CS 446 / ECE 449 — Homework 6

your NetID here

Version 1.1

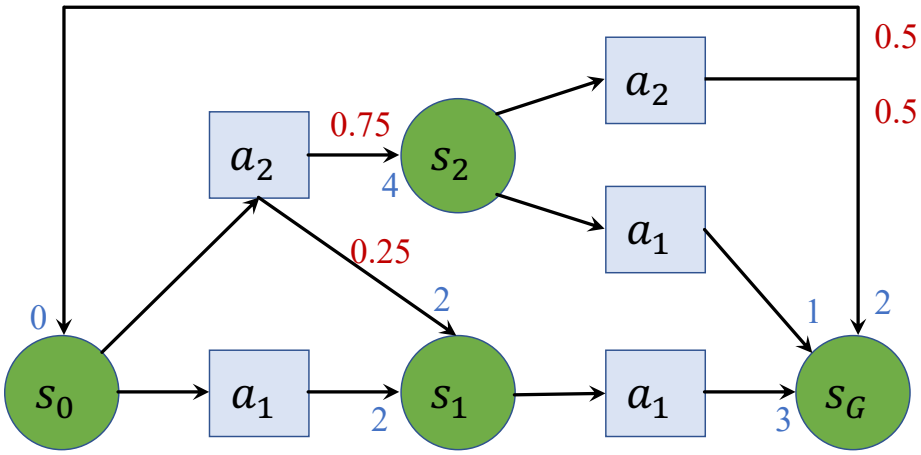
Instructions.

- Homework is due **Tuesday, April 27th, at noon CST**; no late homework accepted.
- Everyone must submit individually at gradescope under `hw6` and `hw6code`.
- The "written" submission at `hw6` **must be typed**, and submitted in any format gradescope accepts (to be safe, submit a PDF). You may use LaTeX, markdown, google docs, MS word, whatever you like; but it must be typed!
- When submitting at `hw6`, gradescope will ask you to mark out boxes around each of your answers; please do this precisely!
- Please make sure your NetID is clear and large on the first page of the homework.
- Your solution **must** be written in your own words. Please see the course webpage for full academic integrity information. Briefly, you may have high-level discussions with at most 3 classmates, whose NetIDs you should place on the first page of your solutions, and you should cite any external reference you use; despite all this, your solution must be written in your own words.
- We reserve the right to reduce the auto-graded score for `hw6code` if we detect funny business (e.g., your solution lacks any algorithm and hard-codes answers you obtained from someone else, or simply via trial-and-error with the autograder).
- When submitting to `hw6code`, only upload `hw6_q_learning.py` and `hw6_reinforce.py`. Additional files will be ignored.

Version 1.1: clean up notations of Problem 2.

1. Markov Decision Process [Written]

Consider the MDP presented in the figure below. States $\{s_0, s_1, s_2, s_G\}$ are represented by green circles. Actions $\{a_1, a_2\}$ are presented by blue squares. Transition probabilities are marked in red and rewards in blue.



(a) Explain the difference between a deterministic and a stochastic Markov Decision Process. Is the given MDP stochastic or deterministic? Explain your answer.

(b) What are three mechanisms to find the optimal policy π^* for a given MDP?

(c) For the policy $\pi(s_0) = a_1$, $\pi(s_1) = a_1$, what is the policy graph and the resulting value functions $V^\pi(s_1)$ and $V^\pi(s_0)$?

(d) For the policy $\pi(s_0) = a_2, \pi(s_2) = a_1$, what is the policy graph and the resulting value functions $V^\pi(s_2)$, $V^\pi(s_1)$ and $V^\pi(s_0)$?

(e) For the policy $\pi(s_0) = a_2$, $\pi(s_2) = a_2$, what is the policy graph and the resulting value functions $V^\pi(s_2)$, $V^\pi(s_1)$ and $V^\pi(s_0)$?

(f) What is the optimal policy for the MDP given in the figure above? Briefly explain your answer.

Solution.

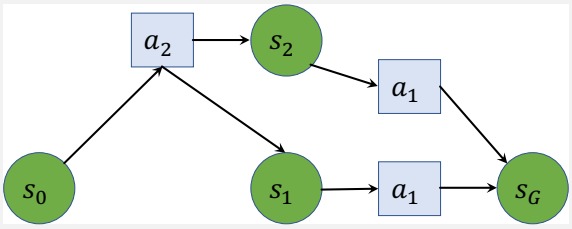
(a) **deterministic:** performing an action always results in a deterministic transition; **stochastic:** performing an action may lead the agent to one of a few possible states; the given MDP is stochastic.

