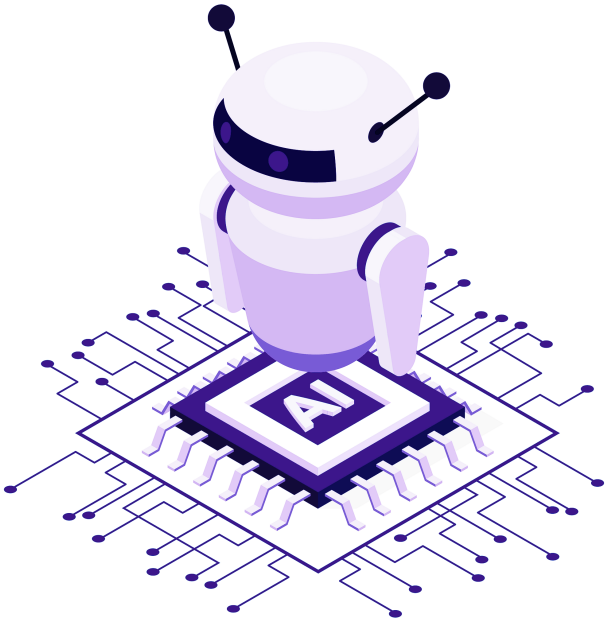

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 11:

Imitation Learning and Inverse RL

Designed By:

Amirreza Velae
amirrezavelae@gmail.com

Amirhossein Asadi
amirhossein.asadi1681@gmail.com

Milad Hosseini
miladhoseini532@gmail.com

RIML

Spring 2025

Preface

One of the central themes in reinforcement learning (RL) is how different learning paradigms approach the problem of acquiring optimal behavior in complex environments. This assignment focuses on comparing reinforcement learning algorithms like Proximal Policy Optimization (PPO) and Advantage Actor-Critic (A2C) with imitation learning methods such as Dataset Aggregation (DAgger) and Generative Adversarial Imitation Learning (GAIL).

Unlike supervised learning, where agents are trained on labeled data, or unsupervised learning, where structure is inferred from data distributions, reinforcement learning agents must learn from trial and error by interacting with an environment. This process of interaction makes learning both powerful and challenging—rewards are often sparse, environments are uncertain, and agents must balance exploration with exploitation. In contrast, imitation learning bypasses some of these challenges by learning directly from expert demonstrations, making it a promising alternative when expert data is available.

This assignment encourages a practical and conceptual understanding of both paradigms. By training PPO and A2C agents directly from environmental feedback, and comparing them to DAgger and GAIL agents that learn from expert policies, we gain valuable insight into the strengths, limitations, and use cases of each approach. This side-by-side analysis allows us to appreciate how reinforcement and imitation learning differ not only in algorithmic design but also in data efficiency, stability, and generalization capability.

The experiments in this homework illustrate how learning from experience and learning from demonstration can complement each other. They also underscore the importance of careful evaluation when choosing between direct environment interaction and expert-driven imitation, especially in tasks where exploration is either expensive or unreliable.

Grading

The grading will be based on the following criteria, with a total of 200 points:

Task	**Points**
Task 1	40
Task 2	60
Task 3	100
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1	0
Bonus 2	0
Bonus 3	15

Submission

The deadline for this homework is 1404/04/13 (July 4th 2025) at 11:59 PM.

Please submit your work by following the instructions below:

- Place your solution alongside the Jupyter notebook(s).
  - Your written solution must be a single PDF file named `HW11_Solution.pdf`.
  - If there is more than one Jupyter notebook, put them in a folder named `Notebooks`.
- Zip all the files together with the following naming format:

  `DRL_HW11_[StudentNumber]_[FullName].zip`

  - Replace `[FullName]` and `[StudentNumber]` with your full name and student number, respectively. Your `[FullName]` must be in CamelCase with no spaces.
- Submit the zip file through Quera in the appropriate section.
- We provided this LaTeX template for writing your homework solution. There is a 5-point bonus for writing your solution in LaTeX using this template and including your LaTeX source code in your submission, named `HW11_Solution.zip`.
- If you have any questions about this homework, please ask them in the Homework section of our Telegram Group.
- If you are using any references to write your answers, consulting anyone, or using AI, please mention them in the appropriate section. In general, you must adhere to all the rules mentioned here and here by registering for this course.

