 1$
  else
    Away step: $d = d^{AS}$, $\gamma_{\max} = \alpha_{z_t} / (1 - \alpha_{z_t})$
  end if
  Line-search: $\gamma_t = \arg\min_{\gamma \in [0, \gamma_{\max}]} h(x_t + \gamma d)$
  Update: $x_{t+1} = x_t + \gamma_t d$
  Update: $S^{(t+1)}$ and **update** $\alpha_y, \forall y \in S^{(t)}$
end for
return x_t

C Specification of Benchmark

Table 1: Specification of task Pendulum.

	Description	Type	Min	Max
Action	torque	float	-2.0	2.0
State	free end x coordinate	float	-1.0	1.0
	free end y coordinate	float	-1.0	1.0
	angular velocity	float	-8.0	8.0

Table 2: Specification of task Mountain Car Continuous.

	Description	Type	Min	Max
Action	directional force	float	-1.0	1.0
State	car x coordinate	float	-1.2	0.6
	car velocity	float	-0.07	0.07

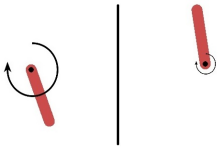

Figure 5: Example starting and target states of Pendulum.

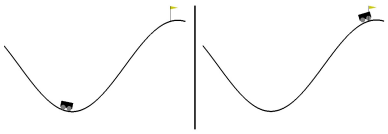

Figure 6: Example starting and target states of Mountain Car Continuous.

D Convergence Rate and Sample Complexity

Table 3: Comparison of Convergence Rate and Sample Complexity for Offline Learning

	Convergence Rate	Sample Complexity
PROJ Abbeel and Ng [2004]	$\mathcal{O}(\frac{k}{(\epsilon(1-\gamma))^2}\log\frac{k}{\epsilon(1-\gamma)})$	$\mathcal{O}(\frac{2k}{(\epsilon(1-\gamma))^2}\log\frac{2k}{\delta})$
FWAS Zahavy et al. [2020]	$\mathcal{O}(\frac{k}{(\epsilon(1-\gamma))^2})$	$\mathcal{O}(\frac{2k}{(\epsilon(1-\gamma))^2}\log\frac{2k}{\delta})$
MWAL Syed and Schapire [2007]	$\mathcal{O}(\frac{\log(k)}{(\epsilon(1-\gamma))^2})$	$\mathcal{O}(\frac{2}{(\epsilon(1-\gamma))^2}\log\frac{2k}{\delta})$