Deep Reinforcement Learning Assignment 2
Policy Gradients

Master IASD - Théophane Vallaeys, Sihan Xie

February 9, 2023

1 CartPole

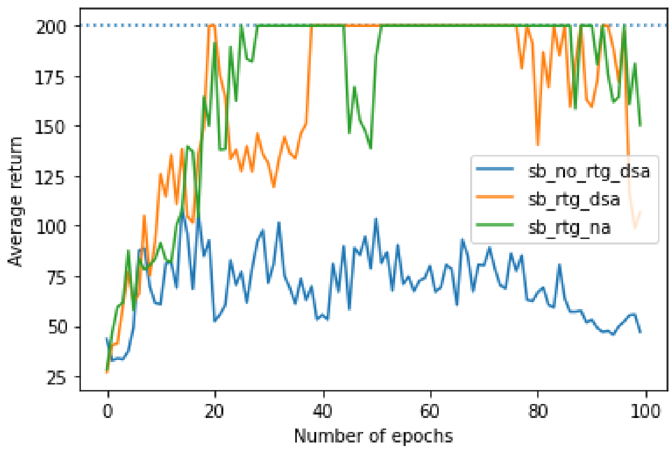

Figure 1: CartPole experiments with a small batch size=1000

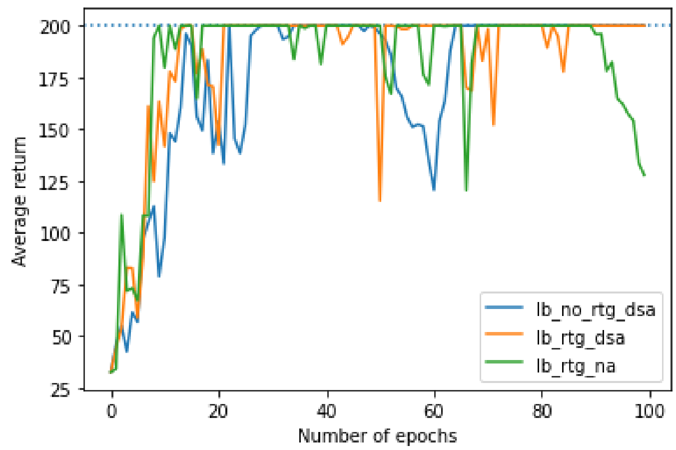

Figure 2: CartPole experiments with a large batch size=5000

Without advantage-standardization (when using the `-dsa` flag), the reward-to-go estimator has much better results than the one without it: in Figure 1, the former oscillates near a return of 200, while the later doesn't even get better than 100. We also notice that batch size has an impact on learning behavior, a larger batch size resulting in faster convergence in the first iterations and less variance in the final iterations.

The command line used for small batch size experiment :

```
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 -dsa --exp_name q1_sb_no_rtg_dsa
```

```
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 -dsa -rtg --exp_name
    q1_sb_rtg_dsa
```

```
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 1000 -rtg --exp_name q1_sb_rtg_na
```

For large batch size experiment :

```
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 5000 -dsa --exp_name q1_lb_no_rtg_dsa
```

```
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 5000 -dsa -rtg --exp_name
    q1_lb_rtg_dsa
```

```
python cs285/scripts/run_hw2.py --env_name CartPole-v0 -n 100 -b 5000 -rtg --exp_name q1_lb_rtg_na
```

Based on our plots, we can't conclude that advantage-standardization improved nor hindered the performances, as all the curves are pretty unstable, and there is no clear difference between `sb_rtg_dsa` and `sb_rtg_na` or `lb_rtg_dsa` and `lb_rtg_na`. So we ran another experiment without any flags (`sb_no_rtg_na`) to compare it with the first experiment, which disables advantage-standardization. The results are shown below (Figure 3). In this experiment, while `sb_no_rtg_na` reaches 200 more quickly, it then worsen and goes down towards small return values, while `sb_no_rtg_dsa` is more stable and stays at high values. So **the advantage-standardization doesn't seem to help at all**.

