B.Tech Project Report

Decision making using deep reinforcement learning

Submitted in partial fulfilment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering

Submitted by

Jayadeep K M (Roll No 13400030)
Kevin Joseph (Roll No 13400032)
Mohammed Nisham K (Roll No 13400038)

Under the guidance of

Mr. Vipin Vasu A V

Department of Computer Science and Engineering
College of Engineering, Trivandrum
Kerala
May 2017

CERTIFICATE

This is to certify that the thesis entitled “Decision making using deep reinforcement learning” is a bonafide record of the major project done by **Jayadeep K M** (Roll No 13400030), **Kevin Joseph** (Roll No 13400032) and **Mohammed Nisham K** (Roll No 13400038) under my supervision and guidance, in partial fulfilment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering from the University of Kerala for the year 2017.

Mr. Vipin Vasu A V
(Guide)
Asst. Professor
Dept. of Computer Science and Engineering

Mrs. Liji
Professor and Head
Dept. of Computer Science and Engineering

Place: Trivandrum
Date: 11-05-2017

Abstract

Creating a general purpose AI has been an area of research since the beginning of computers and programming. Reinforcement learning is a major step towards a general purpose AI.

This project is aimed at creating a program that can learn to make decisions in an environment that is defined by a high-dimensional input, and has sparce and time delayed rewards for these actions. Such programs can be useful in problems where decsion must be made based on high dimensional sensory input such as camera feed. This project uses Q-learning algorithm to assign a quality value to each action in a state of the environment.

Atari games are used to demonstrate this approach, by training the program to play breakout game for upto 50 epochs and observing performance improvement. The trained neural network was saved and tested at the end of every epoch. The performance parameters like average q-value, average reward, games per epoch were also saved. The performace parameters showed a clear rise in performace for breakout (50 epochs) and space invaders (8 epochs).

The project has applications in the field of IOT, security, gaming, stock market analysis and traffic control systems. Any system that can be modelled as an environment with actions and rewards can be trained using this algorithm.

Contents

List of Figures

Abbreviations

AI	Artificial Intelligence
RL	Reinforcement Learning

1 Introduction

1.1 Motivation and Overview

Learning to control agents directly from high-dimensional sensory inputs like vision and speech is one of the long-standing challenges of Artificial Intelligence and Machine Learning. Most successful AI applications have relied on hand-crafted features combined with linear value functions or policy representations. Clearly, the performance of such systems heavily relies on the quality of the feature representation.

Recent advances in deep learning have made it possible to extract high-level features from raw sensory data, leading to breakthroughs in computer vision and speech recognition. These methods utilise a range of neural network architectures, including convolutional networks, multilayer perceptrons, restricted Boltzmann machines and recurrent neural networks, and have exploited both supervised and unsupervised learning.

However the main advantage of Reinforcement Learning is that it does't need huge amounts of hand-labelled data and does not depend too much on the feature representation. It learns from a reward signal that maybe delayed, noisy and sparse. The delay between actions and the rewards, which maybe thousands of timesteps seems like a particularly hard problem in RL when compared to direct association between action and reward in supervised learning. Another issue is that most problems consider all the data samples independent of each other, but in this case we need to consider the fact that the reward at the end of a session is not just the result of the last action but the result of the sequence of actions from the start of the session.

This project is aimed at using a Convolution Neural Network along with the Q-learning algorithm to solve the above problems and make decisions based on video input from the Atari Learning Environment. Our goal is to create a single neural network agent that is able to successfully learn to play atleast 2 games with no change in agent algorithm. The network was not provided with any game-specific information or hand-designed visual features, and was not privy to then internal state of the emulator, it learned from nothing but the video input, the reward and terminal signals, and the set of possible actions just as a human player would.

1.2 Background and Literature Survey

Deep Mind and Reinforcement Learning The presented method and algorithm was first presented in a paper by a company Deep Mind which was later aquired by Google. The paper presented the algorithm as a general purpose RL algorithm that could play seven of the atari 2600 games, it showed human level play in four and super human play in three of them. Consider the game Breakout. In this game you control a paddle at the bottom of the screen and have to bounce the ball back to clear all the bricks in the upper half of the screen. Each time you hit a brick, it disappears and your score increases you get a reward.

