Human-Centered Machine Learning

ISE 5194, Spring 2021

Professor Krening

Homework #4

## Problem 1:

Your task is to implement the tabular Q-learning algorithm for the Taxi domain.

<https://gym.openai.com/envs/Taxi-v3/>

Use the provided code to get you started. If you are using OpenAI gym locally on your computer, use *hw4.py*. If you are using Google Colab to write and run your code through a web browser, use *hw4.ipynb*.

A screenshot of a cell phone

Description automatically generated

Implement ε-greedy for your exploration method. Start with an ε of 0.95 and decay it through time to 0.01.

Deliverables:

1. Make a plot of the cumulative reward your agent earns each episode for the first 2000 episodes. Include this plot in your answer.

Chart

Description automatically generated

1. Make a plot of the exploration scalar (ε) for the first 2000 episodes. Include this plot in your answer.

Chart

Description automatically generated

1. After training, include the screen shot(s) of your trained agent exploiting its policy to solve the task. Did your agent learn a policy that efficiently solves the task?

|  |  |  |
| --- | --- | --- |
| A picture containing graphical user interface  Description automatically generated | Graphical user interface, application  Description automatically generated | Calendar  Description automatically generated with low confidence |

Looking at one run of a random initial environment, it looks like the taxi does a good job finding the minimum distance to the passenger and finding the minimum distance to the destination.

1. Copy your code here. If you are using Google Colab, download your code as a .py file and then copy your code here.

# Specify which environment to use.

env = gym.make("Taxi-v3").env

env.reset()

# Initialize table of Q-values

# Hint: to access a specific value in the q\_table, do this:

#            q\_table[state, action]

q\_table = np.zeros([env.observation\_space.n, env.action\_space.n])

##########################################

# Initialize RL Parameters

##########################################

alpha = 0.5

gamma = 0.6

epsilon = 0.95

# For plotting metrics

cumulative\_reward\_each\_episode = []

epsilon\_each\_episode = []

# For each episode

maxNumEpisodes = 2000

for i in range(maxNumEpisodes):

  # Reset to initial conditions

  state = env.reset()

  # The variable 'cumulative\_reward' will store the sum of the accumulated

  # reward for an entire episode. Set this value to zero at the start of each

  # episode.

  cumulative\_reward = 0

  done = False

  action\_num = 0

  # While the episode is not finished

  while not done:

    ##########################################

    # For every time step, using epsilon-greedy to choose between

    # exploration and exploitation.

    # Implement epsilon-greedy exploration.

    # Hint: to return a random action, do this:

    #           action = env.action\_space.sample()

    ##########################################

    num = random.random()

    if num < epsilon:

      action = env.action\_space.sample()

    else:

      action = np.argmax(q\_table[state])

        # Take the action.

        # This moves the agent to a new state and earns a reward

    next\_state, reward, done, info = env.step(action)

    # Add the reward just earned to the cumulative reward variable

    cumulative\_reward += reward

    ##########################################

    # Update your estimate of Q(s,a)

    # Hint: to access a specific value in the q\_table, do this:

    #            q\_table[state, action]

    ##########################################

    old\_val = q\_table[state, action]

    next\_max = np.max(q\_table[next\_state])

    new\_val = (1-alpha)\* old\_val + alpha \* (reward + gamma \* next\_max)

    #update table

    q\_table[state, action] = new\_val

    # Set your state variable to next\_state for the next loop.

    state = next\_state

    # If this episode is finished, take care of a few things:

    if done:

      # Save the cumulative reward from the previous episode to an array.

      cumulative\_reward\_each\_episode.append(cumulative\_reward)

      # Save the epsilon used in this episode.

      epsilon\_each\_episode.append(epsilon)

      ##########################################

      # Decay epsilon,

      # If you want to decay or change the value of epsilon at the end of

      # each episode, do so here.

      ##########################################

      epsilon = epsilon\*0.95

  if i % 100 == 0:

    print('Episode: {0}'.format(i))

print("Training finished.\n")

# Plot the Cumulative Reward and Epsilon value through time.

fsize = 15

# print('Max reward: ', max(cumulative\_reward\_each\_episode))

plt.plot(cumulative\_reward\_each\_episode)

plt.title('Cumulative Reward through Time', fontsize=fsize)

plt.xlabel('Episode', fontsize=fsize)

plt.ylabel('Cumulative Reward', fontsize=fsize)

plt.show()

plt.plot(epsilon\_each\_episode)

plt.title('Exploration (epsilon) through Time', fontsize=fsize)

plt.xlabel('Episode', fontsize=fsize)

plt.ylabel('epsilon', fontsize=fsize)

plt.show()

## Problem 2:

Your task is to implement the tabular Q-learning algorithm for the Cart-pole domain.

