CS 234: Assignment #3

Due date: February 16, 2022 at 6:00 PM (18:00) PST

These questions require thought, but do not require long answers. Please be as concise as possible.

We encourage students to discuss in groups for assignments. **However, each student must finish the problem set and programming assignment individually, and must turn in her/his assignment.** We ask that you abide by the university Honor Code and that of the Computer Science department, and make sure that all of your submitted work is done by yourself. If you have discussed the problems with others, please include a statement saying who you discussed problems with. Failure to follow these instructions will be reported to the Office of Community Standards. We reserve the right to run a fraud-detection software on your code.

Please review any additional instructions posted on the assignment page at http://web.stanford.edu/class/cs234/assignments.html. When you are ready to submit, please follow the instructions on the course website.

1 Policy Gradient Methods (50 pts coding + 15 pts writeup)

The goal of this problem is to experiment with policy gradient and its variants, including variance reduction methods. Your goals will be to set up policy gradient for both continuous and discrete environments, and implement a neural network baseline for variance reduction. The framework for the policy gradient algorithm is setup in `main.py`, and everything that you need to implement is in the files `network_utils.py`, `policy.py`, `policy_gradient.py` and `baseline_network.py`. The file has detailed instructions for each implementation task, but an overview of key steps in the algorithm is provided here.

1.1 REINFORCE

Recall the policy gradient theorem,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)\right]$$

REINFORCE is a Monte Carlo policy gradient algorithm, so we will be using the sampled returns G_t as unbiased estimates of $Q^{\pi_\theta}(s,a)$. The REINFORCE estimator can be expressed as the gradient of the following objective function:

$$J(\theta) = \frac{1}{\sum T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} \log\big(\pi_\theta(a_t^i|s_t^i)\big) G_t^i$$

where D is the set of all trajectories collected by policy π_θ, and $\tau^i = (s_0^i, a_0^i, r_0^i, s_1^i, \ldots, s_{T_i}^i, a_{T_i}^i, r_{T_i}^i)$ is trajectory i.

1.2 Baseline

One difficulty of training with the REINFORCE algorithm is that the Monte Carlo sampled return(s) G_t can have high variance. To reduce variance, we subtract a baseline $b_\phi(s)$ from the estimated returns when computing the policy gradient. A good baseline is the state value function, $V^{\pi_\theta}(s)$, which requires a training

update to ϕ to minimize the following mean-squared error loss:

$$L_{\text{MSE}}(\phi) = \frac{1}{\sum T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} (b_\phi(s_t^i) - G_t^i)^2$$

1.3 Advantage Normalization

After subtracting the baseline, we get the following new objective function:

$$J(\theta) = \frac{1}{\sum T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} \log\big(\pi_\theta(a_t^i|s_t^i)\big)\hat{A}_t^i$$

where

$$\hat{A}_t^i = G_t^i - b_\phi(s_t^i)$$

A second variance reduction technique is to normalize the computed advantages, $\hat{A}_t^i$, so that they have mean 0 and standard deviation 1. From a theoretical perspective, we can consider centering the advantages to be simply adjusting the advantages by a constant baseline, which does not change the policy gradient. Likewise, rescaling the advantages effectively changes the learning rate by a factor of $1/\sigma$, where σ is the standard deviation of the empirical advantages.

1.4 Coding Questions (50 pts)

The functions that you need to implement in `network_utils.py`, `policy.py`, `policy_gradient.py`, and `baseline_network.py` are enumerated here. Detailed instructions for each function can be found in the comments in each of these files.
Note: The "batch size" for all the arguments is $\sum T_i$ since we already flattened out all the episode observations, actions, and rewards for you.
In `network_utils.py`,

- `build_mlp`

In `policy.py`,

- `BasePolicy.act`
- `CategoricalPolicy.action_distribution`
- `GaussianPolicy.__init__`
- `GaussianPolicy.std`
- `GaussianPolicy.action_distribution`

In `policy_gradient.py`,

- `PolicyGradient.init_policy`
- `PolicyGradient.get_returns`
- `PolicyGradient.normalize_advantage`
- `PolicyGradient.update_policy`

In `baseline_network.py`,

- `BaselineNetwork.__init__`
- `BaselineNetwork.forward`
- `BaselineNetwork.calculate_advantage`
- `BaselineNetwork.update_baseline`

1.5 Testing

We have provided some basic tests to sanity check your implementation. **Please note that the tests are not comprehensive, and passing them does not guarantee a correct implementation**. Use the following command to run the tests:

```
python run_basic_tests.py
```

You can also add additional tests of your own design in `tests/test_basic.py`.

Tests pass in approximately 3.5 seconds.

1.6 Writeup Questions (15 pts)

(a) (3 pts) To compute the REINFORCE estimator, you will need to calculate the values $\{G_t\}_{t=1}^{T}$ (we drop the trajectory index i for simplicity), where

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

Naively, computing all these values takes $O(T^2)$ time. Describe how to compute them in $O(T)$ time.

Performing the calculations in reverse will get us there in $O(T)$ time. Specifically, suppose we have our rewards

$$\langle r_t, r_{t+1}, \ldots, r_{T-1}, r_T \rangle.$$

Now, start at the back. We have the base case:

- $G_T = r_T = \gamma^0 r_T$
- $G_{T-1} = \gamma^0 r_{T-1} + \gamma^1 r_T = r_{T-1} + \gamma^1 \gamma^0 r_T = r_{T-1} + \gamma G_T$.

Now, suppose the inductive hypothesis

$$G_{t+1} = \sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'} \overset{IH}{=} r_{t+1} + \gamma G_{t+2}.$$

We then have

$$\begin{aligned} G_t &= \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \\ &= r_t + \sum_{t'=t+1}^{T} \gamma^{t'-t} r_{t'} \\ &= r_t + \gamma \sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'} \\ &= r_t + \gamma G_{t+1}. \end{aligned}$$

Hence, we have by induction that

$$G_t = r_t + \gamma G_{t+1}.$$

This means we can just work backwards: compute G_T, save it; then compute $G_{T-1} = r_{T-1} + \gamma G_T$, save it; then compute $G_{T-2} = r_{T-2} + \gamma G_{T-1}$, save it, and so forth. In so doing, you'll stop at each timestep $T, T-1, \ldots, t$ once, giving you $O(T)$ runtime.

(b) (12 pts) The general form for running your policy gradient implementation is as follows:

```
python main.py --env-name ENV --seed SEED --no-baseline
```

if not using a baseline, or

```
python main.py --env-name ENV --seed SEED --baseline
```

if using a baseline. Here `ENV` should be `cartpole`, `pendulum`, or `cheetah`, and `SEED` should be a positive integer.

For each of the 3 environments, choose 3 random seeds and run the algorithm both without baseline and with baseline. Then plot the results using

```
python plot.py --env-name ENV --seeds SEEDS
```

where `SEEDS` should be a comma-separated list of seeds which you want to plot (e.g. `--seeds 1,2,3`). **Please include the plots (one for each environment) in your writeup, and comment on whether or not you observe improved performance when using a baseline.**

We have the following expectations about performance to receive full credit:

- cartpole: Should reach the max reward of 200 (although it may not stay there)
- pendulum: Should reach the max reward of 1000 (although it may not stay there)
- cheetah: Should reach at least 200 (Could be as large as 950)

1.7 Cartpole Runs

1.7.1 No Baseline

No baseline Cartpole runs consisted of the following shell script:

```
python main.py --env-name "cartpole" --seed 2020 --no-baseline;
python main.py --env-name "cartpole" --seed 2021 --no-baseline;
python main.py --env-name "cartpole" --seed 2022 --no-baseline;
```

The reward plots corresponding to these runs were then (respectively):

