Maximum Entropy Bayesian Actor Critic

Steven Homer[1][0000−0002−1515−1093]

AI Lab at Vrije Universiteit Brussel, 1050 Brussels, BE
info@ai.vub.ac.be https://ai.vub.ac.be

Abstract. In recent years Deep Reinforcement Learning [12] has achieved human-like performance or better on a variety of benchmarks such as the Atari Arcade [2]; however, Deep RL often has problems with sample efficiency and convergence brittleness. That is, to learn even the simplest tasks, Deep RL requires a huge amount of meaningful samples, and will only converge if the parameters are tuned *just* right [4]. This paper seeks to ameliorate these problems of sample inefficiency and convergence brittleness with the combination of two different reinforcement learning paradigms: Bayesian RL and Maximum Entropy RL.
Bayesian reinforcement learning [8] utilizes Bayesian statistics to model the confidence in a given model, which has been shown to greatly increase sample efficiency [3]. Maximum entropy RL is a technique that modifies the standard reward to promote more exploration in the agent [20]. Hopefully, combining the two will retain the best of both of these properties and avoid the problems faced in deep RL altogether.
This paper first derives a soft policy gradient that introduces a entropy-weighted term to the standard policy gradient function, and then applies this to the the Bayesian actor critic paradigm to augment the parameter update rule to account for the entropy-weighted value function. After determining a closed-form solution of the gradient with the softmax policy, the method was implemented and evaluated on the Cartpole environment, signalling that there are avenues ripe for future research in this area.

Keywords: Bayesian Reinforcement Learning · Maximum Entropy Reinforcement Learning · Gaussian Processes · Bayesian Quadrature · Bayesian Actor Critic

1 Introduction

This paper is structured as follows. The Background section is composed of a decent amount of exposition around Gaussian processes and Bayesian quadrature necessary to understand the Bayesian Actor Critic model. After explaining the details of Bayesian actor critic and maximum entropy RL, the main contribution

of the paper, Maximum Entropy Bayesian Actor Critic (MEBAC), is discussed. After the explication of MEBAC and a few implementation choices, the Results, Discussion, and Conclusion empirically analyze the model.

2 Background

2.1 Policy Gradient and Actor Critic Methods

When formulating the objective of the reinforcement learning problem, the goal is always to maximize the expected return. The way different methods formulate this objective determines the way in which the problem is approached. For instance, the Bellman equations [1] define the optimal state-value and action-value functions in terms of maximizing the expected return, resulting in value-based methods. Alternatively, one can formulate the objective in terms of the parameterization of a policy, and what results are methods that operate directly on the parameters of a policy instead of estimating value functions.

The policy gradient theorem [18] formulates the objective in terms of the parameters and finds the gradient of this objective. By moving along this gradient, the policy improves by directly adjusting its parameters. A baseline function can be added to the value function of the policy gradient to greatly reduce the variance of the gradient without affecting the expectation as long as it does not depend on the actions. By setting this baseline function to be the current estimate of the value function, we come to the actor-critic paradigm [11]. That is, the policy parameterized by $\boldsymbol{\theta}$ is the 'actor' and the estimate of the value function is the 'critic'.

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s;\boldsymbol{\theta}) \\ \text{meaning} \quad \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \beta \delta_t \nabla \log \pi(A_t|S_t;\boldsymbol{\theta_t}) \\ \text{where} \quad \delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1},\boldsymbol{w}) - \hat{v}(S_t,\boldsymbol{w}) \end{aligned} \tag{1}$$

2.2 Bayesian Reinforcement Learning

Bayesian reinforcement learning is a catch-all term for any reinforcement learning method that utilizes Bayes' rule in some capacity. In general, the model will retain a prior distribution, which represents the current approximation, that is updated with a posterior distribution when data is seen. As more data is seen, the variance of the prior tends to decrease, signalling greater confidence in the mean value of the distribution. The problem with this approach is that unlike many other optimization techniques found throughout that operate using derivatives, Bayesian updates tend to contain integrals, making them analytically intractable unless the proper priors are chosen.

