APPRENTICESHIP LEARNING VIA INVERSE REINFORCEMENT LEARNING

Shaz Nazar Karumarot

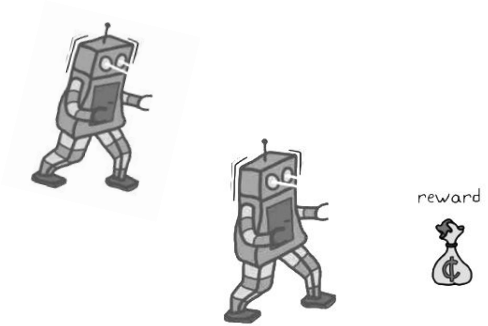
TABLE OF CONTENTS

01

OBJECTIVE

OBJECTIVE

- Learning in a Markov decision process (MDP) without an explicit reward function.
- By **observing an expert's demonstration** of the task.
- Approach:
  - Utilize "**Inverse Reinforcement Learning**" to recover the reward function.
  - Algorithm terminates in a small number of iterations.
- Performance:
  - The policy output by the algorithm achieves **performance close to that of the expert**.
  - Performance is measured based on the expert's unknown reward function.



MIMIC THE EXPERT?

- Traditional Approach: Directly **mimic expert's actions** using supervised learning (Examples: Sammut et al., 1992; Kuniyoshi et al., 1994).
    - Limitation: **Not flexible** for situations with varying contexts.

- Alternative Approach: **Learn the reward function** that guides the expert's behavior (Inverse Reinforcement Learning - Ng & Russell, 2000).
    - Benefit: More **robust and transferable** to new situations.

USING INVERSE RL

- This approach assumes the expert is optimizing an **unknown reward** function based on **known features**.

- We can't guarantee recovering the exact reward function, but the learned policy will still **achieve good performance** based on the expert's unknown reward function.

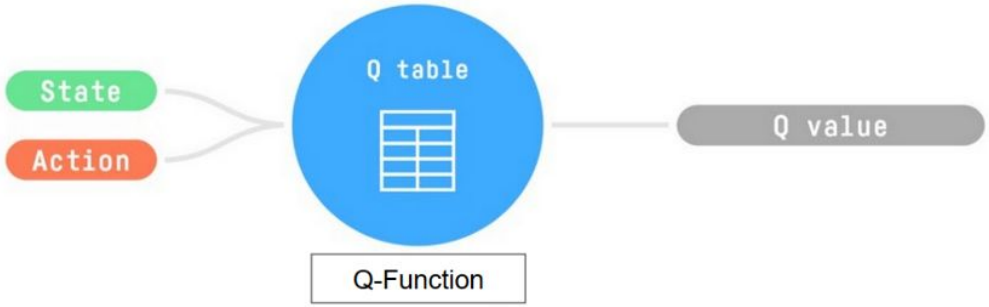
02

TRAINING THE EXPERT

Q-LEARNING

Q-LEARNING

- Q-Learning is an off-policy and **value-based** method that uses a TD (Temporal Difference) approach to train its core function, the Q-function.
    - The value of a state, or a state-action pair is the **expected cumulative reward** our agent gets if it starts at this state (or state-action pair) and then acts accordingly to its policy.
    - The Q-value represents the "**quality**" of an action in a given state.



Source: Hugging Face Deep Reinforcement Learning Course - Hugging Face. (n.d.).

Q-LEARNING

- Trains a Q-table that contains all the **state-action pair values**.
- Given a state and action, search for the corresponding value.
- When the training is done, we have an **optimal Q-table**.
- And if we have an optimal Q-table, we have an **optimal policy** since we know the best action to take at each state.

Actions

States

	A_1	A_2	...	A_M
S_1	$Q(S_1, A_1)$	$Q(S_1, A_2)$		$Q(S_1, A_M)$
S_2	$Q(S_2, A_1)$	$Q(S_2, A_2)$		$Q(S_2, A_M)$
⋮			⋱	⋮
S_N	$Q(S_N, A_1)$	$Q(S_N, A_2)$	...	$Q(S_N, A_M)$

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

Q-LEARNING

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(terminal\text{-}state, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do** Step 1
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g., ϵ-greedy) Step 2
    Take action A_t and observe R_{t+1}, S_{t+1} Step 3
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$ Step 4
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

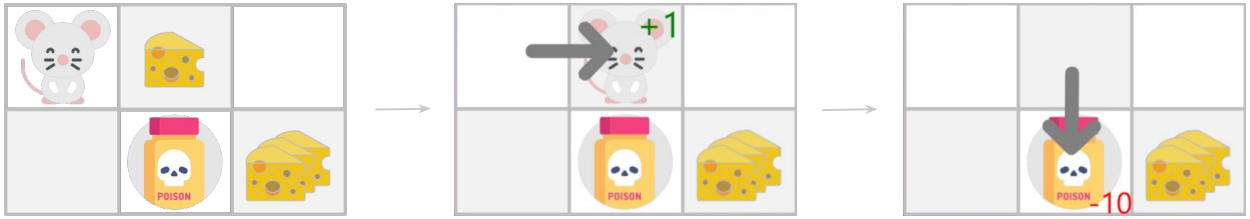
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

- New Q-value estimation
- Former Q-value estimation
- Learning Rate
- Immediate Reward
- Discounted Estimate optimal Q-value of next state
- Former Q-value estimation
- TD Target
- TD Error

Source: Hugging Face Deep Reinforcement Learning Course - Hugging Face. (n.d.).

Q-LEARNING

EXAMPLE



- Learning rate = 0.1
- Gamma = 0.99
- +0: no cheese.
- +1: small cheese.
- +10: large cheese.
- -10: poison.

Q(s, a) = 0

	←	→	↑	↓
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

Q(initial state, Right) = 0 + 0.1 * [1 + 0.99 *0 - 0]

	←	→	↑	↓
	0	0.1	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

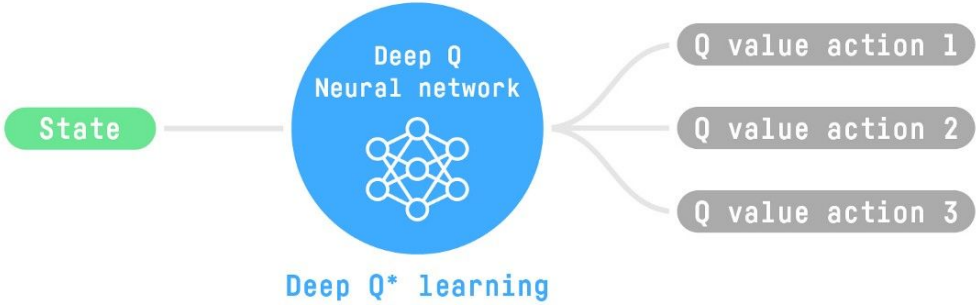
Q(State 2, Down) =0+0.1 * [-10 + 0.99*0-0]

	←	→	↑	↓
	0	0.1	0	0
	0	0	0	-1
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

DEEP Q-LEARNING

Deep Q-LEARNING

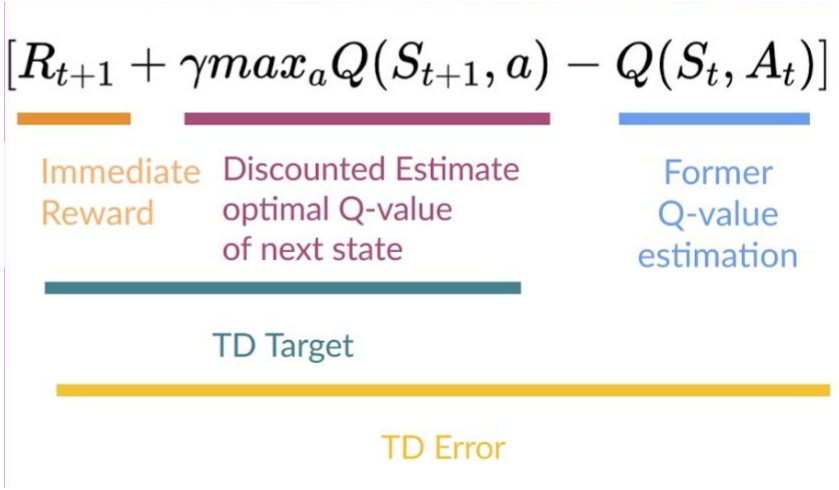
- In large environments, the number of possible **states can be enormous**, making it impractical to create and maintain a Q-table.
- Deep Q-Learning replaces the Q-table with a Deep Q-Network (**DQN**).
- The DQN is a **neural network** that approximates the Q-values for any given state-action pair.
- This allows Deep Q-Learning to **handle large state spaces** effectively.



Source: Hugging Face Deep Reinforcement Learning Course - Hugging Face. (n.d.).

Deep Q-LEARNING

- In Deep Q-Learning, we create a **loss function** that compares our Q-value prediction and the Q-target and uses **gradient descent** to update the weights of our Deep Q-Network to approximate our Q-values better.



Q-Loss

Source: Hugging Face Deep Reinforcement Learning Course - Hugging Face. (n.d.).

Deep Q-LEARNING

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**

With probability ε select a random action a_t
otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
Execute action a_t in emulator and observe reward r_t and image x_{t+1}
Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sampling

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
Every C steps reset $\hat{Q} = Q$

Training

  End For
End For

Source: Hugging Face Deep Reinforcement Learning Course - Hugging Face. (n.d.).

Experience Replay

- Deep Q-Learning can suffer from **training instability** due to non-linear Q-value function (neural network) and bootstrapping (estimating targets).

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Source: Hugging Face Deep Reinforcement Learning Course - Hugging Face. (n.d.).

Experience Replay

A replay buffer that saves experiences that we can efficiently reuse during the training.

It avoids forgetting previous experiences (aka catastrophic interference, or catastrophic forgetting) and reduces the correlation between experiences which avoids action values from oscillating or diverging catastrophically.

Double DQN

- **Double DQNs**, or Double Deep Q-Learning neural networks (**Hasselt et al., 2015**) handles the problem of the **overestimation of Q-values**.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Source: Hugging Face Deep Reinforcement Learning Course - Hugging Face. (n.d.).

Experience Replay

Fixed Q-Target

Since we use the same weights for estimating the TD target and the Q-value, both values shift. This can lead to significant oscillation in training.

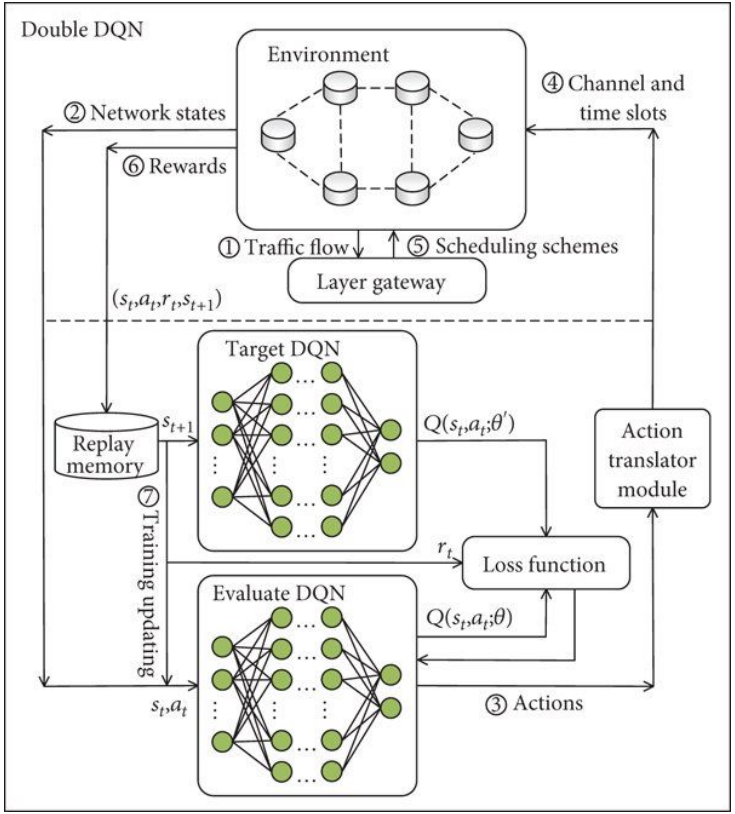
Use a separate network with fixed parameters for estimating the TD Target.

