CMPE 260 RL Midterm Exam

Instructor: Jahan Ghofraniha

Student Name: Anusha Gangasani

ID: 015513160

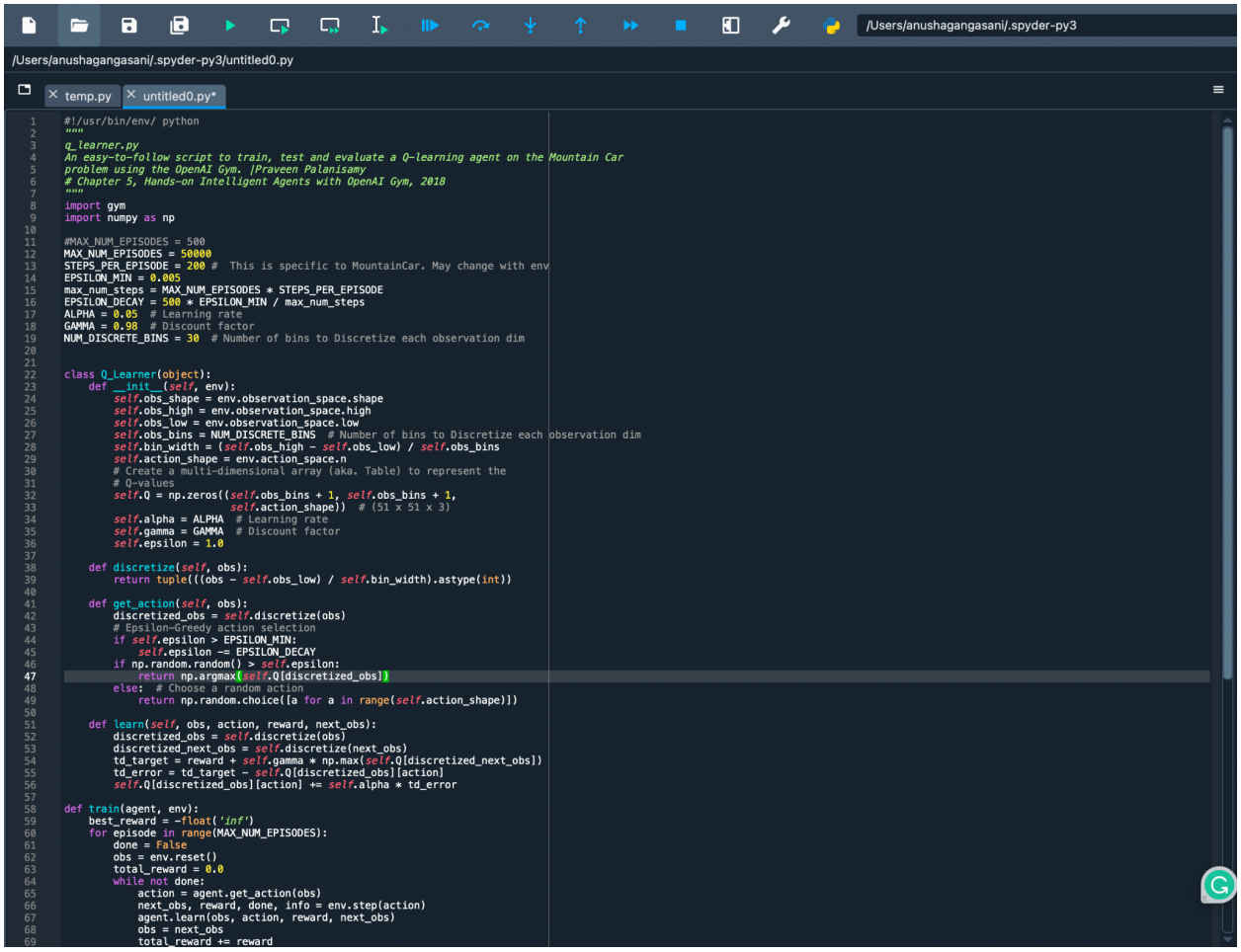
Problem 1 (60 pts):

