CS294-112 Deep Reinforcement Learning HW2: Policy Gradients

Nathan Lambert, nol@berkeley.edu, October 4, 2018

1 Introduction

2 Review

Problem 1. State-dependent baseline: In lecture we saw that the policy gradient is unbiased if the baseline is a constant with respect to τ (Equation **??**). The purpose of this problem is to help convince ourselves that subtracting a state-dependent baseline from the return keeps the policy gradient unbiased. For clarity we will use $p_\theta(\tau)$ instead of $\pi_\theta(\tau)$, although they mean the same thing. Using the law of iterated expectations we will show that the policy gradient is still unbiased if the baseline b is function of a state at a particular timestep of τ (Equation **??**). Recall from equation **??** that the policy gradient can be expressed as

$$\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\nabla_\theta \log p_\theta(\tau) r(\tau)\right].$$

By breaking up $p_\theta(\tau)$ into dynamics and policy terms, we can discard the dynamics terms, which are not functions of θ:

$$\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{t'=1}^{T} r(s_{t'}, a_{t'})\right)\right].$$

When we subtract a state dependent baseline $b(s_t)$ (recall equation **??**) we get

$$\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\left(\left(\sum_{t'=1}^{T} r(s_{t'}, a_{t'})\right) - b(s_t)\right)\right].$$

Our goal for this problem is to show that

$$\mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t) \right] = 0.$$

By linearity of expectation we can consider each term in this sum independently, so we can equivalently show that

$$\sum_{t=1}^{T} \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \left(b(s_t) \right) \right] = 0. \tag{1}$$

(a) Using the chain rule, we can express $p_\theta(\tau)$ as a product of the state-action marginal (s_t, a_t) and the probability of the rest of the trajectory conditioned on (s_t, a_t) (which we denote as $(\tau / s_t, a_t | s_t, a_t)$):

$$p_\theta(\tau) = p_\theta(s_t, a_t) p_\theta(\tau / s_t, a_t | s_t, a_t)$$

Please show equation 1 by using the law of iterated expectations, breaking $\mathbb{E}_{\tau \sim p_\theta(\tau)}$ by decoupling the state-action marginal from the rest of the trajectory.
Recall: The law of iterated expectations gives us the following relation:

$$\mathbb{E}\big[\mathbb{E}\big] = \mathbb{E}_Y \big[\mathbb{E}_X [X|Y] \big]$$

Solution: Along with the chain rule expansion of the state action distribution to get an expand expectation. Starting with equation above for the policy gradient with the state dependent baseline:

$$\mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\left(\sum_{t'=1}^{T} r(s_{t'}, a_{t'}) \right) - b(s_t) \right) \right].$$

Then we break the expectation by decoupling the state-action marginal from the rest of the distribution.

$$\mathbb{E}_{\tau \sim p_\theta(s_t, a_t)} \left[\mathbb{E}_{\tau \sim p_\theta(\tau / s_t, a_t | s_t, a_t)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\left(\sum_{t'=1}^{T} r(s_{t'}, a_{t'}) \right) - b(s_t) \right) \right] \right].$$

From here, we will have to pull terms out of the expectations to show that the estimator is unbiased. First, we can removed the terms dependent only on s_t, a_t from the inner expectation. The clear

$$\mathbb{E}_{\tau \sim p_\theta(s_t, a_t)} \left[\sum_{t=1}^{T} \mathbb{E}_{\tau \sim p_\theta(\tau / s_t, a_t | s_t, a_t)} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \left(\left(\sum_{t'=1}^{T} r(s_{t'}, a_{t'}) \right) - b(s_t) \right) \right] \right].$$

Then looking at the baseline term,

$$\mathbb{E}_{\tau \sim p_\theta(\tau/s_t, a_t|s_t, a_t)} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \left(\left(\sum_{t'=1}^{T} r(s_{t'}, a_{t'}) \right) - b(s_t) \right) \right]$$

Then the baseline term here,

$$\mathbb{E}_{\tau \sim p_\theta(\tau/s_t, a_t|s_t, a_t)} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \left(-b(s_t) \right) \right] = - \int \pi_\theta(\tau/s_t, a_t|s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t) d\tau$$

In this equation, we know s_t, so we can pull it out of the expression:

$$= -b(s_t) \int \pi_\theta(\tau/s_t, a_t|s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t) d\tau$$

$$= -b(s_t) \int \nabla_\theta \pi_\theta(\tau/s_t, a_t|s_t, a_t) d\tau$$

$$= -b(s_t) \nabla_\theta \int \pi_\theta(\tau/s_t, a_t|s_t, a_t) d\tau$$

$$= -b(s_t) \nabla_\theta 1 = 0$$

This shows that the state dependent baseline is unbiased when you can condition the rest of the trajectory on the state-action marginal.

