New York University

Machine Learning

Professor: David Sontag

SnakeAI

A Reinforcement Learning Approach by:

Will Bang,
Alex Hogarth,
Eugen Hotaj

May 17, 2016

Abstract

Applications of Machine Learning to video games might help us better understand how to create AI which can quickly and effectively interact with a dynamic environment. In this project we have applied reinforcement learning to the game Snake. Through the Q-learning algorithm, the snake agent iteratively learns how to make more and more efficient decisions to maximize it's overall score. Our goal for this project was twofold: 1) to create an agent which clearly learned, i.e the agent got better as it played more and more games, and 2) to create and agent which played the game *well*. We feel that we have achieved both goals. However, 2) could be significantly improved upon as we explain in this paper.

1 Introduction

Deep Mind's *AlphaGo* recently beat the World's Go champion, Lee Sedol, in a five game match, marking a great milestone in Machine Learning. Inspired by this, we wanted to develop a similar approach to try to play the game Snake[1] as optimally as possible. Our goal was to both show that the algorithm learned and got better with each game, and that it performed better than a random walk. We extended this goal to compete against a non-learning algorithm which at each step simply decreased the Euclidean distance between the snake and its goal. Finally, to test for robustness we used an extended version of Snake which included obstacles in the game space. To tackle this problem, we used a type of Reinforcement Learning known as Q-learning. Overall, we found that our Q-learning did indeed get better with each iteration (until it reached some asymptote and leveled off) and was able to significantly outperform the random walk. In the original game, the Euclidean distance algorithm outperformed our Q-learning algorithm but Q-learning proved to be much more robust in the extended obstacle version as it learned how to avoid obstacles to reach its goal, thus significantly outperformed the Euclidean distance algorithm which repeatedly got stuck.

2 Related Work

Google's Deep Mind has used a similar Q-learning approach, dubbed Deep Q-Learning, in which they use Convolutional Neural Networks as a function approximator, to play Atari Games [2]. However, since their goal was to develop a generic algorithm which could play any game, we felt their approach was a bit too general for what we were trying to accomplish so we did not rely too heavily on it. DeWolf uses the same Q-learning approach as we do in [3] but applies it to a Cat and Mouse game. Finally, Sirinivasan uses the exact same approach we do in [4] in order to play Snake. In fact, we relied on his open source code to guide our implementation of the Q-learning algorithm, though ours had to be implemented differently since the implementation of core mechanics of our game came from [5] which was much different (and much sloppier). We also extended upon Sirinivasan's state representation, as described in (3.2.2).

3 Problem Definition and Algorithm

3.1 Task

Snake is traditionally played on a two dimensional $w \times h$ grid G, $w, h \in \mathbb{N}$. The agent is a snake H of length n composed of a head h_0 at $p_0 \in G$ and a body $\{h_1, \ldots, h_{n-1}\}$ where h_i at $p_i \in G$, and $p_i \neq p_j \forall i, j \in \{0, \ldots, n-1\}$. At any state in the game, the snake can either turn left, right, or keep moving forward. Once an action is taken, h_0 moves form p_0 to an adjacent position and h_i moves from p_i to p_{i-1} so that each component of the snake follows the previous one. An apple a is randomly placed in an empty space in G. The goal of the snake is to "eat" the apple by colliding with it (i.e share the same grid location). Once the snake has eaten the apple, a new apple is randomly placed in an empty grid location and the snake grows in size by one. The snake dies and the game ends if h_0 collides with any $h_i, i \neq 0$, meaning the snake head collides with it's body. The snake also dies if if the head goes

outside the boundaries of the grid. The objective of game is to eat as many apples as possible before the snake dies. The game is considered won if the snake grows so large that it covers up every grid location.

To test the robustness of our algorithm, we also played on an extended version of the game where G contains an obstacle set O where $o_i \in O$ is located at some grid location.

Our task was to develop a machine learning algorithm which learned how to play snake "well", that is maximize $|H|$, or said more plainly achieve the longest snake possible.

3.2 Algorithm

Before the algorithm is presented, a few details must be covered.

3.2.1 Q-learning

To accomplish the task mentioned in (3.1) we used a lookup table based Q-learning approach. Q-learning is a type of Reinforcement Learning which learns a Markov Decision Process to maximize the reward for all states [6]. For a non-stochastic processes, optimality is guaranteed by the Bellman Equation. However, for a stochastic process only a local-maximum is guaranteed [7].