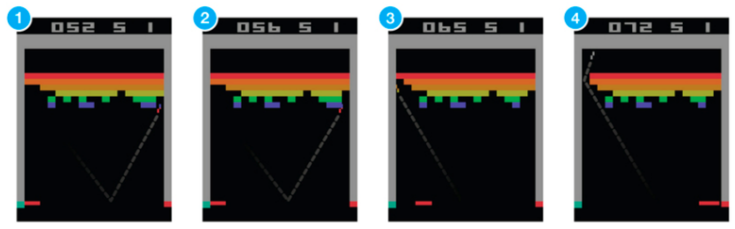

Figure 1: Breakout screens.

Suppose you want to teach a neural network to play this game. Input to your network would be screen images, and output would be one of four actions left, right, do nothing or fire to launch the ball. It would make sense to treat it as a classification problem for each game screen you have to decide, Which action to take. Sounds straightforward, Sure, but then you need training examples, and a lots of them. Of course you could go and record game sessions using expert players, but thats not really how we learn. We dont need somebody to tell us a million times which move to choose at each screen. We just need occasional feedback that we did the right thing and can then figure out everything else ourselves.

This is the task reinforcement learning tries to solve. Reinforcement learning lies somewhere in between supervised and unsupervised learning. Whereas in supervised learning one has a target label for each training example and in unsupervised learning one has no labels at all, in reinforcement learning one has sparse and time-delayed labels, the rewards. Based only on those rewards the agent has to learn to behave in the environment.

While the idea is quite intuitive, in practice there are numerous challenges. For example when you hit a brick and score a reward in the Breakout game, it often has nothing to do with the actions (paddle movements) you did just before getting the reward. All the hard work was already done, when you positioned the paddle correctly and bounced the ball back. This is called the credit assignment problem i.e., which of the preceding actions was responsible for getting the reward and to what extent.

Once you have figured out a strategy to collect a certain number of rewards, should you stick with it or experiment with something that could result in even bigger rewards? In the above Breakout game a simple strategy is to move to the left edge and wait there. When launched, the ball tends to fly left more often than right and you will easily score about 10 points before you die. Will you be satisfied with this or do you want more? This is called the explore-exploit dilemma should you exploit the known working strategy or explore other, possibly better strategies.

Reinforcement learning is an important model of how we (and all animals in general) learn. Praise from our parents, grades in school, salary at work these are all examples of rewards. Credit assignment problems and exploration exploitation dilemmas come up every day both in business and in relationships. Thats why it is important to study this problem, and games form a wonderful sandbox for trying out new approaches.

2 Materials and Methodology

2.1 Algorithms

2.2 Markov Decision Process

Suppose you are an agent, situated in an environment (e.g. Breakout game). The environment is in a certain state (e.g. location of the paddle, location and direction of the ball, existence of every brick and so on). The agent can perform certain actions in the environment (e.g. move the paddle to the left or to the right). These actions sometimes result in a reward (e.g. increase in score). Actions transform the environment and lead to a new state, where the agent can perform another action, and so on. The rules for how you choose those actions are called policy. The environment in general is stochastic, which means the next state may be somewhat random (e.g. when you lose a ball and launch a new one, it goes towards a random direction). The set

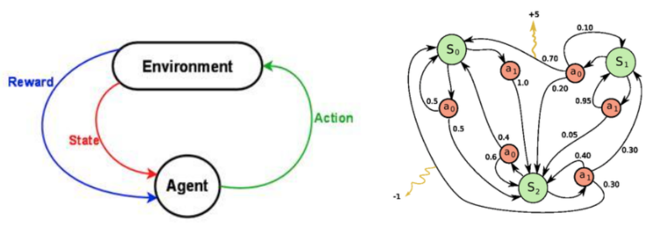

Figure 2: Breakout screens.

of states and actions, together with rules for transitioning from one state to another, make up a Markov decision process. One episode of this process (e.g. one game) forms a finite sequence of states, actions and rewards:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_{n-1}, a_{n-1}, r_n, s_n$$

Here s$_i$ represents the state, ai is the action and ri+1 is the reward after performing the action. The episode ends with terminal state sn (e.g. game over screen). A Markov decision process relies on the Markov assumption, that the probability of the next state si+1 depends only on current state si and action ai, but not on preceding states or actions.

