Imitation Learning

Introduction

- In reinforcement learning, it is sometimes difficult or impossible to compute exact rewards.
  - In driving a car, the exact reward for a specific driving operation is unclear
  - If you are off the road, you can not perform any action and thus can not compute rewards
  - In patient treatments, it is hard to represent in numbers (rewards) how effective this treatment is.

- Sometimes, the accurate computation of the reward is often more useful than the policy itself.

- **Imitation Learning**: find the optimal policy by using a demonstration of the correct behavior from experts.

- When an expert provides a pilot action (pilot data) for a given problem, imitation learning learns the best policy based on them.

Introduction

- There are two kinds of Imitation Learning:
- Direct method:
  - Learning policies using supervisor learning methods from expert data. (e.g., behavioral cloning)
  - This form of imitation learning learns policies directly from expert data, (no need to know the reward value.)

- Indirect method:
  - Compute a reward value function from the expert's data
  - After that, compute a policy from the reward function (using ordinary reinforcement learning)
  - Ordinary reinforcement learning is part of the method.
  - aka inverse reinforcement learning

Introduction

- Imitation learning is a model that has state (S), action (A), but does not have reward (R).

- Instead of rewards, imitation learning has **expert data**.
- Expert's data consists of $(s_0, a_0), (s_1, a_1), \ldots, (s_n, a_n)$
  - (s_i, a_i) represents the expert action a_i in the state s_i.

- Imitation learning finds effective (optimal) policies from these expert data.

Behavioral Cloning

- Behavioral cloning is the simplest form of imitation learning and it mimicking the actions of experts.

- Suppose a_i^* is an optimal action in s_i (given by expert), expert data are given as follows:

$$(s_0, a_0^*), , (s_1, a_1^*), \dots, (s_n, a_n^*)$$

- Since each data is given a target value, behavioral cloning is a classification problem.
  - If a_i^* is continuous value, it is a regression problem.

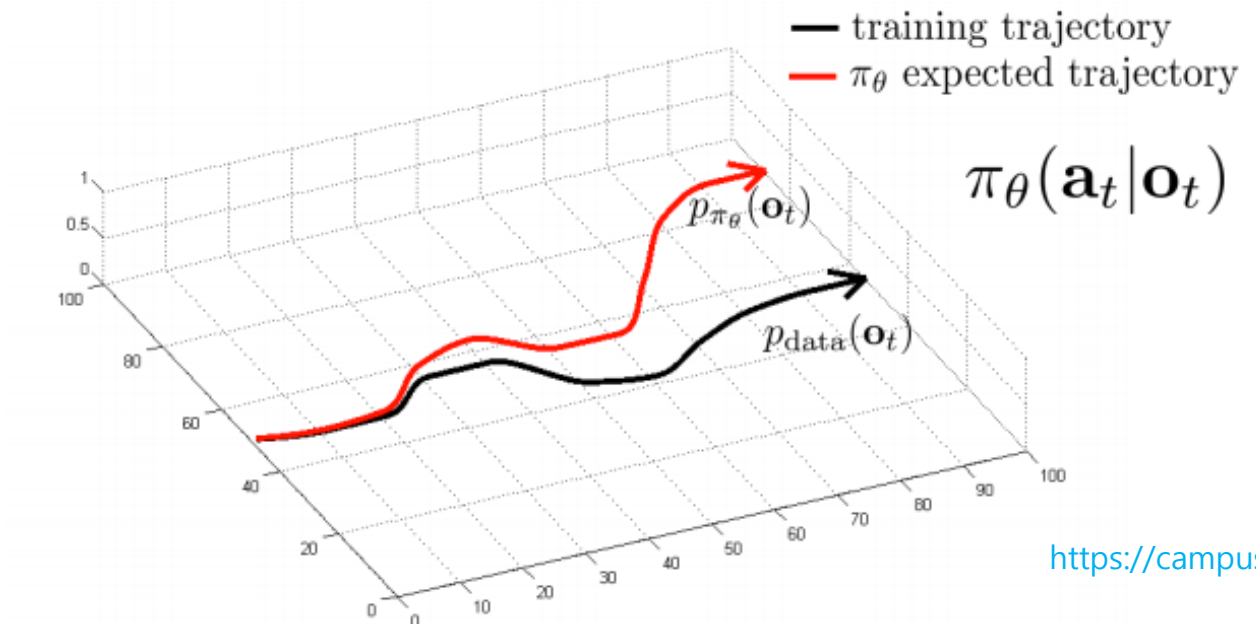
- Behavior cloning minimizes the following error function (supervised learning)

$$\underset{\theta}{\operatorname{argmin}} \underbrace{E_{(s,a^*)\sim P(s|\pi_*)}}_{\text{mean over expert actions}} [\underbrace{Loss(a^*, \underbrace{\pi_\theta(s)}_{\text{action from behavior cloning}})}_{\text{error between actions from policy } \pi_\theta \text{ and expert}}]$$

Behavioral Cloning

- Behavioral cloning is a simple algorithm, but has the problem of data distribution mismatch.

- It can work well in an environment similar to the expert's data (because the data already exists)
- In a new environment (no existing data), it generates error and these errors are amplified.
  - A small mistake makes the subsequent observation distribution to be different from the training data. This snowball effect keeps rising the error between trajectories over time:
  - When the size of error in current step is bounded by ϵ. The size of error in T time step is $O(\epsilon T^2)$



https://campusai.github.io/lectures/lecture2

DAGGER

- Dataset AGGregation:
- Interactive expert can give labels for any state the agent experiences.
- In DAGGER, experts continuously add target values for new data.
- Get more labels of the expert action in the new states along the policy computed by BC.
- By doing so, the algorithm allows experts to match the distribution of expert data with the distribution of data generated by the algorithm
- Obtains a policy with good performance under its induced state distribution.

