Implementation of Neural Network Reinforcement Learning for Game Playing

"Avoiding Falling Asteroids"

In the partial fulfillment of
Mini Project, Artificial Intelligence

Subject Code: COMP 472

Kamal Shrestha
CE 4^{th} year
Roll No: 49

Submitted to:

Santosh Khanal
Lecturer, Artificial Intelligence
Department of Computer Science and Engineering
Kathmandu University
Dhulikhel, Kavre
Nepal

Date of Submission: 12^{th}March, 2020

Contents

List of Figures

List of Tables

Abstract

A reinforcement learning algorithm has been implemented to train an AI model to play a new "asteroid avoidance" game. The learning algorithm takes as an input a low- dimensional continuous game state which is initialized and evolves randomly throughout each game. A simple neural network then estimates the cost of each move (Q-learning), which is modeled after the moment generating function of the number of points collected in future. The learning is performed with pure exploration and parameters are adapted using stochastic gradient descent method.

1 Introduction

Obstacle avoidance is an important element of artificial intelligence research. It is widely used in the games and robots. Obstacle avoidance algorithm is one of the foundations of the AI decision-making in the robots field. In different environments, obstacle avoidance algorithm works in different forms and strategies depending on the type of obstacles. In some condition, the obstacles need to be explored, the scenes are unpredictable, local situation can only be obtained through some acquisition devices, and obstacles in the field of vision may also be unknown, the AI units require a special obstacle avoidance algorithm.

1.1 Problem Definition

The problem defined in this particular game environment for AI is to avoid falling asteroids (obstacles) and hitting the blue oval at the bottom of a movable cylindrical whole frame. The AI is not given any pre weighted Q values nor it is provided with any data to analyze ad take actions. The AI should solely rely on "hit and trail" methods to achieve reward points and make adjustments according to it. [1]

1.2 Objectives

The obstacle avoiding artificial intelligence is built with the following objectives:

- To successfully train an AI to avoid falling asteroids by making correct decisions
- To implement the concept of Reinforcement Learning, Q Learning, where Q values for value functions are predicted using Deep Neural Networks

2 Agent Description

An AI system is composed of an agent and its environment. The agents act in their environment. The environment may contain other agents. An agent is anything that can be viewed as :

- perceiving its environment through sensors and
- acting upon that environment through actuators

The game AI developed in this project is a learning agent, which can learn from its past experiences or it has learning capabilities.It starts to act with basic knowledge and then able to act and adapt automatically through learning.A learning agent has mainly four conceptual components, which are:

1. Learning element :It is responsible for making improvements by learning from the environment
2. Critic: Learning element takes feedback from critic which describes how well the agent is doing with respect to a fixed performance standard.
3. Performance element: It is responsible for selecting external action
4. Problem Generator: This component is responsible for suggesting actions that will lead to new and informative experiences.

PEAS measure for the agent are:

- P (Performance): The performance measure of the agent would be the game score, which is basically the number of asteroids it is able to dodge.
- E (Environment): All the surrounding things and conditions of an agent fall in this section. It basically consists of all the things under which the agents work.: Computer System.
- A (Actuators): The devices, hardware or software through which the agent performs any actions or processes any information to produce a result are the actuators of the agent: Display Screen where the scores are displayed.

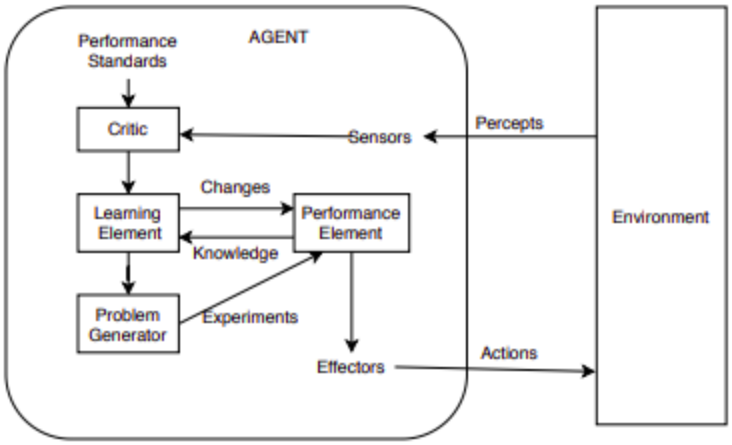

Figure 1: Learning Agent

- S (Sensors): The devices through which the agent observes and perceives its environment are the sensors of the agent. : The Position of Falling Asteroids in Game Environment, Virtual Distance Calibration

The agent here is autonomous, since it can perform tasks in pursuit of a goal with minimum of or direct supervision or direct control, but can interact with trained system to obtain output results. It may or may not have a user interface.

