PS 2

Chris Gagne

September 20th 2017

4.1: CartPole Experiments

Methods

I ran the following experiments (with the default learning rate of 0.001):

```
python train_pg.py CartPole-v0 -n 100 -b 1000 -e 5 -dna \
--exp_name sb_no_rtg_dna -l 1 -s 32 --seed 101

python train_pg.py CartPole-v0 -n 100 -b 1000 -e 5 -rtg -dna \
--exp_name sb_rtg_dna -l 1 -s 32 --seed 101

python train_pg.py CartPole-v0 -n 100 -b 1000 -e 5 -rtg \
--exp_name sb_rtg_na -l 1 -s 32 --seed 101

python plot.py \
data/sb_no_rtg_dna_CartPole-v0_20-09-2017_10-14-40 \
data/sb_rtg_dna_CartPole-v0_20-09-2017_10-18-19 \
data/sb_rtg_na_CartPole-v0_20-09-2017_10-26-17

python train_pg.py CartPole-v0 -n 100 -b 5000 -e 5 -dna \
--exp_name lb_no_rtg_dna -l 1 -s 32 --seed 101

python train_pg.py CartPole-v0 -n 100 -b 5000 -e 5 -rtg -dna \
--exp_name lb_rtg_dna -l 1 -s 32 --seed 101

python plot.py \
data/lb_no_rtg_dna_CartPole-v0_20-09-2017_10-36-39 \
data/lb_rtg_dna_CartPole-v0_20-09-2017_10-49-43
```

Before running these experiments, I generated a bunch of observations from each environment by randomly choosing actions. I calculated the mean and std for observations. I saved this out and used it to normalize observations for each environment. I do this after observing each action in the gym. I also experimented with different network sizes, but realized that 1 hidden layer and 32 units was sufficient.

Results

- The reward to go estimator seemed to have helped at least for this random seed. Using it achieved asymptotic performance quicker, and may have reduced the variance of the gradient a bit, judging by the variance in the average return. In theory, I can't imagine a reason why not having the reward to go would be better. (This was pointed out in lecture). See Figure 1.

