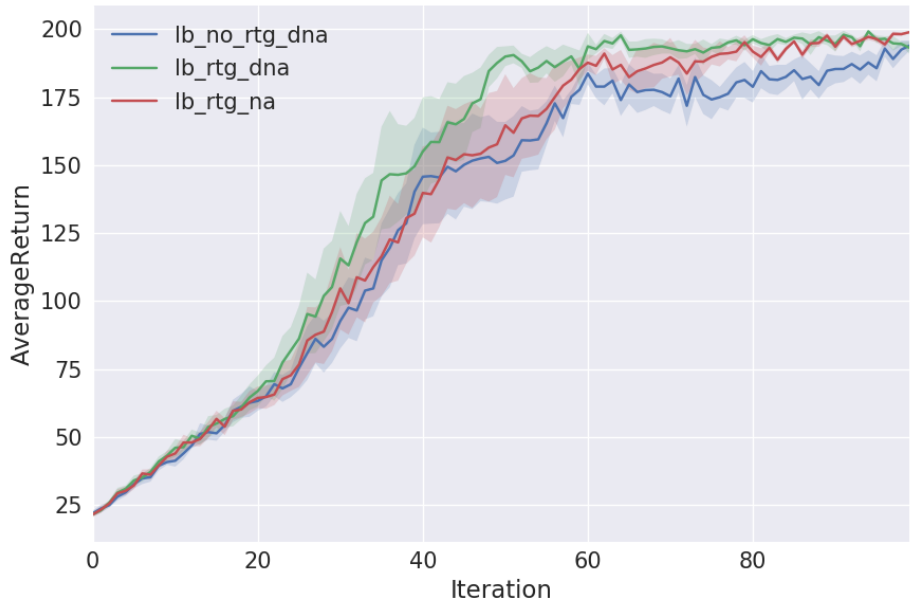
Class: Deep RL
SID: 24274554
Name: Varun Tolani
HW: 2

4

1.

Average Return for Small Batch experiments on Cartpole

Average Return for Large Batch experiments on Cartpole



In this experiment I used a single hidden layer mlp of size 32 with tanh nonlinearities to represent the policy for the cartpole environment. I use a batch size of 1000 per policy gradient

iteration for the small batch and 5000 for the large batch. The pg algorithm was run for 100 iterations and shown above is the reward averaged over 5 different trials. I experimented with using trajectory centric policy gradient (no_rtg) vs reward to go policy gradient(rtg) as well as with advantage normaliziation (na) and unnormalized advantages(dna).

Without advantage normalization the reward to go gradient estimator had better performance as seen above in both the small and large batch cases.

For both small and large batch sizes it seems that advantage normalization(red) is less effective than unnormalized advantage(green).

Given the math I expected reward-to-go gradients to converge more quickly than trajectory-centric ones, as they exploit causality of the system. I also expected the normalized advantage policy gradients to converge more quickly than the unnormalized ones, but they seem to consistently converge slightly slower. Perhaps this is due to the large variance of policy gradient, and the graphs above show could show a statistical anomoly.

Comparing the two graphs it is obvious that the larger batch size exhibits a much more strongly defined upward trend with significantly less variance. I think this can be attributed to the much larger batch size, leading to finer approximiations of the gradient and reducing overall variance (but speeding up computation time significantly).

2

Based on the experiments from the previous section I will be using reward-to-go. While in the carpole experiment advantage normalization had a slightly negative effect here I found that it greatly decreased variance of the algorithm. The algorithm converged much more predictably with advantage normalization. I also found that reducing the learning rate actually helped reduce the overall variance during training. The various experiments I ran were:

python train_pg InvertedPendulum-v1 -n 100 -b 1000 -e 5 -rtg -dna -l 2 -s 32 –exp_name batch1000_rtg_dna

python train_pg InvertedPendulum-v1 -n 100 -b 1000 -e 5 -rtg -l 2 -s 32 –exp_name batch1000_rtg_na

