In the name of God

Interactive Learning Course Final Project:

Policy-Based Reward Estimation for Expert-Driven Social Learning in Multi-Agent Systems

Professor:

Prof. Majid Nili Ahmadabadi

Writers:

Soleiman B.Ganji, Nikoo Paknia

February 2025

Table of Contents

Abstract

In multi-agent reinforcement learning, an agent's ability to leverage the experiences of others can significantly enhance learning efficiency and overall performance. This study explores a social learning framework in which an agent identifies and learns from the most effective peer, referred to as the expert agent. To achieve this, we estimate the policies of all agents and compare them against the social learner's policy using a normalization factor. This factor enables the estimation of the discounted return the social learner would achieve by adopting each reference agent's policy. The agent yielding the highest normalized return is designated as the expert, whose policy is then integrated into the social learner's objective function. This approach accelerates learning, enhances decision-making, and improves convergence rates in multi-agent environments.

Method

This section outlines our proposed methodology for identifying and learning from an expert agent.

First, we define our environment as *Lunar Lander-v2*, a reinforcement learning benchmark within the *Gym* library. This environment features a continuous state space and a discrete action space, necessitating the use of deep learning algorithms for function approximation.

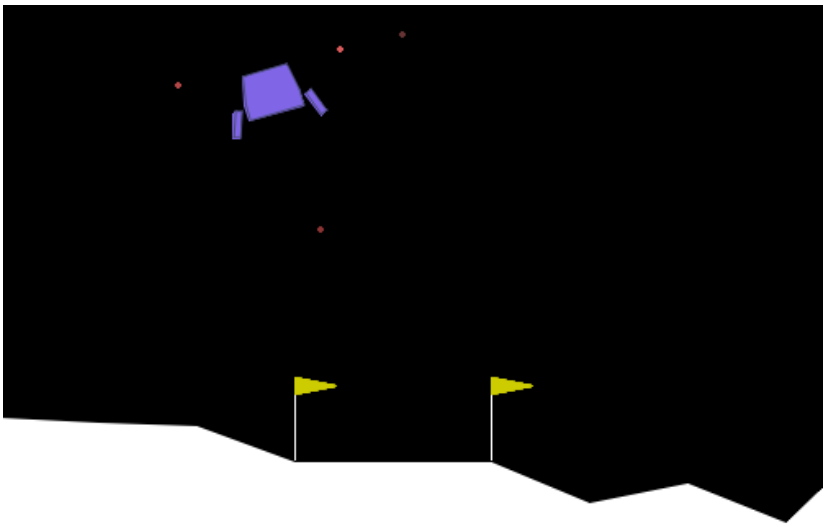
Our experiment involves three types of agents: an expert agent, a social learner, and an agent following a random policy. The procedure begins with training the expert agent using the Q-learning algorithm. Next, we define an objective function for the social learner, which is trained using a policy-based algorithm.

To validate the effectiveness of the objective function and the social agent's policy, we first evaluate performance using the expert's policy directly. Once we confirm that the proposed objective and learning algorithm enables the social agent to learn efficiently, we incorporate policy estimation into the learning framework. Specifically, the social learner estimates the policies of other agents (random and expert) and integrates the expert's policy into its objective function.

A key challenge in this approach is distinguishing the expert agent from the random agent. To address this, we introduce two methods for mapping the social agent's rewards onto other agents' policies. The policy that yields the highest rewards or returns is then identified as the expert's policy and utilized accordingly.

Environment

The environment is *Lunar Lander-v2* one of the environments in the *gym* library. Lunar Lander-v2 is a reinforcement learning benchmark that simulates a controlled descent and landing of a spacecraft. The objective is to guide the lander to a designated landing pad while minimizing fuel consumption and avoiding crashes.



Lunar lander environment

Continuous State Space & Discrete Action Space

The state space is an 8-dimensional vector containing:

- **x, y** coordinates of the lander
- **x, y** linear velocities
- Lander's angle and angular velocity
- Two booleans indicating whether each leg is in contact with the ground

The action space includes four actions as below:

- Do nothing as 0
- Fire left orientation engine as 1
- Fire main engine as 2
- Fire right orientation engine as 3

Reward Function

An episode is considered **solved** if the agent achieves a total score of at least **200 points**.

Reward Contribution

Factor	Reward contribution
Distance to landing pad	Positive for closer proximity, negative otherwise
Velocity	Positive for slower movement, negative otherwise
Orientation	Negative if tilted from horizontal
Leg contact	+10 points per leg touching the ground
Side engine usage	-0.03 points per frame
Main engine usage	-0.3 points per frame
Episode outcome	+100 for successful landing, -100 for crashing

Episode Termination Criteria

An episode ends if:

- The lander crashes (body contact with the moon)
- The lander exits the viewport ($|x| > 1$)
- The lander becomes inactive (stops moving and colliding)

Expert Agent

Since we employ a policy-based algorithm for the social agent (discussed in the next section), we opted for a value-based approach for the expert agent to mitigate potential bias. Specifically, we selected Deep Q-Learning (DQN) as the expert's learning algorithm. An alternative approach, such as Deep SARSA, could also be used.