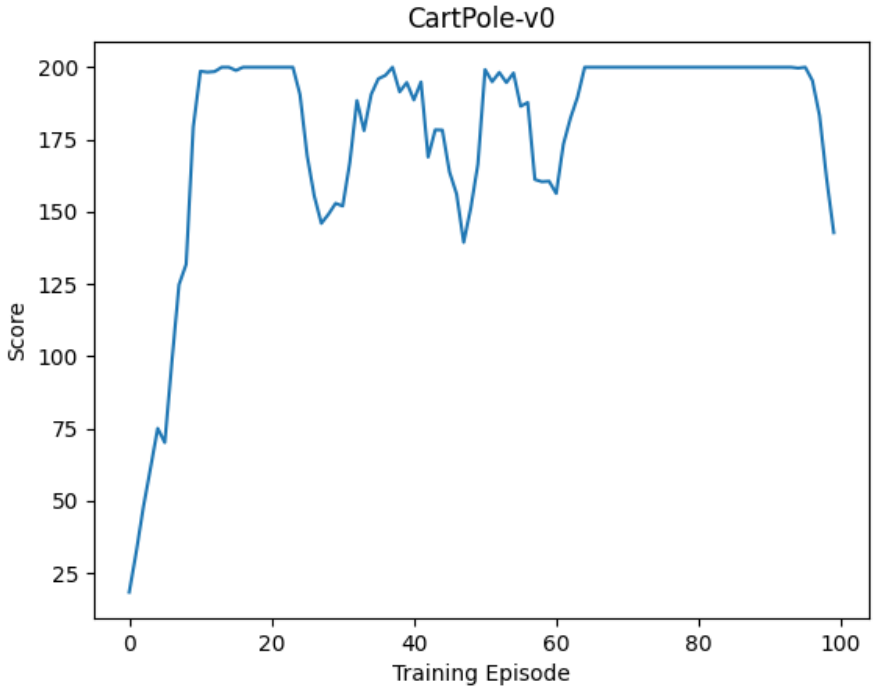

Figure 1: Cartpole // No Baseline // 2020

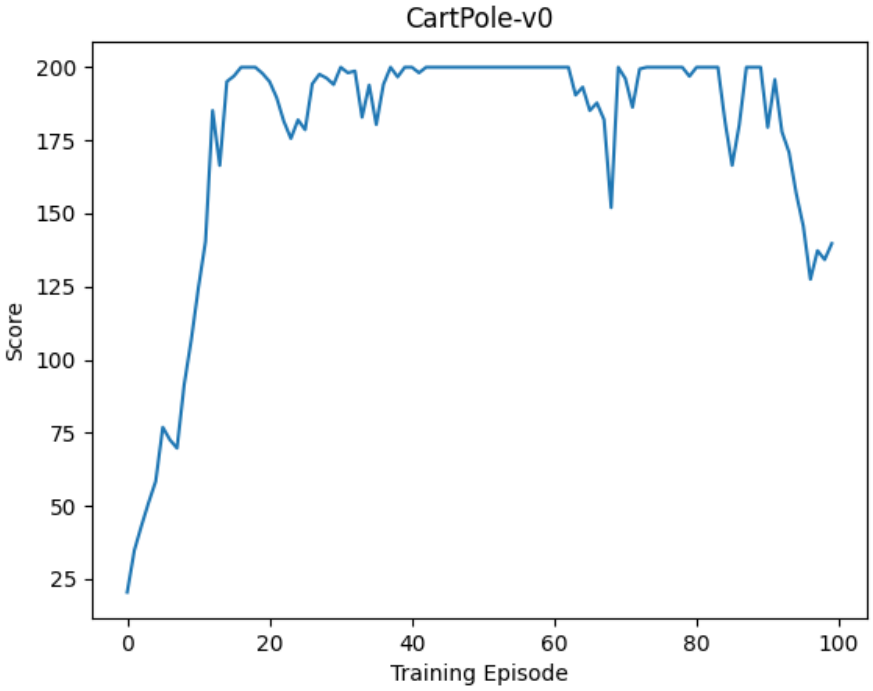

Figure 2: Cartpole // No Baseline // 2021

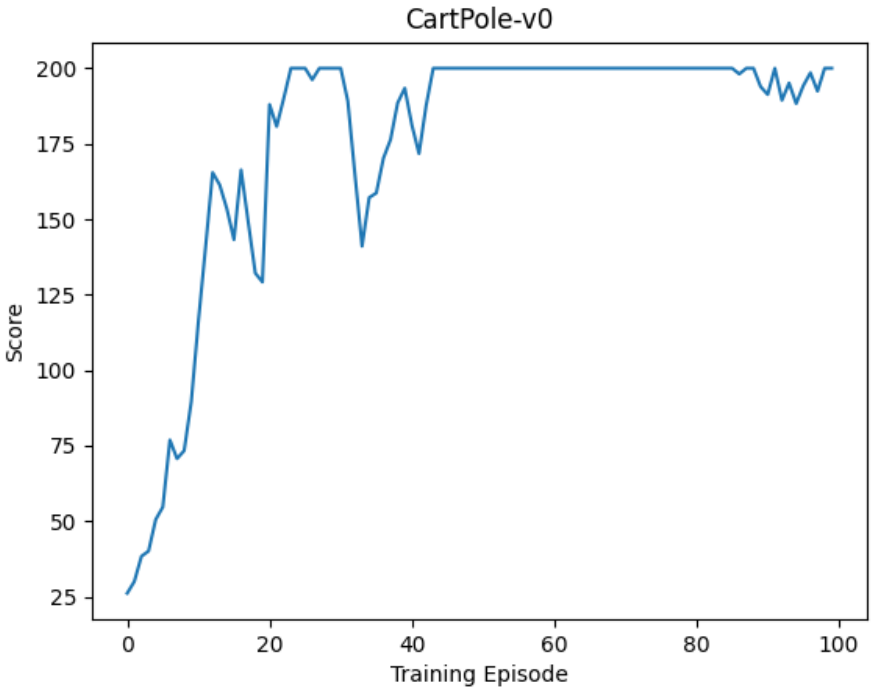

Figure 3: Cartpole // No Baseline // 2022

Each of these three runs – even without the baseline – achieved the maximum reward at some point during the first 100 training episodes.

1.7.2 Baseline

Then, for the baseline model, the shell script was:

```
python main.py --env-name "cartpole" --seed 2020 --baseline;
python main.py --env-name "cartpole" --seed 2021 --baseline;
python main.py --env-name "cartpole" --seed 2022 --baseline;
```

The reward plots corresponding to these runs were then (respectively):

