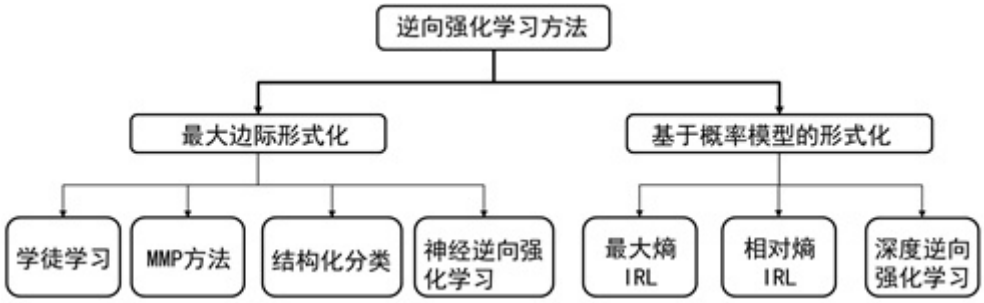
Inverse Reinforement Learning

Zhang Yinmin



逆向强化学习的分类

逆向强化学习一般流程如下：
1. 随机生成一个策略作为初始策略；
2. 通过比较“高手”的交互样本和自己交互样本的差别，学习得到回报函数；
3. 利用回报函数进行强化学习，提高自己策略水平；
4. 如果两个策略差别不大，就可以停止学习了，否则回到步骤2。

逆向强化学习分类如下：

1. 最大边际形式化：学徒学习(Apprenticeship learning)、MMP方法(Maximum Margin Planning)、结构化分类(SCIRL)、神经逆向强化学习(NIRL)。
2. 基于概率模型的形式化：最大熵IRL、相对熵IRL、深度逆向强化学习。

最大边际化方法的缺点是很多时候不存在单独的回报函数使得专家示例行为既是最优的又比其它任何行为好很多，或者不同的回报函数挥导致相同的专家策略，也就是说这种方法无法解决歧义问题。基于概率模型的方法可以解决此问题。

基于最大边际的逆向强化学习

学徒学习

学徒学习指的是从专家示例中学到回报函数，使得在该回报函数下所得的最优策略在专家示例策略附近。设未知的回报函数

$$R(s) = w \cdot \phi(s)$$

其中 $\phi(s)$ 为基函数，可以是多项式基底、傅里叶基底等。此时逆向强化学习要求得的是回报函数中的系数w.

定义**特征期望**为 $\mu(\pi) = E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi]$

给定m条专家轨迹时，我们可以估计专家的特征期望为:

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

我们要找一个策略，使得它的表现与专家策略相近，其实就是找到一个策略$\tilde{\pi}$的特征期望与专家策略的特征期望相近，即

$$\|\hat{\mu}(\pi) - \tilde{\mu}_E\|_2 \leq \epsilon$$

学徒学习的伪代码如下：

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

其中step 2写成标准的优化形式为：

$$\begin{aligned}&\max_{t,w} t\\ \text{s.t. } &w^T\mu_E \geq w^T\mu^(j), j = 0, 1, \cdots, i-1\\ &\|w\|_2 \leq 1\end{aligned}$$

综上所述，学徒学习方法可分两步：

1. 在已经迭代得到的最优策略中，利用最大边际方法求出当前的回报函数的参数值；
2. 将求出的回报函数作为当前系统的回报函数，并利用强化学习方法求出此时的最优策略。

Gridworld(grid_size, wind, discount): Gridworld MDP.

