Playing Atari Games with Deep Reinforcement Learning

Varsha Lalwani (varshajn@iitk.ac.in)
Masare Akshay Sunil (amasare@iitk.ac.in)

IIT Kanpur

CS365A

Artificial Intelligence Programming

Course Project

Instructor: Prof. Amitabha Mukherjee

Acknowledgements

We would like to thank Prof. Amitabha Mukerjee for giving us this opportunity to work on this project. His valuable guidance and various insights throughout have helped us a lot in completing the project. We would also like to thank Mr. Ashudeep for mentoring us and helping us move forward with the project whenever we got stuck. We also would like to thank Prof. Vinay P. Namboodiri for providing us with the GPU required for the project.

Abstract

In this project, we attempt to learn control policies of Atari games using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, taking raw pixels as inputs and giving value function estimating future rewards as output. We applied this method to play 3 Atari games from the Arcade Learning Environment[1], with no adjustment of the architecture or learning algorithm. Moreover, we also tried to play two similar games (space invaders and phoenix) using one single agent trained to play one of those games (phoenix).

Motivation

General Game Playing is the branch of Artificial Intelligence that deals with playing multiple games using a single agent. For many years, it has been possible for a computer to play a single game by using some specially designed algorithm for that particular game. But these algorithms were useless outside their context. For example, an algorithm for chess cannot play checkers. Hence, we need General Game Playing agents to play multiple games. In this project we are trying to implement a deep reinforced learning based agent to play multiple video games.

Previous Work

There have been many attempts in past few years to design general game players using several techniques. The first successful Deep Reinforcement Learning based General Game Player [2] was implemented by Mnih et. al. of DeepMind Technologies which was motivated by the success of model free reinforcement learning approach in a backgammon playing program. Since then, there have been various similar attempts to implement the algorithm. Ours is one such attempt to replicate their work on 3 different games.

We have also experimented by trying to play two similar games with an agent trained on one of these games and we achieved success according to our hypothesis that the agent should be able to play fairly well as compared to the untrained agent.

Playing Atari Games with Deep Reinforcement Learning

Our methodology is similar to the paper by Mnih et. al. So, we are using a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. This approach can be divided into three major parts:

1. Convulational Neural Networks
2. Q-Learning
3. Emulation Interface

We can broadly describe our working algorithm as follows:

- *Initialize the game Emulation Environment Interface*
- *Take the screenshots of the game*
- *Pre-process the screenshots*
- *Use CNNs to extract the features from the screenshots*
- *Choose any action from the list of possible actions according to current state*
- *Observe reward and save it to memory*
- *Repeat and Train*

Convolutional Neural Networks

The figure below explains our CNN very well. We have used CNNs for feature extraction from the screenshot of the game state. We take 4 consecutive images at a time and they form the nodes of the Input layer of our CNN. The images are takes as 2D matrices and are then convolved with linear filters. Multiple images are accounted for by weight matrices. Our Neural Network finally assigns the expected reward value to each possible action. The images come is as 210x160 pixels. We crop the top 50 pixels as they are just HUD to get a 160x160 image which is then downscaled to 84x84 pixels. Our first layer of filters are 8x8 in size and are multiplied with an step size of 4 pixels. Hence, a node in the resulting layer is 20x20 pixels in size. The next filter set is 4x4 in size with a step of 2 pixels resulting in a node of 9x9 pixels. Finally, we have a fully connected neural network that outputs all possible actions of the given state.

