Deep RL Assignment2

Mufei Li mufei.li@nyu.edu

June 13, 2018

4 Implement Policy Gradient

4.2

1

As required by instructions, we perform 6 experiments on OpenAI's Gym environment CartPole-v0 to get a feel for how different settings impact the performance of policy gradient methods and plot below their learning curves (average return at each iteration) averaged over 5 runs using different random seeds. The setting of each experiment can be specified by its name:

- **sb**: small batch size, we use 1000
- **lb**: large batch size, we use 5000
- **rtg**: use reward to go
- **no_rtg**: do not use reward to go
- **na**: normalize advantages
- **dna**: do not normalize advantages

Besides the differences we mention above, all experiments use the same following setting:

- **discount factor**: 1.0 (no discount)
- **number of random seeds**: 5, particularly we use $\{1, 11, 21, 31, 41\}$
- **number of iterations for each random seed**: 100
- **max path length**: 200, which is enforced by OpenAI's design of max number of timesteps allowed in an episode for CartPole-v0
- **learning rate**: 0.005
- **optimization**: Adam with its default settings in PyTorch
- **policy architecture**: in the order of nn.Linear(4, 32), nn.Tanh, nn.Linear(32, 2), followed by a softmax operation

To run the experiments, use the following command line configurations:

```
python train_pg.py CartPole-v0 -n 100 -b 1000 -e 5 -dna -l 1 -s 32
    --exp_name sb_no_rtg_dna
python train_pg.py CartPole-v0 -n 100 -b 1000 -e 5 -rtg -dna -l 1 -s 32
    --exp_name sb_rtg_dna
python train_pg.py CartPole-v0 -n 100 -b 1000 -e 5 -rtg -l 1 -s 32
    --exp_name sb_rtg_na
python train_pg.py CartPole-v0 -n 100 -b 5000 -e 5 -dna -l 1 -s 32
    --exp_name lb_no_rtg_dna
python train_pg.py CartPole-v0 -n 100 -b 5000 -e 5 -rtg -dna -l 1 -s 32
    --exp_name lb_rtg_dna
python train_pg.py CartPole-v0 -n 100 -b 5000 -e 5 -rtg -l 1 -s 32
    --exp_name lb_rtg_na
```

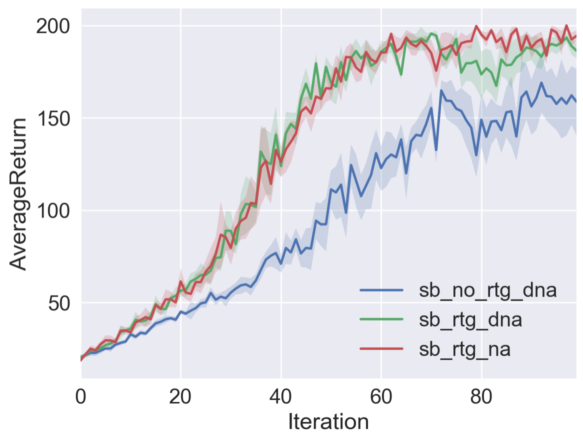

Figure 1: learning curves for experiments using small batch size

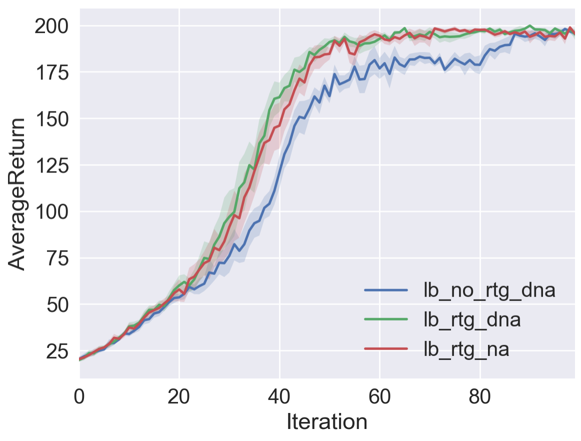

Figure 2: learning curves for experiments using large batch size

Based on the experiment results, we have the following observations:

1. Without advantage-centering (experiments with **dna**), the one using reward to go is better than the trajectory-centric one, for both small batch size and large batch size.
2. We only make advantage centering available when we use reward to go. From the learning curves, it's hard to say advantage centering helps.
3. By using reward to go, we expect to get a better performance from the theory as this does not change the expectation of policy gradient but reduces the variance of the gradient estimator. This matches our empirical results. Meanwhile, theoretically to perform advantage normalization should lower the variance of the gradient estimator and help boost the empirical performance, this is not observed in our empirical results. One possible explanation is that CartPole-v0 is an easy task with all the immediate rewards being constantly 1 before the end of the task. For an environment with more complex reward function, the story may be different.
4. From figure 3 to 5 we can tell that a larger batch size in general help, but the improvement is limited after we use reward to go.