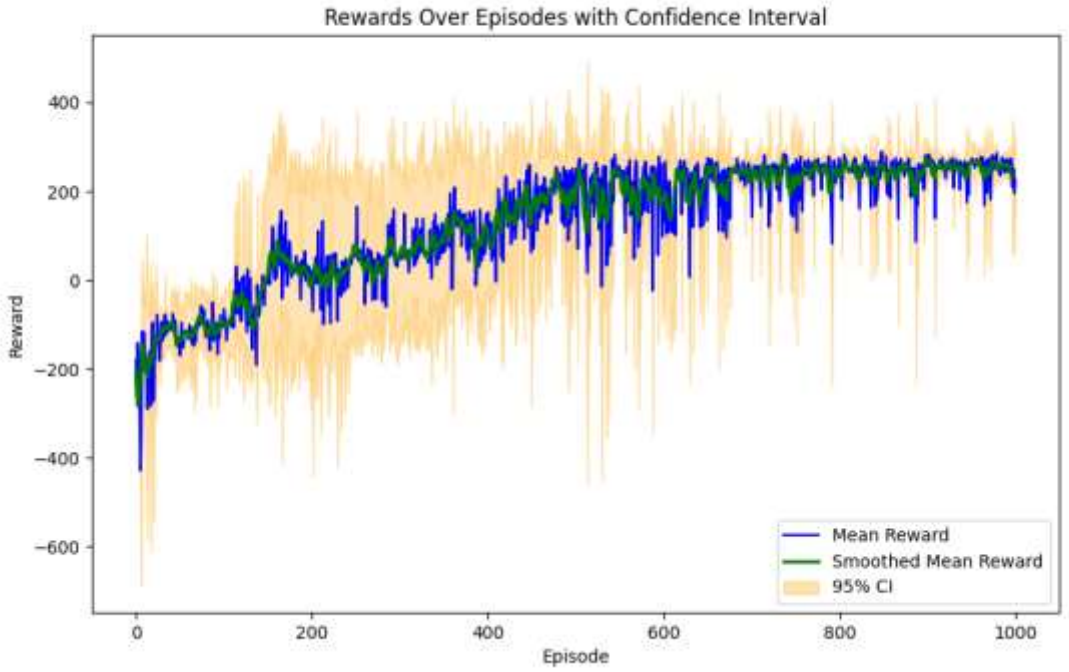
SARSA, as an on-policy method, is guaranteed to converge to the optimal policy with probability one, whereas Q-learning, an off-policy method, converges to the optimal action-value function Q*. When combined with epsilon decay, both algorithms can ultimately reach the optimal policy [Sutton & Barto]. However, DQN demonstrated sufficiently strong performance, making it a choice for our implementation.

DQN network architecture

Layer number	Number of neurons	Activation function
1	128	Relu
2	128	Relu
3	4	Linear

DQN agent hyper-parameters

Hyper-parameters	Discount factor	Initial epsilon	Epsilon decay	Minimum epsilon	Learning rate	Buffer size	Batch size	Replace target network	Number of episodes
Values	0.99	1.0	0.996	0.01	0.001	1000000	128	Every 100 episode	1000



DQN expert agent average reward

Since the agent has achieved an average reward of 200, we can conclude that it has successfully learned the task with an acceptable convergence rate. Additionally, the variance has decreased toward the end of training, indicating stability. This agent is now ready to serve as the expert as a saved model with its final epsilon.

Social Agent

The social agent should be trained using a policy-based algorithm, as our objective in social learning imposes constraints on policies. Additionally, the agent has access only to the states and actions of other agents, rather than their rewards.

$$Objective: \max_{\pi_\theta}(E_\pi[R] - KL(\pi_\theta || \pi_{expert}))$$

Learning from Expert's Policy

The general objective in RL problems is:

$$\max_{\pi_\theta} E_\pi[R]$$

But the constraint of learning from another agent manifests as a policy constraint, typically expressed as a probabilistic distance between the social agent's policy and the expert's policy, such as KL divergence:

$$\min_{\pi_\theta} KL(\pi_\theta || \pi_{expert})$$

The reward maximization term can be transformed into its policy form as below:

$$\max_{\pi_\theta} G_t \ln \pi_\theta(s_t, a_t), \text{ or } \min_{\pi_\theta} -G_t \ln \pi_\theta(s_t, a_t)$$

Since G_t is computed through sampling, the objective functions depend solely, on $\ln \pi_\theta(s_t, a_t)$, a function of the agent's policy. This approach enables the integration of the reward maximization term and the KL divergence term into a unified objective function:

$$\min_{\pi_\theta} -G_t \ln \pi_\theta(s_t, a_t) + cKL(\pi_\theta || \pi_{expert})$$

The key question is: which approach is more appropriate for calculating G_t—the Monte Carlo method or the Temporal Difference (TD) method? Should we use the average reward or the discounted reward? We opted for the discounted reward, as it exhibits a positive correlation with the average reward. While both Monte Carlo and TD methods are known to converge to the optimal policy [Sutton & Barto], the relative speed of convergence remains mathematically unproven. Given this, we initially adopted the Monte Carlo approach, which forms the basis of the REINFORCE algorithm.