A lookup table Q stores the expected reward Q[uantity] of action $a_i \in A$ of state $s_t \in S$ at time t, where A is the action space and S is the state space. Since Q-learning is an iterative algorithm, $Q(s_0, A)$ is set to some initial value (10 for our implementation). Then, for $(s_t, a_i) \in Q$ it's expected reward quantity is calculated as follows

$$Q(s_t, a_i) = Q(s_t, a_i) + \alpha(R(s_{t+1} + \gamma\mathbb{E}[s_{t+1}] - Q(s_t, a_i))$$

where α is the learning rate, γ is the discount factor for the expected future reward, and R is the reward function (explained below). We chose α to be .5 as in [3] since we wanted the algorithm to converge fairly rapidly. We also chose $\gamma = .8$ so that the agent would strive for long term high rewards (as explained in [6]) such as eating the apple instead of just minimizing distance to the apple which could lead to coiling (refer to section (5) for more information on coiling and why it is detrimental).

3.2.2 State Representation

In order to fit the Markov Decision Process approach of Q-learning we first discretized the game into different states. Since Snake is played at $fps \in \mathbb{N}$ frames per second, we consider a state s_t of the game to be particular frame at time t. At such a frame, the game grid G will have some configuration G_t as described in (3.1). Since any Grid cell can either be empty or contain the apple or a piece of the snake, encoding the entire grid as a feature vector would lead to $3^{w \times h}$ different states, a quantity which surpass the number of particles in the know universe quite rapidly. Another shortcoming of this type of state representation is that Q would be entirely dependent on the size of the cardinality of G and so an agent trained in a grid of size 20×20 (the size of the gird we used for training) would not be able to play on a grid of any other size. Due to these shortcomings, we decided to encode our state vector as in [4] with a slight extension.

$$s_t = (f, l, r, t_0, t_1, t_2, x, y) \in S$$

As shown in Figure 3.2.2, f, l, r correspond to the contents of the grid spaces directly adjacent to head of the snake where $f, l, r = \{-1, 0, 1\}$ depending on what the grid contains with -1 corresponding to the body of the snake (as well as obstacles in our extended version), 0 corresponding to an empty space, and 1 corresponding to the apple. Diverging from [4], we introduced t_i which correspond to the last 3 turns the snake took, where $t_i = \{-1, 0, 1\}$ if the snake took a left turn, made no turns or uncoiled

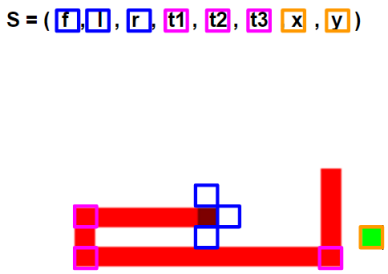

Figure 3.2.2

itself, and a right turn respectively. Finally the x, y correspond to the relative location of the apple based on the snake's head. Essentially, $x = \{-1, 1\}$ depending on if the apple was to the left or the right of the snake's head and $y = \{-1, 1\}$ depending on whether the apple was below or above the snake's head. Using this representation, we were able to cut down the state space from $3^{m \times n}$ to $3^6 \times 2^2 = 2916$ states.

3.2.3 Actions

To transition between states, the agent takes an action. For the snake game each state has only three allowable, identical actions, namely turn left, turn right, or keep going forward. Therefore we can think of an action $a_i \in \{left, right, forward\} = A$ as essentially a function on the state space

$$A : S \to S$$

3.2.4 Rewards

The cornerstone of any reinforcement learning algorithm, and in particular Q-learning, is it's reward function. Essentially, an agent is positively rewarded if it performs an action which leads it to a beneficial state or it is negatively rewarded if it reaches a detrimental state. Therefore we can think of the reward as a function from the state-action space to the real numbers

$$R : S \times A \to \mathbb{R}$$

For our implementation we assigned rewards as follows

1. +500 if the snake ate the apple
2. +10 if the snake reduced its distance to the apple
3. -25 if the snake increased its distance to the apple
4. -100 if $t_0 = t_1 = t_2 \in \{-1, 1\}$
5. -1000 if the snake crashed

where (4) essentially penalizes the agent if it takes 3 left turns or 3 right turns in a row. This again was meant as a way to combat coiling. We also made (3) bigger than (2) in absolute value since in the opposite case the agent would eventually learn that it could achieve a much higher reward by perpetually circling around the apple than by eating it.

