CS285 HW2 Report - Policy Gradients

CS285 Deep Reinforcement Learning, Decision Making, and Control Fall 2022

Report Author: Kevin (Yu-Teng Li) [Project Spec] [Code Repo]

Abstract

In this report we implement different variations of Policy Gradient (PG) methods in Reinforcement Learning, starting from a vanilla PG with the REINFORCE algorithm, then gradually adding techniques to reduce variance such as Actor-Critic and Generalized Advantage Estimation. We evaluate different methods across multiple OpenAI Gym environments, namely Cart Pole, Lunar Lander, Half Cheetah and Hopper.

Vanilla Policy Gradients

For the first set of experiments, we implemented PG with either *full reward* or *reward-to-go.* We evaluated the two variations in Cart Pole environment, which yields the following results.

- `rtg` means applying reward-to-go
- `dsa` means to *not* standardize advantage function, which is just Q-value estimate since no baseline is applied yet

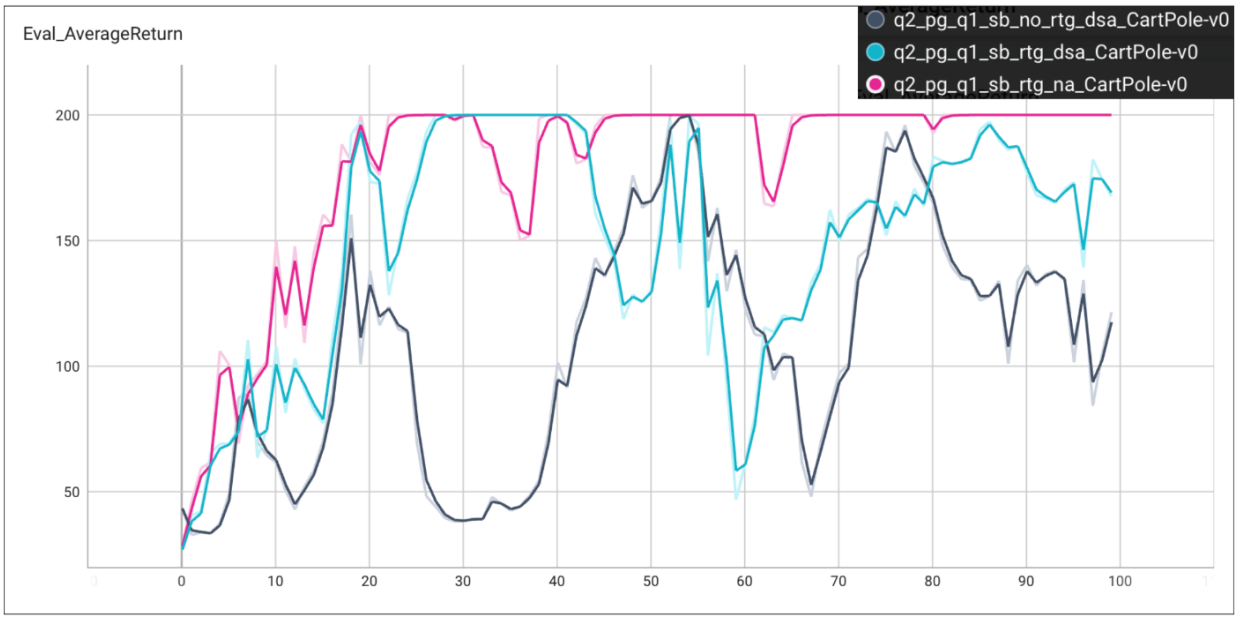

Figure 1 : Runs with batch size = 1000

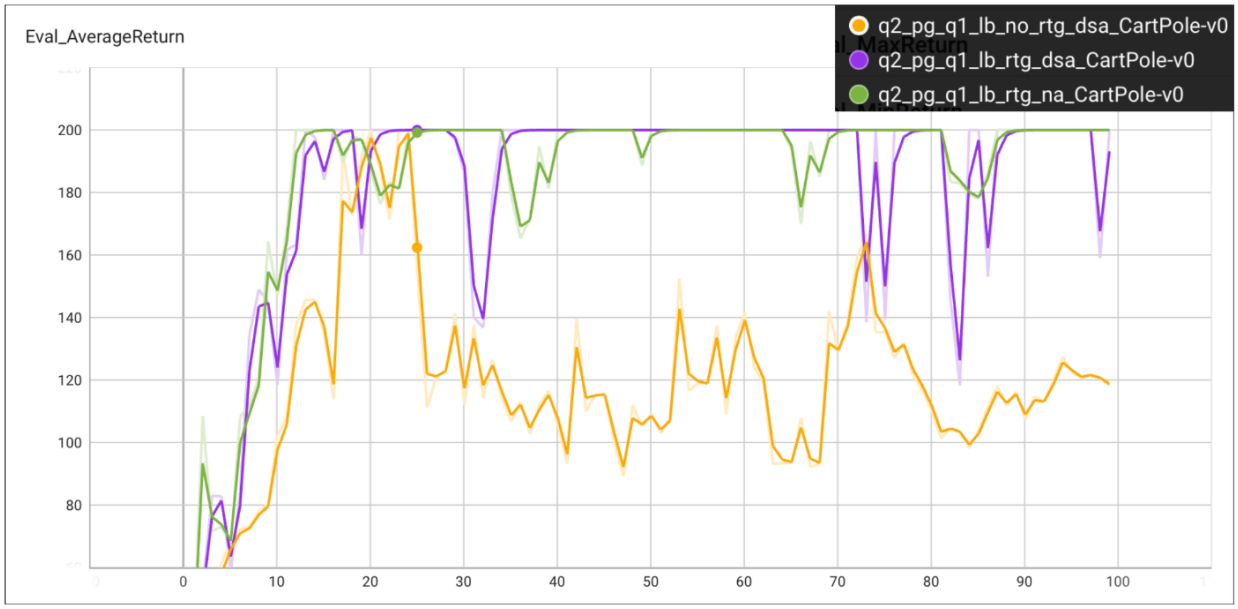

Figure 2 : Runs with batch size = 5000