Reinforce Algorithm

In the REINFORCE algorithm, G_t is computed by sampling the entire episode (for episodic tasks) and applying the following relation:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$$

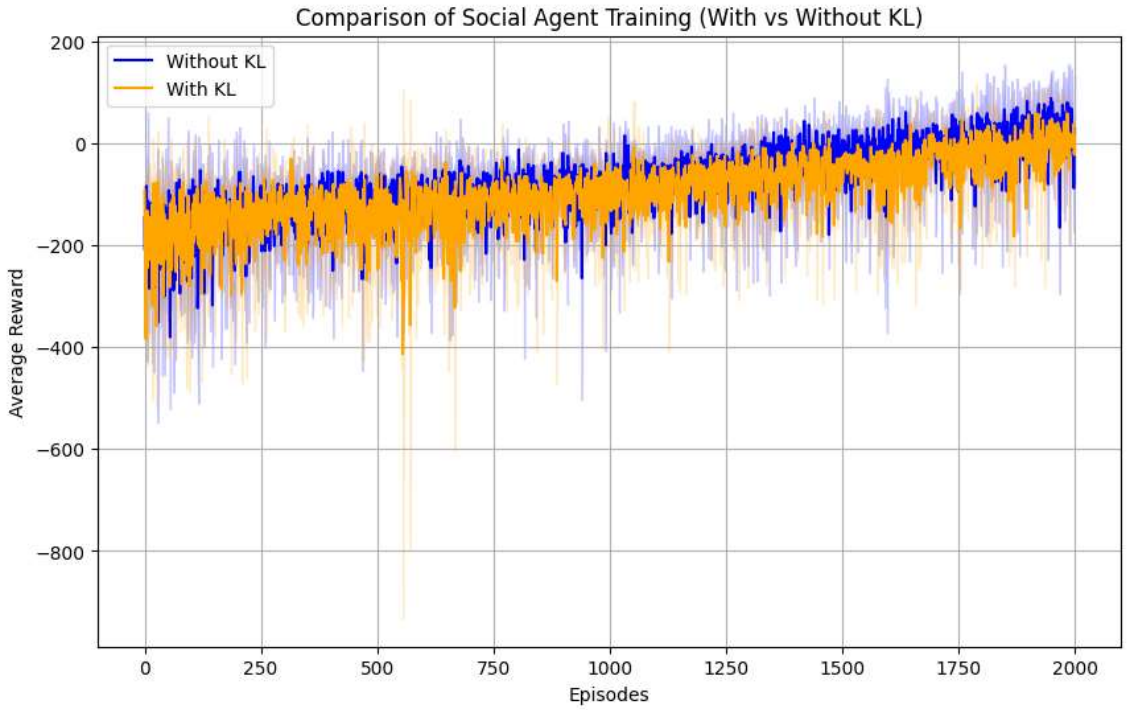
Social agent's policy network architecture

Layer number	Number of neurons	Activation function
1	128	Relu
2	4	Softmax

REINFORCE social agent's hyper-parameters

Hyper-parameters	Discount factor	Learning rate	Buffer size	Batch size	Number of episodes	KL coefficient	Update every
Values	0.99	0.001	100000	64	2000	0.1	10 episode

We first trained the social agent without social learning (excluding the KL term) and then with social learning, varying the KL term coefficient from 0.1 to 1. However, in all cases, the REINFORCE algorithm failed to yield satisfactory results, both in the absence and presence of social learning. The plot below compares the performance of the REINFORCE algorithm alone versus its integration with social learning.



REINFORCE agent with vs. without social learning

Based on the plot above, it is evident that the REINFORCE algorithm may not be well-suited for this task and environment due to the high variance in policy gradient estimates, particularly in such complex, multi-step decision-making task. Furthermore, social learning appears to provide little to no benefit in this context.

On the other hand, *REINFORCE with a baseline*—such as a value function or a learned critic—helps reduce variance in gradient estimates by centering the policy updates around expected returns rather than absolute returns. This makes training more stable and improves learning efficiency, making it more viable for environments like Lunar Lander where reward signals can be highly variable.

REINFORCE With a Baseline

This algorithm is similar to the REINFORCE algorithm; however, instead of using G_t, it utilizes$G_t - V(S_t)$, where $V(S_t)$ serves as the baseline. In this way, the algorithm resembles the Actor-Critic method, but the return is estimated using a Monte Carlo approach.