2.2.1 Q-Learning

2.2.2 Deep Q-Learning

Neural Networks

2.3 Program Development

2.3.1 System Description

2.3.2 Class Diagram

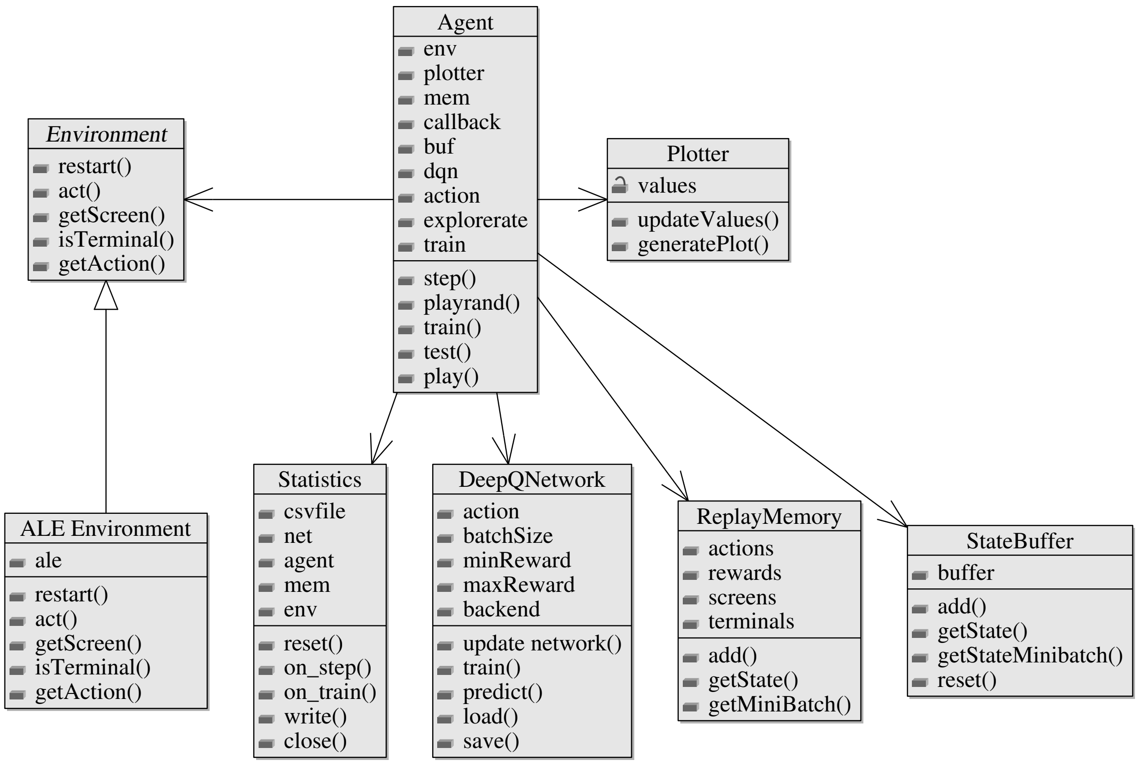

Figure 3: UML Class Diagram.

2.3.3 Code Overview

3 Results and Discussions

3.1 Description of observed strategies

3.2 Screenshots

3.3 Result Visualization

4 Further Work

4.1 Gamification of Problem

Solving real world problems can be achieved using Deep Reinforcement Learning. The following points are to be considered to convert a real world problem into a deep learning optimisation problem:

- The problem is to be modelled as a game.
- The game should include an environment which has a finite set of actions.
- High dimensional input to the environment should be available as input and savable as a state.
- State of the system should change according to the actions performed on it.
- The environment should occasionally give off rewards in response to its current state or action performed.

If all of the criterions are met and a gamified version of the real world problem is made, then it can be quite easily given to a deep reinforcement learning system for optimisation and then later used in real life after training.

4.2 Traffic Light Control

A sample real world problem that could be solved using reinforced learning.

4.2.1 Problem

There is an intersection of two roads, and four traffic lights control the traffic in it, each controlling one road. The control of the traffic light should be given to an AI for optimum traffic flow through the intersection. To design a system that can be used to train this AI.

4.2.2 Minimal Input

Traffic camera images at four different directions are taken as input to the system, mulitple images are taken to incorporate movement of the vehicles. Even though the system could work with the raw images, it would be impossible to train the system in the real world. So a simulation is required to train it. But in a simulation output images cannot be generated as complex as real video stream.

The solution is to simplify the input to the system, and to simulate the output in this simplified version. The approach taken is to convert the vehicles into white rectangles in a black canvas, and the intersection as a line. Thus white rectangles passing over the line will be taken for rewards and any intersection of rectangles will be interpreted as a crash. The rectangles are easier to simulate given the actions to be performed.

To make the simulation closer to reality, delay between the signal turning green and the first car moving can be given. Contraction and elongation of traffic at stop and start can also be simulated.