In both *small* and *large* batch size experiments, we observe several consistent trends. First, large batch size generally yields more stable training results. This phenomenon aligns with the common belief in supervised learning where larger batch size means a better approximation of the overall data distribution, leading to more accurate stochastic gradient descent results. Second, standardizing advantage functions reduces variance of reward return by a substantial amount. One evident example pair of example is `q2_pg_q1_lb_rtg_dsa_CartPole-v0` and `q2_pg_q1_lb_rtg_na_CartPole-v0`, and such a trend is also observed in the small batch size setting. Lastly, using *reward-to-go* indeed reduces the variance in both sets of experiments, which explains why the model that utilizes both **standardized advantage function** and **reward-to-go** outperforms others in the two graphs.

In addition to experiments in Cart Pole, we also tested the vanilla PG method in the Inverted Pendulum environment, using the following parameters (details in repo).

```
python cs285/scripts/run_hw2.py \
    --env_name InvertedPendulum-v4 \
    --ep_len 1000 \      # episode length
    --discount 0.9 \     # reward discount factor
    -n 100 \             # training iterations
    -l 2 \               # layers of MLP
    -s 64 \              # number of neurons per layer
    -b 2000 \            # batch size
    -lr 0.01 \           # learning rate (tested 1e-2 and 5e-3)
    -rtg --exp_name q2_b2000_lr001_d090
```