The policy gradient update in REINFORCE is given by:

$$\nabla_\theta J(\theta) = E_\pi[G_t \nabla_\theta \ln \pi_\theta(s_t, a_t)]$$

Now, considering a baseline function $V(S_t)$, the new gradient update is:

$$\nabla_\theta J(\theta) = E_\pi[(G_t - V(S_t)) \nabla_\theta \ln \pi_\theta(s_t, a_t)]$$

$$\nabla_\theta J(\theta) = E_\pi[G_t \nabla_\theta \ln \pi_\theta(s_t, a_t)] - E_\pi[V(S_t) \nabla_\theta \ln \pi_\theta(s_t, a_t)] \quad (1)$$

$$E_\pi[V(S_t) \nabla_\theta \ln \pi_\theta(s_t, a_t)] = V(S_t) E_\pi[\nabla_\theta \ln \pi_\theta(s_t, a_t)] \quad (2)$$

$$E_\pi[\nabla_\theta \ln \pi_\theta(s_t, a_t)] = \sum_a \pi_\theta(a_t|s_t) \nabla_\theta \ln \pi_\theta(s_t, a_t) = \sum_a \pi_\theta(a_t|s_t) \frac{\nabla_\theta \pi_\theta(s_t, a_t)}{\pi_\theta(a_t|s_t)}$$

$$E_\pi[\nabla_\theta \ln \pi_\theta(s_t, a_t)] = \sum_a \nabla_\theta \pi_\theta(s_t, a_t) = \nabla_\theta \sum_a \pi_\theta(s_t, a_t) = 1 \quad (3)$$

$$(3) \text{ \& } (2): E_\pi[V(S_t) \nabla_\theta \ln \pi_\theta(s_t, a_t)] = 0 \quad (4)$$

$$(4) \text{ \& } (1): \nabla_\theta J(\theta) = E_\pi[G_t \nabla_\theta \ln \pi_\theta(s_t, a_t)] = E_\pi[(G_t - V(S_t)) \nabla_\theta \ln \pi_\theta(s_t, a_t)]$$

Although the expected gradient remains the same, the variance of the gradient estimate is reduced because $G_t - V(S_t)$ has lower variance than G_t alone. The baseline helps by subtracting an estimate of the expected return, reducing fluctuations in the updates, and leading to more stable training.

Thus, while the baseline does not affect convergence, it improves learning efficiency by reducing gradient variance. The following pseudocode outlines the training procedure for the social agent:

1. Initialize policy network π_θ, value function $V_\phi(S)$
2. Set learning rates $\alpha_\theta$and α_ϕ
3. For each episode do:
    a. Collect a trajectory $\tau = \{(S_t, a_t, r_t)\}$ by following π_θ
    b. For each time step t in trajectory τ:
        i. Compute returns: $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
        ii. Compute the advantage function using the baseline: $A_t = G_t - V_\phi(S_t)$
    c. End for
    d. Update policy parameters using the advantage function:

        $\theta \leftarrow \theta + \alpha_\theta * A_t * \nabla_\theta \log \pi_\theta(a_t|S_t) + c * \nabla_\theta(\mathrm{KL}(\pi_\theta \| \pi_{expert}))$

    e. Update value function to minimize mean squared error (MSE):

        $\phi \leftarrow \phi - \alpha_\phi * \nabla_\phi (G_t - V_\phi(S_t))^2$

4. End for

Critic $V(S_t)$ network architecture

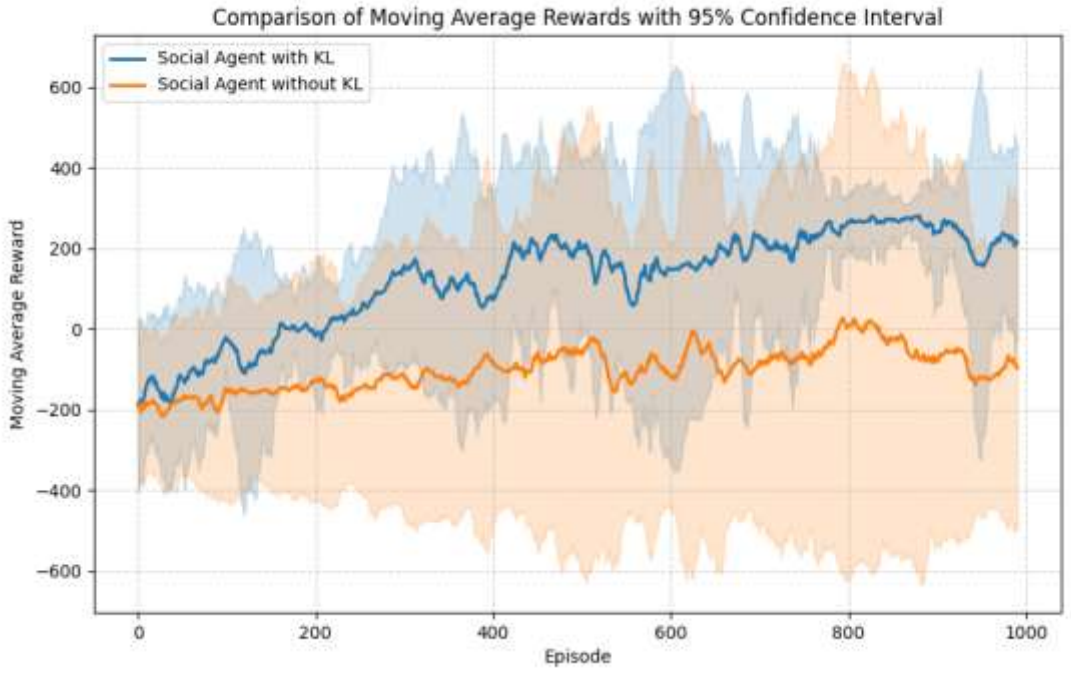
Layer number	Number of neurons	Activation function
1	256	Relu
2	256	Relu
3	4	Linear