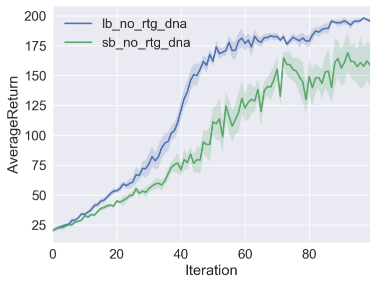

Figure 3: learning curves for experiments using small batch size

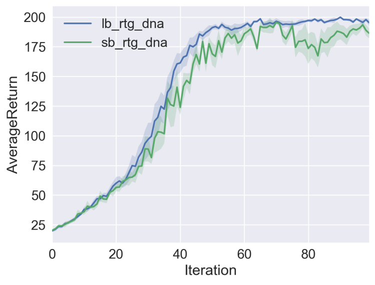

Figure 4: learning curves for experiments using large batch size

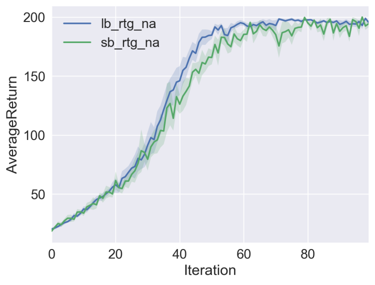

Figure 5: learning curves for experiments using large batch size

2

Using the following hyperparameters:

- **batch size**: 2000
- **use reward to go**
- **use advantage normalization**
- **discount factor**: 1.0 (no discount)
- **number of random seeds**: 1
- **random seed**: 1

- **number of iterations**: 100
- **max path length**: 1000, which is enforced by OpenAI's design of max number of timesteps allowed in an episode for InvertedPendulum-v1
- **learning rate**: 0.005
- **optimization**: Adam with its default setting in PyTorch
- **policy architecture**: see figure 6

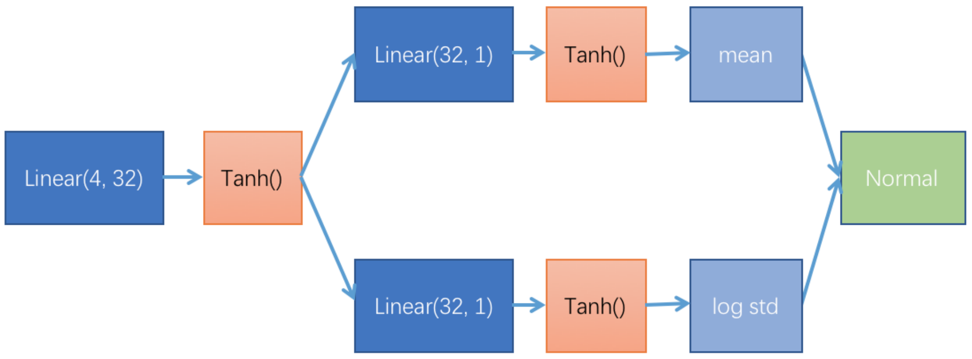

Figure 6: model architecture for policy network in 4.2.2

by running the following command line configuration [1],

```
python train_pg.py InvertedPendulum-v1 -n 100 -b 2000 -l 1 -s 32 -e 1 -rtg
    -lr 5e-3 -ts -tm --exp_name b2000_l1_s32
```

we obtain the learning curve in figure 7.

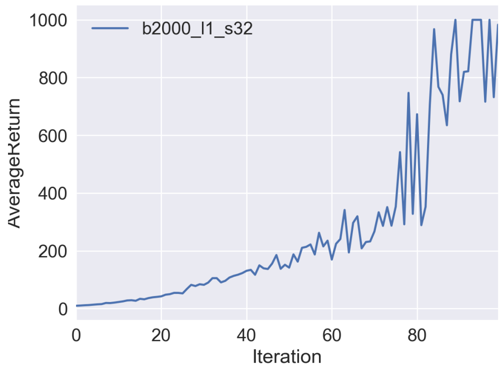

Figure 7: learning curve for the experiment requires in 4.2.2

5 Implement Neural Network Baselines

For the objective of the baseline, we simply use mean squared error with the target being the reward to go for the on-policy experience, i.e.

$$\frac{1}{\sum_{i=1}^{N} T_i} \sum_{i=1}^{N} \sum_{t=1}^{T_i} \left\| \hat{V}_{\phi}^{\pi}(s_{i,t}) - \sum_{t'=t}^{T_i} r(s_{i,t}, a_{i,t}) \right\|^2, \tag{1}$$

where N is the number of trajectories, $s_{i,t}, a_{i,t}$ is the tth state, action in trajectory i and T_i is the length of the ith trajectory.