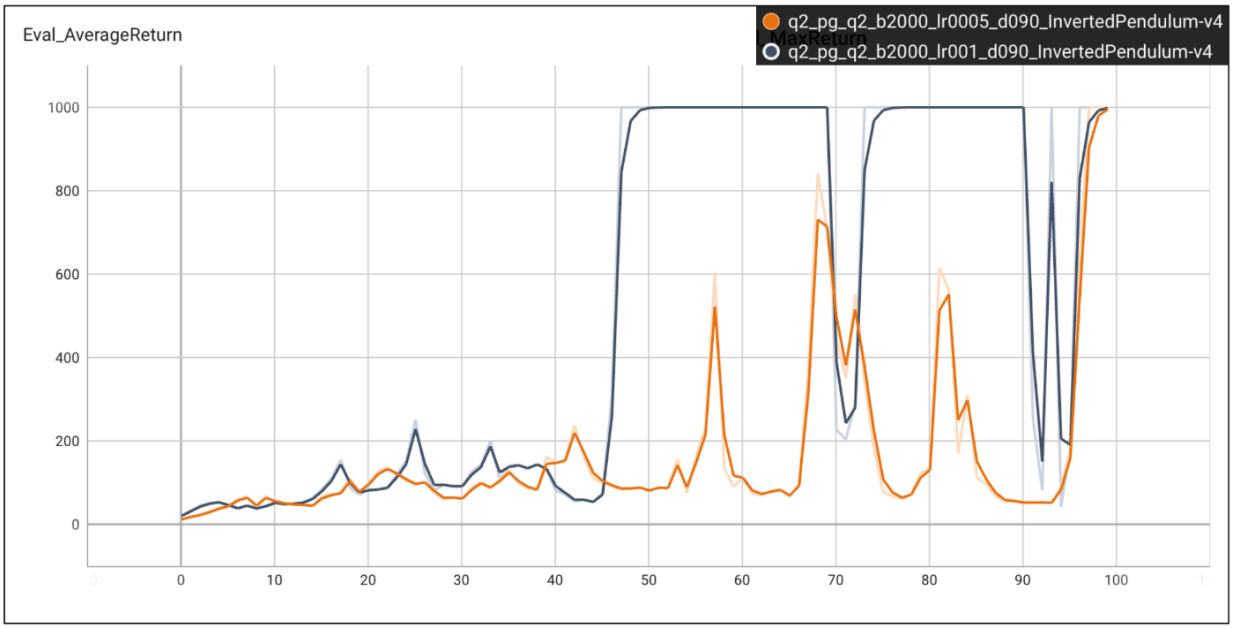

Figure 3 : Inverted Pendulum experiments

Actor-Critic / Neural Network Baselines

Reward-to-go improves the high variance problem in RL training by exploiting the causality assumption (i.e. the present cannot affect the past). To further improve the model, researchers have tried different **baselines** for advantage functions such that model's gradients emphasizes not just samples with *positive rewards* but samples with *above average rewards*. Such a mechanism can be naively implemented using the average trajectory reward for each sample, or better, using a trained Value function baseline predictor, which is what we implement in this experiment. This policy gradient method that uses a learned baseline is as known as **Actor-Critic** method. The equation goes as follows:

$$\left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r(s_{it'}, a_{it'})\right) - V_\phi^\pi(s_{it}) \qquad y_{i,t} \approx \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \qquad \mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

Advantage with Learnt baseline *Learn the Value function as baseline by supervised regression*

To evaluate the effectiveness of our implemented neural network baseline, we test our model in the Lunar Lander environment for 100 training iterations, with batch size of 40k. While we did not have time to test the agent in Lunar Lander without learnt baseline, the learning curve below shows a more stably increasing overall trend compared to graphs in the previous section.

