REINFORCEMENT LEARNING FINAL ASSIGNMENT

Custom OpenAI Gym Environment - Shower Agent

Runcong Wu

CS9170 Reinforcement Learning | April 23, 2021

Table of Contents

1. Overview

1.1 Motivation

Machine learning (ML) is a hot topic these days, it is widely used in industries and research articles. In previous courses, I learned about different algorithms in supervised learning and unsupervised learning. As a student who is interested in data science and data analytics, I feel it is necessary to learn more about ML. Although supervised learning and unsupervised learning provide a data-driven approach to make accurate decisions for us, they cannot replace human behaviours and interactions. A famous quote from statistician George Box (1976), "all models are wrong, but some are useful". Indeed, all models are simplified version of the reality until Reinforcement Learning (RL) came out. The idea behind RL is to mimic human brain to take actions based on the rewards. RL is something new in the field, and I can see the potential of using RL in our daily life. This assignment gives me an opportunity to apply the knowledge I learned into something useful.

Taking a shower is one of the greatest things to relax. However, it is tricky to adjust the temperature to an optimal range. Firstly, the dial itself is unreliable. "I usually like the dial turned to about here". This is something we all do prior to take the shower. Even we have perfect insight into the degree of dial rotation, the water temperature changes under different situations. For example, your girlfriend just finished a 40 minutes shower, and she used every hot water in the house. If you have the expectation of taking a hot shower, you

may be disappointed that the water is still cold even though you turn the dial to the maximum. So, you have to wait for a while to enjoy the shower. Secondly, we might shift the water temperature based on our internal body temperature. Your body temperature after workout is different than your body temperature when you just wake up (maybe not a huge difference because the homeostasis concept from physiology class LOL). Third, there may be random events that interrupt the water temperature. For instance, someone is doing laundry; at the same time, the dishwasher is on for some reason. Furthermore, there is seasonality effect on water temperature. Hot water tends to be more available in the summer and less available in the winter. What if we can use RL agent to adjust the temperature for us? That would be great!

1.2 Human-level Performance

Before I build the Shower Agent and do experiments, I would like to get a sense of how our agent are experiencing with the tasks. I decide to run through a couple episodes on my own and see how well I can handle the task. This will serve as a benchmark for my agent.

So, the objective for me is to know how long it takes me to reach a comfortable water temperature so that I can enjoy the shower. I bring my timer and my bare hand to run the episodes, each episode is several hours apart because I would like to reduce the correlation between them. If you start an episode right after the previous run, the water temperature will be higher than usual, it is like cheating. I average the result after 6 episodes. I am surprised that it takes me around 33 seconds to prepare the hot water for me. I thought I can do it a lot faster, for example, in 5 seconds, but that is not the case.

2. Technical Description

2.1 Normal Version MDP

State Space

- **Water temperature:** it will be initialized randomly around 25 degrees with noise at each episode (25 + random.randint(-3,3)).
- **Time limit:** the total shower length is set to be 180 (timesteps/seconds).

Action Space

- **Turn up the dial:** increase water temperature by 1
- **No change on dial:** doesn't change the water temperature
- **Turn down the dial:** decrease water temperature by 1

Environment

- Initial actual water temperature is noisy (see state space).
- Temperature noise in each episode: random.randint(-1,1)

Reward

- +1 reward if the water temperature is within the range of [36, 39] degree.
- -1 reward if the water temperature is out of the range of [36, 39] degree.

Indication of the agent has 'solve' the environment

- An average reward ≥ 100: our water temperature is within the range of [36, 39] degree in most timesteps, except at the beginning when the water temperature has been reset.