Copy the parameters from our Deep Q-Network every C steps to update the target network.

Double DQN

- If non-optimal actions receive higher Q values than the optimal action, learning becomes complicated.
- The solution to this involves employing **two separate networks** to handle action selection and target Q-value generation when computing the Q target. Specifically:
    - We utilize our **DQN network** to determine the **optimal action** for the subsequent state, selecting the action with the highest Q-value.
    - Our **Target network** is utilized to compute the **target Q-value** associated with taking that action in the subsequent state.
- By implementing Double DQN, we mitigate the issue of overestimating Q-values, resulting in accelerated and more consistent learning during training.

Double DQN



Source: Deep Reinforcement Learning for Scheduling in an Edge Computing-Based Industrial Internet of Things - Scientific Figure on ResearchGate

03

APPRENTICESHIP LEARNING VIA IRL

Preliminaries

- We consider an **MDP\R** - (S, A, T, γ, D)
- State Features (φ) and True Reward Function (R*):
  - **φ(s):** Represents a **vector of features** associated with each state, providing additional information beyond the basic state identity.
  - **R*(s) = w* · φ(s):** Defines a **"true" reward function** based on the **feature vector** and a **weight vector (w*)**.
- **Feature Expectations (μ(π))**: The expected discounted accumulated feature vector for a policy π (captures the long-term effects of a policy on state features).

$$\mu(\pi) = E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi] \in \mathbb{R}^k.$$

Preliminaries

- Learning from Expert Demonstrations:
  - We assume access to **demonstrations** generated by an expert policy (π_E).
  - We can estimate the expert's feature expectations (μ_E) from observed monte carlo **trajectories**.
  - The empirical estimate for $\boldsymbol{\mu_E} = \mu(\pi_E)$ based on a set of m observed expert trajectories is given by:

$$m \text{ trajectories } \{s_0^{(i)}, s_1^{(i)}, \ldots\}_{i=1}^{m}$$

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)})$$

Mean over all demonstrations

Discount-factor weighted sum of feature vectors

MAX-MARGIN

Algorithm

ALGORITHMS: Max Margin

Problem: Find a policy ˜π that induces feature expectations μ(˜π) close to μE.

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Source: Abbeel, P. & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. ICML '04: Proceedings of the twenty-first international conference on Machine learning (p./pp. 1), New York, NY, USA: ACM. ISBN: 1-58113-828-5

ALGORITHMS: Max Margin

Problem: Find a policy ˜π that induces feature expectations μ(˜π) close to μE.

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Initialize a random policy, compute it's feature expectations

Source: Abbeel, P. & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. ICML '04: Proceedings of the twenty-first international conference on Machine learning (p./pp. 1), New York, NY, USA: ACM. ISBN: 1-58113-828-5

ALGORITHMS: Max Margin

Problem: Find a policy ˜π that induces feature expectations μ(˜π) close to μE.

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Inverse Reinforcement Learning step:

Optimize w, using a Quadratic Programming Solver or SVM, to maximize the margin t between the expert and the best policy found thus far.

Source: Abbeel, P. & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. ICML '04: Proceedings of the twenty-first international conference on Machine learning (p./pp. 1), New York, NY, USA: ACM. ISBN: 1-58113-828-5

ALGORITHMS: Max Margin

Problem: Find a policy ˜π that induces feature expectations μ(˜π) close to μE.

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Terminate within the threshold margin, i.e., when margin ≤ ϵ

(Returns a set of policies - pick one with best performance)

Source: Abbeel, P. & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. ICML '04: Proceedings of the twenty-first international conference on Machine learning (p./pp. 1), New York, NY, USA: ACM. ISBN: 1-58113-828-5

ALGORITHMS: Max Margin

Problem: Find a policy ˜π that induces feature expectations μ(˜π) close to μE.

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Retrain using new weights and rewards (reward = $w^T \cdot \varphi$) to obtain new policy.

Source: Abbeel, P. & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. ICML '04: Proceedings of the twenty-first international conference on Machine learning (p./pp. 1), New York, NY, USA: ACM. ISBN: 1-58113-828-5

ALGORITHMS: Max Margin

Problem: Find a policy ˜π that induces feature expectations μ(˜π) close to μE.

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Compute feature expectation of new policy

Source: Abbeel, P. & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. ICML '04: Proceedings of the twenty-first international conference on Machine learning (p./pp. 1), New York, NY, USA: ACM. ISBN: 1-58113-828-5

ALGORITHMS: Max Margin

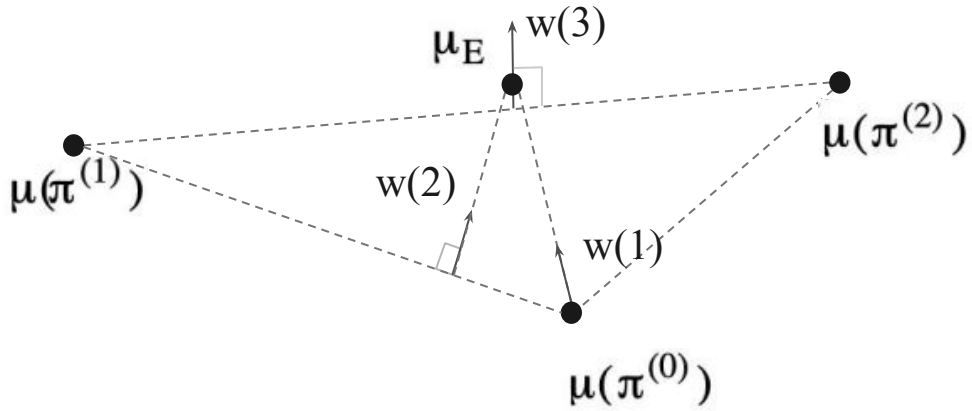
Problem: Find a policy ˜π that induces feature expectations μ(˜π) close to μE.

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Repeat to get new policies until termination

Source: Abbeel, P. & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. ICML '04: Proceedings of the twenty-first international conference on Machine learning (p./pp. 1), New York, NY, USA: ACM. ISBN: 1-58113-828-5

ALGORITHMS: Max Margin



First three iterations of the max-margin algorithm

Source: Abbeel, P. & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. ICML '04: Proceedings of the twenty-first international conference on Machine learning (p./pp. 1), New York, NY, USA: ACM. ISBN: 1-58113-828-5

PROJECTION METHOD

Algorithm

ALGORITHMS: Projection Method

Problem: Find a policy ˜π that induces feature expectations μ(˜π) close to μE.

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

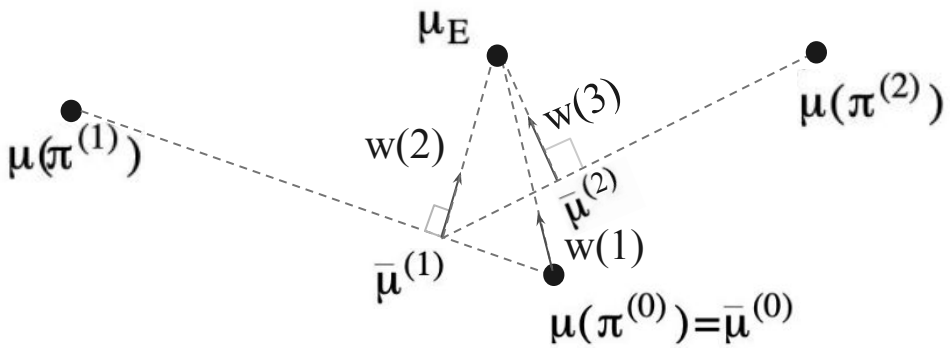
→

- Set $\bar{\mu}^{(i-1)} = \bar{\mu}^{(i-2)} + \frac{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T (\mu_E - \bar{\mu}^{(i-2)})}{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T (\mu^{(i-1)} - \bar{\mu}^{(i-2)})} (\mu^{(i-1)} - \bar{\mu}^{(i-2)})$ (This computes the orthogonal projection of μ_E onto the line through $\bar{\mu}^{(i-2)}$ and $\mu^{(i-1)}$.)
- Set $w^{(i)} = \mu_E - \bar{\mu}^{(i-1)}$
- Set $t^{(i)} = \|\mu_E - \bar{\mu}^{(i-1)}\|_2$

Eliminates need for QP Solver

Source: Abbeel, P. & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. ICML '04: Proceedings of the twenty-first international conference on Machine learning (p./pp. 1), New York, NY, USA: ACM. ISBN: 1-58113-828-5

ALGORITHMS: Projection Method



First three iterations of the projection algorithm

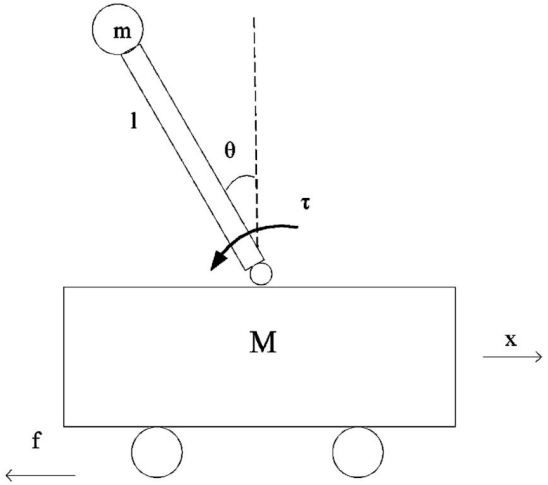
Source: Abbeel, P. & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. ICML '04: Proceedings of the twenty-first international conference on Machine learning (p./pp. 1), New York, NY, USA: ACM. ISBN: 1-58113-828-5

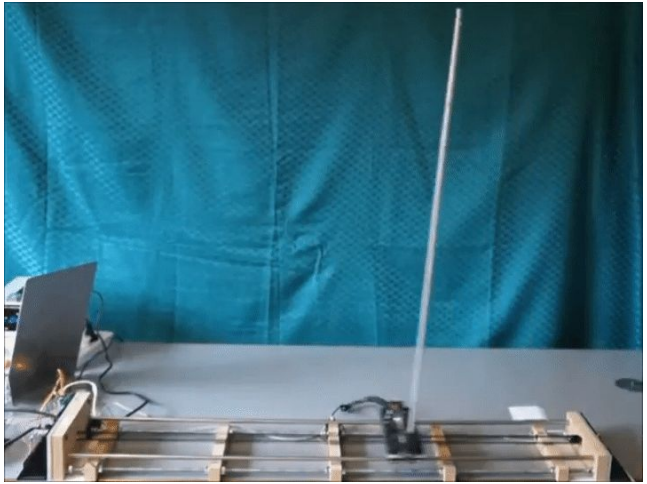
04

EXPERIMENTS

CARTPOLE

CARTPOLE

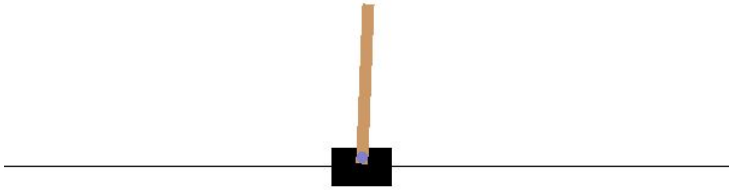




The CartPole environment is a **classic control task** used in reinforcement learning research. It has a pole balanced on a cart, and the goal is to learn a policy to keep the pole **upright**, by applying forces in the **left and right** direction on the cart, for as long as possible.