3.2.5 $\epsilon-$Greedy Exploration

Before the algorithm is presented, one final caveat must be explained. In Q-learning, the agent chooses the action with the highest expected reward at each state. Because of this, once can see that Q-learning is inherently a greedy algorithm. As such, the convergence guaranteed by the Bellman Equation might

lead to a sub-optimal state as explained in [3]. To combat this, we used an ϵ-greedy approach where the agent makes a random action some small percentage of the time (ϵ) regardless of if the chosen action has the highest expected reward. For our implementation, we chose an epsilon of .3 and decreased it by half every 10000 frames.

3.2.6 The Algorithm

With all these pieces, we finally present the algorithm. (Note that this is just meant to be terse pseudo code of our algorithm and our actual implementation differs a bit from what is presented here).

```
Q = {}

eps = .3
eps_counter = 0
eps_max = 10000

alpha = .5
gamma = .8

while(true):
        # epsilon greedy
        eps_counter = eps_counter + 1

        if(eps_counter == eps_max):
                eps = eps/2
                eps_counter = 0

        # get state (get current state as descibed above)
        state = getCurrentState()

        # set initial expected rewards if state does not exist
        # assume optimistic state
        if state not in Q:
        Q[state] = [10,10,10]

        # choose best action or random
        if random() < eps
                action = random.choice(Q[state])
        else
                action = max(Q[state])

        next_state = act(action)

        # set initial expected rewards if state does not exist
        # assume optimistic state
        if next_state not in Q:
                Q[state] = [10,10,10]

        # get transition reward
        reward = get_reward(next_state)

        # check if snake dead at next state
        episode_ended = getEpisodeEnded(next_state)

        # update Q
        if not episode_ended:
        Q[state][action] += alpha*(reward + gamma*max(Q[next_state]) - Q[state][action])
```

```
48          else:
49                  Q[state][action] += alpha*(reward - Q[state][action])
```

4 Experimental Evaluation

4.1 Data

Since we used a Reinforcement Learning approach, we had no prior data and no pre-processing to do. Our entire data consists of the Q table explained in (3.2), which is updated as the agent plays the game.

4.2 Methodology

We used three non-learning algorithms to serve as benchmarks for comparison with our Q-Learning algorithm. The first was a random walk where the snake avoided an action if it lead to dying[1]. The second was a weighted random walk where the snake chose to go straight 80% of the time. Finally, the last and "smartest" algorithm aimed to minimize euclidean distance from the snake's head to the apple at every step, again as long as that action did not cause the snake to die[2].

Our goal was to consistently achieve a higher score with our Q-learning algorithm than the three non-learning algorithms. We found that due to the stochastic nature of the game, the agent's performance across each algorithm for a singe game was very noisy. In order to reduce this noise, we averaged the scores from the last 200 games and used this for comparison. Since only the Q-learning algorithm could see potential improvements with more games, for the three non-learning algorithms we just played the first 200 games, averaged the scores, and used this as a performance measure for any number of games (this is why the results in Figure 4.3 for the three-non learning algorithms are flat).

Finally, in order to test the robustness of our Q-learning algorithm, we also tested on an extended version of Snake as explained in (3.1).

4.3 Results

With no obstacles, our Q-Learning algorithm was able to beat the random walks, but was outperformed by the Euclidean distance algorithm as can be seen in Figure 4.3. As can be seen from the videos[3], unlike the Euclidean distance algorithm in which the snake makes a straight line to the next apple, in the Q-learning algorithm the snake traces a circular path to the next apple which is much more prone to coiling and so much more prone to dying.

Figure 4.3 also clearly demonstrates the effects of learning. As we can see the Q-learning algorithm gets better and better in the beginning. This is due to the fact that it first learns to avoid crashing into the walls and into itself and also that eating apples is beneficial. The later parts of the graph show that learning trails off and so the average score stays relatively the same (with respect to noise).