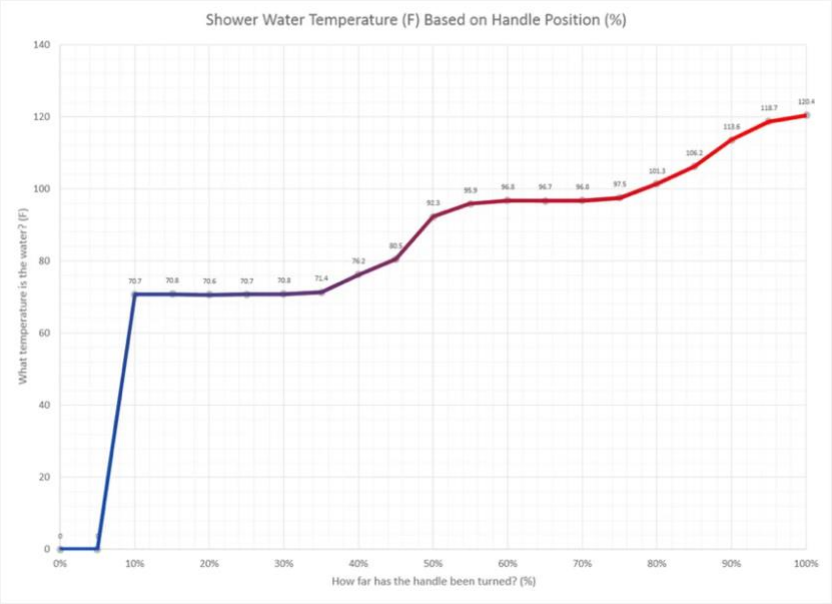
2.2 Hard Version MDP

State Space

- **Perceived water temperature:** it will be initialized randomly around 20 degrees with noise at each episode (20 + random.randint(-5,5)).
- **Perceived internal skin temperature (hidden from agent):** it will be initialized randomly around 38 degrees with noise at each episode (38 + random.randint(-3,3)).
- **Difference between water temperature and body temperature:** since the skin temperature will change according to the change in water temperature, I decide to give a coefficient of 0.05 to update the skin temperature. For example, if the water temperature is increased by 20 degrees; skin temperature will increase by 20*0.05 = 1 degree.
- **Time limit:** the total shower length is set to be 180 (timesteps/seconds).

Action Space

- The action will change the water temperature. Since the relationship between water temperature & action on shower dial is non-linear. I tried to use different slopes to implement non-linearity (e.g., water temp <= 30, slope = 3; else, slope = 0.5).
- **Turn up the dial:** increase water temperature by slope*1
- **No change on dial:** doesn't change the water temperature
- **Turn down the dial:** decrease water temperature by slope*1



Environment

- Initial actual water temperature is noisy (see state space).
- Initial internal skin temperature is noisy (see state space).
- Temperature Noise: 3 Random temperature change events that change the water temperature. In each episode, 80% chance of first event happening, 18% chance of second event happening, 2% chance of third event happening.
    - First event: random.randint(-1,1)
    - Second event: random.randint(-2,2)
    - Third event: random.randint(-3,3)

Reward

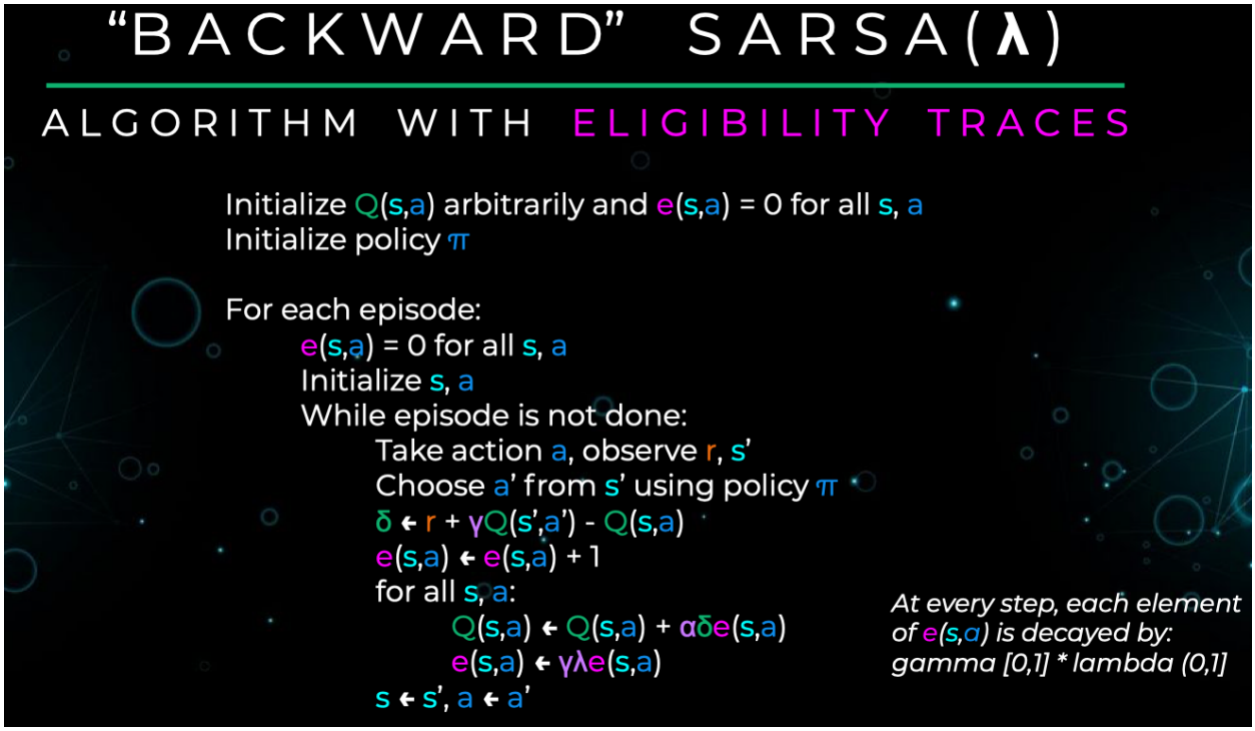
- +1 reward if the skin temperature is within the range of [36, 39] degree.
- -1 reward if the skin temperature is out of the range of [36, 39] degree.

