Summary: Maximum Entropy Inverse Reinforcement Learning

Ziebart, Maas, Bagnell, Dey

Notes by Mauro Comi, maurocomi92@gmail.com

August 22, 2020

1 Important resources

Link to the paper: https://www.aaai.org/Papers/AAAI/2008/AAAI08-227.pdf
C. Finn's explanation, Deep RL Bootcamp at Berkeley: shorturl.at/ikFLM
K. Fragkiadaki's explanation:
http://www.andrew.cmu.edu/course/10-703/slides/Lecture_IRL_GAIL.pdf
Max's Blog: http://178.79.149.207/posts/maxent.html

2 Domain-specific prerequisites

- Markov Decision Process (MDP)
- General notions on Reinforcement Learning
- General notions on Probability Theory
- Principle of Maximum Entropy (explained later)
- Optimization (function maximization/minimization)

3 Paper Notes

What

The authors propose a probabilistic approach based on the principle of maximum entropy to frame problems of imitation learning as solutions to MDP. This approach provides a normalized distribution over decision sequences in order to recover a utility function; this is the reward function that makes the behaviour induced by the learned policy closely mimic the demonstrated behaviour.
The main contributions are:

- A criterion (the use of maximum entropy principle) to resolve the ambiguity in choosing a distribution over decisions among all the possible distributions that satisfy the defined constraint (matching feature expectations between an observed policy and a learner's behaviour).

- An inverse reinforcement learning probabilistic model that normalizes globally over behaviours.

Why

Approaches studied so far present numerous issues. For example, recovering the agent's exact reward weights is very complicated when the demonstrated behaviour is imperfect or when no single reward function makes the demonstrated behaviour optimal. Furthermore, at the time of publishing this paper there were no methods that could solve the ambiguity stemmed from the fact that each policy can be optimal for many reward functions.

How

The authors employ the principle of maximum entropy to resolve the ambiguity in choosing distributions. The principle of maximum entropy states that the "best" distribution among many possible distributions that satisfies the defined constraints is the one with the maximum entropy. That is, the distribution with the maximum entropy is the one with the largest remaining uncertainty and so the one with the smallest bias (among those that satisfy the constraints). The entropy is defined as the :

$$Entropy = -\sum_{x=0}^{\infty} P_{ME}(x) \cdot log P_{ME}(x) \tag{1}$$

where P_{ME} is the probability of maximum entropy. It is specifically used for high dimensional systems. A high-dimensional system is a system for which the possible number of total configurations is much bigger than the available data.
In MaxEnt IRL, the maximum entropy is constrained to match feature expectations:

$$\sum_{\tau_i} P(\tau_i) \cdot f_{\tau_i} = \tilde{f} \tag{2}$$

where

- τ_i is the i-th trajectory generated by the expert policy π^*
- $\tilde{f} = \frac{1}{m} \sum_i f_{\tau_i}$ (m = number of demonstrated trajectories).
- f_s are the features of each state.
- $f_\tau = \sum_{s_j \in \tau} f_{s_j}$ is the path feature count.

For the i-th trajectory τ_i, the *reward function* $R(f_{\tau_i}) = \theta^T f_{\tau_i}$. That is, the reward is a (linear) combination of weights and features. Since a single trajectory is a sequence of multiple states s:

$$R(f_{\tau_i}) = \sum_{s_j \in \tau_i} \theta^T f_{s_j} \tag{3}$$

The probability of each trajectory $P(\tau) \propto e^{R(\tau)}$ is defined as proportional to an exponential function, as stated by the principle of maximum entropy. Hence, trajectories with equal rewards are equally likely to be executed and trajectories with higher rewards are

exponentially more likely to be executed. Since the probability function defined over the entire space needs to be equal to 1, we need to introduce a normalization factor Z. For a **deterministic MDP**, the probability distribution function is:

$$P(\tau_i|\theta) = \frac{1}{Z(\theta)} e^{\theta^T f_{\tau_i}} = \frac{1}{Z(\theta)} e^{\sum_{s_j \in \tau_i} \theta^T f_{\tau_i}} \tag{4}$$