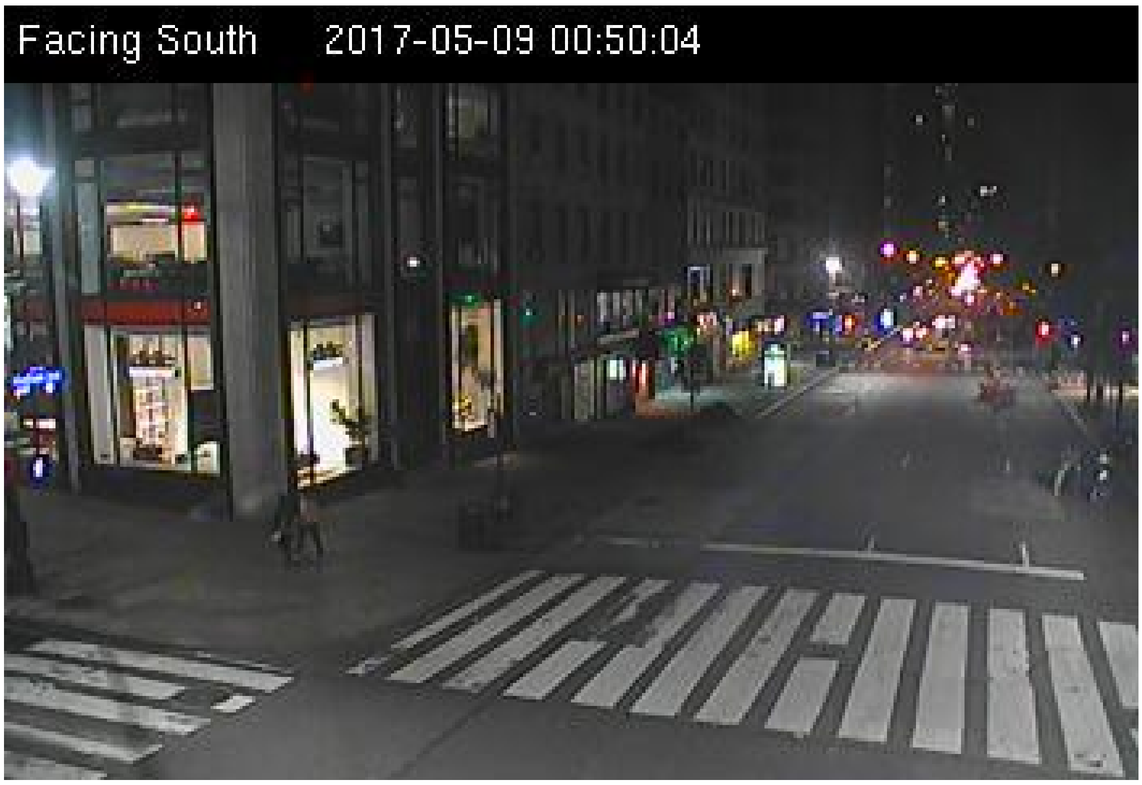

Figure 4: Empty Intersection.