However, with obstacles in play, the Q-learning algorithm's ability to try learn an optimal strategy to maximize performance proved to be the key to outperforming the Euclidean distance algorithm, which has no notion of performance. The Euclidean distance algorithm gets stuck trying to minimize the distance to the apple, located directly on the other side of an obstacle, yet also tries to not collide with the obstacle itself[4]. On the other hand, the Q-learning agent eventually learns to get unstuck and follow a sequence of actions which lead it around obstacles instead of trying to go through them[5].

Figure 4.3

5 Conclusions

Our Q-Learning algorithm easily outperformed the random walk algorithms, and was more robust than the Euclidean distance algorithm in more complicated situations. However, we were not able to beat the Euclidean distance algorithm for the original Snake game.

One of the biggest issues we faced was coiling and it is essentially what prevented us from learning an agent which could play the game optimally (i.e beat the game). As the snake grows larger, the agent makes moves which lead it to coil around itself in effect cutting off any escape route. Thus the snake is forced to crash into itself. While we tried to combat this phenomenon with the different techniques mentioned throughout the paper, we were unable to do so. Furthermore we believe that with our current implementation it is impossible to beat coiling and thus impossible to play optimally.

However, we also think that an optimal performance could be achieved through other implementations of Q-learning. The biggest reason why our algorithm fails to perform optimally is due to our lookup table approach. In this approach, the algorithm essentially relies on visiting every single state multiple times until it reaches convergence. With our current, very basic state representation this is possible since we only have 2916 possible states and we achieve convergence after about 1000 games. However for a much bigger state representation (for example encoding the entire grid) we would never converge since the universe would cease to exist by the time we visited all our states. One way to extend to such large state spaces would be to use a function approximator as described in [8]. A very beneficial extension of this would be to use a Convolution Neural Network as a function approximator as done in [2]. Since the latter relies on individual pixel input (meaning the whole grid G) as a state representation, we believe it to be the best approach to achieving optimality. A demonstration of this approach can be seen in [9].

Appendix - Videos

1. Random Walk - https://www.youtube.com/watch?v=AiyYqRkhBtk
2. Euclidean Distance Minimization - https://www.youtube.com/watch?v=WGA_FvdsBEQ
3. Q-Learning - https://www.youtube.com/watch?v=7HMOyUju04w
4. Euclidean Distance Minimization with Obstacles - https://www.youtube.com/watch?v=A1XvREzKac0
5. Q-learning with Obstacles - https://www.youtube.com/watch?v=TDSTmdOolgo

References

[1] Snake (video game). (2016, May 8). In Wikipedia, The Free Encyclopedia. Retrieved 18:42, May 16, 2016, from https://en.wikipedia.org/w/index.php?title=Snake_(video_game)&oldid=719256180

[2] Playing Atari with Deep Reinforcement Learning. Volodymyr Mnih et al. (2013). Retrieved 00:35, May 5, 2016, from http://arxiv.org/abs/1312.5602

[3] Reinforcement Learning Part I: Q-Learning and Exploration. (November 25, 2012). Travis DeWolf. Retrieved 00:35, May 5, 2016, from https://studywolf.wordpress.com/2012/11/25/reinforcement-learning-q-learning-and-exploration

[4] Srinivasan, Pranesh. *RL Snake*. Retrieved 00:35, May 5, 2016, from http://spranesh.github.io/rl-snake

[5] Snake in 35 Lines. (July 25, 2008). Samuel Backman. Retrieved May 16, 2016 from http://pygame.org/project-Snake+in+35+lines-818-.html

[6] Q-learning. (2016, April 9). In Wikipedia, The Free Encyclopedia. Retrieved 21:42, May 16, 2016, from https://en.wikipedia.org/w/index.php?title=Q-learning&oldid=714388815

[7] Bellman equation. (2016, May 11). In Wikipedia, The Free Encyclopedia. Retrieved 21:51, May 16, 2016, from https://en.wikipedia.org/w/index.php?title=Bellman_equation&oldid=719801339

[8] Artificial Intelligence: A Modern Approach. Russell, Stuart J., Peter Norvig, and Ernest Davis. Upper Saddle River, NJ: Prentice Hall, 2010. Print.

[9] Q-Learning Snake. (June 3, 2013). bobildoktor. In YouTube. Retried May 16, 2016, from https://www.youtube.com/watch?v=kM89G5dE-og.