<https://gym.openai.com/envs/CartPole-v1/>

The Cart-pole domain has a continuous state space. In order to use tabular Q-learning on the Cart-pole domain, you will need to discretize the state space.

Use the provided code to get you started. If you are using OpenAI gym locally on your computer, use *hw4.py*. If you are using Google Colab to write and run your code through a web browser, use *hw4.ipynb*.

Deliverables:

1. Make a plot of the cumulative reward your agent earns each episode during training. Include this plot in your answer.

Chart, bar chart, histogram

Description automatically generated

1. Make a plot of the exploration scalar (ε) your agent used during training. Include this plot in your answer.

Chart

Description automatically generated

1. Why did we discretize the state space? What would have happened if we used tabular Q-learning but did not discretize the state space?
   * The state space needed to be discrete when using tabular q learning, as we needed to represent every possible state for every possible action. If we didn’t discretize the state space, the tabular q learning would not have any place to store and back propagate the values. One possible method if we couldn’t discretize the state space would be to use deep learning.
2. The reward for the Cart-pole domain is +1 at every time step. Normally, we have a small step cost (punishment) at each time step. Why is Cart-pole different? Is this a mistake?
   * In this case, the objective is to keep the pole from falling or the cart from moving too much which would terminate the simulation. The longer the episode the greater the reward. In other cases, taking a long time to complete a task was not ideal, so that’s why it received a penalty.
3. Copy your code here. If you are using Google Colab, download your code as a .py file and then copy your code here.

# Discretize input state to make Q-table and to reduce dimensionality

def discretize(state):

  #print ( state )

  # -3.4\*10\*\*38, , 3.4\*10\*\*38

  # First, set up arrays of the left bin edges

  # Note: your bin sizes do not need to be of uniform width.

  bins\_w = [-40, -30, -20, -10, -5, -2, 0, 2, 5, 10, 20, 30, 40]

  angle\_range = 0.43\*2

  num\_angle\_bins = 12

  angle\_bin\_stepsize = angle\_range / num\_angle\_bins

  bins\_ang = [-0.43]

  for i in range(1, num\_angle\_bins):

    bins\_ang.append( bins\_ang[i-1]+ angle\_bin\_stepsize )

  pole\_angle\_bin    = pd.cut([state[2]], bins=bins\_ang, include\_lowest=True)

  angle\_rate\_bin    = pd.cut([state[3]], bins=bins\_w  , include\_lowest=True)

  # To verify the order of the state variables:

  #   https://github.com/openai/gym/blob/master/gym/envs/classic\_control/cartpole.py

  return tuple([0, 0, int(pole\_angle\_bin[0].left), int(angle\_rate\_bin[0].left)])

# Specify which environment to use.

env = gym.make('CartPole-v0')

#env = wrap\_env(env)

state = env.reset()

##########################################

# Initialize your Q-values.

# Note: you may use whichever data structure you wish.

#       I used a dictionary, but a list works, too.

##########################################

# number of bins for each area of the state space

bins = (1, 1, 6, 13)

q\_table = np.zeros(bins + (2,))

print(np.shape(q\_table))

##########################################

# Initialize RL Parameters

##########################################

alpha = 0.5

gamma = 1

epsilon = 0.95

epsilon\_min = 0.05

# For plotting metrics

cumulative\_reward\_each\_episode = []

epsilon\_each\_episode = []

# To start off wish debugging your code, use 1 episode. Increase this once

# your code starts to work.

maxNumEpisodes = 500

# For each episode

for i in range(maxNumEpisodes):

  # Reset to initial conditions

  state = env.reset()

  ##########################################

  # Discretize the state

  # Note: you'll need to modify the discretize function

  #       provided above.

  ##########################################

  state = discretize2(state)

    # At the beginning of each episode, set the cumulative reward variable to zero.

  cumulative\_reward = 0

  done = False

  # For every step in the episode

  while not done:

    #env.render()

    ##########################################

    # For every time step, using epsilon-greedy to choose between

    # exploration and exploitation.

    # Implement epsilon-greedy exploration.

    # Hint: to return a random action, do this:

    #           action = env.action\_space.sample()

    ##########################################

    num = random.random()

    action = env.action\_space.sample()

    if num < epsilon:

      action = env.action\_space.sample()

    else:

      action = np.argmax(q\_table[state])

        # Take the action.

        # This moves the agent to a new state and earns a reward

    next\_state, reward, done, info = env.step(action)

