Playing Atari Games with Deep Reinforcement Learning & using Neuroevolution to train Neural Networks

Prateek Grover, Jainam Shah and Sanil Patel
2018A3PS0338P, 2018A7PS0212P, 2018A7PS0236P

Submitted as part of Assignment - 3, Neural Networks and Fuzzy Logic, BITS Pilani.

Primary Reference

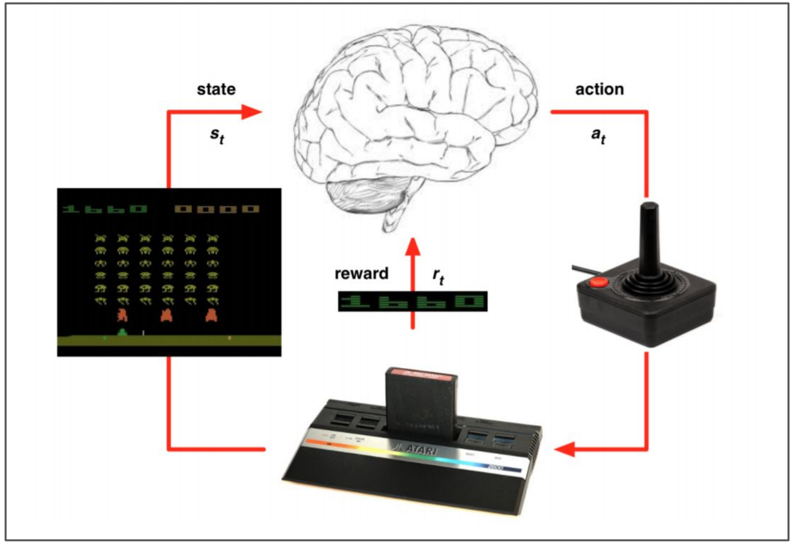
Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller

Playing Atari with Deep Reinforcement Learning presented at *the Conference and Workshop on Neural Information Processing Systems (abbreviated as NeurIPS and formerly NIPS) Deep Learning Workshop 2013.*

Special Thanks to Manan Soni (Teaching Assistant) for his guidance and support.

Aim

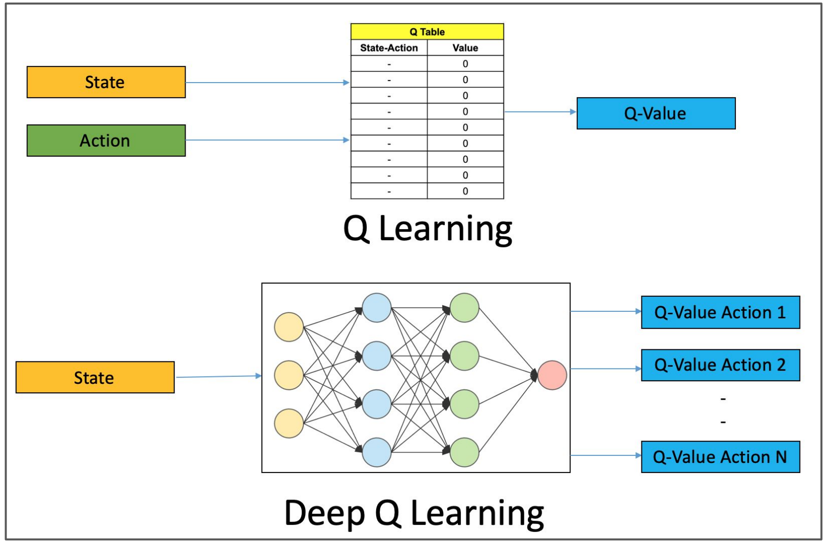
The goal of the research paper was to create an RL Agent that could play several games of the Atari 2600 series with the help of Q Learning along with Neural Networks. This was done by providing images of the gameplay to the agent which in turn provided actions to take to maximize the cumulative reward.



