Alexander Mervar

Professor Brown

COGS-Q 355

4.28.2022

Final Project – Learning to Code Reinforcement Q-Learning

Introduction

When initially beginning this project, I'll be the first to admit that my original scope was overly ambitious. At first, I wished to create a reinforced learning (RL) network to be utilized within a novel RL environment of my own design. I looked forward to creating simulated Braitenberg vehicles with continuous action spaces and having them track a stationary light source without any direct instruction from their creator. (I still look forward to building this environment.) But, without any knowledge of Q-learning and deep Q-learning outside of the theory and material discussed in class. It became rapidly apparent that the bounds of this project exceeded my capabilities if I was going to accomplish my task in time.

Because of this, I took a step back and decided to create a deep Q-learning neural network fit for a much simpler RL environment. In my research I found an extremely useful guide into Q-learning, unique environments, and deep RL with Q-learning using a simpler question and problem space. This simpler environment was designed by Sentdex, who acted as a virtual guide for the sharpening of my skills and the betterment of this project. Although there are plenty of resources online on RL, I found that many of the tutorials and lessons were flawed by being mutually dependent. Many bugs and outdated information

were across multiple resources due to the fact that there was a lot of code sharing/stealing from what I could find. Therefore, I chose these tutorials and lessons to be the best avenue for my education.

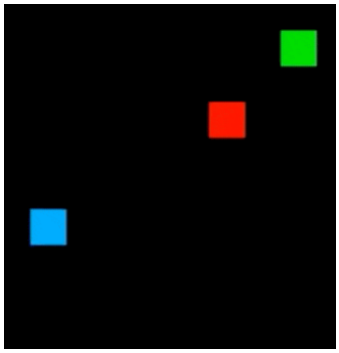


Figure 1: A demonstration of the 10x10 gridworld I used as a minimum viable product to learn Q-learning and deep Q-learning programming skills

This 10x10 grid therefore acted as the minimum viable product to teach myself the programming skills necessary to tackle the future task of designing my own RL environment involving simulated Braitenberg vehicles with discrete and continuous action spaces. I looked forward to answering the questions:

- How do I create unique environments that can be used to train RL models?
- How is Q-learning accomplished without the benefits of a deep neural network?
- In what way must an environment be modified to be able to implement a deep neural network rather than a brute force Q-learning algorithm

Because of this project, I was able to answer all these questions. I feel confident that with these skills, I will be able to accomplish my original task of creating a unique environment with simulated Braitenberg vehicles in my future research in the Fall semester of 2022.

Methods

My project can be broken into three distinct chunks. I began by doing a literature review and designing a Python class that could act as the basis for this particular project. Unfortunately, following the completion of this literature review and introductory programming, I realized that I would need to be reliant on some sort of external guide/tutorial if I was to learn how to synthesize what I have learned in class with the novel world of programming and deep neural network for Q-learning.

