CS 440 Problem Set 4

Gordon Ng

TOTAL POINTS

39 / 45

QUESTION 1

1 Q1 **31 / 31**

✓ **- 0 pts** Correct

QUESTION 2

2 Q2 **4 / 5**

- 0 pts Correct

- 1 Point adjustment

2b

QUESTION 3

3 Q3 **0 / 5**

- 0 pts Correct

- 5 Point adjustment

3a) Increase epsilon, 3b) unrelated solution

QUESTION 4

4 Q4 **4 / 4**

✓ **- 0 pts** Correct

Problem Set 4

Total: 45 points, Due: Dec 10, 11:59pm

Q1 [31 points] Reinforcement Learning

Q1a. [11 points] *Written RL problem*

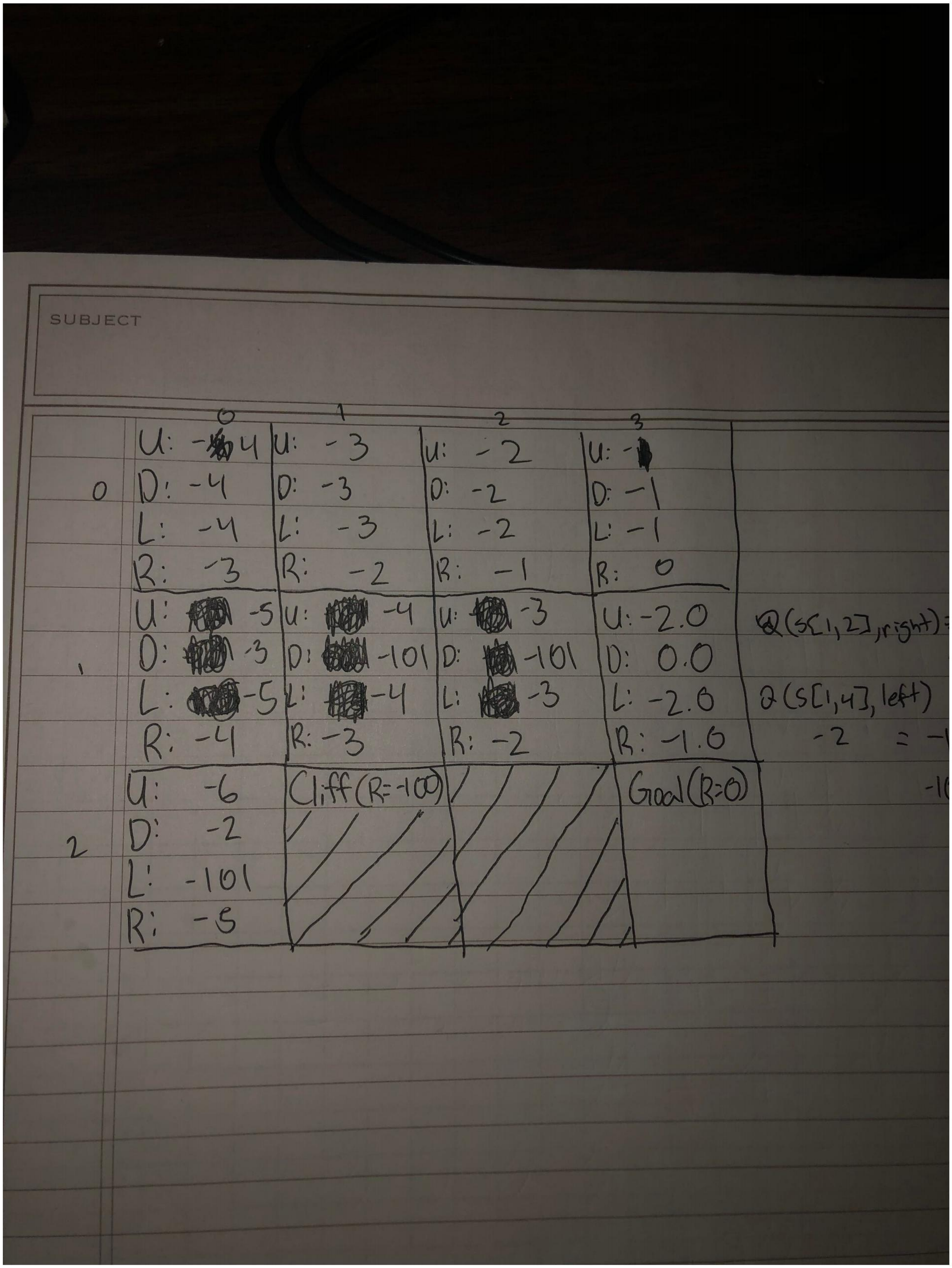
For a simple cliff-walker Q-value problem, compute the Q-values at each state. The goal is the cell marked in green (with a reward of 0), and stepping on the red cells results in immediate failure with reward -100. All other states get a reward of -1.