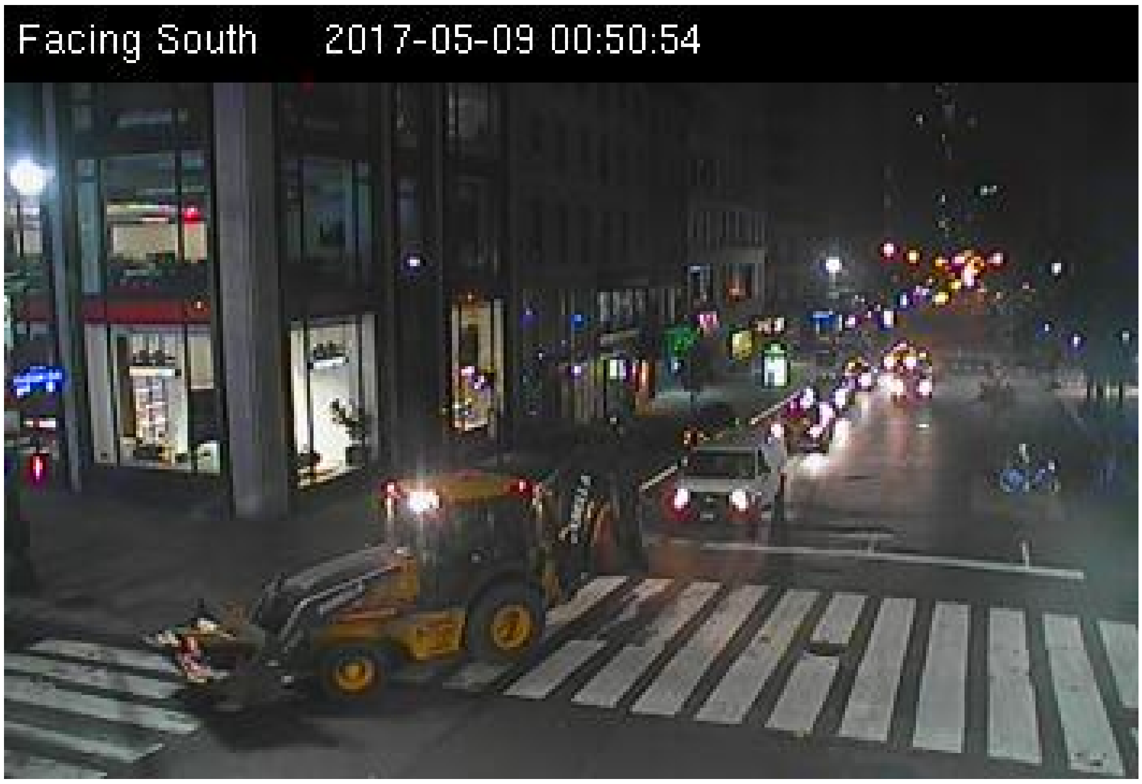

Figure 5: Intersection with Vehicles.

4.2.3 Action set

At any instant, any of the traffic lights can be any of the following:

- Red
- Orange
- Green in the Right, Left or Straight direction or any combination of it.

Orange can be eliminated by making it mandatory for transitions between Green and Red. Green has seven states (not to be confused with state of the system found from the input) it can be in for all the combinations. Since only Green or Red can be active at a given time, the total no of states for a single traffic light is 8.

The intersection consists of 4 traffic lights, operating independantly (No constraints to its operation) making the total no of actions the system can take to 8^4 *ie* 4096 actions.

Thus at every state of the system, it can decide between 4096 actions to take.

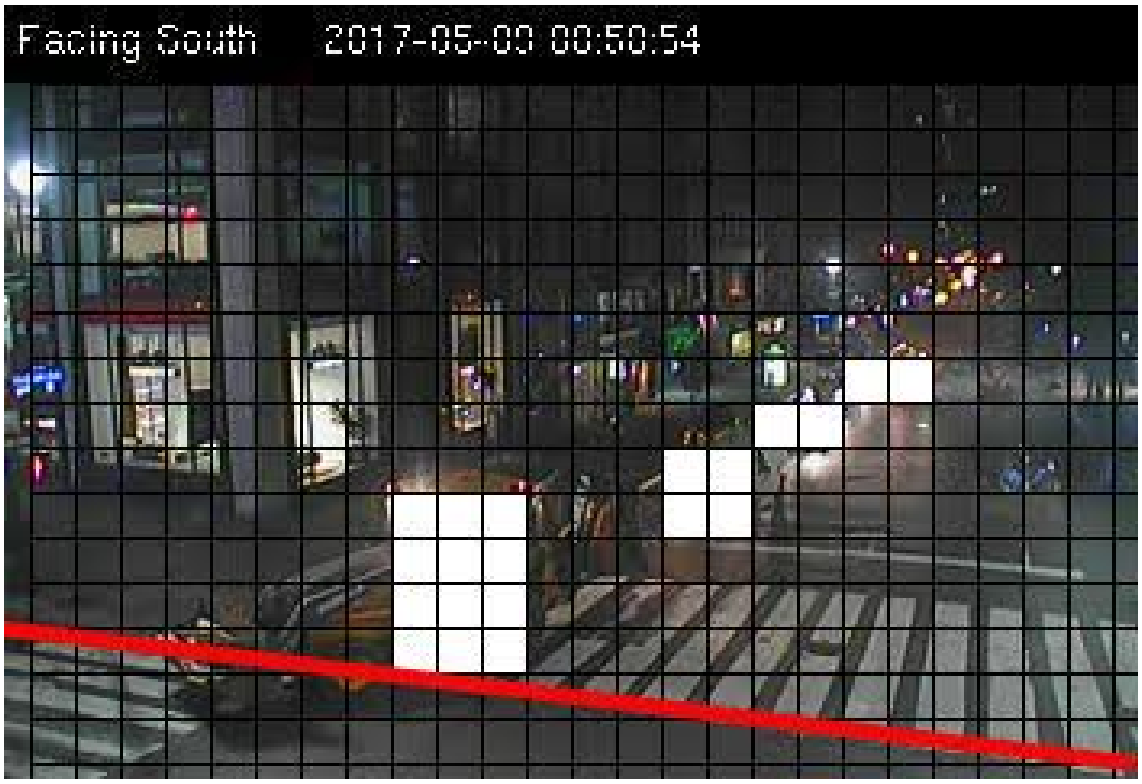

Figure 6: Generation of Simplified Input.