3 Agent Environment

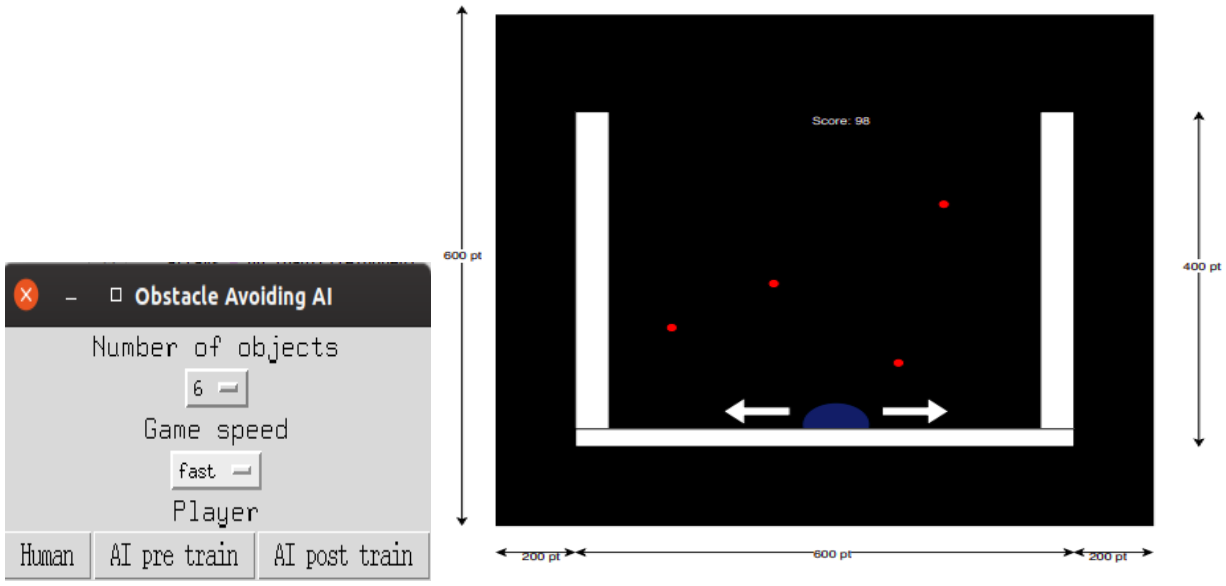

Figure 2: Game Environment

The complete description of the agent environment is given in the enumeration below:

1. **Fully Vs Partially Observable**: If an agent sensor can sense or access the complete state of an

environment at each point of time then it is a fully observable environment, else it is partially observable.An agent with no sensors in all environments then such an environment is called as un observable. Here the environment is a partially observable.

2. **Deterministic Vs Stochastic:** If an agent's current state and selected action can completely determine the next state of the environment, then such environment is called a deterministic environment.A stochastic environment is random in nature and cannot be determined completely by an agent. Here the environment is Deterministic.

3. **Episodic Vs Sequential:** In an episodic environment, there is a series of one-shot actions, and only the current precept is required for the action.In an episodic environment, there is a series of one-shot actions, and only the current precept is required for the action. here the Environment is Episodic

4. **Static Vs Dynamic:** If the environment can change itself while an agent is deliberating then such environment is called a dynamic environment else it is called a static environment. Static environments are easy to deal because an agent does not need to continue looking at the world while deciding for an action. Here the Environment is Dynamic because of the falling asteroids.

