CS 446 / ECE 449 — Homework 6

mukaiyu2

Version 1.1

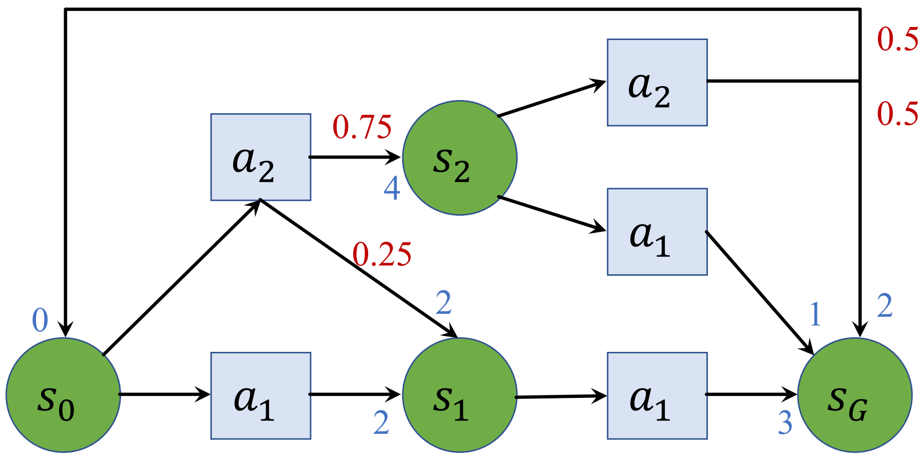
Instructions.

- Homework is due **Tuesday, April 26th, at noon CT**; 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 what are 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 what are 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 what are 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) The essential difference is that **deterministic** Markov Decision Process has an underlying assumption that an action $a \in A_S$ under a state $s \in S$ leads to **a unique consequence** $s' \in S$, whereas **stochastic** Markov Decision Process assumes that an action $a \in A_S$ under a state $s \in S$ can lead to **multiple consequences** $s_1, s_2, ..., s_n \in S$.
The given MDP is of course **stochastic**, since a_2 under s_0 can lead to s_2 and s_1.

(b) The 3 mechanisms are:

i. Exhaustive search
ii. Policy iteration
iii. Value iteration

(c) Policy Graph:

$$\begin{tikzcd}
s_0 \arrow[r] & a_1 \arrow[r] & s_1 \arrow[r] & a_1 \arrow[r] & s_G
\end{tikzcd}$$

$$V^\pi(s_1) = 3$$
$$V^\pi(s_0) = 2 + V^\pi(s_1) = 2 + 3 = 5$$

(d) Policy Graph:

$$\begin{tikzcd}
s_0 \arrow[r] & a_2 \arrow[r] \arrow[dr] & s_1 \arrow[r] & a_1 \arrow[r] & s_G \\
 & & s_2 \arrow[r] & a_1 \arrow[ur] &
\end{tikzcd}$$

$$\begin{aligned}
V^\pi(s_2) &= 1 \\
V^\pi(s_1) &= 3 \\
V^\pi(s_0) &= P(s_1|a_2, s_0)(R(s_1, a_2, s_0) + V^\pi(s_1)) + P(s_2|a_2, s_0)(R(s_1, a_2, s_0) + V^\pi(s_2)) \\
&= 0.25 \times (2 + 3) + 0.75 \times (4 + 1) \\
&= 5
\end{aligned}$$

(e) Policy Graph:

$$\begin{tikzcd}
s_0 \arrow[r] & a_2 \arrow[r] \arrow[dr] & s_1 \arrow[r] & a_1 \arrow[r] & s_G \\
 & & s_2 \arrow[r] & a_2 \arrow[ur] \arrow[ulll, bend left]
\end{tikzcd}$$