(b) Alternatively, we can consider the structure of the MDP and express $p_\theta(\tau)$ as a product of the trajectory distribution up to s_t (which we denote as $(s_{1:t}, a_{1:t-1})$) and the trajectory distribution after s_t conditioned on the first part (which we denote as $(s_{t+1:T}, a_{t:T}|s_{1:t}, a_{1:t-1})$):

$$p_\theta(\tau) = p_\theta(s_{1:t}, a_{1:t-1}) p_\theta(s_{t+1:T}, a_{t:T}|s_{1:t}, a_{1:t-1})$$

(a) Explain why, for the inner expectation, conditioning on $(s_1, a_1, ..., a_{t^*-1}, s_{t^*})$ is equivalent to conditioning only on s_{t^*}.
Solution: The inner expectation is equivalent when conditioning on all previous actions because of the structure of a MDP. The current state encompasses all the information needed to know how the state will propagate!

(b) Please show equation 1 by using the law of iterated expectations, breaking $\mathbb{E}_{\tau \sim p_\theta(\tau)}$ by decoupling trajectory up to s_t from the trajectory after s_t.
Solution:

$$\mathbb{E}_{\tau\sim p_\theta(s_{1:t},a_{1:t-1})}\left[\mathbb{E}_{\tau\sim p_\theta(s_{t+1:T},a_{t:T}|s_{1:t},a_{1:t-1})}\left[\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_t|s_t)\left(\left(\sum_{t'=1}^{T}r(s_{t'},a_{t'})\right)-b(s_t)\right)\right]\right]$$

Then as before, we can move the first sum outside the first expectation and show that the inner expectation goes to zero, to show proof.

$$\mathbb{E}_{\tau\sim p_\theta(s_{1:t},a_{1:t-1})}\left[\sum_{t=1}^{T}\mathbb{E}_{\tau\sim p_\theta(s_{t+1:T},a_{t:T}|s_{1:t},a_{1:t-1})}\left[\nabla_\theta\log\pi_\theta(a_t|s_t)\left(\left(\sum_{t'=1}^{T}r(s_{t'},a_{t'})\right)-b(s_t)\right)\right]\right]$$

What we are focusing on is the baseline term, so consider:

$$\mathbb{E}_{\tau\sim p_\theta(s_{1:t},a_{1:t-1})}\left[\sum_{t=1}^{T}\mathbb{E}_{\tau\sim p_\theta(s_{t+1:T},a_{t:T}|s_{1:t},a_{1:t-1})}\left[\nabla_\theta\log\pi_\theta(a_t|s_t)\left(b(s_t)\right)\right]\right]$$

Now as above, we look at the calculation of the inner expectation and that the conditioning of the past trajectory gives us a constant b_t over out time frame.

$$= b(s_t)\int\pi_\theta(s_{t+1:T},a_{t:T}|s_{1:t},a_{1:t-1})\nabla_\theta\log\pi_\theta(a_t|s_t)d\tau$$

Which becomes the integral over the entire space of a probability distribution, giving us the solution — the baseline is unbiased!

$$= b(s_t)\nabla_\theta 1 = 0$$

Since the policy gradient with respect to θ can be decoupled as a summation of terms over timesteps $t\in[1,T]$, because we have shown that the policy gradient is unbiased for each of these terms, the entire policy gradient is also unbiased with respect to a vector of state-dependent baselines over the timesteps: $[b(s_1),b(s_2),...b(s_T)]$.

3 Code Setup

4 Constructing the computation graph

Problem 2. Neural networks: We will now begin to implement a neural network that parametrizes π_θ.

5 Implement Policy Gradient

5.1 Implementing the policy gradient loop

Problem 3. Policy Gradient: Recall from lecture that an RL algorithm can viewed as consisting of three parts, which are reflected in the training loop of

5.2 Experiments

After you have implemented the code, we will run experiments to get a feel for how different settings impact the performance of policy gradient methods.

Problem 4. CartPole: Run the PG algorithm in the discrete `CartPole-v0` environment from the command line. **Deliverables for report:**

- Graph the results of your experiments **using the plot.py file we provide.** Create two graphs.
  - In the first graph, compare the learning curves (average return at each iteration) for the experiments prefixed with `sb_`. (The small batch experiments.)