- However, DAGGER has to perform actual action with a policy that is NOT yet complete.
  - For example, while driving a car, you may act the wrong way.
  - Requiring a lot of expert intervention at each learning stage.
- Although the DAGGER method is simple, it has been applied to real problems such as autonomous driving and has significant effects

DAGGER

- **Pseudo code:**
  1) create expert dataset $D_{\pi_*} = \{(s_0, a_0), \dots, (s_N, a_N)\}$
  2) train policy $\pi_\theta(a|s)$ using supervised learning with data D_{π_*}
  3) create a new state data $D_\pi = \{s_0, s_1, \dots, s_M\}$ using the trained policy $\pi_\theta(a_t|s_t)$
  4) experts create correct action a_t for each state s_t in D_π.
     we have a new dataset $D_\pi = \{(s_0, a_0), \dots, (s_M, a_M)\}$
  5) $D_{\pi_*} \leftarrow D_{\pi_*} \cup D_\pi$
  6) repeat step 2)

- In some Dagger algorithm, in step 2) of the pseudo code, the current policy is a mixture of the old policy $\pi_\theta(a_t|s_t)^{(n-1)}$ and a new policy $\pi_\theta(a_t|s_t)^{(n)}$ (step n)

$$\pi_\theta(a|s) = (1-\alpha)\pi_\theta(a_t|s_t)^{(n-1)} + \alpha\,\pi_\theta(a_t|s_t)^{(n)}$$

Inverse Reinforcement Learning

- DAgger and other IRL algorithms need data from human, which is finite and expensive
  - Deep learning works best when data is plentiful
- Can they do better than the human expert?
  - Humans are not good at providing some kinds to actions
  - Combine of IL and RL?

- Inverse reinforcement learning: does not directly learn behavior from expert data
  - predict the **reward function** first using expert data
  - and then obtain **a policy** using these reward values (ordinary reinforcement learning)
- The goal of inverse reinforcement learning is to obtain the reward values from the expert data. (Using the reward value calculated, the optimal policy is obtained)

Inverse Reinforcement Learning

- In inverse reinforcement learning, the MDP model has state (S), behavior (A), and discount factor, but does not have reward value (R). Instead, it contains the following expert data.

$$\left\{\left(s_0^{[1]}, a_0^{[1]}, s_1^{[1]}, a_1^{[1]}, \dots\right), \left(s_0^{[2]}, a_0^{[2]}, s_1^{[2]}, a_1^{[2]}, \dots\right), \dots\right\}$$

Reinforcement Learning

MDP
- state : $s \in S$
- action : $a \in A$
- (transition prob. : $\Pr(s_t|s_{t-1}, a_{t-1})$)
- **reward : $\boldsymbol{r} \in \mathbb{R}$**

Goal : optimal policy π_*

Inverse Reinforcement Learning

MD without R
- state : $s \in S$
- action : $a \in A$
- (transition prob. : $\Pr(s_t|s_{t-1}, a_{t-1})$)
- **expert data : $\left(\boldsymbol{s_0^{[1]}}, \boldsymbol{a_0^{[1]}}, \boldsymbol{s_1^{[1]}}, \boldsymbol{a_1^{[1]}}, \dots\right)$**

Goal: Rewars function($\boldsymbol{R}$)

Inverse Reinforcement Learning

- In behavioral cloning and DAGGER, it is not possible to find the optimal policy for new states due to data distribution discrepancies.
- If the reward values can be learned as a function, the reward value for a new state can be predicted.

- Inverse reinforcement learning has possibilities for transfer learning.
- For example, in the path detection problem, reward values learned in a specific city can be used (with only a little additional learning) in other environments (other cities) with similar features of the reward value function.

Inverse Reinforcement Learning

- The basic idea of inverse reinforcement learning is to obtain a reward function by alternating the following two steps.

  1) Obtain the reward function: Update the parameter w of the reward value function to modify the reward value
  2) Computing Policy: Policy π is computed using the reward value in Step 1)

Feature Matching

- The performance of policy π is defined as the state value function of initial state s_0

$$V_\pi(s_0) = E_\pi\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)\right] \quad (1)$$

- Suppose π_* is the policy of expert. We must find the reward value function that makes the expert policy better than any other policy.

- In other words, the reward value function we obtain in inverse reinforcement learning must satisfy the following characteristics.
  - The state value function of expert policy is better than (or equal to) that of other policies.

$$V_{\pi_*}(s_0) \geq V_\pi(s_0), \forall\pi \quad (2)$$

Feature Matching

- By combining (1) and (2)

$$E_{\pi_*}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)\right] \geq E_{\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)\right], \forall\pi$$

- Suppose reward function is represented as a linear function of features.

$$r(s) = \sum_i w_i \emptyset_i(s) = w^T \emptyset(s)$$

where $w_i \in R^n$ represents the weight of feature.

Linear Reward Function

- In previous slide, we assumed the reward function is a linear function of features.
- There is an advantage to using a nonlinear reward function (such as a neural network), as it can better represent the reward values.
- The number of expert data points is usually limited, making it difficult to learn complex reward functions from this data.
- Furthermore, methods like apprenticeship learning in inverse reinforcement learning involve multiple iterations of reinforcement learning
- Therefore, when using complex reward functions, it takes a lot of time to train inverse reinforcement learning methods.

- Even though the reward function is expressed as a linear function, the features can be represented as nonlinear combinations of features, allowing it to be applied to nonlinear problems.

- Since we assumed $r(s) = w^T \emptyset(s)$,

$$V_\pi(s_0) = E_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t)] = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t\, w^T \emptyset(s)\right] = w^T E_\pi[\sum_{t=0}^{\infty} \gamma^t\, \emptyset_i(s_t)] \qquad (3)$$

Feature Matching

- Suppose $\mu_i(\pi)$ is the mean value of feature i (based on policy π).

$$\mu_i(\pi) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t \emptyset_i(s_t)\right]$$

- Therefore,

$$V_\pi(s_0) = w^T \mu(\pi) \qquad (4)$$

- It shows that the performance of a policy is a linear function of the mean value of the features.

- Combining formula (2) and (4), we have

$$w^T \mu(\pi_*) \geq w^T \mu(\pi), \forall \pi \qquad (5)$$

- The goal of inverse reinforcement learning is to find a weight vector w such that the performance of the expert policy ($wT\mu(\pi*)$) is better than the performance of any other non-expert policies ($wT\mu(\pi)$).