CARTPOLE - v1

- **OpenAI Gym** is a standard API for reinforcement learning, and a diverse collection of reference environments.
- This environment **simulation** corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson.
- Cart moves on **frictionless** track.
- Actions (**discrete**) and Observations (**continuous**):



Num	Action
0	Push cart to the left
1	Push cart to the right

Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
3	Pole Angular Velocity	-Inf	Inf

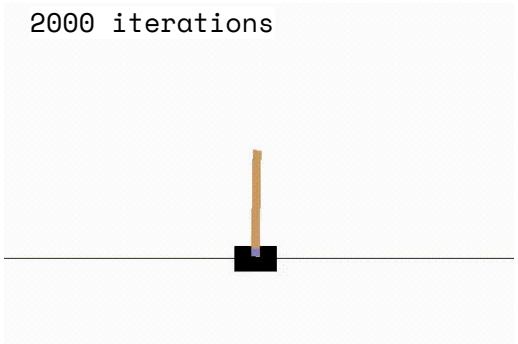
https://www.gymlibrary.dev/

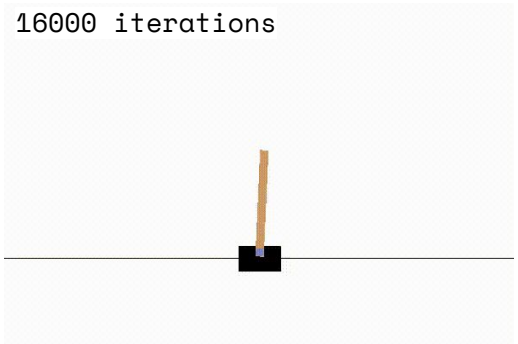
CARTPOLE - v1

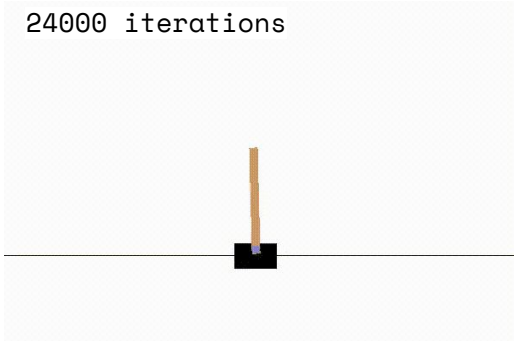
- Since the goal is to keep the pole upright for as long as possible, a reward of `+1` for every step taken, including the termination step, is allotted.
- The episode ends if any one of the following occurs:
  - Pole Angle is greater than **±12°**
  - Cart Position is greater than **±2.4** (center of the cart reaches the edge of the display)
  - Episode length is greater than **500** (200 for v0)

Q LEARNING

- Since the states are continuous, they are **discretized** into 14641 state combinations.
- Expert is trained using Q Learning algorithm with a **Q-Table** of dimension 14641 x 2.
- Expert training is done for **30,000 iterations** within which the expert obtains the goal.
- This algorithm is also used to train apprentices but using irl derived rewards.

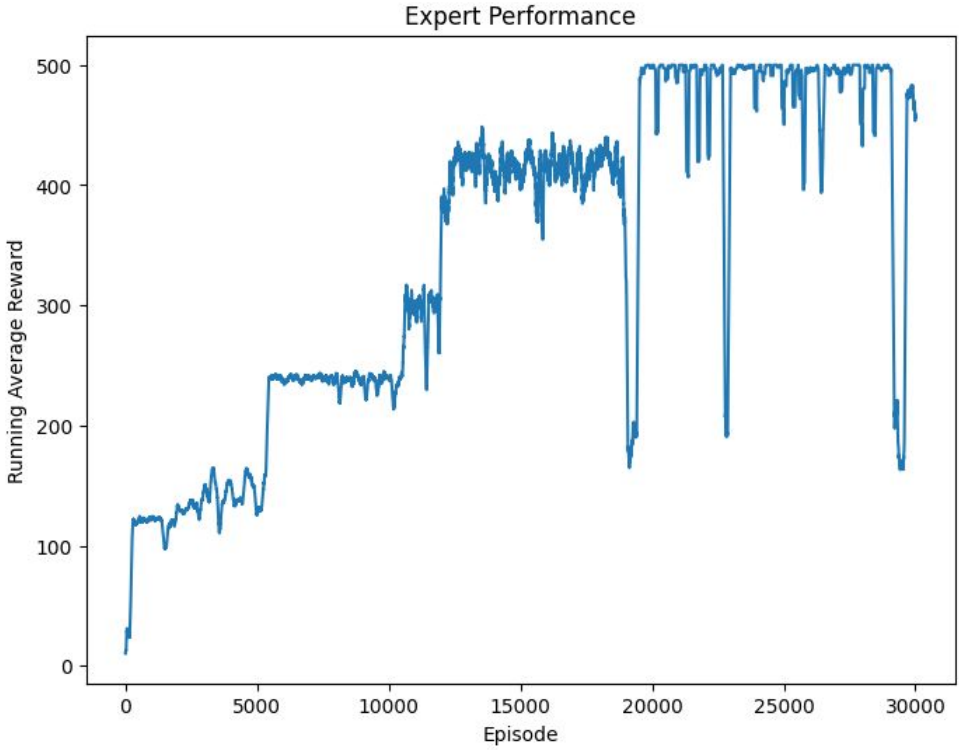




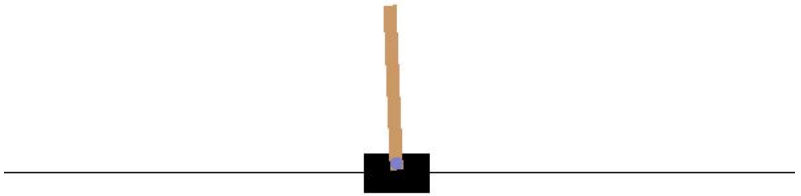


Code: https://github.com/Shaz-5/apprenticeship-IRL

Q Learning - CartPole



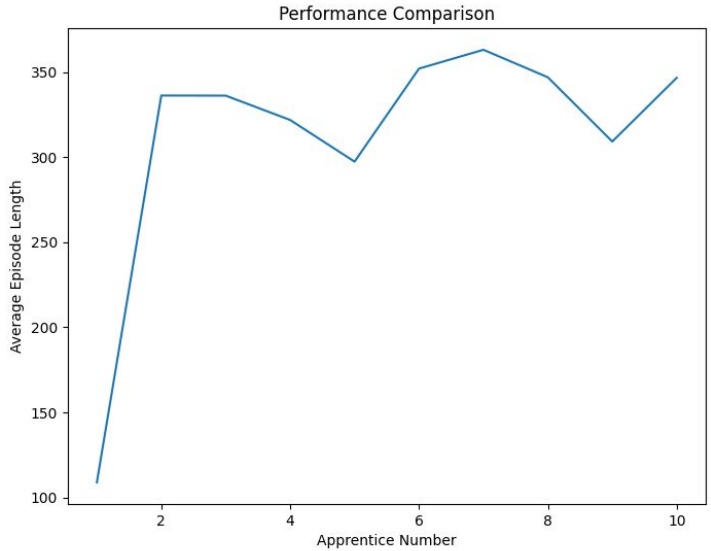
Q Learning Trained Expert

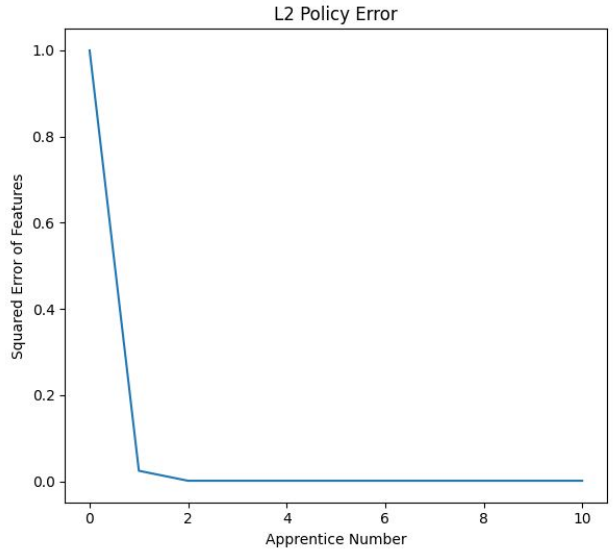


Code: https://github.com/Shaz-5/apprenticeship-IRL

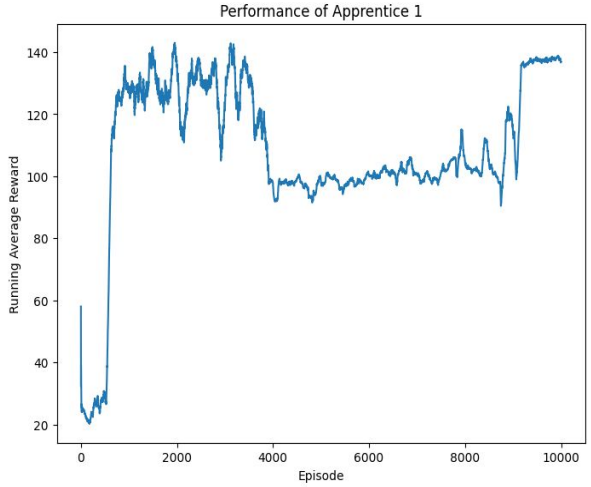
IRL ALGORITHM

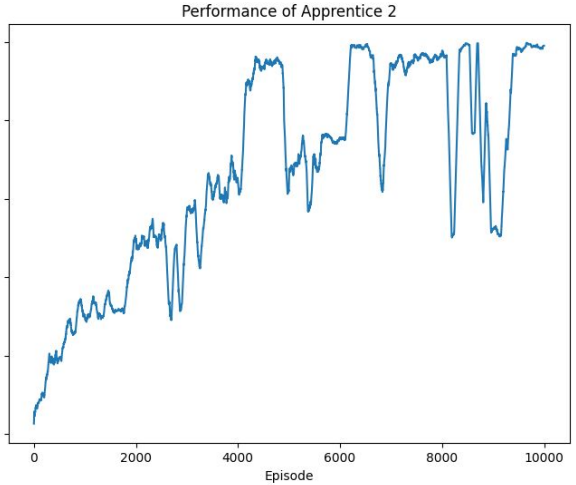
- The IRL **Projection Method Algorithm** is used to train apprentice agents that derive their policy by observing the expert policy.
- At least one of the trained apprentices perform at least as well as the expert within ϵ.