Keep up the great work and best of luck with your submission!

Contents

1 Distribution Shift and Performance Bounds

1.1 Problem Setup

In Imitation Learning, the goal is to learn a policy π_θ that imitates an expert policy π^* based on observed expert demonstrations. Consider a discrete MDP with finite horizon T, and assume we have access to a set of trajectories generated by executing π^*. The learned policy π_θ is trained such that:

$$\mathbb{E}_{p_{\pi^*}(s)}\left[\pi_\theta(a \neq \pi^*(s) \mid s)\right] = \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{p_{\pi^*}(s_t)}\left[\pi_\theta(a_t \neq \pi^*(s_t) \mid s_t)\right] \leq \varepsilon,$$

where ε is a small error term bounding the disagreement between the learned policy and the expert on the expert's state distribution. The symbol $p_\pi(s_t)$ denotes the state distribution at time t when following policy π.

However, this formulation suffers from a covariate shift: the learned policy π_θ may induce state distributions significantly different from those observed under π^*, potentially compounding small errors over time.

1.2 Distributional Shift and Performance

We aim to quantify how this imitation error affects the divergence between state distributions of the expert and the learner, and how it impacts the expected return. The expected return of a policy π under a bounded reward function $|r(s_t)| \leq R_{\max}$ is defined as:

$$J(\pi) = \sum_{t=1}^{T}\mathbb{E}_{p_\pi(s_t)}[r(s_t)].$$

1.3 Tasks

1. **Distribution Shift Bound** Show that the total variation distance between state distributions induced by the learned policy and the expert satisfies:

$$\sum_{s_t}|p_{\pi_\theta}(s_t) - p_{\pi^*}(s_t)| \leq 2T\varepsilon.$$

   Hint: Express the imitation error as an expectation and apply the union bound across all time steps.

2. **Return Gap for Terminal Rewards** Assume that the reward is only received at the final step (i.e., $r(s_t) = 0$ for all $t < T$). Show that:

$$J(\pi^*) - J(\pi_\theta) = \mathcal{O}(T\varepsilon).$$

3. **Return Gap for General Rewards** For a general reward function (i.e., $r(s_t) \neq 0$ for arbitrary t), show that:

$$J(\pi^*) - J(\pi_\theta) = \mathcal{O}(T^2\varepsilon).$$

1.4 Deliverables

- Detailed derivations for each task, including clear explanations of how the bounds are obtained.
- Discussion on the implications of these bounds in the context of Imitation Learning.
- (Optional) Comparison of these theoretical guarantees with alternative methods such as Behavioral Cloning and DAgger.

2 Training PPO, A2C, and DAgger

In this exercise, we will implement and compare several on-policy reinforcement learning and imitation learning approaches in the classic `CartPole-v1` environment. The algorithms to be used are:

1) **Proximal Policy Optimization (PPO)**: a stable, on-policy RL algorithm.
2) **Advantage Actor-Critic (A2C)**: an on-policy RL algorithm known for simplicity and efficiency.
3) **DAgger (Dataset Aggregation)**: an imitation learning algorithm that iteratively refines a policy using expert guidance.

2.1 Why Imitation?

In many real–world control problems we can *show* an agent how to act but cannot easily write down a reward function that captures the desired behaviour. Imitation Learning (IL) aims to recover a policy that *behaves like an expert* directly from demonstration data, bypassing explicit reward engineering. Classical behavioural cloning tackles IL as supervised learning on state–action pairs, yet suffers from compounding error once the learner drifts away from the states the expert visited.[1]

2.2 Main Objectives

1. Train an expert policy using PPO.
2. Use the PPO expert to train a new agent via DAgger.
3. Train a separate agent using A2C directly.
4. Compare performance between PPO, DAgger, and A2C.

