HW 4

EECE 571N – Sequential Decision-Making (EECE 571N)

Instructor: Cyrus Neary
Due: 2025-12-12 at 23:59 PT

Name: Dominic Klukas **Student Number:** 64348378

Instructions

Submit a single PDF to Canvas. Please include your name and student number at the top of that PDF. For all questions, show your work and clearly justify your steps and thinking. Please feel free to include any code as an attachment at the end of the PDF. State any assumptions. Unless otherwise specified, you may collaborate conceptually but must write up your own solutions independently.

Grading

Points for each part are indicated. The total number of achievable points is 100. Partial credit is available for incorrect answers with clear reasoning.

1. Gradients in Value Function Approximation

As we saw in Lecture 11 (function approximation in RL), we often use the semi-gradient rule when using TD learning with function approximation. Consider the mean squared Bellman error (MSBE) objective for a given policy π:

$$J(\theta) = \mathbb{E}_{(s,r,s')\sim\pi}\left[(r + \gamma V_\theta(s') - V_\theta(s))^2\right].$$

(a) Derive the full gradient $\nabla_\theta J(\theta)$ of the MSBE objective.

(b) Write the semi-gradient TD update rule. Highlight the difference between this update rule and the full gradient.

(c) Why are semi-gradient methods often preferred in practice over full gradient methods, despite being biased with respect to the MSBE?

Solution.

(a) Since the gradient of the expectation is the expectation of the gradient for the nice functions that we are dealing with, we have

$$\begin{aligned}
J(\theta) &= \mathbb{E}_{(s,r,s')\sim\pi}[(r + \gamma V_\theta(s') - V_\theta(s))^2] \\
\nabla_\theta J(\theta) &= \mathbb{E}_{(s,r,s')\sim\pi}[\nabla_\theta(r + \gamma V_\theta(s') - V_\theta(s))^2] \\
&= \mathbb{E}_{(s,r,s')\sim\pi}[2(r + \gamma V_\theta(s') - V_\theta(s))(\gamma\nabla_\theta V_\theta(s') - \nabla_\theta V_\theta(s))].
\end{aligned}$$

(b) The semi-gradient simply ignores the gradient of the backpropagation, so that

$$\nabla_\theta^{\text{semi}} J(\theta) = \mathbb{E}_{(s,r,s')\sim\pi}[2(r + \gamma V_\theta(s') - V_\theta(s))\nabla_\theta V_\theta(s)].$$

(c) Although they are biased, they are more stable than the full gradient, they are more efficient (it is quicker to compute), and also more stable. In a sense, we are comparing the value $V(s)$ with our approximation of the updated value of $V(s)$, that being $R + V(s')$. Intuitively, if we change $V(s')$ at the same time as $V(s)$, we are "moving our target" for $V(s)$, which can create feedback loops, and divergence. Therefore, the semi-gradient method is more stable.

2. The Deadly Triad

(a) Describe the three points of the "deadly triad" in deep RL. What does the deadly triad tell us about the fundamental limitations of deep RL algorithms?

(b) Describe in your own words how these three elements interact to cause instability in deep learning algorithms.

(c) Modern algorithms that we've seen in class (like DQN, DDPG, and TD3) employ all three elements. Discuss how the techniques used in these papers mitigate the risks associated with the deadly triad and stabilize training.

Solution.

(a) The three points of the deadly triad are

- Function Approximation
- Bootstrapping
- Off-policy learning

The deadly triad is a qualitative rule that says that any algorithm which makes uses all of these at the same time will be unstable. The fundamental limitation of RL algorithms, is that practically, they NEED all three of these to work. On-policy learning is too time consuming, without function approximation we can't deal with continuous or large state spaces, and without bootstrapping learning is again, too slow.

(b) Function approximation and Bootstrapping both have the effect that the value at one state affects nearby states as well. This can create feedback loops, for example when the bootstrapped value increases when you increase a state's value. Finally, the importance-sampling coefficient in off-policy learning can be much greater than 1 in states where the policies differ, which can magnify feedback loops. Also, off-policy learning will sample states that the current policy will not visit as much which can cause problems (such as we see in Baird's counterexample).

(c) The algorithms we have seen in class use a number of techniques to increase stability and reduce feedback loops.

- Using two value networks instead of one, and using the bellman backup from one value function to update the other value function. This then means that any feedback from function approximation doesn't carry over from one state to another, but also has similar benefits to Double Q learning, where the policy's tendency to favor actions that it by random chance has deemed more valuable than it should is reduced. Some policies also take the minimum of the two value functions when updating policies, to prevent overestimation bias which can result in positive feedback loops and divergence.
- Next, we use the Polyak average for the networks that are being used to evaluate policies, or compute backups. This damps feedback loops by preventing recent "good" information

from having too big of an effect on the target network.