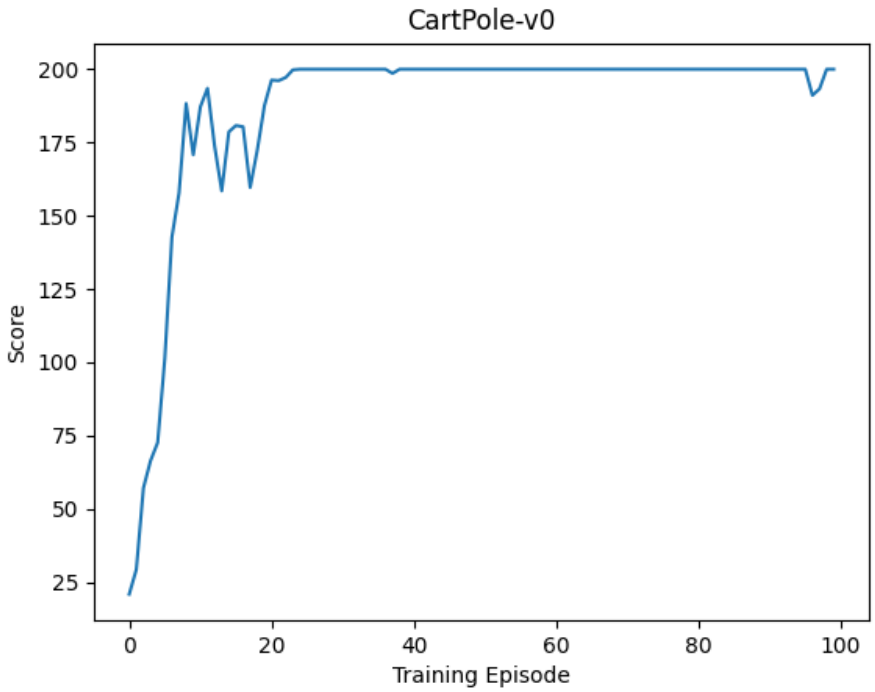

Figure 4: Cartpole // Baseline // 2020

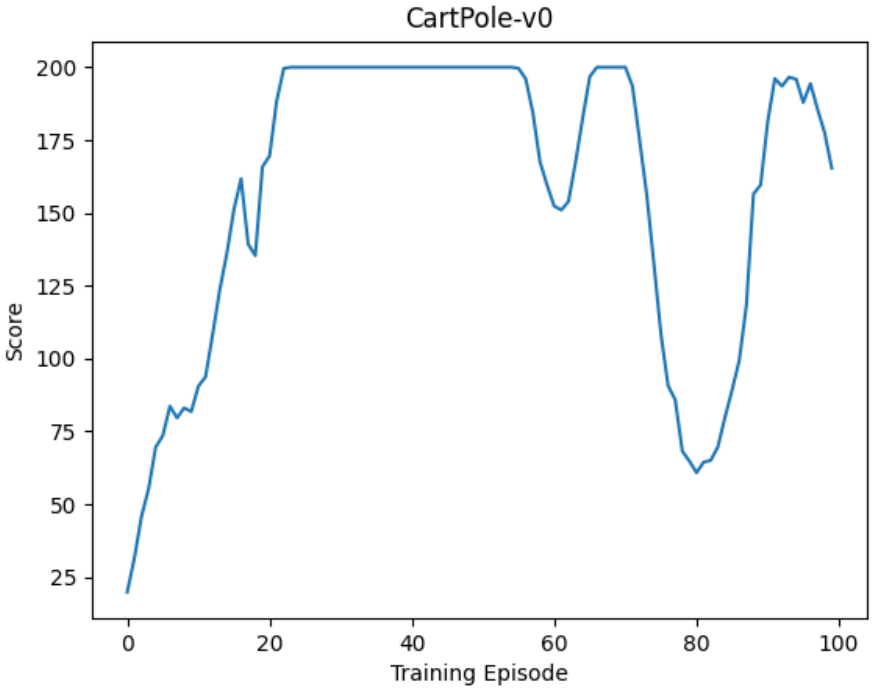

Figure 5: Cartpole // Baseline // 2021

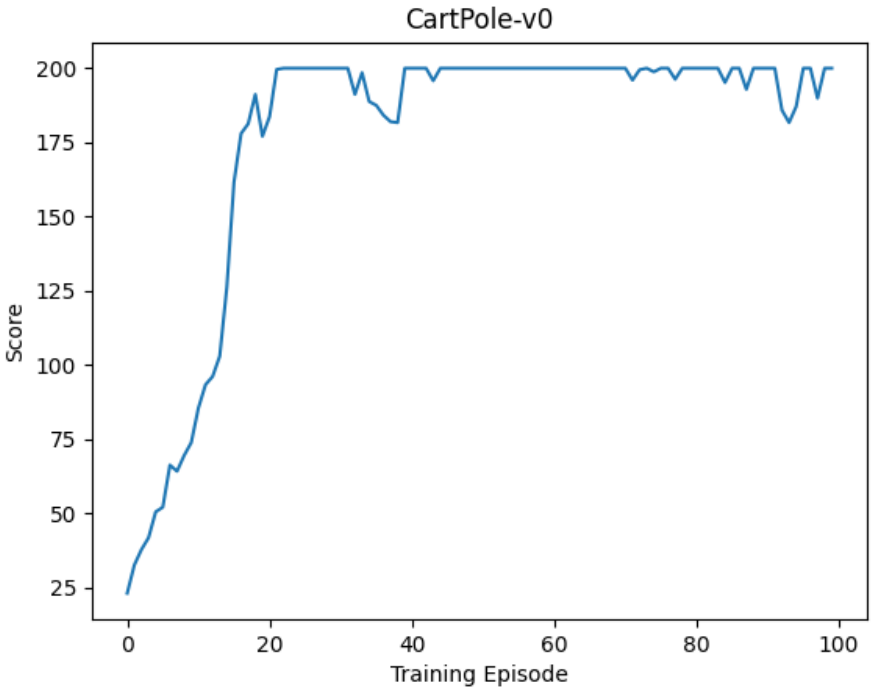

Figure 6: Cartpole // Baseline // 2022

This too achieved the maximum 200; for the first and third runs, the baseline version "stuck" at 200 much better. However, in the second run (seed 2021), the baseline model had a notable relapse/"falling" away from 200, though it ultimately recovered. Obviously, more runs are necessary for a concrete determination, but it looks like the baseline at least adds some "stickiness" to the maximum reward. So while both got the maximum reward, this makes the performance case for the baseline, as it appeared to help keep things close to the 200 max, with less drastic jumps away.

1.8 Pendulum Runs

1.8.1 No Baseline

No baseline Pendulum runs consisted of the following shell script:

```
python main.py --env-name "pendulum" --seed 2020 --no-baseline;
python main.py --env-name "pendulum" --seed 2021 --no-baseline;
python main.py --env-name "pendulum" --seed 2022 --no-baseline;
```

The reward plots corresponding to these runs were then (respectively):

