Machine Learning: Creating An Artificial Intelligence for StarCraft 2

Jack Gooding

23rd March 2018

Abstract

This report investigates the use of machine learning and artificial intelligence in the StarCraft II game environment. I looked at Q Learning and its implementation with an agent using Python 3.0 and the pysc2 library. I found that although the reward system in Q Learning is good for an agent to solve simple tasks; the complexity of the StarCraft II environment meant that Q Learning was not sufficient for the agent to find a way to win. The agents using Q Learning were unable to win a single game against Blizzard's Easy difficulty AI.

1 Introduction

1.1 What is StarCraft II?

StarCraft II is a real-time strategy game (RTS) which was released in 2010 and developed by Blizzard Inc.[1] There are three playable races each with unique traits, units and play styles. You can read more about the races and download the game for free at https://starcraft2.com/en-us/game

The objective in the 1 vs 1 game mode is to eliminate your opponent, and this can be achieved by destroying all of their structures. With this in mind we can reduce StarCraft II to 3 sub-objectives, construct a base, build an army, and attack. Constructing and expanding our base allows us to collect more minerals and vespene gas, the in-game currency, to purchase units and structures as well as giving us access to upgrades for our units. With enough income we can purchase upgrades and build up our forces until we eventually overwhelm our opponent. How well you are able to control your army at the same time as building and expanding your base generally defines how skilled you are as a player.[2]

There are a wide variety of units and strategies available for each race, each game you play will be different in some way. The timing of your expansion, when you purchase upgrades, how many additional workers you build, the position of your army and your opponent's, the list goes on. Furthermore you have imperfect information about your opponent. You know where the spawn the location is but, unless you have a unit which has vision, you have no idea what your opponent is up to. This fog of war can offer a tactical advantage, if you hide a technology or production structure, you can catch your opponent off guard and achieve a swift victory.[3]

1.2 A Game Scenario

Let's look at an example with a game of two players, Red and Blue. For the sake of simplicity, in this example we will only be considering the army value of each player. The army value is a hidden statistic within the game that scores the army.

- The Red Player decides to spend 100 minerals and 100 gas on a Marine and Marauder. He can see that losing minerals and gas is a small price to pay for a larger army.
- The Red Player continues to create Marines and Marauders.
- The Blue Player decides to spend 100 minerals and 100 gas on +1 Infantry Weapons. He can see that losing the minerals and gas is a small price to pay to have higher quality units.
- The Blue Player now decides to create Marines and Marauders.

Let's say that the Red Player decides to hold back. The longer he waits to attack the more likely it is that the Blue Player's +1 upgrade finishes. If the Red Player attacks before +1 finishes for Blue Player, the Red Player will have the advantage because he will have the larger army. On the other hand, if the +1 upgrade finishes for the Blue Player then he will have the advantage. Despite his army being smaller the +1 upgrade means all his units, even those created before the upgrade, are dealing more damage.

As a player, it is sometimes difficult to decide between upgrades and units. This is only one of the many decisions that must be made in a game of Starcraft II.

2 Creating an Artificial Intelligence to play StarCraft II

With the release of SC2LE (StarCraft II Learning Environment) we have a sophisticated tool to train agents to play StarCraft II. [4] What makes this so interesting and exciting is that to master the game of StarCraft II the agent must overcome several challenges. The game itself provides imperfect information, vision of your opponent's base and forces is limited by a fog of war which means that the agent must actively scout its opponent and adapt its play style in real time. There are few instant rewards, the decisions made in the early game can have a small effect on the current state and these decisions may not be rewarded until later states of the game where they have significant impact. An example of this can be seen in upgrades for units, for example the time taken for +1 Infantry Weapons is 114 seconds, along with 100 minerals and 100 gas. This sees no immediate benefit, a reduction of minerals and gas results in a negative reward, however the reward for acquiring the +1 upgrade is significantly greater than the negative. The reward for the upgrade is not seen until a later state in the game. How can we create an artificial intelligence which can overcome these challenges?