$$
\begin{aligned}
V^\pi(s_1) &= 3 \\
V^\pi(s_0) &= P(s_1|a_2, s_0)(R(s_1, a_2, s_0) + V^\pi(s_1)) + P(s_2|a_2, s_0)(R(s_1, a_2, s_0) + V^\pi(s_2)) \\
&= 0.25 \times (2 + 3) + 0.75 \times (4 + V^\pi(s_2)) \\
V^\pi(s_2) &= P(s_G|a_2, s_2)R(s_G, a_2, s_2) + P(s_0|a_2, s_2)(R(s_0, a_2, s_2) + V^\pi(s_0)) \\
&= 0.5 \times 1 + 0.5 \times (0 + V^\pi(s_0)) \\
&\Downarrow \\
&\begin{cases} V^\pi(s_0) = 4.25 + 0.75\ V^\pi(s_2) \\ V^\pi(s_2) = 0.5 + 0.5\ V^\pi(s_0) \end{cases} \\
&\Downarrow \\
&\begin{cases} V^\pi(s_0) = 7.4 \\ V^\pi(s_1) = 3 \\ V^\pi(s_2) = 4.2 \end{cases}
\end{aligned}
$$

(f) The optimal policy is (e):

$$\begin{cases} \pi(s_0) = a_2 \\ \pi(s_1) = a_1 \\ \pi(s_2) = a_2 \end{cases}$$

Because the expected future reward $V^\pi(s_i)$ of every state s_i are the greatest, which means that the policy has greater possibility to gain higher reward.

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 the previous question is 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 a random action with probability ϵ while using the 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: $s_i; a_k : s_i \rightarrow s_j; r_i = X$; 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; $a_1 : s_1 \rightarrow s_1$; $r = -10$;
ii. s_1; $a_2 : s_1 \rightarrow s_2$; $r = -10$;
iii. s_2; $a_1 : s_2 \rightarrow s_1$; $r = 18.5$;
iv. s_1; $a_2 : s_1 \rightarrow s_2$; $r = -10$;

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

Solution:

(a)

$$Q^*(s,a) = \sum_{s' \in S} P(s'|s,a) \left(r(s,a,s') + \max_{a' \in \mathcal{A}_{s'}} Q^*(s',a') \right)$$

(b)

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a' \in \mathcal{A}_{s'}} Q(s',a'))$$

(c) This is a trade-off between using past experience and exploring the potential of higher unseen reward. ϵ-greedy strategy allow us to explore unseen path that could lead us to higher Q(s, a) if we ever chose some suboptimal action during initial steps.

(d) Replay-memory is basically sampling a few reward and consequent state given current state and action, then randomly sample from this ”past experience” during the actual training stage.

i. This is benefitial when gaining real world experience is costly, and the random sample can still simulate underlying probabilities.

ii. Since the number of possible outcome of a state and action is now limited, it’s easier for the network to converge, especially when a state-action pair can lead to complicated reward-consequence.

iii. It can also break the correlations between consecutive samples, enforcing the markov assumption (forcing the decision-making to be more independent of consecutive state-action pairs), making the network more robust.

(e) i.

Q	s_1	s_2
a_1	-5	0
a_2	0	0

ii.

Q	s_1	s_2
a_1	-5	0
a_2	-5	0

iii.

Q	s_1	s_2
a_1	-5	8
a_2	-5	0

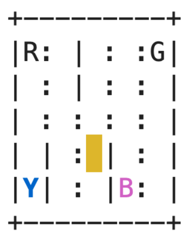
iv.

Q	s_1	s_2
a_1	-5	8
a_2	-5.5	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 the figure below. Your agent should be performing well after around 1000 episodes.