4.2.4 Reward

The no of vehicles that pass through the intersection in a time interval or the throughput of the intersection can be taken as a rudimentary reward function.

To avoid any traffic signals that might incur collisions, any collisions detected are penalised with a negative reward. This still presents the problem of starvation *ie.* a low amount of vehicles could wait for green indefinitely. It can be solved by aging the throughput negatively, which makes the throughput reward smaller and finally negative the longer it waits.

4.2.5 Scalability

The system proposed could be implemented on a larger scale, like a town or city. But in the current form of neural networks, adding a single traffic light to an existing system makes retraining the entire system a requirement. As the number of intersections increase, the number of actions rise exponentially. Thus making their efficient implementation a problem.

The bottleneck is the non scalability of the current architecture, which could be tentatively solved if parallel neural networks are introduced. Par-

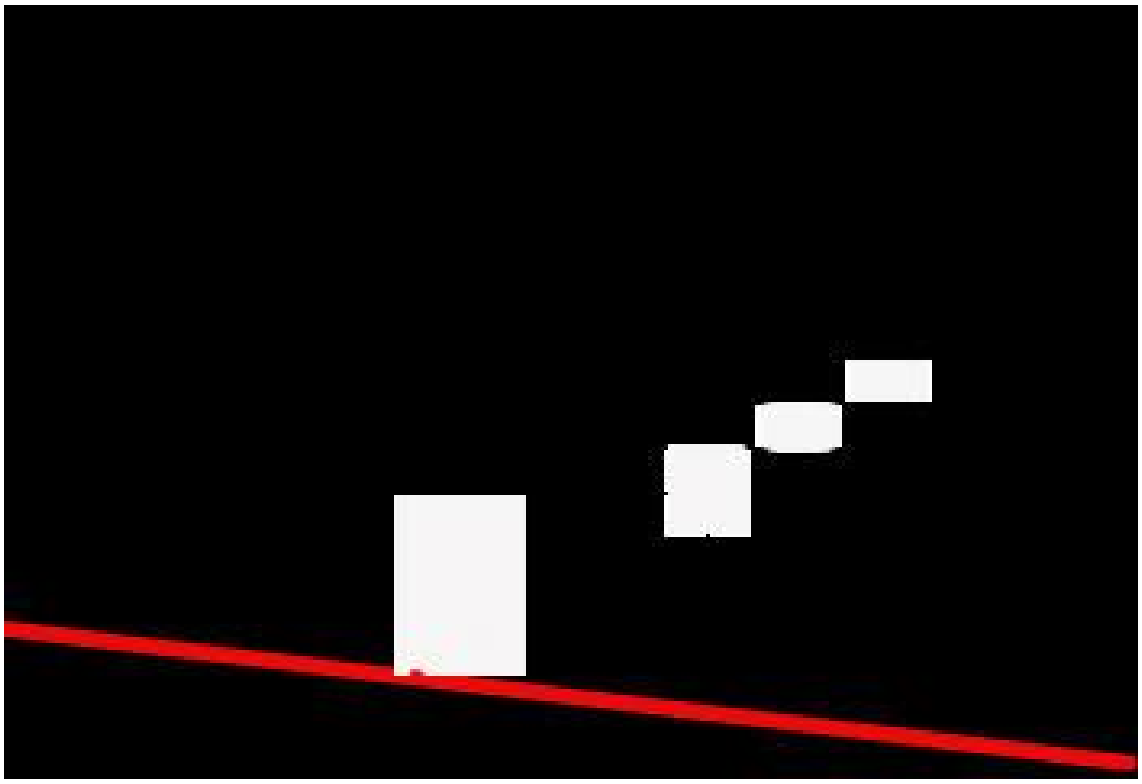

Figure 7: Simplified Input.

allel neural networks are still in its infancy and outside the scope of this project.

5 Conclusion

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.