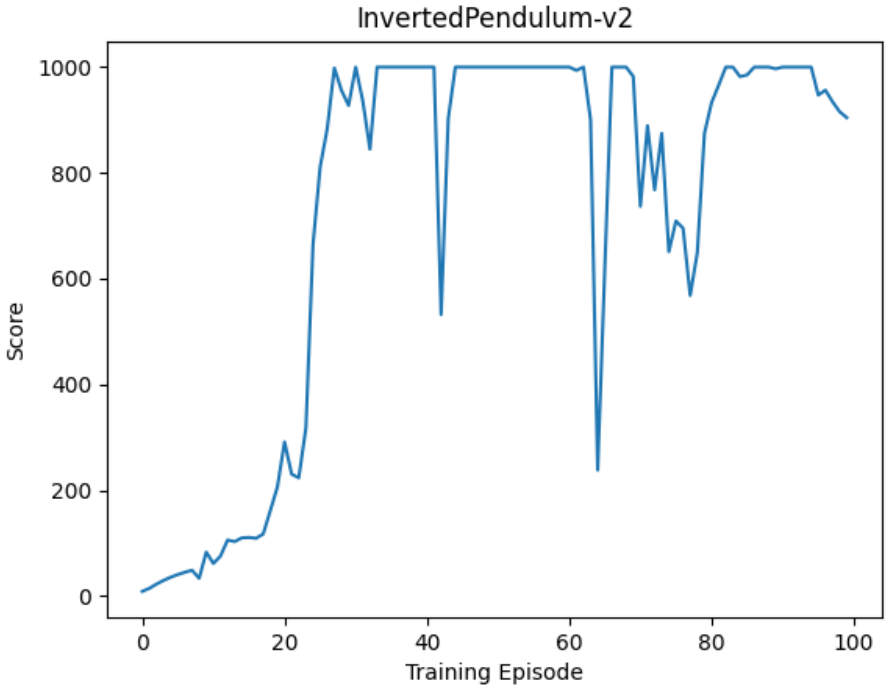

Figure 7: Inv. Pendulum // No Baseline // 2020

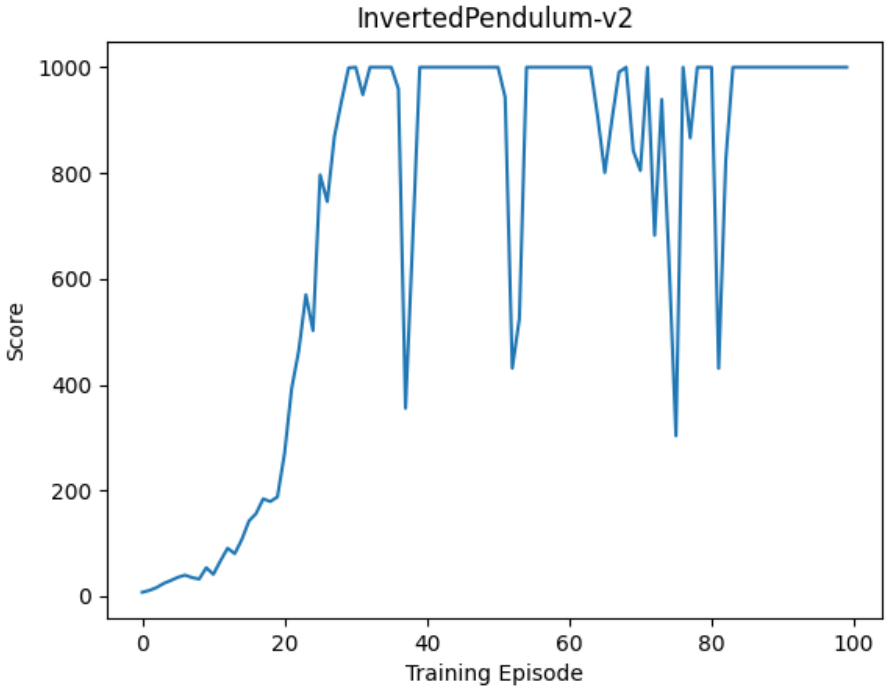

Figure 8: Inv. Pendulum // No Baseline // 2021

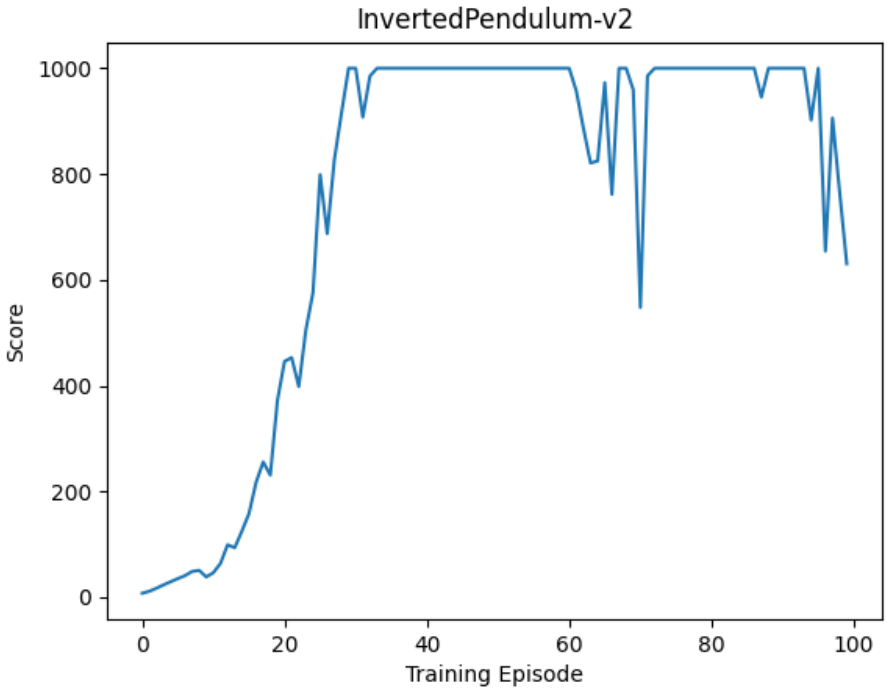

Figure 9: Inv. Pendulum // No Baseline // 2022

Each of these three runs – even without the baseline – achieved the requisite reward of 1000. However, at times, there were substantial deviations downward – i.e. it would get to 1000, and then fall as low as ~ 300

1.8.2 Baseline

Then, for the baseline model, the shell script was:

```
python main.py --env-name "pendulum" --seed 2020 --baseline;
python main.py --env-name "pendulum" --seed 2021 --baseline;
python main.py --env-name "pendulum" --seed 2022 --baseline;
```

The reward plots corresponding to these runs were then (respectively):

