Snake Game using Deep Reinforcement Learning

[1]*Department of Mathematics and Informatics, St. Kliment Ohridski University of Sofia, 5 James Bourchier Blvd, 1164, Sofia, Bulgaria*

(Dated: January 23, 2021)

Abstract The objective of this project is to conduct experiments with different approaches towards solving the popular 2D "Snake" game. Several studies have reported achieving better results using Deep Reinforcement Learning techniques in comparison to the traditional Machine Learning methods. Thus, the proposed solution to the problem at hand is to apply Deep Reinforcement Learning algorithms for 3 modes of the game: standard, with a wall and with a maze. Our research compares Deep Q-Learning using Prioritized Experience Replay, Double Deep-Q-Networks and Dueling Deep-Q-Networks applied on each mode for 100 games and to draw a conclusion which of them achieves the best results in the respective situations.

I. INTRODUCTION

This report presents a comparison between the overall performance of 3 reinforcement learning agents, along with relevant technical information regarding the Deep-Q neural networks that have been used for the purpose of solving the 2D "Snake" game. The results of this investigation provide insight into how the different optimizations on top of the Deep-Q Networks [DQN] perform in multiple game modes and allow us to draw a conclusion which of them achieves the highest result in the respective situation. The experiments outlined in this paper can be easily replicated using the command-line interface, allowing further extensibility of the proposed comparative analysis.

II. METHODOLOGY

The base approach that we have selected is to use Deep Q-Networks as they have been proven to be efficient for playing Atari games (Mnih & Kavukcuoglu, 2013 [1]) and other games such as Cartpole (Roibu, 2019 [2]). However, to alleviate the risk of sparse and delayed rewards due to the Deep-Q Learning naive reward mechanism in the context of the "Snake" game, our team has used DQN optimization techniques to increase the performance. The main focus of the experiments was to compare both the average and highest scores of Deep Q-Learning using Prioritized Experience Replay, Double Deep-Q-Networks and Dueling Deep-Q-Networks for 100 games for each algorithm and game mode. All of the experiments were conducted on a custom 'Snake' setup (pulled from 'pygame' package in Python) running on Google Colab GPUs with 11 inputs of the network as proposed in (Vinay & Aman, 2019) [3]):

1. information about whether there is an immediate danger in any of the directions;
2. the direction of the snake relative to the game board at a specific point in time;
3. relative position of the snake's head - whether it is below, to the left, to the right, or above the food on the board.

At each step, the agent can receive one of the following rewards:

- -10 - if the snake agent hits wall or bites its body;
- 10 - if the snake agent eats the apple;
- 0 - otherwise.

The action space consists of the snake's directions - UP, DOWN, RIGHT, LEFT, and the grid coordinates form the state space. Adam algorithm was used for optimization.

The following sections elaborate on the algorithms that we have experimented with.

A. Deep Q-Network using Prioritized Experience Replay

Deep Q-Network requires a lot of data and even then, it is not guaranteed to converge on the optimal value function [4]. In fact, there are situations where the network weights can diverge or oscillate, due to the high correlation between actions and states. The technique experience replay is usually used to resolve that problem. This technique consists of a large buffer containing the past experience, called replay buffer/memory. Each experience is a tuple (S, A, R, S'). In the original experience replay method, the DQN samples experiences randomly from the replay buffer without considering their quality. Because of that, its optimized version, called prioritized experience replay, was used when training the network. Prioritized experience replay assigns weights to experiences in memory to select batches that may be richer for learning. However, using just priorities to select batches would cause severe overfitting. Thus, priorities are adjusted to act as probabilities for samples to be selected.

Priorities are calculated by the formula:

$$p_i = err_i + e$$

where e is a predetermined constant.

Priorities calculation is based on the error because samples with higher error have big difference between expectations and reality, meaning there is more to learn from these samples. The constant e ensures a probability of selecting other samples to avoid overfitting.

The probabilities for selection are chosen using the following equation based on priorities:

$$P(i) = \frac{p_i^a}{\sum_k p_k^a}$$

The exponent $a \in [0, 1]$ determines how much prioritization is used. When a is 0, it corresponds to the uniform case, when a is 1, then the highest priority samples will be preferred.

The replay memory is implemented as a binary tree structure, called sumtree, because each node is equal to the sum of its leaves [3]. The leaves represent priorities. The complexity of update and access operations is $O(log(n))$, which is faster than an ordinary array.