1. Q-Learning for mountain car

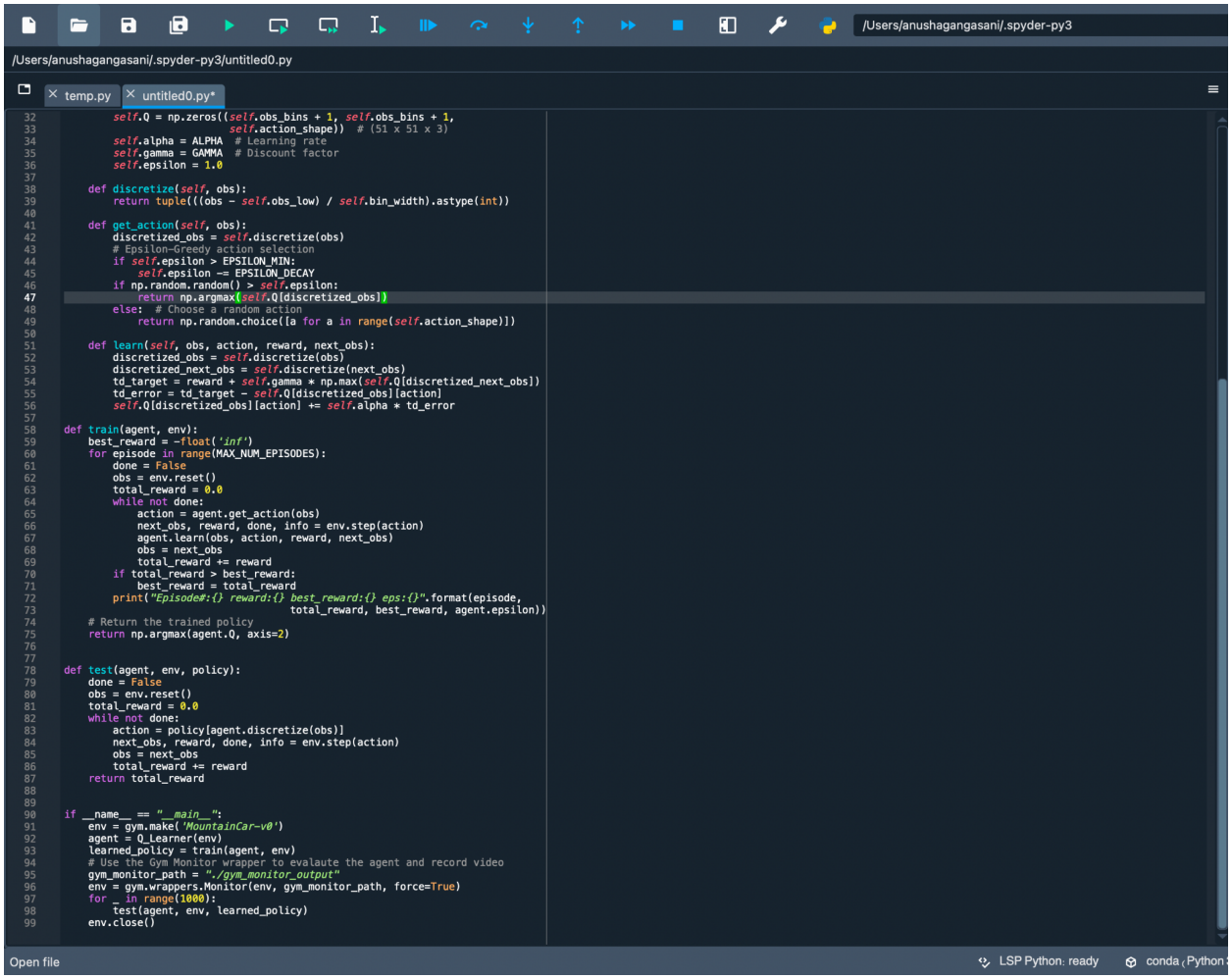
Reference:

https://sjsu.instructure.com/courses/1529630/files/69364522?module_item_id=13628553