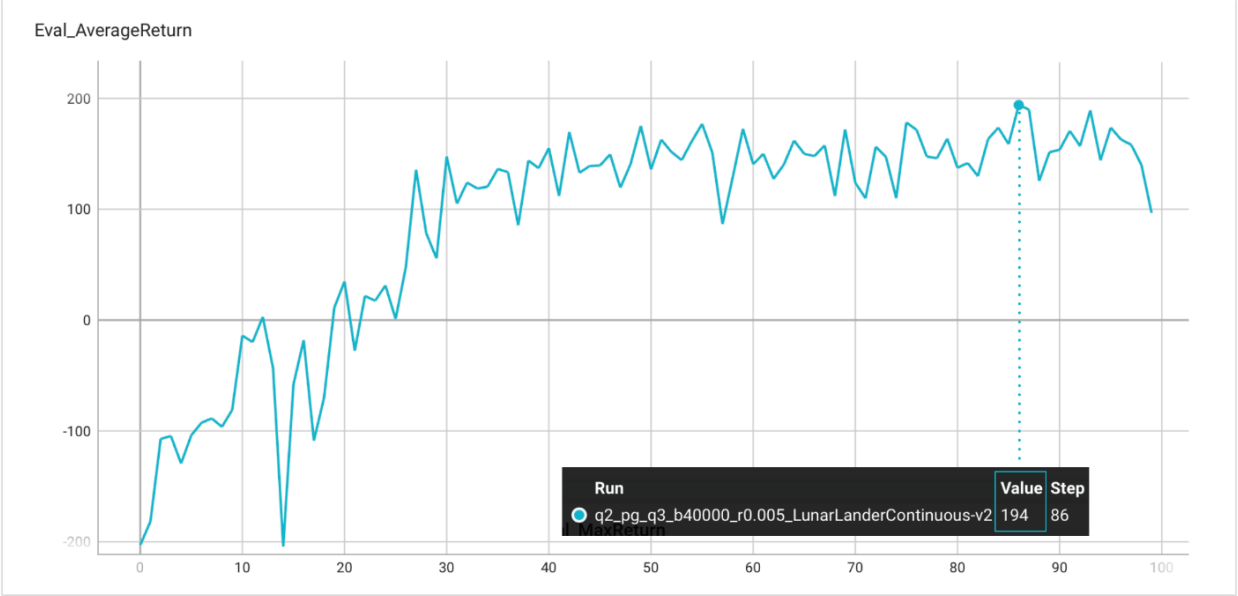

Figure 4 : Lunar Lander experiment with Actor-Critic agent

We then study the effect of batch sizes and learning rates to our policy gradient agent with learnt baselines in the Half Cheetah environment. First, with reward-to-go and learnt baseline applied, we tested 9 variations of **batch size** $b \in$ [10000, 30000, 50000] and **learning rates** $r \in$ [0.005, 0.01, 0.02].