Flowchart depicting sequence of RL Agent

Methodology

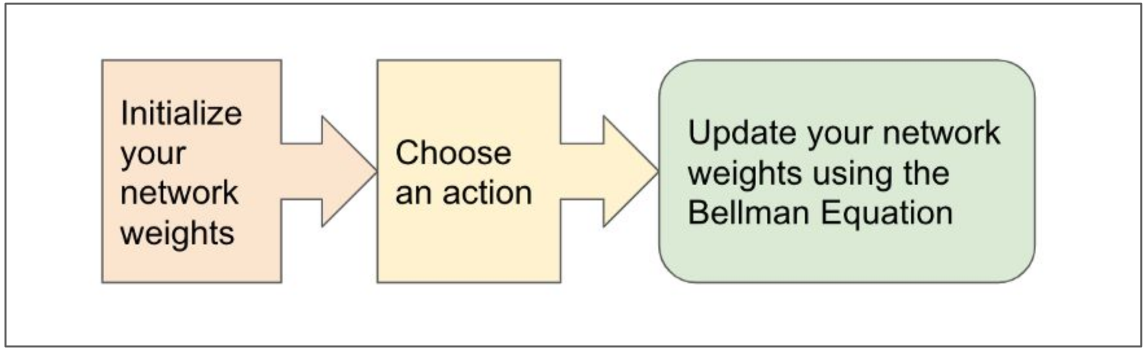
The model used to create such RL Agent is a Convolutional Neural Network, trained with a variant of Q-learning, which accepts input as raw pixels of the Atari Gameplay and whose output is a value function estimating future rewards.



Use of Neural Networks in Q Learning

Methodology (Continued …)

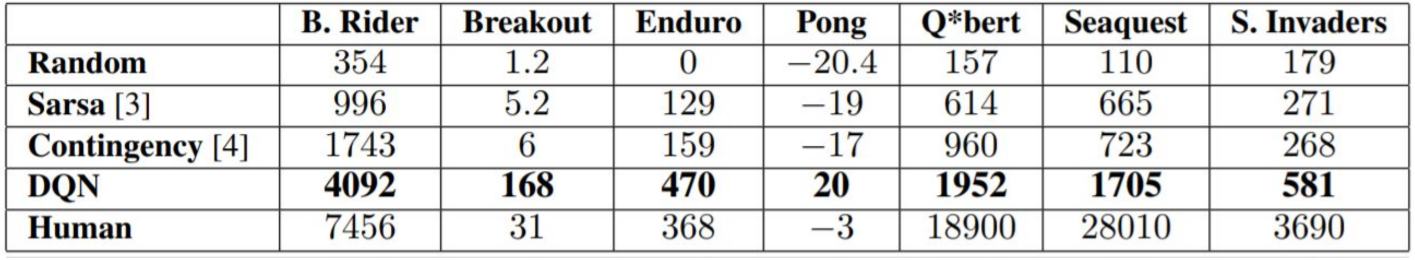
As shown in the figure, the network weights are initialized, after which the Action is chosen using Epsilon- Greedy Strategy, and finally with the rewards obtained, network weights are updated after taking gradients for the loss function. This cycle goes on and on until the game comes to an end.


Initialize your network weights
Choose an action
Update your network weights using the Bellman Equation

Procedure for Updating Weights using Epsilon Greedy Strategy

Final Outcome

The methodology described is applied to numerous Atari 2600 Games simulated via Learning Environment, and it is observed that the results surpass that of human expert in some of these cases.



	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690

Table showing Performance of DQN in comparison to other methods

Theory & Background

Overview of Reinforcement Learning

We consider tasks in which an agent interacts with the Atari emulator, in a sequence of actions, observations and rewards.

- At each time-step the agent selects an action at from the set of legal game actions. $A_t = \{A_1, A_2, ..., A_k\}$
- The action is passed to the emulator and modifies its internal state and the game score.
- The agent observes an image, x_t which is a vector of raw pixel values representing the current screen.
- In addition it receives a reward r_t representing the change in game score.

We therefore consider sequences of actions and observations and learn game strategies that depend upon these sequences. $s_t = \{x_1, a_1, x_2, ..., a_{t-1}, x_t\}$

The goal of the agent is to select actions in a way that maximises future rewards. We make the standard assumption that future rewards are discounted by a factor of γ per time-step, and define the future discounted return at time t as

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

where T is the time-step at which the game terminates.

Overview of Reinforcement Learning (continued ..)

We define the optimal action-value function Q(s, a) as the maximum expected return achievable by following any strategy, after seeing some sequence s and then taking some action a,

$$Q(s,a) = max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$$

where π is a policy mapping sequences to actions (or distributions over actions).