5. **Discrete Vs Continuous:** If in an environment there are a finite number of precepts and actions that can be performed within it, then such an environment is called a discrete environment else it is called continuous environment. A chess game comes under discrete environment as there is a finite number of moves that can be performed. A self-driving car is an example of a continuous environment. Here the environment is Discrete.

6. **Accessible Vs Inaccessible:** If an agent can obtain complete and accurate information about the state's environment, then such an environment is called an Accessible environment else it is called inaccessible. Here the environment is accessible.

4 Problem Specifications

We consider a game in which a blue semicircular object has to avoid red falling balls. The blue object is attached to the bottom of a U-shaped white frame, which is always moving either left or right. The direction of the frame is controlled by the player, and is the only player input throughout the game. The whole frame (with the blue object) moves left-right around the vertical 3-dimensional cylinder, in other words, it wraps around the left and right screen edges. The game ends when a red falling ball hits the blue object. A point is scored every time a red ball hits the white frame. At that point, the red ball in question disappears and a new red ball appears at the uniformly random location at the top of the white frame, keeping the number of red balls fixed. The red balls fall vertically with small random vertical and horizontal noise added. Various parameters control the number of red balls, sizes of game objects, relative speed of frame vs balls, amplitude and frequency of the noise ball movement, etc. Note that the parameters have to be adjust properly to avoid certain trivialities, such as balls never being able to reach the blue object when the frame direction is fixed. [5]

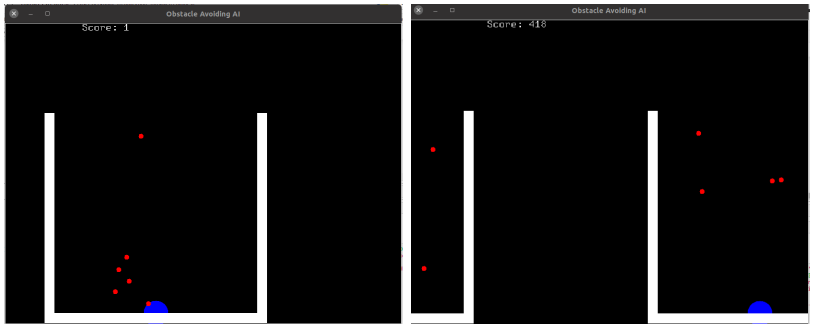

Figure 3: Collision (Left) and Avoidance (Right)

5 State Space Representation

Before an AI problem can be solved it must be represented as a state space. The state space is then searched to find a solution to the problem. A state space essentially consists of a set of nodes representing each state of the problem, arcs between nodes representing the legal moves from one state to another, an initial state and a goal state. Each state space takes the form of a tree or a graph. State-space searching assumes that:

1. the agent has perfect knowledge of the state space and can observe what state it is in (i.e., there is full observability)
2. the agent has a set of actions that have known deterministic effects
3. some states are goal states, the agent wants to reach one of these goal states, and the agent can recognize a goal state
4. a solution is a sequence of actions that will get the agent from its current state to a goal state.

Here for this particular avoiding game, the actions of the agent is to movement in left or right direction based on the position of the falling asteroids. These asteroids are randomly generated so there won't be a predefined goal state rather the goal of the agent is to keep on avoiding the objects. Now the decision to move either right or left is based on the value function or Q value determined using the deep neural network. Based on the position of the falling asteroids, which are the linear inputs to the neural network, weighted by θ factors, which are adjusted using Backpropagation ALgorithm [4] and the Respective Q value is provided to the agent to make its decision either to move left or right.