Figure 1: Small batch experiments.

– In the second graph, compare the learning curves for the experiments prefixed with `lb_`. (The large batch experiments.)

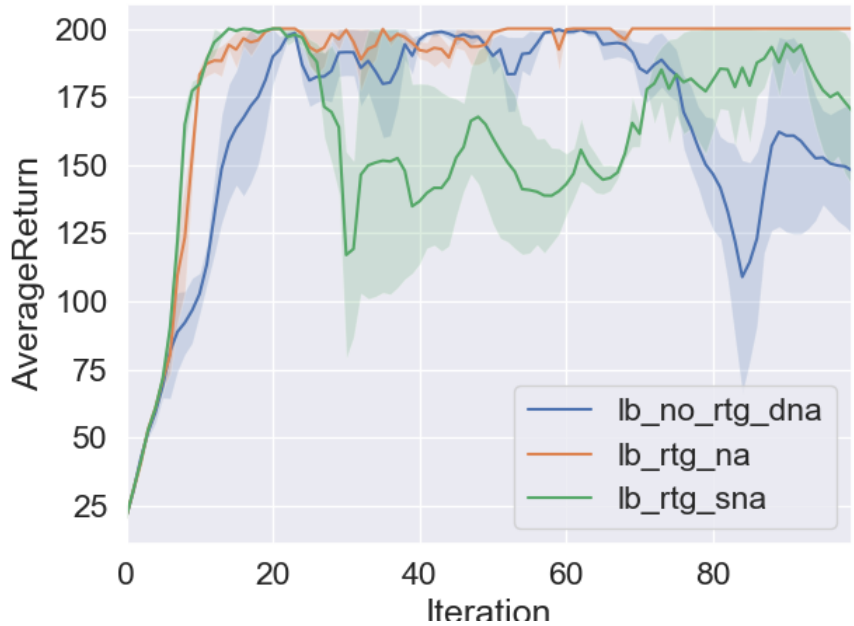

Figure 2: Large batch experiments.

- Answer the following questions briefly:
  - Which gradient estimator has better performance without advantage-centering—the trajectory-centric one, or the one using reward-to-go?
    Answer: Reward-to-go reached max performance quicker in both small and large batch mode, showing it is superior.
  - Did advantage centering help?
    Answer: Yes. Average centering helped in both small and large batch regime.
  - Did the batch size make an impact?
    Answer: Yes. Larger batch size lowered the variance substantially and increased convergence time.

- My command line calls to run the experiments.

```
(1) $ python train_pg_f18.py CartPole-v0 -n 100 -b 1000 -e 3 -dna --
    exp_name sb_no_rtg_dna
```

```
(2) $ python train_pg_f18.py CartPole-v0 -n 100 -b 1000 -e 3 -rtg -dna --
   exp_name sb_rtg_dna
(3) $python train_pg_f18.py CartPole-v0 -n 100 -b 1000 -e 3 -rtg --
   exp_name sb_rtg_na
(4) $ python train_pg_f18.py CartPole-v0 -n 100 -b 5000 -e 3 -dna --
   exp_name lb_no_rtg_dna
(5) $ python train_pg_f18.py CartPole-v0 -n 100 -b 5000 -e 3 -rtg -dna --
   exp_name lb_rtg_dna
(6) $ python train_pg_f18.py CartPole-v0 -n 100 -b 5000 -e 3 -rtg --
   exp_name lb_rtg_na
```

What to Expect:

- The best configuration of CartPole in both the large and small batch cases converge to a maximum score of 200.

Problem 5. InvertedPendulum: Run experiments in `InvertedPendulum-v2` continuous control environment.

Deliverables:

- Given the `b*` and `r*` you found, provide a learning curve where the policy gets to optimum (maximum score of 1000) in less than 100 iterations. (This may be for a single random seed, or averaged over multiple.)
  Answer: I found a learning rate of above .01 with a batch size below 100 to diverge and not regularly reach the solution with smaller batch sizes. For me, **BATCH: 250, LR .01** converged regularly, as shown below. If using this for real experiments, it may make sense to make learning slightly slower for more safety.

