N-step Bootstrapping in Actor Critic Methods

Davide Barbieri, Jan Schutte, Hinrik Snær Guðmundsson, Zi Long Zhu

I. Introduction

In model-free reinforcement learning there are three options for learning to solve the problem at hand: learning a value function, learning a policy or learning both. These methods are called critic-only, actor-only and actor-critic methods respectively. In this paper we will investigate how bootstrapping affects actor-critic methods and how they compare to Monte Carlo returns and 1-step methods.

Bootstrapping mitigates the problem of high variance in Monte Carlo methods, but might in return introduce biases if our estimates do not converge to the true policy/value function.

It is thus essential to understand how this bias-variance trade-off can impact reinforcement learning agents and how n-step bootstrapping can mitigate this problem.

In this report, we will mainly focus on generalized advantage estimation (GAE) (Schulman et al., 2015) as an n-step bootstrapping actor-critic method. To understand the effects that n-steps bootstrapping have using GAE, we decided to also test a simpler 1-step bootstrapping predecessor of GAE, Advantage Estimators (AE). We will perform tests using the Open AI *CartPole-v0* and *Acrobot-v1* gym environments. (Sutton, 1996)

We will analyze what impact the hyperparameters from GAE can have on the optimality and the convergence-speed of the algorithm and compare it to the Monte-Carlo method (REINFORCE). We will also investigate what impact the aforementioned measurements using Generalized Advantage Estimation has over vanilla Advantage Estimation.

II. Method

In this section, we will cover in detail the experiments we performed. First we outline the environments we used and then the actor-critic methods will be discussed.

A. Environments

We have chosen the CartPole and Acrobot environments for our experiments. The rationale for choosing these two, is that their goals have the following differences: CartPole tries to sustain a certain state while Acrobot tries to reach a specific goal. This suggests that the exploration of the latter will have a bigger impact on the overall algorithm's performance. The aim is to see if there are differences in performance between the methods under these two different environmental setups. We also want to explore if in some settings, the weaknesses of some of these methods can be mitigated. In the following subsections we explain the environments in more detail. In the **CartPole-v0** environment, a pole is attached to a cart by a free joint and the cart moves along a friction track. The goal is for the cart to prevent the pole from falling over by applying a force towards one of two possible directions using a unit value, indicated with +1 and -1. In the **Acrobot-v1** environment there is a single agent in the system, where the agent consists of two joints and two links. The joint between the two links is actuated. The agent starts hanging downwards and the goal is to swing the lower link up to a certain height. The three possible actions are applying either +1, 0 or 1 torque on the joint between the two links.

B. Algorithms

All the methods we will consider, involve policy gradients. Policy gradient methods are in general advantageous as they naturally learn stochastic policies and can also account for continuous actions (a feature not considered in the aforementioned experiments).

The general idea is to have a differentiable function approximate the policy. The problem is then reduced to computing the gradients that will update the parameter of the approximation function in such a way that at convergence, the approximation will be (as close as possible to) the optimal strategy.

1) Monte Carlo Methods: One way to tackle the aforementioned problem by utilizing Monte-Carlo methods is by using the REINFORCE algorithm. REINFORCE aims to find the gradient that locally maximizes the expected future discounted return, formally:

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau} \left(G(\tau) \right) \approx \frac{1}{N} \sum_{i=1}^{N} \left(G(\tau_i) \sum_{t=0}^{|\tau_i|} \nabla_{\boldsymbol{\theta}} \log \pi(a_t^i \mid s_t^i) \right) \tag{1}$$

where τ and τ_i are arbitrary trajectories. We can see from equation 1 that to compute such a gradient, we estimate the overall return for a trajectory. This allows for an unbiased estimate of the true return value, as more samples are collected.

However, this also requires the algorithm to wait until the end of an episode to perform a training step. Such an algorithm directly estimates returns according to their definition and observations, but makes no use of the relationships that state values have (mainly given by the Bellman equation) which can speed up learning and reduce the variance during training.