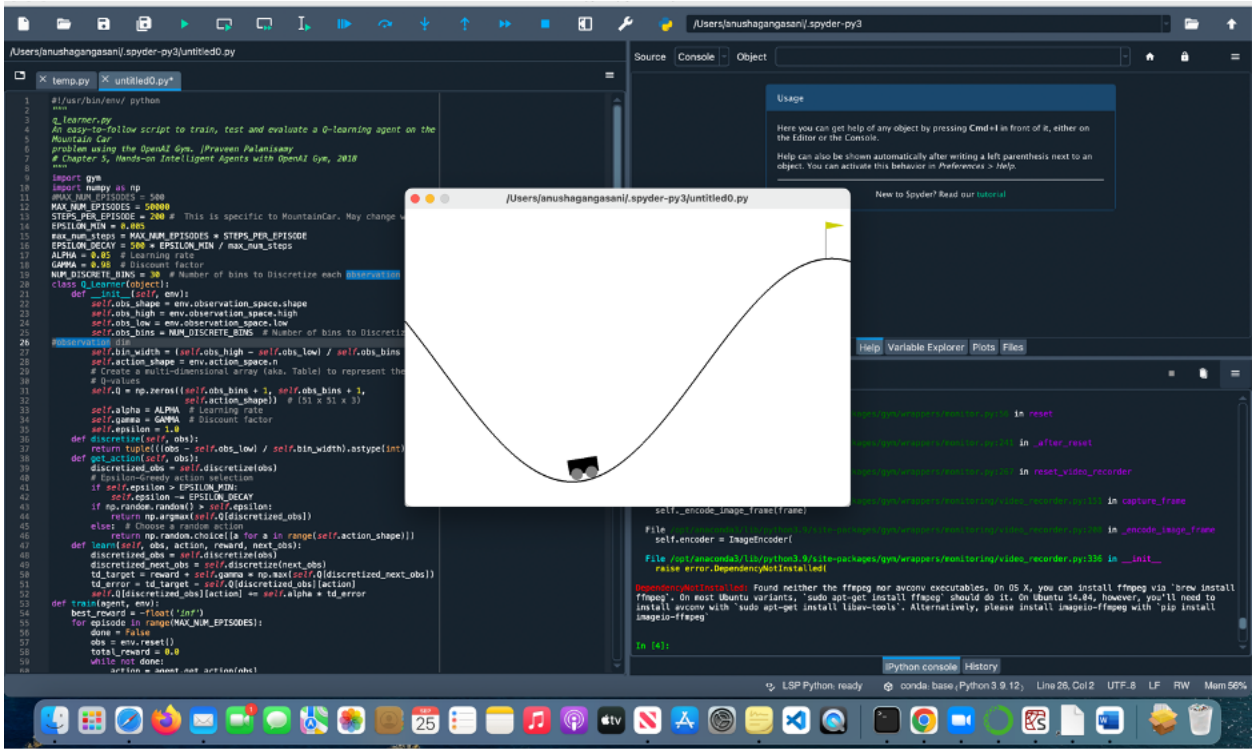
Code:



```
#!/usr/bin/env/ python
"""
q_learner.py
An easy-to-follow script to train, test and evaluate a Q-learning agent on the Mountain Car
problem using the OpenAI Gym. |Praveen Palanisamy
# Chapter 5, Hands-on Intelligent Agents with OpenAI Gym, 2018
"""
import gym
import numpy as np

#MAX_NUM_EPISODES = 500
MAX_NUM_EPISODES = 50000
STEPS_PER_EPISODE = 200 #  This is specific to MountainCar. May change with env
EPSILON_MIN = 0.005
max_num_steps = MAX_NUM_EPISODES * STEPS_PER_EPISODE
EPSILON_DECAY = 500 * EPSILON_MIN / max_num_steps
ALPHA = 0.05  # Learning rate
GAMMA = 0.98  # Discount factor
NUM_DISCRETE_BINS = 30  # Number of bins to Discretize each observation dim


class Q_Learner(object):
    def __init__(self, env):
        self.obs_shape = env.observation_space.shape
        self.obs_high = env.observation_space.high
        self.obs_low = env.observation_space.low
        self.obs_bins = NUM_DISCRETE_BINS  # Number of bins to Discretize each observation dim
        self.bin_width = (self.obs_high - self.obs_low) / self.obs_bins
        self.action_shape = env.action_space.n
        # Create a multi-dimensional array (aka. Table) to represent the
        # Q-values
        self.Q = np.zeros((self.obs_bins + 1, self.obs_bins + 1,
                           self.action_shape))  # (51 x 51 x 3)
        self.alpha = ALPHA  # Learning rate
        self.gamma = GAMMA  # Discount factor
        self.epsilon = 1.0

    def discretize(self, obs):
        return tuple(((obs - self.obs_low) / self.bin_width).astype(int))

    def get_action(self, obs):
        discretized_obs = self.discretize(obs)
        # Epsilon-Greedy action selection
        if self.epsilon > EPSILON_MIN:
            self.epsilon -= EPSILON_DECAY
        if np.random.random() > self.epsilon:
            return np.argmax(self.Q[discretized_obs])
        else:  # Choose a random action
            return np.random.choice([a for a in range(self.action_shape)])

    def learn(self, obs, action, reward, next_obs):
        discretized_obs = self.discretize(obs)
        discretized_next_obs = self.discretize(next_obs)
        td_target = reward + self.gamma * np.max(self.Q[discretized_next_obs])
        td_error = td_target - self.Q[discretized_obs][action]
        self.Q[discretized_obs][action] += self.alpha * td_error

def train(agent, env):
    best_reward = -float('inf')
    for episode in range(MAX_NUM_EPISODES):
        done = False
        obs = env.reset()
        total_reward = 0.0
        while not done:
            action = agent.get_action(obs)
            next_obs, reward, done, info = env.step(action)
            agent.learn(obs, action, reward, next_obs)
            obs = next_obs
            total_reward += reward
```

