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.

Game Videos at: http://home.iitk.ac.in/~amasare/cs365/project/dqrl.html
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\cite{arcade}, 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\cite{dgp} 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.

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

Game Videos at: http://home.iitk.ac.in/~amasare/cs365/project/dqrl.html
Playing Atari Games with Deep Reinforcement Learning

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

Figure 1. Neural Network Structure

Figure 2. Second 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.
Q-Learning is a model-free form of Reinforcement Learning. If $S$ is a set of states, $A$ is a set of actions, $\gamma$ is the discount factor, $\alpha$ is the step size. Then we can understand Q-Learning by this Algorithm:\n
Initialize $Q(s, a)$ arbitrarily

Repeat (for each episode):

Initialize $S$

Repeat (for each step of episode):

Choose $a$ from $s$ using policy derived from $Q$

\hspace{1cm} (e.g. $\epsilon$-greedy)

Take action $a$, observe $r, s'$

\[ Q'(s', a') < Q(s, a) + \alpha [r + \gamma \max Q(s', a') - Q(s, a)] \]

$s < s'$

until $s$ 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.

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

![Figure 5. Scores on the game 'Breakout' by random agent.](http://home.iitk.ac.in/~amasare/cs365/project/dqrl.html)
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.

![Graph showing average score and loss](image)

**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.

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

Game Videos at: [http://home.iitk.ac.in/~amasare/cs365/project/dqrl.html](http://home.iitk.ac.in/~amasare/cs365/project/dqrl.html)
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
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 it 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.

Game Videos at: http://home.iitk.ac.in/~amasare/cs365/project/dqrl.html
Given the high time complexity of the training process, using an already available data of similar game can be very helpful.

![Scores of Phoenix on agent trained on Space Invaders](image)

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

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


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


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