Actor $\pi_\theta(a_t|s_t)$ network architecture

Layer number	Number of neurons	Activation function
1	256	Relu
2	256	Relu
3	4	Softmax

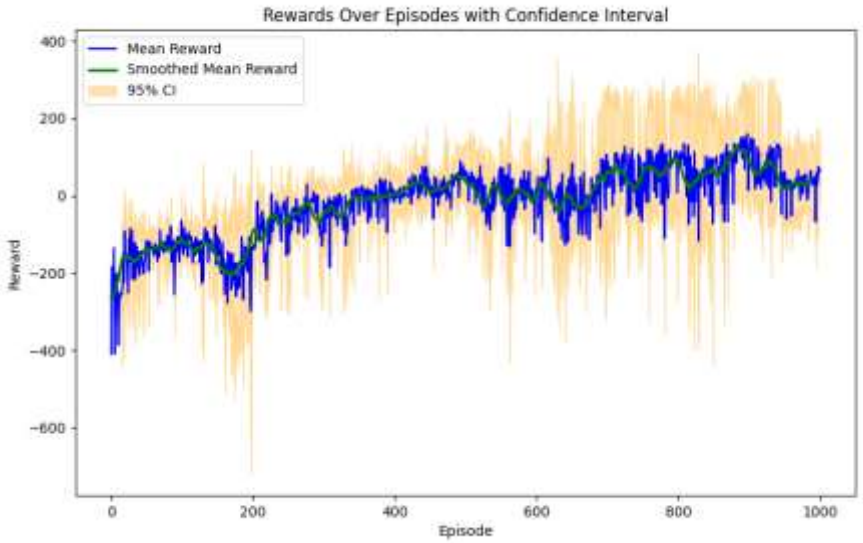
REINFORCE With Baseline Social Learning Hyper-parameters

Hyper-parameters	Discount factor	Actor learning rate	Critic learning rate	Batch size	Number of episodes	KL coefficient
Values	0.99	0.002	0.002	128	1000	0.05

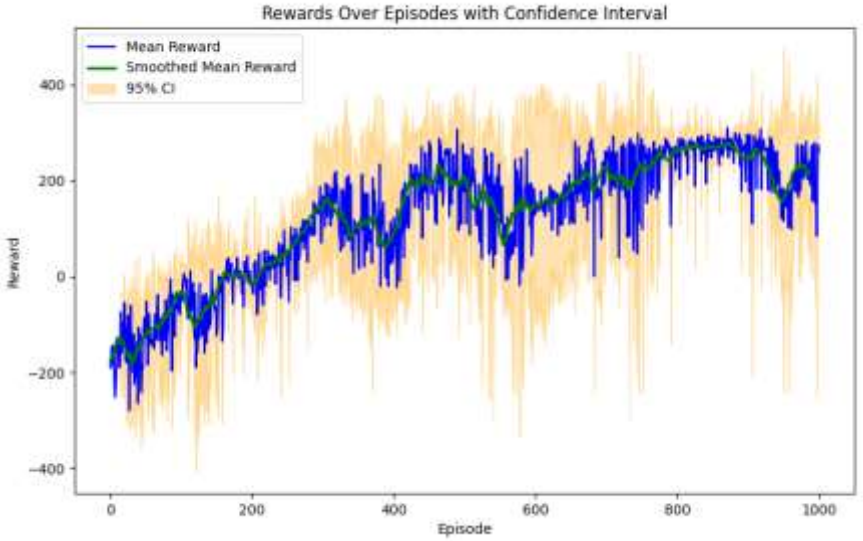


REINFORCE with Baseline with vs. without social learning

As demonstrated above, incorporating an expert agent as a guide in policy space significantly enhances the agent's performance.



Average reward plot for the social agent before using social learning



Average reward plot for the social agent after using social learning

Having established that leveraging an expert's policy can enhance an agent's learning with the new algorithm, we now extend this approach further. In real-world tasks, agents do not have direct access to others' policies but can estimate them. Therefore, it is necessary to develop a policy estimator to approximate other agents' policies and incorporate it into the KL term.