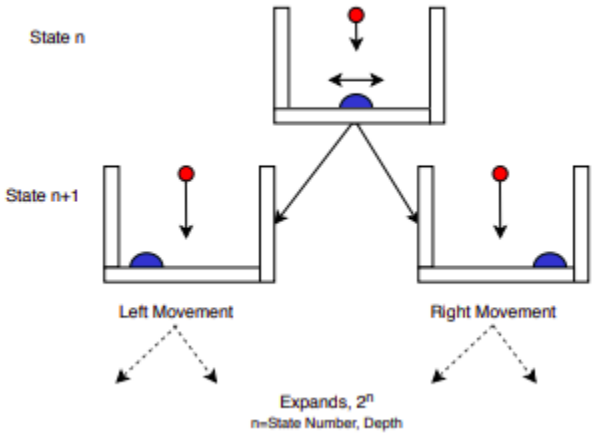

Figure 4: State Space Tree Representation

6 Proposed Approach:

6.1 Reinforcement Learning

Reinforcement Learning is an area of machine learning where an agent tries to maximize the reward in a particular situation by learning to behave and perform certain actions in an environment.[6] Reinforcement learning solves a particular kind of problem where decision making is sequential, and the goal is long-term, such as game playing, robotics, resource management, or logistics.
We've seen a lot of improvements in this fascinating area of research in the recent years. DeepMind and the Deep Q learning architecture in 2014 won against the champion of the game of Go with AlphaGo in 2016, OpenAI and the PPO in 2017.With the advancements in Robotics Arm Manipulation, Google Deep

Mind beating a professional Alpha Go Player, and recently the OpenAI team beating a professional DOTA player,advancements in Robotics Arm Manipulation, the field of reinforcement learning has really exploded in recent years.

In reinforcement learning, a learning agent take actions affecting the state of the environment and have goals relating to the environment. The input is an initial state from which the model start. There are many possible outputs as a particular problem can be solved by variety of solution. The training is based upon the input, the model returns a state and the user decides whether to reward or punish the model based on its output. Thereafter, the best solution is decided based on the maximum reward. Reinforcement Learning is the closest to the kind of learning that humans and other animals do among all the other forms of machine learning algorithms.

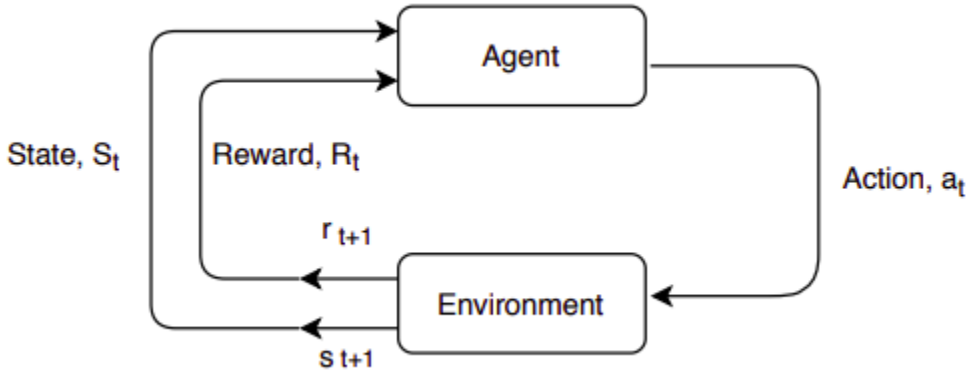

Figure 5: Reinforcement Learning

Some key terms that describe the elements of a RL problem are:

- Environment: The physical world in which an agent operates : The current situation of an agent
- State: The current situation of an agent
- Policy: Method to map agent's state to actions.
- Value: Future reward that an agent would receive by taking an action in a particular state

A Markov decision process (MDP) is a discrete time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. [2] Basic reinforcement is modeled as a Markov decision process:

- a set of environment and agent states, S
- a set of actions, A, of the agent
- $$Pa(s, s) = Pr(st + 1 = s|st = s, at = a)$$

  is the probability of transition from state s to state s under action a
- Ra (s, s) is the immediate reward after transition from s to s with action a
- rules that describe what the agent observes

6.1.1 Elements of Reinforcement Learning

Other than the agent and the environment, one can identify four main sub elements of RL

- Policy — The policy is the core of a reinforcement learning agent which maps from perceived states of the environment to actions to be taken when in those states
- Rewards — A reward is a scalar feedback signal which is sent by the environment to the reinforcement learning agent on each time step. The agent's sole objective is to maximize the total reward it receives in the long run at the end or in the intermediate period.
- Value Function — Value function the total amount of reward an agent can expect to accumulate over the future, starting from that state.
- Model of the environment — The model mimics the behavior of the environment, that allows inferences to be made about how the environment will behave.