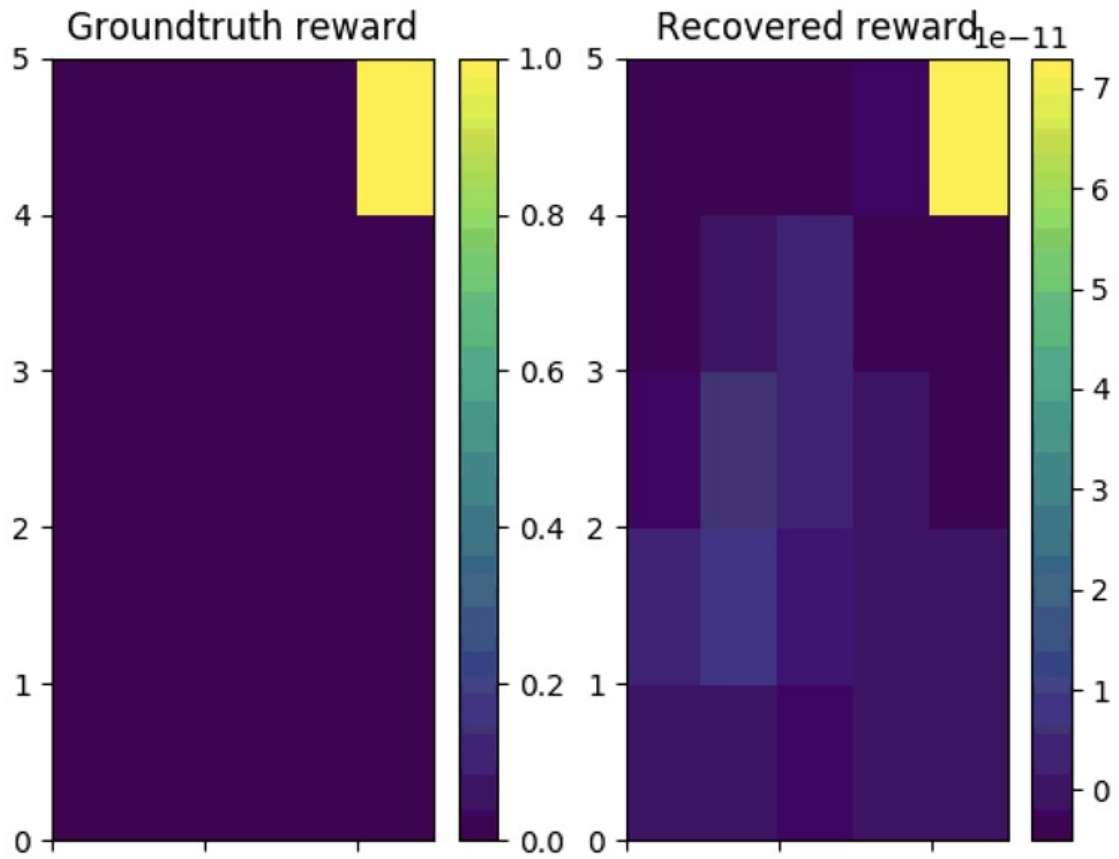
- actions: Tuple of (dx, dy) actions.
- n_actions: Number of actions. int.
- n_states: Number of states. int.
- grid_size: Size of grid. int.
- wind: Chance of moving randomly. float.
- discount: MDP discount factor. float.
- transition_probability: NumPy array with shape (n_states, n_actions, n_states) where transition_probability[si, a, sk] is the probability of transitioning from state si to state sk under action a.
- feature_vector(i, feature_map="ident"): Get the feature vector associated with a state integer.
- feature_matrix(feature_map="ident"): Get the feature matrix for this gridworld.
- int_to_point(i): Convert a state int into the corresponding coordinate.

- point_to_int(p): Convert a coordinate into the corresponding state int.
- neighbouring(i, k): Get whether two points neighbour each other. Also returns true if they are the same point.
- reward(state_int): Reward for being in state state_int.
- average_reward(n_trajectories, trajectory_length, policy): Calculate the average total reward obtained by following a given policy over n_paths paths.
- optimal_policy(state_int): The optimal policy for this gridworld.
- optimal_policy_deterministic(state_int): Deterministic version of the optimal policy for this gridworld.
- generate_trajectories(n_trajectories, trajectory_length, policy, random_start=False): Generate n_trajectories trajectories with length trajectory_length, following the given policy.

Apprenticeship Learning

```
def irl(n_states, n_actions, transition_probability, policy, disc
ount, Rmax,
        l1):
    """
    Find a reward function with inverse RL as described in Ng & R
ussell, 2000.
    n_states: Number of states. int.
    n_actions: Number of actions. int.
    transition_probability: NumPy array mapping (state_i, action,
state_k) to
        the probability of transitioning from state_i to state_k
 under action.
        Shape (N, A, N).
    policy: Vector mapping state ints to action ints. Shape (N,).
    discount: Discount factor. float.
    Rmax: Maximum reward. float.
    l1: l1 regularisation. float.
    -> Reward vector
    """
```

结果展示



最大边际规划(MMP)