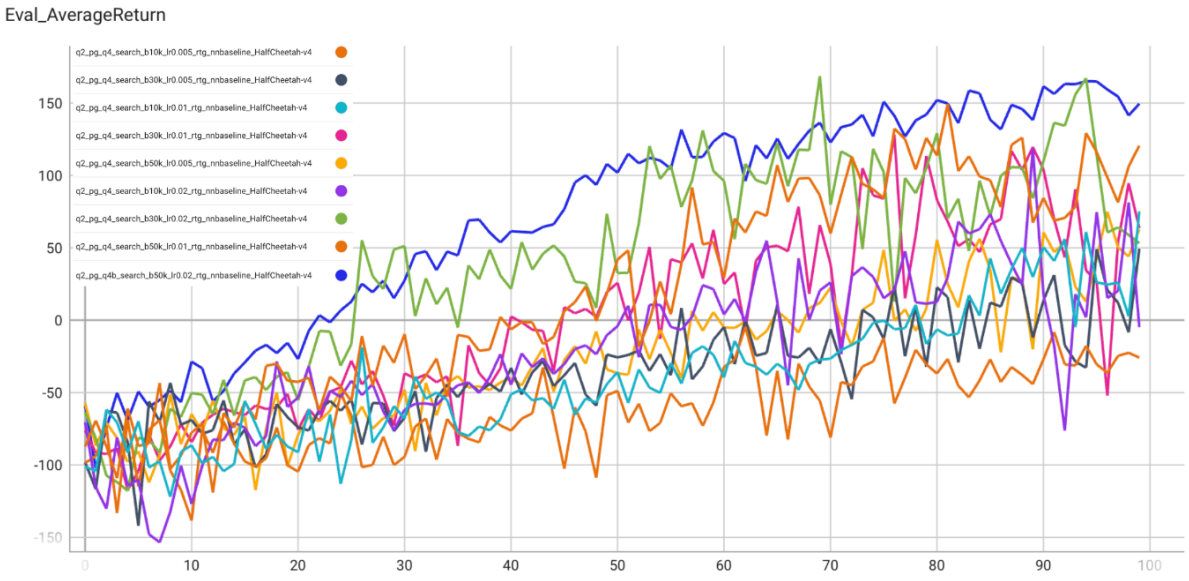

Figure 5 : Experiments on Half Cheetah in search of optimal batch size and learning rate

In the graph above, we observed that the experiment run with batch size = 50k, learning rate of 0.02, reward-to-go and learned baselines, performs the best. We therefore proceed with this set of hyperparameters to understand the effect of reward-to-go and learned baselines in the context of Half Cheetah environment.

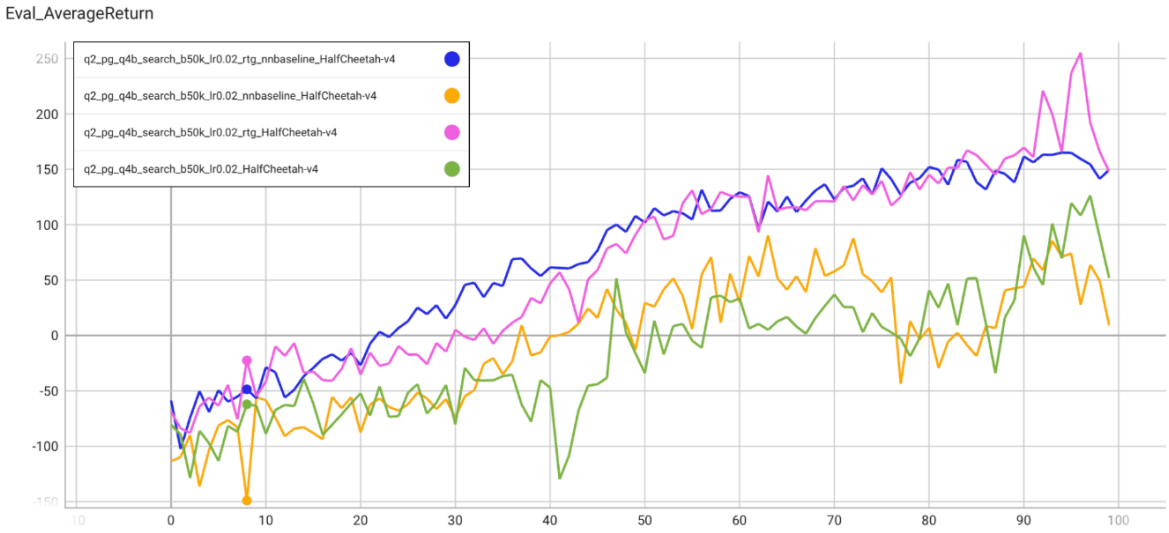

Figure 6 : Ablation study for reward-to-go and learnt baselines in Half Cheetah environment

The experiment runs in *Fig. 6* shows two types of performances — both methods that applied reward-to-go (RTG) were able to attain better results than ones with full trajectory rewards. Interestingly, when comparing two models that have RTG applied, the run with neural network baselines converges earlier, but the one without baseline actually obtained better performance at the end. This could happen due to the stochasticity of policy gradient experiments and perhaps after more runs with different random seeds, the agent equipped with both RTG and learnt baselines will prevail.