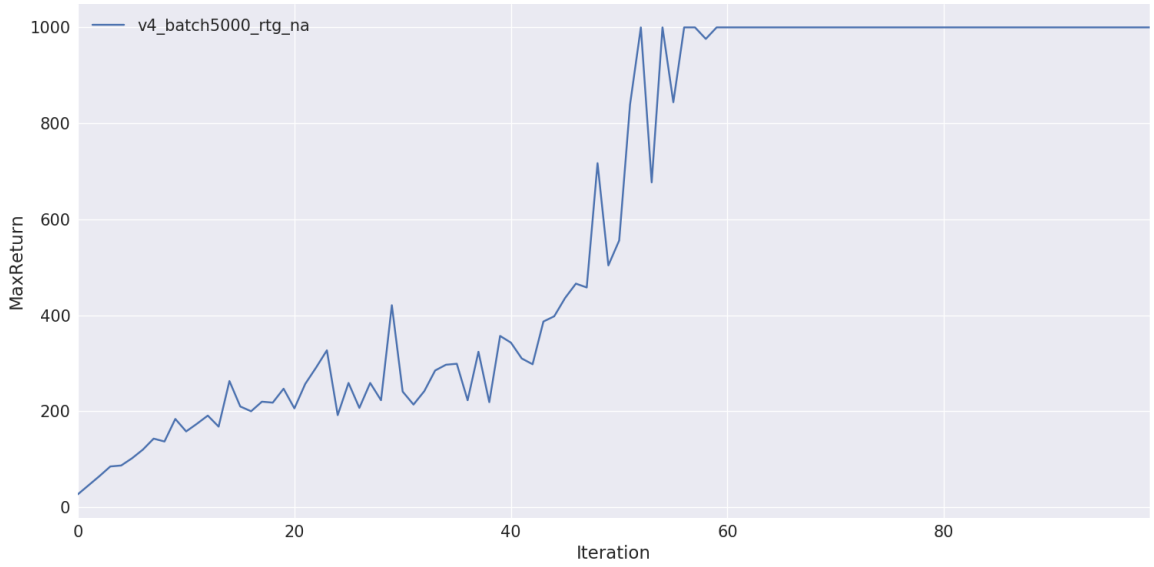
python train_pg.py InvertedPendulum-v1 -n 100 -b 2000 -e 5 -rtg -l 2 -s 32 –exp_name batch2000_rtg_na

python train_pg.py InvertedPendulum-v1 -n 100 -b 5000 -e 1 -rtg -l 2 -s 64 -lr 2.5e-3 –exp_name v4_batch5000_rtg_na

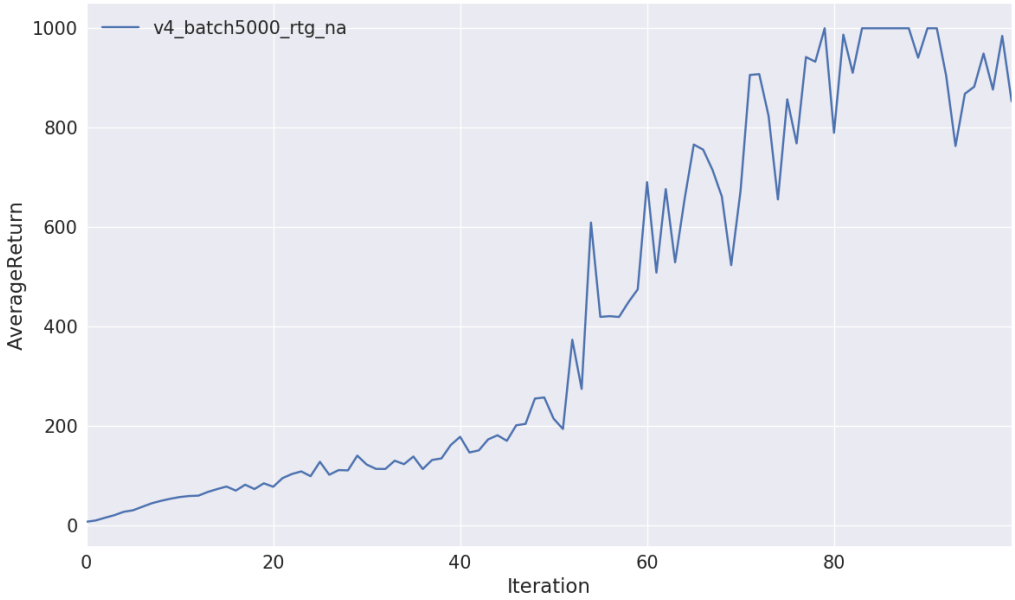
Class: Deep RL
SID: 24274554
Name: Varun Tolani
HW: 2