6.1.2 Model policy

A policy is a function that takes a state 's' as an input and returns a action 'a':

$$\pi(s) \rightarrow a$$

Thus, the policy is typically used by the agent to decide an action 'a' when it is in a given state 's'. Sometimes, the policy returns a probability distribution over a set of actions instead of a unique action. In such a case, the policy is stochastic.

6.1.3 Value Function

A value function defines the expected cumulative reward of taking action 'a' in state 's' for a given policy.The value function gives you the value at each state. With the bellman equation, it looks like this.

$$V(S) = max_a[R(s,a) + \gamma \sum s'T(s,a,s')V(s')]*$$

Where the value at a particular state is the maximum action that gives you the reward plus the discounted reward.

6.2 Q-Learning

By now, we have got the following equation which gives us a value of going to a particular state (form now on, we will refer to the rooms as states) taking the stochasticity of the environment into the account:

$$V(S) = max_a[R(s,a) + \gamma \sum s'T(s,a,s')V(s')]*$$

Q-Learning [7] poses an idea of assessing the quality of an action that is taken to move to a state rather than determining the possible value of the state (value footprint) being moved to. We select our next action based on our behavior policy, but we also consider an alternative action that we might have taken, had we followed our target policy. This allows the behavior and target policies to improve, making use of the action-values Q(s, a). The process works similarly to off-policy Monte Carlo methods.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma maxQ(S_{t+1}, a) - Q(s_t, A_t)]$$

6.2.1 Deep Q Learning

Neural networks [3] are function approximators, which are particularly useful in reinforcement learning when the state space or action space are too large to be completely known.

A neural network can be used to approximate a value function, or a policy function. That is, neural nets can learn to map states to values, or state-action pairs to Q values. Rather than use a lookup table to store, index and update all possible states and their values, which impossible with very large problems, we can train a neural network on samples from the state or action space to learn to predict how valuable those are relative to our target in reinforcement learning. At the beginning of reinforcement learning, the neural network coefficients may be initialized stochastic ally, or randomly. Using feedback from the environment, the neural net can use the difference between its expected reward and the ground-truth reward to adjust its weights and improve its interpretation of state-action pairs.This feedback loop is analogous to the back propagation of error in supervised learning.

6.3 System Architecture

The overall system architecture consists of three Neural Network Layers that maximize the Q value for the actions to be taken by the AI based upon immediate and future rewards, squeezed by the sigmoid activation functions. There are two parameters to be adjusted, which are the significant weights to each position of the N falling objects, namely $\theta 1$ and$\theta 2$.

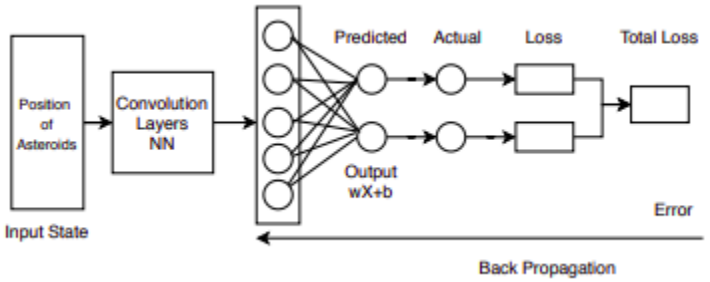

Figure 6: Condensed Architecture (Deep Q Learning)