(b) (1) Exhaustive search, (2) policy iteration, (3) value iteration

(c) Policy graph:

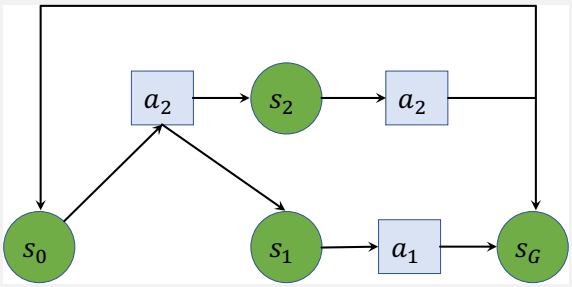
$V^\pi(s_1) = 3$, $V^\pi(s_0) = 5$

(d) Policy graph:



$V^\pi(s_1) = 3$, $V^\pi(s_2) = 1$, $V^\pi(s_0) = 5$

(e) Policy graph:



$V^\pi(s_1) = 3$, $V^\pi(s_2) = 5$, $V^\pi(s_0) = 8$

(f) $\pi^*(s_0) = a_2$, $\pi^*(s_2) = a_2$; Policy with highest $V^\pi(s_0)$ from part (c), (d), and (e).

2. Q-Learning [Written]

(a) State the Bellman optimality principle as a function of the optimal Q-function $Q^*(s,a)$, the reward function r and the transition probability $P(s'|s,a)$, where s is the current state, s' is the next state and a is the action taken in state s.

(b) In case the transition probability $P(s'|s,a)$ in previous question are unknown, a stochastic approach is used to approximate the optimal Q-function. After observing a transition of the form (s,a,r,s'), write down the update of the Q-function at the observed state-action pair (s,a) as a function of the reward r, learning rate α, the discount factor γ, $Q(s,a)$ and $Q(s',a')$.

(c) What is the advantage of an ϵ-greedy strategy?
Remark: ϵ-greedy strategy chooses random action with probability ϵ while using action with maximum Q value otherwise.

(d) What is the advantage of using a replay-memory?

(e) Consider a system with two states s_1 and s_2 and two actions a_1 and a_2. You perform actions and observe the rewards and transitions listed below. Each step lists the current state, action, resulting transition, and reward as: $s_i; a_k : s_i \to s_j; r_i = X$; where X is some value. Perform Q-learning using a learning rate of $\alpha = 0.5$ and a discount factor of $\gamma = 0.5$ for each step by applying the formula from part (b). The Q-table entries are initialized to zero. Fill in the tables below corresponding to the following four transitions. What is the optimal policy after having observed the four transitions?

Q	s_1	s_2
a_1	.	.
a_2	.	.

i. s_1; $a_1 : s_1 \to s_1$; $r = -10$;
ii. s_1; $a_2 : s_1 \to s_2$; $r = -10$;
iii. s_2; $a_1 : s_2 \to s_1$; $r = 18.5$;
iv. s_1; $a_2 : s_1 \to s_2$; $r = -10$;

Remark: you need to provide Q-table at every time step. In total, you need to have four tables.

Solution.

(a)

$$Q^* = \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[r + \max_{a'} Q^*(s',a') \right] \text{ (or use R(s,a,s') instead of } r)$$

(b)

$$y = r + \gamma \max_{a'} Q(s',a')$$
$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha y$$

(c) Trade-off between exploration and exploitation. The best long-term strategy may involve short-term sacrifies resulting in not taking the best action at the beginning of the train to better explore the environment.

(d) Learning from batches of consecutive samples is problematic, as the samples are correlated. This can lead to inefficient learning. For example, if maximizing action is to move left, training samples will be dominated by samples from left-hand size. Instead, a replay memory is used to store the transitions (s_t, a_t, r_t, s_{t+1}) as game episodes are played. The Q-network

is then trained on random minibatches of transitions from the replay memory, instead of consecutive samples.

(e) i.

Q	S_1	S_2
a_1	-5	.
a_2	.	.

ii.

Q	S_1	S_2
a_1	-5	.
a_2	-5	.

iii.

Q	S_1	S_2
a_1	-5	8
a_2	-5	.

iv.

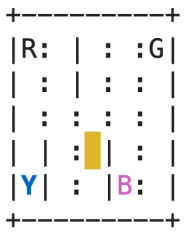
Q	S_1	S_2
a_1	-5	8
a_2	-5.5	.

Optimal policy: $\pi(a_1|S_1) = 1.0, \pi(a_1|S_2) = 1.0$

3. Q-Learning [Coding]

In this problem, you need to implement a *tabular* Q-learning algorithm to train an agent to play the game `Taxi-v3`[1]. The game is shown in figure below. Your agent should be performing well after around 1000 episodes.