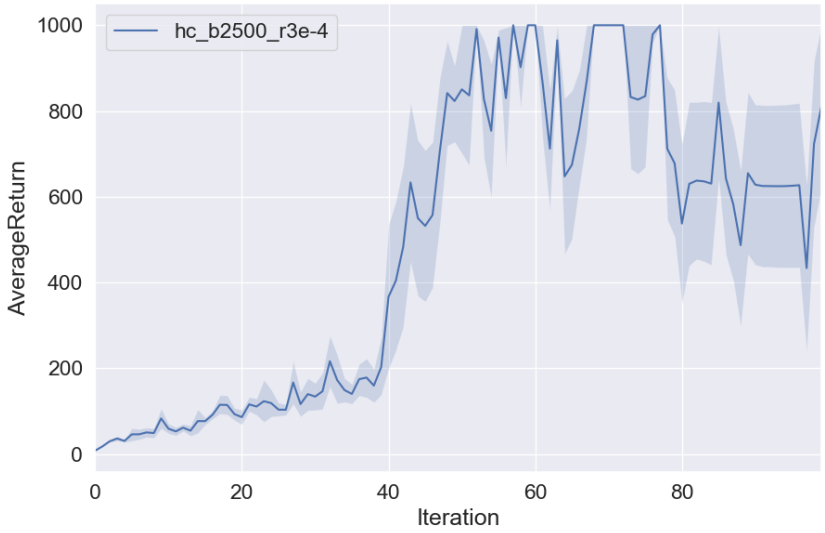

Figure 3: Convergence with largest learning rate and smallest batch size.

- Provide the exact command line configurations you used to run your experiments. **My command**:

```
(1) $ python train_pg_f18.py InvertedPendulum-v2 -ep 1250 --discount 0.9
   -n 100 -e 5 -l 2 -s 64 -b 250 -lr .01 -rtg --exp_name hc_b250_r1e-2
```

6 Implement Neural Network Baselines

For the rest of the assignment we will use "reward-to-go."

Problem 6. Neural network baseline: We will now implement a value function as a state-dependent neural network baseline. The sections in the code are marked by "Problem 6."

(a) In `Agent.build_computation_graph` implement V_ϕ^π, a neural network that predicts the expected return conditioned on a state. Also implement the loss function to train this network and its update operation `self.baseline_op`.

(b) In `Agent.compute_advantage`, use the neural network to predict the expected state-conditioned return (use the session to call `self.baseline_prediction`), normalize it to match the statistics of the current batch of "reward-to-go", and subtract this value from the "reward-to-go" to yield an estimate of the advantage. This implements $\left(\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{it'}, a_{it'})\right) - V_\phi^\pi(s_{it})$.

(c) In `Agent.update_parameters`, update the parameters of the the neural network baseline by using the Tensorflow session to call `self.baseline_op`. "Rescale" the target values for the neural network baseline to have a mean of zero and a standard deviation of one.

7 More Complex Tasks

Note: The following tasks would take quite a bit of time to train. Please start early!

Problem 7: LunarLander For this problem, you will use your policy gradient implementation to solve `LunarLanderContinuous-v2`. Use an episode length of 1000. The purpose of this problem is to help you debug your baseline implementation. Run the following command:

```
python train_pg_f18.py LunarLanderContinuous-v2 -ep 1000 --discount 0.99 -n
   100 -e 3 -l 2 -s 64 -b 40000 -lr 0.005 -rtg --nn_baseline --exp_name
   ll_b40000_r0.005
```

Deliverables:

- Plot a learning curve for the above command.

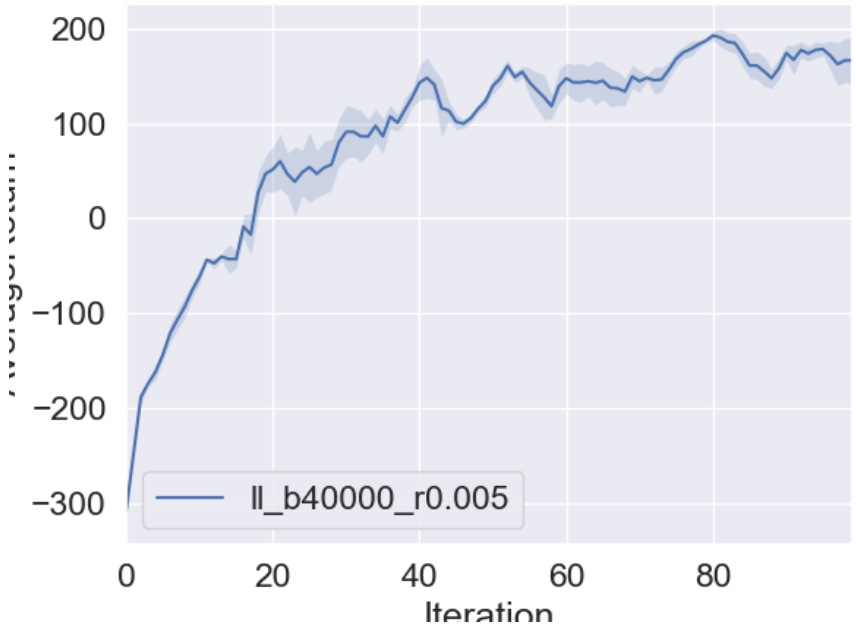

Figure 4: My LunarLander learning curve. It works!