The optimal action-value function obeys an important identity known as the Bellman equation: if the optimal value Q(s' , a') of the sequence s' at the next time-step was known for all possible actions a' , then the optimal strategy is to select the action a maximising the expected value of r + γQ(s', a')

$$Q(s,a) = \mathbb{E}_{s' \backsim \varepsilon}[r + \gamma \, max_{a'} Q^*(s', a') | s, a]$$

The basic idea behind many reinforcement learning algorithms is to estimate the action value function, by using the Bellman equation as an iterative update,

$$Q_{i+1}(s,a) = \mathbb{E}[r + \gamma \, max_{a'} Q_i(s', a') | s, a]$$

Such value iteration algorithms converge to the optimal action value function, Qi → Q∗ as i → ∞ .

Role of Neural Networks in Deep Q Learning

We refer to a neural network function approximator with weights θ as a Q-network. A Q-network can be trained by minimising a sequence of loss functions Li(θi) that changes at each iteration i,

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.)}[(y_i - Q(s,a;\theta_i))^2]$$

where $y_i = \mathbb{E}_{s'\sim\varepsilon}[r + \gamma\, max_{a'}\; Q(s',a';\theta_{i-1})|s,a]$ is the target for iteration i and ρ(s, a) is a probability distribution over sequences s and actions a that we refer to as the behaviour distribution. Differentiating the loss function with respect to the weights we arrive at the following gradient,

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.);s'\sim\varepsilon}[(r + \gamma\, max_{a'}\; Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))\; \nabla_{\theta_i} Q(s,a;\theta_i)]$$

Rather than computing the full expectations in the above gradient, it is better to optimise the loss function by stochastic gradient descent. If the weights are updated after every time-step, and the expectations are replaced by single samples from the behaviour distribution ρ and the emulator E respectively, then we arrive at the familiar Q-learning algorithm.

Dataset and Environments

Learning Environment

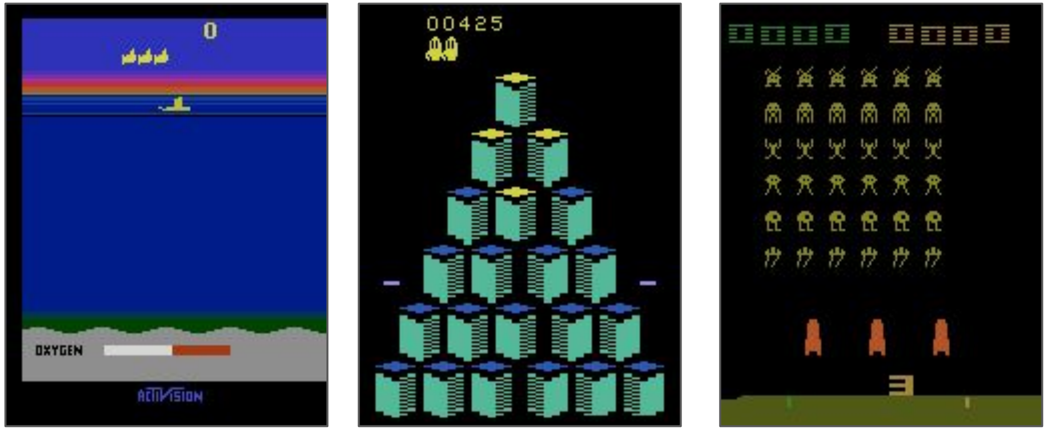
- The environment used in our project for simulating Atari Games was OpenAI’s Gym. It is a toolkit used for comparing and developing reinforcement learning algorithms.

- It is compatible with various numerical computation libraries and can be used to test domain-independent agents making no assumptions about their structure.

- This paper uses the Arcade Learning Environment (ALE) but we have used OpenAI Gym as it more standardised, has better support and resource availability. Infact, OpenAI uses an integrated form of ALE to simulate the Atari games in an easy-to-install form.

Dataset

Dataset in our paper consisted of images generated from Atari Games like Breakout or Space Invaders upon taking steps based on a policy through our model. These images were preprocessed and further processed via CNNs to generate suitable steps.

