Application of Bayesian inverse reinforcement learning with bayesian reward extrapolation algorithm

Fyras Telmini, Touzi Mohamed

January 28, 2024

1 The reinforcement learning paradigm

Reinforcement learning is the sub-field of Machine Learning that deals with decision making. It's about learning the optimal Policy in an environment to obtain maximum reward. The decision making entity learns this threw trial and error [3]. This entity is not told which decision to take ,as in the presence of a supervisor, but instead it must learn by itself, by interacting with its environment. The quality of actions is measured by not just the immediate reward they return, but also the cumulative reward they might fetch. Formally, we define a policy π in the context of a Markov Decision Process which is a tuple structured as follows:

- S: State space
- A: Action Space
- P: Transition function which gives you the probability of transitioning to State S' after encating action A in State S: $P(S'|S, A)$
- R: Immediate Reward function

We can now formally define the policy a mapping from actions to States $\pi : S \rightarrow A$. Reinforcement learning Algorithms are divided into two categories:

- Off-Policy: learning algorithms evaluate and improve a policy that is different from Policy that is used for action selection. One of the most famous off-Policy algorithms is Q Learning.
- On-Policy: learning algorithms are the algorithms that evaluate and improve the same policy which is being used to select actions. That means we will try to evaluate and improve the same policy that the agent is already using for action selection.

Note that in this project after determining the Reward function we will use a usual RL model that we will not explain due to the scope of this work.

2 Bayesian inverse reinforcement learning

Inverse reinforcement learning (IRL) is a reformulation of the RL problem that tries to infer the reward function from a given agent's performance. This agent is a demonstrator, often a human, and IRL tries to deduce the reward function this agent is trying to optimize through the observation of their policy[4]. An issue with this

approach is that the available number of observations of the demonstrator's behaviour are often not sufficient to allow for a single reward function to be determined, this uncertainty over the reward function is better expressed in a probabilistic context. Thus in our case we're looking to sample values of the reward function from its distribution. This is more intuitive in the Bayesian context which reformulates the IRL problem as a Bayesian-IRL problem:
Given the same Markov-Decision-Process Model explained in the previous part, we assume there's an agent X that operates within it. This agent represents the expert that we're trying to learn from. We have access to a series of observations from this agent's behaviour that are pairs of environment states and agent's actions (s_i, a_i). Two assumptions are made for the expert's behaviour:

1. X is trying to maximize the total accumulated reward $\mathbf{R}$
2. X executes a stationary policy (invarient w.r.t time and actions and observations of previous time steps)

These assumptions are intuitive, but most importantly they allow us to better formulate the expert's policy:
The first assumption indicates that the goal of the expert is to maximize the accumulated reward which is exactly equivalent to saying that the expert finds the action that maximizes the Q^* in a given state. This is modeled by writing the likelihood of an observation (s_i, a_i) as:

$$Pr_\chi((s_i, a_i)|R) = \frac{1}{Z_i} exp(\alpha \chi Q^*(s_i, a_i, R)) \tag{1}$$

This exponential distribution reflects that the agent is more likely to take an action the larger its Q^*. α_x is the degree of confidence we have of the agent acting optimally.
The second assumption allows to write the independence formula:

$$Pr_\chi(O_\chi|R) = Pr_\chi((s_1, a_1)|R) * Pr_\chi((s_2, a_2)|R)...Pr_\chi((s_k, a_k)|R) \tag{2}$$

Which allows for the likelihood to be written as:

$$Pr(D|R) = \prod_{(s,a)\in D} \pi_R^\beta(a|s) = \prod_{(s,a)\in D} \frac{exp(\beta Q_R^*(s,a))}{\sum_{b\in A} exp(\beta Q_R^*(s,b))} \tag{3}$$

Then, using the Bayes theorem using an already known prior distribution $P(R)$, its possible to sample from the posterior distribution $P(R|D) \sim P(D|R)P(R)$ using Monte-Carlo-Markov-Chain (MCMC) sampling. This sampling is used during an RL training session to obtain the reward function values for the current state of the agent.
In the paper we implemented, it was explained that with the current formulation of the likelihood function, it's very computationally expensive to sample from the posterior. Because every sample needs to calculate Q^*. This makes the current formulation of B-IRL not implementable for high dimention problems such as training an agent to play an Atari game. The paper uses a different formulation of the likelihood that is less computationally expensive which allows for a computationally efficient training procedure to take place.

3 Finding a better likelihood function with BREX

The first major step is to compute the likelihood function in (3) which requires optimal q-value. Reformulating a new likelihood function would be approach since calculating an optima Q value is computationally challenging. T-REX: Trajectory ranked reward Extrapolation was proposed[1] as an efficient reward inference algorithm that

transforms reward function learning into classification problem via a pairwise ranking loss. However, T-REX only solves for a point estimate of the reward function.

One approach based on a pairwise preference likelihood allows for efficient sampling from the posterior distribution over reward functions. for a sample of trajectories $D = \tau_1, .., \tau_m$ with a set of pairwise preferences over trajectories $P = (i, j) : \tau_i < \tau_j$[1].Pairwise ranking likelihood is used to compute the likelihood of a set of preferences over demonstrations P, given a parameterized reward function hypothesis R_θ. Using Bradley-Terry model[2] we get:

$$P(D, \mathbf{P}|R_\theta) = \prod_{(i,j)\in P} \frac{exp(\beta R_\theta(\tau_j))}{\exp(\beta R_\theta(\tau_j)) + \exp(\beta R_\theta(\tau_i))} \tag{4}$$

where$R_\theta(\tau)$ is the predicted return of trajectory τ under the reward function R_θ. We can then perform Bayesian inference via MCMC to obtain samples from $P(R_\theta|D, P) \propto P(D, P|R_\theta)P(R_\theta)$. This approach is called Bayesian REX.