You can install gym via `pip install gym`.



Please take a look at the game definition page to understand how to play the game[2]. Essentially, there is a single passenger needing taxi delivery service. The agent controls the taxi and needs to pick up and drop off that passenger at designate locations.

- There are six possible actions: `move south/north/east/west`, `pick up passenger`, `and drop off passenger`.
- There are four possible locations that passenger may appear, namely `R`, `G`, `Y`, and `B`.
- There are four possible destinations, namely `R`, `G`, `Y`, and `B`. Destination and pick-up location will not be the same.
- Every episode, colored texts indicate the *actual* pick-up and drop-off locations. For example, in the snapshot, the agent (yellow rectangle) needs to pick up passenger at blue text (`Y`) and then drive to pink text (`B`) to drop off. Note, colors are only used for visualizations. When coding, you do not need to deal with colors since all states are encoded with numerics.
- Reward settings:
  - successfully deliver the passenger: +20;
  - incorrect pick-up or drop-off: -10;
  - to encourage fast task completion, there is a reward of -1 for each step.

(a) Implement action selection in evaluation mode. Corresponding function: `_act_eval` of class `Agent`.

(b) Implement ϵ-greedy exploration strategy for action selection in training mode. Corresponding function: `_act_train` of class `Agent`.

Hint: you may find `env.action_space.sample()` useful.

(c) Implement Q-value update taught in lectures in function `update` of class `Agent`. You should update `q_table` of `Agent`.

(d) Plot the training curve. By default, the script will save a `train_moving_avg.png`. Attach it here.

(e) Attach the episode generated during evaluation to show the qualitative performance of your trained agent. You can just take a series of snapshots printed by the script.

Solution.

[1] `https://gym.openai.com/envs/Taxi-v3/`

[2] `https://github.com/openai/gym/blob/master/gym/envs/toy_text/taxi.py`

- (a) - (d)

```
class Agent:
    def __init__(self, state_space, action_space):
        """Initialize table for Q-value to all zeros.
        Please make sure the table has the shape [n_states, n_actions].
        """
        self._state_space = state_space
        self._action_space = action_space
        self.q_table = np.zeros([self._state_space.n, self._action_space.n])

    def act(self, state, epsilon, train=False):
        if train:
            return self._act_train(state, epsilon)
        else:
            return self._act_eval(state, epsilon)

    def _act_train(self, state, epsilon):
        """Implement epsilon-greedy strategy for action selection in training.
        """
        if random.uniform(0, 1) < epsilon:
            action = self._action_space.sample() # Explore action space
        else:
            action = np.argmax(self.q_table[state]) # Exploit learned values
        return action

    def _act_eval(self, state, epsilon):
        """Implement action selection in evaluation.
        """
        return np.argmax(self.q_table[state])

    def update(self, state, action, reward, next_state, alpha, gamma):
        """Implement Q-value table update here.
        """
        old_value = self.q_table[state, action]
        next_max = np.max(self.q_table[next_state])

        new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next_max)
        self.q_table[state, action] = new_value
```

- (e)

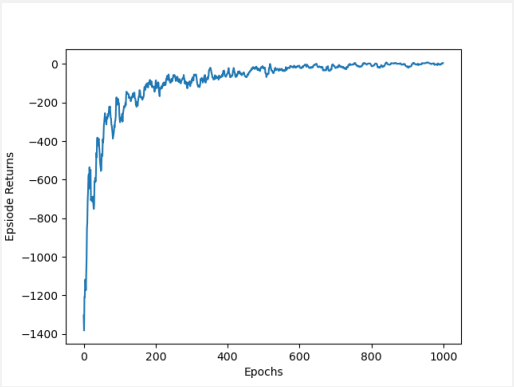


Figure 1: Training curve.

- (f) Read column by column.