2.3 DAgger Algorithm Overview

Below is the full pseudocode for the DAgger algorithm (adapted from Ross *et al.*, 2011). Following the pseudocode, we provide a detailed, code-agnostic explanation of how each step of the algorithm is realized in our experimental pipeline for `CartPole-v1`.

[1] See Ross & Bagnell's DAGGER algorithm for a remedy based on dataset aggregation.

Algorithm 1 DAgger Algorithm

Require: expert policy π^*, mixture coefficients $\{\beta_i\}_{i=1}^N$, number of iterations N, rollout length T
Ensure: learned policy $\hat{\pi}$

Initialize dataset $\mathcal{D} \leftarrow \varnothing$
Initialize $\hat{\pi}_1$ to any policy in Π (e.g., random or pretrained)
for $i = 1$ to N **do**
    Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$
    Sample T-step trajectories in the environment using π_i
    Collect dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states s and expert actions $\pi^*(s)$
    Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
    Train a new classifier (policy) $\hat{\pi}_{i+1}$ on $\mathcal{D}$
end for
return best policy $\hat{\pi}_i$ (by validation performance)

2.3.1 Explanation and End-to-End Workflow

The following describes, in a code-agnostic manner, how each step of Algorithm 1 is implemented in our CartPole-v1 experiment. It unifies the high-level rationale with a concise sequence of operations.

1. **Expert Policy Preparation.** Train a PPO agent until convergence to obtain an expert policy π^*. Save π^* so that for any state s, querying $\pi^*(s)$ yields the "ground-truth" action. This expert serves as an oracle throughout DAgger.

2. **Learner Initialization.** Instantiate a new policy network $\hat{\pi}_1$ with the same architecture as π^* but with random initial weights. At this stage, $\hat{\pi}_1$ has not seen any expert data.

3. **Iterative DAgger Loop ($i = 1, \ldots, N$).**

    1) *Compute Mixing Coefficient β_i.* Choose β_i (for example, $\beta_i = 1/(i+1)$), which controls the probability of following the expert versus the learner during rollout.

    2) *Form and Roll Out Mixture Policy π_i.* Define

    $$\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i,$$

    meaning each action is drawn from π^* with probability β_i and from $\hat{\pi}_i$ with probability $1 - \beta_i$. Execute π_i in the environment for T timesteps (or until termination), recording every visited state s_t. Early iterations rely heavily on π^*, while later ones allow $\hat{\pi}_i$ to drive more of the trajectory.

    3) *Expert Labeling of Recorded States.* For each recorded state s_t, query the expert π^* to obtain the optimal action label $\pi^*(s_t)$. Collect these pairs into the new batch:

    $$\mathcal{D}_i = \{(s_t, \pi^*(s_t))\}_{t=1}^T.$$

    This guarantees that, regardless of whether $\hat{\pi}_i$ contributed to the trajectory, every state is labeled by the expert.

    4) *Aggregate Dataset.* Maintain a growing dataset of expert-labeled pairs:

    $$\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i.$$

    Over successive iterations, $\mathcal{D}$ accumulates states from mixed expert–learner rollouts, ensuring coverage of regions where $\hat{\pi}$ has previously erred.

5) *Learner Retraining.* Treat $\mathcal{D}$ as a supervised learning dataset: inputs are states s, targets are expert actions $\pi^*(s)$. Retrain (or fine-tune) the learner on all of $\mathcal{D}$—minimizing a cross-entropy loss over state–action pairs—to obtain an updated policy $\hat{\pi}_{i+1}$. This update corrects past mistakes by incorporating every labeled sample.

6) *Validation and Tracking.* After retraining, evaluate $\hat{\pi}_{i+1}$ on a fixed set of validation episodes (e.g., 10–20 CartPole episodes without expert intervention). Record its average return. Tracking these validation returns across iterations allows selection of the best performing policy at the end.