Indication of the agent has 'solve' the environment

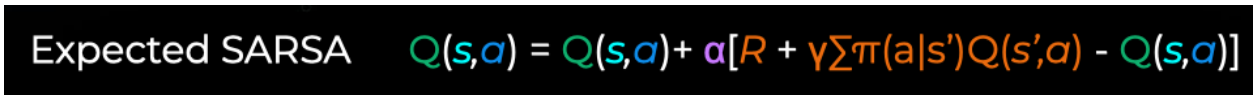
- An average reward ≥ 100: our skin temperature is within the range of [36, 39] degree in most timesteps, except at the beginning when the skin temperature has been reset.

2.3 Algorithms

2.3.1 Tabular Method – Expected SARSA (λ) with Eligibility Traces



Changing SARSA to Expected SARSA:

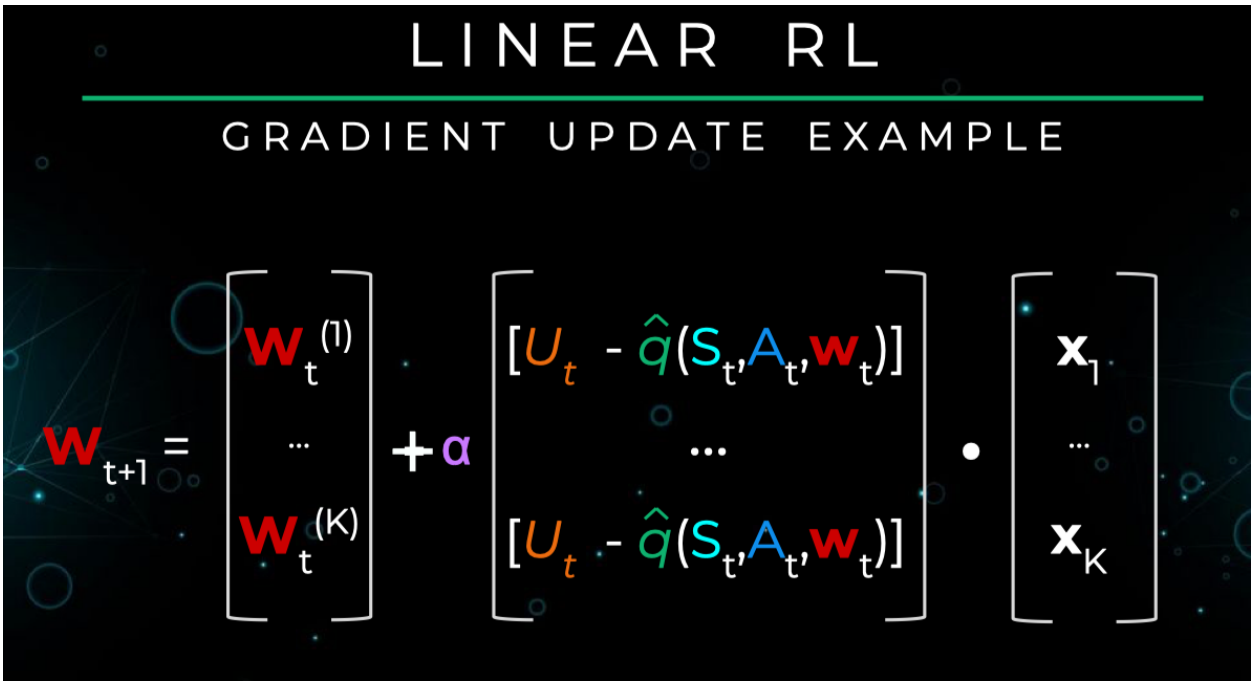


I would like to use the backward approach of Expected SARSA (λ) with Eligibility Traces because it is very powerful to solve complex environment. The eligibility trace can help us update the state action pair more effectively; the more recent and frequent the state action pair

you visit, the larger the weights. But it will start decaying as you move to the next state. Just like you mentioned in the lecture, Hansel and Gretel were laying down the breadcrumbs (eligibility trace) as they go so that they don't get loss in the woods. But the breadcrumbs were decaying for every state action pair you visited in the past. The ones that closest to you are fine, the ones that further away from you will decay to nothing. I choose to

use Expected SARSA because it is a nice combination of SARSA and Q-Learning. It gets all the safe conservatism from SARSA along with the power of Q-Learning, but without the large variance of Q-Learning.