A Deep Neural Network is used to estimate R_θ, we will not get into the method of approximating R_θ since it was explained in the paper.

$$R_\theta = \sum_{s\in\tau} \omega^T \phi(s) = \omega^T \sum_{s\in\tau} \phi(s) = \omega^T \Phi_\tau \tag{5}$$

$P(D, \mathbf{P}|R_\theta)$ becomes:

$$P(D, \mathbf{P}|R_\theta) = \prod_{(i,j)\in\mathbf{P}} \frac{e^{\beta\omega^T\Phi_{\tau_j}}}{e^{\beta\omega^T\Phi_{\tau_j}} + e^{\beta\omega^T\Phi_{\tau_i}}} \tag{6}$$

4 Application: learning to play atari games

The paper provides a clear implementation of this technique made for Atari games, we chose to expand on this implementation by running it on another Atari game that the paper did not account for. We chose to do this because fundamentally a large part of the code is too complicated to reproduce, yet we still valued having a working example since reinforcement learning is more intuitive through examples. The full implementation is far from an exact copy of the paper's since we spent a lot of work and effort to debug it (due to compatibility issues) and to simplify it (in order to provide you with a working example). We chose to cite here some parts of the code that are relevant to the theory we just exposed to give an insight on how this problem is approached in reality.

4.1 Prior

The prior in the application is defined by:

$$P(\omega) = \begin{cases} 1, & \text{if } e^{\beta\omega^T\Phi_{\tau_1}} < 1 \\ 0, & \text{otherwise} \end{cases}$$

This expresses how the return of the worse demonstration is non-negative, (if it's negative, it would correspond to $\beta\omega^T\Phi_{\tau_1} < 0$ as $\beta = 1$ in the paper). The python version of this is:

```
demo_returns = confidence * torch.mv(demo_cnts, weights)
if demo_returns[0] < 0.0:
    return torch.Tensor([-float("Inf")])
```

This is an exact reproduction of the prior's formula, as **torch.mv** just does the matrix multiplication of demo_cnts and the weights (mathematically: $\omega^T\Phi_\tau$). While checking the condition in the formula (if the

returns are negative then this is worse than the worst strategy). We also note that this calculation is done on the log() of the prior, thats why -infinity is returned instead of zero.

4.2 Bayesian posterior via reward extrapolation

As for the Bayesian reward extrapolation algorithm, we prefer to show it in pseudo-code as the python implementation does not show how it works in a clear way.

Algorithm 1 Algorithm 1 Bayesian REX: Bayesian Reward Extrapolation

Input: demonstrations D, pairwise preferences P, MCMC proposal width σ, number of proposals to generate N, deep network architecture R_θ, and prior $P(w)$.

Pre-train R_θ using auxiliary tasks (see Section 5.2).
Freeze all but last layer, w, of R_θ.
$\phi(s) :=$ activations of the penultimate layer of R_θ.
Precompute and cache $\Phi_\tau = \sum_{s \in \tau} \phi(s)$ for all $\tau \in D$.
Initialize w randomly.
$Chain[0] \leftarrow w$
Compute $P(P, D|w)P(w)$ using Equation (9)
for $i \leftarrow 1$ to N **do**
    $w' \leftarrow \text{normalize}(N(w_t, \sigma))$
    Compute $P(P, D|w')P(w')$ using Equation (9)
    $u \leftarrow \text{Uniform}(0, 1)$
    if $u < \frac{P(P,D|w')P(w')}{P(P,D|w)P(w)}$ **then**
        $Chain[i] \leftarrow w'$
        $w \leftarrow w'$
    else
        $Chain[i] \leftarrow w$
return $Chain$

5 Conclusion

This was a very interesting project for us since it was our first time seeing the applications of Bayesian statistics in a domain that interests us Both: Reinforcement Learning. The main challenge was to get familiar with the application of Bayesian Statistics in RL before diving into the code which took quite some time on its own. Lastly we would have loved to rewrite the code from scratch but it felt obsolete to repeat a process that has been well optimized so we took our time to understand each part of it.

Bibliography

[1] Daniel S. Brown et al. *Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations*. 2019. DOI: 10.48550/ARXIV.1904.06387. URL: https://arxiv.org/abs/1904.06387.

[2] Daniel S. Brown et al. *Safe Imitation Learning via Fast Bayesian Reward Inference from Preferences*. 2020. DOI: 10.48550/ARXIV.2002.09089. URL: https://arxiv.org/abs/2002.09089.

[3] Olivier Buffet, Olivier Pietquin, and Paul Weng. *Reinforcement Learning*. 2020. DOI: 10.48550/ARXIV.2005.14419. URL: https://arxiv.org/abs/2005.14419.

[4] Deepak Ramachandran and Eyal Amir. "Bayesian Inverse Reinforcement Learning". In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI'07. Hyderabad, India: Morgan Kaufmann Publishers Inc., 2007, pp. 2586–2591.