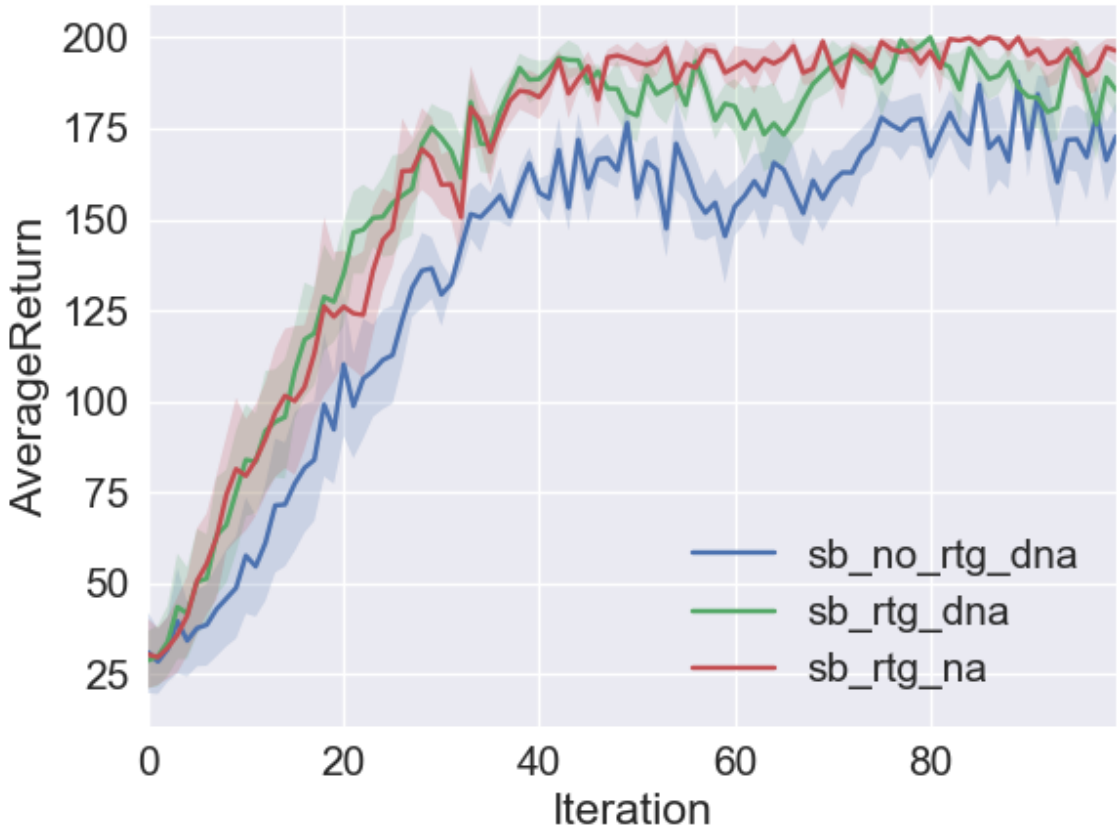

Figure 1: CartPole (batch size 1000)

- Advantage centering did not seem to help in this case. Intuitively, advantage centering rescales the returns so that they are relative to one an another. These act as weights applied to the log likelihood to emphasize (state,action) pairs that were particularly good in the long run. This should help in cases where there is a wide range of possible returns. Since CartPole is such a short game however perhaps the relative importance of returns with and without advantage centering is similar enough. See Figure 1.
- Batch size seems reduces the variance of the policy gradient, as judged by variance in average return. It also seems to have sped up the rate at which the algorithm achieves assymptomic performance, which is not surprising since it uses more data at each step. See Figure 2.

4.2: Inverted Pendulum

Methods

For the inverted pendulum environment, I used a larger network (2 hidden layers with 64 units each) and a faster learning rate (0.01).

```
python train_pg.py InvertedPendulum-v1 -n 100 -b 4000 -rtg \
--exp_name mb_no_rtg_dna --learning_rate=0.01 --seed 101 --n_layers 2 --size 64
```

