CS 277 (W26): Control and Reinforcement Learning
Exercise 3

Due date: Tuesday, February 17, 2026 (Pacific Time)

Roy Fox
https://royf.org/crs/CS277/W26

Instructions: In theory questions, a formal proof is not needed (unless specified otherwise); instead, briefly explain informally the reasoning behind your answers. In practice questions, include a printout of your code as a page in your PDF, and a screenshot of TensorBoard learning curves (`episode_reward_mean`, unless specified otherwise) as another page.

Part 1 Memory-based control (30 points + 10 bonus)

Consider a memory-based policy π that induces a memory state update distribution $\pi(m_t|m_{t-1}, a_{t-1}, o_t)$, giving the probability of updating from internal state m_{t-1}, upon taking a_{t-1} and observing o_t, to m_t; and an action distribution $\pi(a_t|m_t)$. For simplicity, assume m_t always explicitly says what t is.

Question 1.1 (10 points) Given a POMDP with dynamics $p(s_{t+1}|s_t, a_t)$ and observation model $p(o_t|s_t)$, and a policy π, write down a forward recursion for computing the joint distribution of s_t and m_t. That is, show how to compute $p_\pi(s_{t+1}, m_{t+1})$ using p, π, and $p_\pi(s_t, m_t)$.

Question 1.2 (10 points) Given also a reward function $r(s_t, a_t)$, write down a backward recursion for evaluating $V_\pi(s_t, m_t) = \mathbb{E}[R_{\geq t}|s_t, m_t]$. That is, show how to compute $V_\pi(s_t, m_t)$ using p, π, r, and $V_\pi(s_{t+1}, m_{t+1})$.

Question 1.3 (5 points) Given the joint distribution $p_\pi(s_t, m_t)$ from above, show how to compute the belief $b_t^m = p_\pi(s_t|m_t)$. Is this the same as the Bayesian belief $b_t^m \stackrel{?}{=} p(s_t|h_t)$? Justify briefly.

Question 1.4 (5 points) Given the joint value function $V_\pi(s_t, m_t)$ from above, show how to compute the expected memory-based value function $V_\pi(m_t) = \mathbb{E}[R_{\geq t}|m_t]$.

Question 1.5 (bonus 10 points) Is it possible to write a backward recursion for $V_\pi(m_t)$ using p, π, r, and $V_\pi(m_{t+1})$? Justify briefly.

Part 2 Actor–Critic Policy Gradient (35 points)

In this part you'll implement an Actor–Critic Policy-Gradient algorithm. Download and read the code at https://royf.org/crs/CS277/W26/CS277E3.zip. Each part asks you to complete a code placeholder in file `a2c.py`.

Question 2.1 (10 points) Complete the placeholders marked as `Part 2.1` by writing PyTorch code that calculates the actor loss. The actor loss is a policy-gradient loss with pre-computed advantage estimates (`advantage`) plus a negative-entropy loss on the actor policy, weighted by `entropy_loss_coeff` (i.e. a slight push to *maximize* entropy).
Hint: You might want to use Distribution.entropy to compute the entropy.

Question 2.2 (10 points) Complete the placeholders marked as `Part 2.2` by writing PyTorch code that calculates the critic loss. The critic loss is a temporal-difference loss, the square error between the pre-computed value targets and the critic values.
In the function `update`, `traj` is part of a single trajectory, but in this assignment we will **not** assume that it's the entire episode. The batch contains tuples $(s_t, a_t, r_t, s'_t, \text{done}_t, \log \pi(a_t|s_t))$ for some consecutive steps $t \in \{t_1, \ldots, t_2\}$ in a trajectory.
Useful: (a) `Actor.critic`, a function that gets an array of observations and returns a same-size tensor of value predictions; (b) `dones`, a boolean array indicating episode termination in each time step (think: why is this useful here?); and (c) make sure to use `detach()` on tensors that are supposed to be the target.

Question 2.3 (10 points) Complete the placeholders marked as `Part 2.3` by writing PyTorch code that calculates for each step the discounted one-step advantages for the actor's policy gradient.
Hint: advantage should `detach()`.

Question 2.4 (5 points) Run your code on the `CartPole-v1` environment for 1,000,000 time steps and report the results.

```
python run.py --training-steps 1000000 \
              --env CartPole-v1
```

Part 3 Generalized Advantage Estimation (35 points)

Recall the definition of the GAE[1] as

$$A^{\lambda}(s_t, a_t) = \sum_{\Delta t} (\lambda\gamma)^{\Delta t} A(s_{t+\Delta t}, a_{t+\Delta t}).$$

Question 3.1 (10 points) Write down a mathematical expression for the advantage estimate $A^{\lambda}(s_t, a_t)$ using the rewards $r_t, r_{t+1}, \ldots$ and the value estimates $V_\phi(s_t), V_\phi(s_{t+1}), \ldots$. You can use the one we saw in class as a starting point, but in your expression each $V_\phi(s_{t'})$ must appear once.

Question 3.2 (10 points) Complete the placeholders marked as `Part 3.2` by using A^{λ} as the advantage estimates.

Question 3.3 (5 points) Run your code on `CartPole-v1` with a variety of λ values. To train the agent with GAE use

```
python run.py --training-steps 1000000 \
```

[1] https://arxiv.org/abs/1506.02438

```
                --env CartPole-v1 \
                --GAE \
                --_lambda <lambda>
```

Visualize the results in TensorBoard, and attach the resulting plots.

Question 3.4 (10 points) Briefly discuss the results, including:

- What was the best value of λ in your experiments?
- What happens as $\lambda \to 0$?
- What happens as $\lambda \to 1$ in theory? What happens in practice?