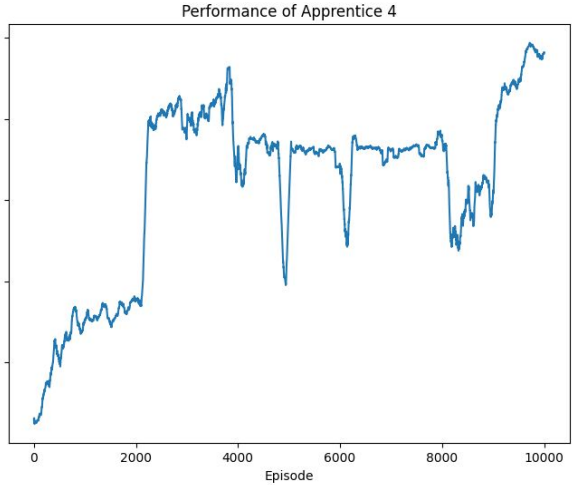


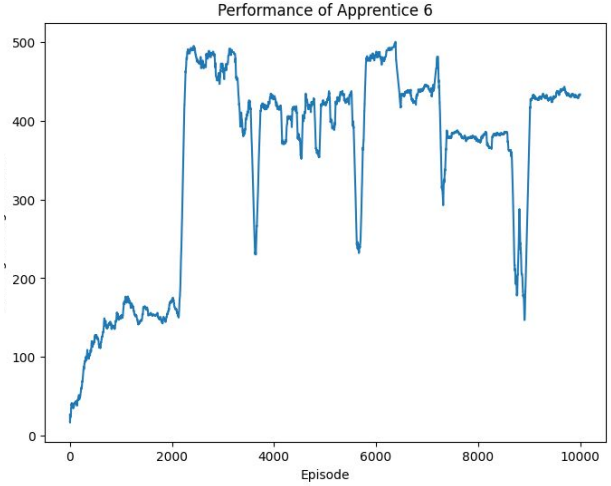


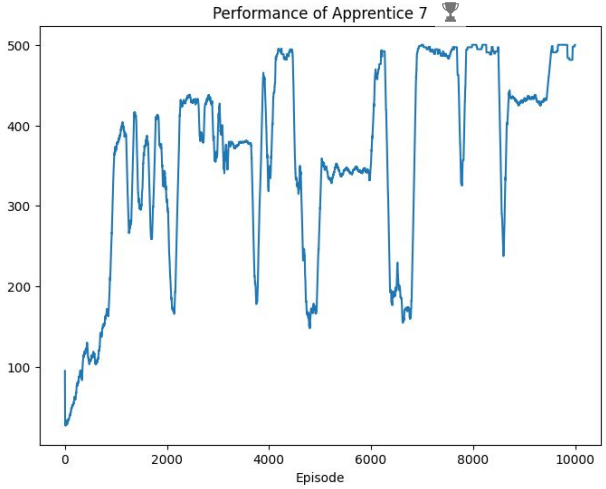
Code: https://github.com/Shaz-5/apprenticeship-IRL

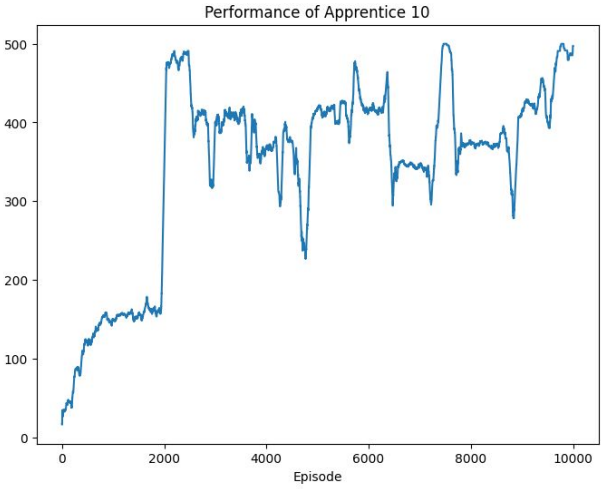






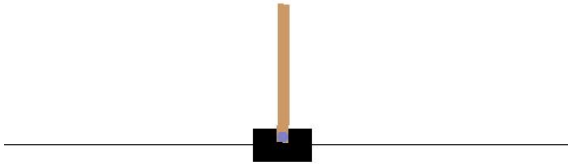




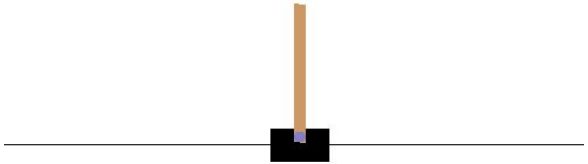


Code: https://github.com/Shaz-5/apprenticeship-IRL

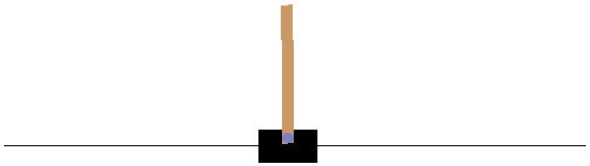
Apprentice 1



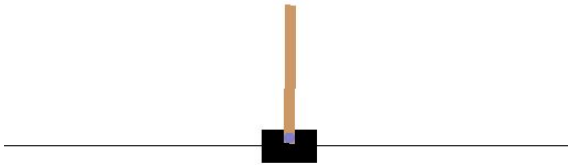
Apprentice 2



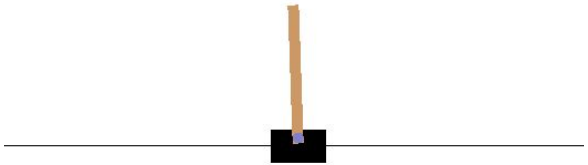
Apprentice 4



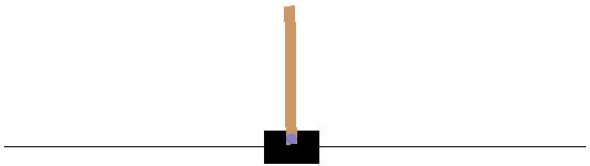
Apprentice 6



Apprentice 7 🏆



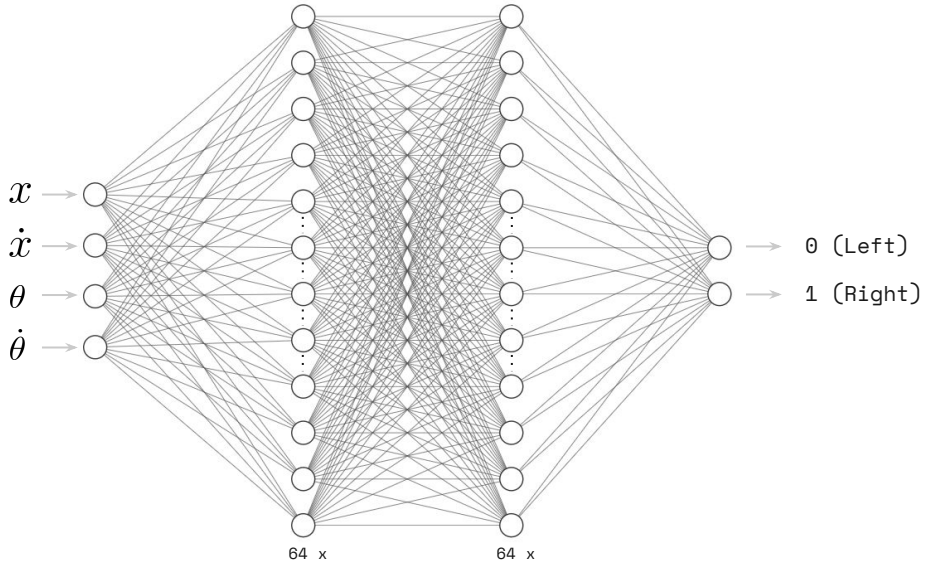
Apprentice 10



Code: https://github.com/Shaz-5/apprenticeship-IRL

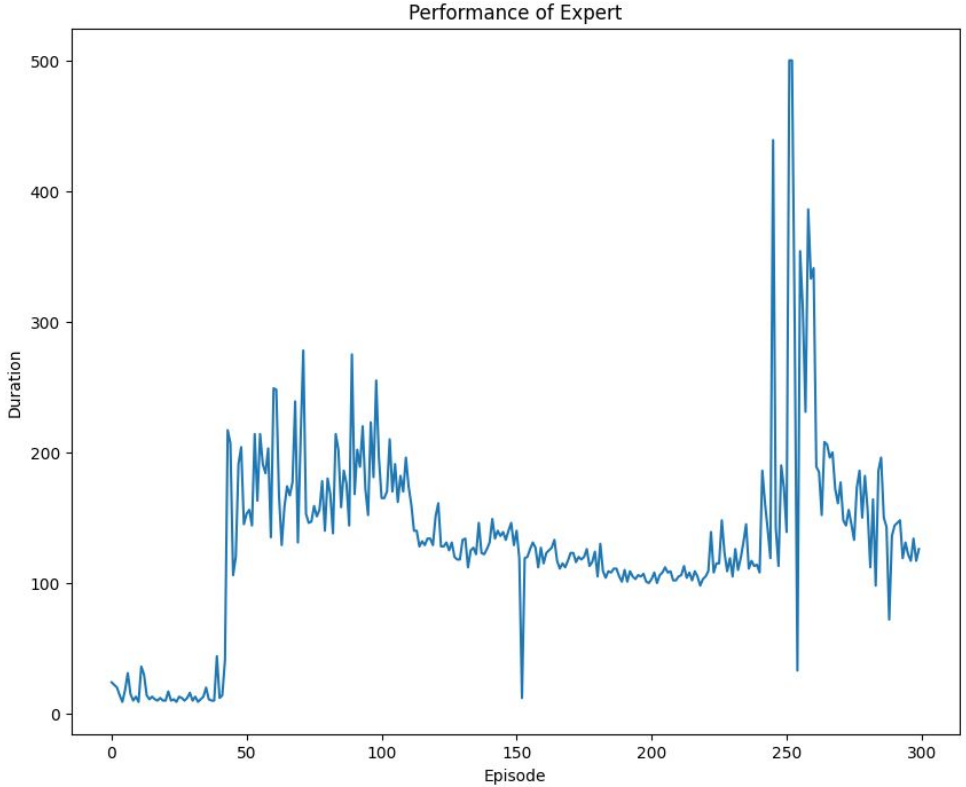
DEEP Q LEARNING

- We utilize **Double Deep Q Learning** to train an expert cartpole agent for **300 iterations**.
- The policy is represented by the **Q-value function** learned by the neural network.
- A **policy network** and **target network** is used in the Double DQN.