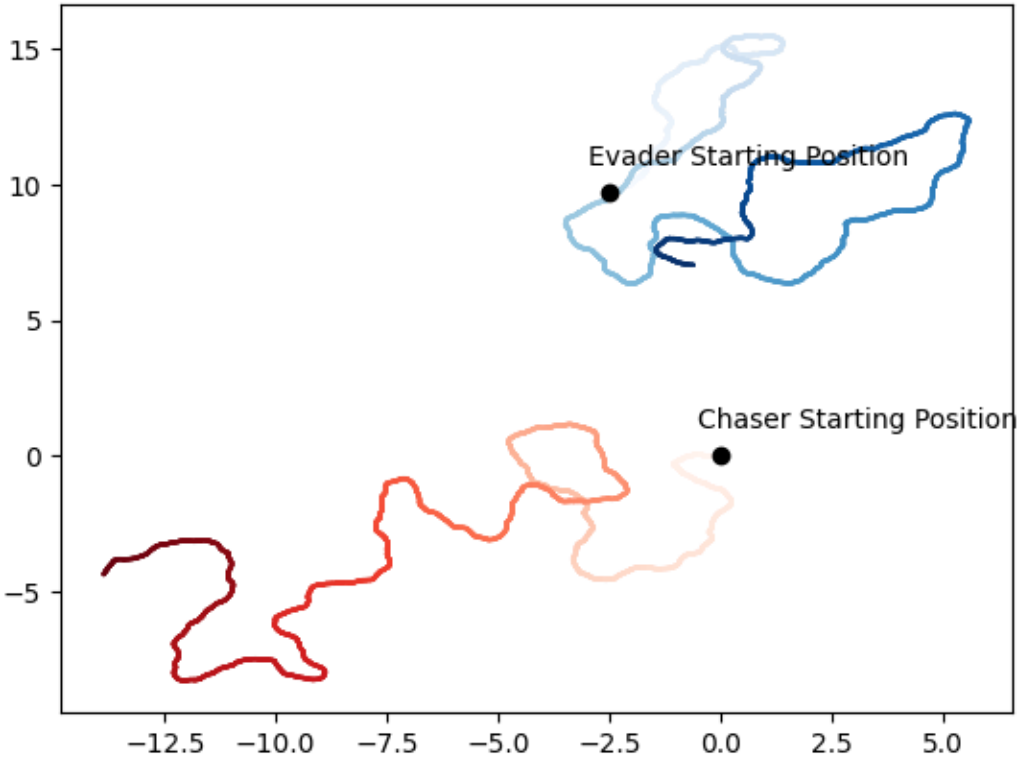
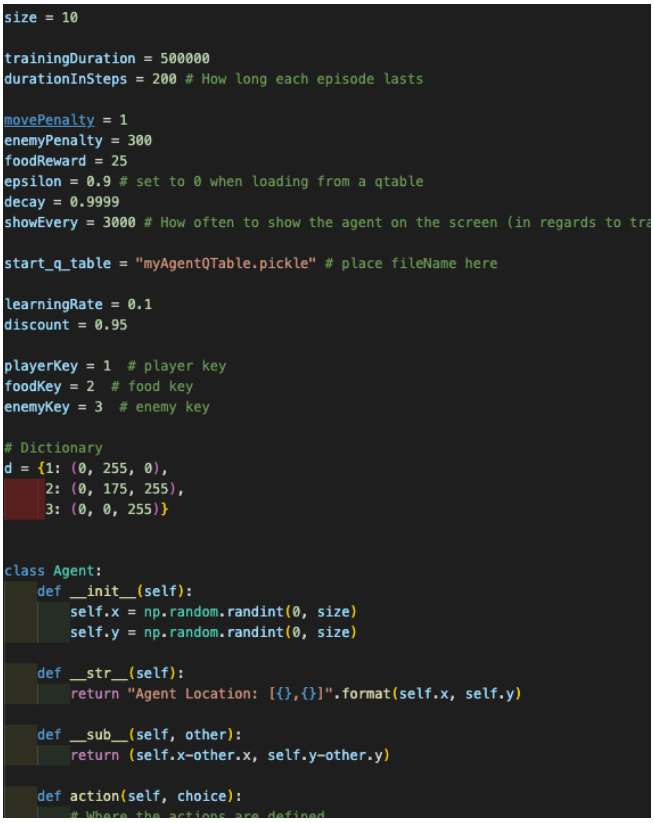


Figure 2: A generated graphic from my initial work involving Braitenberg vehicles at the start of this project (originally, I planned on making this project an evasion/chase simulation between two active agents)

Second, I followed Sentdex's tutorial (https://pythonprogramming.net/q-learning-reinforcement-learning-python-tutorial/) on Q-learning and designed a 10x10 gridworld with their instructions to master the basics of Q-learning without the added challenge of