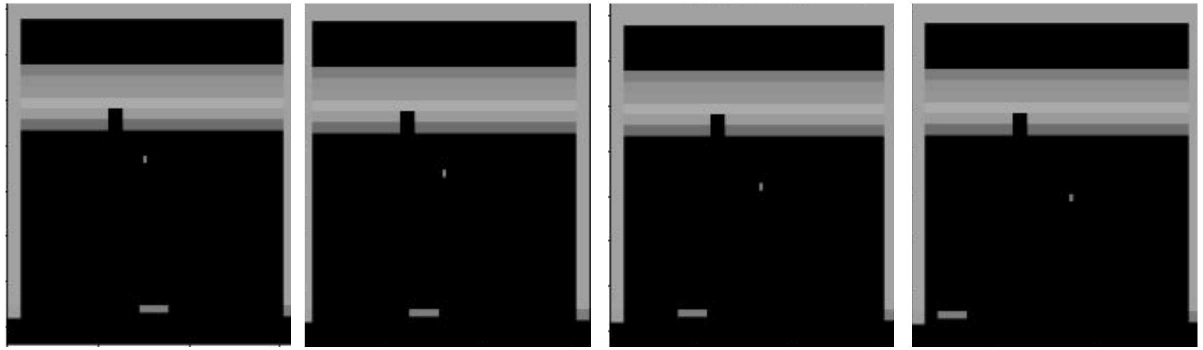


Images generated by Atari Games Environment created by OpenAI’s Gym Framework

Algorithms & Implementation

Algorithm : Preprocessing Observations from Environment

- Crop and Resize Images as a numpy array from (214, 160) to (110, 84) using cv2.resize()
- Convert to Grayscale using cv2.cvtColor()
- Crop to (84, 84) to get only playing area
- Stack N_FRAMES (4 here) number of Images together to obtain Observation of (4, 110, 84)



4 Grayscale Images stacked together to form an observation for the game Breakout

Neural Network Architecture used in Deep Q Network

Input Observation
(4, 84, 84)

16 - 8x8 Filters with Stride 4 and ReLU Activation

Dimensions
(16, 20, 20)

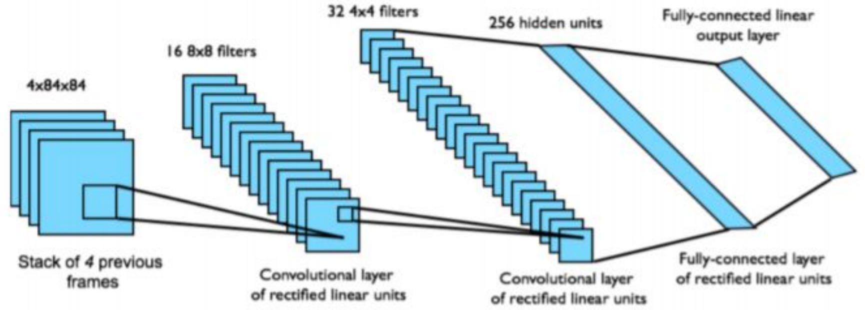
32 - 4x4 Filters with Stride 2 and ReLU Activation

Dimensions
(32, 9, 9)

Fully Connected 256 Neurons

Dimensions
(1, 256)

Fully Connected Number of Actions


4x84x84
16 8x8 filters
32 4x4 filters
256 hidden units
Fully-connected linear output layer
Stack of 4 previous frames
Convolutional layer of rectified linear units
Convolutional layer of rectified linear units
Fully-connected layer of rectified linear units

Mnih et.al., Nature, 2014

Algorithm : Deep Q- Learning with Experience Replay

```
Initialize Replay Memory D to Capacity N
Initialize Action-Value Function Q with Random Weights
Initialize States Array with Specific States of Game

Initialize Empty Rewards Array
Initialize Empty Maximum Action Value Array

for episode = 1 … M do

          While (episode does not finish) do:
                    Using Epsilon Greedy Strategy:
                              Select Random Action $a_t$ or
                              Select Action that maximizes Q Value from state $a_t = max_a Q * (\phi(s_t), a, \theta)$
                    Execute $a_t$ and observe reward $r_t$ and state reached
                    Store observation (Current State, Action, Reward, State Reached) in Memory D
                    Sample random batch of observation from D
                    Obtain estimated Q Value $y_t = r_t + \gamma max_{a'} Q(\phi(s_{t+1}), a'; \theta)))$

                    Perform Gradient Descent on the loss :$(y_t - Q(\phi_t, a_t; \theta))^2$
          End While

          Store Reward in Rewards Array
```