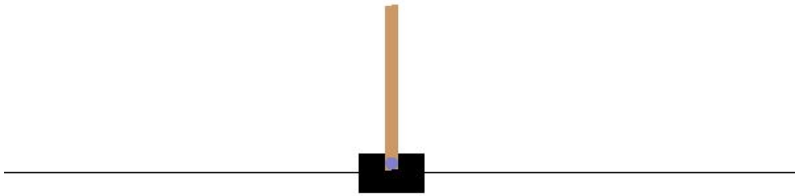


Code: https://github.com/Shaz-5/apprenticeship-IRL

DDQN - CartPole

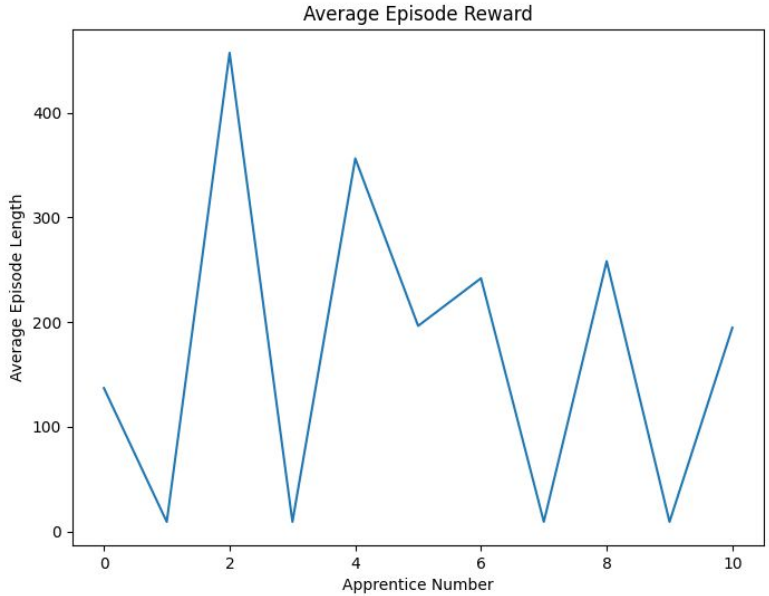


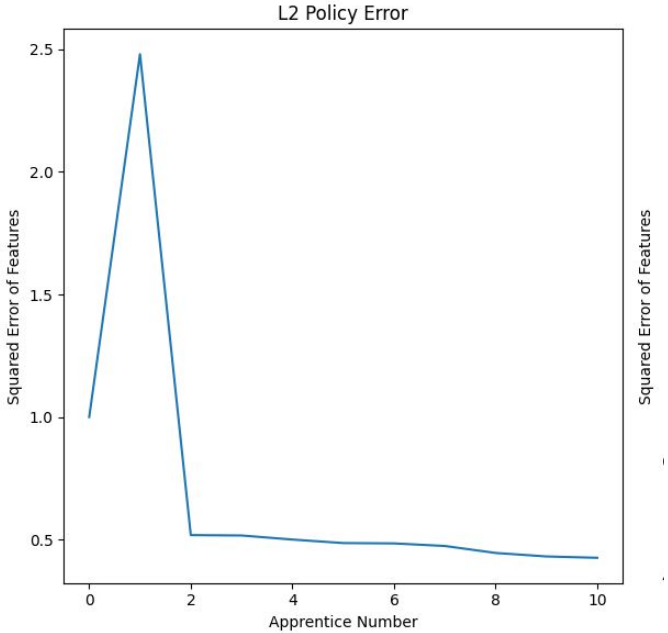
DDQN Trained Expert



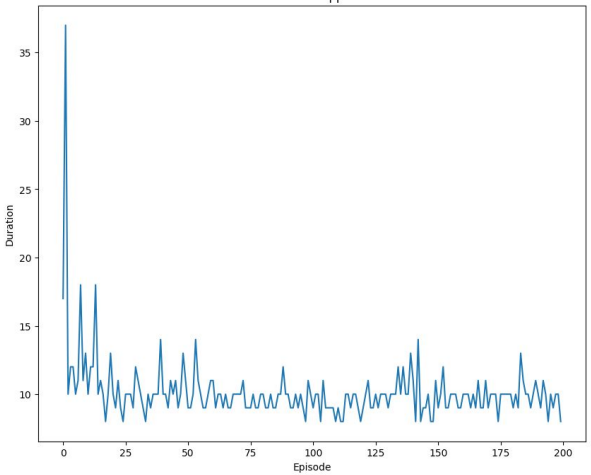
Code: https://github.com/Shaz-5/apprenticeship-IRL

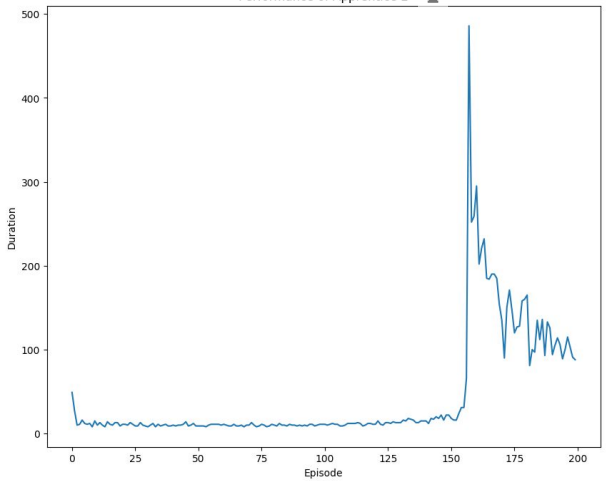
IRL Projection ALGORITHM

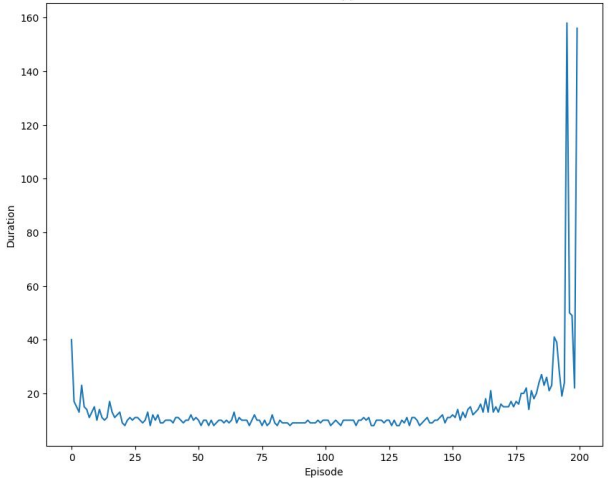


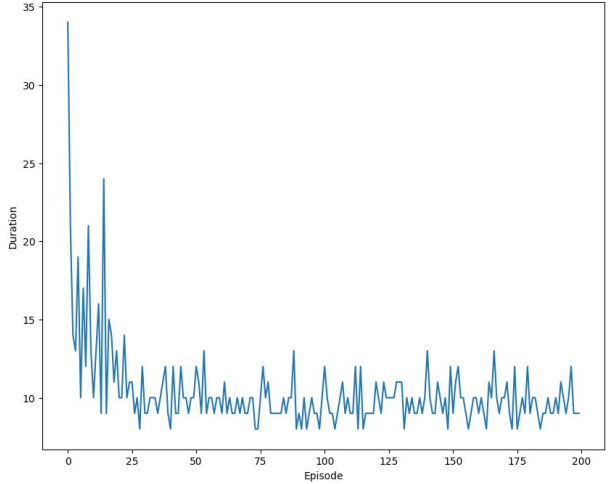


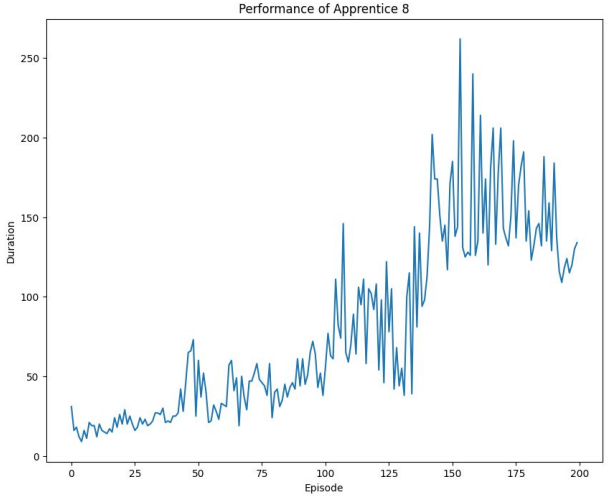
Code: https://github.com/Shaz-5/apprenticeship-IRL

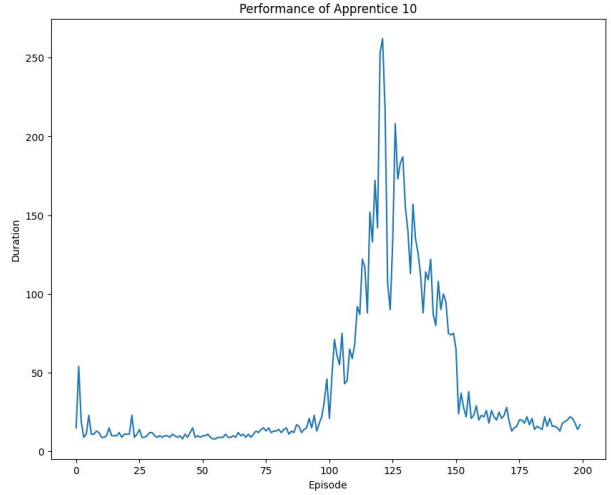






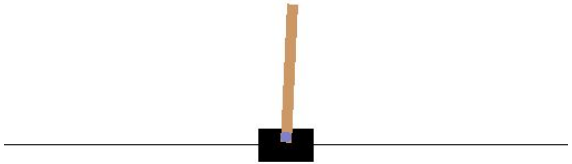




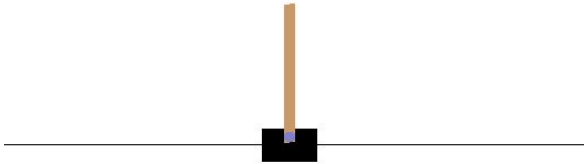


Code: https://github.com/Shaz-5/apprenticeship-IRL

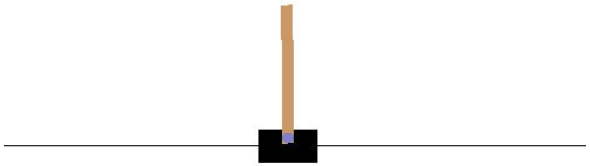
Apprentice 1



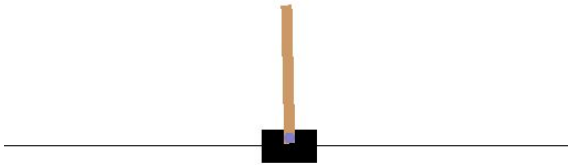
Apprentice 2



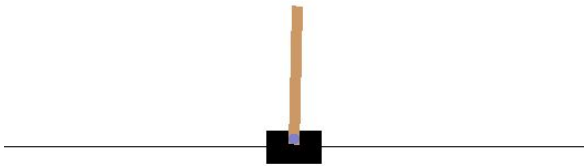
Apprentice 5



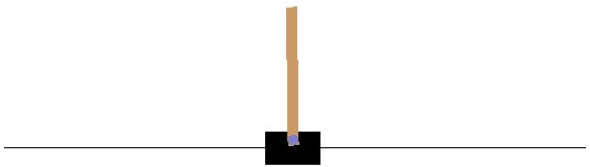
Apprentice 7



Apprentice 8



Apprentice 10

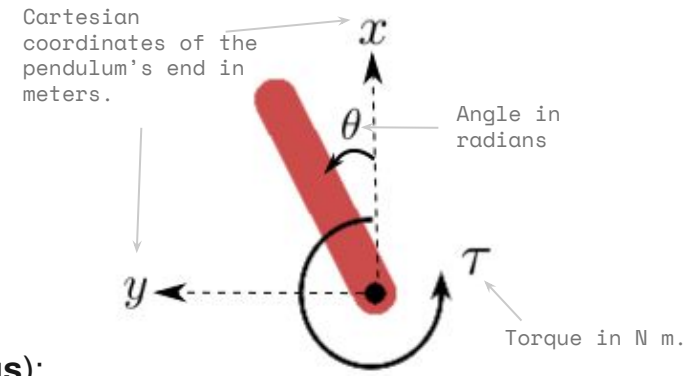


Code: https://github.com/Shaz-5/apprenticeship-IRL

PENDULUM

PENDULUM-v0

- Based on a classic problem in **control theory**.
- The goal is to apply **torque** on the free end to swing it into an **upright position**, with its center of gravity right above the fixed point.
- **g = 9.8**
- Actions (**continuous**) and Observations (**continuous**):



Num	Action	Min	Max
0	Torque	-2.0	2.0

Num	Observation	Min	Max
0	x = cos(theta)	-1.0	1.0
1	y = sin(angle)	-1.0	1.0
2	Angular Velocity	-8.0	8.0

PENDULUM-v0

- The reward function is defined as:

$$r = -(\theta^2 + 0.1 \cdot \dot{\theta}_2 + 0.001 \cdot \tau^2)$$

  - where θ is the **pendulum's angle normalized** between [-π, π] (with 0 being in the upright position).