- We update policy networks less frequently than value networks, so that the estimated values for the value networks are yield better approximations for policy network updates. This also prevents erronious feedback loops.
- We use a replay buffer, which contains previous transitions. Although not exactly on policy, the buffer fills up with transitions that are recent enough that they were collected with a policy that is similar to the one we are updating.
- We limit how much a policy/value function can change in one iteration, through policy gradient methods, and PPO optimation.

3. The Discounted Policy Gradient Theorem

In Lecture 12, we proved the undiscounted, episodic case of the policy gradient theorem.

(a) State the undiscounted, episodic case of the policy gradient theorem from class. Describe in your own words the distribution that is used to define the policy gradient. Why is this theorem practically useful in RL algorithms?

(b) Let $d_\pi^\gamma(s)$ denote the discounted occupancy measure,

$$d_\pi^\gamma(s) = (1-\gamma)\sum_{s_0} \rho(s_0) \sum_{k=0}^{\infty} \gamma^k \Pr_\pi(S_k = s \mid S_0 = s_0),$$

where $\Pr_\pi(S_k = s \mid S_0 = s_0)$ is the probability of being in state s at time k given policy π is followed from initial state s_0, and ρ is the initial state distribution. Show that the policy gradient in the discounted episodic case may be written as

$$\nabla_\theta J(\theta) = \frac{1}{1-\gamma} \sum_{s\in\mathcal{S}} d_{\pi_\theta}^\gamma(s) \sum_{a\in\mathcal{A}} Q(s,a)\, \nabla_\theta \pi_\theta(a \mid s).$$

Please do not skip steps just because we did them in class. Show that you understood the logic by following it through yourself. (Hint: Follow the steps used in the undiscounted case while keeping track of γ.)

Solution.

(a) The quantity that we are trying to comptute is

$$\begin{aligned}\nabla J(\theta) &= \nabla \mathbb{E}\left[V^\pi(s_0) | s_0 \sim \mu\right] \\ &= \nabla \sum_\tau \varphi(s_0) \prod_{t=0}^{\tau-1} \pi_\theta(a_t|s_t) T(s_{t+1}|s_t, a_t).\end{aligned}$$

This is intractable to compute. Let

$$\eta(s) = \sum_{s_0 \in S} \varphi(S_0) \sum_{K=0}^{\infty} \Pr_{\pi_\theta}(s_0 \to s, k)$$

be the expected number of visits to state s during an episode starting from initial distribution $s_0 \sim \varphi$ and following policy π_θ. Then, let $\mu(s) = \frac{\eta(s)}{\sum_{s' \in S} \eta(s')}$. The policy gradient theorem says that

$$\nabla J(\theta) \propto \sum_{s \in S} \mu(s) \sum_a Q^{\pi_\theta}(s,a) \nabla \pi_\theta(a|s).$$

Now, $\mu(s)$ is just the expected visitation frequency over an episode induced by the policy π_θ: this is precisely why the theorem is useful, because by simply running the RL policy in the environment from initial states $s_0 \sim \varphi$, we will naturally collect samples (s,a,r) where s follows distribution $\mu(s)$.

This becomes especially useful if we continue:

$$\begin{aligned}\sum_{s \in S} \mu(s) \sum_a Q^{\pi_\theta}(s,a) \nabla \pi_\theta(a|s) &= \mathbb{E}_{S \sim \mu}\left[\sum_{a \in A} Q^{\pi_\theta}(S,a) \nabla_{\pi_\theta}(a|S)\right] \\ &= \mathbb{E}_{S \sim \mu}\left[\sum_{a \in A} \pi_\theta(a|S) Q^{\pi_\theta}(S,a) \nabla \ln(\pi_\theta(a|S))\right] \\ &= \mathbb{E}_{S \sim \mu, A \sim \pi_\theta(\cdot|S)}\left[Q^{\pi_\theta}(S,A) \nabla \ln(\pi_\theta(A|S))\right].\end{aligned}$$

Then, simply by sampling states and samples from regular trajectories, we will have an unbiased approximation of $S \sim \mu$ and $A \sim \pi_\theta(\cdot|S)$.