Feature Matching

- There are many reward functions that satisfy $w^T\mu(\pi_*) \geq w^T\mu(\pi), \forall\pi$
  1) If the reward values are the same for all states (e.g., all w values are 0, making the reward 0), the above condition is easily satisfied.
  2) the number of policies that satisfy the equation is nearly infinite

- **Feature Matching Theorem**: if the following satisfies
$$\| \mu(\pi_*) - \mu(\pi) \|_1 \leq \epsilon, \qquad \| w \|_1 \leq 1$$
the following is also true:
$$\left|w^T\mu(\pi_*) - w^T\mu(\pi)\right| \leq \epsilon \quad \forall w \qquad (6)$$

- **Proof:** From formula (4) and (5),
$$\left|V_{\pi_*}(s_0) - V_\pi(s_0)\right| = \left|w^T\mu(\pi_*) - w^T\mu(\pi)\right|$$
Due to Cauchy–Schwarz inequality.
$$\left|w^T\mu(\pi_*) - w^T\mu(\pi)\right| \leq \|w\|_2\|\mu(\pi_*) - \mu(\pi)\|_2$$
Since $\| w \|_1 \leq 1$ and we know that $\| w \|_2 \leq \| w \|_1 \leq \epsilon$.
$$\| \mu(\pi_*) - \mu(\pi) \|_2 \leq \| \mu(\pi_*) - \mu(\pi) \|_1 \leq \epsilon \qquad (7)$$
Therefore,
$$\|w\|_2\|\mu(\pi_*) - \mu(\pi)\|_2 \leq 1 \cdot \epsilon \qquad (8)$$
From (7) and (8), formula (6) holds

Feature Matching

- Feature matching theorem implies that:
- If the feature mean vector $\mu(\pi)$ calculated by a policy π is very similar to the feature mean vector $\mu(\pi_*)$ from expert data, the performance difference between these policies is also very close.

- When comparing the performance difference of policies, we only need to compare the differences in their feature mean vectors.
- The policy we are looking for is now the one whose feature mean vector is most similar to that of the expert policy.

$$\left|w^T\mu(\pi_*) - w^T\mu(\pi)\right| \leq \epsilon$$

Apprenticeship Learning

- One method to find this policy is Apprenticeship Learning.
- The apprenticeship learning method finds a policy π that generates the performance $w^T\mu(\pi)$ closest to the performance of the expert policy $w^T\mu(\pi_*)$
- Due to feature matching theorem, this ultimately translates to finding the value of w that minimizes the difference between $\mu(\pi_*)$ and $\mu(\pi)$.

- Suppose we have the following expert data.

$$D_{\pi_*} = \left\{\left(s_0^{[1]}, a_0^{[1]}, s_1^{[1]}, a_1^{[1]}, \dots\right), \left(s_0^{[2]}, a_0^{[2]}, s_1^{[2]}, a_1^{[2]}, \dots\right), \dots \left(s_0^{[m]}, a_0^{[m]}, s_1^{[m]}, a_1^{[m]}, \dots\right)\right\}$$

- The feature mean $\mu_i(\pi_*)$ (of expert policy π_*) calculated by D_{π_*} is as follows:

$$\mu_i(\pi_*) = \frac{1}{m}\sum_{j=1}^{m}\sum_{t=0}^{\infty}\gamma^t \emptyset_i(s_t^{[j]}|\boldsymbol{D_{\pi_*}})$$

- The feature mean $\mu_i(\pi)$ (of policy π) using D_π is given as follows:

$$\mu_i(\pi) = \frac{1}{m}\sum_{j=1}^{m}\sum_{t=0}^{\infty}\gamma^t \emptyset_i(s_t^{[j]}|\boldsymbol{D_{\pi}})$$

- Using $\mu_i(\pi_*)$ and $\mu_i(\pi)$, our goal is to find w that maximizes the following formula:

$$\underset{w}{\text{argmax}} \min_j w^T\mu(\pi_*) - w^T\mu(\pi_j)$$

Apprenticeship Learning

Pseudo code:

1) compute $\mu(\pi_0)$ using a policy π_0
2) $j = 1$
3) while $(d > \epsilon)$ **do**
4) compute $\mu(\pi_j)$ (using Monte Carlo method etc)
5) compute rewards: Using the the policies $\pi_0, \pi_1 \ldots, \pi_j$, compute the value of d and w that satisfies the following ($\| w \|_2 \leq 1$).

$$d = \max_{w} \min_{i<j} w^T \mu(\pi_*) - w^T \mu(\pi_i)$$

6) reinforcement learning: Using the value of from step 5), compute reward function $R_w = \left(w^{(j)}\right)^T \emptyset$.
Using R_w, compute the optimal policy π_j (using reinforcement learning)

$$j = j + 1$$

7) **end while**
8) return $\{\pi_j\}$

Apprenticeship Learning

- In step 5) of pseudo code, d refers to the difference between the expert's policy value $\left(w^T\mu(\pi_*)\right)$ and the policy value $\left(w^T\mu\left(\pi_j\right)\right)$ closest to the expert's policy.
- This means that the expert policy performs better by at least d than any other policy. Calculating w values that maximize this difference value d is the core of the algorithm.

- There are many ways to obtain these w values, and we will use a method of support vector machine (SVM).

- We know that the step 5) in pseudo code

$$\max_{w} \min_{i} w^T\mu(\pi_*) - w^T\mu(\pi_i)$$

  is equivalent to the following

$$\max_{w,\|w\|_2\leq 1} d$$

$$\text{s.t. } w^T\mu(\pi_*) \geq w^T\mu(\pi_i) + d, \qquad \forall\pi_i$$

- This formula is similar to the equation of SVM, thus problems can be solved by using SVM.