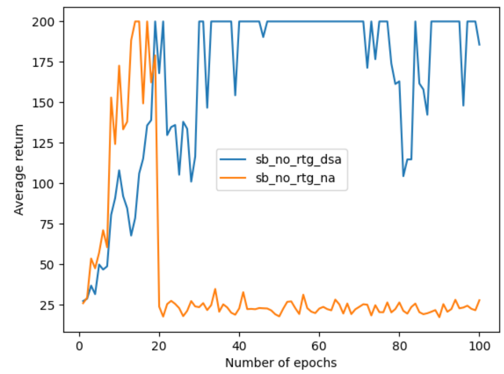

Figure 3: CartPole experiments with and without advantage-standardization

2 InvertedPendulum

To find the best possible values for `b*` and `r*`, we used the hyperparameter optimization library **optuna**, which by default uses a Tree-structured Parzen Estimator, which can be seen a a lightweight Bayesian Search algorithm. We defined a search space over the batch size and learning rate, with a logarithmic scaling for both, and the following reward function:

$$R + \ln(r) - \lambda \ln(b) \tag{1}$$

with R the final average evaluation return, b the batch size, r the learning rate, and λ a parameter used to control the tradeoff. With different values of λ, we obtained using the script `hp_optim/hp_optim.py` the following results:

λ	return value R	batch size b	learning rate r
0.5	1000	2759	0.09955244631066584
1	1000	1097	0.03283988437943506
2.5	1000	790	0.029399377942098733

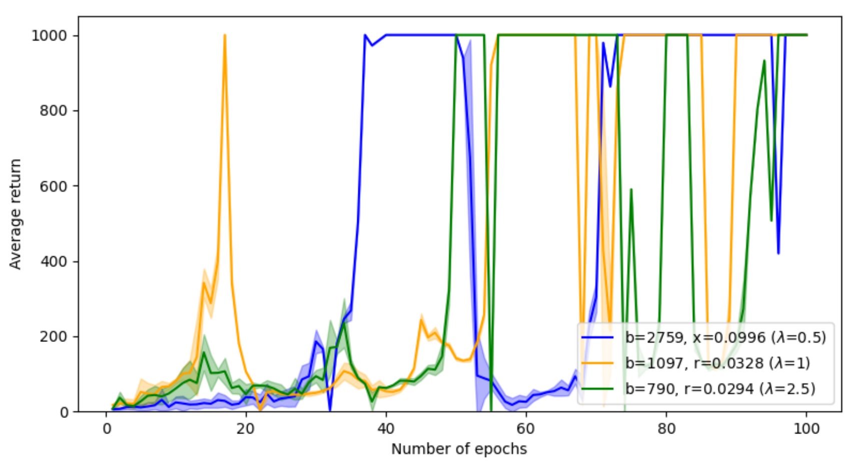

Figure 4: InvertedPendulum experiments reaching the maximum return value quickly, but with very unstable curves

The command lines used are as follows :

```
# lambda=0.5
python run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 2759 -lr 0.09955244631066584 -rtg \
    --exp_name q2_l0.5

# lambda=1
python run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 1097 -lr 0.03283988437943506 -rtg \
    --exp_name q2_l1

# lambda=2.5
python run_hw2.py --env_name InvertedPendulum-v4 \
    --ep_len 1000 --discount 0.9 -n 100 -l 2 -s 64 -b 790 -lr 0.029399377942098733 -rtg \
    --exp_name q2_l2.5
```