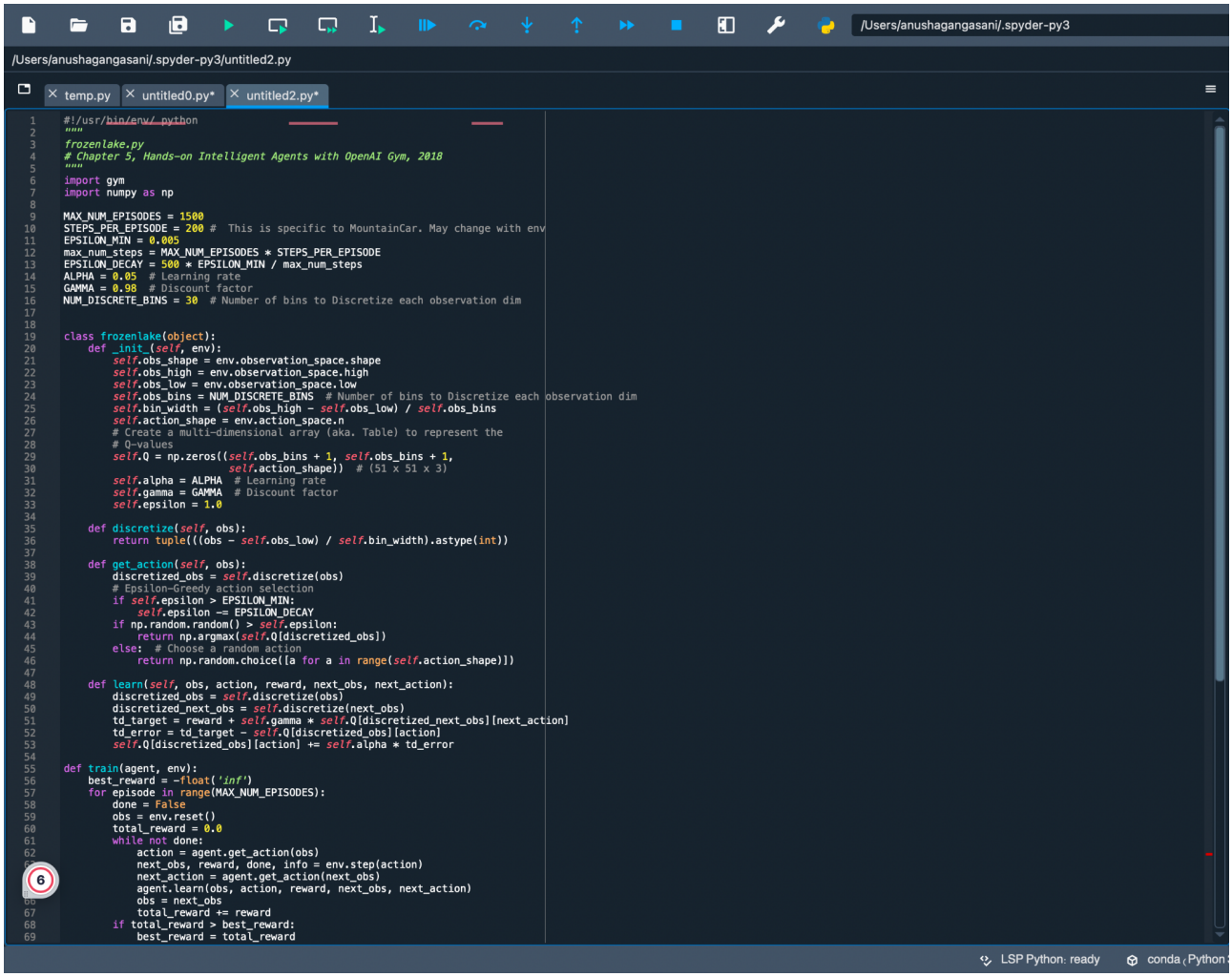

/Users/anushagangasani/.spyder-py3
/Users/anushagangasani/.spyder-py3/untitled0.py
temp.py
untitled0.py*
self.Q = np.zeros((self.obs_bins + 1, self.obs_bins + 1,
self.action_shape)) # (51 x 51 x 3)
self.alpha = ALPHA # Learning rate
self.gamma = GAMMA # Discount factor
self.epsilon = 1.0
def discretize(self, obs):
return tuple(((obs - self.obs_low) / self.bin_width).astype(int))
def get_action(self, obs):
discretized_obs = self.discretize(obs)
# Epsilon-Greedy action selection
if self.epsilon > EPSILON_MIN:
self.epsilon -= EPSILON_DECAY
if np.random.random() > self.epsilon:
return np.argmax(self.Q[discretized_obs])
else: # Choose a random action
return np.random.choice([a for a in range(self.action_shape)])
def learn(self, obs, action, reward, next_obs):
discretized_obs = self.discretize(obs)
discretized_next_obs = self.discretize(next_obs)
td_target = reward + self.gamma * np.max(self.Q[discretized_next_obs])
td_error = td_target - self.Q[discretized_obs][action]
self.Q[discretized_obs][action] += self.alpha * td_error
def train(agent, env):
best_reward = -float('inf')
for episode in range(MAX_NUM_EPISODES):
done = False
obs = env.reset()
total_reward = 0.0
while not done:
action = agent.get_action(obs)
next_obs, reward, done, info = env.step(action)
agent.learn(obs, action, reward, next_obs)
obs = next_obs
total_reward += reward
if total_reward > best_reward:
best_reward = total_reward
print("Episode#:{} reward:{} best_reward:{} eps:{}".format(episode,
total_reward, best_reward, agent.epsilon))
Return the trained policy
return np.argmax(agent.Q, axis=2)
def test(agent, env, policy):
done = False
obs = env.reset()
total_reward = 0.0
while not done:
action = policy[agent.discretize(obs)]
next_obs, reward, done, info = env.step(action)
obs = next_obs
total_reward += reward
return total_reward
if __name__ == "__main__":
env = gym.make('MountainCar-v0')
agent = Q_Learner(env)
learned_policy = train(agent, env)
# Use the Gym Monitor wrapper to evalaute the agent and record video
gym_monitor_path = "./gym_monitor_output"
env = gym.wrappers.Monitor(env, gym_monitor_path, force=True)
for _ in range(1000):
test(agent, env, learned_policy)
env.close()
Open file
LSP Python: ready
conda (Python