最大边际规划将逆向强化学习建模为$D = \{(\mathcal{X}_i, \mathcal{A}_i, p_i, F_i, y_i, \mathcal{L}_i)\}_{i=1}^n$ 该式从左到右依次为状态空间,动作空间,状态转移概率,回报函数的特征向量,专家轨迹和策略损失函数.在MMP框架下,学习者试图找到一个特征到回报的线性映射也就是参数w,在这个线性映射下最好的策略在专家示例策略附近.该问题可形式化为:

$$\min_{w_i,\zeta_i} \frac{1}{2}\|w\|^2 + \frac{\gamma}{n}\sum_i \beta_i \zeta_i^q$$

$$\text{s.t. } \forall i \;\; w^T F_i \mu_i + \zeta_i \geq \max_{\mu \in \mathcal{G}_i} w^T F_i \mu + l_i^T \mu$$

第二行为约束，其含义如下:

1. 约束只允许专家示例得到最好的回报的权值存在;
2. 回报的边际差，即专家示例的值函数与其他策略的值函数的差值,与策略损失函数成正比.

损失函数可以利用轨迹中两种策略选择不同动作的综合来衡量，此处策略μ 指的是每个状态被访问的频次.

基于结构化分类的方法

MMP方法在约束不等式部分,变量是策略,需要迭代求解MDP的解,计算代价很高.为了避免迭代计算MDP的解,我们可以这样考虑问题:对于一个行为空间很小的问题,最终的策略其实是找到每个状态所对应的最优动作.若把每个动作看作是一个

标签,那么所谓的策略其实就是把所有的装填分成几类,分类的表示是值函数,最好的分类对应着最大的值函数.利用这个思想，逆向强化学习可以形式化为:

$$\min_{\theta,\zeta} \frac{1}{2}\|\theta\|^2 + \frac{\eta}{N}\sum_{i=1}^{N}\zeta_i$$

$$\text{s.t. } \forall i,\ \ \theta^T\hat{\mu}^{\pi E}(s_i,a_i) + \zeta_i \geq \max_{\alpha}\theta^T\hat{\mu}^{\pi E}(s_i,a) + L(s_i,a)$$

约束中的$\{s_i,a_i\}$为专家轨迹,$\hat{\mu}^{\pi_E}(s_i,a_i)$ 可以利用Monte Carlo方法求解,而对于 $\hat{\mu}^{\pi_E}(s_i,a\neq a_i)$ 则可利用启发的方法得到.
从数学形式看,结构化分类的方法与MMP非常相似,但是两者有本质不同:

1. 结构化分类的方法约束每个状态处的每个动作；
2. MMP约束一个MDP的解.

基于最大熵的逆向强化学习

基于最大边际的方法往往会产生歧义，比如许多不同的回报函数会导致相同的专家策略。在这种情况下，所学到的回报函数往往具有随机的偏好。最大熵方法就是为解决此问题提出的。
原因在于通过熵最大所选取的模型，没有对未知做任何主观假设。从概率模型的角度建模逆向强化学习，我们可以这样考虑：**存在一个潜在的概率分布，在该概率分布下，产生了专家轨迹。**此时，已知条件为

$$\sum_{\zeta_i} P(\zeta_i)f = \tilde{f}$$

这里用f表示特征期望，$\tilde{f}$ 表示专家特征期望。在满足上述约束条件的所有概率分布中，熵最大的概率分布式除了约束外对其他任何位置信息没有做任何假设的分布。所以，最大熵的方法可以避免歧义性的问题。

熵最大，是最优问题，我们将该问题转化为标准型：

$$\max -p\log p$$

$$\text{s.t. } \sum_{\zeta_i} P(\zeta_i)f_{\zeta_i} = \tilde{f}$$

$$\sum P = 1$$

基于最大信息熵的逆向强化学习

上述公式可用拉格朗日乘子法求解:

$$\min L = \sum_{\zeta_i} plogp - \sum_{j=1}^{n}\lambda_j(pf_i - \tilde{f}) - \lambda_0(\sum p - 1)$$

对概率p进行微分,并令导数为0,得到有最大熵的概率为:

$$p = \frac{\exp(\sum_{j=1}^{n} \lambda_j f_j)}{\exp(1-\lambda_0)} = \frac{1}{Z}\exp(\sum_{j=1} n\lambda_j f_j)$$

