CAPSTONE PROJECT

Study of Methodologies of Reinforcement Learning and Understanding of Problem Representation in Code

BUSINESS REPORT

Pranav Panchal

Introduction:

Machine learning can be broadly classified into three types: supervised learning, unsupervised learning, and reinforcement learning. Reinforcement learning is a machine learning approach in which an agent is introduced to an environment, with which it is supposed to interact and gather information. Using this information, the agent is supposed to make informed decisions to achieve its target objective.

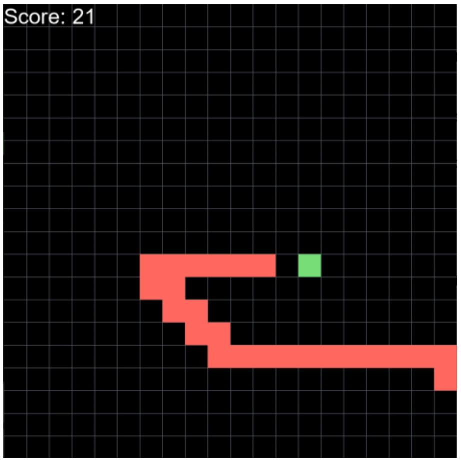
Real tangible problems can be solved using reinforcement learning. Though, one of the major hurdles while doing this is being able to properly represent the problem within a machine environment. Carefully selecting the parameters of the model, to optimize the solution in the way deemed desirable is a key skill for successfully implementing reinforcement learning.

Problem Statement:

To Study the Methodologies of Reinforcement Learning and to Develop an Understanding of Problem Representation in Code.

Scope:

For the purpose of this project, we have focused our attention on the q-learning algorithm which is based on the Markov Decision Process. This is because the objective of the project was to develop an understanding of approaching a problem statement via reinforcement learning and not to create the best reinforcement learning model. We have also dived into the gym library developed by OpenAI to extract ideas for environment modeling.



Snake Game

Feasibility:

There are multiple reinforcement learning models, however for our scope, it is sufficient to implement a relatively simple model. In the project, we have implemented a q-learning algorithm on a custom snake game environment, where we have the choice of representing the state and action spaces of the environment. This was crucial, as these parameters affect the behaviour of the learning agent.

Methodology:

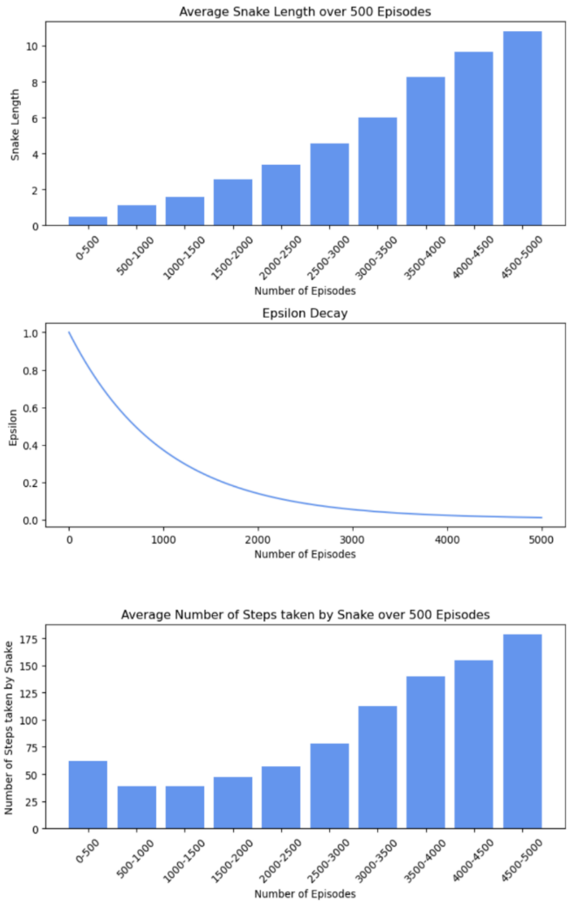
The Q-learning algorithm consists of three main components: the environment state, the possible actions for that state and the reward of that state. Using this information the agent populates a q-table which it can use as a guide to navigate the environment at the later stage.

Initially the q-table is empty as the agent has no experience. During this phase the agent is supposed to randomly pick an action from the available actions without worrying about the consequences. This is known as the exploration phase, and it is used to learn the environment and populate the q-table.