The Q-value equation is given by:

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

Assume a discount factor of 1.0 (i.e. $\gamma = 1$). As an example, Q-values for one cell have been computed for you.

(R = -1) U: D: L: R:	(R = -1) U: D: L: R:	(R = -1) U: D: L: R:	(R = -1) U: D: L: R:
(R = -1) U: D: L: R:	(R = -1) U: D: L: R:	(R = -1) U: D: L: R:	(R = -1) U: -2.0 D: 0.0 L: -2.0 R: -1.0
(R = -1) U: D: L: R:	Cliff (R = -100)		Goal (R = 0)



Q1b. [20 points] *Coding RL problem*

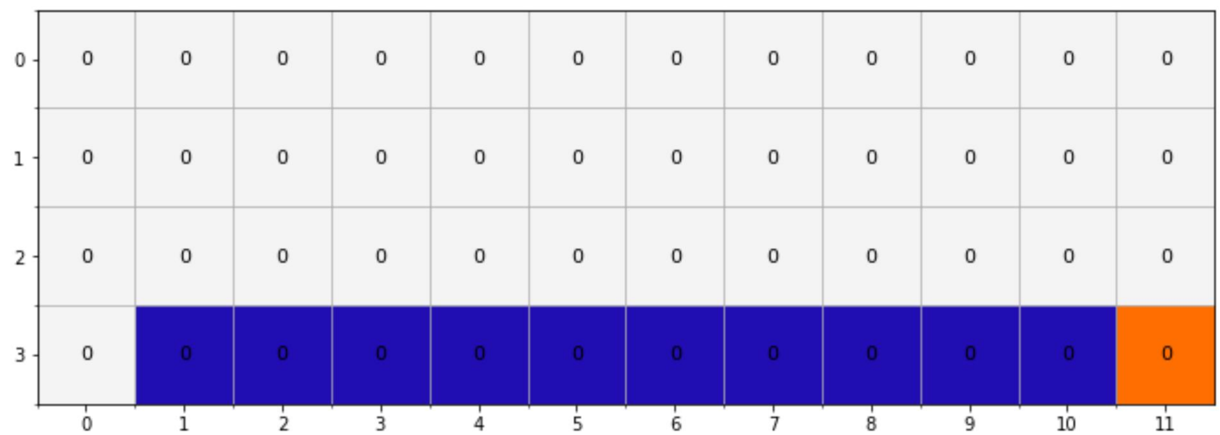
Let's start by reading about the [Cliff Walking Problem (https://medium.com/@lgvaz/understanding-q-learning-the-cliff-walking-problem-80198921abbc)](https://medium.com/@lgvaz/understanding-q-learning-the-cliff-walking-problem-80198921abbc)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from CliffWalker import GridWorld
```

We create a 4×12 grid, similar to the written problem in 1a. above on which you will implement a Q-learning algorithm.

In [2]:
```
env = GridWorld()

# The number of states in simply the number of "squares" in our grid world, in th
num_states = 4 * 12
# We have 4 possible actions, up, down, right and left
num_actions = 4
```



Tasks

We ask you to implement two functions:

- an ϵ-greedy action picker
- a basic Q-learning algorithm

ϵ-greedy choices make the greedy choice most of the time but choose a random action ϵ fraction of the time. For example, for $\epsilon = 0.1$, if a random number is ≤ 0.1, then a random action is taken.

In [3]:
```
def egreedy_policy(q_values, state, epsilon=0.1):
    '''
    Choose an action based on a epsilon greedy policy.
    A random action is selected with epsilon probability, else select the best a
    '''