参数 λ_j 对应着回报函数中的参数,改参数可以利用**最大似然法**求解.
一般而言,利用最大似然法求解上式中的参数时,往往会遇到未知的配分函数项 Z因此不能直接求解.一种可行的方法是利用次梯度的方法,如:

$$\nabla L(\lambda) = \tilde{f} - \sum_{\zeta} P(\zeta|\lambda, T) f_\zeta$$

其中轨迹的概率$P(\zeta)$可表示为:

$$P_\tau(\tau|\theta, T) \propto d_0(s_1)\exp(\sum_{i=1}^{k} \theta_i f_i^\tau) \prod_{t=1}^{H} T(s_{t+1}|s_t, a_t)$$

求解该式的前提是状态转移概率T是已知的

基于相对熵的逆向强化学习

在无模型强化学习中,状态转移概率T往往是未知的,为解决此问题,我们可将问题建模为求解相对熵最大.设 Q为利用均匀分布策略产生的轨迹分布,要求解的概率分布为$P(\tau)$,问题可形式化为:

$$\min_P \sum_{\tau\in\mathcal{T}} P(\tau)\ln\frac{P(\tau)}{Q(\tau)}$$

$$\text{s.t. } \forall i \in \{1, 2, \dots, k\}:$$

$$\left|\sum_{\tau\in\mathcal{T}} P(\tau) f_i^\tau - \hat{f}_i\right| \le \epsilon$$

$$\sum_{\tau\in\mathcal{T}} P(\tau) = 1$$

$$\forall \tau \in \mathcal{T}: P(\tau) \ge 0$$

同样利用拉格朗日乘子法和KKT条件求解,可以得到相对熵最大的解.参数的求解过程同样利用次梯度方法.

GAIL

GAIL (Generative Adversarial Imitation Learning) 是一种使用 GAN (Generative Adversarial Network) 完成 Imitation Learning 工作的方法。我们知道回报函数的目标是使专家轨迹的长期回报尽可能高于其它策略生成的轨迹。把这个目标对应到 GAN 的场景下：*在 GAN 中我们希望生成分布尽可能靠近真实分布；而在逆向强化学习中我们希望策略模型轨迹尽可能靠近专家轨迹*。我们可以采用类似的方法，由策略模型充当 GAN 中的生产模型，以状态为输入生成行动；而回报函数模型可以充当判别模型，用于判别行动近似专家行动的程度。

1. 判别模型的目标是最大化公式$E_x[D(x)] + E_z[1 - D(G(z))]$,也就是分出哪些图来自真实数据分布,哪些是由生成模型生成的.
2. 生成模型的目标是最大化公式$E_z[D(G(z))]$,也就是让自己生成的图被判别模型判断为来自真实数据分布. 两个模型的目标可 以统一成一个目标函数.

$$\min_G \max_D V(D,G) = E_x[logD(x)] + E_z[log(l - D(G(z)))]$$

令回报模型为D，其参数为w，策略模型为π，其参数为θ，这样我们的目标函数就变为

$$E_\pi[logD_w(s,a)] + E_{\pi E}[log(1 - D_w(s,a))] - \lambda H(\pi)$$

Algorithm

1:**Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
2:**for** $i = 0, 1, 2, \cdots$ **do**

3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$

4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{E}_{\tau_i}[\nabla_w log(D_w(s,a))] + \hat{E}_{\tau_E}[\nabla_w log(1 - D_w(s,a)]$$

5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with the cost function $log(D_{w_{i+1}}(s,a))$.

Specifically, take a KL-constrained natural gradient step with

$$\hat{E}_{\tau_i}[\nabla_\theta log\pi_\theta(a|s)Q(s,a)] - \lambda\nabla_\theta H(\pi_\theta)$$
$$\text{where} Q(\bar{s},\bar{a}) = \hat{E}_{\tau_i}[log(D_{w_{i+1}}(s,a))|s_0 = \bar{s}, a_0 = \bar{a}]$$