Algorithm : Deep Q- Learning with Experience Replay (Continued ..)

```
        Sample Specific State from States Array

        While (episode does not finish) do:
            Using Epsilon Greedy Strategy:
                Select Random Action $a_t$ or $a_t = max_a Q * (\phi(s_t), a, \theta)$
                Select Action that maximizes Q Value from state
            Execute $a_t$ and observe reward $r_t$ and state reached)
            Store Normalized Estimated Maximum Action Value in Array
        End While

    End For
```

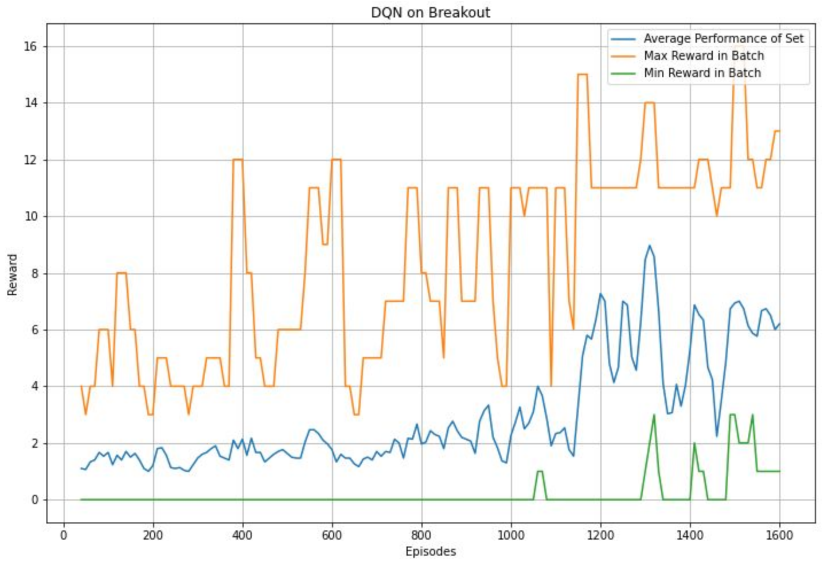
Algorithm : Neuroevolution Techniques to tune Neural Network Hyperparameters

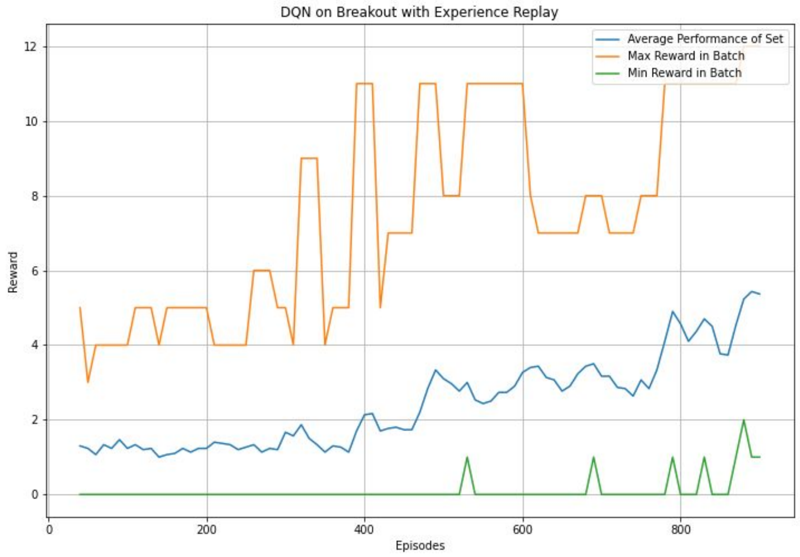
```
**Initialize** Population Size P
**Initialize** Number of Generations G
**Initialize** Stagnation Limit S

**Randomize** Generation of First Population of Genomes

**for generations = 1 … G do**

        Create New Population from Created Children and Elite Genomes
        Find Reward for each Genome in Population
        Add fittest proportionately to mating pool, removing the weakest and Stagnant ones
        Create New Genomes from Crossover between parents
        Apply Mutation to genes of each of the new Genomes

**end for**
```

The genes here contain information about number of neurons in each layer, activation function used, weights etc.