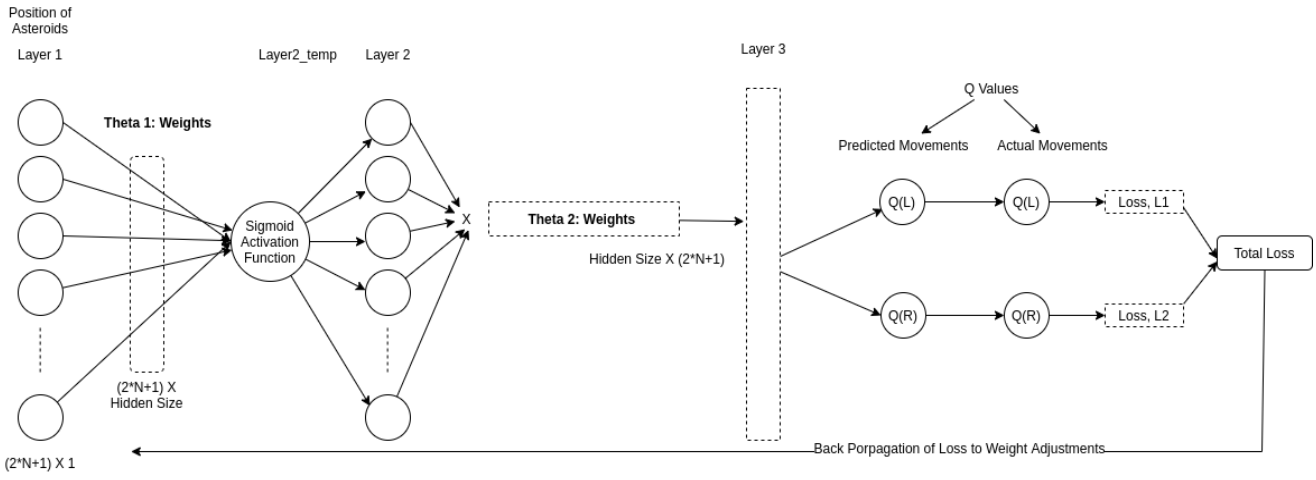

Figure 7: Overall System Architecture

7 Discussion on Results

Pre-train performance with random parameters is very poor, scoring around 10 points. The performance for the trained model is tested with N = 4, 5, 6 red balls. The average, median score, deviation, and 95% confidence intervals are shown in the following table under the training scheme I) is as follows.

N	median	average	95% conf. interval for average
4	238	342.14	342.14 ±9.33
5	180	256.30	256.30 ±6.99
6	162	233.62	233.62 ±6.33

Table 1: Table 1: Results of a Post Trained AI

Under the training scheme there was a serious improvement for N = 4 balls case, achieving an average score of 906 points (±10% for 95% confidence interval)with the median score of 656 points. Each training was performed on Intel's i3 CPU with 4GB of RAM and took several hours. No serious human performance has been explored to compare with the above values, and naturally human performance will be drastically affected by the "game speed". However, for the only experienced player (the author) it seems to be extremely difficult to consistently achieve thee above results when played on the implemented "fast" level.

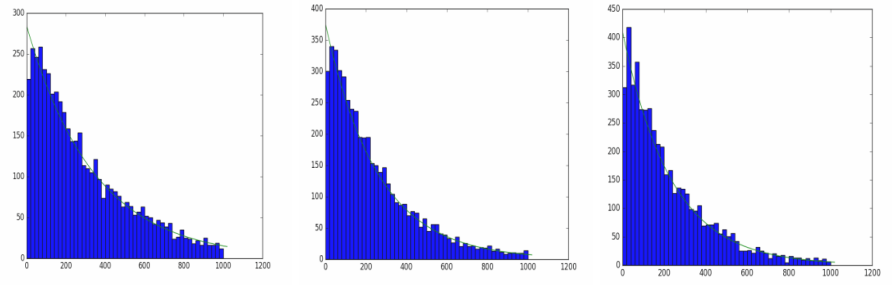

Figure 8: Histogram of trained scores for N = 4, 5, 6 (under training scheme I) and the predicted smoothed histogram for a geometric distribution with the appropriate mean. Scores are cutoff at 1000 points.

Below are learning curves for average game scores, for the model trained with the same parameters stated above and used in the application. Under the scheme I) the curves are the result of 200 learning episodes each having n train = 1000 learning games and n test = 400 test games.

8 Conclusion and Future Works

The post trained AI performed really well scoring an average score above 230 in every level of the avoiding game. The training complexity of the network is $O(n^2)$ as with in each training session, there are a number of games to be played. The reward is based on the number of successful obstacles avoided and negative reward for getting hit by it. Overall, the score of 230+ is pretty inspiring taking into account the input to the network is only the position of the falling obstacles. In the future improvements, the AI will be subjected to increased number of asteroids with increased speed, and the results will be evaluated. At this moment on the project, the game speed is made constant as pre-selected by the player in the initial splash screen, where as in the future updates the game speed can be increased with the increase in game scores. With that addition to the game environment, the AI need to be trained in the basis of the speed of the falling objects as well, not only the position of N given asteroids.