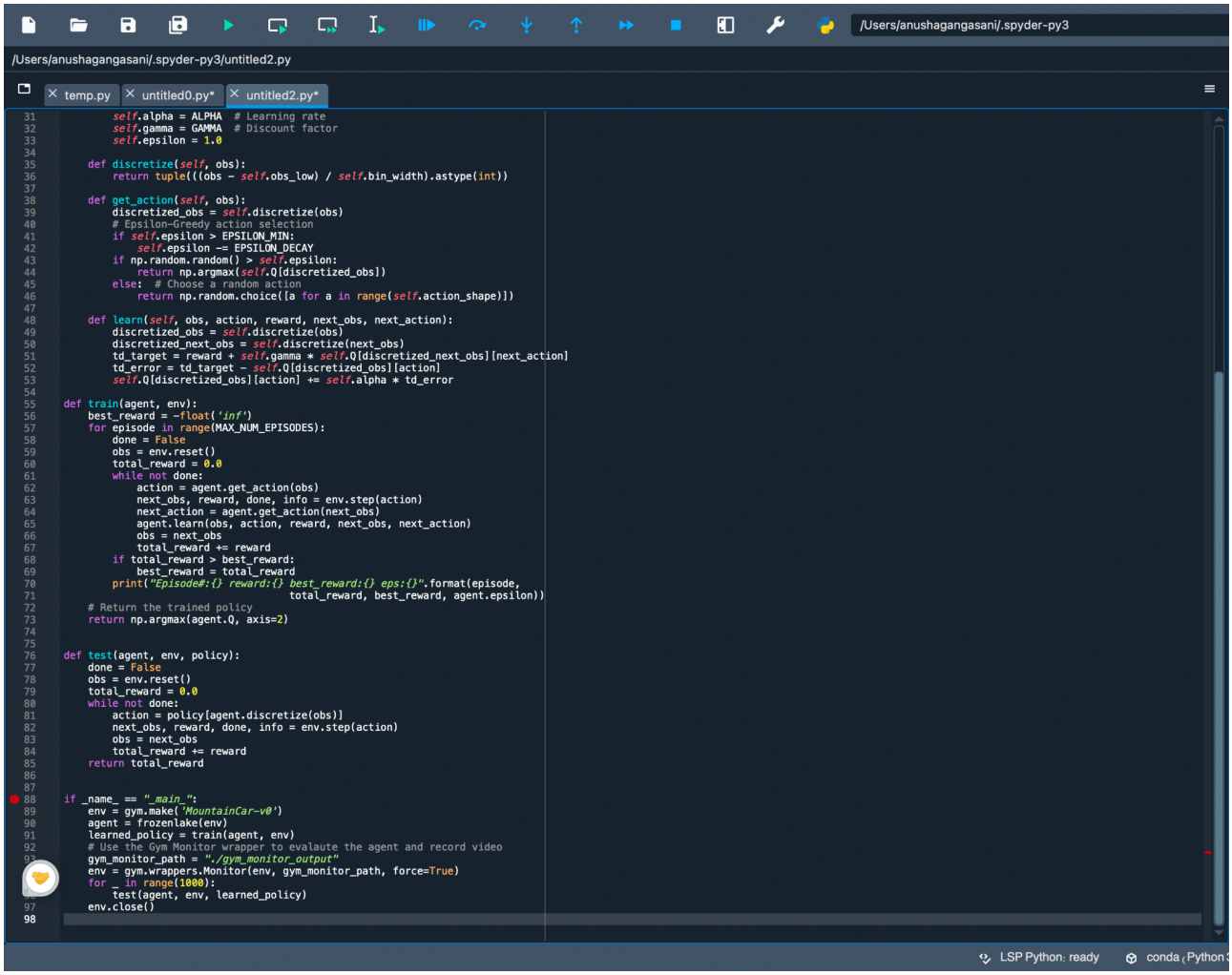
Output:



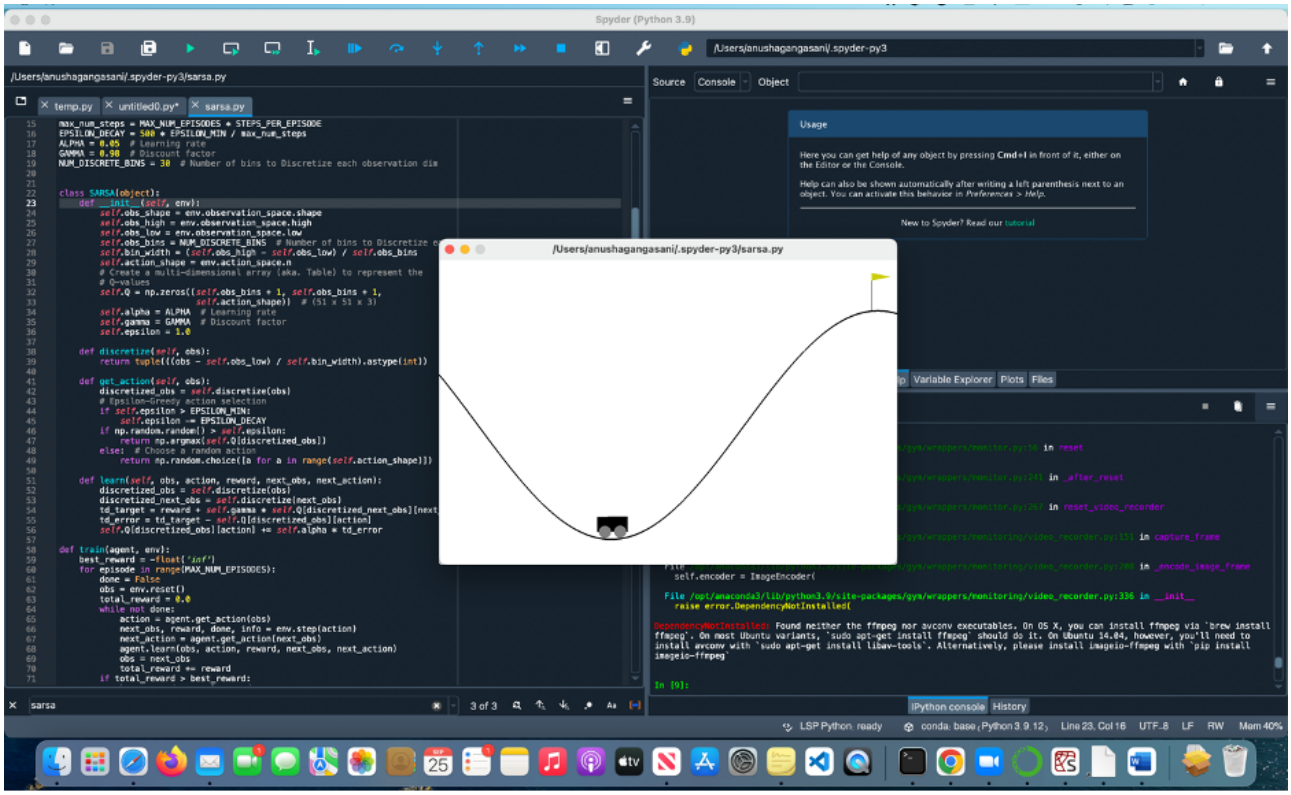
2. Q-iteration sample code based on frozen lake environment