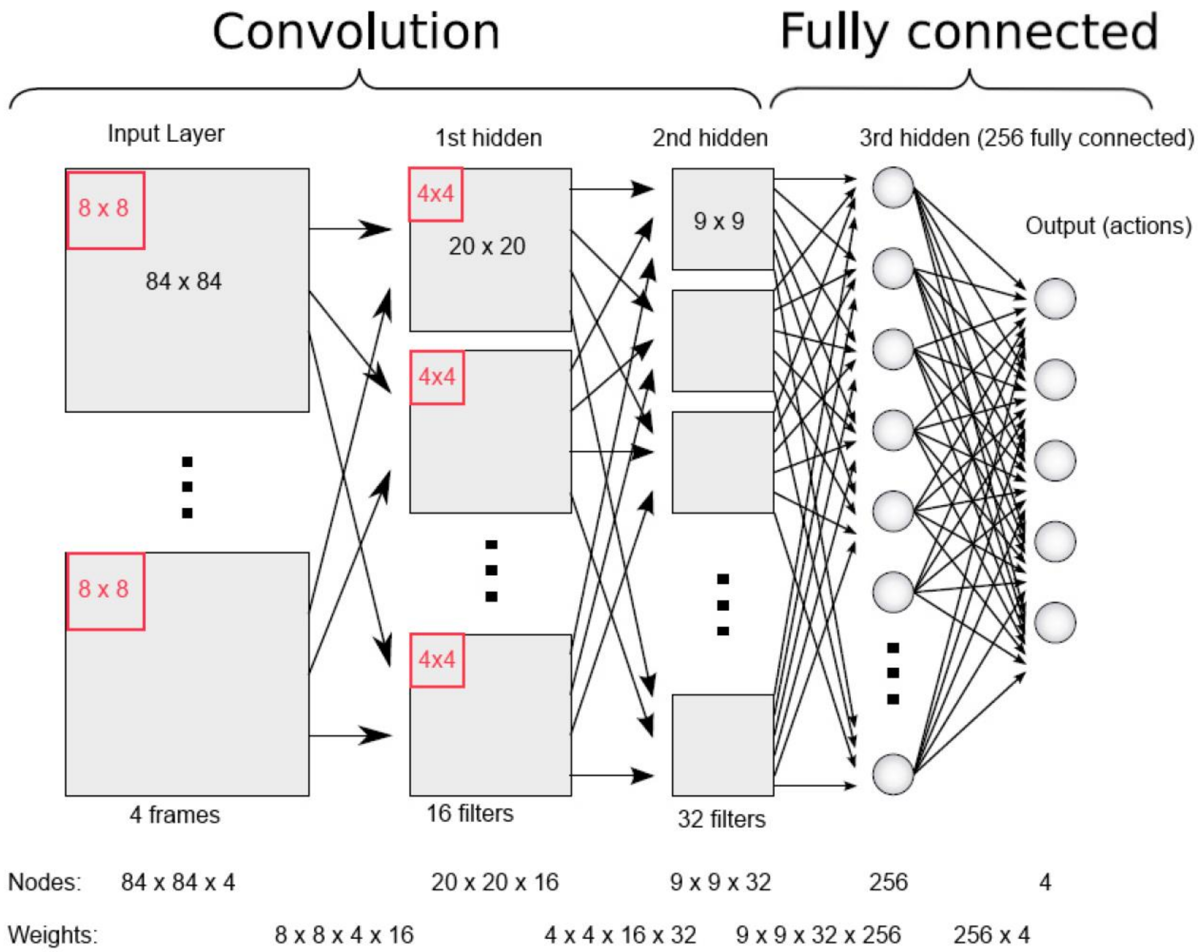

Figure 1. Neural Network Structure[3]

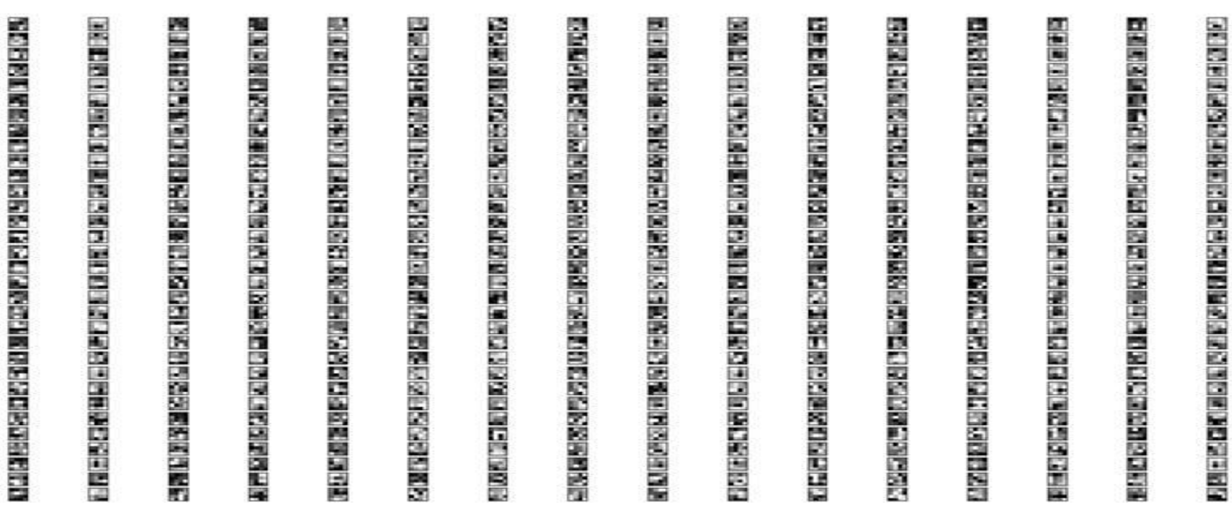

Figure 2. Second Filter set for the game 'Breakout'