Results and Discussion

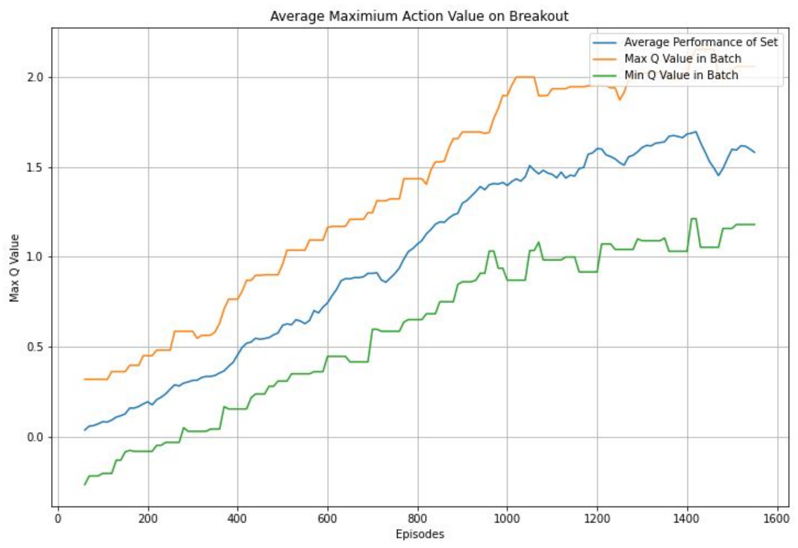
Performance of DQN on Breakout in terms of Rewards

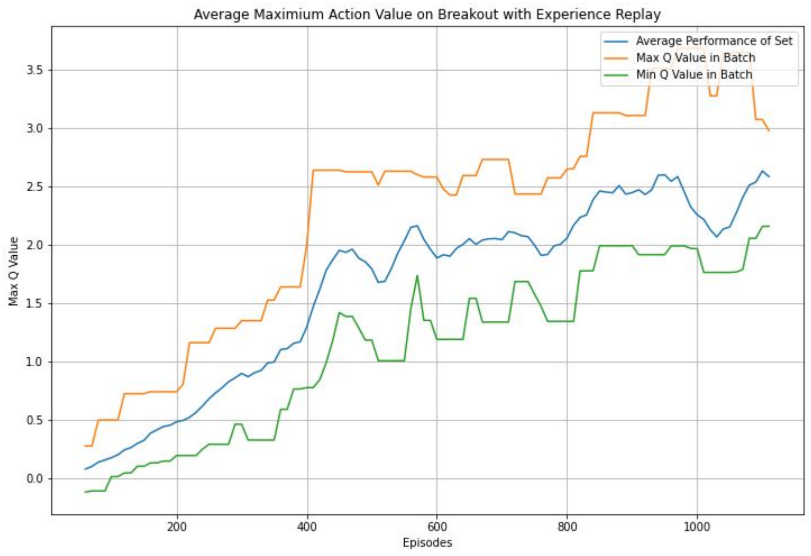




As it can be observed, with Experience Replay, the network trains almost twice as fast in terms of episodes, and an average of Reward > 6 is reached steadily.

Performance of DQN on Breakout in terms of Maximum Action Values





As it can be observed, with Experience Replay, the network gets better at finding maximum reward from a sample state almost twice as fast in terms of episodes, and an average of Maximum Q Values greater than 2.25 is reached steadily.

Performance of DQN on other Atari Games

12 other plots of Rewards and Q Values with and without Experience Replay is provided in the supplementary material. This was done by training the DQN on Beam Rider, QBert, Seaquest and Breakout four other games mentioned in the paper.