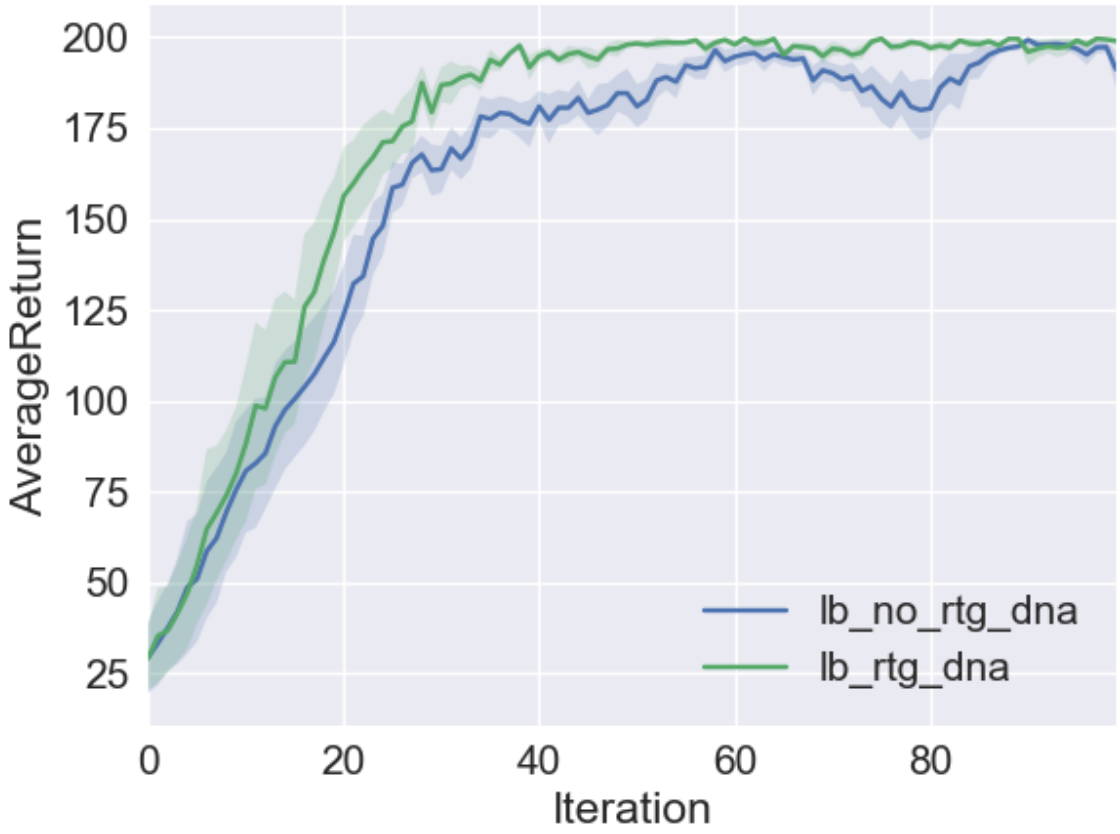

Figure 2: CartPole (batch size 5000)

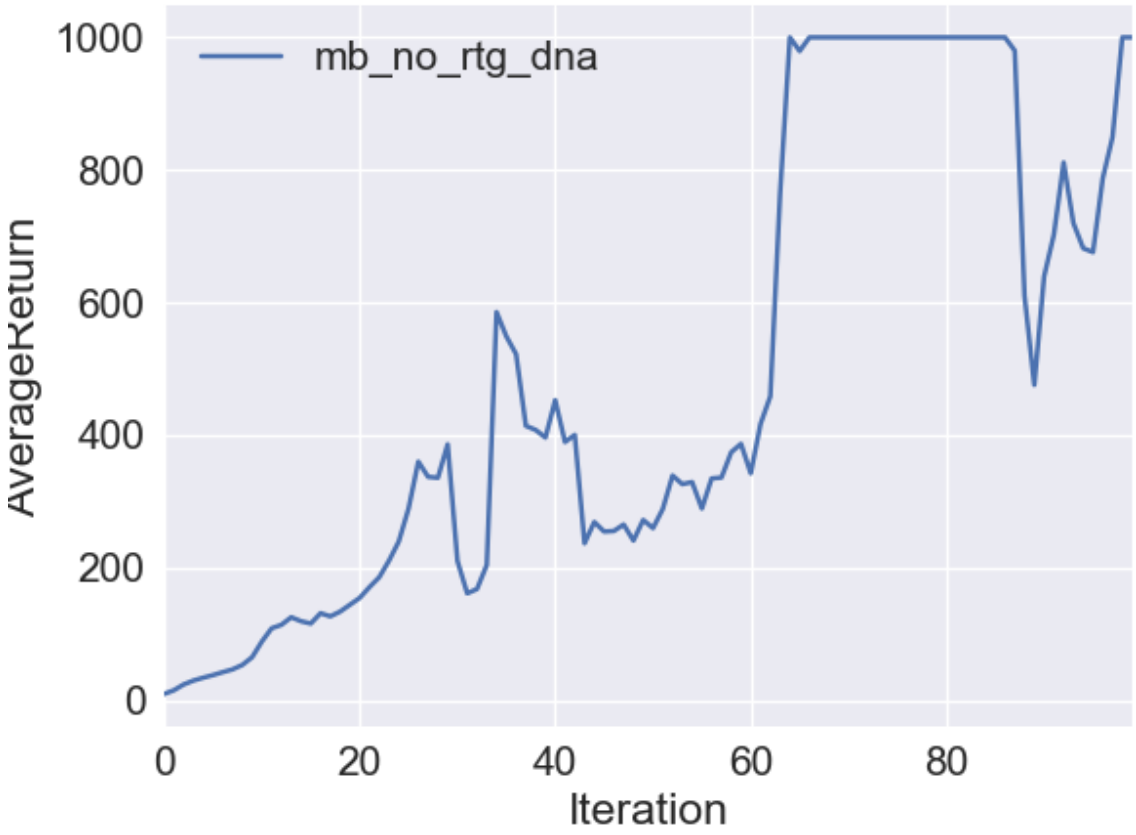

Figure 3: Inverted Pendulum

```
python plot.py data/mb_no_rtg_dna_InvertedPendulum-v1_19-09-2017_22-19-57
```

Results

I was able to achieve maximum performance (1000) in under 100 iterations. However, after achieving maximum performance, it still has a pretty high variance in returns. Perhaps this is due to my relatively high learning rate. See figure 3.