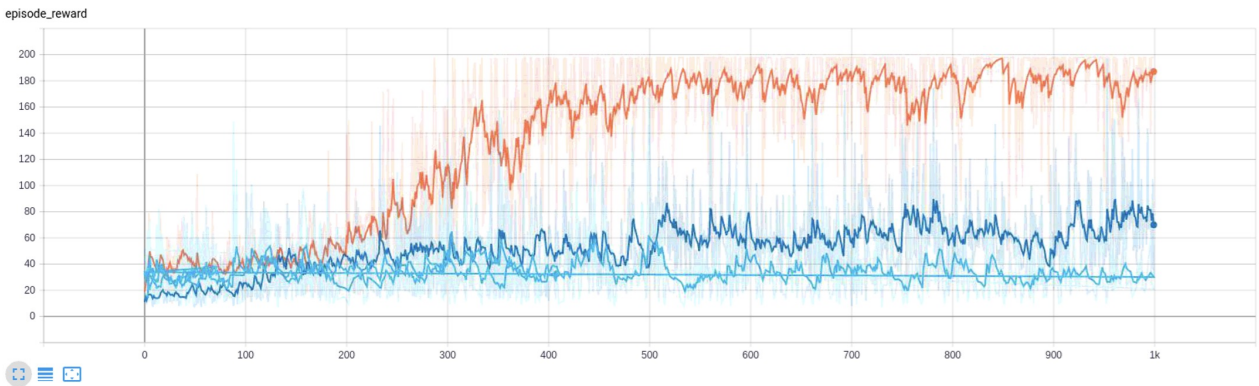
6: **end for**

1. Generate expert trajectory data Reinforcement Learning algorithm: PPO, is used for generating the expert trajectory data for the CartPole-v0 environment.
2. Sample the expert trajectory data from the PPO generated trajectories.
3. Execute Imitation Learning

```
class Policy_net:
    def __init__(self, name: str, env):
        """
        :param name: string
        :param env: gym env
        """

        with tf.variable_scope('policy_net'):
            layer_1 = tf.layers.dense(inputs=self.obs, units=20,
```

```
activation=tf.tanh)
            layer_2 = tf.layers.dense(inputs=layer_1, units=20, a
ctivation=tf.tanh)
            layer_3 = tf.layers.dense(inputs=layer_2, units=act_s
pace.n, activation=tf.tanh)
            self.act_probs = tf.layers.dense(inputs=layer_3, unit
s=act_space.n, activation=tf.nn.softmax)

        with tf.variable_scope('value_net'):
            layer_1 = tf.layers.dense(inputs=self.obs, units=20,
activation=tf.tanh)
            layer_2 = tf.layers.dense(inputs=layer_1, units=20, a
ctivation=tf.tanh)
            self.v_preds = tf.layers.dense(inputs=layer_2, units=
1, activation=None)

class Discriminator:
    def __init__(self, env):
        """
        :param env:
        Output of this Discriminator is reward for learning agen
t. Not the cost.
        Because discriminator predicts  P(expert|s,a) = 1 - P(age
nt|s,a).
        """
    def construct_network(self, input):
        layer_1 = tf.layers.dense(inputs=input, units=20, activat
ion=tf.nn.leaky_relu, name='layer1')
        layer_2 = tf.layers.dense(inputs=layer_1, units=20, activ
ation=tf.nn.leaky_relu, name='layer2')
        layer_3 = tf.layers.dense(inputs=layer_2, units=20, activ
ation=tf.nn.leaky_relu, name='layer3')
        prob = tf.layers.dense(inputs=layer_3, units=1, activatio
n=tf.sigmoid, name='prob')
        return prob
```

结果展示



训练曲线

Reference:

1. Ng A Y, Russell S J. Algorithms for inverse reinforcement learning[C]//Icml. 2000, 1: 2.
2. Abbeel P, Ng A Y. Apprenticeship learning via inverse reinforcement learning[C]//Proceedings of the twenty-first international conference on Machine learning. ACM, 2004: 1.
3. Ratliff N D, Bagnell J A, Zinkevich M A. Maximum margin planning[C]//Proceedings of the 23rd international conference on Machine learning. ACM, 2006: 729-736.
4. Klein E, Geist M, Piot B, et al. Inverse reinforcement learning through structured classification[C]//Advances in neural information processing systems. 2012: 1007-1015.
5. Xia C, El Kamel A. Neural inverse reinforcement learning in autonomous navigation[J]. Robotics and Autonomous Systems, 2016, 84: 1-14.
6. Ziebart B D, Maas A, Bagnell J A, et al. Maximum entropy inverse reinforcement learning[J]. 2008.
7. Boularias A, Kober J, Peters J. Relative entropy inverse reinforcement learning[C]//Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. 2011: 182-189.
8. Goodfellow I, Pouget-Abadie J, Mirza M, et al. Generative adversarial nets[C]//Advances in neural information processing systems. 2014: 2672-2680.
9. Ho J, Ermon S. Generative adversarial imitation learning[C]//Advances in neural information processing systems. 2016: 4565-4573.