	Beam Rider	Seaquest	Breakout	Space Invaders
DQN	521	292	6.07	253
DQN with Experience Replay	478	237	5.85	235

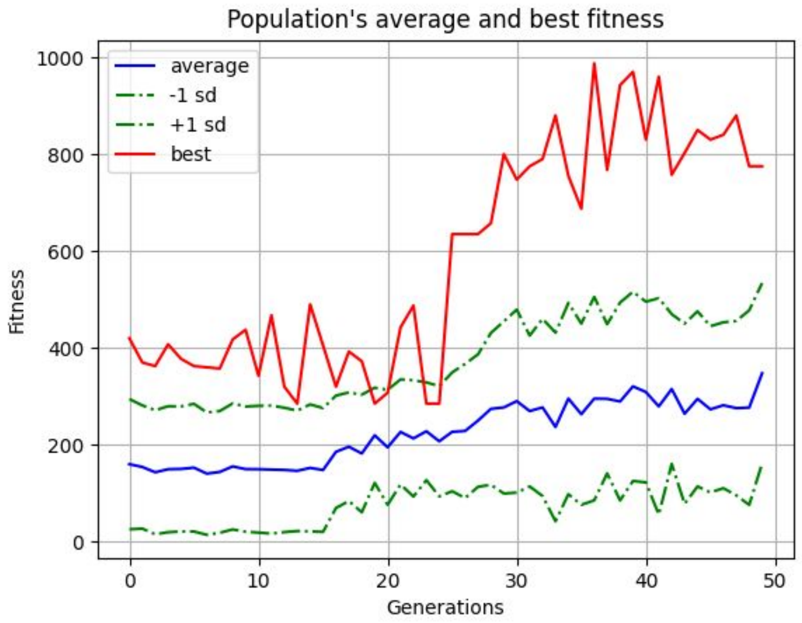
Table 1: Rewards obtained from the model for each of the games, note that Experience Replay model was trained for roughly half of the time the Vanilla model was trained

	Seaquest	QBert	Breakout	Space Invaders
DQN	1.45	2.13	1.65	1.71
DQN with Experience Replay	2.32	2.27	2.6	2.08

Table 2: Maximum Action Values obtained from the model for each of the games, note that Experience Replay model was trained for roughly half of the time the Vanilla model was trained

NEAT Optimization for Tuning Hyperparameters of Neural Network

Upon advice from the TAs, we were also encouraged to train the DQN using Genetic Algorithm, in specific the NEAT Optimization Techniques.



Surprisingly, we were able to obtain stable Average Rewards of > 300 on Space Invaders, using a comparatively Simpler Neural Network in shorter duration.

The supplementary material includes the neural network architecture that was obtained using this method.

Comparing Rewards obtained by the Original Author and our Implementation

	Breakout	Beam Rider	Seaquest	Space Invaders
Random	1.2	354	110	179
DQN (by Paper)	168	4096	1952	581
DQN (by our Results)	6.07	521	292	253

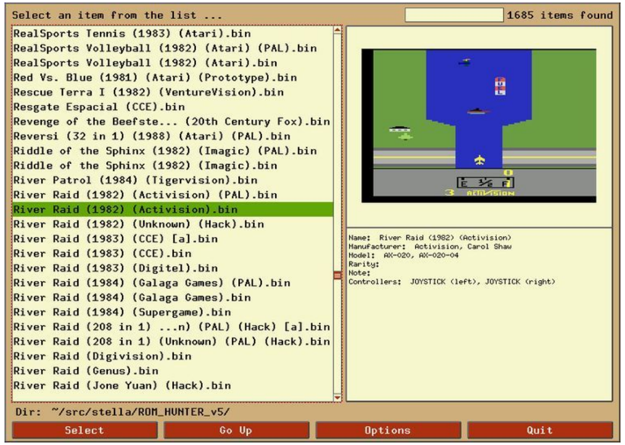
Challenges, Future Scope & Learning Outcomes

Challenges Faced in Implementation

- Choosing a suitable environment that could help simulate Atari Games

The research paper released in 2014, worked in **Arcade Learning Environment** using *ALE-Python Library* or *ale-py*, resources and support for which has slowly declined due to release of new libraries like **OpenAI's Gym** and **Google's Dopamine**, which helped simplify creation of Atari environment greatly.