implementing a deep neural network. This codebase uses RL learning but instead of utilizing a neural network, it uses a Q-table to exhaustively learn every possible coordinate value for the agent, a goal (food), and a (stationary) adversarial agent. Training this model exhaustively takes less time than a deep neural network. (It takes around ½ hour to train this model with 20,000 training iterations. Running the deep neural network locally gave me an estimate of AT LEAST 4 days to complete 20,000 training iterations.) This second codebase is where I did the most tinkering and learning about agent behavior. Although initially I was just following the tutorial and not adding anything unique, upon completion I modified multiple variables and methods by increasing/decreasing the discrete action space, changing the size of the environment, allowing for the food/adversary to move randomly with the 2d plane, and modifying the learning constants described at the top of the file. (You can find this file within the *BRUTEFORCEGridworld* directory of my project submission.)



```
size = 10

trainingDuration = 500000
durationInSteps = 200 # How long each episode lasts

movePenalty = 1
enemyPenalty = 300
foodReward = 25
epsilon = 0.9 # set to 0 when loading from a qtable
decay = 0.9999
showEvery = 3000 # How often to show the agent on the screen (in regards to tra

start_q_table = "myAgentQTable.pickle" # place fileName here

learningRate = 0.1
discount = 0.95

playerKey = 1  # player key
foodKey = 2  # food key
enemyKey = 3  # enemy key

# Dictionary
d = {1: (0, 255, 0),
     2: (0, 175, 255),
     3: (0, 0, 255)}


class Agent:
    def __init__(self):
        self.x = np.random.randint(0, size)
        self.y = np.random.randint(0, size)

    def __str__(self):
        return "Agent Location: [{},{}]".format(self.x, self.y)

    def __sub__(self, other):
        return (self.x-other.x, self.y-other.y)

    def action(self, choice):
        # Where the actions are defined
```

Figure 3: A screenshot of the brute force Q-learning code I wrote with the help/instruction of the resources I found online

Finally, I followed Sentdex's tutorial on how to take this gridworld and implement deep Q-learning using a neural network. (You can find the corresponding files within the *DeepQLearning* directory of my project submission.) Admittedly, due to the complexity of this section of my project, I was too constrained on time for experimentation with the original codebase. Between debugging outdated code, investigating sections of the code I didn't understand well, and trying to get a functional neural network to run in order to fit the requirements of this project, I was unable to find time to add something of my own to this section of my project. Despite this, I would like to share that this was the most educational section of my project for me, and I'd like to share some of the findings I made while undergoing this particular task.

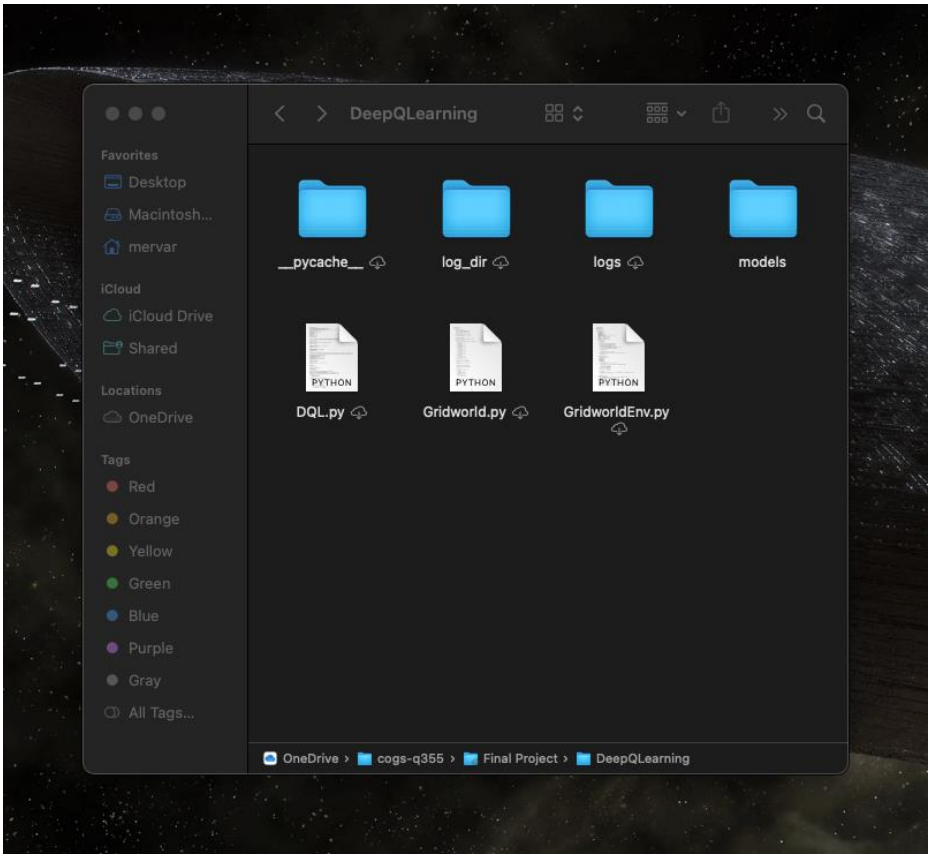


Figure 4: A screenshot of the different code and directories I spent the majority of my time in. Here, you can see the three classes I designed to hold the separate classes needed in order to create a neural network inside of the Sentdex RL environment.