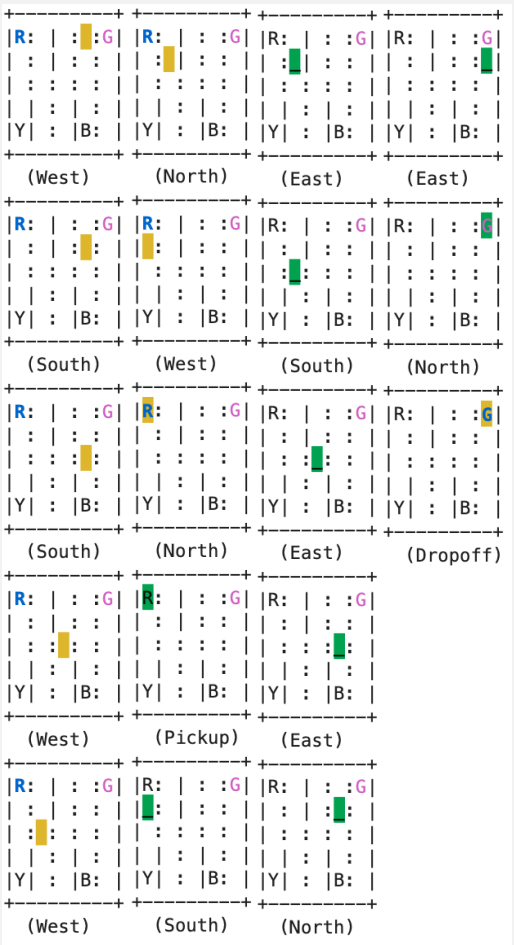


Figure 2: Qualitative.

4. REINFORCE [Written + Coding]

In this problem we investigate the difference between maximum likelihood estimation (MLE) and reinforcement learning.

We are given a utility $U(\theta) = \mathbb{E}_{p_\theta}[R(y)] = \sum_{y\in\mathcal{Y}} p_\theta(y)R(y)$ which is the expected value of the (potentially) non-differentiable reward $R(y)$ defined over a discrete domain $y \in \mathcal{Y} = \{1, \dots, |\mathcal{Y}|\}$. Our goal is to learn the parameters θ of a probability distribution $p_\theta(y)$ so as to obtain a high utility (high expected reward), *i.e.*, we want to find $\theta^* = \arg\max_\theta U(\theta)$. To this end we define the probability distribution to read

$$p_\theta(y) = \frac{\exp F_\theta(y)}{\sum_{\hat{y}\in\mathcal{Y}} \exp F_\theta(\hat{y})}, \tag{1}$$

where, $F_\theta(y) \in \mathbb{R}$ is a scoring function.

(a) If we are given a dataset $\mathcal{D}$ with i.i.d. samples, we can learn the parameters θ of a distribution via maximum likelihood, *i.e.*, by addressing

$$\max_\theta \sum_{y\in\mathcal{D}} \log p_\theta(y)$$

via gradient descent. State the cost function and its gradient when plugging the model specified in Eq. (1) into this program.

(b) If we aren't given a dataset but if we are instead given a reward function $R(y)$ we search for the parameters θ by maximizing the utility $U(\theta)$, *i.e.*, the expected reward. Explain how we can approximate the utility with N samples from the probability distribution $p_\theta(y)$.

(c) Using general notation, what is the gradient of the utility $U(\theta)$ w.r.t. θ, *i.e.*, what is $\nabla_\theta U(\theta)$. How can we approximate this value with N samples $\{\tilde{y}_i\}_{i=1}^N$from $p_\theta(y)$? Make sure that you state the gradient in the form which ensures that computation via sampling from $p_\theta(y)$ is possible.

(d) Using the parametric probability distribution defined in Eq. (1), what is the approximated gradient of the utility (use N samples from $p_\theta(y)$)? How is this gradient related to the result obtained in part (a)?

(e) In hw6_reinforce.py we compare the two forms of learning.

- Let the size of the domain $|\mathcal{Y}| = 6$, and let the groundtruth data distribution $p_{\text{GT}}(y) = 1/12$ for $y \in \{1, 6\}$, $p_{\text{GT}}(y) = 2/12$ for $y \in \{2, 5\}$, and $p_{\text{GT}}(y) = 3/12$ for $y = \{3, 4\}$.
- The dataset $\mathcal{D}$ in (a) contains $|\mathcal{D}| = 1000$ points sampled from this distribution.
- Further let $F_\theta(y) = [\theta]_y$, where $\theta \in \mathbb{R}^6$ and where $[a]_y$ returns the y-th entry of vector a. The reward function happens to equal the groundtruth distribution, *i.e.*, $R(y) = p_{\text{GT}}(y)$.

Complete hw6_reinforce.py:

i. In function `reinforce`, sample 1000 points from current step distribution p_θ.
Hint: You may find `torch.multinomial` useful.

ii. Compute gradient based on your answer in (d).

Answer the following questions:

i. What distribution p_θ is learned with the maximum likelihood approach? Paste the distribution here.
Hint: the script default prints `torch.nn.Softmax(dim=0)(theta_mle)`. You can directly paste the default printed information here.

ii. What distribution is learned with the REINFORCE approach? Paste the distribution here.
Hint: the script default prints `torch.nn.Softmax(dim=0)(theta_rl)`. You can directly paste the default printed information here.

iii. Explain why this is expected.