When sampling past experience from the memory, a random number is generated between 0 and the total number of leaves. That number is called a sampling value. Then, the tree is traversed using the following conditions:

1. if the left child value is greater than the current sampling value: visit the left child node;
2. else, visit the right child node with a new sampling value equals to the current sampling value minus the left child node's value

B. Double Deep Q-Networks

According to the optimal policy in Deep Q-Learning, the agent tends to take the non-optimal action in any given state only because it has the maximum Q-value. Such problem is called the overestimations of the action value (Q-value). To prevent this, Double Q-Learning can be used. Double Q-Learning uses two different action-value functions, Q and Q', as estimators. Even if Q and Q' are noisy, these noises can be viewed as uniform distribution. Q function is for selecting the best action a with maximum Q-value of next state. Q' function is for calculating expected Q-value by using the selected action a from Q. Double Q-Learning implementation with Deep Neural Networks is called Double Deep Q-Networks (Double DQN). Double DQN uses two different Deep Neural Networks: Deep Q-Network (responsible to calculate Q) and Target Network (responsible to calculate Q'). This method is particularly useful to avoid those situations where the discrepancy between the neural networks causes them to recommend totally different actions given the same state, thus, reducing harmful overestimations and improving the performance[5].

The implementation of Double DQN is with a periodic update of the Target Network after every 5 moves by the snake agent, and also after every game. This optimization allows the Target Network to be updated periodically so that it does not have the same parameters as Q-Network. As a consequence Double DQN has independent action selection and evaluation.

Basic Double Q-Learning equation:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_{t+1} + \gamma(Q'(s_{t+1}, a)) - Q(s_t, a_t)) \quad (1)$$

$$a = \max_a(Q(s_{t+1}, a)) \quad (2)$$

$$q_{estimated} = Q'(s_{t+1}, a) \quad (3)$$

C. Dueling Deep Q-Networks

In the regular DQN both the value of the state s V(s) and the advantage of taking action a in state s, A(s,a) are calculated while estimating the Q-value. However, if the value of the state is bad and all of the possible actions lead to death, it brings no value to estimate the effect of each action since the V(s) has also been calculated. The Dueling Deep Q-Networks [6] are useful for the cases in which it is unnecessary to know the value of each action at every timestep - such as the snake game.

By decoupling state value and action advantage, the network can learn if the state itself is valuable. For instance, if the snake has curled in on itself: regardless of what action it takes, it will eventually hit its own body and die. Using Dueling DQN, the network learns that the state itself is very poor, and no action it could take would be helpful. Knowing that all possible actions are irrelevant and ineffective, it should learn to avoid entering that state in the first place. [7]

This architecture helps in accelerating the training process as only the value of a state can be calculated without the Q(s,a) for each action at that state. Therefore, much more reliable Q values can be found for each action by decoupling the estimation between two streams. [8]

Because the dueling architecture shares the same input-output interface with the standard DQN architecture, the training process is identical. We define the loss of the model as the mean squared error:

$$L(\theta) = 1/N \sum (Q_\theta(s_i, a_i) - Q'_\Theta(s_i, a_i))^2 \quad (4)$$

where

$$Q'(\theta) = R(s_t, a_t) + \gamma \max_{a'_i} Q_\theta(s'_i, a'_i) \quad (5)$$

and take the gradient descent step to update the model parameters.

The exact architecture of the network that has been implemented as part of this research is outlined in Fig. 1. The input represents the 11 properties that characterize the state of the game that are passed to the first fully-connected [FC] layer. The activation function used for the hidden layers is ReLu. The output from the second FC layer is split into 2 layers - the first one to calculate

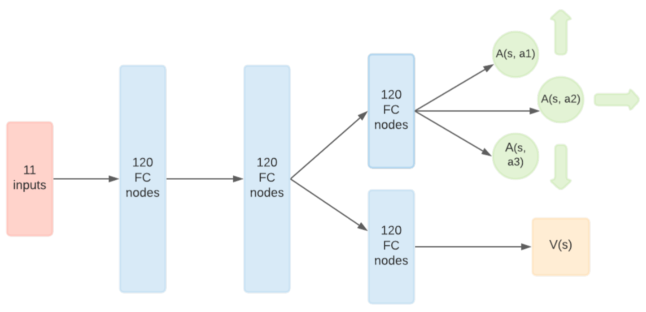

FIG. 1: Architecture of the Dueling DQN

the value of the state, and another to output the advantages of the potential actions for that state. For these layers, SoftMax activation function is used in order to reduce the output of the network in the range (0, 1).

III. RESULTS

Each agent played 100 games on all game modes: standard, with a wall and with a maze. The selected metrics for comparison are mean score and max score. The results are presented in tables I and II

	Prioritized DQN	Double DQN	Dueling DQN
Standard	45	37	45
Wall	33	9	21
Maze	4	3	5

TABLE I: Max score in 100 games

	Prioritized DQN	Double DQN	Dueling DQN
Standard	7.30	8.04	7.47
Wall	4.57	2.5	3.83
Maze	0.6	0.9	1.02

TABLE II: Mean score in 100 games

Prioritized DQN and Dueling DQN have equal max score for Standard Snake Game as shown in I and Double DQN achieved lower max score. On the other hand, Double DQN has higher mean score, so it has better overall score.