5.: Inverted Pendulum with Neural Net Baselines

Methods

I used the same parameters as before for the Inverted Pendulum. I compared with and without using a neural network to predict the baseline.

```
python train_pg.py InvertedPendulum-v1 -n 100 -b 4000 -rtg \
--exp_name mb_no_rtg_dna_baseline --learning_rate=0.01 \
```

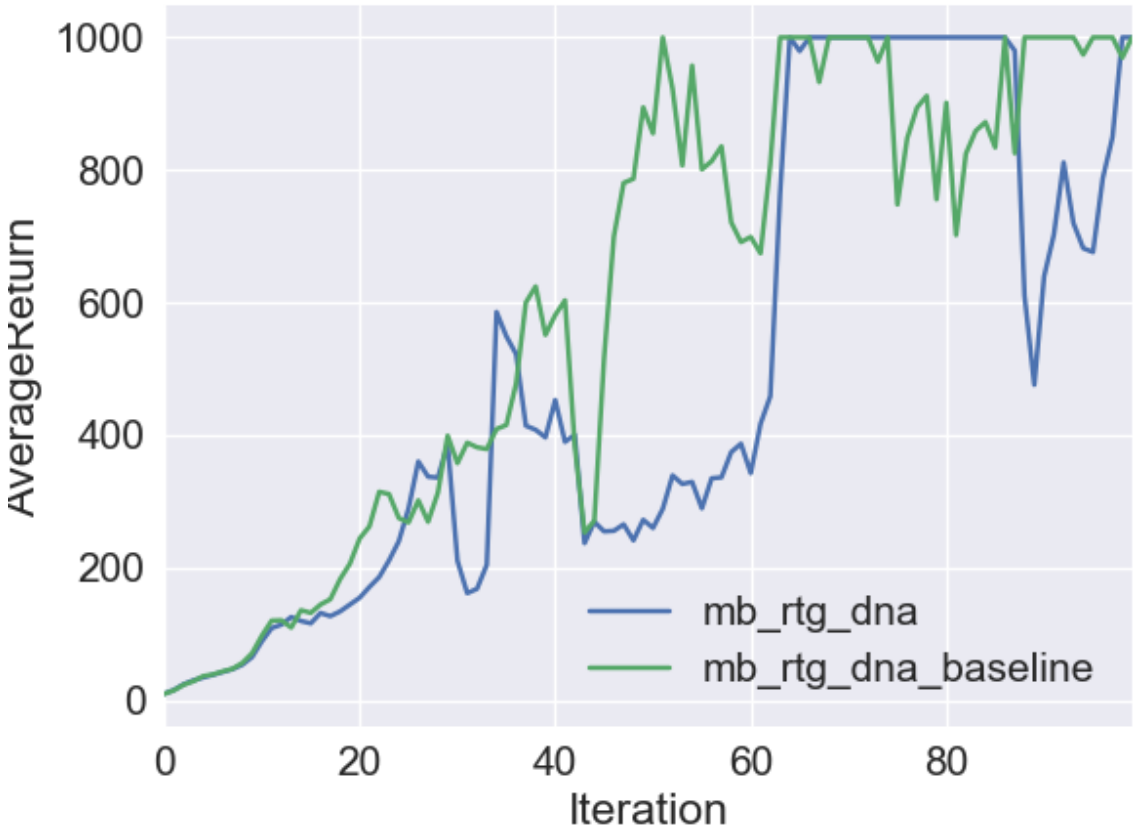

Figure 4: Inverted Pendulum

```
--seed 101 --n_layers 2 --size 64 --nn_baseline

python plot.py \
data/mb_rtg_dna_InvertedPendulum-v1_19-09-2017_22-19-57 \
data/mb_rtg_dna_baseline_InvertedPendulum-v1_19-09-2017_22-33-05
```

Results

With this random seed and single run, using baseline's actually improved the rate of convergence and the assymptomtic variance. If I had more time today, I would have run several experiments to see if this result holds across multiple random seeds. See figure 4.

