Comparison of reinforcement learning algorithms on a Simple Snake testbed

Giovanni Gheriglio
Alma Mater Studiorum
University of Bologna
Bologna, Italy
giovanni.gheriglio@studio.unibo.it

***Abstract*—Snake used to be a very popular mobile game in older Nokia phones. Its semplicity made it very easy to approach. This dissertation shows a simple implementation of the game to be used as testbed. Two different AI-agents from value approximation functions and policy gradient methods are presented to test the environment.**

***Index Terms*—snake, game, ddqn, a2c**

I. Introduction

Single-player Snake video games became popular with the rise of Nokia phones in the late 90s. The game was preloaded on the majority of their phones. As many 2D games, Snake has been used to test different reinforcement learning approaches. Some versions of it used to be featured on the popular OpenAI Gym environment. This dissertation presents a simple version of the game and its structure. Two different agents are compared to test the environment. One agent uses a *value function approximation* algorithm with two networks and a replay buffer. The other agent adopts a *actor-critic method* with the help of an advantage function to stabilize the training. The algorithms are briefly presented with their training specifications. In the end we'll show the results during training and subsequent tests.

II. Background

Snake is a single-player video game. The game is played on a two-dimensional square board. A snake is randomly placed on the board and is represented by a head and a body. The snake always moves horizontally or vertically. Its goal is to eat a fruit also placed randomly on the board. After the fruit gets eaten the snake gets longer and another fruit spawns. At the same time the game score goes up. The game ends when the snake goes out of the board or when it hits its own body.

A. Environment implementation

Our implementation is highly based on the original game. The board of choice is a 5x5 matrix. The snake and the fruits are placed randomly at each episode. There are four possible actions the agent can take at each step: UP, RIGHT, DOWN, LEFT. Differently from the original game the snake is only made by its head. So it cannot get longer after eating the fruit voiding the possibility of hitting against itself.
The environment offers the same basic methods OpenAI Gym's environments have: *reset()*, *step()*, *render()*. The *reset()* function initializes the game board. Each element is assigned with one of three possible values:

- 0 - *empty*
- 1 - *snake*
- 2 - *fruit*

The *step()* function takes the agent action evaluating the movement. In case the snake was going out of the board a variable *done* is sent to the agent. Both functions return the state of the game, which is made of three values:

- Position of the snake on the board
- Position of the current fruit to eat
- Minimum number of actions for the snake to reach the fruit (relative distance)

The positions of the elements on the board are calculated according to this formula:

$$element\_row * board\_rows + element\_column$$

The reward structure is set to mimic the game dynamics.

State	Reward
Lose	-1
Standard movement for 30 consecutive steps*	- 0.1
Standard movement	0
Eat fruit	+1

**This specific reward was necessary for avoiding the agent taking circular steps without any punishment.*
Despite the trivial nature of the game the number of possible states for a 5x5 matrix is about 3^{25}.

III. Experimental Algorithms

In order to test our Snake environment we experimented with two radical approaches. In particular tests have been carried out with double DQN as a value function approximation and A2C as an actor-critic method. To keep consistency for a true analysis both were trained with the same parameters:

- *discount factor* = 0.95
- *learning rate* = 0.001

The optimizers and value loss functions used are also the same from *tensorflow.Keras*:

- *optimizers.Adam*
- *losses.mean_squared_error*

A. DDQN

Double DQN [1] is an improvement over the regular DQN algorithm with buffer replay. Instead of using only one network for action selection and action evaluation there are two networks, one for each job. The network for action selection is called the *Online network*. The one for evaluating is the *Target network*. The presence of the second network prevents the algorithm to overstimate action values. The two networks are identical. In our case they are made of two hidden layers fully connected with 32 units each.

B. A2C

A2C is an *Actor-Critic* method [2] with the use of an advantage function. It is made of two networks: a *Policy network* and a *State-value network*. The actor uses the policy network to select the actions. At the same time the critic calculates the advantage function, which measures how much better an action is compared to the average action in a given state. The advantage applied to the backpropagation of the actor helps to stabilize learning. In our tests the networks have a similar structure using two hidden layers fully connected with 32 units each. The weights are learned and updated separately.