3 LunarLander

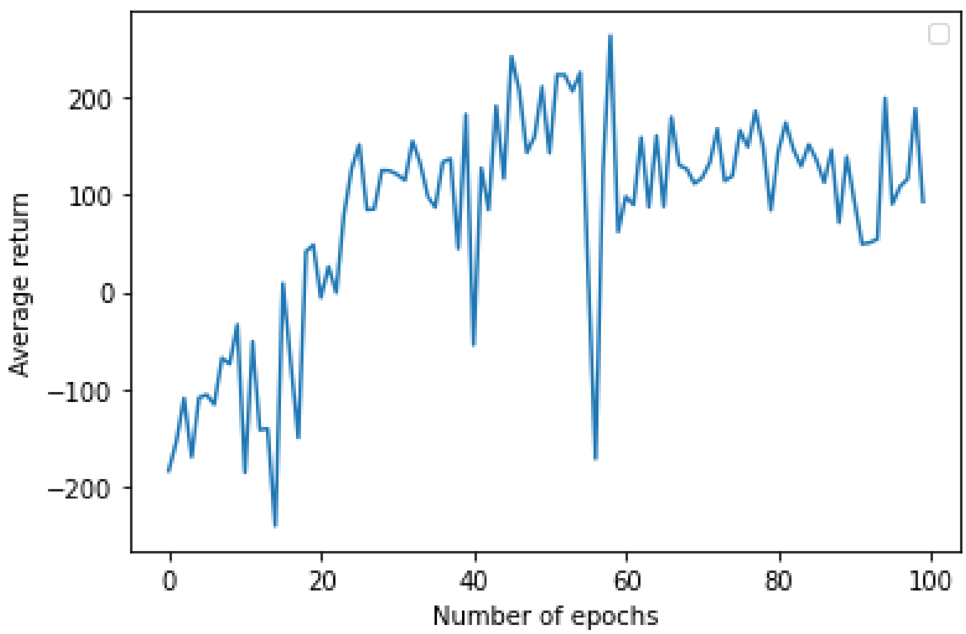

Figure 5: LunarLander experiment with reward-to-go estimator and baseline subtraction

The command line used is :

```
python cs285/scripts/run_hw2.py \
--env_name LunarLanderContinuous-v2 --ep_len 1000 \
--discount 0.99 -n 100 -l 2 -s 64 -b 40000 -lr 0.005 \
--reward_to_go --nn_baseline --exp_name q3_b40000_r0.005
```

4 HalfCheetah

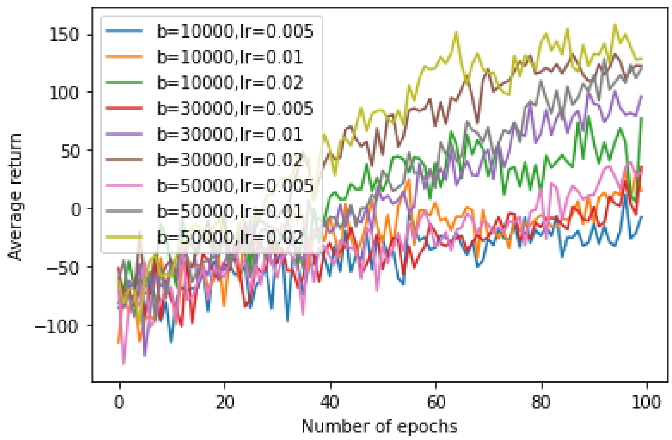

Figure 6: HalfCheetah experiments with different batch size and learning rate

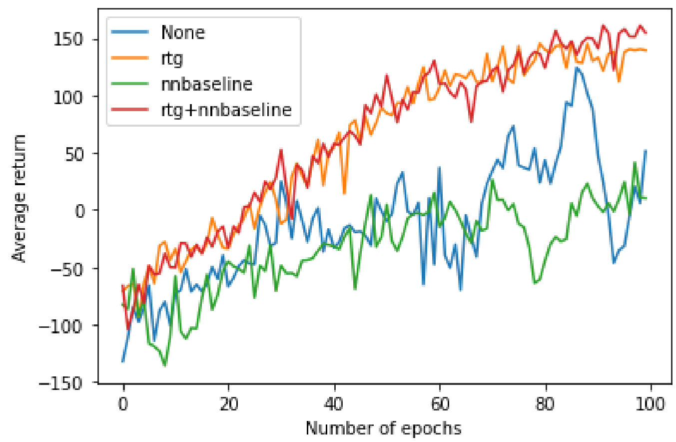

Figure 7: HalfCheetah experiments with/without reward-to-go and baseline subtraction when using batch_size=50000 and learning_rate=0.02

Figure 6 shows the use of grid search to find the best combination of batch size and learning rate for the HalfCheetah task. The b^* we found is 50000 and r^* is 0.02. The model learns more quickly when using a large learning rate. A large batch size also helps the model learn faster and reach better target values within the same number of iterations.