Code:

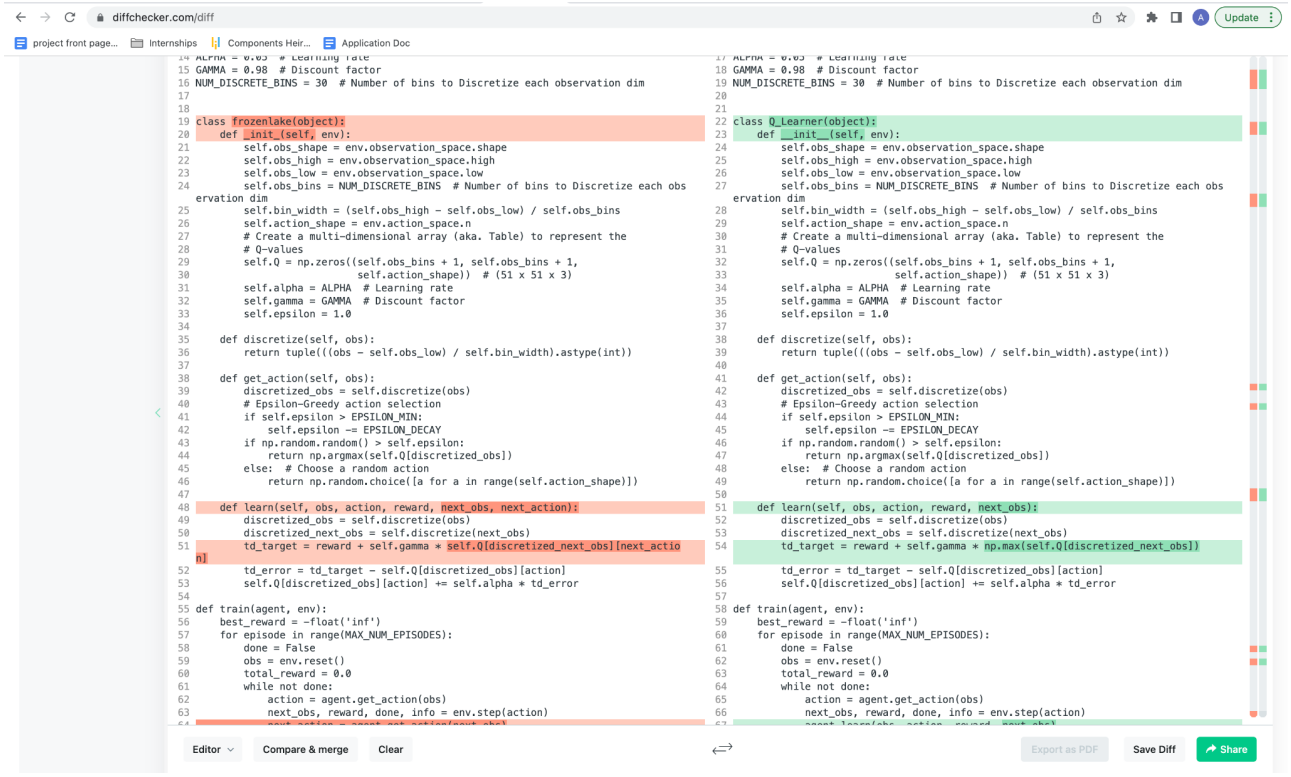

/Users/anushagangasani/.spyder-py3
/Users/anushagangasani/.spyder-py3/untitled2.py
temp.py
untitled0.py*
untitled2.py*
#!/usr/bin/env/ python
"""
frozenlake.py
Chapter 5, Hands-on Intelligent Agents with OpenAI Gym, 2018
"""
import gym
import numpy as np
MAX_NUM_EPISODES = 1500
STEPS_PER_EPISODE = 200 # This is specific to MountainCar. May change with env
EPSILON_MIN = 0.005
max_num_steps = MAX_NUM_EPISODES * STEPS_PER_EPISODE
EPSILON_DECAY = 500 * EPSILON_MIN / max_num_steps
ALPHA = 0.05 # Learning rate
GAMMA = 0.98 # Discount factor
NUM_DISCRETE_BINS = 30 # Number of bins to Discretize each observation dim
class frozenlake(object):
def _init_(self, env):
self.obs_shape = env.observation_space.shape
self.obs_high = env.observation_space.high
self.obs_low = env.observation_space.low
self.obs_bins = NUM_DISCRETE_BINS # Number of bins to Discretize each observation dim
self.bin_width = (self.obs_high - self.obs_low) / self.obs_bins
self.action_shape = env.action_space.n
# Create a multi-dimensional array (aka. Table) to represent the
# Q-values
self.Q = np.zeros((self.obs_bins + 1, self.obs_bins + 1,
self.action_shape)) # (51 x 51 x 3)
self.alpha = ALPHA # Learning rate
self.gamma = GAMMA # Discount factor
self.epsilon = 1.0
def discretize(self, obs):
return tuple(((obs - self.obs_low) / self.bin_width).astype(int))
def get_action(self, obs):
discretized_obs = self.discretize(obs)
# Epsilon-Greedy action selection
if self.epsilon > EPSILON_MIN:
self.epsilon -= EPSILON_DECAY
if np.random.random() > self.epsilon:
return np.argmax(self.Q[discretized_obs])
else: # Choose a random action
return np.random.choice([a for a in range(self.action_shape)])
def learn(self, obs, action, reward, next_obs, next_action):
discretized_obs = self.discretize(obs)
discretized_next_obs = self.discretize(next_obs)
td_target = reward + self.gamma * self.Q[discretized_next_obs][next_action]
td_error = td_target - self.Q[discretized_obs][action]
self.Q[discretized_obs][action] += self.alpha * td_error
def train(agent, env):
best_reward = -float('inf')
for episode in range(MAX_NUM_EPISODES):
done = False
obs = env.reset()
total_reward = 0.0
while not done:
action = agent.get_action(obs)
next_obs, reward, done, info = env.step(action)
next_action = agent.get_action(next_obs)
agent.learn(obs, action, reward, next_obs, next_action)
obs = next_obs
total_reward += reward
if total_reward > best_reward:
best_reward = total_reward
6
LSP Python: ready
conda (Python


/Users/anushagangasani/.spyder-py3
/Users/anushagangasani/.spyder-py3/untitled2.py
temp.py
untitled0.py*
untitled2.py*