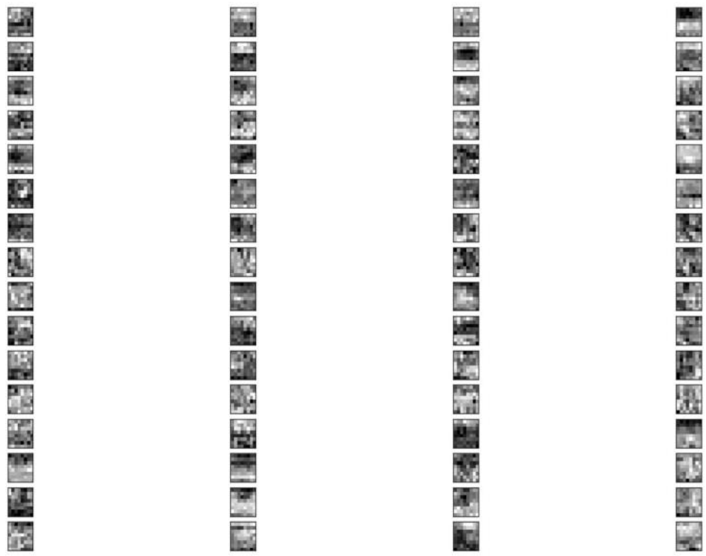

Figure 3. First Filter set for the game 'Breakout'

Q-Learning

In a reinforcement learning model, an agent takes actions in an environment with the goal of maximizing a cumulative reward. The basic reinforcement learning model consists of: a set of environment states S; a set of actions A; rules of transitioning between states; rules that determine the scalar immediate reward of a transition; and rules that describe what the agent observes.

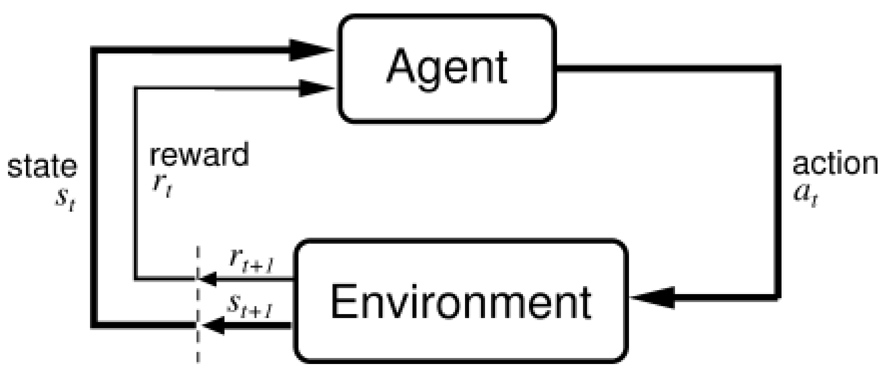

Figure 4. Reinforcement Learning[4]

Q-Learning is a model-free form of Reinforcement Learning. If S is a set of states, A is a set of actions, γ is the discount factor, α is the step size. Then we can understand Q-Learning by this Algorithm [5]:

$$Initialize\ \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})\ arbitrarily$$

$Repeat\ (for\ each\ episode)$:

$Initialize\ \boldsymbol{S}$

$Repeat\ (for\ each\ step\ of\ episode)$:

$Choose\ \boldsymbol{a}\ from\ \boldsymbol{s}\ using\ policy\ derived\ from\ \boldsymbol{Q}$

$(e.g.\ \ \in -greedy)$

$Take\ action\ \boldsymbol{a},\ observe\ \boldsymbol{r},\ \boldsymbol{s}'$

$\boldsymbol{Q}'(\boldsymbol{s}',\boldsymbol{a}') <--\ \boldsymbol{Q}(\boldsymbol{s},\ \boldsymbol{a}) +\ \boldsymbol{\alpha}[\boldsymbol{r}\ +\ \boldsymbol{\gamma}.\boldsymbol{max}\ \boldsymbol{Q}(\boldsymbol{s}',\boldsymbol{a}') - \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})]$

$\boldsymbol{s} <--\boldsymbol{s}'$

$until\ \boldsymbol{s}\ is\ terminal$

Arcade Learning Environment

As our emulation interface we are using Arcade Learning Environment (ALE). It is built on top of Stella, an open-source Atari 2600 emulator. It is built in C++ and supports nearly 50 games. It can also output the end of the game signal for the supported games. It also supports FIFO queues for input to the games and taking output from it. This results in a smooth learning experience of our agent.

Implementation

For the Implementation of the project we needed a powerful GPU and a lot of memory. So, the system we used hosted a Nvidia 760GTX CUDA compatible GPU and 8 gigabytes of memory. Even with such a powerful system, we faced a lot of problem with direct implementation of cuda-convnet2 library as given in the paper of replicating deep mind[6]. So, instead we chose an indirect implementation of this library with Theano library of Python[7]. The Implementation of Arcade Learning Environment was pretty easy as it is open source[8]. Other libraries we used were: SDL for display, RL-GLUE for communication between CNN and ALE, numpy and pylearn2 for training.