IV. Experimental Results

All tests are carried out using the Tensorflow library. In each algorithm the state gets preprocessed before being given to the networks. The preprocess consists of a normalization of the values to [0, 1] and a conversion to tensor, through the function *tf.convert_to_tensor*(). In order to be more memory efficient in the DDQN memory state values are saved as int. Each episode has a maximum number of steps set to 300. If the agents make 30 consecutive steps without getting any fruit the episode ends with a negative reward of -0.1.

A. Training

The two agents were trained for several episodes. DDQN was trained for 4000 episodes. While A2C was trained for 100000 episodes. The *Running average* shown in the following graphs is the average over the last 100 and 5000 scores respectively. The average is only meant as a guidance for the progression in training.

As we can see from Fig.1 DDQN needs far less episodes to understand the game and achieve high scores. Regarding the A2C agent shown in Fig.2 the training is much slower. In addition the agent suffers from high variance in the scores achieved. Even when the agent starts gaining confidence in the game it can get very poor scores, hence resulting in low average scores.

From a very early stage of training both agents seem to understand well the boundaries of the board. This is more noticeable in the Fig.1 where the number of reached steps stops at 30 for several episodes. Efficiently eating the fruits is a skill both develop afterwards.

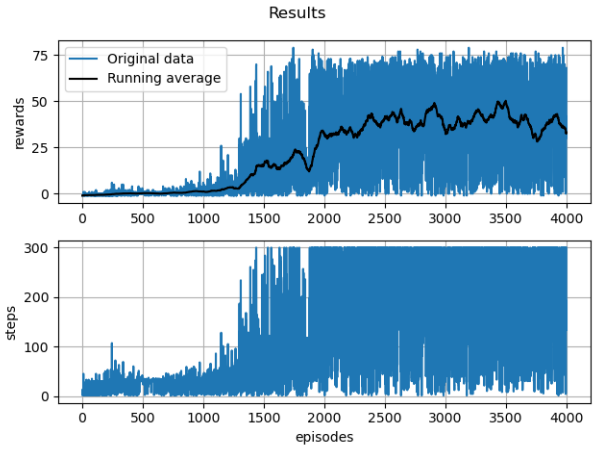

Fig. 1. DDQN Training

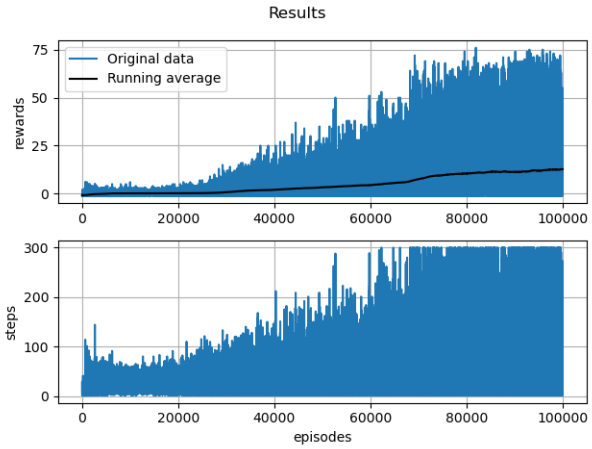

Fig. 2. A2C Training

B. Agent performances

Performances of the trained agents were tested on 100 consecutives runs of the game. This was made to flatten the differences created by the stochastic placements on the board. The table below shows the results.

Agent	Mean Score	Max Score	Training
DDQN	24.99	75	4k episodes
A2C	14.93	71	100k episodes

Both agents demonstrate to be very good at the game. The max scores show how the agents have become efficient in taking almost the minimum number of steps to eat a fruit. Max scores are very similar. Although considering the mean values the trained A2C shows its limitation. This result may be due to insufficient training or simply due to the more difficulties the agent finds in the environment tested.

V. Conclusion

Snake is a trivial videogame that is not as popular as it used to be. Although it still represents an ideal challenge for AI-agents. In our work we showed how a simple version of it still

makes a suitable testbed for comparing algorithms and networks. In particular the DDQN network proved significantly capable of mastering it. On the other hand A2C agent achieved very good results with less robustness overall.

REFERENCES

[1] Hado van Hasselt, Arthur Guez and David Silver, "Deep Reinforcement Learning with Double Q-learning," 2015.

[2] Barto and Sutton "Introduction to Reinforcement Learning," Chapter 13, Second Edition, MIT Press 2018.