You can install gym via `pip install gym`.



```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

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

- There are six possible actions: `move south/north/east/west`, `pick up passenger`, and `drop off passenger`.
- There are four possible locations that the 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 the passenger at the blue text (`Y`) and then drive to the pink text (`B`) for drop off. Note, colors are only used for visualization. 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`.

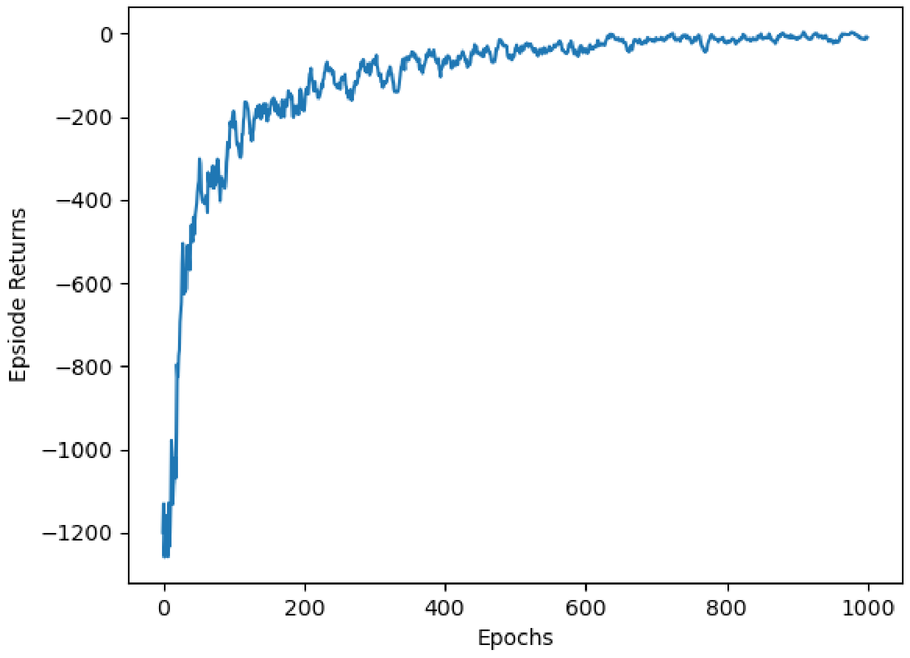
(d) Plot the training curve. By default, the script will save a `train_moving_avg.png`. Attach it here.

(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.

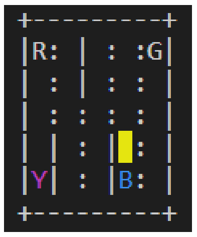
[1]`https://gym.openai.com/envs/Taxi-v3/`
[2]`https://github.com/openai/gym/blob/master/gym/envs/toy_text/taxi.py`

Solution:

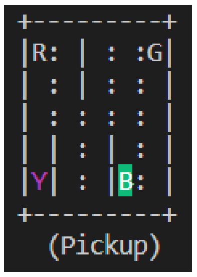
(d)



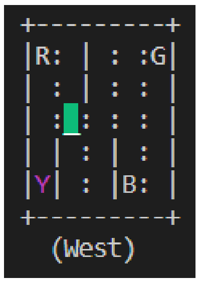
(e) Obtian total reward of 11.0 after 11 steps.



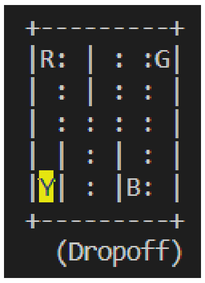
```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```



```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)
```



```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
```



```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
```

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 the 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 is the negative log likelihood:

$$\begin{aligned}
&-\sum_{y\in\mathcal{D}} \log p_\theta(y) \\
=&-\sum_{y\in\mathcal{D}} \log \frac{e^{F_\theta(y)}}{\sum\limits_{\hat{y}\in\mathcal{Y}} e^{F_\theta(\hat{y})}} \\
=&\sum_{y\in\mathcal{D}} \left(\log \sum_{\hat{y}\in\mathcal{Y}} e^{F_\theta(\hat{y})} - \log e^{F_\theta(y)} \right) \\
=&|\mathcal{D}| \log \sum_{\hat{y}\in\mathcal{Y}} e^{F_\theta(\hat{y})} - \sum_{y\in\mathcal{D}} F_\theta(y)
\end{aligned}$$

And the gradient is:

$$\begin{aligned}
&\nabla_\theta \left(|\mathcal{D}| \log \sum_{\hat{y}\in\mathcal{Y}} e^{F_\theta(\hat{y})} - \sum_{y\in\mathcal{D}} F_\theta(y) \right) \\
=&|\mathcal{D}| \frac{\sum\limits_{\hat{y}\in\mathcal{Y}} e^{F_\theta(\hat{y})} \nabla_\theta F_\theta(\hat{y})}{\sum\limits_{\hat{y}\in\mathcal{Y}} e^{F_\theta(\hat{y})}} - \sum_{y\in\mathcal{D}} \nabla_\theta F_\theta(y)
\end{aligned}$$