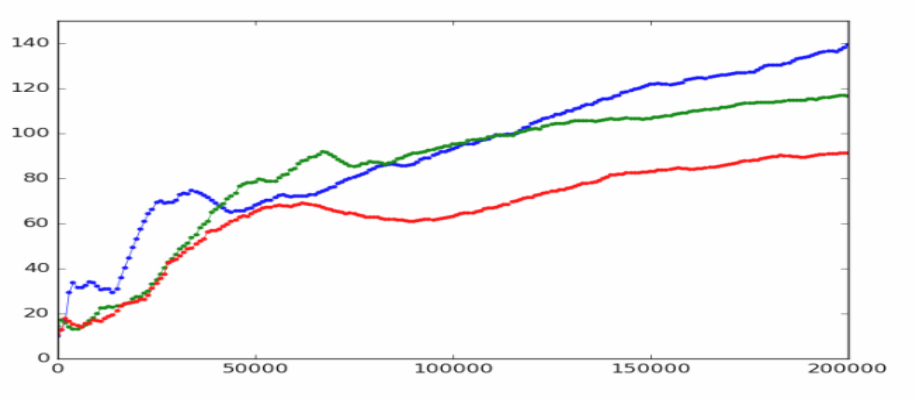

Figure 9: Learning curves for N = 4 (blue), N = 5 (green) and N = 6 (red) under the training scheme I). There are 200 train sessions, each with 1000 games.The first figure represents the average test scores of each test episode, while the second one running average across all the test games so far since the beginning of the training. Final average test scores for N = 4, 5, 6 are 367.51, 100.7325, and 70.5825 respectively. Best average test scores for N = 4, 5, 6 are 367.51, 249.975, and 204.315 respectively.

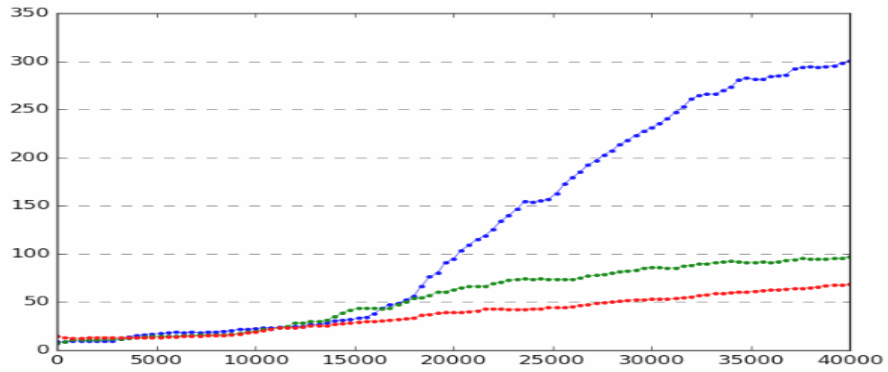

Figure 10: Same learning curves for N = 4 (blue), N = 5 (green) and N = 6 (red) under the training scheme II). There are 100 train sessions, each with 400 games. Final average test scores for N = 4, 5, 6 are 544.605, 234.2275, and 181.65 respectively. Best average test scores for N = 4, 5, 6 are 905.8475, 255.195, and 181.65 respectively.

References

[1] Richard S Chang and George J Klose. *Obstacle game*. US Patent 4,162,792. July 1979.

[2] Robert Givan, Sonia Leach, and Thomas Dean. "Bounded parameter Markov decision processes". In: *European Conference on Planning*. Springer. 1997, pp. 234–246.

[3] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[4] Jing Li et al. "Brief introduction of back propagation (BP) neural network algorithm and its improvement". In: *Advances in computer science and information engineering*. Springer, 2012, pp. 553–558.

[5] Haiqing Shen et al. "Automatic collision avoidance of multiple ships based on deep Q-learning". In: *Applied Ocean Research* 86 (2019), pp. 268–288.

[6] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge, 1998.

[7] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.