Breakout

The very first game we trained was Breakout. We chose breakout due to its simple nature. It has only two states: dead or alive, has only two actions: right or left, and it is very simple to create the reward function: positive value for alive states and a large negative one for dead states. Initially, we tested a random agent on the game, the results of which are in the figure below.

```
Episode 1 ended, score: 2
Episode 2 ended, score: 0
Episode 3 ended, score: 1
Episode 4 ended, score: 2
Episode 5 ended, score: 1
Episode 6 ended, score: 2
Episode 7 ended, score: 0
Episode 8 ended, score: 0
Episode 9 ended, score: 2
Episode 10 ended, score: 0
Episode 11 ended, score: 3
Episode 12 ended, score: 1
Episode 13 ended, score: 2
Episode 14 ended, score: 0
Episode 15 ended, score: 0
Episode 16 ended, score: 1
Episode 17 ended, score: 1
Episode 18 ended, score: 1
Episode 19 ended, score: 3
Episode 20 ended, score: 1
Episode 21 ended, score: 2
Episode 22 ended, score: 0
Episode 23 ended, score: 2
Episode 24 ended, score: 2
Episode 25 ended, score: 0
Episode 26 ended, score: 0
Episode 27 ended, score: 1
Episode 28 ended, score: 2
Episode 29 ended, score: 3
Episode 30 ended, score: 2
Episode 31 ended, score: 0
Episode 32 ended, score: 6
Episode 33 ended, score: 2
Episode 34 ended, score: 1
Episode 35 ended, score: 2
Episode 36 ended, score: 1
Episode 37 ended, score: 2
Episode 38 ended, score: 2
Episode 39 ended, score: 3
Episode 40 ended, score: 0
```

Figure 5. Scores on the game ‘Breakout’ by random agent.

We then trained the agent to 26 epochs, and we achieved an average score of 48 on the 26th epoch. The graph below shows the average score and average loss at each epoch.

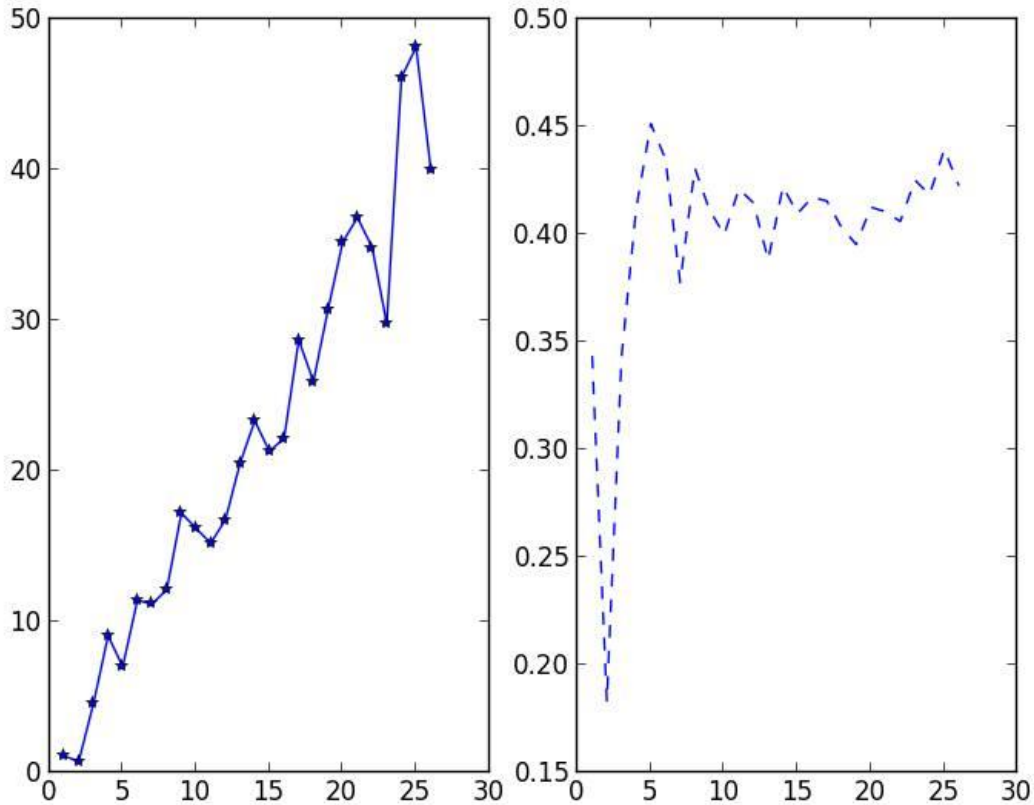

Figure 5. Average Score and Average Loss at each Epoch for the game 'Breakout'

Space Invaders

Space Invaders is another game on Atari 2600. Like Breakout, it was also used in training in the paper by Mnih et. al. But unlike Breakout, Space Invaders is a lot more complex. It has 3 different actions: Left, Right and Shoot. There are enemies to shoot and who in return shoot at us. But, the complexity mostly increases due to increase in action set. So, the training graph for this game is not exactly monotonously increasing. During training, it randomly trains with a specific restricted action set for different epochs. So, if the left or right moment is restricted in any epoch, its performance is badly affected. Still, after 39 Epochs of training the average score bumped up from nearly 160 in random agent to 428.