2) Actor Critic Methods: Often in machine learning there is a trade-off between variance and bias. Simple policy gradient methods such as REINFORCE are unbiased, but

suffer from high variance. While one-step methods suffer from high bias, but have low variance. To mitigate this trade-off, we need to look at methods in between, which includes the n-step bootstrapping method. Actor-critic methods allow for bootstrapping (performed on value functions) to be used in policy gradient methods.

In general, bootstrapping methods make use of the aforementioned relationships between state values expressed by the Bellman equation, which allow us to express the value of a given state S_t as

$$Q(s_t, a_t) = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n V(s_{t+n}) \tag{2}$$

which intuitively computes an unbiased estimate of the return up to S_{t+n} and then approximates the rest of the return with the utility of S_{t+n} (formally $\gamma^n V(S_{t+n})$).

Actor-critic methods allow us to make use of this bootstrapping of the state values, while still retaining the advantages of policy gradient methods, or better, the advantages of learning the policy directly. The general idea is that the policy can be trained with the following loss

$$\nabla_{\boldsymbol{\theta}} J \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{|\tau_i|} \Psi_t \nabla_{\boldsymbol{\theta}} \log \pi(a_t^i \mid s_t^i) \right) \tag{3}$$

It is worth noting that Ψ_t is now a generalization of the estimated return for a certain action a_t^i in a state s_t^i. In fact, if $\psi_t = G(\tau_i)$ then this would be equivalent to the REINFORCE loss.

To exploit bootstrapping, a 'critic' can then be used to approximate Ψ_t.

3) Advantage Estimation: Advantage estimators use the advantage function to estimate the value of an action a_t^i in a state s_t^i for the policy gradient $\nabla_{\boldsymbol{\theta}} J$. Formally we have

$$\Psi_t = Q(s,a_t) - V(s_t) \approx A_t^{(n)}$$

where given formula 2 we can express the advantage for a given n as

$$A_t^{(n)} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n V(s_{t+n}) - V(s_t)$$

given an approximation of V (which is our critic), and hyperaprameter n. We can notice that for $n \to \infty$ then $A_t^{(n)}$ tends to have a lower bias (being closer to a shifted version of the REINFORCE loss), but consequently a higher variance. Since the bootstrapping in the computation of the advantage already introduces a bias, it is usually advisable to learn V with an unbiased approximation (e.g. Monte-Carlo methods with trust regions).

The key concept of advantage estimations is that an approximated value function can be used to compute the estimated values of state-action pairs in the trajectory (for the policy gradient) using bootstrapping. This is expected to reduce the variance of the algorithm (while it introduces a bias).

4) Generalized Advantage Estimation: In generalized advantage estimation all estimates $A_t^{(n)}$ are used for Ψ_t, and are combined with parameter λ into a single exponentially-weighted average:

$$A_t^{GAE(\gamma,\lambda)} = (1-\lambda)\Big(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + \cdots\Big) \tag{4}$$

Which can be rewritten using Bellman residual terms into:

$$\sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \tag{5}$$

The hyperparameter λ, for $0 < \lambda < 1$, controls the bias variance trade-off. Where values near 0 introduce bias but have low variance, and values near 1 have high variance but low bias.

What this algorithm is effectively doing, as suggested by equation 5, is performing a discounted sum of future advantages, allowing the model to consider multiple possible ratios of estimating and approximating parts of a state's future discounted return. Intuitively, λ is a soft boundary on n.

C. Metrics

We aim at comparing the trade-off between bias and variance of these algorithms, and the effect the hyperparameters have on these two quantities.

To do so, as proposed by Taylor and Stone (2009), we will use two main metrics.

a) Asymptotic Performance: This metrics aims at measuring the asymptotic value to which the average return per episode of the agent converges to over a series of runs. We opted for this metrics as a proxy for the bias in the models. The intuition is that a higher bias will result in a sub-optimal performance of the strategy to which the algorithm converges.

b) (Normalized) Area under Curve (AUC): This metric calculates the area under the curve of the normalized returns per episode. These returns are normalized so that the minimum return equals 0 and their maximum return 1. Because these returns are normalized this metric measures speed of convergence of the algorithms independently to the asymptotic performance. An algorithm that converges slowly has values near 0% and an algorithm that converges quickly has values near 100%.