$$p(h|\mathcal{D}) = \frac{p(\mathcal{D}|h)p(h)}{p(\mathcal{D})} \tag{2}$$

2.3 Bayesian Expectation

Many core algorithms in reinforcement learning must calculate some kind of expectation. This can be taken over any distribution, though often the expectation is taken over reward trajectories or value functions. In fact, value functions and the Bellman equation [1], perhaps the most core formalisms in all of reinforcement learning, are posed as an expectation over returns.

$$\begin{gathered} v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s], \quad \text{and} \quad q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a], \\ v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \end{gathered} \tag{3}$$

The primary difference between Bayesian Reinforcement Learning and other types of reinforcement learning is that other methods tend to use a frequentist, Monte Carlo approach to calculating the expectation. That is, by averaging over many samples of the value in question, Monte Carlo approaches can find unbiased approximations of the expectation of that value. This method has the benefit of being conceptually and practically simple, but may take many, many values to converge to a good approximation. In contrast, Bayesian reinforcement learning utilizes Bayesian methods to approximate this expectation. By performing a posterior update of a prior distribution whenever new data is seen, Bayesian methods can be much more sample efficient than Monte Carlo methods.

2.4 Gaussian Processes

Since in reinforcement learning we are often concerned with estimating functions, like state-value or action-value functions, Bayesian RL methods tend to place Gaussian processes over those functions in order to apprximate them. In the same way that a Gaussian distribution is probability distribution over points, a Gaussian process can be thought of as a distribution over functions [14]. By thinking of functions in terms of Hilbert spaces [10], one can equivalently think of a Gaussian process as just a multivariate distribution over infinite dimensions. A Gaussian process is fully defined by its mean and covariance, such that $f(\cdot) \sim \mathcal{N}(\bar{f}(\cdot), k(\cdot,\cdot))$, where $\bar{f}$ is the mean function and k is the kernel function.

$$\boldsymbol{E}[f(x)] = \bar{f}(x) \quad \text{and} \quad \boldsymbol{Cov}[f(x), f(x')] = k(x,x') \tag{4}$$

With posterior moments,

$$\begin{aligned} \boldsymbol{E}[f(x)|\mathcal{D}_M] &= \bar{f}(x) + \boldsymbol{k}(x)^\top \boldsymbol{C}(\boldsymbol{y} - \bar{\boldsymbol{f}}), \\ \boldsymbol{Cov}[f(x), f(x')|\mathcal{D}_M] &= k(x,x') - \boldsymbol{k}(x)^\top \boldsymbol{C}\boldsymbol{k}(x'). \end{aligned} \tag{5}$$

Where,

$$\begin{gathered} \bar{\boldsymbol{f}} = (\bar{f}(x_1), \ldots, \bar{f}(x_M))^\top, \quad \boldsymbol{y} = (y(x_1), \ldots, y(x_M))^\top, \\ \boldsymbol{k}(x) = (k(x_1,x), \ldots, k(x_M,x))^\top, \quad [\boldsymbol{K}]_{i,j} = k(x_i,x_j), \quad \boldsymbol{C} = (\boldsymbol{K} + \boldsymbol{\Sigma})^{-1}, \end{gathered} \tag{6}$$

2.5 Bayesian Quadrature

Bayesian Quadrature [13] is a method aimed at approximating integrals composed of the form:

$$\rho = \int f(x)g(x)dx \tag{7}$$

One can immediately see that if either $f(x)$ or $g(x)$ is a probability density distribution, then this integral is just an expectation. Bayesian quadrature aims to approximate this integral by modeling one of the component functions as a Gaussian process, and performing a posterior update when new data is seen. It has been shown that we can achieve much more accurate approximation of expectations in comparison to Monte Carlo techniques using this method [15].