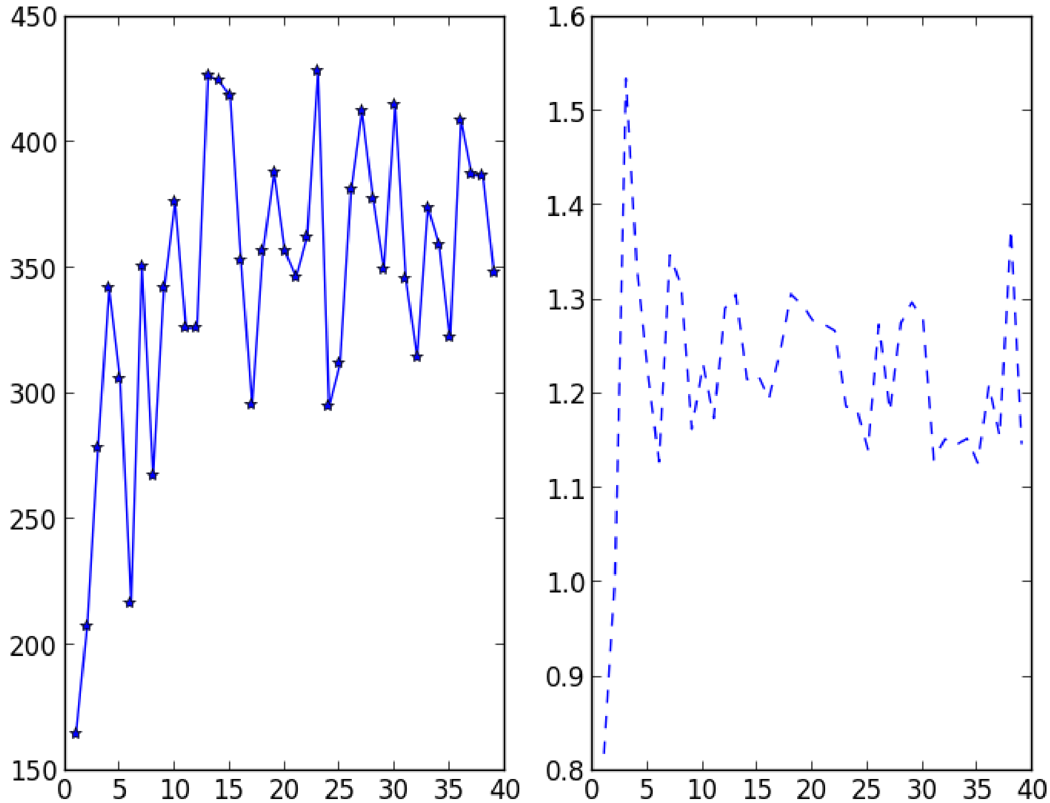

Figure 6. Average Score and Average Loss at each Epoch for the game 'Space Invaders'

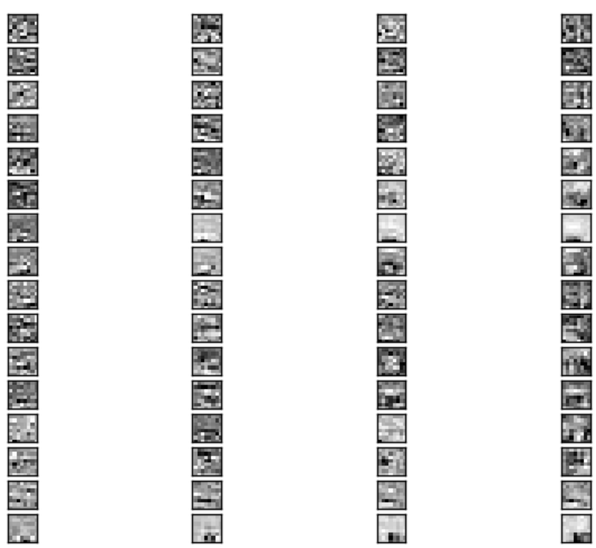

Figure 7. First Filter set in the CNN of the game 'Space Invaders'