Policy Estimator Network

To address the absence of directly accessible policies, we propose a policy estimator network that learns to approximate the expert agent's policy from observed state-action pairs. The following pseudocode outlines the training procedure for the social agent:

1. Initialize policy network π_θ, value function $V_\phi(S)$
2. Set learning rates $\alpha_\theta$and α_ϕ
3. **Import the expert agent**
4. For each episode do:
    a. **Let the expert live and collect its state-action pairs in a buffer**
    b. Collect a trajectory $\tau = \{(S_t, a_t, r_t)\}$ by following π_θ
    c. For each time step t in trajectory τ:
        i. Compute returns: $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
        ii. Compute the advantage function using the baseline: $A_t = G_t - V_\phi(S_t)$
    d. End for
    e. **Train the policy estimator on the expert's buffer**
    f. Update policy parameters using the advantage function:

$$\boldsymbol{\theta \leftarrow \theta + \alpha_\theta * A_t * \nabla_\theta \log \pi_\theta (a_t|S_t) + c * \nabla_\theta (KL(\pi_\theta || \pi_{expert\ estimator}))}$$

    g. Update value function to minimize mean squared error (MSE):

$$\phi \leftarrow \phi - \alpha_\phi * \nabla_\phi (G_t - V_\phi(S_t))^2$$

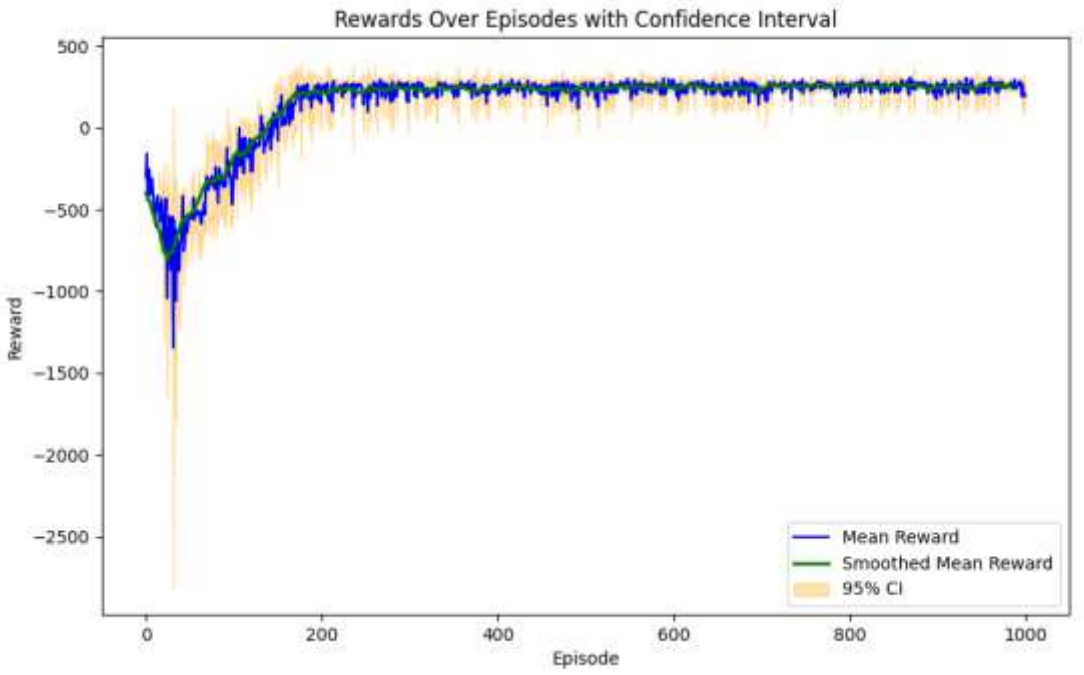
5. End for

Policy estimator network architecture

Layer number	Number of neurons	Activation function
1	128	Relu
2	128	Relu
3	4	Softmax

Policy estimator hyper-parameters

Hyper-parameters	Learning rate	Batch size	Number of epochs	KL coefficient
Values	0.001	64	10	30



Rewards of the social agent after using policy estimator for KL term

In implementing the above algorithm, we introduced modifications to the training schedule. Specifically, for the first 300 episodes, the policy network estimator is trained once every 10 episodes using the stored buffer of state-action pairs from the expert. After episode 300, the training frequency is adjusted to once every 50 episodes.

Due to the use of an estimated policy network instead of the original expert network, the social agent's reward loss (actor loss) was significantly larger in scale compared to its KL divergence loss. Consequently, unlike the previous setting where a KL coefficient of 0.05 was optimal, a coefficient of 30 proved to be the most effective in facilitating stable and efficient learning.

The plot above demonstrates that even when relying on an estimated expert policy, the social agent is still able to achieve accelerated learning.