(b) We start with $J(\theta) = \mathbb{E}\left[V^{\pi_\theta}(s_0)|s_0 \sim \varphi\right]$. We are seeking to compute $\nabla J(\theta)$. Then, $\nabla J(\theta) = \nabla \mathbb{E}_{s_0 \sim \varphi}[V^{\pi_\theta}(s_0)] = \sum_{s_0 \in S} \varphi(s_0) \nabla V^{\pi_\theta}(s_0)$. Then,

$$\begin{aligned}\nabla V^{\pi_\theta}(s) &= \nabla\left[\sum_a \pi_\theta(a|s) Q^{\pi_\theta}(s,a)\right] \\ &= \sum_a \left\{Q^{\pi_\theta}(s,a) \nabla \pi_\theta(a|s) + \nabla Q^{\pi_\theta}(s,a) \pi_\theta(a|s)\right\} \\ &= \sum_a \left\{Q^{\pi_\theta}(s,a) \nabla \pi_\theta(a|s) + \pi_\theta(a|s) \nabla \sum_{s'} T(s'|s,a)\left(r + \gamma V^{\pi_\theta}(s')\right)\right\} \\ &= \sum_a \left\{Q^{\pi_\theta}(s,a) \nabla \pi_\theta(a|s) + \pi_\theta(a|s) \sum_{s'} T(s'|s,a)\left(\cancelto{0}{\nabla r} + \nabla \gamma V^{\pi_\theta}(s')\right)\right\} \\ &= \sum_a \left\{Q^{\pi_\theta}(s,a) \nabla \pi_\theta(a|s) + \gamma \pi_\theta(a|s) \sum_{s',r} T(s'|s,a) \nabla V^{\pi_\theta}(s')\right\}.\end{aligned}$$

At this point, we define $\sigma(s) = \sum_a Q^{\pi_\theta}(s,a) \nabla \pi_\theta(a|s)$ for notational convenience. Then,

$$\begin{aligned}\nabla V^{\pi_\theta}(s) &= \sigma(s) + \gamma \sum_a \sum_{s'} \pi_\theta(a|s) T(s'|s,a) \nabla V^{\pi_\theta}(s') \\ &= \sigma(s) + \gamma \sum_a \pi_\theta(a|s) \sum_{s'} T(s'|s,a) \Big\{ \\ &\qquad \sigma(s') + \gamma \sum_{a'} \sum_{s''} \pi_\theta(a'|s') T(s''|s',a') \nabla V^{\pi_\theta}(s'') \Big\}.\end{aligned}$$

Next, we note:

$$\begin{aligned}\sum_a \sum_{s'} \pi_\theta(a|s) T(s'|s,a) &= \sum_{s'} \sum_a \pi_\theta(a|s) T(s'|s,a) \\ &= \sum_{s'} \Pr_{\pi_\theta}(s \to s', 1),\end{aligned}$$

that is, the probability of transitioning from s to s' in 1 step under policy π_θ. Then, we have:

$$\begin{aligned}\nabla V^{\pi_\theta}(s) &= \sigma(s) + \gamma \sum_a \sum_{s'} \pi_\theta(a|s) T(s'|s,a) \left\{ \sigma(s') + \gamma \sum_{a'} \sum_{s''} \pi_\theta(a'|s') T(s''|s',a') \nabla V^{\pi_\theta}(s'') \right\} \\ &= \sigma(s) + \gamma \sum_{s'} \Pr_{\pi_\theta}(s \to s', 1) \left\{ \sigma(s') + \gamma \sum_{a'} \sum_{s''} \pi_\theta(a'|s') T(s''|s',a') \nabla V^{\pi_\theta}(s'') \right\} \\ &= \sigma(s) + \gamma \sum_{s'} \Pr_{\pi_\theta}(s \to s', 1) \sigma(s') + \gamma^2 \sum_{s'} \sum_{s''} \Pr_{\pi_\theta}(s \to s', 1) \Pr_{\pi_\theta}(s' \to s'', 1) \nabla V^{\pi_\theta}(s'').\end{aligned}$$