2.3.2 Function Approximation Method – Q-Learning

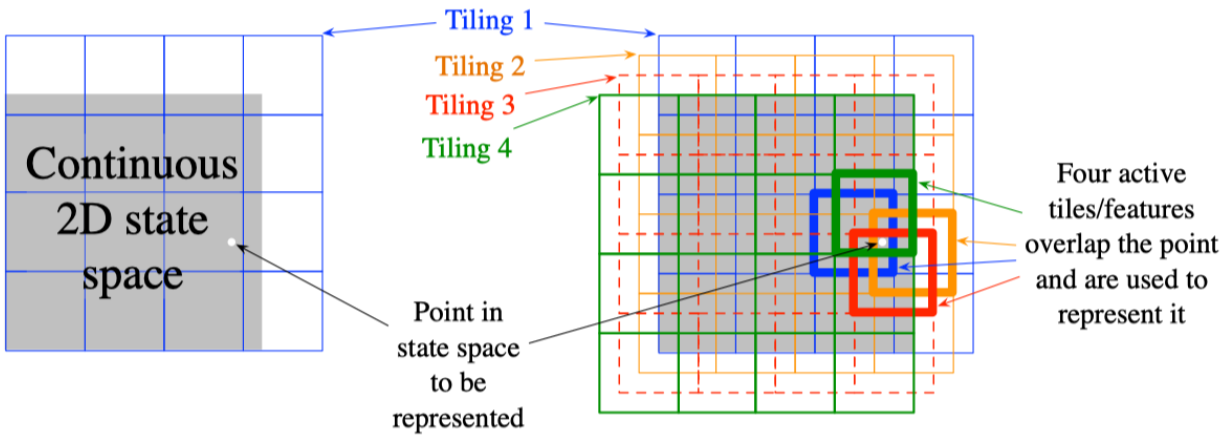


Function approximation method is powerful when the discrete state space is too large, state space is continuous, or action space is continuous. Although the shower environment has small state space and action space, I would like to compare with the tabular method and see which one fit the environment better.

The idea behind function approximation is using gradient descent to update the weights. The weights are the coefficients of each feature/state, we can use the weights to find the action-value estimates. This is exactly what we do in supervised learning; we find the best coefficients by minimizing the loss function iteratively.

2.3.3 Tile Coding Method

Linear methods guarantee convergence and can be very efficient in terms of both data and computation; however, the linear form of the features cannot take into account any interactions between features. Choosing a correct feature representation is critical for our agent to solve complex environment. The Tile coding is the most practical feature representation for modern sequential digital computers.



The features are grouped into partitions of the state space. Each partition is called a tiling, and each element of the partition is called a tile. Tile coding also gains computational advantages from its use of binary feature vectors, each component is either 0 or 1. One-hot encoding is used to implement the binary feature vectors.

3. My Experience

3.1 Struggle with the Environment

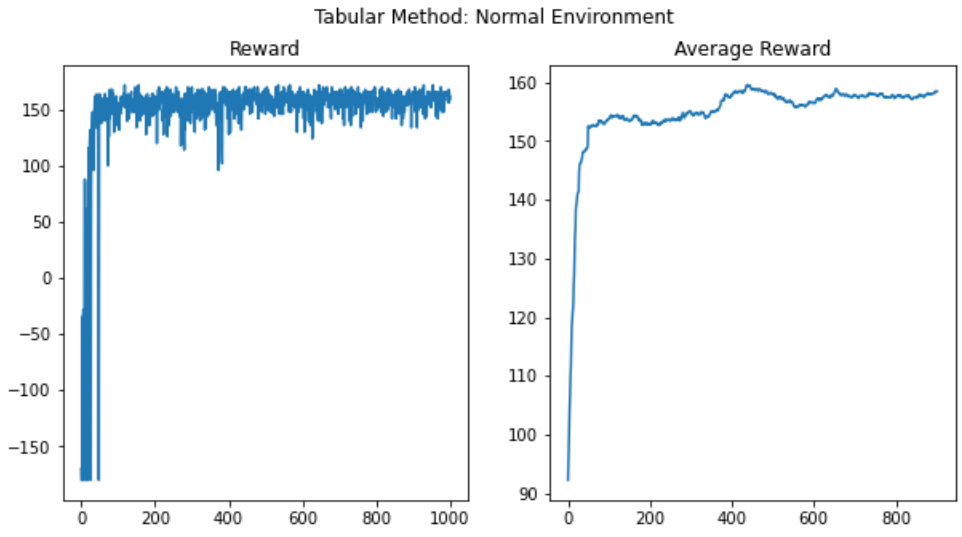
Creating a custom OpenAI Gym Environment is my first obstacle because it is my first time implementing RL environment using gym. Although I worked on FrozenLake and Taxi Environments in the previous assignments, I struggled to build the environment. I try to look at some example source code on OpenAI Gym webpage and figure out the patterns. To implement a custom environment, I need to write a subclass that inherits from gym.Env superclass. There are several key methods to begin with. Firstly, in the constructor, I need to initialize the action space, observation space, and timesteps. Secondly, the step method will provide the reward and state information in one timestep in each episode. Next, the reset method will reset the environment to the initial state. Once the agent finishes each episode, the environment will reset itself. Last one is the render method, I didn't write anything; however, I would love to implement a visualization in the future.

```
import gym
from gym import spaces

class CustomEnv(gym.Env):
  """Custom Environment that follows gym interface"""
  metadata = {'render.modes': ['human']}

  def __init__(self, arg1, arg2, ...):
    super(CustomEnv, self).__init__()


    # Define action and observation space
    # They must be gym.spaces objects


    # Example when using discrete actions:
    self.action_space = spaces.Discrete(N_DISCRETE_ACTIONS)


    # Example for using image as input:
    self.observation_space = spaces.Box(low=0, high=255, shape=
                    (HEIGHT, WIDTH, N_CHANNELS), dtype=np.uint8)

  def step(self, action):
    # Execute one time step within the environment
    ...

  def reset(self):
    # Reset the state of the environment to an initial state
    ...

  def render(self, mode='human', close=False):
    # Render the environment to the screen
    ...
```