D. Experimental Setup

For each setting we will perform 100 runs (to average the results) and in each run we will run 1000 episodes. The latter was selected based on the fact that on average, the observed methods would be close to convergence long before the 1000th episode, which is a good compromise between having a reliable quantity of data, and computational feasibility. The returns per episode observed are averaged over 100 random

initializations. Asymptotic Performance and AUC metrics are then computed for each method and each hyperparameter setting. The metrics are then compared with the baselines. We also report the standard error of the mean (SEM) for all metrics.

For the experiments we ran two different GAE with $\lambda \in \{0.92, 0.99\}$. Our justification for this is that in the work of Schulman et al. (2015) they have shown that the λ values between $[0.92, 0.99]$ work the most effective, and we selected the extreme values of this interval since we do not aim at optimizing for λ but at better observing what it's effect is on the model's performance.

For the AE experiment we performed only 1-step ($n = 1$) bootstrappings since, as for λ, we wanted to examine the effects of extreme (low-high) values of n in the bootstrapping

To keep the comparison fair between the different methods, we use the Adam optimizer (Kingma and Ba, 2017) with a starting learning rate $\alpha = 0.01$ for all methods except REINFORCE, with a fitted $\alpha = 0.001$ (to compensate for the higher variance). The training is done with a batch size of 1. Lastly, each method uses a discount factor of $\gamma = 0.99$.

III. Results

In Figure 1 and 2 we see the averaged returns over the number of episode learned for each environment, respectively. We observe that REINFORCE performs the best in the long run for both environments. This is supported by the asymptotic performance estimate in Figure 4, where in both environments the lower bound of the estimated asymptotic performance of REINFORCE (subtracting the SEM) is similar or higher than the upper bounds of the asymptotic performances of the other algorithms. However, in Acrobot (Figure 2) REINFORCE is the slowest to converge, as quantified by the normalized AUC shown in Figure 3, while for Cartpole the only algorithms slower than it is GAE($\lambda = 0.92$), which seems to underfit the task, on average slowing down the average return per episode after around 50 episodes.

In fact, the difference between the two environments is the performance between the other three methods, GAE($\lambda = 0.99$), GAE($\lambda = 0.92$) and AE. In the CartPole environment, on average, the three methods converge with similar rates for about the first 50 episodes. Afterwards, the rate of learning decelerates for GAE($\lambda = 0.92$) abruptly and performs the worst out of the three. Figure 3 supports that in CartPole GAE($\lambda = 0.99$) and AE are the fastest algorithms to converge, while GAE($\lambda = 0.92$) is the slowest. However, the normalized AUC also shows that the three are reliably faster to converge than REINFORCE in Acrobot, with GAE($\lambda = 0.92$) similarly also having the worst and the best asymptotic performance in CartPole and Acrobot respectively out of the four methods.

In general GAE($\lambda = 0.99$) and AE converge at a much faster pace compared to the REINFORCE method, and from Figures 1 and 2 it can be observed that on average this is especially true in the first 150 episodes. The performances in either environment of GAE($\lambda = 0.99$) do not seem to be significantly different from that of REINFORCE.

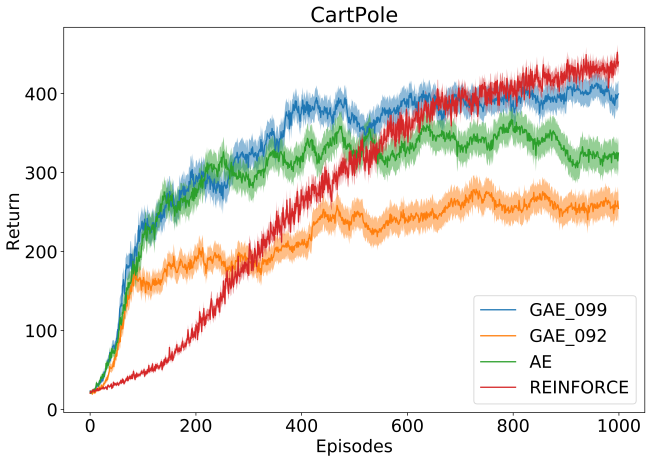

Fig. 1: The average learning curve for the CartPole environment, including SEM confidence intervals.