However, we recognize from Markov chain theory that $\sum_{s'} \Pr_{\pi_\theta}(s \to s', 1) \Pr_{\pi_\theta}(s' \to s'', 1) = \Pr_{\pi_\theta}(s \to s'', 2)$. Indeed, in general we have $\sum_{s'} \Pr_{\pi_\theta}(s \to s', 1) \Pr_{\pi_\theta}(s' \to s'', k) = \Pr_{\pi_\theta}(s \to s'', k+1)$. So then, can simplify, and continue:

$$\begin{aligned}\nabla V^{\pi}(s) &= \sigma(s) + \gamma \sum_{s'} \Pr_{\pi_\theta}(s \to s', 1) \sigma(s') + \gamma^2 \sum_{s'} \Pr_{\pi_\theta}(s \to s', 2) \sigma(s') \\ &\quad + \gamma^3 \sum_{s'} \Pr_{\pi_\theta}(s \to s', 3) \sigma(s') + \dots \\ &= \sum_{s'} \sum_{k=0}^{\infty} \gamma^k \Pr_{\pi_\theta}(s \to s', k) \sigma(s') \\ &= \sum_{s'} \sum_{k=0}^{\infty} \gamma^k \Pr_{\pi_\theta}(s \to s', k) \sum_a Q^{\pi_\theta}(s', a) \nabla \pi_\theta(a|s').\end{aligned}$$

Where in the last line we used the notation $\Pr_{\pi_\theta}(s \to s', 0) = \begin{cases} 1 & \text{if } s' = s \\ 0 & \text{else} \end{cases}$. But then,

$$\begin{aligned}\nabla J(\theta) &= \sum_{s_0 \in S} \rho(s_0) \nabla V^{\pi_\theta}(s_0) \\ &= \sum_{s_0 \in S} \rho(s_0) \sum_{s \in S} \sum_{k=0}^{\infty} \gamma^k \Pr_{\pi_\theta}(s_0 \to s, k) \sum_a Q^{\pi_\theta}(s, a) \nabla \pi_\theta(a|s) \\ &= \sum_{s \in S} \left(\sum_a Q^{\pi_\theta}(s, a) \nabla \pi_\theta(a, s) \right) \frac{(1-\gamma)}{(1-\gamma)} \sum_{s_0 \in S} \rho(s_0) \sum_{k=0}^{\infty} \gamma^k \Pr_{\pi_\theta}(s_0 \to s, k).\end{aligned}$$

Next, we introduce $d_\pi^\gamma(s) = (1-\gamma) \sum_{s_0 \in S} \rho(s_0) \sum_{k=0}^{\infty} \gamma^k \Pr_\pi(s_0 \to s, k)$, so that we can simplify the expression:

$$\nabla J(\theta) = \frac{1}{1-\gamma} \sum_{s \in S} d_{\pi_\theta}^\gamma \sum_{a \in \mathcal{A}} Q^{\pi_\theta}(s, a) \nabla \pi_\theta(a, s),$$

as desired.

4. Running the TD3 Algorithm

Run the TD3 algorithm on the `HalfCheetah-v4` environment by editing and running the corresponding Python file in the CleanRL repository. Make the following changes in the file before running it:

- `env_id: str = "HalfCheetah-v4"`
- `total_timesteps: int = 500000`
- `track: bool = True`
- `capture_video: bool = True` (optional)

Run the code and include the `charts/episodic_return` plot from the Weights & Biases project after training completes.

5. Ablating the TD3 Replay Buffer

Deep RL algorithms assume that samples used for training are i.i.d., but data collected by an agent is sequential and correlated. Replay buffers break this correlation by shuffling experiences.

(a) Modify the TD3 argument `buffer_size` so that it equals the batch size (256).

(b) Compare the `episodic_return` against the baseline, as well as the `qf1_value` plot.

(c) Comment on what you observe. Does the algorithm fail to learn entirely? Why does performance change when the replay buffer is removed?

6. Ablating the TD3 Target Networks

TD3 uses Polyak averaging to update parameters of the target actor $\bar{\mu}_\theta$ and target critics $\bar{Q}_{w_1}, \bar{Q}_{w_2}$:

$$\theta_{\text{target}} \leftarrow \tau\theta + (1-\tau)\theta_{\text{target}}.$$

(a) Remove the target network mechanism by setting $\tau = 1$, making all target networks identical to the main networks.

(b) Compare the `episodic_return` against the baseline, as well as the `qf_loss` plot.

(c) Comment on what you observe. Does the algorithm fail to learn entirely? Why does performance change when target networks are removed?

7. Ablating the Twin Critic: Overestimation Bias

TD3 uses two critics Q_1, Q_2 and computes the target

$$y = r + \gamma \min_{i=1,2} Q_i(s,a)$$

to reduce overestimation.

(a) Modify `td3_continuous_action.py` to disable clipped double Q-learning by using only the first target critic Q_1 when computing the target.

(b) Include plots comparing the `episodic_return` and `qf1_values` of this modified version with the baseline.

(c) What do you observe? Does the ablation lead to lower returns? Does it show evidence of value overestimation?

(d) Explain why taking the minimum of two independent estimators reduces value overestimation.