3.2 Struggle with the Tabular Method

Fixing Mistake: taking the timestep as the state

For the tabular method, the data structure to store the action value is a matrix with dimension number of states * number of actions. I need an index (integer) to retrieve the action value estimates for the given state. But the state (water temperature) in ShowerEnvHard is a floating number and it cannot be used as the index for the action value matrix. My initial approach (wrong approach) is to use the timestep as the index, but that didn't make sense because the agent can visit a state multiple times and learn from the previous experience. Also, it is not reasonable to assume the timestep is the state, so I tried to convert the state from a float to an int by doing "next_state = int(round(next_state,0))". It works, and the agent is able to solve the normal environment.



But the agent still suffers to learn in the hard version. So, I did some experiment to investigate this issue. Then I found out it is the issue from my custom environment. ; (

Environment Update: swap the water temperature and skin temperature

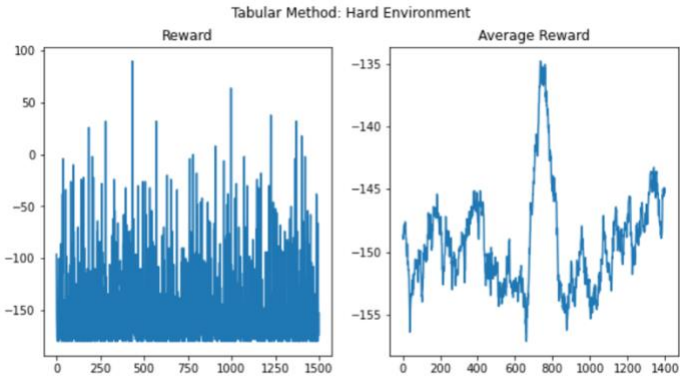
Since the hard version environment has two temperature, I have to decide which temperature should be hidden from the agent. My initial approach is to let the agent know the skin temperature and make the water temperature hidden from the agent. But the tabular method is not working, I started to reconsider the environment design.

Back to the environment design, the action of turning the dial change the water temperature, then it will have an effect on our skin temperature. The goal is to keep the skin temperature within the optimal range, agent will get positive reward if the goal is reached.

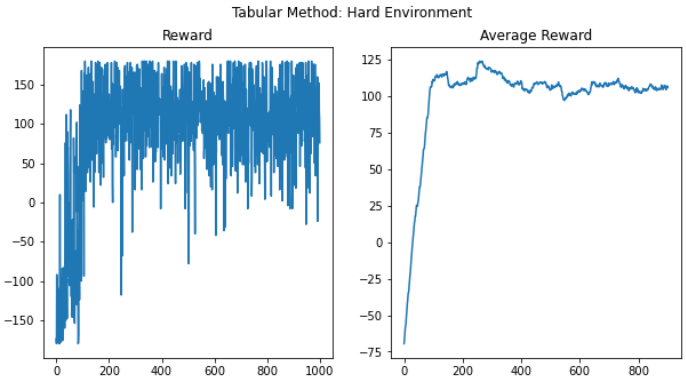
$$Action \rightarrow Water\ Temperature \rightarrow Skin\ Temperature$$

$$Goal: keep\ the\ skin\ temperature\ within\ the\ optimal\ range$$

Since the agent can feel the water temperature using its virtual "hand", it is reasonable to give water temperature to the agent instead of skin temperature. Also, the water temperature is directly related to the action. So, I re-run the program and the get significant better results.