A Meter for Expert Identification

Step-wise Mapped Reward (SWMR)

We introduce a measure to distinguish the expert agent in the presence of an agent following a random policy. Given access to either the actual policies of other agents or their estimations, this metric allows us to infer the potential rewards they would obtain if they were in the same state as the observed agent. This approach enables reward estimation for other agents within a given episode, even in the absence of direct access to their actual rewards.

The Step-Wise Importance Sampling with Model-Based Reward (SWMR) method is inspired by importance sampling, which estimates the Q-values of a target policy based on a behavior policy. However, unlike traditional importance sampling, this approach operates in a step-wise manner, allowing estimations to be made from individual samples within an episode without requiring a full sequence of observations.

The proposed metric is defined as follows:

$$\mathrm{SWMR}_i = \frac{1}{T}\sum_{t=1}^{T} \frac{\pi^{i}_{estimated}(a_t|S_t)}{\pi_{scoial\ learner}(a_t|S_t)} r_{t+1}$$

The rationale behind this metric is that an expert agent, when in state S_t at time t, is expected to select the optimal action. If an agent selects the action a_t that leads to a high reward, the expert would likely have chosen the same action, resulting in a significant effect on r_{t+1}. Conversely, if an agent selects an action that results in a penalty, the expert is less likely to have chosen it, thereby reducing the impact of r_{t+1} for that action. Over time, the expert is expected to accumulate greater rewards across newly encountered states. While the states encountered also depend on the agent's policy, our goal is to quantify a step-wise measure of decision-making efficiency, which we refer to as step-wise wisdom.

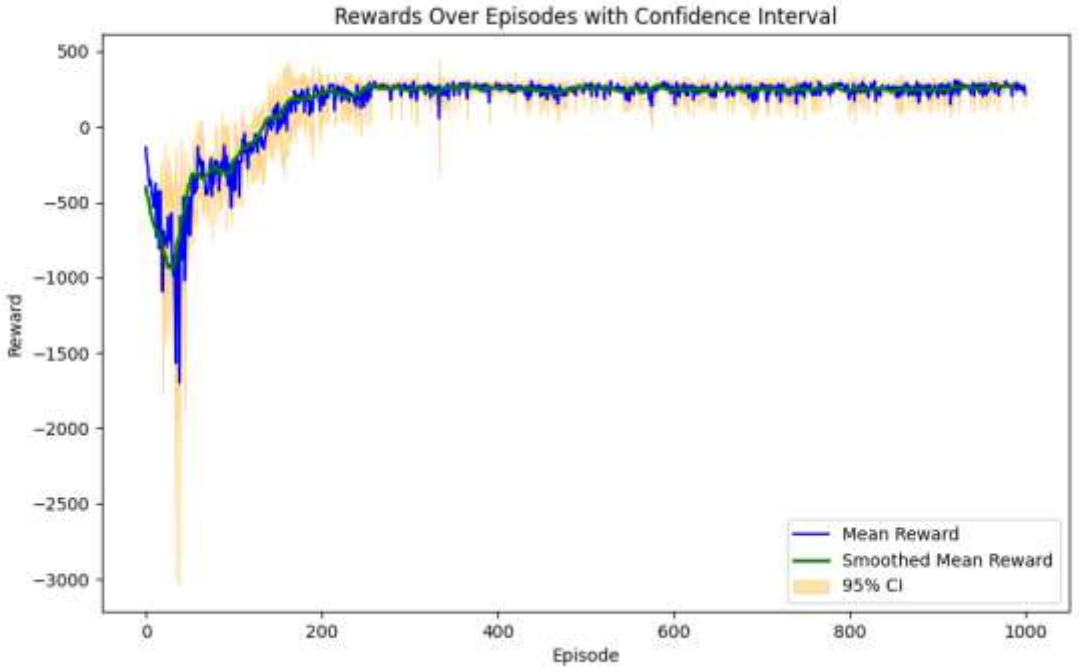
The agent with the highest SWMR score is the estimated expert agent. In the next phase, we aim to integrate this metric into our previous algorithms to develop the final framework. This will allow us to evaluate the performance of the social learner, which will be required first to select an agent to follow—either the expert agent or the agent following a random policy.