- The **maximum reward** is **zero** (pendulum is upright with zero velocity and no torque applied).

- The starting state is a random angle in [-π, π] and a random angular velocity in [-1,1].

- The episode ends at **200** time steps.

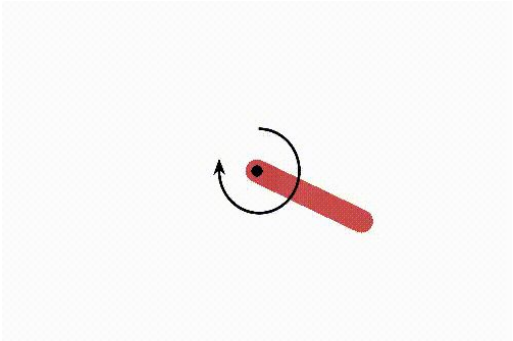
Q LEARNING

- Since the states and actions are both continuous, they are **discretized** into **[21, 21, 65]** state combinations and **9** actions.
- Expert is trained using Q Learning algorithm for **40,000 iterations** within which the expert obtains the goal.

4000 iterations

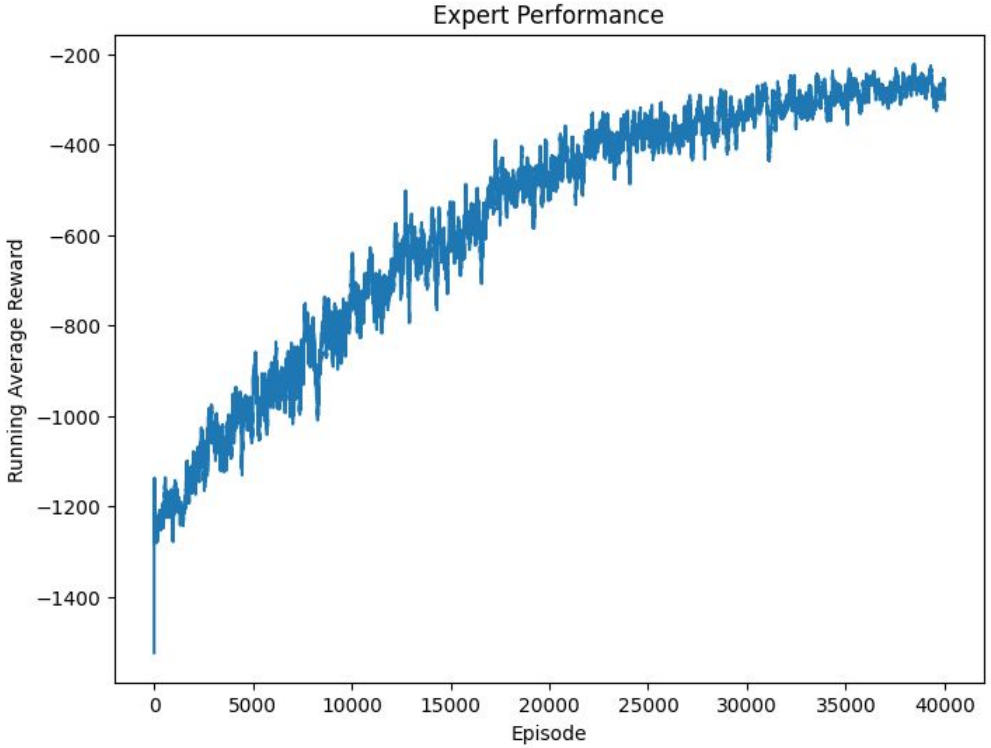
20000 iterations

36000 iterations

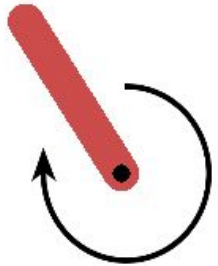


Code: https://github.com/Shaz-5/apprenticeship-IRL

Q Learning - Pendulum

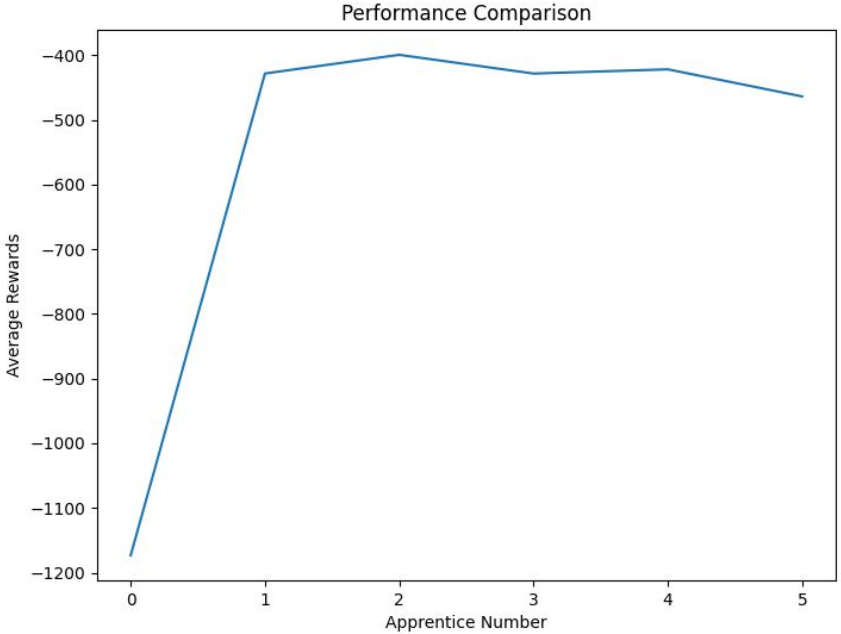


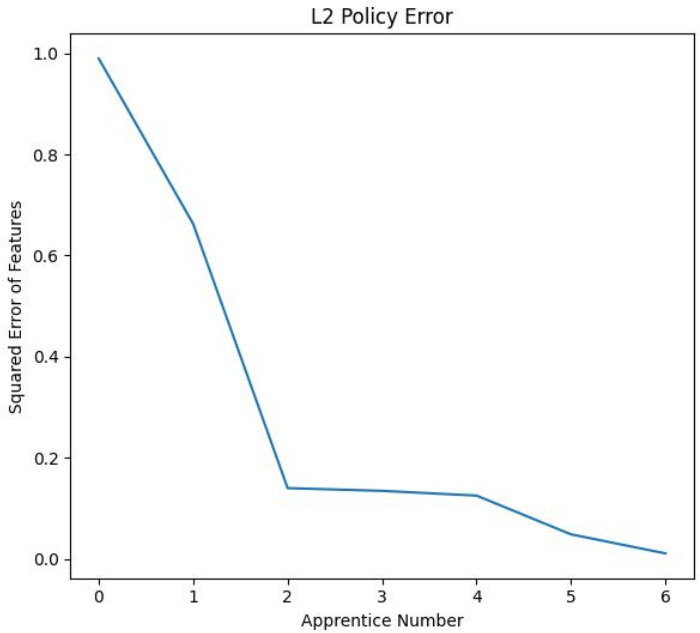
Q Learning Trained Expert



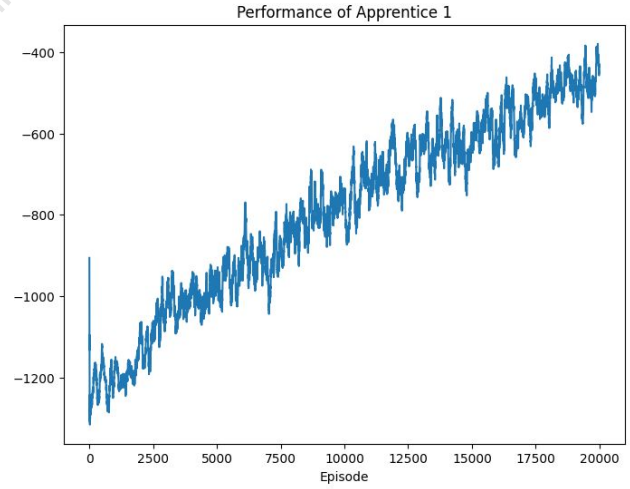
Code: https://github.com/Shaz-5/apprenticeship-IRL