    # Discrete the state

    next\_state = discretize(next\_state)

    # Add the reward just earned to the cumulative reward variable

    cumulative\_reward += reward

        ##########################################

    # Update your estimate of Q(s,a)

    ##########################################

    old\_val = q\_table[state][action]

    next\_max = np.max(q\_table[next\_state])

    new\_val = (1-alpha)\* old\_val + alpha \* (reward + gamma \* next\_max)

    #update table

    q\_table[state, action] = new\_val

    state = next\_state

    # If the episode is finished, do a few things.

    if done:

      # Save the cumulative reward from the previous episode to an array.

      cumulative\_reward\_each\_episode.append(cumulative\_reward)

      # Save the epsilon used in this episode.

      epsilon\_each\_episode.append(epsilon)

      ##########################################

      # Decay epsilon.

      # If you want to decay or change the value of epsilon at the end of

      # each episode, do so here.

      ##########################################

      epsilon = max(epsilon\*.99, epsilon\_min)

  if i % 100 == 0:

    print('Episode: {0}'.format(i))

env.close()

#show\_video()

print("Training finished.\n")

# Plot the Cumulative Reward and Epsilon value through time.

fsize = 15

plt.plot(cumulative\_reward\_each\_episode)

plt.title('Cumulative Reward through Time', fontsize=fsize)

plt.xlabel('Episode', fontsize=fsize)

plt.ylabel('Cumulative Reward', fontsize=fsize)

plt.show()

plt.plot(epsilon\_each\_episode)

plt.title('Exploration (epsilon) through Time', fontsize=fsize)

plt.xlabel('Episode', fontsize=fsize)

plt.ylabel('epsilon', fontsize=fsize)

plt.show()

1. For extra credit, include a video of your trained agent balancing the Cart-pole.

## Problem 3:

1. Define exploration and exploitation. Is one better than the other?
   * Exploration is the algorithm making random decisions to better understand the reward in different areas of the state space.
   * Exploitation is the algorithm taking the path that has the highest reward for what it knows. There still may be a better path to reward in the state space, but this is maximizing what the algorithm currently knows.
   * One is not better than the other, as they satisfy two different objectives. Exploration is needed to find higher rewards, while exploitation allows the algorithm to complete the desired task. Ideally a model will incorporate both.
2. What is the reward hypothesis? If you improperly define a reward function, which leads to your RL agent exhibiting unexpected behavior, are you violating the reward hypothesis?
   * The reward hypothesis is the maximization of the expected value of all rewards or signals. Defining different signals can lead to unexpected results, though this is not the fault of the reward hypothesis, but it is more so the fault of the user. The user should modify the signals of the state space in order to get a more appropriate model of the state space, though there are still instances where the model will do something unforeseen.
3. Q-learning parameters:
   1. What is alpha?
   * Alpha is the learning rate, which is a means to weight past experience and future rewards in the Q learning equation.
   * Q: What range of numbers can alpha be?
     + Alpha is between 0 and 1 inclusive.
   * Q: What happens if alpha is large?
     + As alpha approaches 1, the equation will take into account the current and expected future reward over the previous values of actions and rewards.
   * Q: What happens if alpha is small?
     + As alpha approaches 0, the equation will emphasize previous rewards and actions over the current and expected future rewards.
   * Q: Should alpha change through time?
     + Alpha doesn’t change over time.
   1. What is gamma?
   * Gamma is a variable that helps with discounting rewards in the future. This helps the model with still having long term rewards to consider but having a main priority on more immediate signals.
   * Q: What range of numbers can gamma be?
     + Gamma can range from 0 to 1 inclusive.
   * Q: What happens if gamma is large?
     + If gamma is 1, then all rewards in the sequence become additive, where the time or sequence of the reward doesn’t matter, and they are all treated equally.
   * Q: What happens if gamma is small?
     + If gamma is 0, then the only non-zero reward is the immediate reward in the sequence, and the model will exploit the immediate reward and not consider future rewards.
   * Q: Should gamma change through time?
     + No, gamma should not change over time, the function handles the decay of the signal farther out in the sequence.
   1. What is epsilon?
   * Epsilon is a threshold that determines if the next action will be one of exploration or exploitation.
   * Q: What range of numbers can epsilon be?
     + Epsilon can be between 0 and 1 inclusive.
   * Q: What happens if epsilon is large?
     + As epsilon approaches 1, the model will take more exploration actions and less exploitation actions.
   * Q: What happens if epsilon is small?
     + As epsilon approaches 0, the model will take a tendency to exploit as opposed to exploring.
   * Should epsilon change through time?
     + Yes epsilon should change over time to reflect the model’s change from more exploration to more exploitation. This can be represented with a decay.
4. List at least 5 potential ways human input can be incorporated into an RL algorithm.