Problem 8: HalfCheetah For this problem, you will use your policy gradient implementation to solve `HalfCheetah-v2`. Use an episode length of 150, which is shorter than the default of 1000 for HalfCheetah (which would speed up your training significantly). Search over batch sizes `b` $\in [10000, 30000, 50000]$ and learning rates `r` $\in [0.005, 0.01, 0.02]$ to replace `<b>` and `<r>` below:

```
python train_pg_f18.py HalfCheetah-v2 -ep 150 --discount 0.9 -n 100 -e 3 -l 2
    -s 32 -b <b> -lr <r> --exp_name hc_b<b>_r<r>
```

Deliverables:

- How did the batch size and learning rate affect the performance?
  Answer: A higher batch size and learning rate gives the most reliable performance. The plot clearly shows the lower batch sizes making slower progress and the lower learning rate learning slower, but with more stability. A larger tradeoff gives you performance at risk of divergence.

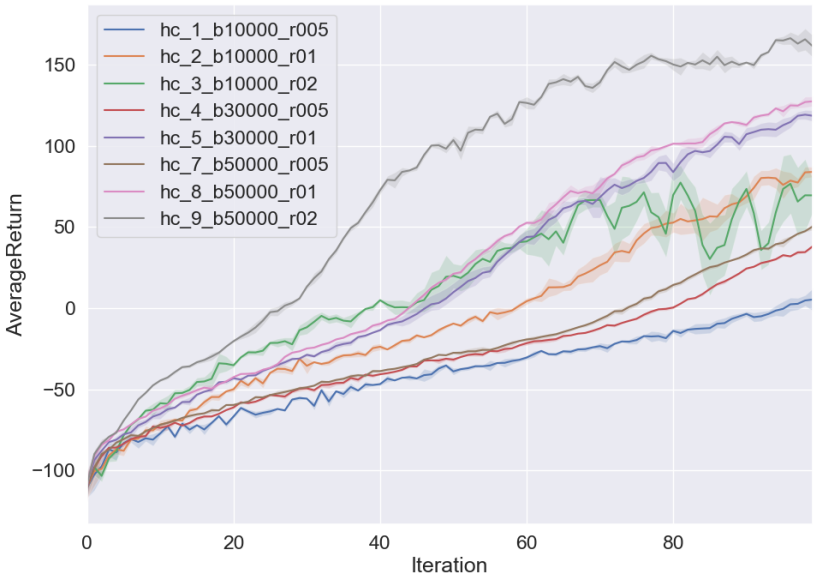

Figure 5: Grid Search for HalfCheetah.

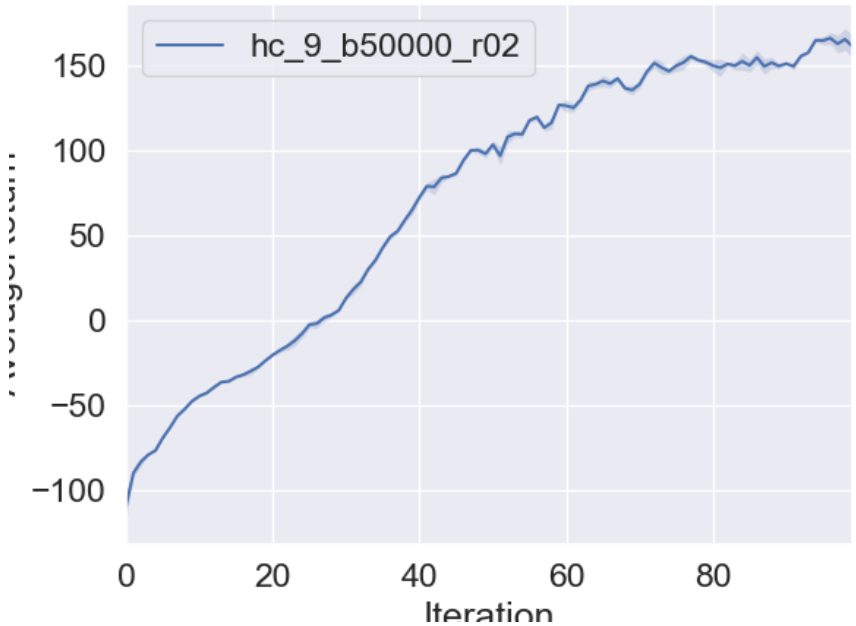

Figure 6: Best batch (50000) and learning rate (.02) for the half cheetah task.