Results

```python
def createModel(self):
    if LOAD_MODEL is not None:
        print(f"Loading model from {LOAD_MODEL}")
        model = load_model(LOAD_MODEL)
        print(f"Loaded model from {LOAD_MODEL}")
    else:
        model = Sequential()

        model.add(Conv2D(256, (3, 3), input_shape=env.OBSERVATION_SPACE_VALUES))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))

        model.add(Conv2D(256, (3, 3)))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))

        model.add(Flatten())
        model.add(Dense(64))

        model.add(Dense(env.ACTION_SPACE_SIZE, activation='linear'))
        #model.compile(loss="mse", optimizer='adam'(lr=0.001), metrics=['accuracy'])
        model.compile(loss="mse", optimizer='adam', metrics=['accuracy'])
    return model
```

Figure 5: A screenshot of the deep RL model that was used in my project

One thing to note is that although I was able to create these models and learn so much through this project, due to the limitations of my own hardware, I was unable to train the deep neural network since the estimated time to complete the training was over 4 days. I'd love to share what I did learn through this project.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

Figure 6: The Q-learning loss function I used in this project

Something that stands out to me now looking back on this project is just how much I was overcomplicating this process in lecture. During class, connecting the pieces of reinforcement learning was very abstract to me and I had no idea how this work was really accomplished. To clarify, I do not want to belittle the complexity of this work. Using supplemental packages like Keras and Tensorflow are beneficial in giving me an understanding that I never would have had before this project.

When looking at the learning rate of the brute force model, I was impressed with the speed it was able to accomplish its task.

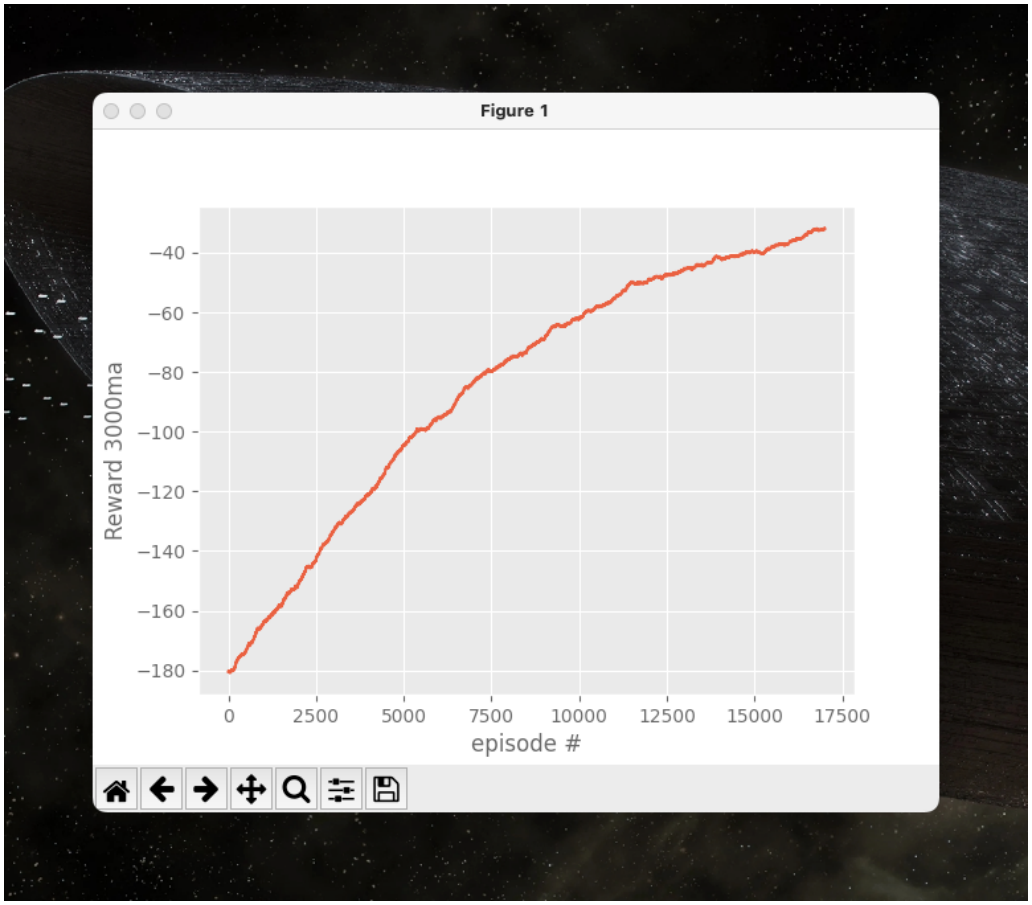


Figure 7: The learning of the brute force model over the course of training episodes