* Human input can be used as an order. In the example of a grid, a human can tell the RL algorithm to go right.
* Human input can be used as critique. In the example of a grid, a human can tell the RL algorithm if the action taken was good or bad.
* Human input can be used an explanation. In the example of mario, a user can tell the model to jump in order to collect coins.
* Human input can be used as a demonstration. A user could complete a level of mario and give it to the RL model as data.
* Human input can be used to shape policies. In the example of a butler robot, the user could create a policy to handwash a select number of items and to use the dishwasher for the rest of the items.

## Problem 5: Extra Credit – RL with continuous variables measured continuously

This problem is not required, but will be worth extra credit.

Your task is to use a non-tabular RL algorithm to train an agent on the Cart-pole domain. You will need to choose a non-tabular RL method such as Q-learning with Function Approximation. You do not need to implement the RL algorithm from scratch; there are baseline algorithms that you can use.

If you do a Google search for “openai gym baselines” you will find the following links:

* <https://stable-baselines.readthedocs.io/en/master/>
* <https://github.com/hill-a/stable-baselines>
* <https://github.com/openai/baselines>

You are welcome to choose any baseline (including deep learning algorithms) so long as they are not tabular. The reason for this is I want you to work with continuous variables.

Note: **You will NOT receive extra credit if you discretize the state space.**

## Information on the Taxi Domain:

The Taxi Problem - from "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition" by Tom Dietterich

**Description**:

There are four designated locations in the grid world indicated by R(ed), G(reen), Y(ellow), and B(lue). When the episode starts, the taxi starts off at a random square and the passenger is at a random location. The taxi drives to the passenger's location, picks up the passenger, drives to the passenger's destination (another one of the four specified locations), and then drops off the passenger. Once the passenger is dropped off, the episode ends.

**Observations (State)**:

There are 500 discrete states since there are 25 taxi positions, 5 possible locations of the passenger (including the case when the passenger is in the taxi), and 4 destination locations.

|  |  |
| --- | --- |
| **Passenger locations:**  - 0: R(ed)  - 1: G(reen)  - 2: Y(ellow)  - 3: B(lue)  - 4: in taxi | **Destinations**:  - 0: R(ed)  - 1: G(reen)  - 2: Y(ellow)  - 3: B(lue) |

**State space** is represented by:

(taxi\_row, taxi\_col, passenger\_location, destination)

**Actions**:

There are 6 discrete deterministic actions:

- 0: move south

- 1: move north

- 2: move east

- 3: move west

- 4: pickup passenger

- 5: dropoff passenger

**Rewards**:

There is a reward of -1 for each action and an additional reward of +20 for delivering the passenger. There is a reward of -10 for executing actions "pickup" and "dropoff" illegally.

**Rendering**:

- blue: passenger

- magenta: destination

- yellow: empty taxi

- green: full taxi

- other letters (R, G, Y and B): locations for passengers and destinations

## Information on the Cartpole Domain:

<https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py>

**Description**:

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart's velocity.

**Source**:

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson.

**Observation (State)**:

Type: Box(4)

**Num Observation Min Max**

0 Cart Position -4.8 4.8

1 Cart Velocity -Inf Inf

2 Pole Angle -24 deg 24 deg

3 Pole Velocity At Tip -Inf Inf

Instructor Note: The values that constitute the state space are continuous variables. In order to use tabular Q-learning effectively, we need to discretize the state space variables. Do you remember when we were talking about Data Types earlier in the semester and we talked about variables that have an underlying continuous nature (e.g. Temperature) but are measured ordinally (low-resolution continuous) or nominally (artificially reduced dichotomous)? We’re going to do the same thing here. Treat these continuous state variables as ordinal data and put them into ‘bins’.

**Actions**:

Type: Discrete(2)

Num Action

0 Push cart to the left

1 Push cart to the right

Note: The amount the velocity that is reduced or increased is not fixed; it depends on the angle the pole is pointing. This is because the center of gravity of the pole increases the amount of energy needed to move the cart underneath it.

**Reward**:

Reward is 1 for every step taken, including the termination step

**Starting State**:

All observations are assigned a uniform random value in [-0.05..0.05]

**Episode Termination**:

Pole Angle is more than 12 degrees

Cart Position is more than 2.4 (center of the cart reaches the edge of the display)

Episode length is greater than 200

**Solved Requirements**

Considered solved when the average reward is greater than or equal to 195.0 over 100 consecutive trials.