Before swapping



After swapping

The agent successfully solves the hard environment! ;)

3.3 Struggle with the Function Approximation Method

Debugging: action-value estimates exploding

I try to modify the Q-learning function approximation method code from your MountainCar example for my custom OpenGym Shower Agent Environment. The MountainCar example has 2 features (observational space), mine has 1 feature. So, I try to change the scalar state to an array with 1 element using np.array(state).reshape(1,1). Then I try to run the experiment, I get an error message from **choose_action_epsilon_greedy** method, "ValueError: 'a' cannot be empty unless no samples are taken". The first thing I did is to replicate the error message, and I found the error is caused by the array is empty in np.random.choise().

Debugging

[40]:
```
# ValueError: 'a' cannot be empty unless no samples are taken
# cannot put empty array in random.choice()
np.random.choice([])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-40-12dda87b300a> in <module>
      1 # ValueError: 'a' cannot be empty unless no samples are taken
      2 # cannot put empty array in random.choice()
----> 3 np.random.choice([])

mtrand.pyx in numpy.random.mtrand.RandomState.choice()

ValueError: 'a' cannot be empty unless no samples are taken
```

[41]:
```
env = ShowerEnvHard()
print(env.action_space.n, env.observation_space.shape[0], env.reset())
```

```
3 1 36
```

[42]:
```
# Try to replicate the error...
state = np.array(env.reset()).reshape((1,1))
weights = np.zeros((1, 3))
action_values = np.dot(state, weights)
np.random.choice(np.where(action_values == action_values.max())[0])
```

[42]:
```
0
```

Debugging is very important in any programming-related work. I use the print function inside the **choose_action_epsilon_greedy** method and found the error is caused by one action-value estimate exploding to infinity while the others remained at 0. This happened because the epsilon (exploration) was set to 0, it wouldn't select other two actions.

```
[[2.04791707e+307 0.00000000e+000 0.00000000e+000]]
[[8.21366227e+307 0.00000000e+000 0.00000000e+000]]
[[8.43558852e+307 0.00000000e+000 0.00000000e+000]]
[[inf  0.  0.]]
[[inf  0.  0.]]
[[nan  0.  0.]]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-22-2f571518bf06> in <module>
----> 1 ret, avg = agentHardFA.learn_task(100, print_progress = False)

<ipython-input-15-af80f7950362> in learn_task(self, num_episodes, print_progress)
    142
    143         for episode in tqdm( range(1, num_episodes+1), position=0, leave=True ):
--> 144             episode_return = self.run_episode(epsilon, episode)
    145             returns.append(episode_return)
    146             if episode > 100:

<ipython-input-15-af80f7950362> in run_episode(self, epsilon, episode)
    103
    104         while not done:
--> 105             action, action_value, _ = self.choose_action_epsilon_greedy(state, epsilon)
    106             next_state, reward, done, _ = self.env.step(action)
    107             #next_state = next_state.reshape((1,self.state_size))

<ipython-input-15-af80f7950362> in choose_action_epsilon_greedy(self, state, epsilon)
     83             action = np.random.choice( self.env.action_space.n )
     84         else:
---> 85             action = np.random.choice(np.where(action_values == action_values.max())[0])
     86         action_value = action_values[0,action]
     87         action_max = np.max(action_values)

mtrand.pyx in numpy.random.mtrand.RandomState.choice()

ValueError: 'a' cannot be empty unless no samples are taken
```

So, I try to increase the epsilon (exploration) to 0.1 and see what happened, and now they all explode with the first action reaching infinity and trigger the error…

```
[[ 2.06157964e+307 -4.28982439e+279  2.40747415e+302]]
[[ 2.11903844e+307 -4.40938715e+279  2.47457346e+302]]
[[             inf  2.02906865e+279 -1.13872501e+302]]
[[             inf  1.96225786e+279 -1.10123041e+302]]
[[             nan  1.96225786e+279 -1.10123041e+302]]
[[             nan  1.90868550e+279 -1.07116529e+302]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-24-2f571518bf06> in <module>
----> 1 ret, avg = agentHardFA.learn_task(100, print_progress = False)

<ipython-input-15-af80f7950362> in learn_task(self, num_episodes, print_progress)
    142
    143         for episode in tqdm( range(1, num_episodes+1), position=0, leave=True ):
--> 144             episode_return = self.run_episode(epsilon, episode)
    145             returns.append(episode_return)
    146             if episode > 100:

<ipython-input-15-af80f7950362> in run_episode(self, epsilon, episode)
    108             next_state = np.array(next_state).reshape((1,self.state_size))
    109             next_action, _, max_q = \
--> 110                 self.choose_action_epsilon_greedy(next_state, epsilon)
    111
    112             """

<ipython-input-15-af80f7950362> in choose_action_epsilon_greedy(self, state, epsilon)
     83             action = np.random.choice( self.env.action_space.n )
     84         else:
---> 85             action = np.random.choice(np.where(action_values == action_values.max())[0])
     86         action_value = action_values[0,action]
     87         action_max = np.max(action_values)

mtrand.pyx in numpy.random.mtrand.RandomState.choice()

ValueError: 'a' cannot be empty unless no samples are taken
```