4. **Select Final Policy.** After N iterations, identify which $\hat{\pi}_i$ achieved the highest validation return. Designate that policy as the final DAgger agent. This selection ensures we keep the learner best aligned with expert behavior.

5. **Summary of Conceptual Flow.**

   - Use PPO to produce and save an expert oracle π^*.
   - Initialize a random learner $\hat{\pi}_1$ with the same architecture.
   - At each iteration, mix expert and learner rollouts to collect new states, always labeling them via π^*.
   - Aggregate all expert-labeled state–action pairs into a single dataset $\mathcal{D}$.
   - Retrain the learner on $\mathcal{D}$ to correct mistakes, then validate and track performance.
   - After all iterations, select the learner iteration with the highest validation return as the final policy.

3 Training a GAIL Model Using an Expert Policy

In this exercise, we implement Generative Adversarial Imitation Learning (GAIL) to train a policy that imitates a pre-trained expert across different Gym environments (e.g., `CartPole-v1`, `Pendulum-v0`, `BipedalWalker-v3`). We assume that an expert policy π_E has already been obtained (e.g., via PPO or TRPO) and saved to disk. The goal is to train a GAIL agent π_θ that minimizes the divergence between its occupancy measure and that of the expert, without directly accessing reward signals during imitation.

3.1 From GANs to GAIL

Ho & Ermon (2016) observed that matching *occupancy measures*—the distribution over state–action pairs induced by a policy—can be framed as a two–player game analogous to a Generative Adversarial Network (GAN):

- The **generator** is the learner's policy π_θ, whose "samples" are trajectories collected by acting in the environment.
- The **discriminator** D_w receives (s, a) pairs and tries to decide whether they came from the expert dataset $\{\tau_E\}$ or from the learner.
- The policy update treats the discriminator's $\log D_w$ output as a *learned cost* and performs a trust–region policy optimisation (e.g. TRPO) step to minimise that cost while keeping successive policies close in KL distance.

When the game converges, the discriminator can no longer tell learner from expert, implying that π_θ has matched the expert's occupancy measure—and therefore imitates the expert's performance without ever having observed the true reward.

3.2 Generative Adversarial Imitation Learning (GAIL)

Generative Adversarial Imitation Learning (GAIL) is a principled framework for learning to act *purely from demonstrations*, without direct access to the expert's reward signal. Inspired by Generative Adversarial Networks (GANs), it frames imitation as a two-player game between:

- **A policy (generator)** π_θ, which produces state–action pairs by interacting with the environment, seeking to *fool* the discriminator into believing the data came from the expert.
- **A discriminator** D_w, trained to distinguish expert trajectories from learner-generated ones and thereby provides a shaping cost that guides the policy.

The key idea is to *match occupancy measures*: if the learner's distribution over state–action pairs becomes indistinguishable from the expert's, then (under mild conditions) the learner will achieve expert-level returns. Unlike behavioural cloning, GAIL is robust to covariate shift because the policy is always updated on the states it *actually visits*. Unlike apprenticeship learning methods based on hand-crafted feature expectations, GAIL lets a deep discriminator automatically learn useful features.

Operationally, GAIL alternates between:

1. Generating fresh trajectories with the current policy.
2. Updating the discriminator to separate expert from learner data.

3. Updating the policy via a trust-region policy-gradient step that *minimises* the discriminator-induced cost while retaining sufficient entropy for exploration.

This adversarial loop converges when the discriminator can no longer tell the two data sources apart, signalling that the policy has effectively *matched* expert behaviour.

3.3 GAIL Algorithm Explanation

Below is the pseudocode for the GAIL algorithm, adapted from Ho & Ermon (2016). Following the pseudocode, we provide a detailed, code-agnostic explanation of how each step of the algorithm is realized in our experimental pipeline for `CartPole-v1`.