6.: Half Cheetah

Methods

I started with a small batch size and small network and then increased the size of each. Increasing batch size and the network genarally increased the average reward by iteration 100. Increasing the learning rate

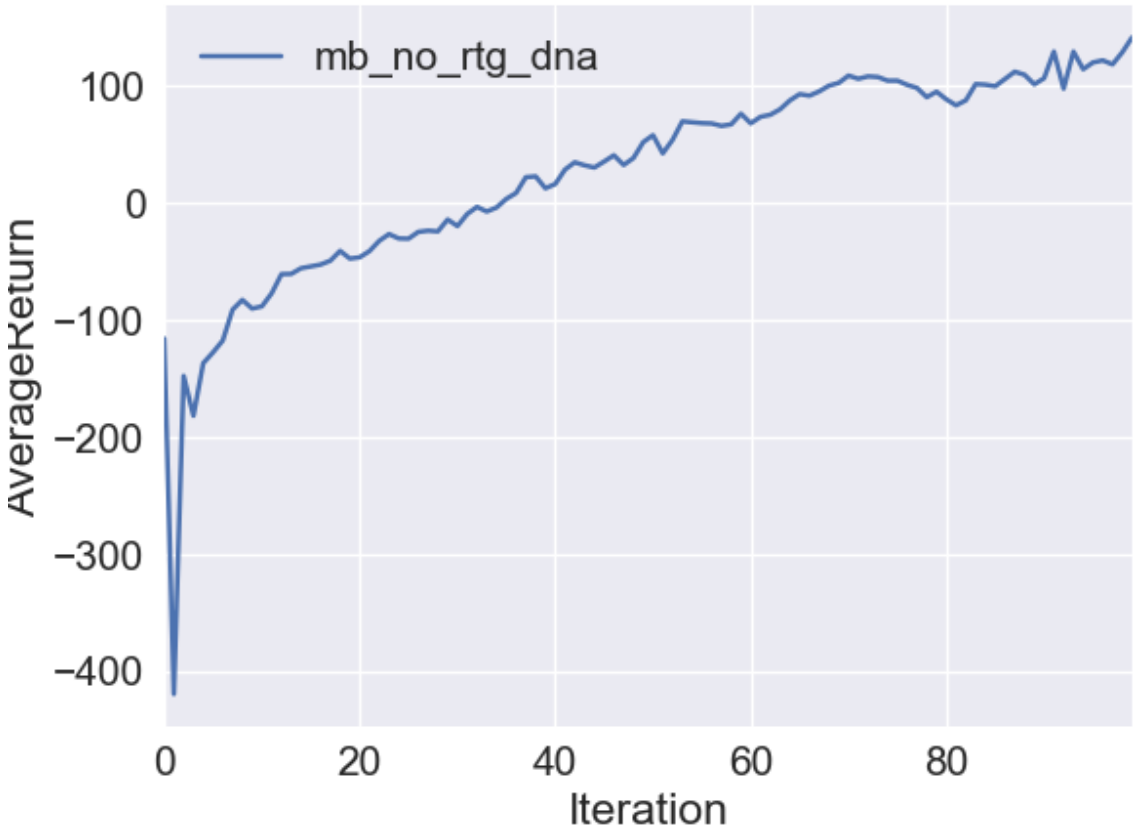

Figure 5: Half Cheetah

beyond 0.01 tended to make things unstable. I tried baseline prediction with smaller batch size, but since it did not seem to help much, I stopped using it for larger batch size (to save time).

```
python train_pg.py HalfCheetah-v1 -n 100 -b 35000 -rtg -ep 150 --discount 0.9 \
--exp_name mb_no_rtg_dna --learning_rate=0.01 --seed 102 --n_layers 3 --size 128

python plot.py data/mb_no_rtg_dna_HalfCheetah-v1_20-09-2017_20-51-02
```

Results

My policy gradient implementation achieves 143 by iteration 100 of the algorithm (just shy of the target). A few more iterations and it would have been over. A larger batchsize and network did help. See Figure 5.