Game Videos at: http://home.iitk.ac.in/~amasare/cs365/project/dqrl.html

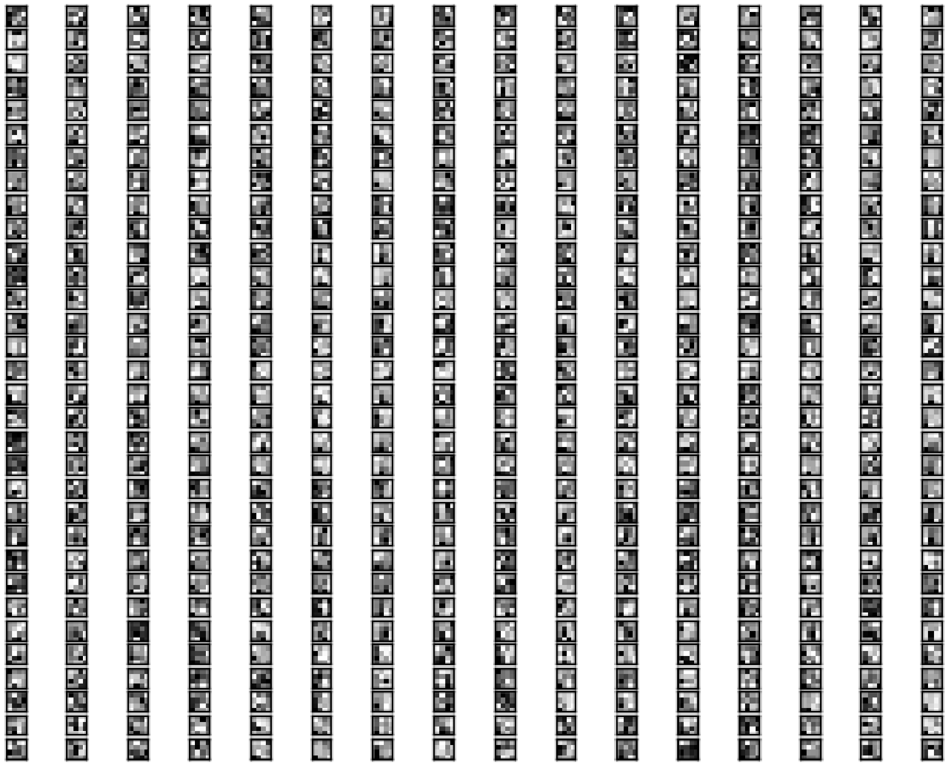

Figure 8. Second Filter set in CNN for the game 'Space Invaders'

Phoenix

Phoenix is a game on Atari 2600, which is similar to the game of Space Invaders. It has the same 3 actions: Left, Right and Shoot, with a similar gameplay. It has a new gameplay element though - a shield which temporarily protects against enemy attack. Due to more than two action, it suffers the same problem as Space Invaders during Training. We chose this game because it was not implemented in the paper by Mnih et. al. Initially, the random agent gave a score of nearly 370 and after 27 epochs we got to a high of 2180 average score. On some runs, we even managed to hit near the 4000 mark.

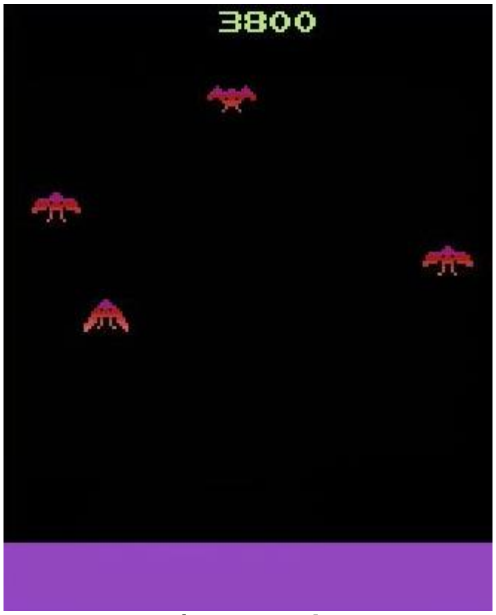

Figure 9. Score of 3800 on Phoenix.