Taking $f(x)$ as the random function to be modeled, the posterior moments of this integral are given by:

$$\begin{aligned} \boldsymbol{E}[\rho|\mathcal{D}_M] &= \int \boldsymbol{E}[f(x)|\mathcal{D}_M]g(x)dx, \\ \boldsymbol{Var}[\rho|\mathcal{D}_M] &= \iint \boldsymbol{Cov}[f(x), f(x')|\mathcal{D}_M]g(x)g(x')dxdx'. \end{aligned} \tag{8}$$

Rewriting,

$$\boldsymbol{E}[\rho|\mathcal{D}_M] = \rho_0 + \boldsymbol{b}^\top \boldsymbol{C}(\boldsymbol{y} - \bar{\boldsymbol{f}}) \quad \text{and} \quad \boldsymbol{Var}[\rho|\mathcal{D}_M] = b_0 - \boldsymbol{b}^\top \boldsymbol{C}\boldsymbol{b}, \tag{9}$$

Where,

$$\begin{aligned} \boldsymbol{b} &= \int \boldsymbol{k}(x)g(x)dx. \\ \rho_0 = \int \bar{f}(x)g(x)dx, \quad b_0 &= \iint k(x,x')g(x)g(x')dxdx'. \end{aligned} \tag{10}$$

The only issue remaining is to ensure that these integrals are analytically tractable, which can be done by choosing $\bar{f}(x)$, $g(x)$, and $k(x, x')$ appropriately.

2.6 Bayesian Policy Gradient

By formulating the Policy Gradient Theorem in terms of Bayesian quadrature, a Bayesian policy gradient method [6] can be derived as follows. First, we specify the policy gradient theorem for the continuous case, choosing modeled random function $f(x)$ and known function $g(x)$ from Bayesian quadrature as follows:

$$\nabla J(\boldsymbol{\theta}) = \int d\boldsymbol{z} \underbrace{\mu_\pi(\boldsymbol{z};\boldsymbol{\theta})\nabla \log \pi(a|s;\boldsymbol{\theta})}_{g(x)} \underbrace{q_\pi(\boldsymbol{z};\boldsymbol{\theta})}_{f(x)} \tag{11}$$

where $\mu_\pi(s,a) = \sum_{t=0}^{\infty} \gamma^t P_t(s,a)$ is the discounted state-action occupancy for policy π, and $\boldsymbol{z} \in \mathcal{Z}$ are state-action pairs (s,a). This results in the posterior moments,

$$\begin{aligned}\boldsymbol{E}[\nabla J(\boldsymbol{\theta})|\mathcal{D}_t] &= \int_{\mathcal{Z}} d\boldsymbol{z}\, \boldsymbol{g}(\boldsymbol{z};\boldsymbol{\theta})\boldsymbol{E}[q_\pi(\boldsymbol{z};\boldsymbol{\theta})|\mathcal{D}_t], \\ \boldsymbol{Cov}[\nabla J(\boldsymbol{\theta})|\mathcal{D}_t] &= \int_{\mathcal{Z}^2} d\boldsymbol{z}d\boldsymbol{z'}\, \boldsymbol{g}(\boldsymbol{z};\boldsymbol{\theta})\boldsymbol{Cov}[q_\pi(\boldsymbol{z}), q_\pi(\boldsymbol{z'})|\mathcal{D}_t]\boldsymbol{g}(\boldsymbol{z'};\boldsymbol{\theta})^\top\end{aligned} \tag{12}$$

Therefore, to calculate this integral, we need the posterior moments of the Gaussian process modelling q_π.

2.7 Gaussian Process Temporal Difference Learning

One way of approximating the action-value function is Gaussian Process Temporal Difference Learning (GPTD) [5]. This method takes the standard temporal difference learning paradigm, and instead of performing max-updates like Q-learning or expected updates like Expected Sarsa [17], GPTD places a Gaussian process over the value function v or q and performs posterior updates using the reward at each time step to approximate it with $\hat{v}$ or $\hat{q}$ respectively. As more data is seen, the variance of the estimate decreases, directly showing the confidence in the estimation.

