Deep Q-Learning for Atari Space Invader

Score Optimization

**Raman Srivastava**   
*Computer Science   
University of Texas at Dallas*Richardson, United States  
rks170530@utdallas.edu

**Dhruv Mehta**  
*Computer Science*  
*University of Texas at Dallas*Richardson, United States  
dxm180048@utdallas.edu

*Abstract*—For this project, we implement a Reinforcement Learning algorithm known as Deep Q-Learning using a Convolutional Neural Network and a Double Deep Q-Learning algorithm in order to optimize scores. The scores in question are optimized from the Atari game, Space Invaders, using the OpenAI Gym library and the algorithms are implemented from scratch, or without the use of the Keras Library.

Keywords—Deep Q-Learning, Convolutional Neural Network, Double Deep-Q Learning, Space Invaders, OpenAI Gym.

# Introduction

In Machine Learning, the area of Reinforcement learning relates to an algorithm training itself with a user defined system of rewards and punishments. This algorithm is particularly used in complex environments, where the algorithm, also called an agent is able to traverse to the environment and discover the optimal paths to maximize the greatest user-defined rewards. In addition to maximization of the rewards, for many environments there is also the element of penalty that the algorithm must choose to avoid. Thus, the agent has another goal in order to minimize the penalty that the user has defined.

Video games are well documented in having complex environments for the player to explore, since the introduction of pong in the late 1900s. Thus, Reinforcement learning has been widely used in order to maximize the reward in videogames, from Pong, to Snake and Frogger, including the popular T-Rex Game that can be found on Google Chrome when the internet is down. The goal of the Reinforcement learning algorithms in questions is to optimize on the actions and the proper states to yield those actions without being told explicitly which path or sub-paths are optimal and which ones are not.

In order to apply a Reinforcement learning algorithm to a 2D space, in this case the Space Invaders game from Atari, one needs to process each image or frame that corresponds to each individual state on the screen. This is possible via a

Convolutional Neural Network, that takes a raw image’s frame and learns the policies that would be hard to discern by a human or a simple algorithm alone. Applying Q-learning, a type of Reinforcement algorithm to a CNN and its respective outputs is called Deep Q-learning, where the “Deep” stands for the Deep Neural Network itself.

# Background Work

## OpenAI Gym

First, we wanted to use a standardized library that was really popular for comparing Reinforcement learning algorithms. Since Reinforcement learning is a relatively new concept, an approach has not been standardized until 2016, with the introduction of *Gym* by OpenAI [1]. Gym was designed by OpenAI to provide the unsupervised learning equivalent of ImageNet, a popular labeled dataset used for CNN accuracy testing. It was also created to provide the same, standard environment that can be used for publications, many of which can be found online.

Overall, it has a dedicated library for Atari games, including two versions of Space Invaders. The first version, called “SpaceInvaders-ram-v0” uses RAM as and input, while the second version, “SpaceInvaders-v0”, uses images itself as input. Since we plan on implementing a CNN for the project, we decided to use “SpaceInvaders-v0” for our preprocessing.

## Atari Research Playground

As OpenAI were able to standardize a Reinforcement learning library back in 2016, *Atari* by Gsurma aims to create one specifically catered towards the Atari games using the open MIT license [2]. Using the Keras library, which we are not allowed to use for the project, Gsurma was able to test different Reinforcement learning approaches after about 90 hours of training.

Pertaining to the SpaceInvaders-v0 game itself from Gym, they found that the human average for the game was somewhere in the neighborhood of 300 points. ­­­

# Defining the Task

Using the Gym’s “SpaceInvaders-v0” as a guide, we formally define the rules that the agent will follow. The picture above has an environment, denoted as *E.* This environment, reading Gym’s documentation is an emulated game of Atari’s Space Invaders, that uses images as outputs. In the NTSC version of Atari Space Invaders, intended for the North American and Japanese markets, the game is emulated to run at 60 frames per second, or 60 FPS [3]. Thus, each frame, 1/60th of a second by default represents a state *s.* The player, in this case the agent, is able to move the ship six different choices. These choices are referred to as actions, or *a*, which can be applied generally into any Reinforcement learning algorithm. Since there are six actions for the user to take, *a* becomes an integer between one and six inclusively, or *a =* [1, 6].

## Actions

The six actions that the agent can take are:

1.) *Fire—* Allows the ship to fire a laser without

moving.

2.) *Left—* Allows the ship to move left 1 space.

3.) *Left & Fire—* Allows the ship to shoot first, and

move left 1 space in one action.

4.) *Right—* Allows the ship to move right 1 space.

5.) *Right & Fire—* Allows the ship to shoot first, and

move right 1 space in one action.

6.) *Inaction—* Allows the ship to skip the frame

without action.

## Rewards

Traditionally, rewards in Space Invaders is done through a point system, where destroying a ship yields 5 points without any modifiers. However, for the sake of tracking the experiment, the reward and penalty given to the agent for completing an action is denoted by a single variable, *r*, that is normalized to be in the range between -1, indicating the worst penalty, and 1, indicating the best reward. Thus *r* = [-1, 1].

# Theoretical Approach

The images provided by “SpaceInvaders-v0” from Gym, are RGB or within a 256-bit value, from [0, 255] to denote the different colors. The resolution that Atari uses is 260 pixels, by 160 pixels­­. Thus the space for each frame, or state *s* has a size 210 x 160 x 3, or [210, 160, 3]. In order to apply Reinforcement learning to “SpaceInvaders-v0” game, we had to start with Q-Learning.