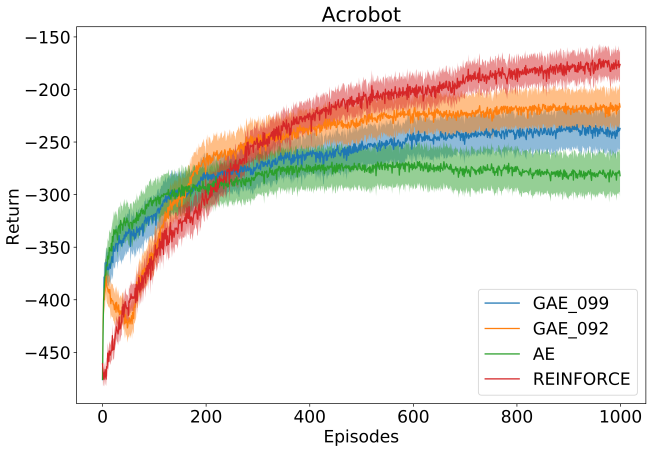

Fig. 2: The average learning curve for the Acrobot environment, including SEM confidence intervals.

IV. Discussion

It is notable that when comparing GAE($\lambda = 0.99$) with REINFORCE, the data matches the predictions from our hypothesis, which is that REINFORCE shows lower bias by having on average a better asymptotic performance, while its higher variance results in slower convergence times (lower normalized AUC), in both environments.

Moreover, AE is expected to have a higher bias with respect to GAE, since, using only 1-step bootstrapping, most of the expected return of action a_t is approximated using the critic's prediction. Consistently with this prediction, the asymptotic performance of AE is on average worse than that of GAE($\lambda = 0.99$). The effect of AE on the variance is not as consistent however, with the algorithm converging faster than GAE($\lambda = 0.99$) in Acrobot (on average), but having a lower mean normalized AUC than GAE in CartPole. In

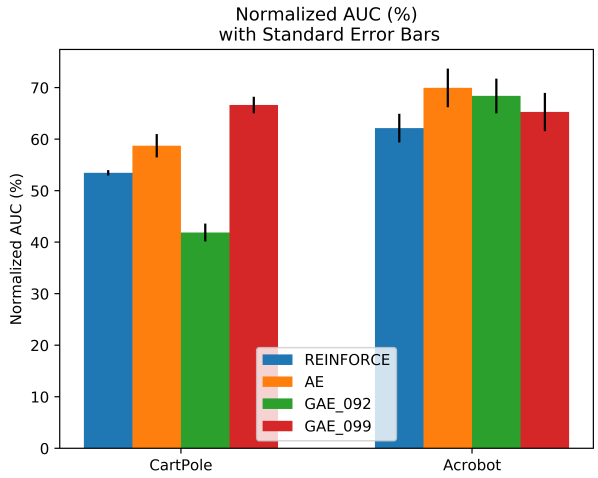

Fig. 3: Bar plots of normalized AUC measured per model and environment with SEM confidence intervals

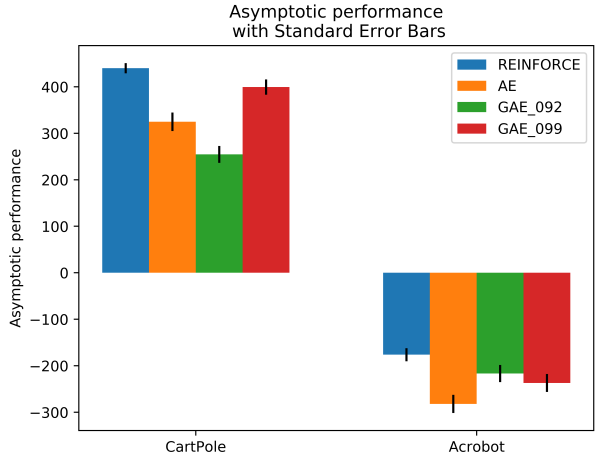

Fig. 4: Bar plots of asymptotic performance measured per model and environment with SEM confidence intervals

fact, AE is expected to have a lower variance with respect to GAE($\lambda = 0.99$), and should hence converge faster. We can look at GAE($\lambda = 0.92$)'s performance to have a better explanation of why this may be the case.