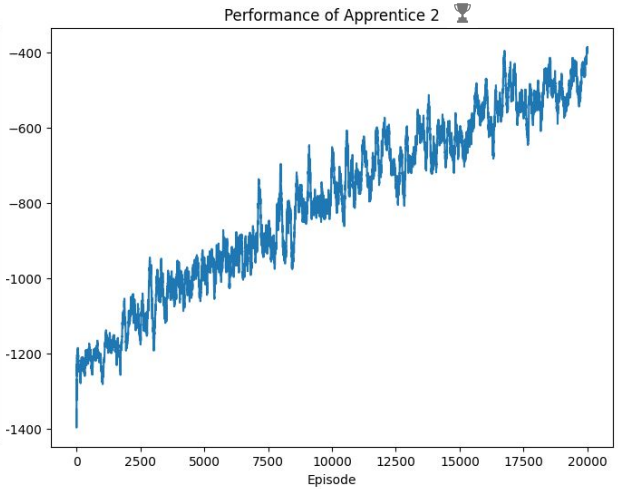
IRL Projection ALGORITHM

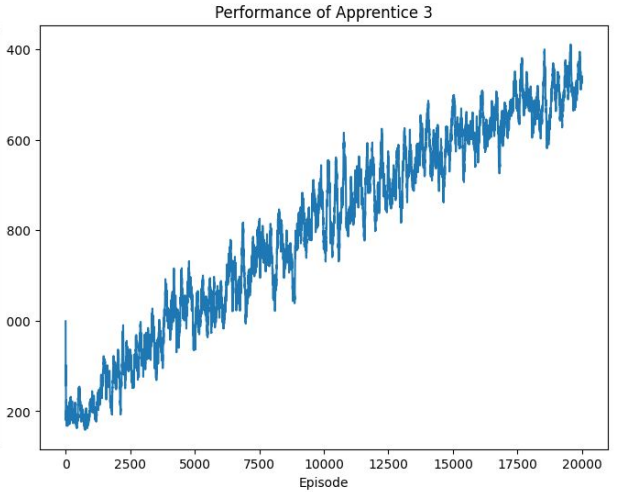


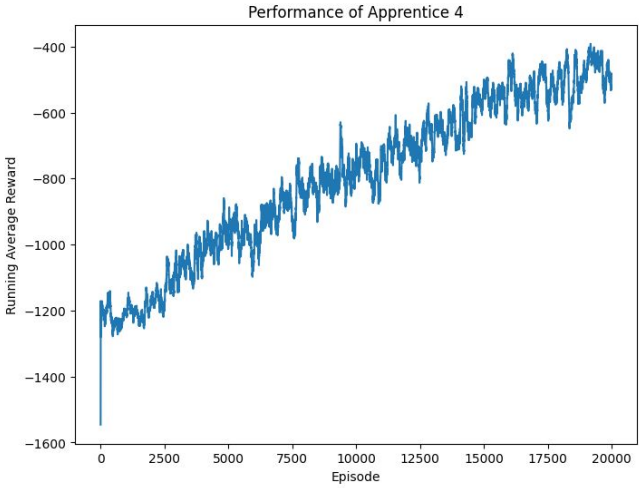


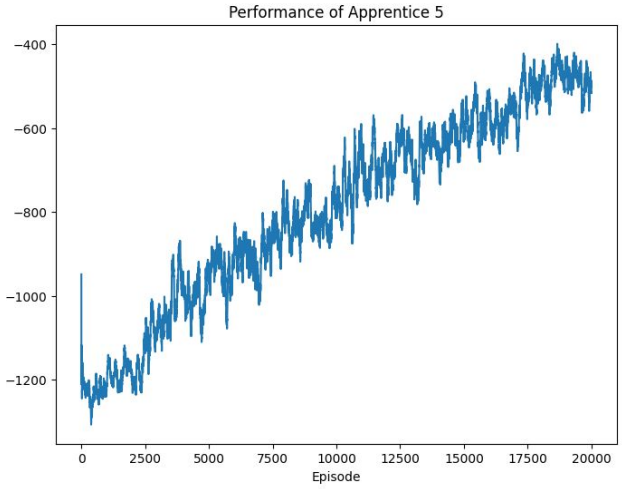
Code: https://github.com/Shaz-5/apprenticeship-IRL





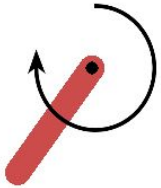




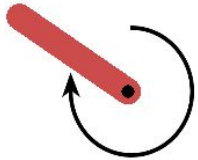


Code: https://github.com/Shaz-5/apprenticeship-IRL

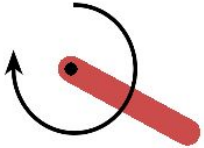
Apprentice 1



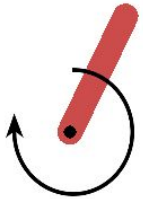
Apprentice 2



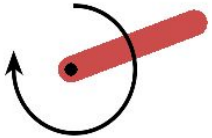
Apprentice 3



Apprentice 4



Apprentice 5

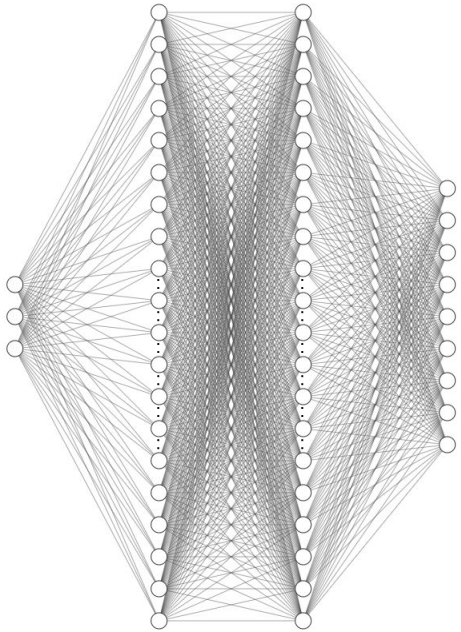


DEEP Q LEARNING

- The continuous actions are discretized into 9 actions.
- Expert agent is trained using **Double Deep Q Learning** for **300 iterations**.

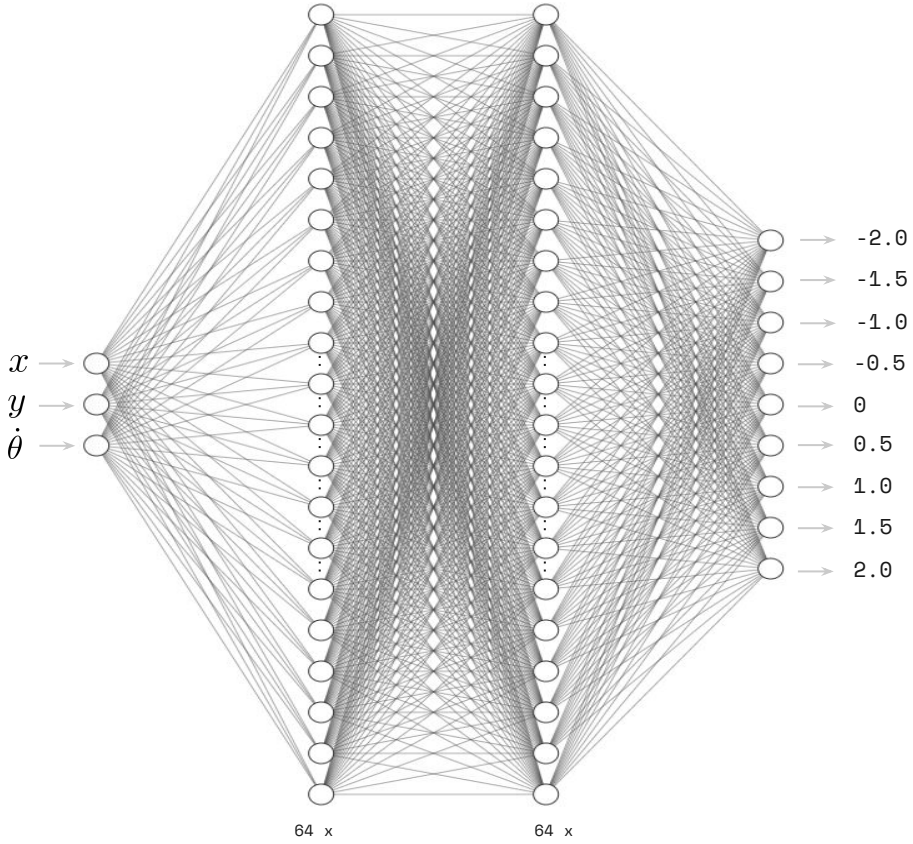
```python
# Neural Network for (Double) Deep Q Learning (DQN)
class DQN(nn.Module):
    def __init__(self, n_observations=3, hidden_layer_size=64, n_actions=9):
        super(DQN, self).__init__()
        self.input_layer = nn.Linear(n_observations, hidden_layer_size)
        self.hidden_layer_1 = nn.Linear(hidden_layer_size, hidden_layer_size)
        self.hidden_layer_2 = nn.Linear(hidden_layer_size, hidden_layer_size)
        self.output_layer = nn.Linear(hidden_layer_size, n_actions)

    def forward(self, x):
        x = F.relu(self.input_layer(x))
        x = F.relu(self.hidden_layer_1(x))
        x = F.relu(self.hidden_layer_2(x))
        x = self.output_layer(x)
        return x
```



Code: https://github.com/Shaz-5/apprenticeship-IRL

DEEP Q LEARNING



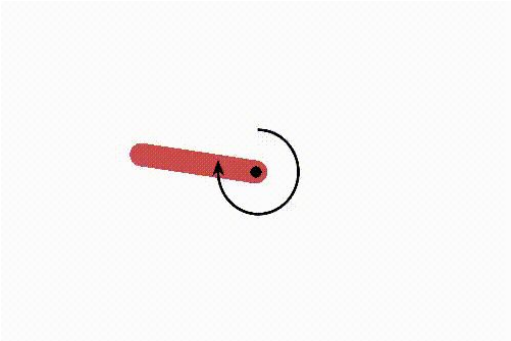
Code: https://github.com/Shaz-5/apprenticeship-IRL

DEEP Q LEARNING

- The continuous actions are discretized into 9 actions.
- Expert agent is trained using **Double Deep Q Learning** for **300 iterations**.