After communicating with you by email, you give me some suggestions:

- The updates are not sufficiently small.
- This must be because the gradient is the delta * the state, and the state is an unnormalized, large integer.
- **Convergence and stability are only guaranteed in the learning rate is "sufficiently small".**
- Let's try a smaller alpha.

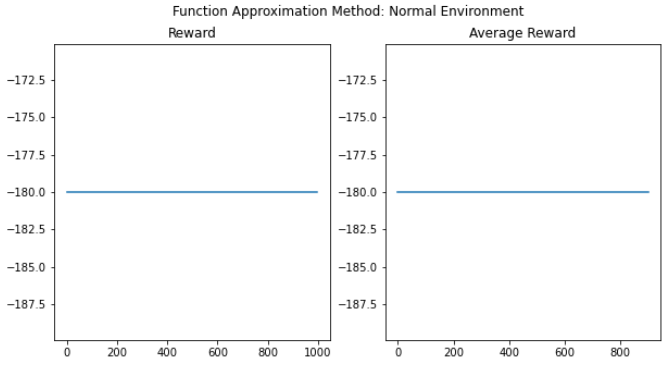
The training process is stabilized after using small alpha, the problem of q-value exploding is solved.

```
gamma = 1.
alpha = .0001
epsilon = 0.1
epsilon_decay = .9999

agentHardFA = LinearQLearner(ShowerEnvHard(), gamma, alpha, epsilon, epsilon_decay)
```

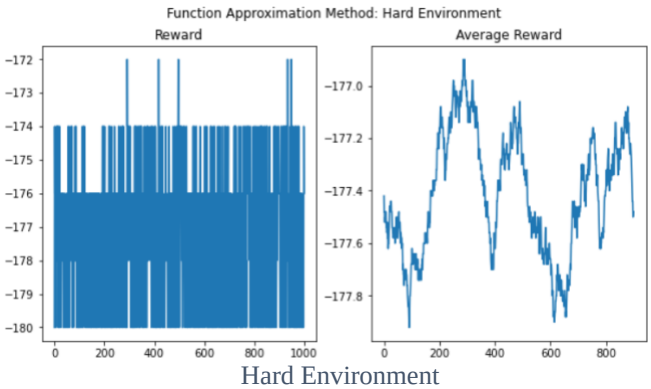
```
ret, avg = agentHardFA.learn_task(1000, print_progress = True)
```

```
 22%|██        | 224/1000 [00:01<00:05, 152.32it/s]
Average Reward for Last 100 Episodes = -58.74
 32%|███       | 324/1000 [00:02<00:04, 153.77it/s]
Average Reward for Last 100 Episodes = -58.68
 44%|████      | 437/1000 [00:02<00:03, 152.39it/s]
Average Reward for Last 100 Episodes = -58.82
 52%|█████     | 519/1000 [00:03<00:03, 153.43it/s]
Average Reward for Last 100 Episodes = -58.66
 63%|██████    | 626/1000 [00:04<00:02, 148.31it/s]
Average Reward for Last 100 Episodes = -58.56
 72%|███████   | 721/1000 [00:04<00:01, 148.10it/s]
Average Reward for Last 100 Episodes = -58.76
 82%|████████  | 816/1000 [00:05<00:01, 149.11it/s]
Average Reward for Last 100 Episodes = -58.84
 93%|█████████ | 927/1000 [00:06<00:00, 144.68it/s]
Average Reward for Last 100 Episodes = -58.82
100%|██████████| 1000/1000 [00:06<00:00, 151.01it/s]
Average Reward for Last 100 Episodes = -58.42
```

But the agent struggles to learn even for normal environment.