Setup of SVM

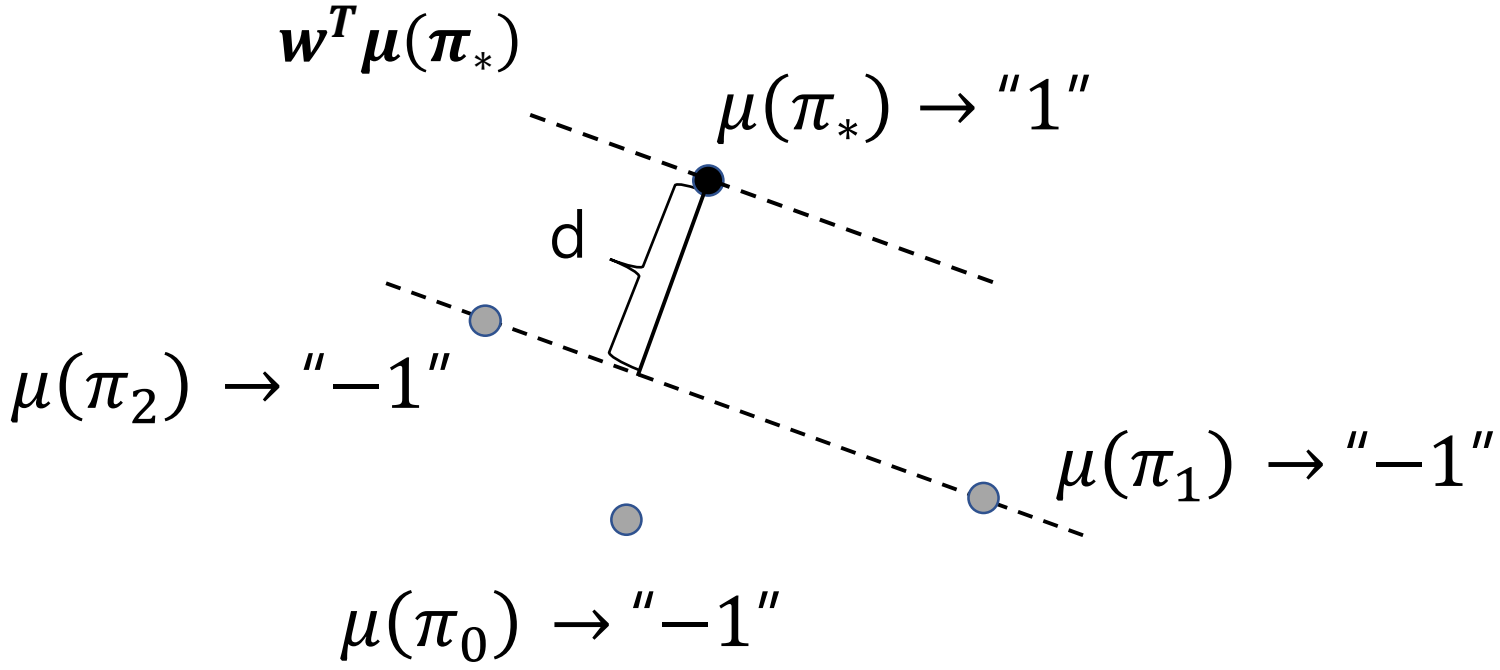
- Since SVM is a classification learning, above formula must be transformed into a classification learning problem.
  - Each data point must have a target value.

- For the expert policy π_*, its feature mean vector $\mu(\pi_*)$ is regarded as a data point. Also its target value is given as “1".
- Similarly, for a certain policy π_i (not expert policy), its feature mean vector $\mu(\pi_i)$ is regarded as a data point and its target value is “-1".
- By assigning a target value to each feature mean vector, the data set consists of one data point with a target value of "1" and data points with a target value of "-1" equal to the number of policies.

Example

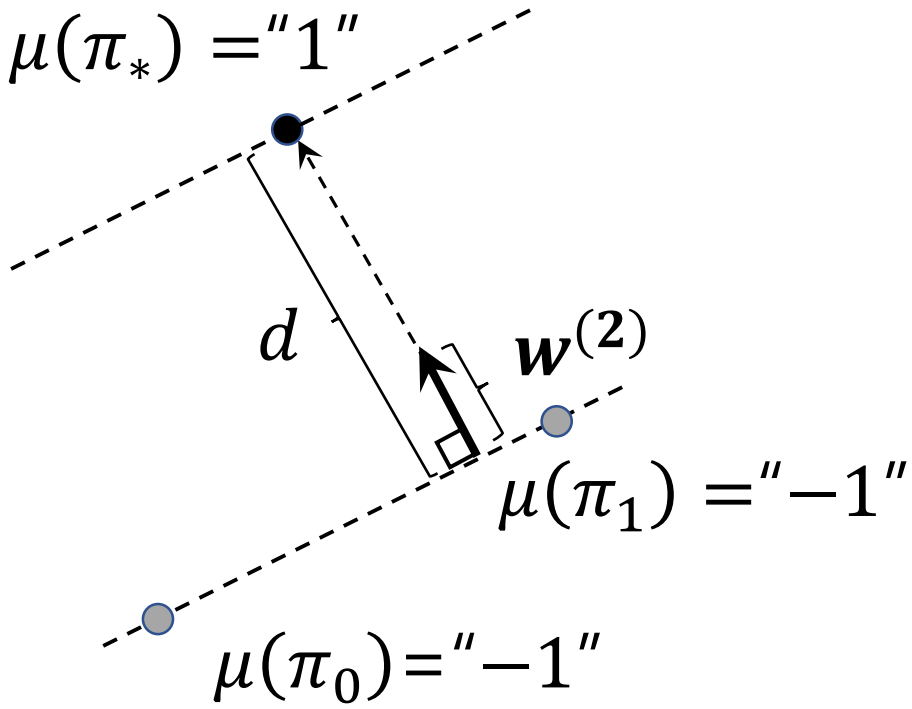
Example:

- One expert mean vector $\mu(\pi_*)$ and three policy mean vectors $\{\mu(\pi_0), \mu(\pi_1), \mu(\pi_2)\}$
- Mapping the mean vectors to target values and finding a line that maximizes the distance between the target values "1" and "-1" becomes a SVM problem.
- The objective of learning is to find the w value that maximizes the distance d using SVM.