By modelling the relationship between the reward signal r and the action-value function q_π as a temporal difference between processes with noise N, the posterior moments of the action-value process can be found, providing an approximation for q_π.

$$\begin{aligned}r(\boldsymbol{z}_t) &= q_\pi(\boldsymbol{z}_t) - \gamma q_\pi(\boldsymbol{z}_{t+1}) + N(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}) \\ \hat{q}_\pi(\boldsymbol{z}) &= \boldsymbol{E}[q_\pi(\boldsymbol{z})|\mathcal{D}_t] = \boldsymbol{k}_t(\boldsymbol{z})^\top \boldsymbol{\alpha}_t, \\ \hat{s}_\pi(\boldsymbol{z}, \boldsymbol{z'}) &= \boldsymbol{Cov}[q_\pi(\boldsymbol{z}), q_\pi(\boldsymbol{z'})|\mathcal{D}_t] = k(\boldsymbol{z}, \boldsymbol{z'}) - \boldsymbol{k}_t(\boldsymbol{z})^\top \boldsymbol{C}\boldsymbol{k}_t(\boldsymbol{z'})\end{aligned} \tag{13}$$

Where,

$$\begin{aligned}\boldsymbol{\alpha}_t &= \boldsymbol{H}_t^\top (\boldsymbol{H}_t \boldsymbol{K}_t \boldsymbol{H}_t^\top + \boldsymbol{\Sigma}_t)^{-1} \boldsymbol{r}_{t-1}, \\ \boldsymbol{C}_t &= \boldsymbol{H}_t^\top (\boldsymbol{H}_t \boldsymbol{K}_t \boldsymbol{H}_t^\top + \boldsymbol{\Sigma}_t)^{-1} \boldsymbol{H}_t.\end{aligned} \tag{14}$$

with $\boldsymbol{H}_t$ representing the discount matrix and $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{H}_t \boldsymbol{H}_t^\top$ representing the noise covariance.

2.8 Bayesian Actor Critic

Now, to move to a Bayesian actor critic [7], we take the Bayesian policy gradient and estimate the action-value function using Gaussian Process Temporal Difference Learning. GPTD provides the posterior moments for the Bayesian policy gradient, and by utilizing the Fisher information kernel [16], we get a closed-form solution to the integral, resulting in the Bayesian Actor Critic.

$$\begin{aligned} \boldsymbol{E}[\nabla J(\boldsymbol{\theta})|\mathcal{D}_t] &= \int_{\mathcal{Z}} d\boldsymbol{z}\boldsymbol{g}(\boldsymbol{z};\boldsymbol{\theta})\boldsymbol{k}_t(\boldsymbol{z})^\top \boldsymbol{\alpha}_t, \\ \boldsymbol{Cov}[\nabla J(\boldsymbol{\theta})|\mathcal{D}_t] &= \int_{\mathcal{Z}^2} d\boldsymbol{z}d\boldsymbol{z'}\, \boldsymbol{g}(\boldsymbol{z};\boldsymbol{\theta})\big(k(\boldsymbol{z},\boldsymbol{z'}) - \boldsymbol{k}_t(\boldsymbol{z})^\top \boldsymbol{C}_t \boldsymbol{k}_t(\boldsymbol{z'})\big)\boldsymbol{g}(\boldsymbol{z'};\boldsymbol{\theta}). \end{aligned} \tag{15}$$

Rewriting,

$$\boldsymbol{E}[\nabla J(\boldsymbol{\theta})|\mathcal{D}_t] = \boldsymbol{B}_t\boldsymbol{\alpha}_t \quad \text{and} \quad \boldsymbol{Cov}[\nabla J(\boldsymbol{\theta})|\mathcal{D}_t] = \boldsymbol{B}_0 - \boldsymbol{B}_t\boldsymbol{C}_t\boldsymbol{B}_t^\top, \tag{16}$$

Where,

$$\begin{aligned} \boldsymbol{B}_t &= \int_{\mathcal{Z}} d\boldsymbol{z}\, \boldsymbol{g}(\boldsymbol{z};\boldsymbol{\theta})\boldsymbol{k}_t(\boldsymbol{z})^\top, \\ \boldsymbol{B}_0 &= \int_{\mathcal{Z}^2} d\boldsymbol{z}d\boldsymbol{z'}\, \boldsymbol{g}(\boldsymbol{z};\boldsymbol{\theta})k(\boldsymbol{z},\boldsymbol{z'})\boldsymbol{g}(\boldsymbol{z'};\boldsymbol{\theta})^\top. \end{aligned} \tag{17}$$