- Here are my commands to run:

```
python train_pg_f18.py HalfCheetah-v2 -ep 150 --discount 0.95 -n 100 -e 3
    -l 2 -s 32 -b 50000 -lr .02 --exp_name hc_f1_b50000_r02
python train_pg_f18.py HalfCheetah-v2 -ep 150 --discount 0.95 -n 100 -e 3
    -l 2 -s 32 -b 50000 -lr .02 -rtg --exp_name hc_f2_b50000_r02
python train_pg_f18.py HalfCheetah-v2 -ep 150 --discount 0.95 -n 100 -e 3
    -l 2 -s 32 -b 50000 -lr .02 --nn_baseline --exp_name hc_f3_b50000_r02
python train_pg_f18.py HalfCheetah-v2 -ep 150 --discount 0.95 -n 100 -e 3
    -l 2 -s 32 -b 50000 -lr .02 -rtg --nn_baseline --exp_name
   hc_f4_b50000_r02
```

  The run with reward-to-go and the baseline should achieve an average score close to 200. Provide a single plot plotting the learning curves for all four runs.

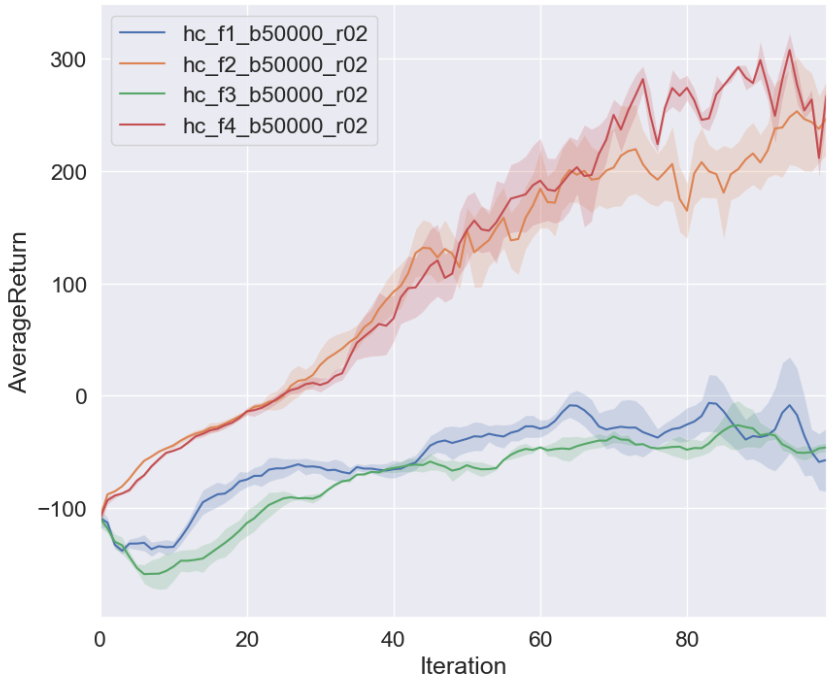

Figure 7: Results for the longer episode half cheetah. The nnbaseline and rtg tasks converge best, as expected!

8 Bonus!

Didn't get to these this time!

9 Submission

Your report should be a document containing

(a) Your mathematical response (written in LaTeX) for Problem 1.

(b) All graphs requested in Problems 4, 5, 7, and 8.

(c) Answers to short explanation questions in section 5 and 7.

(d) All command-line expressions you used to run your experiments.

(e) (Optionally) Your bonus results (command-line expressions, graphs, and a few sentences that comment on your findings).

Please also submit your modified `train_pg_f18.py` file. If your code includes additional files, provide a zip file including your `train_pg_f18.py` and all other files needed to run your code. Please include a `README.md` with instructions needed to exactly duplicate your results (including command-line expressions).

Turn in your assignment by September 19th 11:59pm on Gradescope. Uploade the zip file with your code to **HW2 Code**, and upload the PDF of your report to **HW2**.