After discussion with TAs, we finalized Gym Library as the preferred mode of environment.



```
Select an item from the list ...                1685 items found
RealSports Tennis (1983) (Atari).bin
RealSports Volleyball (1982) (Atari) (PAL).bin
RealSports Volleyball (1982) (Atari).bin
Red Vs. Blue (1981) (Atari) (Prototype).bin
Rescue Terra I (1982) (VentureVision).bin
Resgate Espacial (CCE).bin
Revenge of the Beefste... (20th Century Fox).bin
Reversi (32 in 1) (1988) (Atari) (PAL).bin
Riddle of the Sphinx (1982) (Imagic) (PAL).bin
Riddle of the Sphinx (1982) (Imagic).bin
River Patrol (1984) (Tigervision).bin
River Raid (1982) (Activision) (PAL).bin
River Raid (1982) (Activision).bin
River Raid (1982) (Unknown) (Hack).bin
River Raid (1983) (CCE) [a].bin
River Raid (1983) (CCE).bin
River Raid (1983) (Digitel).bin
River Raid (1984) (Galaga Games) (PAL).bin
River Raid (1984) (Galaga Games).bin
River Raid (1984) (Supergame).bin
River Raid (208 in 1) ...n) (PAL) (Hack) [a].bin
River Raid (208 in 1) (Unknown) (PAL) (Hack).bin
River Raid (Digivision).bin
River Raid (Genus).bin
River Raid (Jone Yuan) (Hack).bin

Name:  River Raid (1982) (Activision)
Manufacturer:  Activision, Carol Shaw
Model:  AX-020, AX-020-04
Rarity:
Note:
Controllers:  JOYSTICK (left), JOYSTICK (right)

Dir:  ~/src/stella/ROM_HUNTER_v5/
```

Stella Emulator used with ALE

Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.

Gym Framework by OpenAI

Dopamine by Google AI

Challenges Faced in Implementation

- Adapting specifications to current Hardware availability

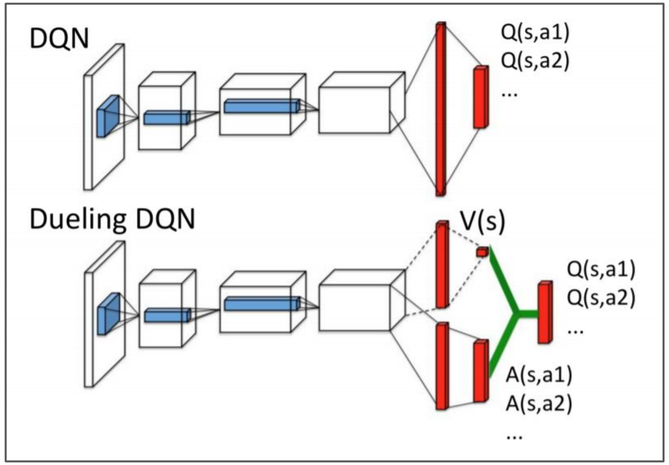
With extensive use of Experience Replay algorithm, large RAM was required to store experiences taken by agent, and sampling through them. The training procedure was also very computationally expensive, which required use of GPUs. Reducing the ER Capacity, Reducing the Number of Annealing Steps, and Parallel Computations using multiple servers was done to adapt to the current Hardware availabilities.

```
GPU: 1xTesla K80 , Compute 3.7, having 2496 CUDA cores , 12GB GDDR5 VRAM
CPU: 1xSingle Core Hyper Threaded Xeon Processors @2.3Ghz i.e (1 Core, 2 Threads)
RAM: ~12.6 GB Available
Disk: ~33 GB Available
```

Hardware Specifications for Google Colab Instance

Future Scope

After the release of paper in 2014, many Immediate Improvements were suggested in following years, including Double DQN and Duelling DQN



The motivation behind Dueling Architecture was it is unnecessary to know Q Value for each action at every step, it resolves this by separating state values and state-dependant action values.

By separating these two categories, the dueling architecture can learn which states are valuable, without having to learn the effect of each action for each state.

Prioritized Experience Replay was also introduced, in which probability of picking a sample from Replay memory is also dependent on the maximum benefit it can provide.

Learning Outcome

- Understanding Resource requirements and Skill requirements
    1. Understanding core concepts of Reinforcement Learning like State, Action, Discounted Reward and Maximum Action Values through Online Lectures.
    2. Reading and Understanding Literature related to Q-Learning and its application to real life problems.
    3. Helped in understanding the use of Neural Networks as an estimator for the Q Function, an application which can be extended to numerous scenarios.

- Had the opportunity to work in Torch and Gym Framework in the development and implementation thereby upskilling our knowledge in Python Libraries.

- Addressing problems, issues and challenges through communication with Professors, Teaching Assistants and teammates in the given time frame.

Thanks!