5 Hopper

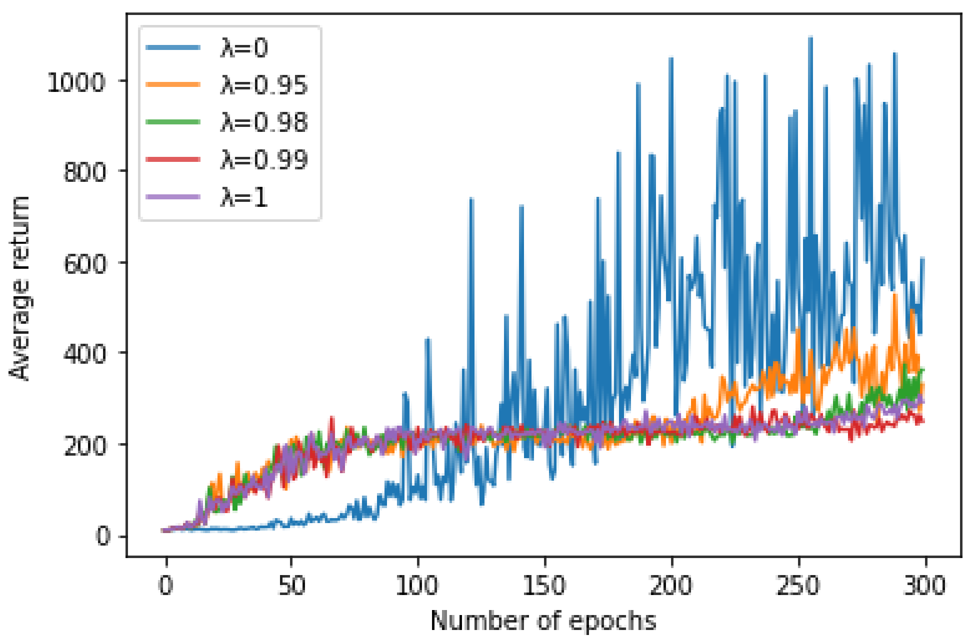

Figure 8: Evolution of the average evaluation return for different values of λ

The generalized estimator of the advantage function allows a trade-off of bias vs variance using the parameter $\lambda \in [0, 1]$. We see that with $\lambda = 0$, the average returns reaches the highest values, but the training is too unstable. With higher values of lambda, the training is more stable. As we increase λ towards 1, we reduce the variance of our estimator but increase the bias. $\lambda = 0.95$ gives us a good trade-off between variance and bias.

We also tested some other λ values and we observed that the performance of the estimator may also degrade after reaching the highest value. We plotted a curve with more lambda values below, confirming this result.

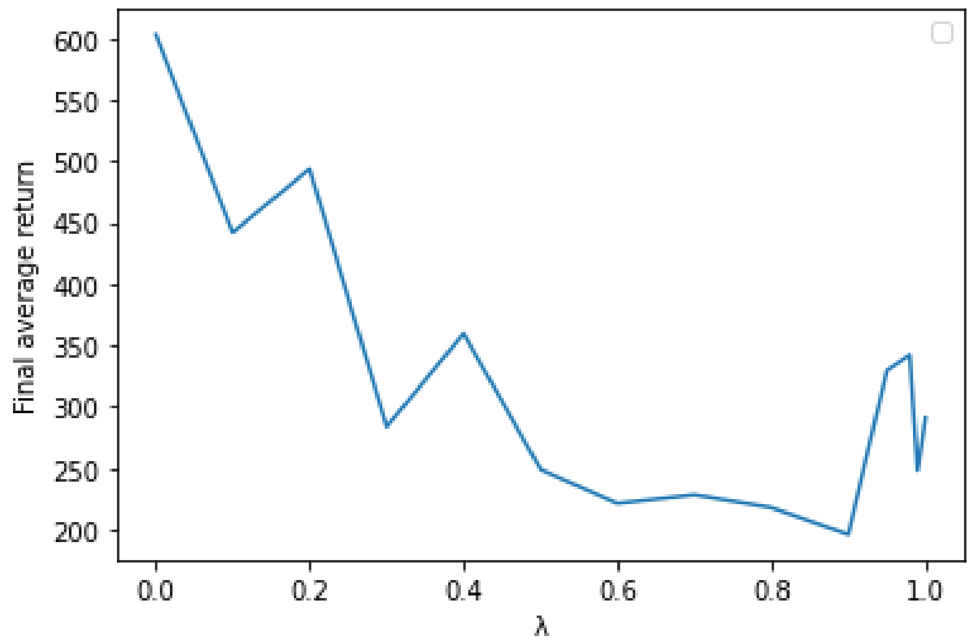

Figure 9: The final average evaluation return for different values of λ