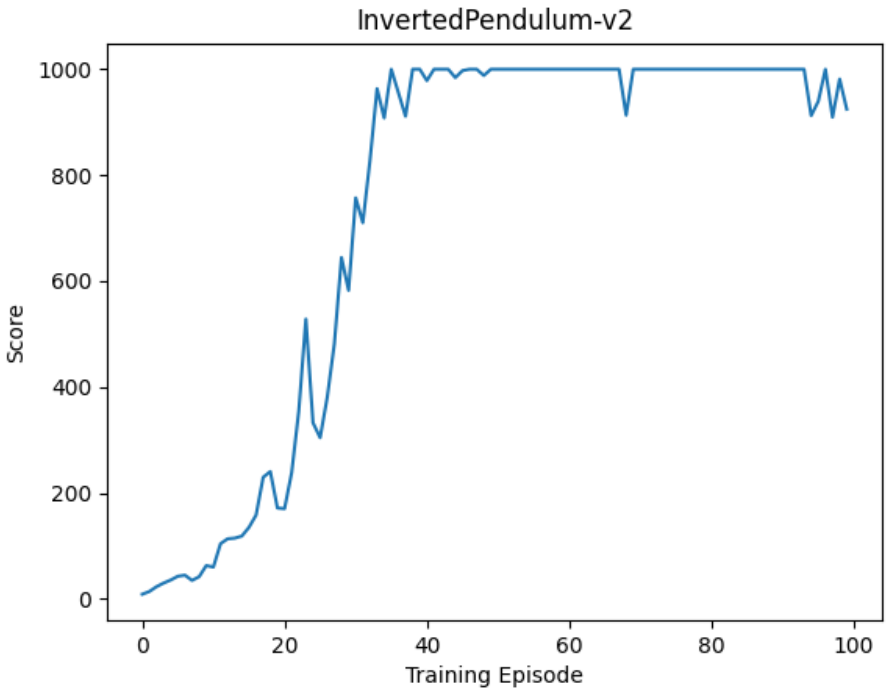

Figure 10: Inv. Pendulum // Baseline // 2020

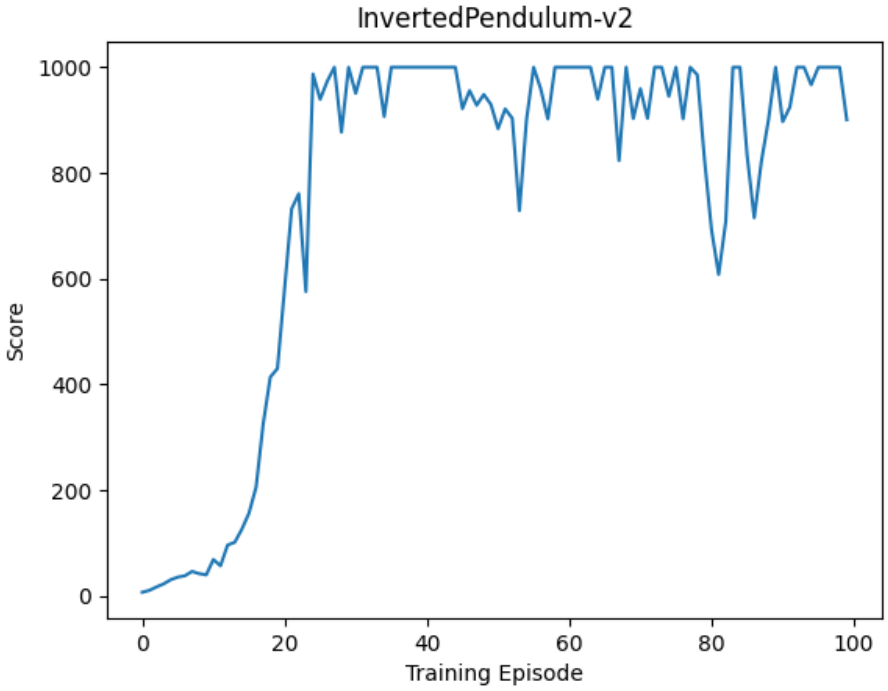

Figure 11: Inv. Pendulum // Baseline // 2021

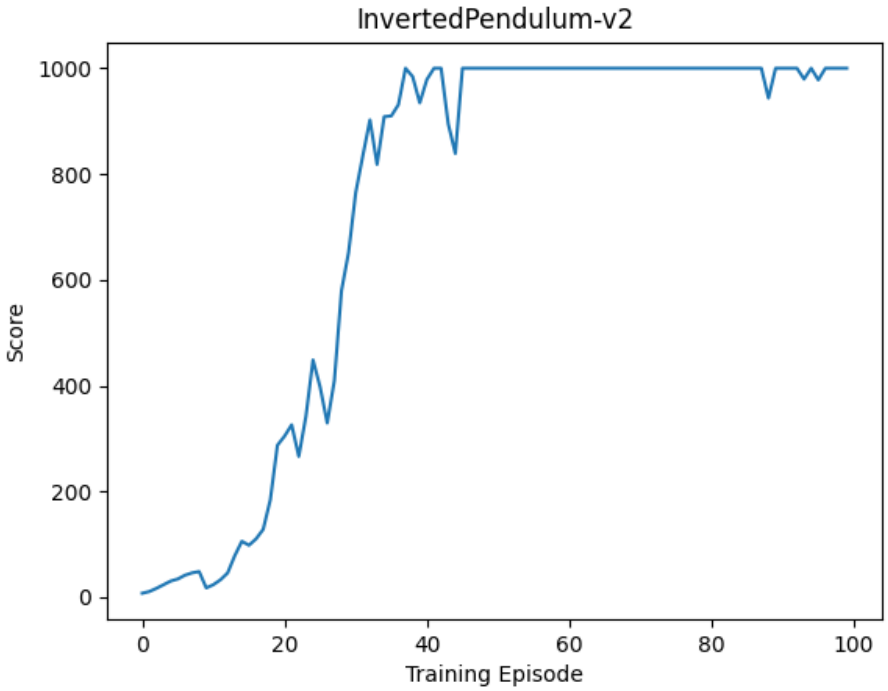

Figure 12: Inv. Pendulum // Baseline // 2022

This too achieved the required 1000; however, just as before, performance appeared a bit stickier, with (i) fewer downward deviations from 1000 and (ii) deviations of generally smaller magnitude. You could maybe argue that the baseline runs took a little bit slower to reach 1000 (looking at Seed 2020 and 2022), but this difference appears marginal and would need to be corroborated by more runs. In general, it seems that the inclusion of the baseline helped performance: while both achieved the max, the baseline did a better job of staying close to the max.

1.9 Cheetah Runs

1.9.1 No Baseline

No baseline Cheetah runs consisted of the following shell script:

```
python main.py --env-name "cheetah" --seed 20 --no-baseline;
python main.py --env-name "cheetah" --seed 21 --no-baseline;
python main.py --env-name "cheetah" --seed 22 --no-baseline;
```

The reward plots corresponding to these runs were then (respectively):

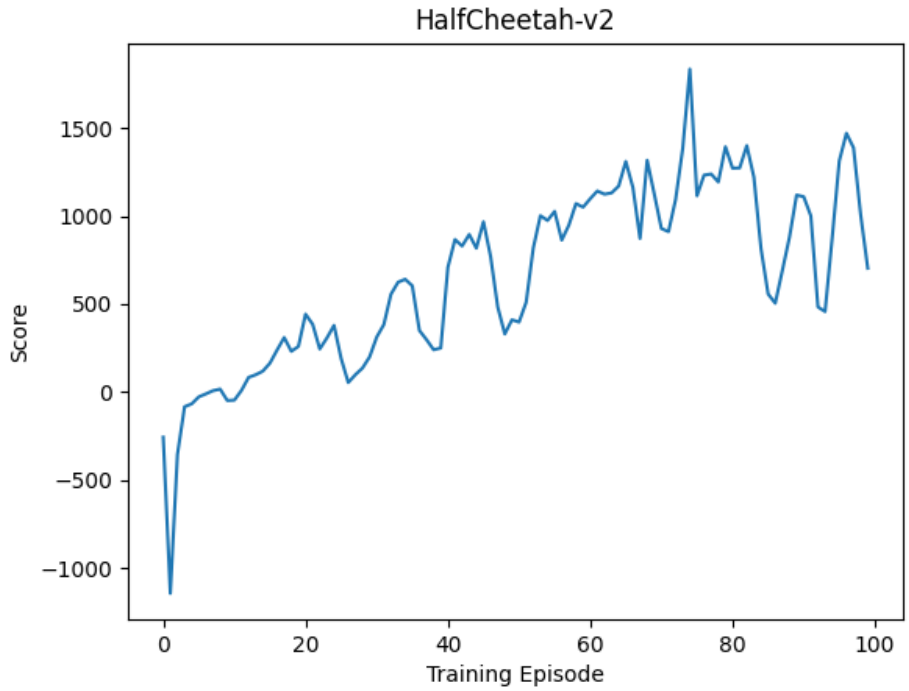

Figure 13: Cheetah // No Baseline // 20

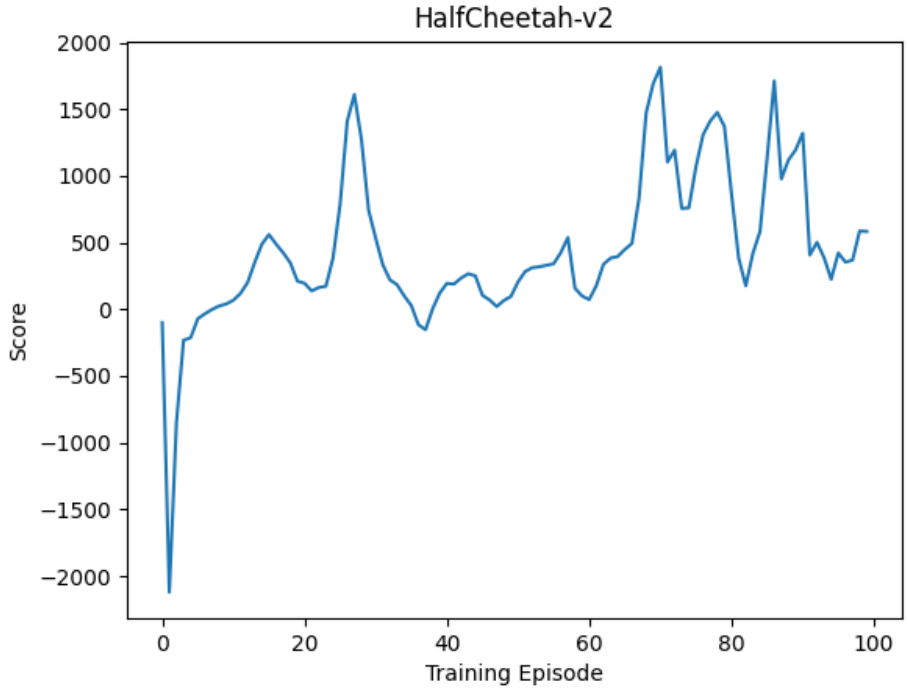

Figure 14: Cheetah // No Baseline // 21

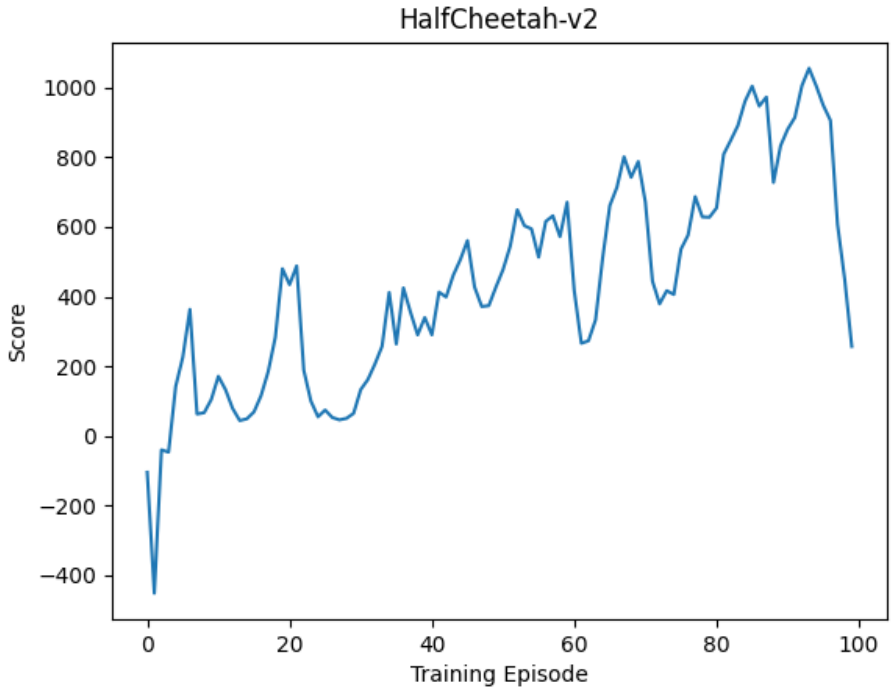

Figure 15: Cheetah // No Baseline // 22

Each of these three runs – even without the baseline – achieved the requisite 200 at some point in their training – in fact, they far exceeded it.