2.1 What is Machine Learning?

Machine Learning is a field of computer science which branches from artificial intelligence and data analysis. Machine Learning algorithms build a model based on samples and make predictions or decisions based on the data, rather than using strict programmed instructions. This can yield better results than other algorithms where the problem would require designing a difficult or impossible explicit algorithm. Some examples of Machine Learning include spam email filtering, handwriting recognition and ranking or grouping new species.

2.2 Reinforcement Learning

Reinforcement Learning is one area of Machine Learning. It investigates how an agent can learn to perform a task by interacting with its environment, using a reward system to define how well the agent is performing said task. The task could be anything from drawing a circle on a screen to attempting to play an entire video game. The agent is rewarded for reaching a target or completing task and gains a higher reward based on how the target is reached or task is completed. Reinforcement Learning appears to be a key element of what we expect from the Science Fiction representation of an artificial intelligence. In theory, with the correct complexity of algorithms, an agent could learn to perform human tasks at a better standard than we can! The reality of that expectation is not far away and with recent developments in technologies the SC2LE is able to perform on par with a novice player in some mini-games. [4, p.13]

2.3 Q Learning

One Reinforcement Learning algorithm we can use is known as Q Learning which uses the Bellman Equation [5]:

$$Q(s,a) = Q(s,a) + \gamma * Max[Q(s',a') - Q(s,a)]$$

We can initialize Q as a matrix of the state of the environment, s, and the available actions the agent can take, a. Our learning rate, γ is a value between 0 and 1 which weights the new Q value. The Max function finds the highest scoring value between the new Q, with state s' and available actions a', and the old Q. By looping this algorithm until the state is terminal we will have a matrix Q which will be able to complete its task. By looping this algorithm for many episodes we

can potentially reach more optimal solutions as the agent will be able to perform actions which get it closest to its reward when in each state.[6]

2.4 Creating an Agent

Python, in conjunction with the pysc2 library, was used to create agents to interact with the game environment.

Three agents are provided with the pysc2 library, the scripted agent, the base agent and the random agent. Our agents are built upon the base agent. The agents were designed as the Terran race and faced a random race as their opponent on the Simple64 map, also provided by pysc2. This map is a simplified multi player map. With a starting location and one expansion for each player. It can be used as a test environment for training agents before using the multi player maps available in matchmaking.

The simple agent was created first. As a Terran, one of the simplest strategies is to build a supply depot, then a barracks, then continuously build marines and attack the enemy.

(1)

The next stage of development was to make the agent "smart". This was done by removing the scripted elements and allowing the agent to decide on its own action based on the state of the game. A Q learning algorithm was used to determine the best action to take in a given state of the game.[7] The agent was rewarded for killing units or buildings.

```
52
53 smart_actions = [
54     ACTION_DO_NOTHING,
55     ACTION_SELECT_SCV,
56     ACTION_BUILD_SUPPLY_DEPOT,
57     ACTION_BUILD_BARRACKS,
58     ACTION_SELECT_BARRACKS,
59     ACTION_BUILD_MARINE,
60     ACTION_SELECT_ARMY,
61     ACTION_ATTACK
62     ]
63
64 #Rewards
65 KILL_UNIT_REWARD = 0.2
66 KILL_BUILDING_REWARD = 0.5
```

(2)

The attack location was still predefined in the "smart" agent and this was changed in the "smart attack" agent so that the agent could chose from different attack locations. The range of a marine is circular with a radius of 5 units so by splitting the minimap into 16 squares we can be sure that

the agent can use its marines to attack the enemy at any location.

```
for mm_x in range(0, 64):
    for mm_y in range(0, 64):
        if (mm_x + 1) % 16 == 0 and (mm_y + 1) % 16 == 0:
            smart_actions.append(ACTION_ATTACK + "_" + str(mm_x - 8) + "_" + str(mm_y - 8))
```

(3)