Generalized Advantage Estimation

One caveat with using a neural network for baselines is that the baseline predictions at the beginning of the training procedure would be close to random, which could lead to suboptimal policy updates due to high bias baseline predictions. On the other hand, Monte Carlo samples of rewards are always an unbiased but high variance estimation of the advantage function. The aforementioned phenomenon leads to the invention of **n-step return** of advantage function computation, which controls the bias-variance tradeoff by adjusting "how many steps of Monte Carlo samples' rewards are considered". **Generalized Advantage Estimation (GAE)** is an approximation that considers all possible n-step return computations with an exponential average between them (Schulman et al). In this project, we experimented with different weighing factor $\lambda \in [\ 0, 0.95, 0.98, 0.99, 1.0\ ]$ to search for the optimal GAE. Experiments conducted in the Hooper environment.

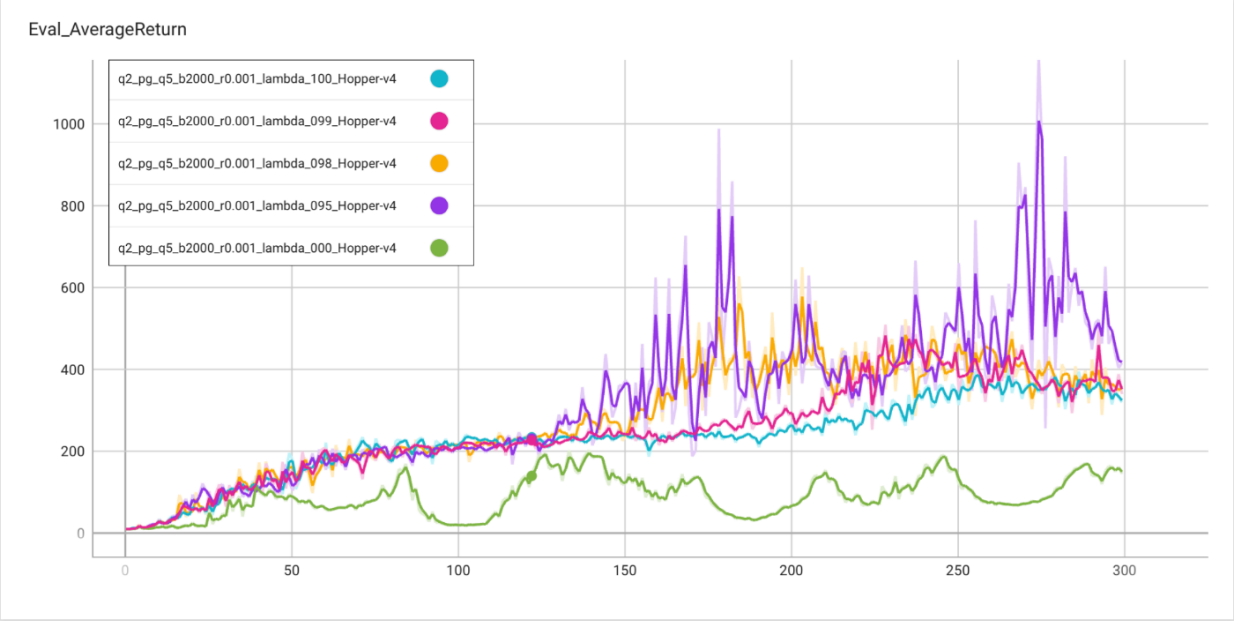

Figure 7 : Ablation study for λ factor in GAE

From the above graph, we find that $\lambda = 0.95$ is the optimal GAE. Specifically, in the interval of $\lambda = (0.95, 1.0]$, we observe that the higher λ is, the lower model performance is. This roughly indicates that GAE does provide a better estimate of advantage function since setting $\lambda = 1.0$ basically yields the vanilla neural network baseline estimator, which is implemented in the "Actor-Critic" section.

Conclusion

In this project, we studied three major variations of policy gradient methods — vanilla implementation, Actor-Critic (AC), and Generalized Advantage Estimation (GAE). While the conducted experiment is not sufficiently comprehensive, we indeed observe the effectiveness of policy gradients compared to imitation learning in general, and how AC and GAE are able to improve the high-variance problem in policy gradients.