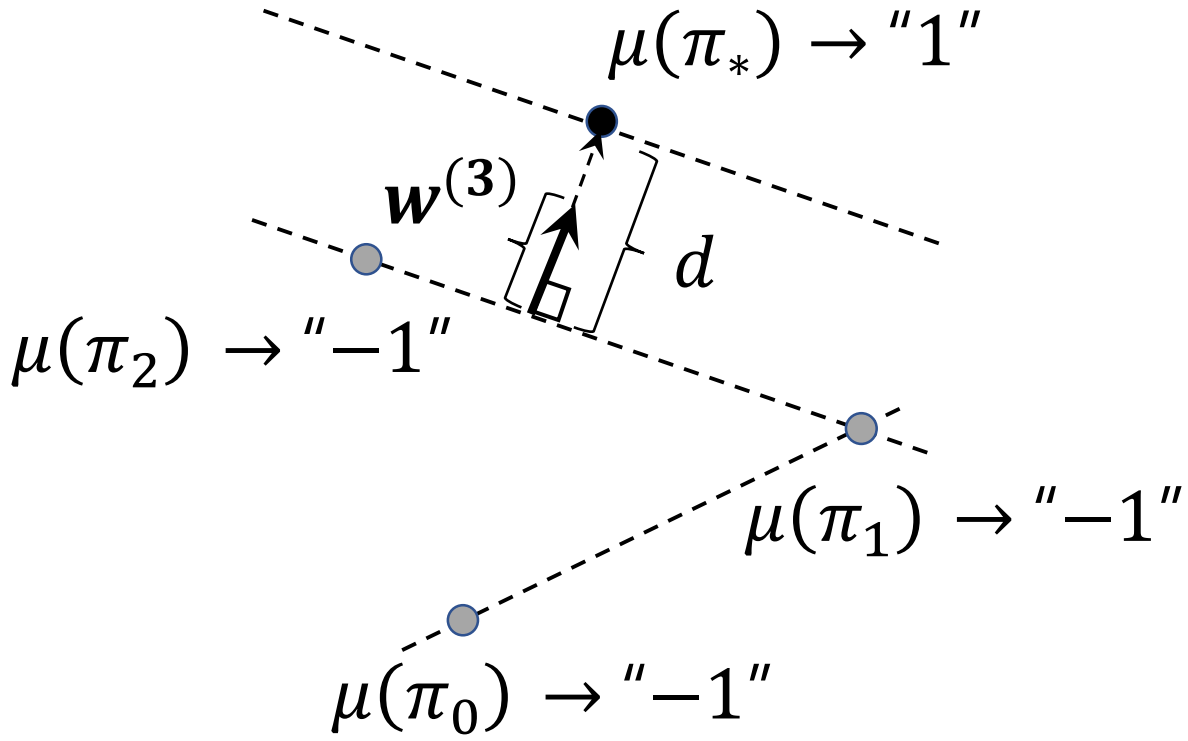


Example

$\mu(\pi_*)$ and $\{\mu(\pi_0), \mu(\pi_1)\}$



$\mu(\pi_*)$ and $\{\mu(\pi_0), \mu(\pi_1), \mu(\pi_2)\}$



- The w vector we are seeking refers to a vector that is orthogonal to the decision boundary.
- Since we assumed that $\| w \|_2 \leq 1$, w becomes a unit vector with a magnitude of 1 or less

Projection Method

- Another method of computing w vector

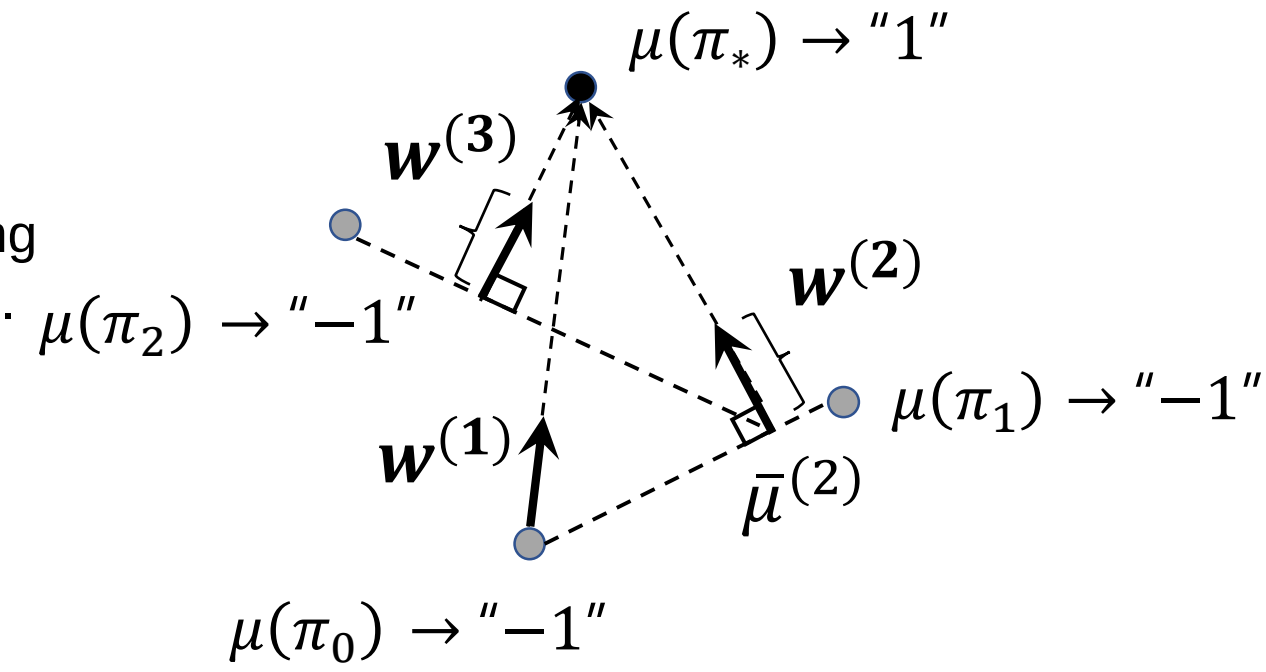
- **Projection Method:**

1) Find the intersection point where the distance between the previous boundary line (computing $w^{(i-1)}$) and the target point $\mu(\pi_*)$ is minimized. Let this position be $\bar{\mu}^{(i-1)}$.

2) Determine the line connecting this intersection point $\bar{\mu}^{(i-1)}$ and the new data point $\mu(\pi_{i-1})$.

3) Find the line that represents the shortest distance between this new line and $\mu(\pi_*)$.

4) The unit vector along this new line becomes $w^{(i)}$



GAIL (Generative Adversarial Imitation Learning)

- Apprenticeship learning computes a reward value function from expert data first and then finds a policy using reinforcement learning.
  - Since reinforcement learning is repeatedly performed in IRL, it makes the algorithm very slow.
  - Another problem is that the reward function is limited to the linear function of features.

- One of the methods that solves these problems is Generative Adversarial Imitation Learning (GAIL) method.
- The GAIL method is 1) a method of obtaining a policy directly from expert data, 2) the reward function is no longer limited to a linear function, and 3) it is possible to learn in a model-free environment.

- This GAIL method utilizes GAN's ideas a lot, and the learning method is similar.