As you can see from figure 7, the model I was able to create that implements Q-learning without a deep neural network is able to have satisfactory performance within about 5,000 episode iterations.

Whenever judging results of a particular model, I was taking into effect computational power, processing time, as well as performance. Although the brute force model was functional in a short amount of time, this model used a considerably larger amount of memory and its methodology in learning wasn't necessarily smart but heuristic in its evaluation of the given inputs and then made an action based on it's predefined recommend action, which would be altered due to any given reward (or negative reward).

Discussion

Looking back on this project, I'm thoroughly satisfied with the work I was able to create and accomplish. Following this work, I plan to continue and complete my original goal of creating a unique RL environment for simulated Braitenberg vehicles through research with Professor Izquierdo.

Looking at the literature, I can see that my models do in fact create parody with the models I was looking to simulate. As seen in figure 5, the model structure for the deep neural network is similar to the model designed in the 2015 paper by Mnih et al.

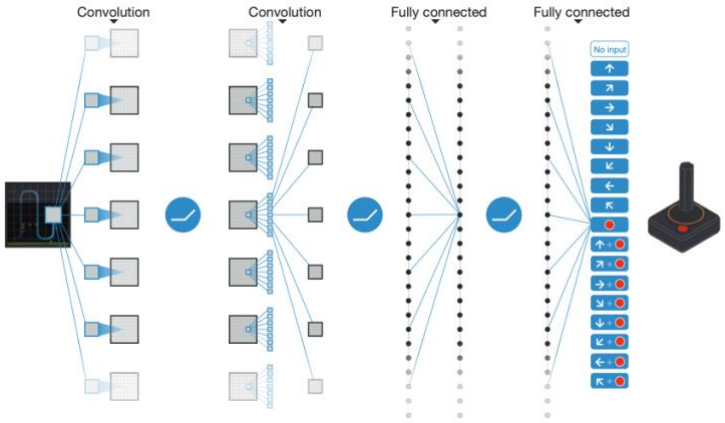


Figure 8: The deep neural network layed out by Minh et al. in their 2015 reinforcement learning paper

Overall, the strength of this work can be found in its similarity to the published literature and theory of reinforcement learning, which we have covered in class. Following this project, I feel that I am ready to take on novel environments of my own design but, I still am unsatisfied with a few flaws in the project at this current juncture. I wish that I was able to have access to hardware (or remote hardware) that was able to train the deep neural network within a reasonable timeframe. Because I know now just how much processing power this requires, I'm currently looking into getting access to super computers through IU and other institutions so that I am able to continue my research and learning futher.

Having that kind of power would allow me to answer some deeper unanswered questions that this project has been unable to answer for me.

If I could do this project over, I'd start in early January. Although that seems quite strange based on the course and its pace, I know now that resources online and my already maturing programming skills could have been utilized in order to make more significant progress in the areas, I wanted using online tutorials and forums. I look forward to taking this work to the next level.

In regard to the biological plausibility of this model, as we've discussed in class, this kind of reinforcement learning through reward is incredibly similar to the way that humans undergo reinforcement learning through the release of dopamine when given an initial stimulus. Like the anecdotal man passing a bar, my model can look at a gridworld and act on the firing of a reward to make decisions that will lead to the model being rewarded through the capture of a simulated goal/food.

Works Cited

Braitenberg, Valentino. *Vehicles: Experiments in synthetic psychology*. MIT press, 1986.

Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *nature* 518.7540 (2015): 529-533.

Sentdex. "Q-Learning Introduction and Q Table - Reinforcement Learning w/ Python Tutorial." *Python Programming Tutorials* https://pythonprogramming.net/q-learning-reinforcement-learning-python-tutorial/.