1.9.2 Baseline

Then, for the baseline model, the shell script was:

```
python main.py --env-name "cheetah" --seed 20 --baseline;
python main.py --env-name "cheetah" --seed 21 --baseline;
python main.py --env-name "cheetah" --seed 22 --baseline;
```

The reward plots corresponding to these runs were then (respectively):

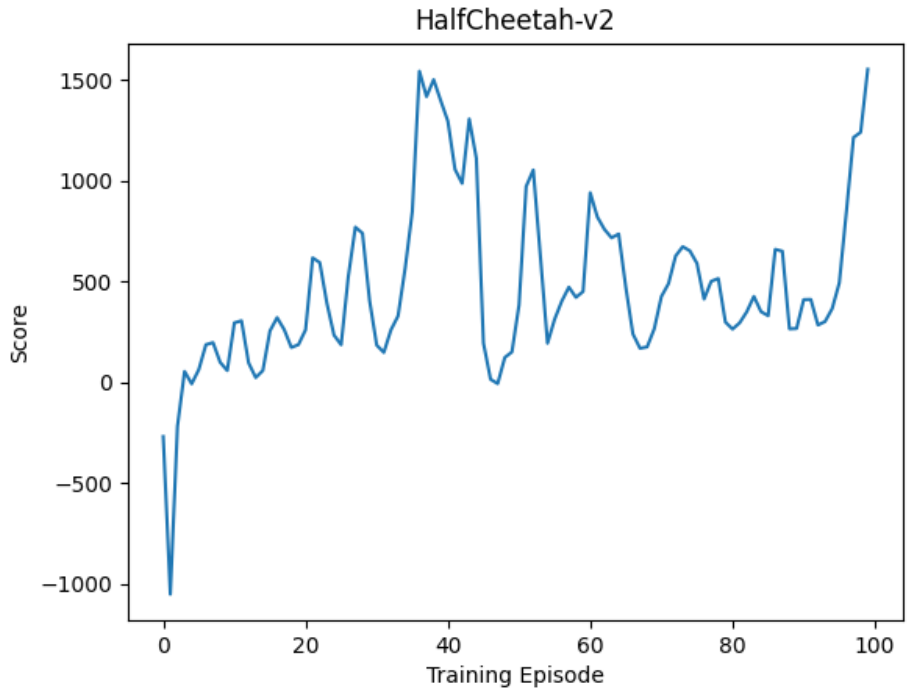

Figure 16: Cheetah // Baseline // 20

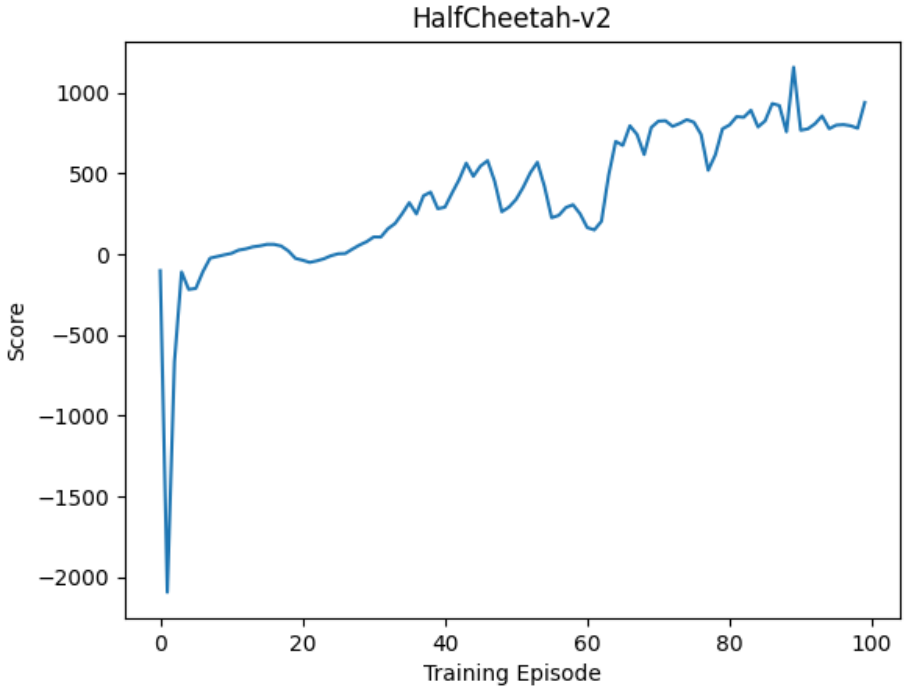

Figure 17: Cheetah // Baseline // 21

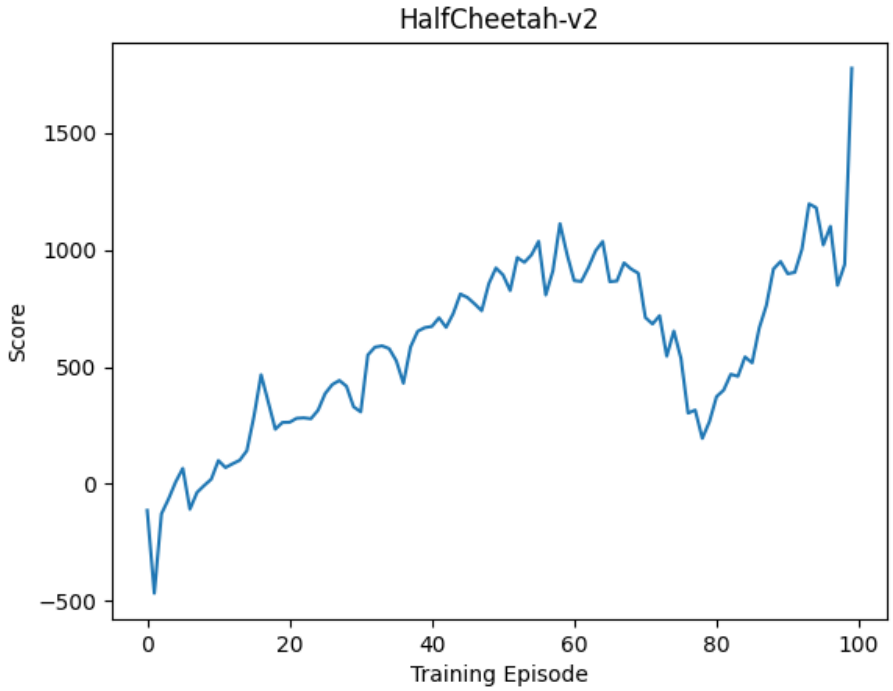

Figure 18: Cheetah // Baseline // 22

This too achieved the requisite 200. And here, it's admittedly hard to judge which one is better: the average max reward of the three no-baseline runs was roughly 1516 (from an approximate $1750 + 1800 + 1000$, while the average max reward of the baseline runs was roughly 1450 (from an approximate $1500 + 1100 + 1750$). So baselining may not have moved the needle much in terms of max reward achieved – however, it is striking that the baseline version was considerably less oscillatory in two of the three runs (21/22), suggesting it contributed some amount of stability, even if performance benefit was close to a wash. Then again, this is only three runs, and to verify any of the aforementioned speculation we would want to replicate over many more seeds.