GAIL

- In reinforcement learning, the optimal policy is the one that yields the maximum return.
- Therefore, for a given reward function r, the objective of reinforcement learning is

$$\mathrm{RL}(r) = \underset{\pi}{\operatorname{argmax}}\, E_\pi[r(s,a)]$$

- In GAIL, we use cost function $c(s,a)$, instead of reward function $r(s,a)$.

$$\mathrm{RL}(c) = \underset{\pi}{\operatorname{argmin}}\, E_\pi[c(s,a)]$$

where

$$E_\pi[c(s,a)] = E\left[\sum_{t=0}^{\infty} \gamma^t c(s_t,a_t)\right]$$

- In GAIL, the objective of learning is to find a policy that minimizes the cost value, instead of maximizing the return value

GAIL

- Maximum Entropy Reinforcement Learning (MaxEnt RL) is employed in GAIL
- MaxEnt RL adds an entropy term to the conventional reinforcement learning objective function.

$$\mathrm{RL(c)} = \underset{\pi}{\operatorname{argmin}} -H(\pi) + E_{\pi}[c(s,a)]$$

  - $H(\pi)$ represents the entropy of the policy π. This entropy measures the uncertainty/randomness in the actions chosen by the policy.

$$H(\pi) = E_{\pi}[-\log \pi(a|s)]$$

  - $H(\pi)$ term is used to prevent overfitting, effectively serving as a regularizer for the model.

- This approach encourages more exploratory and diverse policy behavior.
- By increasing entropy in the objective function, the uncertainty of the policy is increased, which in turn enhances the policy's exploration and randomness.

GAIL

- The objective function of inverse reinforcement learning can be defined as follows.

$$\mathrm{IRL}(\pi_E) = \underset{c}{\operatorname{argmax}} \left(\min_{\pi} -H(\pi) + E_{\pi}[c(s,a)] \right) - E_{\pi_E}[c(s,a)]$$

(A): cost function c that maximizes the difference between the cost of policy π_E and that of policy π

(B): cost of best policy under cost function c

(C): cost of policy π_E under cost function c

- In above formula, IRL contains RL(reinforcement learning) process and part (B) represents RL.
- In GAIL, cost function is not limited to linear function.
- When using complex cost functions (e.g., deep neural network), this can lead to overfitting
- GAIL uses an additional regularizer, $\varphi(c)$, which is a convex cost function to prevent overfitting.
- Thus, GAIL employs two regularizers. The first is the entropy regularizer, $H(\pi)$, which prevents overfitting in the policy. The second is the cost regularizer, $\varphi(c)$, which prevents overfitting in the cost function.

GAIL

- Therefore, the goal of GAIL is

$$\mathrm{IRL}_{\varphi}(\pi_E) = \underset{c}{\mathrm{argmax}} -\varphi(c) + \left(\min_{\pi} -H(\pi) + E_{\pi}[c(s,a)]\right) - E_{\pi_E}[c(s,a)]$$

- Since RL is used within IRL, we change the notation of GAIL to $\mathrm{RL}\circ\mathrm{IRL}_{\varphi}$

$$\mathrm{RL}\circ\mathrm{IRL}_{\varphi}(\pi_E) = \underset{\pi}{\mathrm{argmin}} \max_{c} -H(\pi) + E_{\pi}[c(s,a)] - E_{\pi_E}[c(s,a)] - \varphi(c)$$

- Above formula is slow since "$\underset{\pi}{\mathrm{argmin}} -H(\pi) + E_{\pi}[c(s,a)]$" (reinforcement learning part) is repeated for each cost function c
- Therefore, training $\mathrm{RL}\circ\mathrm{IRL}_{\varphi}(\pi_E)$ takes a long time.

Occupancy Measure

- To solve the problem, GAIL uses Occupancy Measure.
- Occupancy measure refers to the distribution of state-action occurrences when an agent repeatedly acts according to a specific policy.
- In a policy π, the occupancy of that policy $\rho_\pi(s,a)$ can be defined as

$$\rho_\pi(s,a) = \pi(a|s)\sum_{t=0}^{\infty}\gamma^t P(s_t = s|\pi)$$

- Therefore, $E_\pi[c(s,a)]$ can redefined using occupancy measure.

$$E_\pi[c(s,a)] = E_\pi\left[\sum_{t=0}^{\infty}\gamma^t c(s_t,a_t)\right] = \sum_{s,a}\rho_\pi(s,a)c(s,a)$$

GAIL

- It is known that the occupancy distribution for each policy is distinct.
- That is, a specific policy and its occupancy distribution are known to have a one-to-one mapping relationship.
- Therefore, the problem of finding a specific policy in GAIL becomes equivalent to finding the occupancy distribution that matches that policy.
- For instance, if the occupancy distribution of a policy matches that of an expert policy, it can be assumed to be the expert's policy.

- This concept allows GAIL to effectively learn and replicate expert behaviors by matching occupancy distributions.

GAIL

- Now the goa of GAIL is to find a policy π that minimizes the difference between the two occupancy distributions ρ_{π_E} and ρ_{π}.
- Suppose $\varphi^*(\rho_\pi - \rho_{\pi_E})$ is a function that represents the difference between $\rho_\pi(s,a)$ and $\rho_{\pi_E}(s,a)$.
- **Theorem**: $\mathrm{RL}\circ\mathrm{IRL}_\varphi(\pi_E)$ is equivalent to the following

$$\mathrm{RL}\circ\mathrm{IRL}_\varphi(\pi_E) = \underset{\pi}{\operatorname{argmin}} -H(\pi) + \varphi^*(\rho_\pi - \rho_{\pi_E})$$

- **Proof:**

$$\mathrm{RL}\circ\mathrm{IRL}_\varphi(\pi_E) = \underset{\pi}{\operatorname{argmin}} \max_c -H(\pi) + E_\pi[c(s,a)] - E_{\pi_E}[c(s,a)] - \varphi(c)$$

$$= \underset{\pi}{\operatorname{argmin}} \max_c -H(\pi) + \sum_{s,a} \rho_\pi(s,a)c(s,a) - \sum_{s,a} \rho_{\pi_E}(s,a)c(s,a) - \varphi(c)$$

$$= \underset{\pi}{\operatorname{argmin}} \max_c -H(\pi) + \sum_{s,a} \left(\rho_\pi(s,a) - \rho_{\pi_E}(s,a)\right) c(s,a) - \varphi(c)$$

$$= \underset{\pi}{\operatorname{argmin}} -H(\pi) + \varphi^*(\rho_\pi - \rho_{\pi_E})$$

- The advantage of using the occupancy distribution is that we don't need to use reinforcement learning part, thereby speeding up the learning process."