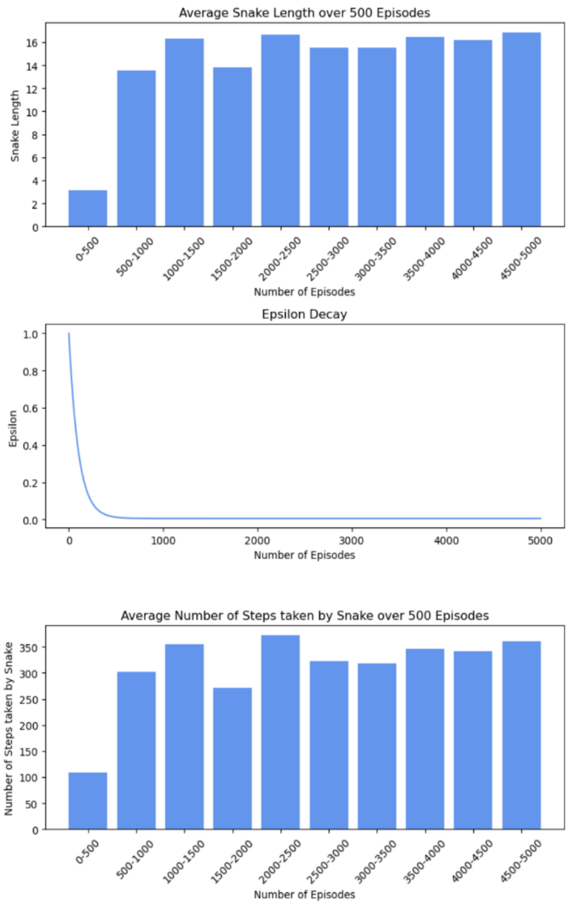
As the agent gains experience it can start utilising this to exploit the environment is increase the rewards it receives. Even while exploiting the environment the agent continues to learn and update the q-table values.

Our snake environment was setup such that we could adjust the span of the game grid as well as adjust the speed of the snake. The adjustment of snake speed was helpful as we were able to run the training cycles at a faster pace and perform the testing cycle at a slower pace. We also have a functionality wherein; we could specify the times when we want the environment to be visually rendered. The agent can train much faster if the environment is not visually rendered.

The action space is represented by 3 integers, depicting moving straight, turning left and turning right. The observation space is represented by a tuple of 3 integers. The first integer represents danger of collision, the value of integer mapped to the relative direction of danger. The second number represents the direction of the snake head motion. The third integer specifies the relative position of food with respect to the snake head. In total there are 256 different states for our snake, and each state has 3 actions. Thus, our q-table is a 4-dimensional matrix. The algorithm needs to cleverly balance its exploration and exploitation phases, as too little exploration would result in the agent being stuck in exploiting rewards which might be less than the maximum rewards given by the environment. This happens as the agent stops exploration at an early stage and never comes across the state which gives it the maximum reward. On the contrary, to little exploitation would mean that the agent is not able to optimise its time to get maximum rewards. It was observed that in such cases the performance of the agent begins to decline.



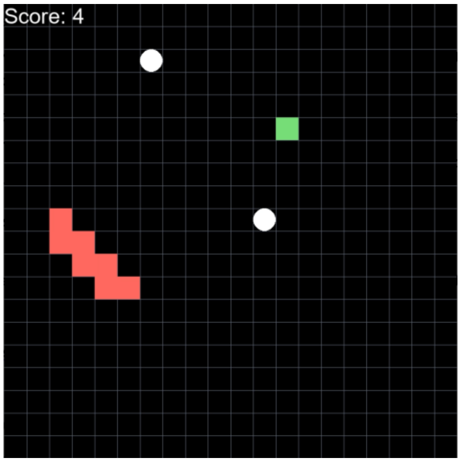
Model 1



Model 2

For our snake environment we ended up running our model multiple times to optimise the parameters of the environment. Below we see the performance of two such train cycles.

Both training cycles were run for 5000 episodes. We notice that the second model was able to exceed the average snake length of the first model. It was able to better the performance of the first model in first 1000 episodes. The second model thus seems to have performed nearly 10 times better in terms of snake length.



Snake Game - Advanced

Notice how the epsilon decay is different for both models. The first model seems to be getting stuck in an exploration loop, whereas the second model seems to have gone the exploitation route. Though, we notice that the performance of the second model continues to improve even after the epsilon has reached its minimum value.

The average number of steps taken by the snake gives an idea of how the snake has adapted to avoid collisions with the boundaries and its body. Higher the number, better is the performance.

We later used this trained model to on a few test scenarios and were able to consistently score high. As a bonus we used the same trained model and ran it on a advanced version of the snake game. In this new environment, along with food there were 2 obstacles spawning at random location on the board. Our snake was still able to navigate successfully around this environment and was getting good scores.

Plan of Action:

This project can be taken further in the following ways:

- Other advanced RL models can be used on the snake environments and their behaviors and relative performances can be studied.
- The game environment can be further modified to depict a maze and we can observe how a model trained on one environment performs on the other.

Conclusion:

We have successfully implemented the q-learning algorithm and have developed an understanding of the state representation of the environment. We have also shown that it is possible to train a model on one environment and use it on another environment as long as the observation space and action space remains the same.