Furthermore we can add hot squares, locations where the enemy is located, to help the agent attack where the enemy is seen and likely to be. This helps to prevent the agent from randomly attacking areas of the map when it has already engaged its enemy.

```
        current_state = numpy.zeros(20)
        current_state[0] = supply_depot_count
        current_state[1] = barracks_count
        current_state[2] = supply_limit
        current_state[3] = army_supply

        hot_squares = numpy.zeros(16)
        enemy_y, enemy_x = (obs.observation["minimap"][_PLAYER_RELATIVE] == _PLAYER_HOSTILE).nonzero()
        for i in range(0, len(enemy_y)):
            y = int(math.ceil((enemy_y[i] + 1) / 16))
            x = int(math.ceil((enemy_x[i] + 1) / 16))

            hot_squares[((y - 1) * 4) + (x - 1)] = 1

        if not self.base_top_left:
            hot_squares = hot_squares[::-1]

        for i in range(0, 16):
            current_state[i + 4] = hot_squares[i]

        if self.previous_action is not None:
            reward = 0
            if killed_unit_score > self.previous_killed_unit_score:
                reward += KILL_UNIT_reward
            if killed_building_score > self.previous_killed_building_score:
                reward += KILL_BUILDING_reward
            self.qlearn.learn(str(self.previous_state), self.previous_action, reward, str(current_state))

        rl_action = self.qlearn.choose_action(str(current_state))
        smart_action = smart_actions[rl_action]
```

(4)

Once this was added a new method of rewarding the agent had to be implemented. Whilst the "smart" agents were being rewarded for killing units and killing buildings, remember the win condition in the game is to destroy all enemy buildings, this had led to some odd behaviour. The agent would get a higher reward if it allowed its opponent to survive. By allowing its opponent to produce units and buildings at a rate so that the agent could destroy them as they were constructed; the agent got a large reward for these long games. This is because the agent was not be rewarded sufficiently for the win itself and not negatively impacted by delaying a win. By changing to a sparse reward system, giving the reward only when the game was over. The Q table is left with negative values until a win is achieved. Therefore the agent will still choose the best action, the least negative, and have a higher chance of performing actions which get a win.