self.alpha = ALPHA # Learning rate
self.gamma = GAMMA # Discount factor
self.epsilon = 1.0
def discretize(self, obs):
return tuple(((obs - self.obs_low) / self.bin_width).astype(int))
def get_action(self, obs):
discretized_obs = self.discretize(obs)
# Epsilon-Greedy action selection
if self.epsilon > EPSILON_MIN:
self.epsilon -= EPSILON_DECAY
if np.random.random() > self.epsilon:
return np.argmax(self.Q[discretized_obs])
else: # Choose a random action
return np.random.choice([a for a in range(self.action_shape)])
def learn(self, obs, action, reward, next_obs, next_action):
discretized_obs = self.discretize(obs)
discretized_next_obs = self.discretize(next_obs)
td_target = reward + self.gamma * self.Q[discretized_next_obs][next_action]
td_error = td_target - self.Q[discretized_obs][action]
self.Q[discretized_obs][action] += self.alpha * td_error
def train(agent, env):
best_reward = -float('inf')
for episode in range(MAX_NUM_EPISODES):
done = False
obs = env.reset()
total_reward = 0.0
while not done:
action = agent.get_action(obs)
next_obs, reward, done, info = env.step(action)
next_action = agent.get_action(next_obs)
agent.learn(obs, action, reward, next_obs, next_action)
obs = next_obs
total_reward += reward
if total_reward > best_reward:
best_reward = total_reward
print("Episode#:{} reward:{} best_reward:{} eps:{}".format(episode,
total_reward, best_reward, agent.epsilon))
Return the trained policy
return np.argmax(agent.Q, axis=2)
def test(agent, env, policy):
done = False
obs = env.reset()
total_reward = 0.0
while not done:
action = policy[agent.discretize(obs)]
next_obs, reward, done, info = env.step(action)
obs = next_obs
total_reward += reward
return total_reward
if __name__ == "__main__":
env = gym.make('MountainCar-v0')
agent = frozenlake(env)
learned_policy = train(agent, env)
# Use the Gym Monitor wrapper to evalaute the agent and record video
gym_monitor_path = "./gym_monitor_output"
env = gym.wrappers.Monitor(env, gym_monitor_path, force=True)
for _ in range(1000):
test(agent, env, learned_policy)
env.close()
LSP Python: ready
conda (Python

Output:


Spyder (Python 3.9)
/Users/anushagangasani/.spyder-py3/sarsa.py
temp.py
untitled0.py*
sarsa.py
Usage
/Users/anushagangasani/.spyder-py3/sarsa.py
Variable Explorer
Plots
Files
IPython console
History

3.


diffchecker.com/diff
Update
Editor
Compare & merge
Clear
Export as PDF
Save Diff
Share

4. https://github.com/gangasani-anusha/RL_Midterm.git

Problem 2 (20 pts):

Reference:

https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/

https://sjsu.instructure.com/courses/1529630/files/69364473?module_item_id=13628585

How does Batch Normalization work?