(b) We can simply just sample N ys from $p_\theta(y)$, and average $R(y)$.
So for consistency with (c), assume that the N samples we draw from $p_\theta(y)$ are $\{\tilde{y_i}\}_{i=1}^N$, then:

$$U(\theta) = \mathbb{E}_{p_\theta}[R(y)] = \frac{1}{N} \sum_{\tilde{y}\in\{\tilde{y_i}\}_{i=1}^N} R(\tilde{y})$$

(c) For general notation, gradient of $U(\theta)$ is:

$$\begin{aligned}
\nabla_\theta U(\theta) &= \sum_{y\in\mathcal{Y}} \nabla_\theta p_\theta(y) R(y) \\
&= \sum_{y\in\mathcal{Y}} p_\theta(y) \frac{\nabla_\theta p_\theta(y)}{p_\theta(y)} R(y) \\
&= \sum_{y\in\mathcal{Y}} p_\theta(y) \nabla_\theta \log p_\theta(y) R(y)
\end{aligned}$$

This is possible to compute via sampling, because $\sum\limits_{y\in\mathcal{Y}} p_\theta(y)...$ can be approximated as $\frac{1}{N} \sum\limits_{\tilde{y}\in\{\tilde{y_i}\}_{i=1}^N} ...$:

$$\nabla_\theta U(\theta) \approx \frac{1}{N} \sum_{\tilde{y}\in\{\tilde{y_i}\}_{i=1}^N} \nabla_\theta \log p_\theta(\tilde{y}) R(\tilde{y})$$

(d) Plugging in Eq. (1):

$$
\begin{aligned}
\nabla_\theta U(\theta) &\approx \frac{1}{N} \sum_{\tilde{y} \in \{\tilde{y_i}\}_{i=1}^N} \nabla_\theta \log p_\theta(\tilde{y}) R(\tilde{y}) \\
&= \frac{1}{N} \sum_{\tilde{y} \in \{\tilde{y_i}\}_{i=1}^N} \left(\nabla_\theta F_\theta(\tilde{y}) - \frac{\sum\limits_{\hat{y} \in \mathcal{Y}} e^{F_\theta(\hat{y})} \nabla_\theta F_\theta(\hat{y})}{\sum\limits_{\hat{y} \in \mathcal{Y}} e^{F_\theta(\hat{y})}} \right) R(\tilde{y}) \\
&= \frac{1}{N} \sum_{\tilde{y} \in \{\tilde{y_i}\}_{i=1}^N} R(\tilde{y}) \nabla_\theta F_\theta(\tilde{y}) - \left(\frac{\sum\limits_{\hat{y} \in \mathcal{Y}} e^{F_\theta(\hat{y})} \nabla_\theta F_\theta(\hat{y})}{\sum\limits_{\hat{y} \in \mathcal{Y}} e^{F_\theta(\hat{y})}} \right) \frac{1}{N} \sum_{\tilde{y} \in \{\tilde{y_i}\}_{i=1}^N} R(\tilde{y})
\end{aligned}
$$

This is just the negated gradient in (a) multiplied by $R(\tilde{y})$.

(e) i. `torch.nn.Softmax(dim=0)(theta_mle)` :
0.0900, 0.1590, 0.2450, 0.2520, 0.1750, 0.0790

ii. `torch.nn.Softmax(dim=0)(theta_rl)` :
1.2459e-04, 1.4795e-04, 3.7490e-04, **9.9894e-01**, 2.7167e-04, 1.4000e-04
or
1.4452e-04, 1.9596e-04, **9.9845e-01**, 8.1436e-04, 2.7399e-04, 1.1982e-04

iii.
- Calculating $U(\theta)$ for both maximum likelihood and reinforcement learning:

$$
\begin{cases} U_{mle}(\theta) = 0.1940 \\ U_{rl}(\theta) = 0.2499 \end{cases}
$$

- These are 2 totally different things, maximum likelihood is trying to learn $p_\theta(y)$ that is close to $R(y)$ (equivalent to reinforcement learning with $R(y)$ set to all 1), while reinforcement learning is trying to maximize $U(\theta)$, and it keeps refining from previous learned $p_\theta(y)$, thus will ultimately choose the best y with the highest reward ($y = \{3, 4\}$).
- Because $p_\theta(y)$ in reinforcement learning is SoftMax, all the probabilities are > 0, otherwise other probabilites could be push to ultimately 0, resulting in a single discrete choice.