[1]-tm is used to indicate whether we want to apply a tanh activation function for the mean of the Normal distribution and -ts is used to indicate whether we want to apply a tanh activation function for $\log \text{std}$ of the Normal distribution.

We perform two experiments to compare the learning curve with both the neural network baseline function and advantage normalization to that without the neural network baseline function but with advantage normalization. We use the same hyperparameters as in the previous section except that the learning curve is now averaged over 10 random seeds $(1, 11, 21, 31, 41, 51, 61, 71, 81, 91)$.

The difference between using baseline or not is so small that we cannot conclude here whether the use of baseline helps.

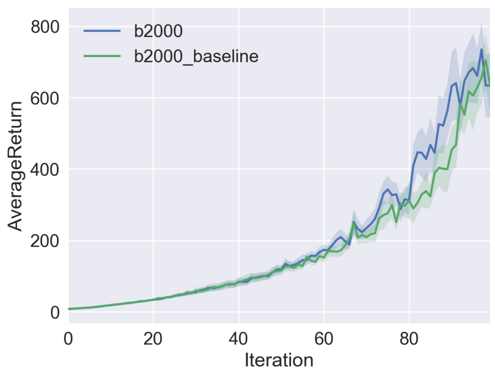

Figure 8: b2000 refers to the learning curve without baseline while b2000_baseline refers to the one with baseline

It might be interesting to explore whether to set the target to be n-step bootstrapping, i.e.,

$$\sum_{\tau=1}^{n} r(s_{i,t+\tau-1}, a_{i,t+\tau-1}) + \hat{V}_{\phi}^{\pi}(s_{i,t+n}) \text{ for state } s_{i,t}, \tag{2}$$

in mean squared error would make a significant difference.

6 HalfCheetah

We use the following hyperparameters and settings:

- **batch size**: 50000
- **use reward to go**
- **use advantage normalization**
- **use baseline**
- **baseline objective**: same as the previous section
- **discount factor**: 0.9
- **number of random seeds**: 1
- **random seed**: 1
- **number of iterations**: 100
- **max path length**: 150
- **learning rate**: 0.025
- **optimization**: Adam with its default setting in PyTorch
- **policy architecture**: see figure 9 [2]
- **critic (state value function estimator) architecture**:see figure 10

[2]Strictly speaking, we should use a multivariate normal distribution rather than a normal distribution which assumes that the covariance between any two distinct actions is 0. However in practice this often make little difference.

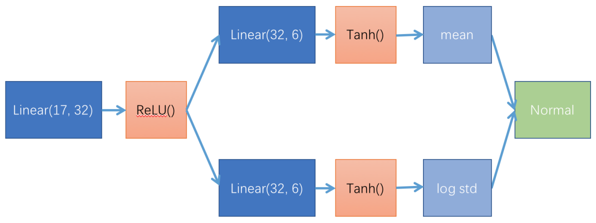

Figure 9: policy network

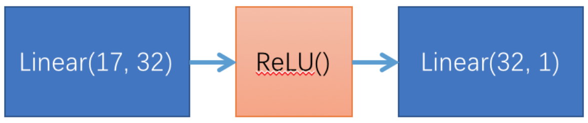

Figure 10: value network

, which corresponding to the command line configuration:

```
python train_pg.py HalfCheetah-v1 -ep 150 --discount 0.9 -n 100 -b 50000 -l 1
        -s 32 -e 1 -rtg -lr 0.025 -ts -tm -bl -ia relu
        --exp_name b50000_l1_s32_e1_rtg_ts_tm_bl_relu
```

We obtain the learning curve in figure 11 after performing the experiment.

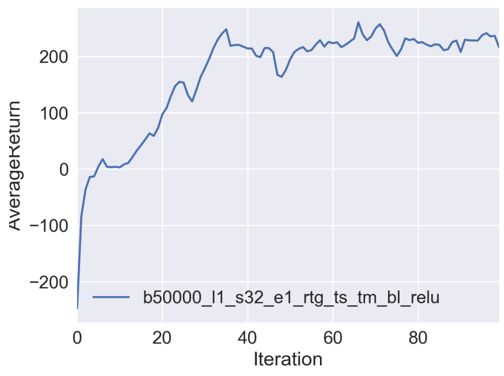

Figure 11: The average return fluctuates around 220.