Training Social Agent with An Estimation of Expert's Policy

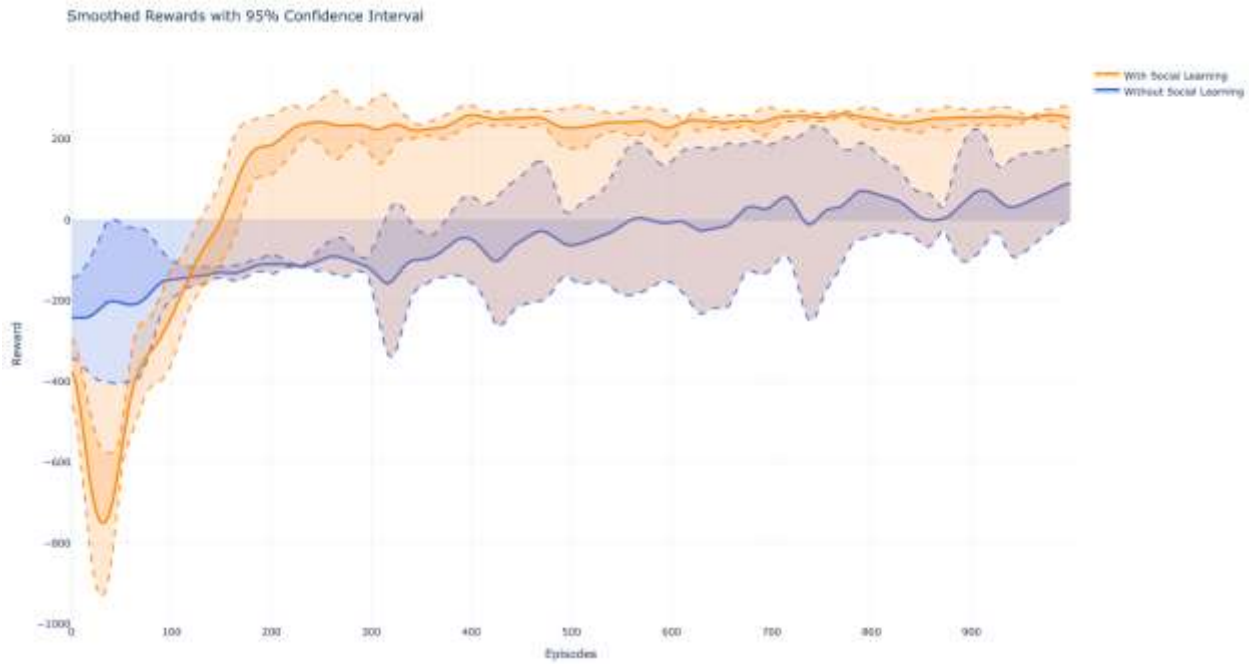
In this section, until episode 400, the expert agent is identified at each episode using the SWMR metric introduced earlier. After episode 400, the expert is determined based on the history of these 400 episodes, where the agent with the highest selection rate is designated as the expert. This approach accelerates the learning process of the social agent; however, it is most effective in scenarios where the expert's policy remains relatively stable over time. In cases where only pseudo-expert agents are present, more frequent comparisons may be necessary to ensure accurate expert identification.

The policy estimator network update follows the same procedure as before. Additionally, all networks and hyper-parameters remain consistent with those defined in the policy estimator section, and the KL loss coefficient β is set to 30.

The plots below demonstrate that the social learner has not only achieved effective and rapid learning but also exhibits a significant performance improvement when using our proposed method compared to when it is not applied.



Social learner agent when it found the expert agent by itself and used its estimated policy to minimize the KL distance loss as well as REINFORCE with baseline loss



Comparison between REINFORCE with baseline agent before and after using our method for its social learning

As shown in the plot above, the social agent's performance is significantly improved, despite undergoing a brief warm-up phase. This initial phase is expected, as the agent first needs to identify the expert while its policy estimator networks gradually learn and improve. However, the warm-up period is relatively short, and the rapid learning speed compensates for this initial cost.

Conclusion

In this project, we first demonstrated that KL divergence can significantly influence social learning, provided that the reference agent is an expert and the social agent employs stable algorithms such as REINFORCE with a baseline. We also established that a policy estimator can effectively replace the original policies of other agents while still enabling successful social learning.

Furthermore, we introduced the SWMR metric, designed to identify the expert agent in the presence of a weaker agent. The combination of this metric and the policy estimator approach greatly enhanced the social agent's ability to converge to the optimal policy in a fast and stable manner.

Algorithm 1 The Social Learning Algorithm

Initialize c, π_b and $\pi_i \forall i \in Agents$
for each trial t **do**
    if t mod $k = 0$, sample mini-batches of state-actions and train the policy estimators for all agents
    Sample $A_t \sim \pi_b$ and observe reward R_t, and other agents' actions
    Calculate average reward $\mathbb{E}_{\pi_b}[r_t]$
    Estimate $\pi_i \forall i \in Agents$
    Select the agent which maximizes $\rho \bar{R}$ as the reference agent
    Set the loss function to $J(\theta) = \mathbb{E}[r \mid \pi] + cF(\pi_b, \pi_{\text{ref}})$ and backpropagate
end for

Our final algorithm for social learning used to train the Lunar lander agent

Future Works

In this project, we considered a setting with only an expert agent and an agent following a random policy. However, these tests could also be extended to scenarios involving pseudo-experts or cases where no expert is present, and all agents are learning simultaneously.

Additionally, alternative metrics, such as a Monte Carlo variant of our proposed measure, could be explored in conjunction with actor-critic or other policy-based algorithms for the social agent. This approach may further accelerate convergence by leveraging social learning.