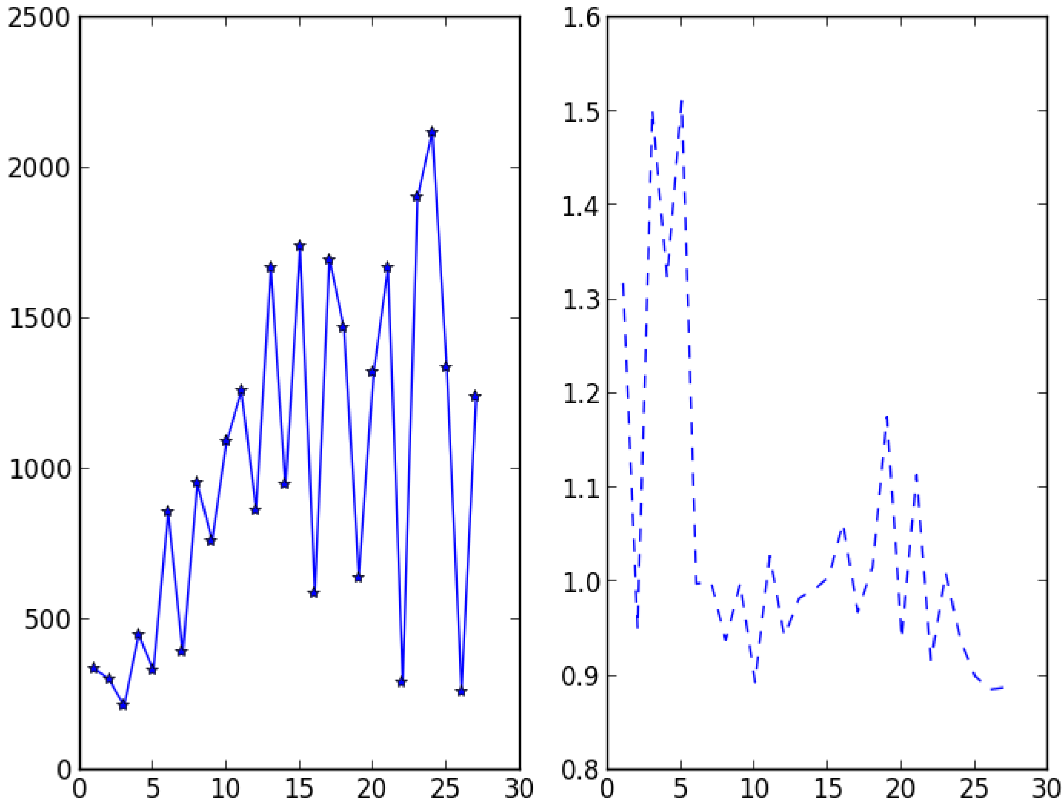

Figure 10. Average Score and Average Loss at each Epoch for the game 'Phoenix'

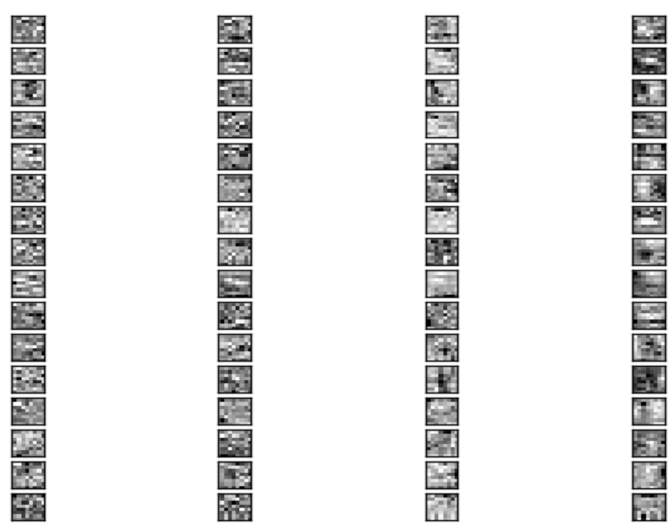

Figure 11. First Filter set in the CNN of the game 'Phoenix'