2 Natural Gradient Methods (20 pts)

Consider a finite MDP given by (S, s_0, A, R, P) where S is a set of states, s_0 is the start state, A is a set of actions, R is a reward function $R : S \times A \to [0, R_{max}]$, and P is the transition model. A policy for this MDP can be characterized as $\pi(a; s, \theta)$ denoting the probability of taking an action a in state s parameterized by $\theta \in \mathbb{R}^{|\theta|}$. The average reward $\eta(\theta)$[1] of this policy is defined in the equation below, where $d^\pi(s)$ is the stationary probability of the policy $\pi(a; s, \theta)$ to be in state s:

$$\eta(\theta) = \sum_{s,a} d^\pi(s)\pi(a; s, \theta)R(s, a) \tag{1}$$

To find a policy parameterized by θ that maximizes the average reward, we can perform gradient ascent on $\eta(\theta)$ wrt. θ. Let's define the gradient of the average reward with respect to θ as:

$$\nabla\eta(\theta) = \sum_{s,a} d^\pi(s)\nabla\pi(a; s, \theta)Q^\pi(s, a) \tag{2}$$

And the gradient ascent step to maximize $\eta(\theta)$ wrt. θ will look like:

$$\theta' \leftarrow \theta + \nabla\eta(\theta) \tag{3}$$

By definition, $\nabla\eta(\theta)$ is the steepest ascent in the parameter space $\mathbb{R}^{|\theta|}$ measured by Euclidean metric. However, in this case, θ parameterizes the distribution $\pi(a; s, \theta)$; therefore, it might be useful to instead ascent in the steepest direction in $\mathbb{R}^{|\theta|}$ measured by divergence between different $\pi(a; s, \theta)$ distributions (e.g. KL-Divergence). We introduce such a method called Natural Gradient which redefines the gradient ascent step as:

$$\theta' \leftarrow \theta + \mathbf{F}^{-1}(\theta)\nabla\eta(\theta) \tag{4}$$

where $\mathbf{F}(\theta)$ is the Fisher Information Matrix for $\pi(a; s, \theta)$:

$$\mathbf{F}(\theta) = \mathbb{E}_{d^\pi(s)}\left[\mathbb{E}_{\pi(a;s,\theta)}\left[\nabla \log \pi(a; s, \theta)\nabla \log \pi(a; s, \theta)^\top\right]\right] \tag{5}$$

(a) (10 pts) In this problem, we will use linear function approximation to approximate $Q^\pi(s, a)$, and show that for a specific linear approximation $\hat{Q}^\pi(s, a; \mathbf{w})$ and particular objective function, the weights $\mathbf{w}^*$ that minimize the objective are precisely the natural gradient of the average reward: $\mathbf{F}^{-1}(\theta)\nabla\eta(\theta)$.

We define our feature vector $\mathbf{x}$ and linear approximation $\hat{Q}$ as follows:

$$\mathbf{x}^\pi(s, a) = \nabla \log \pi(a; s, \theta), \qquad \hat{Q}^\pi(s, a; \mathbf{w}) = \mathbf{w}^\top\mathbf{x}^\pi(s, a)$$

where the i-th element $\mathbf{x}^\pi(s, a)_i = [\nabla \log \pi(a; s, \theta)]_i = \dfrac{\partial \log \pi(a; s, \theta)}{\partial \theta_i}$. Our objective function is

$$J^\pi(\mathbf{w}) = \sum_{s,a} d^\pi(s)\pi(a; s, \theta)(\hat{Q}^\pi(s, a; \mathbf{w}) - Q^\pi(s, a))^2$$

Let $\mathbf{w}^*$ minimize this objective $J^\pi(\mathbf{w})$. Prove that

$$\mathbf{w}^* = \mathbf{F}^{-1}(\theta)\nabla\eta(\theta).$$

[1] We are abusing notation by writing $\eta(\theta)$ instead of $\eta(\pi_\theta)$, since it is really a function on $\{\pi(a; s, \theta) : \theta \in \mathbb{R}^{|\theta|}\}$, but we can ignore this.

If we halve the loss and apply the given equations, we have:

$$\begin{aligned} J^\pi(\mathbf{w}) &= \frac{1}{2}\sum_{s,a} d^\pi(s)\pi(a;s,\theta)(\hat{Q}^\pi(s,a;\mathbf{w}) - Q^\pi(s,a))^2 \\ &= \frac{1}{2}\sum_{s,a} d^\pi(s)\pi(a;s,\theta)(\mathbf{w}^T\mathbf{x}^\pi(s,a) - Q^\pi(s,a))^2 \\ &= \frac{1}{2}\sum_{s,a} d^\pi(s)\pi(a;s,\theta)(\mathbf{w}^T\nabla\log\pi(a;s,\theta) - Q^\pi(s,a))^2 \end{aligned}$$

We have:

$$\begin{aligned} \nabla_w J(w) &= \frac{2}{2}\sum_{s,a} d^\pi(s)\pi(a;s,\theta)\Big(\mathbf{w}^T\nabla\log\pi(a;s,\theta) - Q^\pi(s,a)\Big)\nabla\log\pi(a;s,\theta)^T \\ &= \sum_{s,a} d^\pi(s)\pi(a;s,\theta)\mathbf{w}^T\nabla\log\pi(a;s,\theta)\nabla\log\pi(a;s,\theta)^T \\ &\qquad - \sum_{s,a} d^\pi(s)\pi(a;s,\theta)Q^\pi(s,a)\nabla\log\pi(a;s,\theta)^T \\ &= \mathbf{w}^T\sum_s d^\pi(s)\sum_a \pi(a;s,\theta)\nabla\log\pi(a;s,\theta)\nabla\log\pi(a;s,\theta)^T \\ &\qquad - \sum_{s,a} d^\pi(s)\pi(a;s,\theta)Q^\pi(s,a)\frac{\nabla\pi(a;s,\theta)^T}{\pi(a;s,\theta)} \\ &= \mathbf{w}^T E_{s\sim d^\pi}E_{a\sim\pi(a;s,\theta)}[\nabla\log\pi(a;s,\theta)\nabla\log\pi(a;s,\theta)^T] \\ &\qquad - \sum_{s,a} d^\pi(s)Q^\pi(s,a)\nabla\pi(a;s,\theta)^T \\ &= \mathbf{w}^T F(\theta) - \nabla\eta(\theta)^T. \end{aligned}$$

Here, the moves are justified by: (i) just taking a gradient with a chain rule; (ii) separation of the sum; (iii) extraction of the weights, as well as differentiating a log followed by a chain rule; (iv) definition of expectation and cancellation; (v) substitution of given equations. We then set to zero to solve (the Hessian F is symmetric, which is why it remains the same under the transpose), giving

$$\mathbf{w}^{*T}F(\theta) - \nabla\eta(\theta)^T = 0 \implies F(\theta)\mathbf{w}^* = \nabla\eta(\theta) \implies \mathbf{w}^* = F(\theta)^{-1}\nabla\eta(\theta),$$

as desired.