Algorithm 2 GAIL Algorithm (Ho & Ermon, 2016)

Require: Expert trajectories $\{\tau_E\}$, initial policy parameters θ_0, discriminator parameters w_0, entropy weight $\lambda \geq 0$

1: **for** $i = 0, 1, 2, \ldots$ **do**

2: **(a) Sample Learner Trajectories:** Roll out π_{θ_i} for M episodes, collecting $\tau_i = \{(s_t, a_t)\}$.

3: **(b) Discriminator Update:**

- Gather a minibatch of state–action pairs from τ_i (learner data) and from expert buffer $\{\tau_E\}$ (expert data).
- Perform k gradient ascent steps on w to maximize

$$\hat{\mathbb{E}}_{(s,a)\sim\tau_i}\big[\log D_w(s,a)\big] \;+\; \hat{\mathbb{E}}_{(s,a)\sim\tau_E}\big[\log(1 - D_w(s,a))\big].$$

4: **(c) Policy (Generator) Update:**

- Define the surrogate cost function

$$c_{w_{i+1}}(s,a) \;=\; \log D_{w_{i+1}}(s,a)\,,$$

  which serves as the instantaneous cost signal for the policy.
- Perform one TRPO step (KL-constrained natural gradient) on θ to minimize

$$J(\theta) \;=\; \underbrace{\hat{\mathbb{E}}_{(s,a)\sim\tau_i}\big[c_{w_{i+1}}(s,a)\big]}_{\text{Expected discriminator cost}} \;-\; \lambda\, H\big(\pi_\theta\big),$$

  where $H(\pi_\theta)$ is the causal entropy of the policy.

5: **end for**

Ensure: Trained policy π_{θ^*}

Explanation of Each Step

- **(a) Sampling Learner Trajectories.** At iteration i, fix the current policy π_{θ_i}. Run M episodes of interaction in the environment (e.g., `CartPole-v1`), following π_{θ_i} to collect a batch of state–action pairs

$$\tau_i = \{\,(s_t, a_t) \;:\; t = 1, \ldots, T_{\text{max}}\,\}\,.$$

  These are *synthetic* trajectories generated by the learner. We store them in a temporary buffer for discriminator training.

- **(b) Discriminator Update.** The discriminator D_w aims to distinguish whether a given (s, a) came from the *expert* dataset or the *learner* dataset. We randomly sample minibatches of size n from:

$$\{\,(s,a) \mid (s,a) \in \tau_i\,\} \quad \text{(learner data)}, \qquad \{\,(s,a) \mid (s,a) \in \tau_E\,\} \quad \text{(expert data)}.$$

  Then, we update w by ascending the stochastic gradient of

$$\mathcal{L}_D(w) = \hat{\mathbb{E}}_{(s,a)\sim\tau_i}\big[\log D_w(s,a)\big] \;+\; \hat{\mathbb{E}}_{(s,a)\sim\tau_E}\big[\log(1 - D_w(s,a))\big].$$

  Typically, we perform k Adam steps on this objective to ensure the discriminator remains near its optimum for the current policy distribution. Intuitively, $D_w(s, a)$ estimates the likelihood that (s, a) is "generated by the learner" rather than "seen from the expert."

- **(c) Policy (Generator) Update.** Once the discriminator parameters are updated to w_{i+1}, we define the instantaneous cost function

$$c_{w_{i+1}}(s,a) \;=\; \log\big(D_{w_{i+1}}(s,a)\big).$$

  This cost penalizes the learner for generating (s, a) pairs that the discriminator believes are likely from the learner (i.e., (s, a) with high D_w). We then update θ by taking a trust-region policy gradient step that minimizes

$$J(\theta) \;=\; \hat{\mathbb{E}}_{(s,a)\sim\tau_i}\big[\,c_{w_{i+1}}(s,a)\big] \;-\; \lambda\, H(\pi_\theta)\,,$$

  where $H(\pi_\theta)$ is the causal (discounted) entropy

$$H(\pi_\theta) = \mathbb{E}_{s_0\sim p_0}\Big[\sum_{t=0}^{\infty} \gamma^t \big(-\log \pi_\theta(a_t \mid s_t)\big)\Big].$$