GAE($\lambda = 0.92$) performs differently compared to the other methods in the two different environments. This is most likely due to the different nature of the environments. As explained in Section II-A the goals are different. In CartPole the agent tries to prevent the pole from falling. Therefore, in each timestep it is easy to apply an action that benefits this goal. In fact, it may be that case that AE can exploit this characteristic of the environment, given how it is more biased towards valuing higher short term positive reinforcements (given the 1-step bootstrapping), and hence performs rather well compared to GAE($\lambda = 0.92$). This is, however, not the case in Acrobot, where the agent needs to perform multiple actions to potentially reach its goal, so the benefit of a single move is not clear.

Why GAE($\lambda = 0.92$) performs worse than GAE($\lambda = 0.99$) for the CartPole environment is not obvious. The λ parameter controls the bias-variance trade-off. When $\lambda = 0$ it is close to a 1-step method and very biased and when $\lambda = 1$ it is close to the Monte Carlo method. One could also interpret λ as a non-linear parameter that controls how important subsequent time-steps are. This suggests that GAE($\lambda = 0.92$) should mitigate between the performance of AE and GAE($\lambda = 0.99$). While this seems to be the case in Acrobot, or at least there is no significant difference between the metrics to suggest otherwise, in Cartpole this algorithm has a much greater bias than 1-step bootstrapping. Hence a lower λ, while making GAE similar to having 1-step bootstrapping, has the additional effect of the model fitting a sub-optimal policy. This is supported by the fact that in Figure 1 we can see that, on average, after approximately 50 episodes, the rate of the average episode return of the algorithm notably decreases. What this suggests is that, since the bias GAE($\lambda = 0.92$) is higher than that of AE, if λ is low enough (depending on the environment), the biases of the first few n-step bootstrappings are accumulated, resulting in a worse bias compared to 1-step bootstrapping. This would also explain why such behavior is observed only in CartPole, since the algorithm is immediately presented with actions that sustain a sub-optimal goal. Hence, GAE($\lambda = 0.92$)'s multiple bootstrapping enforce this sub-optimal strategy, never allowing the algorithm to improve or being able to consider (fit) a better one.

Another way that GAE performed differently between the environments, is that the GAE did not always converge in the Acrobot environment. Even after 1000 episodes, the Acrobot did not reach its goal. This never happened in the CartPole environment. It seems that learning with GAE is rather unstable and is not guaranteed to optimize in each step, if the goal needs to be reached instead of to be maintained. A possible solution for this is to implement GAE with a stronger algorithm such as Trust Region Policy Estimation (Schulman et al., 2017). This can help the GAE method to converge more consistently.

V. Conclusions

First we have confirmed that REINFORCE has the highest asymptotic return, as expected by the underlying theory of being unbiased, but with high variance.

Next, we have shown that the performance of the GAEs are partly dependent on the environment. Especially, the difference between maintaining a status quo and trying to reach a goal. We have shown that GAE with either very low (AE) or high λ is preferable in the former case, or at least in the CartPole environment. In the latter case, or Acrobot environment it is more advantageous to pick something in between.

A possible direction for future work could be investigating if the aforementioned trend really holds between the two different environments.

REFERENCES

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2017. Trust Region Policy Optimization. arXiv:1502.05477 [cs.LG]

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

Richard S Sutton. 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*. 1038–1044.

Matthew E. Taylor and Peter Stone. 2009. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research* 10, 56 (2009), 1633–1685. `http://jmlr.org/papers/v10/taylor09a.html`