Karolina Drabent
Patryk Fijałkowski
Artur Haczek
Krzysztof Kamiński
Pamela Krzypkowska

June 3, 2019

Abstract

This short paper will cover the topic of Q-Learning tested on an Agent base input game, which in our scenario is a simple Snake game. We will first explain what Q-Learning is, how and why it works and what is it used for. Later we will present our solution with the usage of Q-Learning methods which goal is to teach an Agent to play our Snake game, maximise its score and amount of moves per game. Finally we would show how our method is working in practise, what changes could be made further on as well as different approaches that we had encountering numerous problems along with their actual outcomes.

1 Background Knowledge

1.1 Q Learning

Q Learning is one of the Reinforcement Learning paradigm methods. This is a method somewhere close to unsupervised learning, which is a Machine Learning method which centres around learning without a teacher. A teacher is usually represented by a data-set or a set of labels and observed outcomes. Q Learning is based on a reward given to an agent after an action the agent has performed in a given environment. The environment is always a model providing positive or negative rewards to agent.

The goal of this learning method is to teach an agent, which decisions to make under which circumstances. Therefore the agent's goal is to create its own model of the environment, not known to the agent at all before the active learning process starts. The agent is changing its future decisions based on the rewards from its actions - building the model previously mentioned.

The classic Q Learning is based on a Q table which is a reward function presented as a matrix. This function returns the value of a Q function having the previous action chosen being A and previous state (before performing a given action A) being Q (The function will be explained later, the main focus is that the system counts the values in Q table based on previous state, action chosen

and reward given from the environment). Firstly the Q table is filled with a neutral element (it can also be any arbitrary value chosen by a programmer), when we are using real numbers the neutral element would be 0. Therefore the Q table has a following function signature:

$$Q : S \times A \to \mathbb{R} \tag{1}$$

The process of updating values in the Q table which tells us what reward can be given to an agent being in state Q, performing action action A is made based on iterations. Before deciding the next actions, agent checks the Q-table and learns which action would be the most profitable based on its previous knowledge of rewards provided. Every iterations gives the system more information because after every action it updates the Q table with the new value of reward-based-value given on action A performed from state Q. The value put inside the Q table is obtained with the following equation 1, presented below:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Figure 1: Q function

The values such as the learning rate and discount factor are there to be tweaked by human agents building the model. Learning rate being a level to which new information overrides old information (how much the Q table is going to be changed after new information is gained) and discount factor is a way of making agent more far-sighted, so it would also value long term rewards that might be gained, not only short term rewards which might be smaller.

The system finished the learning when it reaches the final state. After the final state, the model (meaning the Q table) can be used for next iterations to enable to agent to teach itself better while working with the environment, updating its knowledge with every iteration. It's important to note that there might be no final state, and the Q Learning algorithm can also be used in such situations.

1.2 Deep Q Learning

The method we have chosen in our project is a Deep Q Learning, which is a method of Q Learning using Deep neural networks. In Deep Q Learning there is no Q table, which is a model of environment created by the agent. The model of the environment is contained in the network. This network can be called a Q Network, this network outputs the predicted reward for the agent, given actions it can perform.

1.3 Agent input games

Snake is a very simple game, which originated back in the XXth Century. It is quite popular to use simple games in which a human agent inputs sequences, to showcase popular Machine Learning algorithms, especially neural networks. In this examples a input usually generated by a human is changed so that the input is generated by a computer program (we can also call it an agent).

Having games with points to collect or score like Pacman, Space Impact or Snake it is an interesting case study to implement reinforcement learning methods, trying to make the agent learn to play the game from externally provided rewards. Rewards correlate with point that the player is collecting and the time for which it can stay alive. Many of those games not only enable the player to score points, but also have some actions lead to a death of a player and loosing all points. A learning agent is then facing a problem of optimizing the risks along with potential gains.

It is quite important to note that this is a simplified model of a very low-level human-like learning process based on rewards.

2 Our methods

2.1 Environment

As mentioned before we are training an agent to learn how to play snake. I won't get into details of architecture of the game itself, because it is not too important. The crucial part lies in a model of the environment and what we want to teach our agent. We want to teach our snake to learn how to score as many points as possible and live as long as possible. Points are scored when the agent is moving on a board of fixed size, and is eating apples. The maximum number of points available for the player is bounded by the board size. The game UI is showed on the picture 2 below:

Figure 2: Our snake game UI

The first thing we need for our agent was to implement a way to tell the agent the state of the game, our environment state. The way we structured our state is shown on the picture 3 below:

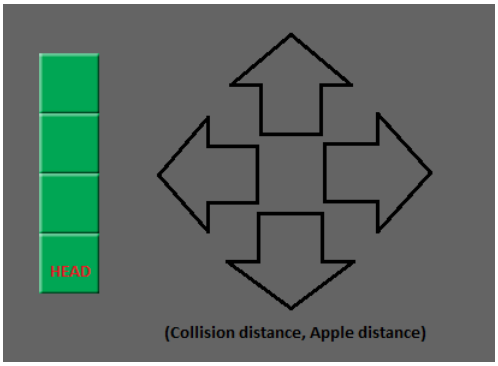

Figure 3: Our snake game UI

As we can see we give the agent information in four directions. It is because the agent can only choose from four possible moves, top, down, left and right. The information is for every direction is the distance to the nearest collision and distance to the apple (the point). The distances were counted in multiple ways, because as it is obvious not all of them were equally good. The collision distances were always the same - how many moves in this direction snake needs to make to die, the apple distance though varied. We started from Euclidean distance to the apple from every 'next possible move' and finally based our model on a distance which gave zero when the apple was not in this direction at all and binary 0/1 when apple was in the direction given, with partial points in the range from 0 to 1 on apple being somewhere diagonally. So finally the state is counted in a given way:

Algorithm 1 Count state

```
1:  procedure CountState
2:      applePos ← position of the apple
3:      snakePos ← position of head
4:      for d in directions do
5:          collisionD ← Nearest Wall/Snake in this direction
6:      for d in directions do
7:          if apple is in d then
8:              add 1 to state
9:      if appleD is  empty then
10:         count partial apple distance
        return collisionD  + apple
```

This was presented the final way of counting the state before putting it into the neural network which is set to provide us with the values with which the neurons corresponding to each action/move is fired. From the neurons activity the agent is choosing the mostly rewarded action and that's how the process is going in an iterative way.

On Game Over the environment is reset to the starting state which is having the snake of length four, facing the bottom direction with an apple in a random spot of the board. The neural network is preserved with every iteration and the information is later used for training.

2.2 Training