Game Videos at: http://home.iitk.ac.in/~amasare/cs365/project/dqrl.html

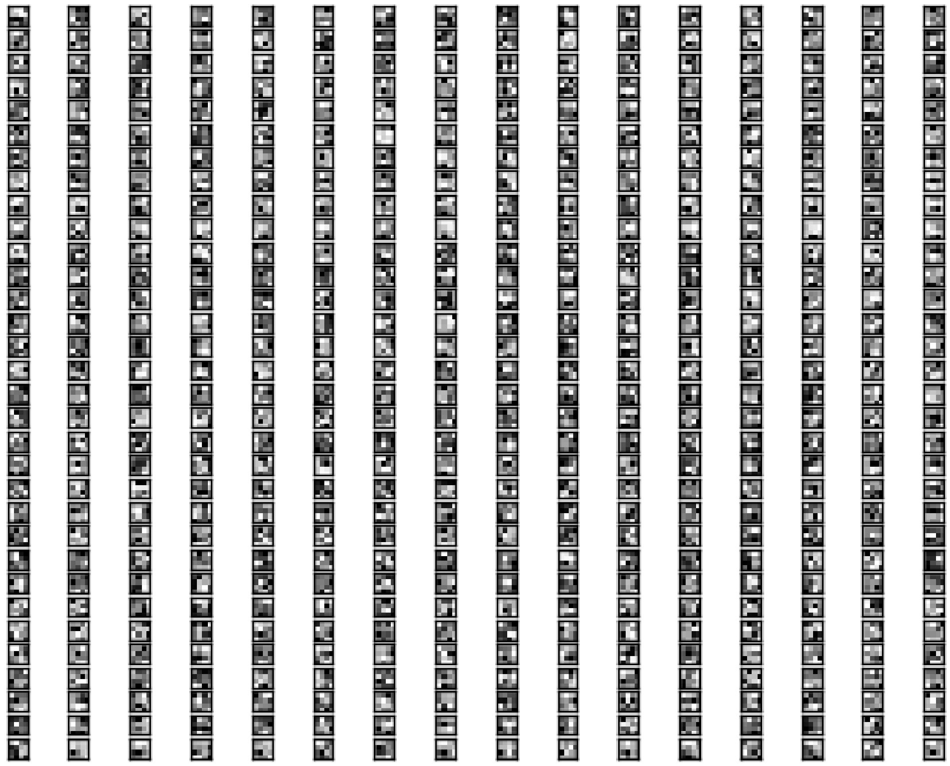

Figure 12. First Filter set in the CNN of the game 'Phoenix'

Inter-Play

What we mean by inter-play is training the agent on one game and testing it out on another game. The results will be particularly good in case the two games are very similar. So, we tried is out on Space Invaders and Phoenix. We trained on Space Invaders and tested on Phoenix. The results we obtained weren't the best, but were significantly better than the random agent. The scores of each episode of Phoenix played on an agent trained on Space Invaders to 39 Epochs is given below in the figure, each of which is significantly higher than the 370 average of the random agent. To train the 27 Epochs of Phoenix takes nearly 15-16 hours on our machine.

Given the high time complexity of the training process, using an already available data of similar game can be very helpful.

```
Episode 1 ended, score: 440
Episode 2 ended, score: 920
Episode 3 ended, score: 320
Episode 4 ended, score: 2900
Episode 5 ended, score: 2140
Episode 6 ended, score: 2300
Episode 7 ended, score: 620
Episode 8 ended, score: 600
Episode 9 ended, score: 1760
Episode 10 ended, score: 620
Episode 11 ended, score: 420
Episode 12 ended, score: 1030
Episode 13 ended, score: 2460
Episode 14 ended, score: 620
Episode 15 ended, score: 1440
Episode 16 ended, score: 560
Episode 17 ended, score: 560
Episode 18 ended, score: 2310
Episode 19 ended, score: 480
Episode 20 ended, score: 1400
Episode 21 ended, score: 910
Episode 22 ended, score: 640
Episode 23 ended, score: 540
Episode 24 ended, score: 440
Episode 25 ended, score: 1030
Episode 26 ended, score: 600
Episode 27 ended, score: 540
Episode 28 ended, score: 1370
Episode 29 ended, score: 2070
Episode 30 ended, score: 3050
Episode 31 ended, score: 500
Episode 32 ended, score: 1820
Episode 33 ended, score: 620
Episode 34 ended, score: 600
Episode 35 ended, score: 540
Episode 36 ended, score: 1950
Episode 37 ended, score: 2040
Episode 38 ended, score: 3710
Episode 39 ended, score: 1840
Episode 40 ended, score: 2740
Episode 41 ended, score: 540
Episode 42 ended, score: 850
Episode 43 ended, score: 2170
Episode 44 ended, score: 640
Episode 45 ended, score: 580
Episode 46 ended, score: 1770
Episode 47 ended, score: 1260
Episode 48 ended, score: 2140
Episode 49 ended, score: 400
Episode 50 ended, score: 1060
```

Figure 13. Scores of Phoenix on agent trained on Space Invaders

Conclusion

The AI agent designed can learn to play various Atari games without any tweaks in the architecture or algorithm. It becomes better as we train it more: increasing average scores per epoch. Also, as we observed, the player trained on one game is performing significantly better than the random player on other similar games.

Future Extension

One general game player trained on multiple games simultaneously and tested on various different games individually to see if one can avoid training the agent for every game. It probably won't be as good as the individually trained agents, but it sure would save many resources in training the agents.

References

1. M.G. Bellemare, Y. Naddaf, J Veness and M. Bowling. (2013).The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research 47*, Pages 253-279.

2. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller. (2013). Playing Atari With Deep Reinforcement Learning. *NIPS Deep Learning Workshop, 2013.*

3. Kristjan Korjus, Ilya Kuzovkin, Ardi Tampuu, Taivo Pungas. (2013). Replicating the Paper “Playing Atari with Deep Reinforcement Learning”. *Technical Report, Introduction to Computational Neuroscience, University of Tartu.*

4. http://webdocs.cs.ualberta.ca/~sutton/book/ebook/figtmp7.png

5. http://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html

6. https://github.com/kristjankorjus/Replicating-DeepMind

7. https://github.com/spragunr/deep_q_rl

8. http://www.arcadelearningenvironment.org/downloads/