    # === STUDENT CODE GOES HERE ===
    if np.random.random() < epsilon:
        return np.random.choice(4)
    else:
        return np.argmax(q_values[state])
    # ==============================
```

Now, you can implement a basic Q-learning algorithm. For your reference, use the following:

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

We provide a skeleton code, leaving the Q-value update for you to implement.

Note: learning rate α, exploration rate ϵ, and discount factor γ are provided as inputs to the function

In [4]:
```
def q_learning(env, num_episodes=200, render=True, epsilon=0.1,
               learning_rate=0.5, gamma=0.9):
    q_values = np.zeros((num_states, num_actions))
    ep_rewards = []

    for _ in range(num_episodes):
        state = env.reset()
        done = False
        reward_sum = 0

        while not done:
            # Choose action
            action = egreedy_policy(q_values, state, epsilon)
            # Do the action
            next_state, reward, done = env.step(action)
            reward_sum += reward
            # Update Q-values
            # === STUDENT CODE GOES HERE ===
            td_target = reward + 0.9 * np.max(q_values[next_state])
            td_error = td_target - q_values[state][action]
            q_values[state][action] += learning_rate * td_error
            # ==============================
            # Update state
            state = next_state
            if render:
                env.render(q_values, action=actions[action], colorize_q=True)

        ep_rewards.append(reward_sum)

    return ep_rewards, q_values
```

Now, let's the run Q-learning

1 Q1 31 / 31

✓ - 0 pts Correct

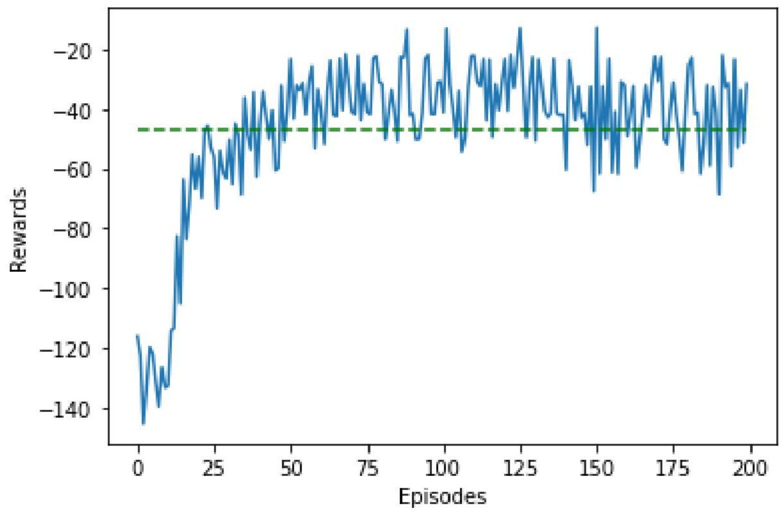
In [5]:
```
q_learning_rewards, q_values = q_learning(env, gamma=0.9, learning_rate=1, render

q_learning_rewards, _ = zip(*[q_learning(env, render=False, epsilon=0.1, learnin
avg_rewards = np.mean(q_learning_rewards, axis=0)
mean_reward = [np.mean(avg_rewards)] * len(avg_rewards)

fig, ax = plt.subplots()
ax.set_xlabel('Episodes')
ax.set_ylabel('Rewards')
ax.plot(avg_rewards)
ax.plot(mean_reward, 'g--')

print('Mean Reward: {}'.format(mean_reward[0]))
```

```
Mean Reward: -46.67
```



Visualization

Finally, let's look at the policy learned

In [6]:
```
def play(q_values):
    env = GridWorld()
    state = env.reset()
    done = False

    while not done:
        # Select action
        action = egreedy_policy(q_values, state, 0.0)
        # Do the action
        next_state, reward, done = env.step(action)