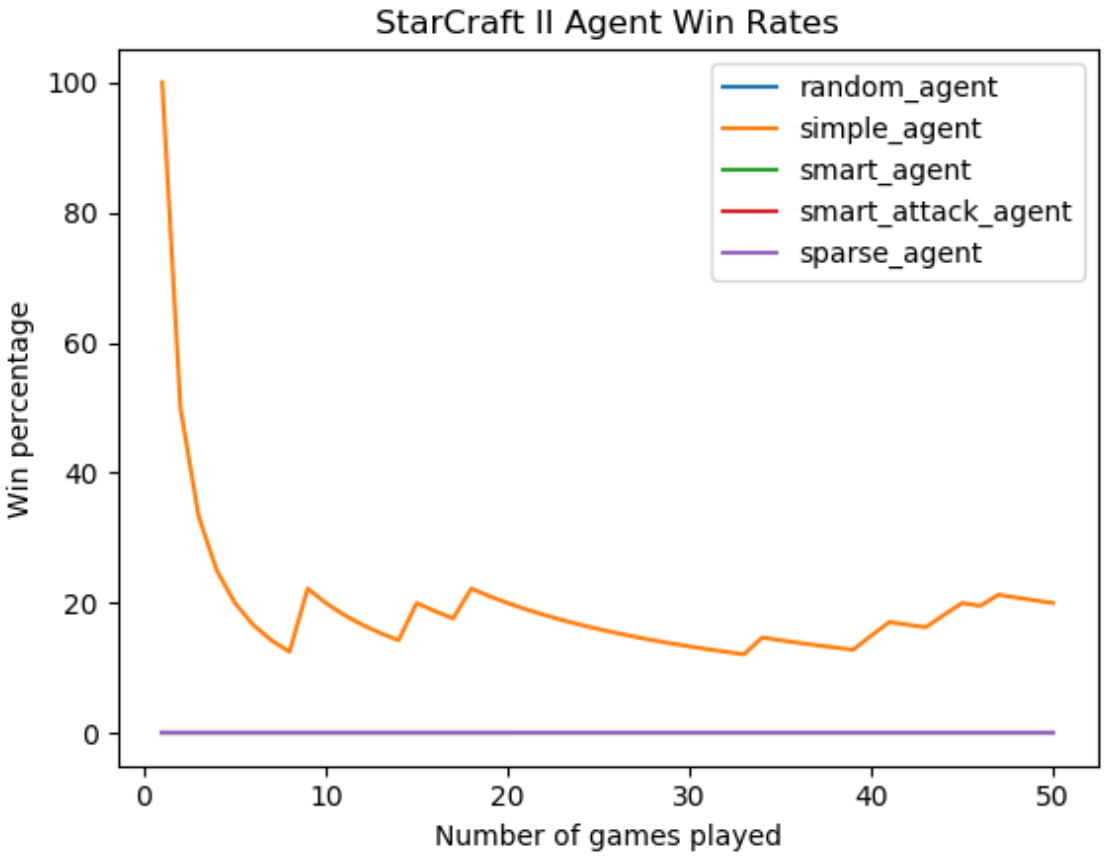
3 Analysis

3.1 Methodology

The agents were ran until a total of 50 games were played. The base for comparisons is the random agent because we want our AI to be showing some problem solving which is better than randomly selecting actions. The sparse agent, the only agent with a Q Matrix being written to and read from a file, was trained for 1000 episodes (games played) prior to the testing. The outcome of each game played was recorded in a results text file as a 1 for a win, 0 for a draw and -1 for a loss. A graph of the win percentage as the games played increases for each agent was plotted.

3.2 Agent Analysis

The theory of the Q learning algorithm [8] is that the agent should be able to find the local minimum for every state. However Q Learning is not sufficient for an agent to learn how to play StarCraft II.



(5)

We can see from the results, that the win rate of every agent is at or below 20% and that most of the agents failed to achieve above 1% this is both interesting and significantly worse than expected.

The scripted agent was the most successful and achieved the highest win rate with 20% and whilst this is good, in comparison to the base test random agent, it was to be expected as the action space is far to large for the random agent to achieve one win. We can also see that all the agents using the Q Learning algorithms performed no better than the random agent. These results support the fact that Q Learning is not sufficient for an agent to learn how to play StarCraft II.

4 Conclusion

The SC2LE was a great tool for implementing machine learning algorithms for agent to use whilst in the StarCraft II game environment. The results from my testing showed that the Q learning algorithm was not sufficient for an agent to learn how to win a game of StarCraft II. The agents using the Q learning algorithm were no better than the agent performing random actions. This was unexpected but shows how StarCraft II is not a simple problem and has suitable complexity for it to be useful for research and development in artificial intelligence.

References

[1] Blizzard, "Starcraft ii home page." https://starcraft2.com/en-us/.

[2] SC2HL, "Byunn's reaper micro." https://www.youtube.com/watch?v=xrAlhk98WxE.

[3] SC2HL, "Proxy dark shrine in enemy base?." https://www.youtube.com/watch?v=bM_VnWTadXQ.

[4] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, *et al.*, "Starcraft ii: a new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[5] W. Authors, "Bellman equation." https://en.wikipedia.org/wiki/Bellman_equation#Example.

[6] M. Zhou, "Q learning reinforcement learning (eng python tutorial)." https://www.youtube.com/watch?v=qPE4CPQY7mc&list=PLXO45tsB95cIplu-fLMpUEEZTwrDNh6Ba&index=4.

[7] M. Zhou, "Reinforcement learning with tensorflow." https://github.com/MorvanZhou/Reinforcement-learning-with-tensorflow/tree/master/contents/2_Q_Learning_maze.

[8] J. Collis, "How does q learning work?." https://www.quora.com/How-does-Q-learning-work-1.