Normal Environment

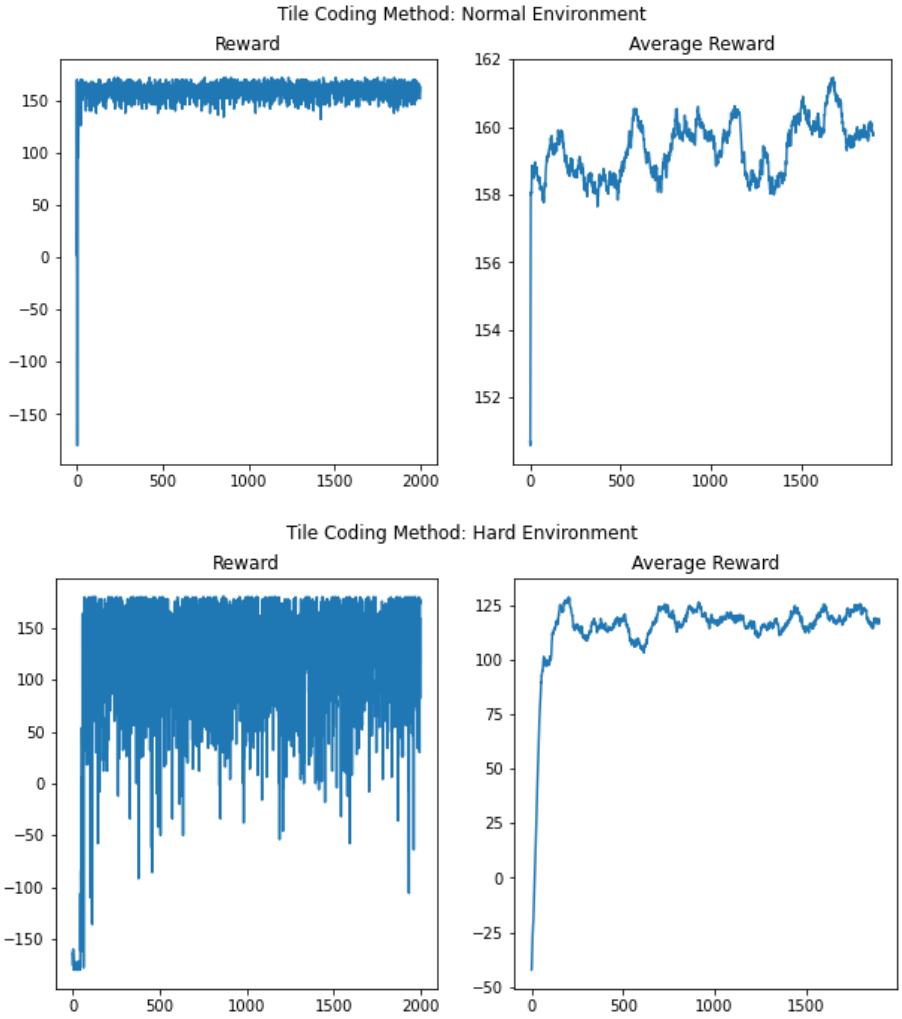


Hard Environment

Method Update: Tile Coding Representation

The linear form does not perform well, I would like to know the reason behind it. So, I start thinking about my reward functions. Both ShowerEnvNormal and ShowerEnvHard are non-linear, the agent gets reward only for 4 states (36, 37, 38, 39). No linear coefficient can learn state 35 = up and 40 = down. Learning the reward function is the key to solve the environments, this makes me think about using another approach – Tile coding.

For the normal environment, I use 3 tiling with 9 tiles per tiling; for hard environment, I use 2 tiling with 16 tiles per tiling. And YES! It works for both environments.



4. Conclusion and Limitations

Conclusion:

The Tabular method works well to solved both environments. Although the Linear Approximation method suffers to learn the task, the Tile Coding representation provides a powerful boost to solve the environments.

Environment limitation: delay between dial turn and temperature

There are still improvements can be made to my shower environment. In reality, there may be delay between dial turn and water temperature. The action we take in previous timesteps will have an effect to current state. It will add additional complexity to the environment. I wish I could have more time to work on the project so that I can implement this feature to my environment.

Hyperparameter search:

For each method and environment, there is an optimal hyperparameter combination. I could get better results by using Optuna package to search parameters. But since our agent is able to solve both normal and hard version environments, I will leave this part for future me ╰(*°▽°*)╯.

Method of solving the environment: implementing non-linear methods

Although the Tile Coding representation successfully solve both environments, non-linear method like tree and a q-network can be implemented in future work.

5. References

Building a Custom Environment for Deep Reinforcement Learning with OpenAI Gym and Python. (2021, January 10). [Video]. YouTube. https://www.youtube.com/watch?v=bD6V3rcr_54&list=PLmc_a2PakC7pSQPHxFvABqWqnQXuBICAZ&index=1&t=957s&ab_channel=NicholasRenotte

King, A. (2019, October 17). *Create custom gym environments from scratch — A stock market example*. Medium. https://towardsdatascience.com/creating-a-custom-openai-gym-environment-for-stock-trading-be532be3910e

[OC] Shower Temperature Compared to Handle Position. (2020, July 6). Reddit. https://www.reddit.com/r/dataisbeautiful/comments/hm5skw/oc_shower_temperature_compared_to_handle_position/