The tricky aspect is that Z(θ) is usually intractable, especially in high-dimensional cases. Indeed, Z(θ) is an integral defined over all the possible trajectories of the exponential of the reward:

$$Z = \int e^{R(\tau)} d\tau \tag{5}$$

Generally, MDPs are not deterministic: actions produce stochastic state transitions, according to the state transition function T. For a stochastic MDP, this randomness needs to be taken into account. In this summary only the deterministic case is explained.
How is the reward actually computed? To infer the reward, we employ Maximum Likelihood Estimation: we want to maximize the likelihood of the probability function under parameters θ. For practical purposes, the actual quantity that is maximized is the logarithm of the probability and not the probability itself (this does not affect the $argmax\theta$ whatsoever). This means:

$$\begin{aligned}
\theta^* &= argmax_\theta \; log \prod_\tau P(\tau|\theta) \\
&= argmax_\theta \sum_\tau log \; P(\tau|\theta) \\
&= argmax_\theta \sum_\tau log \; \frac{1}{Z} e^{R(\tau)} \\
&= argmax_\theta \sum_\tau R(\tau) - M \; logZ \\
&= argmax_\theta \sum_\tau R(\tau) - M \; log \sum_\tau e^{R(\tau)} \\
&= argmax_\theta \sum_\tau \theta^T f_\tau - M \; log \sum_\tau e^{\theta^T f_\tau}
\end{aligned} \tag{6}$$

where M is the number of trajectories. To optimize the resulting objective, we use gradient descend (the objective function is convex):

$$\nabla_\theta L = \sum_\tau \frac{R(\tau)}{d\theta} - M \; \frac{1}{\sum_\tau e^{R(\tau)}} \sum_\tau e^{R(\tau)} \frac{R(\tau)}{d\theta} \tag{7}$$

The gradient can be subdivided into two components, A - B:

- Component A:

$$\sum_\tau \frac{R(\tau)}{d\theta} = \sum_\tau \frac{\theta^T \; f_\tau}{d\theta} = \sum_\tau f_\tau \tag{8}$$

- Component B:

$\sum_\tau e^{R(\tau)}$ can be moved inside the sum:

$$\begin{aligned}&\sum_\tau \frac{e^{R(\tau)}}{\sum_\tau e^{R(\tau)}} \frac{R(\tau)}{d\theta} \\ &= \sum_\tau p(\tau|\theta) \frac{R(\tau)}{d\theta}\end{aligned} \tag{9}$$

as seen in Equation 4 and since τ is composed of states:

$$\sum_s p(s|\theta) \frac{R(s)}{d\theta} = \sum_s p(s|\theta) \ \frac{\theta^T \ f_s}{d\theta} = \sum_s p(s|\theta) \ f_s \tag{10}$$

Combining these two components, expressed wrt s:

$$\nabla_\theta L = \sum_s f_s - M \ \sum_s p(s|\theta) \ f_s \tag{11}$$

f_s is the feature vector for state s and $p(s|\theta)$ is the state visitation freuency (SVF) for state s. The SVF can be calculated with dynamic programming. In order to do so, we define $\mu_t(s)$ as the probability of visiting state s at time t. This way:

$$p(s|\theta) = \sum_t \mu_t(s) \tag{12}$$

The dynamic programming algorithm works as follow:

$\mu_1(s) = p(S = S_1)$
for $t = (1 \ ... \ T)$ **do**
| $\mu_{t+1}(s') = \sum_a \sum_s \mu_t(s) \ \pi(a|s) \ p(s'|a, s)$
end

Interesting references

- Jaynes, Edwin T. "Information theory and statistical mechanics." Physical review 106.4 (1957): 620.

4 Comments

- Reward function is linear: not suitable for complex autonomous driving scenarios. Read Deep MaxEnt IRL to overcome this limitation.
- It is necessary to know the dynamic of the environment