To make these tractable, the kernel $k = k_f + k_s$ becomes a combination of the state kernel $k_s(s, s')$ and Fisher information kernel $k_f(\boldsymbol{z}, \boldsymbol{z'}) = \boldsymbol{u}(\boldsymbol{z};\boldsymbol{\theta})^\top \boldsymbol{G}^{-1}\boldsymbol{u}(\boldsymbol{z'};\boldsymbol{\theta})$, with $\boldsymbol{u}(\boldsymbol{z};\boldsymbol{\theta})$ the information score function and $\boldsymbol{G}$ the Fisher information matrix, where

$$\begin{gathered} \boldsymbol{u}(\boldsymbol{z};\boldsymbol{\theta}) = \nabla \log \pi(a|s;\boldsymbol{\theta}), \quad \boldsymbol{U}_t = [\boldsymbol{u}(\boldsymbol{z}_0), \ldots, \boldsymbol{u}(\boldsymbol{z}_t)], \\ \text{estimating} \quad \hat{\boldsymbol{G}}_t = \frac{1}{t+1}\boldsymbol{U}_t\boldsymbol{U}_t^\top \\ \text{meaning} \quad \boldsymbol{B}_t = \boldsymbol{U}_t \quad \text{and} \quad \boldsymbol{B}_0 = \boldsymbol{G} \end{gathered} \tag{18}$$

Finally, this leads to the Bayesian actor critic policy update rule, using the conventional or natural gradient [7].

$$\begin{gathered} \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \beta\Delta\boldsymbol{\theta} \quad \text{where} \\ \Delta\boldsymbol{\theta} = \boldsymbol{U}_t\boldsymbol{\alpha}_t \quad \text{(Convential gradient)} \\ \Delta\boldsymbol{\theta} = \hat{\boldsymbol{G}}_t^{-1}\boldsymbol{U}_t\boldsymbol{\alpha}_t \quad \text{(Natural Gradient)} \end{gathered} \tag{19}$$

Therefore, the Bayesian actor critic improves the parameters of its policy by iteratively moving along the gradient by applying the following three steps for each cycle, where a cycle is M steps long:

- Use GPTD to estimate $\hat{q}_\pi$, returning $\boldsymbol{\alpha}_t$ and $\boldsymbol{C}_t$.
- Compute $\boldsymbol{U}_t$ and estimate $\hat{\boldsymbol{G}}_t$
- Calculate $\Delta\boldsymbol{\theta}$ and update the parameters $\boldsymbol{\theta}$

2.9 Maximum Entropy Reinforcement Learning

Normally, the standard objective function is all about maximizing the return. Maximum entropy reinforcement learning makes a change to this core goal. Instead of solely maximizing the return, the maximum entropy objective [20] seeks to maximize the return and policy entropy.

$$J^s(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\big[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\big] \tag{20}$$

By simultaneously seeking out high rewards while seeking high entropy – that is, acting as randomly as possible – the agent will explore more. In addition, if there are multiple optimal or near optimal policies, the agent will be able to give equal weighting to all of them instead of assigning all the mass to a single policy of the set. In practice, it has been shown [9] that this objective also improves the learning rate and stability of the algorithms.

3 Maximum Entropy Bayesian Actor Critic

The goal of Maximum Entropy Bayesian Actor Critic (MEBAC) is to take the maximum entropy objective function and alter the Bayesian actor critic machinery to work with it efficiently. Though the Soft Actor Critic [9] algorithm was the initial inspiration for utilizing the maximum entropy objective, the only things MEBAC takes from it are that objective, and the soft value function. The soft value function alters the standard value function with an entropy-like term that will give weight to more random policy states.