  In practice, this policy update is implemented using a TRPO subroutine:

  - Estimate the advantage-weighted gradient $\hat{\mathbb{E}}_{(s,a)\sim\tau_i}[\nabla_\theta \log \pi_\theta(a \mid s)\, Q^{\log}_{w_{i+1}}(s,a)]$, where

$$Q^{\log}_{w_{i+1}}(s,a) = \sum_{t'=0}^{T-1} \gamma^{t'}\, \log\big(D_{w_{i+1}}(s_{t'+t}, a_{t'+t})\big)$$

    is the return of discriminator costs along the future rollout, and $\nabla_\theta H(\pi_\theta)$ is computed via standard policy-gradient formulas for entropy.

  - Constrain the KL divergence $\mathrm{KL}(\pi_{\theta_i} \,\|\, \pi_\theta)$ to be below a small threshold δ, ensuring stable policy updates.

  The net effect is that $\pi_{\theta_{i+1}}$ "fools" the discriminator by generating state–action pairs that resemble those in the expert dataset, while still maintaining sufficient entropy to encourage exploration (controlled by λ).

- **(d) Repeat Until Convergence.** We loop steps (a)–(c) until a termination criterion is met (e.g., discriminator loss plateaus, or policy performance on held-out expert validation exceeds a threshold). In practice, we run for a fixed number of iterations (e.g., 2000 discriminator updates) and monitor:

$$\mathbb{E}_{(s,a)\sim\tau_i}\big[\log(1 - D_{w_i}(s,a))\big] \quad \text{and} \quad \mathbb{E}_{(s,a)\sim\tau_E}\big[\log D_{w_i}(s,a)\big],$$

  ensuring the generator is making the discriminator uncertain.

3.4 High-Level Pipeline

1. Environment Setup & Configuration You first construct a Gym–compatible environment (e.g. `CartPole-v1`) and *seed* every source of pseudo-randomness to guarantee reproducibility. Capturing the observation– and action-space dimensions here is crucial because the shapes determine how you wire the subsequent neural networks.

2. Loading and Validating the Expert A reliable expert policy π_E is the backbone of GAIL. After loading, you *roll it out* for a handful of episodes (ten is common) to verify it still attains near-optimal return. @200 on CartPole-v1 is the canonical sanity check. The resulting collection $\{\tau_E\} = \{(s_t, a_t)\}_{t=1}^{T}$ is stored once and reused forever during discriminator training.

3. Initialising GAIL Components Two multilayer perceptrons are spun up:

- a *policy / generator* π_θ (outputs a Gaussian or categorical distribution over actions);
- a *discriminator* D_w finishing with a sigmoid.

Hyperparameters are where most practical performance lives: distinct learning rates for w and θ, batch sizes for learner and expert samples, and the number k of discriminator updates per outer iteration.

4. Main Training Loop (Algorithm 2) Each outer iteration alternates between: **(a)** sampling learner trajectories; **(b)** performing k gradient-ascent steps on $\mathcal{L}_D(w)$; and **(c)** executing a single TRPO update on θ using the learned cost $\log D_{w_{i+1}}$. The KL constraint in TRPO (often $\delta = 0.01$) is indispensable—it prevents policy collapse when the discriminator suddenly sharpens.

5. Evaluation and Baselines Once training halts, you roll out the learned $\pi_{\theta^\star}$ for K test episodes and report MeanReturn $\pm$ StdReturn, side-by-side with:

- the *expert* (upper bound),
- *behavioural cloning* (supervised baseline),
- and optionally other occupancy-matching methods such as FEM.

If GAIL is healthy, its performance will nestle snugly between BC and the expert, often matching the latter on simple tasks like CartPole and reaching parity on harder domains with sufficient training.

References

[1] Cover image designed by freepik

[2] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. 2011. https://arxiv.org/abs/1011.0686.

[3] Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. 2016. arXiv:1606.03476 [cs.LG].