(b) (10 pts) We have now computed the weights $\mathbf{w}^*$ that minimize our error objective $J^\pi(\mathbf{w})$. We then make our gradient ascent update to our parameter θ:

$$\theta' \leftarrow \theta + \alpha\mathbf{F}^{-1}(\theta)\nabla\eta(\theta)$$

where α is the step size. We will show that updating θ only changes the policy $\pi(a;s,\theta)$ locally, that is, by a factor that depends on our linear approximation $\hat{Q}^\pi(s,a;\mathbf{w}^*)$. Prove

$$\pi(a;s,\theta') = \pi(a;s,\theta)(1 + \alpha\hat{Q}^\pi(s,a;\mathbf{w}^*)) + O(\alpha^2).$$

Hint: Use a Taylor series expansion of $\pi(s,a;\theta')$ centered at $\pi(s,a;\theta)$. The update rule gives

$$\theta' = \theta + \alpha F^{-1}(\theta)\nabla\eta(\theta) = \theta + \alpha\mathbf{w}^* \implies \theta' - \theta = \alpha\mathbf{w}^*.$$

By Taylor, we have

$$\begin{aligned}
\pi(a;s,\theta') &= \pi(a;s,\theta) + (\theta'-\theta)^T\nabla\pi(a;s,\theta) + O((\theta'-\theta)^T(\theta'-\theta)) \\
&= \pi(a;s,\theta) + (\alpha\mathbf{w}^*)^T\nabla\pi(a;s,\theta) + O((\alpha\mathbf{w}^*)^T(\alpha\mathbf{w}^*)) \\
&= \pi(a;s,\theta) + (\alpha\mathbf{w}^*)^T\frac{\pi(a;s,\theta)}{\pi(a;s,\theta)}\nabla\pi(a;s,\theta) + O((\alpha\mathbf{w}^*)^2) \\
&= \pi(a;s,\theta) + \pi(a;s,\theta)(\alpha\mathbf{w}^*)^T\frac{\nabla\pi(a;s,\theta)}{\pi(a;s,\theta)} + O(\alpha^2) \\
&= \pi(a;s,\theta) + \pi(a;s,\theta)(\alpha\mathbf{w}^*)^T\nabla\log\pi(a;s,\theta) + O(\alpha^2) \\
&= \pi(a;s,\theta) + \pi(a;s,\theta)\alpha\hat{Q}^\pi(s,a,\mathbf{w}^*) + O(\alpha^2) \\
&= \pi(a;s,\theta)(1+\alpha\hat{Q}^\pi(s,a,\mathbf{w}^*)) + O(\alpha^2).
\end{aligned}$$

Again, we leverage the fact that

$$\nabla\log\pi(a;s,\theta) = \frac{\nabla\pi(a;s,\theta)}{\pi(a;s,\theta)}.$$

Moreover, we are treating $\mathbf{w}^*$ as invariant w.r.t. θ in the Taylor expansion, the ${\mathbf{w}^*}^T\mathbf{w}^*$ is effectively constant and we may reduce $O((\alpha\mathbf{w}^*)^T(\alpha\mathbf{w}^*))$ to ${\mathbf{w}^*}^T\mathbf{w}^*O(\alpha^2) \approx O(\alpha^2)$.

Natural gradient methods like the ones you have explored in this problem are closely related to modern RL algorithms like Trust Region Policy Optimization (TRPO). TRPO updates policies to take the largest possible step while ensuring the KL-Divergence of the policy update isn't too large. At a high level, this prevents the policy from straying outside of the area where our function approximation is accurate. In part (b), you showed a related result for our natural gradient method: the policy change differs by a factor that depends on our linear approximation.

3 Ethical concerns with Policy Gradients (5 pts)

In this assignment, we focus on policy gradients, an extremely popular and useful model-free technique for RL. However, policy gradients collect data from the environment with a potentially suboptimal policy during the learning process. While this is acceptable in simulators like Mujoco or Atari, such exploration in real world settings such as healthcare and education presents challenges.

Consider a case study of a Stanford CS course considering introducing a RL-based chat bot for office hours. For each assignment, some students will be given 100% human CA office hours; others 100% chatbot; others a mix of both. The reward signal is the student grades on each assignment. Since the AI chatbot will learn through experience, at any given point in the quarter, the help given by the chatbot might be better or worse than the help given by a randomly selected human CA.

If each time students are randomly assigned to each condition, some students will be assigned more chatbot hours and others fewer. In addition, some students will be assigned more chatbot hours at the beginning of the term (when the chatbot has had fewer interactions and may have lower effectiveness) and fewer at the end, and vice versa. All students will be graded according to the same standards, regardless of which type of help they have received.

Researchers who experiment on human subjects are morally responsible for ensuring their well being and protecting them from being harmed by the study. A foundational document in research ethics, the Belmont Report, identifies three core principles of responsible research:

1. **Respect for persons:** individuals are capable of making choices about their own lives on the basis of their personal goals. Research participants should be informed about the study they are considering undergoing, asked for their consent, and not coerced into giving it. Individuals who are less capable of giving informed consent, such as young children, should be protected in other ways.

2. **Beneficence:** the principle of beneficence describes an obligation to ensure the well-being of subjects. It has been summarized as "do not harm" or "maximize possible benefits and minimize possible harms."

3. **Justice:** the principle of justice requires treating all people equally and distributing benefits and harms to them equitably.

(a) **[4 pts]** In 4-6 sentences, describe **two** experimental design or research choices that researchers planning the above experiment ought to make in order to respect these principles. Justify the importance of these choices using one of the three ethical principles above and indicating which principle you have chosen. For example, "Researchers ought to ensure that students advised by the chatbot are able to revise their assignments after submission with the benefit of human advice if needed. If they did not take this precaution, the principle of justice would be violated because the risk of harm from poor advice from the AI chatbot would be distributed unevenly."

First, certain structural decisions already serve the interests of consent and justice, the randomized assignment of who gets automated OH when serves justice, as benefit (no cha OH) and harm (OH) is distributed uniformly. Second, participation in such a study would likely be a requirement for enrollment in the class ("just as you agree to record zoom if you're watching a lecture live"), so in some sense consent will have been addressed. However, the two experimental design choices I would consider would most likely include:

(a) Have instructors maintain constant and scrupulous supervision of the chatbot. This way, if the chatbot is returning erroneous, incomplete, or otherwise misleading information, instructors can quickly intervene, and clarify any matters (for all in the class to see, in the interest of the justice principle) via a public thread on Ed or in-class announcement. In this way, if things go off the rails, it can be caught early, in order to minimize possible harms to students selected to the chatbot (Beneficence).

(b) Since the chatbot must start from scratch and learn only from students and their rewards, perhaps announce after the first one or two HW assignments in which chatbot was used that in fact, those to HW assignments were scored on a participation/ yes-no submission basis (for all students, even those with in-person OH). This way, the chatbot can hopefully get its struggles out of the way early at no real costs to the student. Of course, this approach involves a bit of deception – students originally turn in their first one or two assignments expecting to be graded (so as to incentivize their best effort), only to be told later that it is in fact a boolean score for the assignment. However, I would not consider this deception wildly unethical or even harmful: for one, it guarantees that every students' score improves, and two, instructors routinely change grading/syllabus criteria due to time constraints, external factors, unusually difficult assignments. Note here that this does not the requirement that "all students will be graded according to the same standards," as even those students who received in-person OH office hours will receive the boolean score, thus ensuring a consistent grading schema across students.

Overall, this policy would serve justice, as all would end up being treated equally in terms of grading (i.e. as long as you put in an effort, you wouldn't be penalized for having been assigned/not to the chatbot), but meaningful data could still be collected for the RL experiment.

At universities, research experiments that involve human subjects are subject by federal law to Institutional Review Board (IRB) approval. The purpose of IRB is to protect human subjects of research: to "assure, both in advance and by periodic review, that appropriate steps are taken to protect the rights and welfare of humans participating as subjects in the research" (reference). The IRB process was established in response to abuses of human subjects in the name of medical research performed during WWII (reference). The IRB is primarily intended to address the responsibilities of the researcher towards the subjects. Familiarize yourself with Stanford's IRB Research Compliance process at this link.

(b) **[1 pt]** If you were conducting the above experiment, what process would you need to follow at Stanford (who would you email/ where would you upload a research protocol) to get clearance? My procedure would go as follows:

(a) Follow the decision tree set forth in the "Does my Project Need IRB Review"[2] document to determine whether in fact you need IRB Review.

(b) If the determination is yes, enter your SUNET ID into the protocol application, via the `eprotocol` portal[3]

(c) Register for an eProtocol/CITI training session.

(d) Follow the `Create Protocol > Create IRB Protocol` steps to propose your study.

(e) Enter the names of a Protocol Director, Administrative Contact, and academic sponsor.

(f) Presumably, a university official and/or your sponsors will follow up with you to discuss next steps and viability.

(g) More generally, consult your peers, members of the department, etc., to get a feel for any ethical concerns in the study and how to best address them.

[2] https://stanfordmedicine.box.com/shared/static/vqeuewr5axycjpu8h0wat77vqdpua9ru.pdf
[3] https://eprotocol.stanford.edu/irb/ControlServlet