$$v^s_\pi(s_t) = \sum_a \pi(a|s)[q_\pi(s_t,a_t) - \log\pi(a_t|s_t)] \tag{21}$$

3.1 Soft Policy Gradient

In the same way as the policy gradient theorem is derived from the standard value function, we begin with the soft value function v^s_π and derive the soft policy gradient $\nabla J^s(\boldsymbol{\theta})$:

$$\begin{aligned}
\nabla v^s_\pi &= \nabla \sum_a \pi(a|s)[q_\pi(s,a) - \log\pi(a|s)] \\
&= \sum_a \big[\nabla\pi(a|s)[q_\pi(s,a) - \log\pi(a|s)] + \pi(a|s)\nabla[q_\pi(s,a) - \log\pi(a|s)]\big] \\
&= \sum_a \big[\nabla\pi(a|s)[q_\pi(s,a) - \log\pi(a|s)] + \pi(a|s)\nabla q_\pi(s,a) - \nabla\pi(a|s)\big] \\
&= \sum_a \big[\nabla\pi(a|s)[q_\pi(s,a) - \log\pi(a|s) - 1] + \pi(a|s)\nabla q_\pi(s,a)\big] \\
&= \sum_{x\in\mathcal{S}}\sum_{k=1}^{\infty} Pr(s\to x,k,\pi)\sum_a \big[\nabla\pi(a|s)[q_\pi(s,a) - \log\pi(a|s) - 1]\big] \\
\nabla J^s(\boldsymbol{\theta}) = \nabla v^s_\pi(s_0) &\propto \sum_s \mu(s)\sum_a \nabla\pi(a|s)[q_\pi(s,a) \underbrace{-\log\pi(a|s) - 1}_{\text{Entropy "Baseline"}}]
\end{aligned} \tag{22}$$

It is clear that this entropy term occupies the same location as a baseline; however, since it is a function of both the state and actions, it not a true baseline, and therefore will have an effect on the mean. This "baseline" term will nudge the gradient in the direction of higher entropy, which matches the intuition of using a maximum entropy objective.

3.2 Max Entropy Bayesian Actor Critic

Converting the soft policy gradient to the integral form,

$$\begin{aligned}\nabla J^s(\boldsymbol{\theta}) &= \int d\boldsymbol{z}\, \mu_\pi(s,a) \nabla \log \pi(a|s)[q_\pi(s,a) - \log \pi(a|s) - 1] \\ &= \int d\boldsymbol{z}\, \mu_\pi(s,a) \nabla \log \pi(a|s) q_\pi(s,a) \\ &\quad + \int d\boldsymbol{z}\, \mu_\pi(s,a) \nabla \log \pi(a|s)(-\log \pi(a|s) - 1)\end{aligned} \tag{23}$$

The first integral is the same as the regular Bayesian Actor Critic that will update the gradient in response to newly observed rewards, whereas the second integral is the contribution of the policy entropy to the gradient. This means we can utilize the same machinery as discussed earlier for the regular Bayesian Actor Critic, and augment that term with the entropy contribution to the gradient.

We could utilize the Bayesian quadrature approach to the approximate the entropy contribution integral, which would likely result in more accurate estimations of the integral; however, here we will instead elect to estimate it using a simple unweighted average. Similar to the estimate of the Fisher kernel $\hat{\boldsymbol{G}}$, we define an augmented score function $\boldsymbol{w}(\boldsymbol{z})$ and estimate an unbiased estimate of the integral.

$$\begin{aligned}\boldsymbol{w}(\boldsymbol{z}) &= \nabla \log \pi(a|s;\boldsymbol{\theta})[-\log \pi(a|s;\boldsymbol{\theta}) - 1] \\ \boldsymbol{w}_t &= \frac{1}{t+1} \sum_{i=0}^{t} \boldsymbol{w}(\boldsymbol{z}_i)\end{aligned} \tag{24}$$

Finally, the maximum entropy parameter update becomes,

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \beta \Delta\boldsymbol{\theta} \quad \text{where} \\ \Delta\boldsymbol{\theta} &= \boldsymbol{U}_t \boldsymbol{\alpha}_t + \boldsymbol{w}_t \quad \text{(Convential gradient)} \\ \Delta\boldsymbol{\theta} &= \hat{\boldsymbol{G}}_t^{-1} \boldsymbol{U}_t \boldsymbol{\alpha}_t + \boldsymbol{w}_t \quad \text{(Natural Gradient)}\end{aligned} \tag{25}$$

4 Implementation

4.1 Softmax Policy for Discrete Actions

Since we are taking the gradient of the policy, it is necessary to have a differentiable policy. Even more, we need the gradient of the log of the policy to also be

differentiable, and would ideally like all of those derivatives to play well together. For the case of discrete actions, the Softmax policy fits the bill.