## Q-Learning.

Using state, action, and reward *s, a,* and *r* respectively, the agent’s goal is to maximize the future rewards. Using a discount factor, γ, to penalize future actions the agent will take, we can represent the value function Qopt(s, a) to follow an optimal policy. This policy will be described with a transitional probability between *s* 🡪 *s’*, representing the action to take it from one state *s* to the next state *s’.* Thus, with a transition probability of *T*, and the expected rewards *r* from an action, the theoretical Q-function becomes:

However, for the Space Invaders game, and many video games in general, the transitional probability *T* and its respective reward from the action *r* are hidden and thus unknown for each frame transition. Thus, a Neural Network is used in order to approximate the Q-function via weights. Thus Qopt(s, a) becomes ω\* ∅(s, a) with a learning rate η in order to apply gradient descent. Thus, the approximate Q-function to Eq. 1 becomes:

## Deep Q-Networks

In the case of Space Invaders, however, the resolution is complicated for a neural network since it contains over 33,600 pixels worth of information. Thus, looking at each pixel and trying to interpolate data from it becomes quite challenging for a normal Neural Network to handle the process. Thus, using a pixel space of *n* by *m* for a matrix, we can map each Atari Space Invaders frame to a Convolutional Neural Network, or CNN. Thus, the dimensions are mapped from the frames, or states *n* to the number of possible actions, mapped *m*, where *m* > *n.* Thus, using *θi* as the new weights to be updated, the updated equation for the Q-network, based on Eq. 2 becomes:

## Double Q-Learning

#### Since Q-learning is updated with Max(s’, a’), denoted as Vopt(s’), Q-learning tends to overestimate the correct weights trying to find the optimum weights. With an overoptimistic learning, an improvement can be made by keeping two values for the wieghts everytime they would be updated. This can be donoted simply be θ and θ’ where θ can continue to use the greedy policy assigned to it. The other set of weights, θ’, however tries to determine the value of θ. This process can be written for the target value of YtQ as r + γMax(s’, a’). Thus, the new target value can be written as:

# Results and Analysis

#### Explanation of table 1:

* Minibatch\_size= the size of training dataset used by RMSProp Algorithm
* **Replay memory size**: It will store the most recent states, actions, next stare , reward and whether terminated or not
* Discount Factor : It is used to reduce the effect of future q values while calculating q value for current state
* Learning Rate : Learning rate used to update the weight parameter of CNN
* Min squared gradien(epsilon): constant added to the gradient
* Square Gradient ( Rho) : It is used in RMS Prop to
* Initial exploration/ Final exploration (eplison) :It is used to decide whether to select action randomly or using CNN
* No op = maximum number of none action the agent can take at start

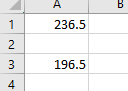
**Output Results**

We train the game using the following parameters:

Maximum number of steps : 500000

Maximum number of runs :20

The highest score the agent was able to get at the end of 20 runs is 236.5



The highest score after running the program for same configuration four times was 260.

# Conclusion and Future work

Based on our results, we argue that using Q-learning is a viable and effective form of Reinforcement learning that can apply to many different environments. Specifically, in a game such as Space Invaders, the Deep-Q learning algo-rithms can interpret a highly complex state representation, and yield respectable outputs.

If given enough time accuracy and score would continue to increase until it would stagnate, depending on the run, results can vary significantly. This is shown in Figure 1, where although the average of 100 runs makes it appear that the algorithm was consistent in the runs tested by Gsurma, the reality was that they were widely inconsistent, sometimes dropping below that of a human level.

Thus, we believe that although Deep-Q and Double Deep Q-learning algorithms are very powerful tools in environments with very complex state spaces, they are very difficult to train. Whether to explore or exploit is a decision which becomes very difficult when given only a limited amount of information, where the states cannot be explored to the fullest of their ability.

Given the timeframe we were allotted to for the project, we feel that the results were respectable. However, in future works we would like to compare against the outputs of the Keras library as well as compare Double Deep Q-learning algorithm versus the regular Deep Q-learning algorithm and compare how much of a score point, on average, separate them. We are well aware that Deep Q-learning tends to overestimate, as discussed in our theoretical approach, but we were not able to test directly and prove such a thing due to time constraints.

As for the future, we would enjoy testing these algorithms to other Atari and even Nintendo Entertainment System games like Tetris to compare if Deep-Q does better or worse in those environments. Although implementing Deep Q-learning algorithms was a constraint assigned to us for the project, for the future we would stick to the Keras library, as it allows for more consistent comparisons between games and algorithms.

# References

1. “Gym: A Toolkit for Developing and Comparing Reinforcement Learning Algorithms,” *Gym*, 2016. [Online]. Available: https://gym.openai.com/docs/.
2. G. Surma, “Atari - Solving Games with AI (Part 1: Reinforcement Learning),” *Medium*, 17-Jan-2019. [Online]. Available: https://towardsdatascience.com/atari-reinforcement-learning-in-depth-part-1-ddqn-ceaa762a546f.
3. A. Sagar, “Deep Reinforcement Learning Tutorial with Open AI Gym,” *Medium*, 26-Nov-2019. [Online]. Available: https://towardsdatascience.com/deep-reinforcement-learning-tutorial-with-open-ai-gym-c0de4471f368.
4. Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou Daan Wierstra Martin Riedmiller <https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf>
5. Charel, “Learn by example Reinforcement Learning with Gym,” *Kaggle*, 04-May-2019. [Online]. Available: https://www.kaggle.com/charel/learn-by-example-reinforcement-learning-with-gym.