Batch Normalization is a process to make neural networks faster and more stable by adding extra layers in a DNN. It works in a two-step process. Initially, the input is normalized, and then offsetting, and rescaling is executed.

1. **Normalization of the Input:**

Normalization is a process of converting the data to get the mean as zero and the standard deviation as one. To calculate the mean of hidden activation:

$$\mu = \frac{1}{m} \sum h_i$$

Here m is the number of neurons at layer h.

Once the mean is obtained, let's calculate the standard deviation of the hidden activations.

$$\sigma = \left[\frac{1}{m} \sum (h_i - \mu)^2 \right]^{1/2}$$

We will normalize the hidden activations with the derived mean and standard deviation. The smoothing term(ε) satisfies the numerical stability within the operation. We normalize as follows:

$$h_{i(norm)} = \frac{(h_i - \mu)}{\sigma + \varepsilon}$$

2. **Rescaling of offsetting:**

The final step is to rescale and offset the input. Here we use two important components gamma and beta of the batch normalization algorithm. These parameters are used for shifting(beta) and re-scaling(gamma) of the vector from the previous operations.

$$h_i = \gamma h_{i(norm)} + \beta$$

These two parameters can be learned, and a neural network ensures that the best values are employed during training. This will make it possible to accurately normalize each batch.

What kind of problem does Batch Normalization solve?

- By normalizing the hidden layer activation, batch normalization speeds up the training process.
- BN solves the problem of internal covariate shift. By this, the input is distributed for every layer of mean and standard deviation. It helps by making the data flow between the layers of the neural network look, this means we can use a higher learning rate. It also has a regularizing effect which means we can remove dropout.
- BN smoothens the loss function which optimizes the model parameters to improve the training speed of the model.

Explain how BN works for the test data?

In Renormalization, the best practice of BN work for test data is to normalize the activations using the moving averages like mean and variance instead of mean and variance of minibatch. This depends only on a single input example instead of using a whole mini-batch.

$$y_{\text{inference}} = \frac{x - \mu}{\sigma} \cdot \gamma + \beta$$

Problem 3 (20 pts):

References:

https://sjsu.instructure.com/courses/1529630/files/69364531?module_item_id=13628595

https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

https://uu.diva-portal.org/smash/get/diva2:1162949/FULLTEXT02.pdf

We typically have three layers (Input, Hidden, and Output) depending on the image that is used for training convolutional neural networks. Here, we can actually change the processing's hidden layers to get rid of erroneous positive results. In this instance, executing these actions to enhance hidden layers after obtaining an image and applying CNN on it may eliminate or reduce the false positives.

To categorize it further in few steps:

- Altering the hidden layers in a rhombus format that is adjusting it in the descending order, where as for example it can be in the order of 726, 512, 256, 128 for the 1st to 4th hidden layer respectively.
- Increase the Epoch by 100 as more the iterations over training the more the efficiency could be increased.
- Increasing the dropout layer by 0.3 or training it by altering the percentage ranging from 0.2 to 0.9 that is 20% to 90%.
- If it comes to a complete dataset, it may be caused due to imbalanced data as well.
- In this case we can actually increase the F-score.
- If the images are TEM graded images that is low-resolution.
- Over-Sampling: Just like down-sampling but here we combine two or more classes to higher value average and training with the classifier again.
- Also lowering the NCC threshold may increase the number of candidates to analyze and automate the CNN again.
- In this case we can also down-sample the data: combining two different classes/image_features and reducing it to the minimum class and training with the classifier again.

As per these techniques/assumptions, there is a possibility to reduce the false positivity.

Remedies:

1) Cross Validation: train and test your model k-times on different subsets of training data and build up an estimate of the performance of a machine learning model on unseen data.

2 Regularization:

a) Max-Norm Regularization:for each neuron, it constrains the weights w of the incoming connections such that $\| w \|_2 \leq r$, where r is the max - norm hyperparameter and $\| \cdot \|_2$ is the $\ell 2$ norm. Reducing r increases the amount of regularization and helps reduce Overfitting. This can be

implemented in keras as :

b) L2 regularization:

you can use $\ell 2$ regularization to constrain a neural network's connection weights, and/or $\ell 1$ regularization if you want a sparse model (with many weights equal to 0). Here is how to apply $\ell 2$ regularization to a. Keras layer's connection weights, using a regularization factor of 0.01.

The l2() function returns a regularizer that will be called at each step during training to compute the regularization loss. This is then added to the final loss. As you might expect, you can just use keras.regularizers.l1() if you want $\ell 1$ regularization; if you want both $\ell 1$ and $\ell 2$ regularization, use keras.regularizers.l1_l2() (specifying both regularization factors)