$$\begin{aligned}
\pi(a_i|s) &= \frac{\exp(\boldsymbol{\phi}(s,a_i)^\top \boldsymbol{\theta})}{\sum_{a_j \in A} \exp(\boldsymbol{\phi}(s,a_j)^\top \boldsymbol{\theta})} \\
\frac{\nabla \pi(a|s)}{\boldsymbol{\phi}(s,a)} &= \begin{cases} \pi(a_i|s)[1-\pi(a_j|s)] & \text{if } i = j \\ -\pi(a_i|s)\pi(a_j|s) & \text{if } i \neq j \end{cases} \\
\frac{\nabla \log \pi(a|s)}{\boldsymbol{\phi}(s,a)} &= \begin{cases} 1-\pi(a_i|s) & \text{if } i = j \\ -\pi(a_j|s) & \text{if } i \neq j \end{cases}
\end{aligned} \tag{26}$$

where $\boldsymbol{\phi}(s,a)$ is the state-action feature vector.

4.2 Technologies

The GPTD [5] and BAC [7] algorithms were re-implemented from scratch primarily using the NumPy [19] scientific Python library. They were evaluated using OpenAI Gym [2] CartPole environment with *Discrete* action spaces and *Box* observation spaces. The sparsification procedure defined and used in the literature led to no learning at all, so it was removed in this implementation.

5 Results

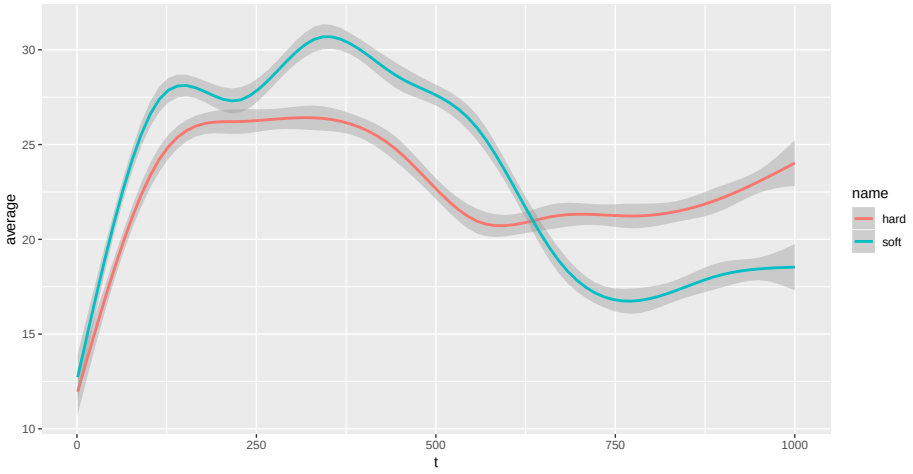

Fig. 1. Cumulative Return for Cartpole-V0 Environment

As can be seen in the plot of cumulative return, neither method performs particularly well. Though it cannot be seen in this plot, it was found that either

the algorithms converged to the optimal policy almost immediately, or didn't learn at all. Interestingly, a common occurrence was to improve for the first couple hundred episodes, followed by a "collapse" where it seems that everything learned up to that point was thrown away. This is most likely due to the choice of the Softmax policy overflowing the capabilities of NumPy, resulting in essentially random gradient updates and therefore behavior. In the future, more effort could be put into finding a good parameterization or policy such that this "collapse" does not happen.

6 Discussion

At least for the Cartpole environment, the algorithm either doesn't learn, or converges to a deterministic policy almost immediately, for both the hard version and the soft version. This identical behavior makes sense, since a deterministic policy is zero-entropy, meaning that the entropy term will not factor in, resulting in identical behavior between the two methods. Therefore, unfortunately these results do not give any insight as to whether the MEBAC is a feasible reinforcement learning model.