While batch sizes of 1000 and 2000 did achieve a maximum return of 1000 it was quite difficult to make the policy converge, as the maximum return would oscillate wildly. By raising the batch size to 5000 I was able to vastly reduce the variance in the algorithm and converge much more smoothly.

Max Return for Batch Size 5000 on Inverted Pendulum



Average Return for Batch Size 5000 on Inverted Pendulum



Class: Deep RL
SID: 24274554
Name: Varun Tolani
HW: 2

5

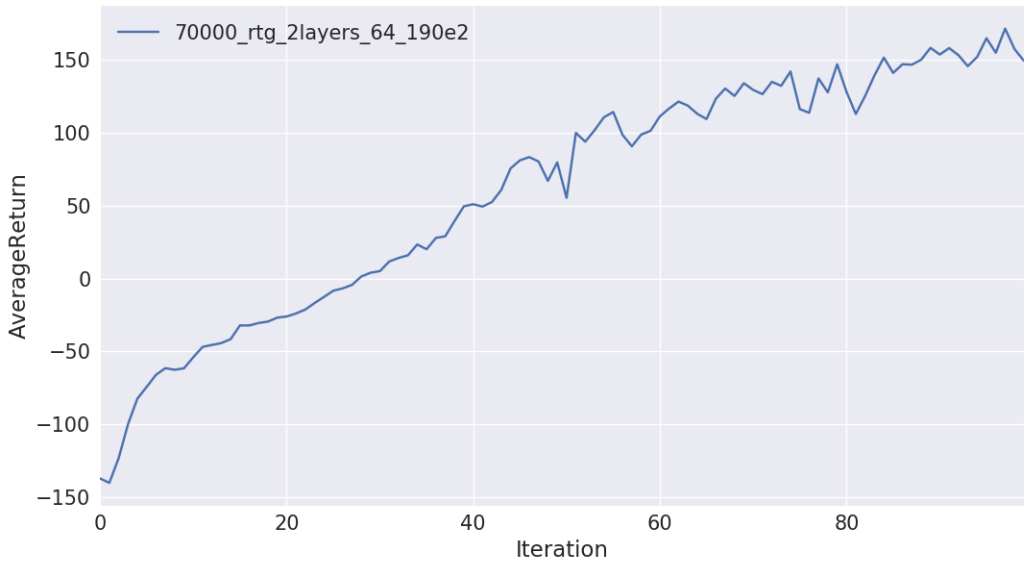
NN Baseline vs No Baseline Inverted Pendulum

The nn baseline I implimented was a 2 layer mlp size with 2 hidden layers size 64. In each iteration of policy gradient I update the baseline nn with one gradient step of size 1e-3. From the plots it can be seen that averaged over multiple trajectories the nn baseline performs on average around the same as the non baseline method.

Class: Deep RL
SID: 24274554
Name: Varun Tolani
HW: 2

6

Average Return for Batch Size 70000 on HalfCheetah



I achieved this plot using reward to go, batch size 70000, a neural net with 2 hiddden layers size 64 each, and learning rate 1.9e-2

python train_pg.py HalfCheetah-v1 -ep 150 –discount 0.9 -n 100 -b 70000 -e 2 -rtg -l 2 -s 64 -lr 1.9e-2 –exp_name 70000_rtg_2layers_64_190e2