        # Update state and action
        state = next_state

        env.render(q_values=q_values, action=action, colorize_q=True)
```

In [ ]:
```
%matplotlib
play(q_values)
```

Q2 [5 points] Metrics

Q2a. [3 points]

Give one example each of error metrics that can be used to evaluate: classification, regression, clustering.

=== ANSWER GOES HERE === Classification - log loss Regression - Mean Squared Error, Root mean squared error, mean absolute error Clustering - Dunn's index

========================

Q2b. [2 points]

Which are the correct definitions of precision and recall? Here 'actual positives' are examples labeled positive (by humans), and 'predicted positives' are examples for which the algorithm predicts a positive label.

1. *precision=(true positives)/(predicted positives)*
2. *precision=(true positives)/(actual positives)*
3. *recall=(predicted positives)/(actual positives)*
4. *recall=(true positives)/(actual positives)*

=== ANSWER GOES HERE === 2 and 4

========================

Q3 [5 points] Unsupervised Learning

Q3a. [2 points]

Suppose you have trained an anomaly detection system for intruder detection in a security camera, and your system flags anomalies when p(x) is less than ε. You find on the cross-validation set that it is missing many intruder events. What should you do?

=== ANSWER GOES HERE === Decrease Epsilon

========================

Q3b. [3 points]

Suppose we are given inputs $x^i \in \mathbb{R}^n, i = 1, \ldots, m$ and we want to learn a lower-dimensional (k-dim) PCA projection of the data onto basis vectors $U = [u^1 \ldots u^k]$ where each $u^j \in \mathbb{R}^n$. Write down the equation for the general k-dimensional point z^i obtained by projecting an n-dimensional point x^i onto the k basis vectors.

=== ANSWER GOES HERE ===

$$\tilde{x} = \begin{bmatrix} x_{\mathrm{rot},1} \\ \vdots \\ x_{\mathrm{rot},k} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \approx \begin{bmatrix} x_{\mathrm{rot},1} \\ \vdots \\ x_{\mathrm{rot},k} \\ x_{\mathrm{rot},k+1} \\ \vdots \\ x_{\mathrm{rot},n} \end{bmatrix} = x_{\mathrm{rot}}$$

Q4 [4 points] Bayesian Methods

Q4a. [2 points]

What in the Bayesian model is equivalent to changing the regularization parameter λ?

```
=== ANSWER GOES HERE ===
Prior distributions p(θ) are probability distributions of model
parameters based on some a priori knowledge about the
parameters.

========================
```

Q4b. [2 points]

Write the posterior probability function, and comment on its relation to the likelihood.

=== ANSWER GOES HERE ===

2 Q2 **4 / 5**

- **- 0 pts** Correct

- 1 Point adjustment

2b

Q3b. [3 points]

Suppose we are given inputs $x^i \in \mathbb{R}^n, i = 1, \ldots, m$ and we want to learn a lower-dimensional (k-dim) PCA projection of the data onto basis vectors $U = [u^1 \ldots u^k]$ where each $u^j \in \mathbb{R}^n$. Write down the equation for the general k-dimensional point z^i obtained by projecting an n-dimensional point x^i onto the k basis vectors.

=== ANSWER GOES HERE ===

$$\tilde{x} = \begin{bmatrix} x_{\mathrm{rot},1} \\ \vdots \\ x_{\mathrm{rot},k} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \approx \begin{bmatrix} x_{\mathrm{rot},1} \\ \vdots \\ x_{\mathrm{rot},k} \\ x_{\mathrm{rot},k+1} \\ \vdots \\ x_{\mathrm{rot},n} \end{bmatrix} = x_{\mathrm{rot}}$$

Q4 [4 points] Bayesian Methods

Q4a. [2 points]

What in the Bayesian model is equivalent to changing the regularization parameter λ?

=== ANSWER GOES HERE ===
Prior distributions $p(\theta)$ are probability distributions of model
parameters based on some a priori knowledge about the
parameters.

========================

Q4b. [2 points]

Write the posterior probability function, and comment on its relation to the likelihood.

=== ANSWER GOES HERE ===

3 Q3 **0 / 5**

- **- 0 pts** Correct

- 5 Point adjustment

3a) Increase epsilon, 3b) unrelated solution

> Given a prior belief that a probability distribution function is $p(\theta)$ and that the observations x have a likelihood $p(x|\theta)$, then the posterior probability is defined as
>
> $$p(\theta|x) = \frac{p(x|\theta)}{p(x)} p(\theta)$$ [1]
>
> where $p(x)$ is the normalizing constant and is calculated as
>
> $$p(x) = \int p(x|\theta)p(\theta)d\theta$$
>
> for continuous θ, or by summing $p(x|\theta)p(\theta)$ over all possible values of θ for discrete θ.[2]
>
> The posterior probability is therefore proportional to the product *Likelihood · Prior probability*.

In [ ]:

4 Q4 **4 / 4**

✓ **- 0 pts** Correct