φ^* function

- The term φ^* is called the conjugate function of the function φ, and the definition of the conjugate function φ^* is as follows.

$$\varphi^*(y) = \sup_x (y)^T x - \varphi(x)$$

- In the GAIL method, depending on the type of this function φ, the characteristics of the GAIL algorithm can be varied
- GAIL uses φ_{GA} function defined as follows:

$$\varphi_{GA}(\mathrm{c}) = \begin{cases} \mathrm{E}_{\pi_E}\left[g\left(c(s,a)\right)\right], & \text{if } \mathrm{c} < 0 \\ +\infty, & \text{otherwise} \end{cases}$$

$$\text{where } g(\mathrm{x}) = \begin{cases} -x - \log(1 - e^x), & \text{if } \mathrm{x} < 0 \\ +\infty, & \text{otherwise} \end{cases}$$

GAIL

- The conjugate function $\varphi^*_{\mathrm{GA}}(\rho_\pi - \rho_{\pi_E})$ is

$$\varphi^*_{\mathrm{GA}}(\rho_\pi - \rho_{\pi_E}) = \max_D \underbrace{E_\pi\left[\log\left(D(s,a)\right)\right]}_{(s,a) \text{ from general policies maximize } D(s,a)} + \underbrace{E_{\pi_E}\left[\log\left(1 - D(s,a)\right)\right]}_{(s,a) \text{ from expert policy minimizes } D(s,a)}$$

- In the above formula, $D: S \times A \rightarrow (0,1)$ can be thought of as a classifier that predicts whether a given occupancy distribution is the occupancy distribution of expert policy.
  - The occupancy distribution of the expert is classified as 0, while the occupancy distribution of a regular policy is classified as 1.
  - (In a standard GAN, the classifier categorizes real images as 1 and fake data as 0)
- Maximizing these two terms together means that the classifier $D(s,a)$ should be trained to produce classification values close to 1 for regular policies and classification values close to 0 for expert policies.
- When the training of classifier D is complete (i.e., when the occupancy distributions of the expert policy and regular policy are almost identical), $\varphi^*_{\mathrm{GA}}(\rho_\pi - \rho_{\pi_E})$ represents the optimal negative log loss value.

GAIL and GAN

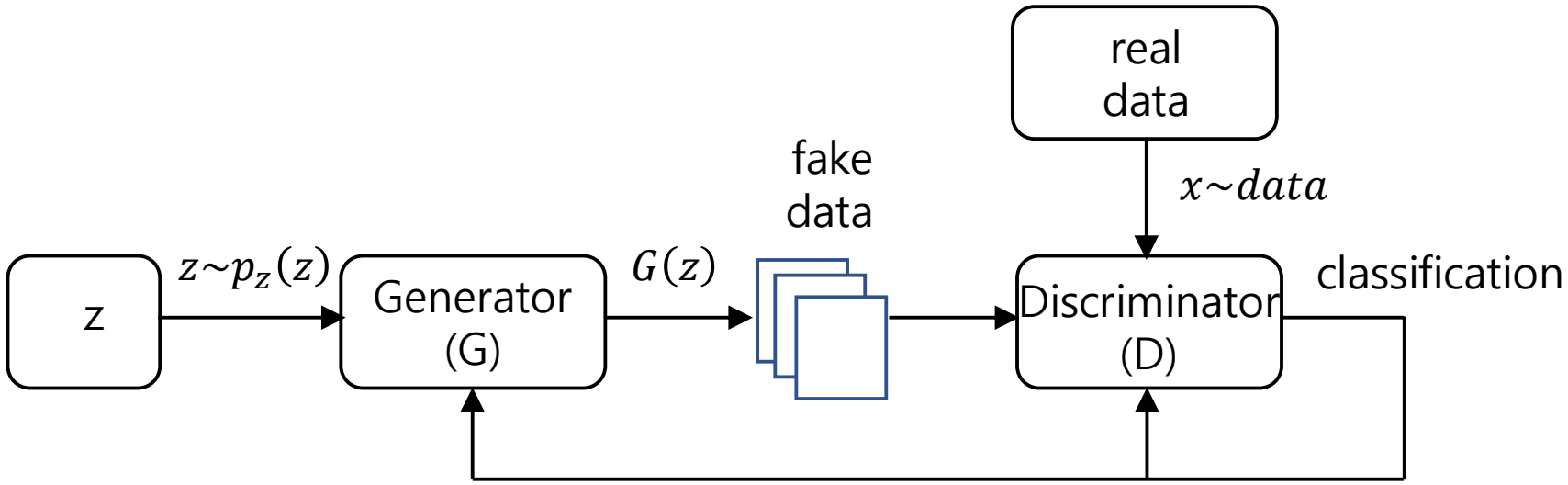
- Using $\varphi^*_{\mathrm{GA}}(\rho_\pi - \rho_{\pi_E})$, the objective function of GAIL becomes:

$$\min_\pi \max_D E_\pi[\log(D(s,a))] + E_{\pi_E}[\log(1 - D(s,a))] - \lambda H(\pi)$$

- Above formula is similar to the objective function of GAN (except for $\lambda H(\pi)$)

$$\min_G \max_D E_{x\sim data}[\log(D(x))] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

- Remember: The occupancy distribution of the expert is classified as 0, while the occupancy distribution of a regular policy is classified as 1. (In a standard GAN, the classifier categorizes real images as 1 and fake data as 0)



GAIL

- The theoretical background why GAIL uses the same learning method as GAN:
- GAIL repeats the following:

  1) first optimizes $\varphi^*_{\mathrm{GA}}(\rho_\pi - \rho_{\pi_E})$. (i.e., classifier D is optimized)

$$\max_D E_\pi\left[\log\left(D(s,a)\right)\right] + E_{\pi_E}\left[\log\left(1 - D(s,a)\right)\right]$$

  2) with the optimized D, finds a policy π that minimizes

$$-H(\pi) + \varphi^*(\rho_\pi - \rho_{\pi_E})$$

- GAIL finds a policy with the optimized value of $\varphi^*_{\mathrm{GA}}(\rho_\pi - \rho_{\pi_E})$.