7 Conclusion

7.1 Problems & Limitations

The main problem with the implementation was that the gradient was usually so large as to move the policy almost immediately into a deterministic policy. This makes the addition of the entropy term useless, not allowing us to investigate the efficacy of the method. Another option would be to evaluate on a different, more difficult environment with a larger state-action space. This would likely reduce the rate of the gradient change and allow for the entropy term to have some sort of impact.

Many different methods were attempted to slow down the gradient change beyond altering the temperature coefficient. Normalizing the gradient succeeded in slowing down the growth, but simply resulted in poorer performance overall, since the gradient is moving the right direction in general. Normalizing and exponentiating the state vector was also attempted, which had little to no effect on the growth, as the state vector is relatively small in magnitude to begin with. Gradient clipping is another method that might be used to slow down the gradient change, but was not utilized here.

In the end, the state-action feature vector $\boldsymbol{\phi}(s, a)$ was purposely naïve, meaning it did not use any domain knowledge to create features, and was simply dotted with the parameter vector. This was intended to keep the method general, so as to be tested against multiple environments; however, it seems to have backfired. Using a domain knowledge to create a feature vector would most likely result in more interesting results and performance.

7.2 Future Work

Though the results of the implementation were inconclusive, the derivation of the Maximum Entropy Bayesian Actor Critic model seems ripe for future research. There are a few avenues of inquiry that could be investigated going forward. As already mentioned, using domain knowledge to create an environment-specific feature vector may yield more interesting results on more difficult environments. Next, by opening up to a continuous action space, one could maintain a prior distribution of a Gaussian for each action parameter, which results in a nice closed-form solution for the gradient and log-gradient like the Softmax derivation. Finally, instead of using hand-crafted feature vectors, one could learn the feature vector through any number of methods, including deep RL. This is an interesting avenue because it maintains generality while circumventing the feature vector problem.

References

1. Bellman, R.: Dynamic programming. Science **153**(3731), 34–37 (1966)
2. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
3. Chapelle, O., Li, L.: An empirical evaluation of thompson sampling. In: Advances in neural information processing systems. pp. 2249–2257 (2011)
4. Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P.: Benchmarking deep reinforcement learning for continuous control. In: International Conference on Machine Learning. pp. 1329–1338 (2016)
5. Engel, Y., Mannor, S., Meir, R.: Reinforcement learning with gaussian processes. In: Proceedings of the 22nd international conference on Machine learning. pp. 201–208. ACM (2005)
6. Ghavamzadeh, M., Engel, Y.: Bayesian policy gradient algorithms. In: Advances in neural information processing systems. pp. 457–464 (2007)
7. Ghavamzadeh, M., Engel, Y., Valko, M.: Bayesian policy gradient and actor-critic algorithms. The Journal of Machine Learning Research **17**(1), 2319–2371 (2016)
8. Ghavamzadeh, M., Mannor, S., Pineau, J., Tamar, A., et al.: Bayesian reinforcement learning: A survey. Foundations and Trends® in Machine Learning **8**(5-6), 359–483 (2015)
9. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290 (2018)
10. Kennedy, R.A., Sadeghi, P.: Hilbert space methods in signal processing. Cambridge University Press (2013)
11. Konda, V.R., Tsitsiklis, J.N.: Actor-critic algorithms. In: Advances in neural information processing systems. pp. 1008–1014 (2000)
12. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436 (2015)
13. O'Hagan, A.: Bayes–hermite quadrature. Journal of statistical planning and inference **29**(3), 245–260 (1991)
14. Rasmussen, C.E.: Gaussian processes in machine learning. In: Summer School on Machine Learning. pp. 63–71. Springer (2003)
15. Rasmussen, C.E., Ghahramani, Z.: Bayesian monte carlo. Advances in neural information processing systems pp. 505–512 (2003)

16. Shawe-Taylor, J., Cristianini, N., et al.: Kernel methods for pattern analysis. Cambridge university press (2004)
17. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
18. Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in neural information processing systems. pp. 1057–1063 (2000)
19. Van Der Walt, S., Colbert, S.C., Varoquaux, G.: The numpy array: a structure for efficient numerical computation. Computing in Science & Engineering **13**(2), 22 (2011)
20. Ziebart, B.D.: Modeling purposeful adaptive behavior with the principle of maximum causal entropy (2010)