30 iterations

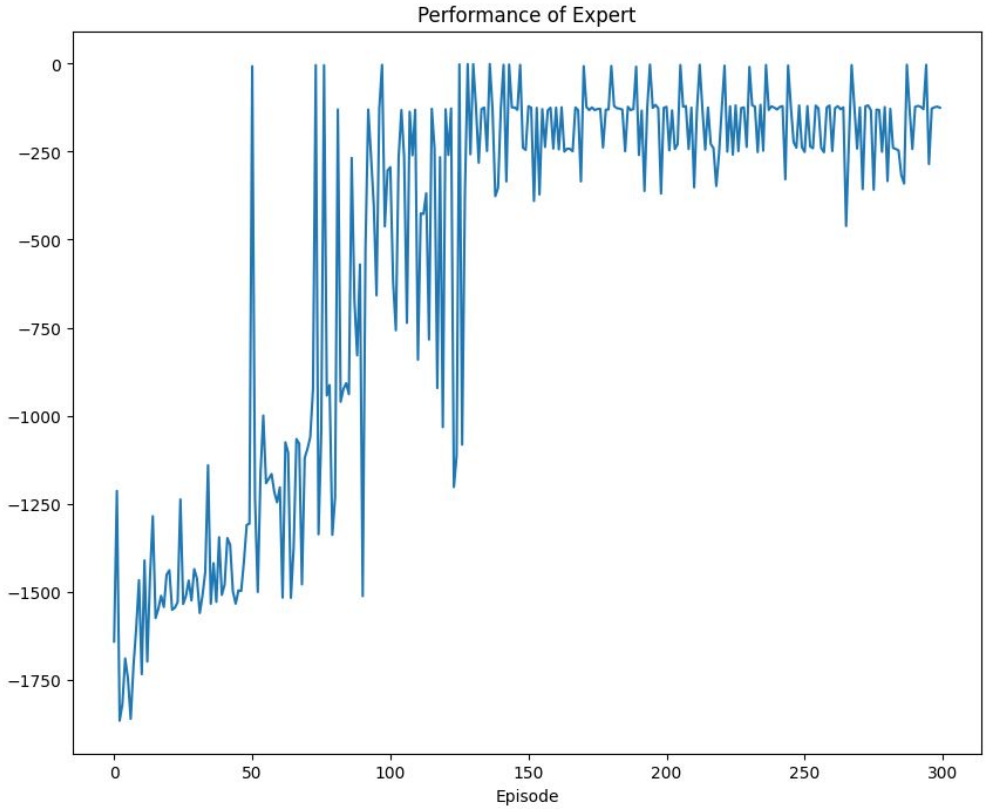
150 iterations



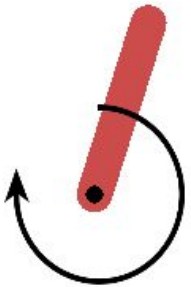
270 iterations

Code: https://github.com/Shaz-5/apprenticeship-IRL

DDQN - Pendulum

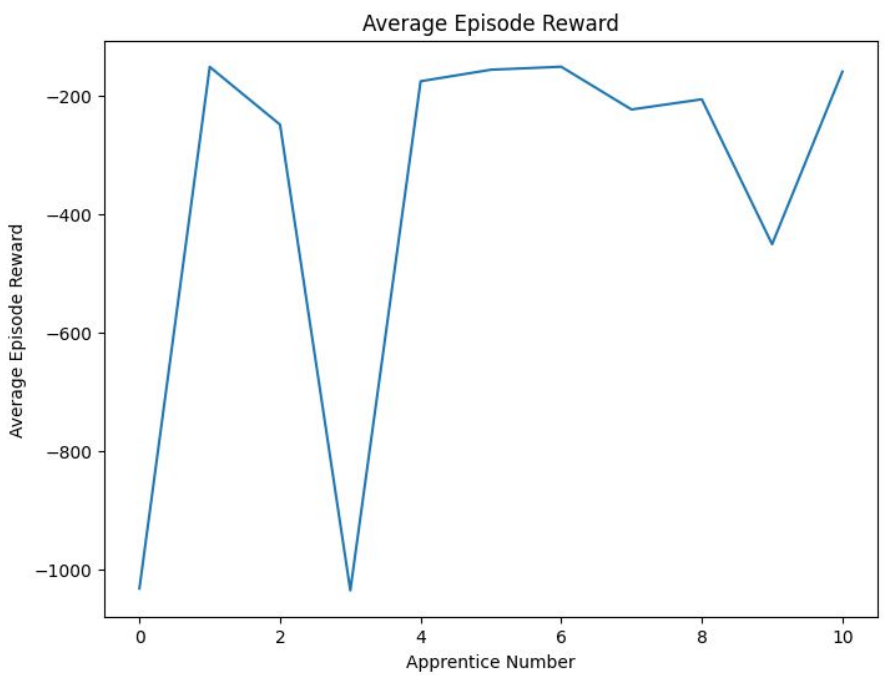


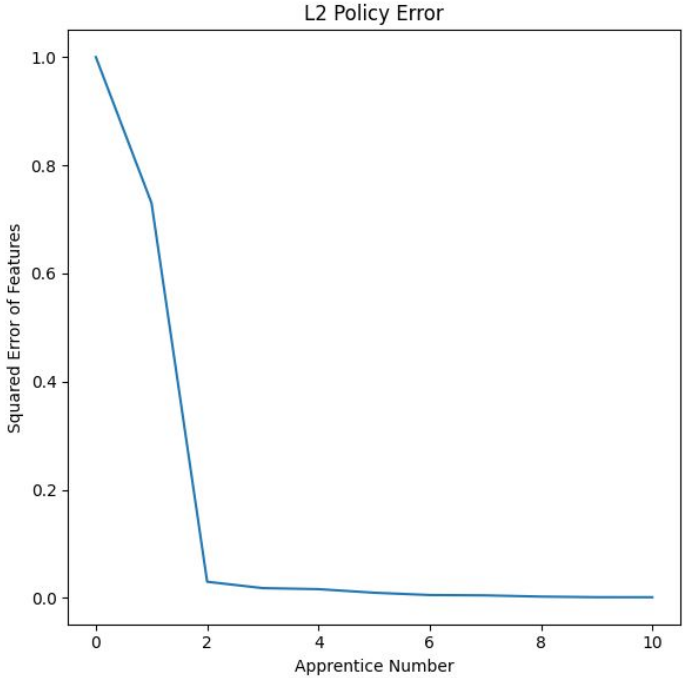
DDQN Trained Expert



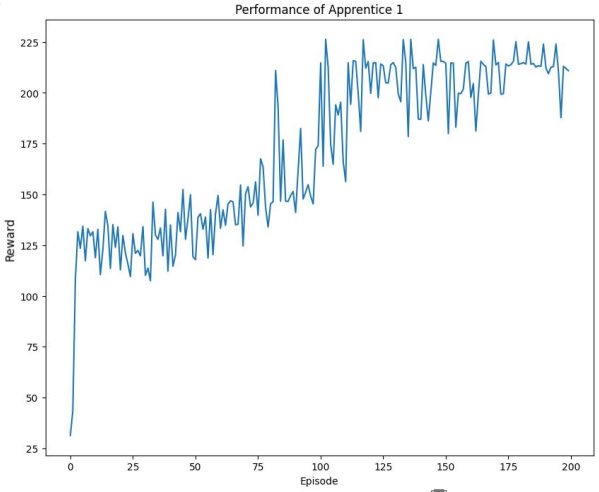
Code: https://github.com/Shaz-5/apprenticeship-IRL

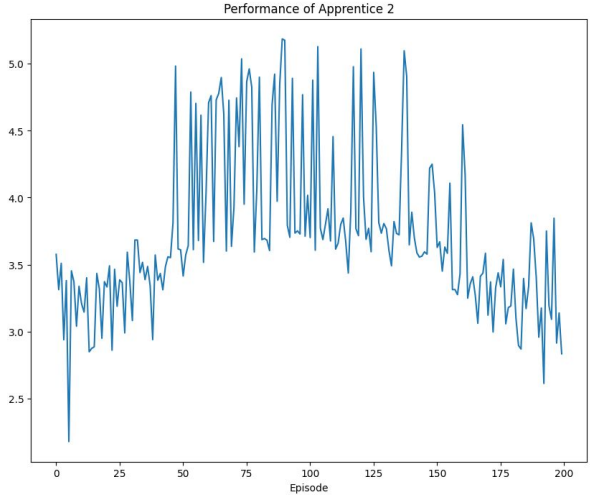
IRL Projection ALGORITHM

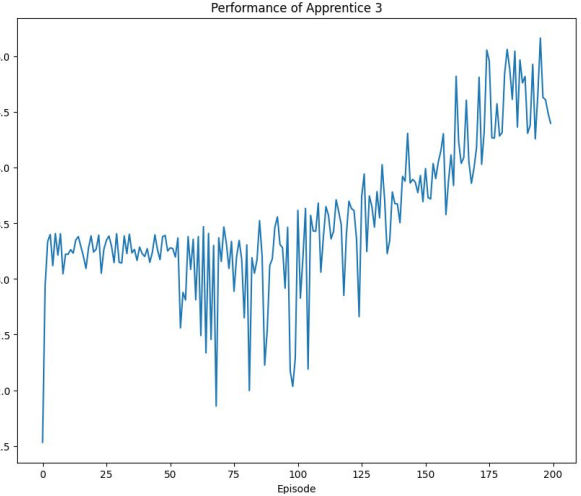


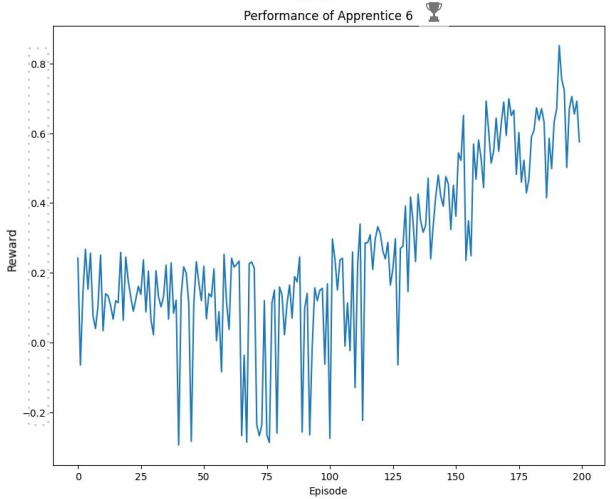


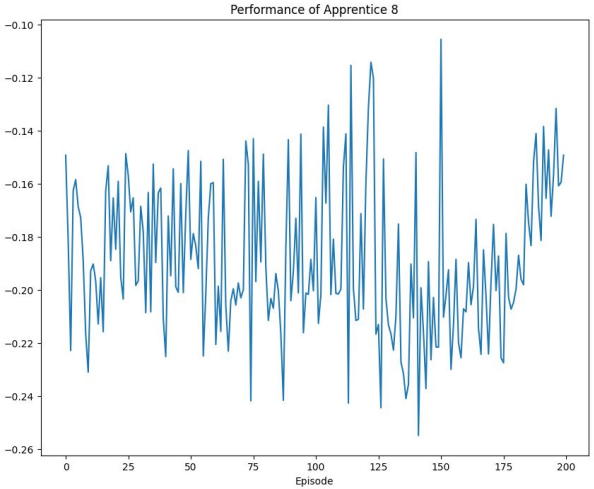
Code: https://github.com/Shaz-5/apprenticeship-IRL

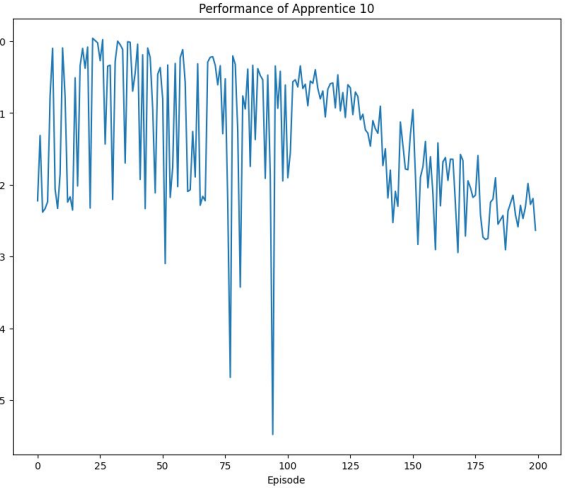






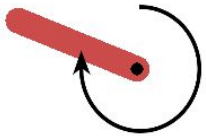




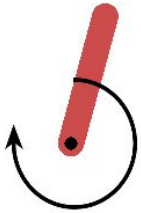


Code: https://github.com/Shaz-5/apprenticeship-IRL

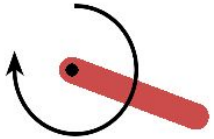
Apprentice 1



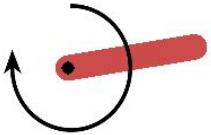
Apprentice 2



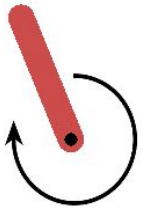
Apprentice 3



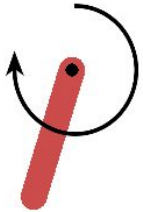
Apprentice 6 🏆



Apprentice 8



Apprentice 10



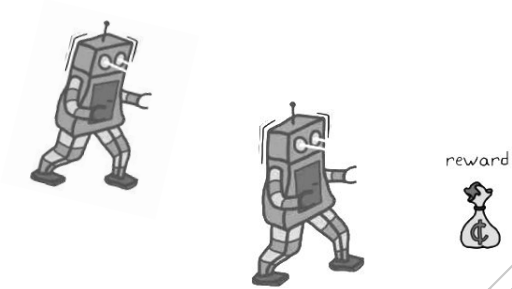
Code: https://github.com/Shaz-5/apprenticeship-IRL

05

CONCLUSION

Conclusion

- Apprenticeship learning via Inverse Reinforcement Learning offers a promising approach for agents to **learn from expert** demonstrations.
- By focusing on matching long-term effects on state features (using feature expectations), the algorithm achieves performance comparable to the expert on an unknown reward function.
- The expert's reward function is taken as a **linear combination of known features** and a sufficient set of features allows for a good approximation of the reward function.
- This method proves to be fairly efficient, requiring few iterations.
- Future research could explore extending this approach to more complex nonlinear reward functions or settings with richer feature spaces.
- Overall, this method helps in developing agents that can effectively learn from expert guidance.



THANK YOU!

References:

1. Abbeel, P. & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning.
2. Hugging Face Deep Reinforcement Learning Course - Hugging Face. (n.d.).
3. Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, & Wojciech Zaremba. (2016). OpenAI Gym.
4. Reinforcement Learning (DQN) Tutorial - PyTorch Tutorials 2.2.1+cu121 documentation. (n.d.).