In wall mode Prioritized DQN's result are significantlly better than Dueling. Also, Double DQN's result is quite unsatisfactory compared to the other algorithms.

Maze mode was difficult for all the agents and their max score is low. From II we conclude that Dueling DQN achieved best results for this mode, because for most of the games its score is at least 1.

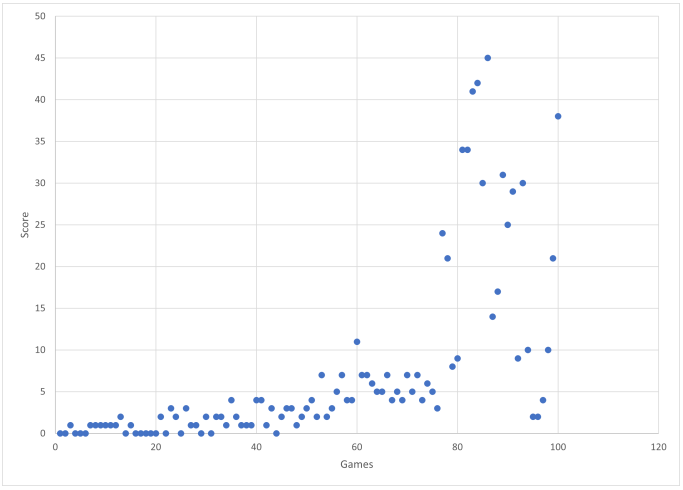

FIG. 2: Prioritized DQN for Standard Snake Game, 100 games

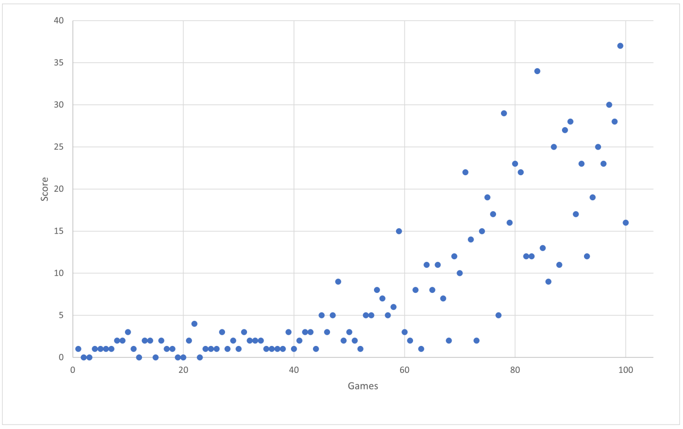

FIG. 3: Double DQN for Standard Snake Game, 100 games

From 2 and 3 can be seen that, although Double DQN achieves lower max score than Prioritized DQN, it tends to have increasing score and much less spread.

An attempt to use Convolutional Neural Network was made but the training was unsatisfactory slow and did not achieve good results - for 1000 games, max score was equal to 2. [7]

IV. CONCLUSION

This experiment adds to a growing corpus of research showing that the Deep Q-Learning approach provides a vast space for experimenting with games such as the "Snake". Our data indicates that the Dueling DQN and Prioritizied DQN are the reasonable choice when the environment contains obstacles, while the Double DQN can be used to reduce overoptimism, resulting in more stable and reliable learning. Maximum scores around 40-45 were achieved, but the agents reveal different behaviour for the multiple modes, in which they have been tested – for instance, in how they react for obstacles. Future research should consider the potential reward options more carefully, for example – the closer the snake gets to the food item, the higher the reward. Furthermore, given our results for Dueling DQN and Double DQN, an interesting experiment that might provide better performance would be to implement a Dueling Double DQN. The outcome of our experiments suggests that we still have a long way to go to in order to achieve extremely good results that outperform human results, but also verifies that the chosen approach is in the right direction as the results are optimistic.

V. ACKNOWLEDGEMENT

The implementation and results of the algorithms can be found in the following GitHub repository: `https://github.com/evigilatus/Snake_Bot`.

[1] D. S. A. G. I. A. D. W. M. R. Volodymyr Mnih, Koray Kavukcuoglu, (Playing Atari with Deep Reinforcement Learning).

[2] A.-C. Roibu, “Design of artificial intelligence agents for games using deep reinforcement learning,” .

[3] T. L. M. S. G. T. S. P. Aman Hemani, Vinay Shah, “Snake ai,” .

[4] J. Torres, “Deep q-network (dqn)-ii experience replay and target networks,” .

[5] D. Liu, *Introduction to Deep Reinforcement Learning* (2020).

[6] M. H. H. v. H. M. L. N. d. F. Ziyu Wang, Tom Schaul, (Dueling Network Architectures for Deep Reinforcement Learning).

[7] M. Z. A.-H. T. C. M. Y. Z. Zhepei Wei, Di Wang, “Autonomous agents in snake game via deep reinforcement learning,” .

[8] C. Yoon, “Dueling deep q networks,” .