Solution.

(a) Cost function:

$$\sum_{y \in \mathcal{D}} \left[F_\theta(y) - \log \sum_{\hat{y} \in \mathcal{Y}} \exp F_\theta(\hat{y}) \right]$$

Gradient:

$$\sum_{y \in \mathcal{D}} \left[\frac{\partial F_\theta(y)}{\partial \theta} - \sum_{\hat{y} \in \mathcal{Y}} \frac{\exp F_\theta(\hat{y})}{\sum_{\hat{y} \in \mathcal{Y}} \exp F_\theta(\hat{y})} \frac{\partial F_\theta(\hat{y})}{\partial \theta} \right] = \sum_{y \in \mathcal{D}, \hat{y} \in \mathcal{Y}} \left[\delta(y = \hat{y}) - p_\theta(\hat{y}) \right] \frac{\partial F_\theta(\hat{y})}{\partial \theta}$$

where, $\delta(y = \hat{y}) = \begin{cases} 1 & \text{if } y = \hat{y} \\ 0 & \text{otherwise} \end{cases}$.

(b) Draw samples $\tilde{y}_i \sim p_\theta(y)$ and average their rewards, *i.e.*, we approximate:

$$U(\theta) = \mathbb{E}_{p_\theta}[R(y)] = \sum_{y \in \mathcal{Y}} p_\theta(y) R(y) \approx \frac{1}{N} \sum_{i=1}^{N} R(\tilde{y}_i) \qquad \text{where} \quad \tilde{y}_i \sim p_\theta(y)$$

(c) Derivation in class slides

$$\nabla_\theta U(\theta) = \sum_{y \in \mathcal{Y}} p_\theta(y) R(y) \nabla_\theta \log p_\theta(y) = \mathbb{E}_{p_\theta}[R(y) \nabla_\theta \log p_\theta(y)] \approx \frac{1}{N} \sum_{i=1}^{N} R(\tilde{y}_i) \nabla_\theta \log p_\theta(\tilde{y}_i)$$

where

$$\tilde{y}_i \sim p_\theta(y)$$

(d)

$$\nabla_\theta U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} R(\tilde{y}_i) \nabla_\theta \log p_\theta(\tilde{y}_i) = \frac{1}{N} \sum_{i=1}^{N} R(\tilde{y}_i) \sum_{\hat{y} \in \mathcal{Y}} \left\{ \left[\delta(\tilde{y}_i = \hat{y}) - p_\theta(\hat{y}) \right] \frac{\partial F_\theta(\hat{y})}{\partial \theta} \right\}$$

We no longer have a groundtruth signal that tells us in what direction to change the prediction. Instead we use the reward $R(\tilde{y}_i)$ to emphasize directions which give large rewards.

(e)
```
def reinforce(R, n_iters, theta=None):
    """REINFORCE with given reward function."""
    alpha = 1

    if theta is None:
        theta = torch.randn(6)

    for i in tqdm.tqdm(range(n_iters)):

        # current distribution
        p_theta = torch.nn.Softmax(dim=0)(theta)

        # sample from current distribution and compute reward

        # TODO: sample from p_theta, [#samples, 1]
        y = torch.multinomial(p_theta, 1000, replacement=True).view((-1, 1))
```

```
        # TODO: use your equation from 4(d) to comppute gradient
        delta = (y == torch.arange(p_theta.shape[0]).view(1, -1)).float()

        # [#samples, 1]
        cur_rew = R[y]

        g = torch.mean(cur_rew * (delta - p_theta), 0)

        # update the parameter
        theta = theta + alpha * g

    return theta
```

i. [0.0900, 0.1590, 0.2450, 0.2520, 0.1750, 0.0790]

ii. [1.3120e−04, 2.3131e−04, 4.3729e−04, 9.9884e−01, 2.4427e−04, 1.1998e−04]

iii. Maximum likelihood: since the dataset size $|\mathcal{D}|$ is reasonably large compared to the domain size $|\mathcal{Y}|$ an accurate distribution can be learned, *i.e.*,

$$p_{\mathrm{GT}}(y) \approx p_{\theta_{\mathrm{ML}}}(y)$$

REINFORCE: REINFORCE attempts to learn a distribution which maximizes the utility when sampled from it, hence we expect

$$p_{\theta_{\mathrm{R}}}(3) = 1 \qquad \text{or} \qquad p_{\theta_{\mathrm{R}}}(4) = 1$$

and all other entries to equal zero. Likely only one of the entires will equal one due to sampling inbalance.