- Let D^* be the optimal classifier. Then, $D^*(s,a)$ would be a Bayesian classifier, and the following relationship is established.

$$D^*(s,a) = \frac{\rho_{\pi_E}}{\rho_{\pi_E} + \rho_\pi}$$

GAIL

- **Theorem:** When classifier D achieves its optimality, $\varphi^*_{\mathrm{GA}}(\rho_\pi - \rho_{\pi_E})$ is equivalent to the Jensen-Shannon divergence $\mathrm{D}_{\mathrm{JS}}(\rho_\pi, \rho_{\pi_E})$.
- **Proof:** Suppose D^* is the optimal classifier,

$$E_\pi[\log(D(s,a))] + E_{\pi_E}[\log(1 - D(s,a))] = E_\pi[\log(D^*(s,a))] + E_{\pi_E}[\log(1 - D^*(s,a))]$$

$$= E_\pi\left[\log\left(\frac{\rho_{\pi_E}}{\rho_{\pi_E}+\rho_\pi}\right)\right] + E_{\pi_E}\left[\log\left(\frac{\rho_\pi}{\rho_{\pi_E}+\rho_\pi}\right)\right] \text{ (since } D^*(s,a) = \frac{\rho_{\pi_E}}{\rho_{\pi_E}+\rho_\pi})$$

$$= -\log(4) + D_{KL}\left[\rho_{\pi_E} || \frac{\rho_{\pi_E} + \rho_\pi}{2}\right] + D_{KL}\left[\rho_\pi || \frac{\rho_{\pi_E} + \rho_\pi}{2}\right]$$

$$= -\log(4) + 2 * \mathrm{D}_{\mathrm{JS}}(\rho_\pi, \rho_{\pi_E}) \approx \mathrm{D}_{\mathrm{JS}}(\rho_\pi, \rho_{\pi_E})$$

- Therefore the objective function of GAIL is

$$\mathrm{RL}\circ\mathrm{IRL}_\varphi(\pi_E) = \underset{\pi}{\operatorname{argmin}} -H(\pi) + \varphi^*(\rho_\pi - \rho_{\pi_E}) \approx \underset{\pi}{\operatorname{argmin}} -\lambda H(\pi) + \mathrm{D}_{\mathrm{JS}}(\rho_\pi, \rho_{\pi_E})$$

- It is known that training GAN is equivalent to minimizing Jensen-Shannon divergence between generator and data distributions.

GAIL with Neural Networks

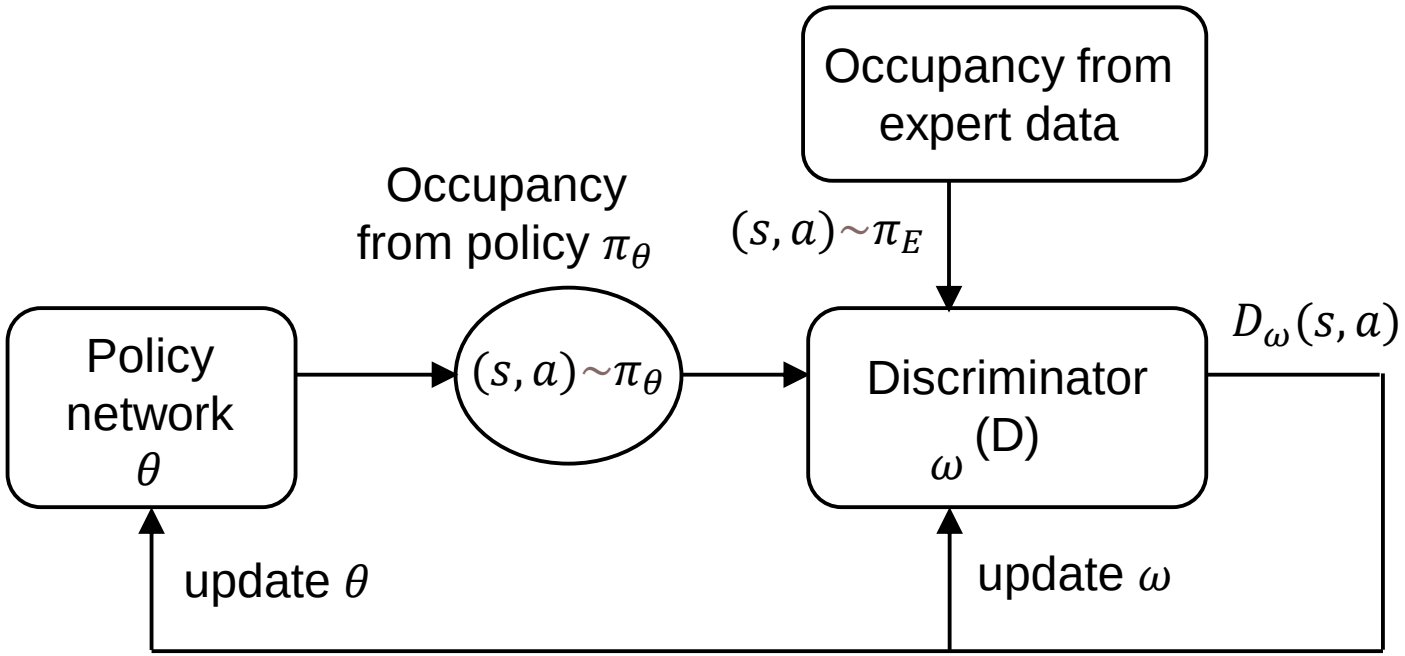
- One of the advantages of the GAIL is that it allows to use general functions such as neural networks to represent the cost function (i.e., the reward function), not limited to using linear functions (as in apprenticeship learning).
- Therefore, both the cost function and the policy are represented using neural networks.

- When using these neural networks, final objective function of GAIL is expressed as

$$\min_{\theta} \max_{\omega} E_{\pi_\theta}[\log(D_\omega(s,a))] + E_{\pi_E}[\log(1 - D_\omega(s,a))] - \lambda H(\pi_\theta)$$

  where ω be the parameters of the cost function network and θ be the parameters of the policy network, respectively.

GAIL



GAIL

Pseudo code:

Input: Expert data $\tau_E = (s_0, a_0, s_1, a_1, \dots) \sim \pi_E$
Initialize parameter θ of policy network π_θ and parameter w of classifier d_w, respectively
for $i = 0,1, \dots$ **do**
create episode set τ_i using policy π_θ
update classifier network parameter:
$\delta_w = E_{\tau_i}[\nabla_w \log D_\omega(s,a)] + E_{\tau_E}[\nabla_w \log(1 - D_\omega(s,a))]$
$w \leftarrow w + \alpha_w \delta_w$
update policy network parameter(using TRPO or PPO):
$\hat{q}(\check{s}, \check{a}) = E_{\tau_i}[\log D_\omega(s,a) \,|s_0 = \check{s}, a_0 = \check{a}]$
$\delta_\theta = E_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s)\, \hat{q}(s,a)] - \lambda \nabla_\theta H(\pi_\theta)$
$\theta \leftarrow \theta - \alpha_\theta \delta_\theta$
end for