**Atari Breakout using Deep Q-Learning**

K. Vishnu Sainadh, K. Satwik, V. Ashrith, Amudha J

Computer Science and Engineering, Amrita School of

Engineering, Bengaluru, Amrita Vishwa Vidyapeetham, India.

Department of Computer Science and Engineering, Amrita School of Engineering, Bengaluru,

Amrita Vishwa Vidyapeetham, India.

[bl.en.u4aie19028@bl.students.amrita.edu](mailto:bl.en.u4aie19009@bl.students.amrita.edu), [bl.en.u4aie190034@bl.students.amrita.edu](mailto:bl.en.u4aie190016@bl.students.amrita.edu),[bl.en.u4aie19066@bl.students.amrita.edu](mailto:bl.en.u4aie19066@bl.students.amrita.edu), j\_amudha@blr.amrita.edu

*Abstract—* The difficulties of applying reinforcement learning to current AI applications are intriguing, especially in unexpected environments with delayed rewards. Classic arcade games have recently attracted a lot of attention as a testing ground for these types of algorithms. In this study, we use deep Q-learning to create a system that can train a computer agent on how to play the popular classic arcade game Breakout. The non-human player (agent) is given no prior information of the game and must learn from the same sensory input that a human would typically receive when playing the game. The goal is to reproduce prior accomplishments by optimizing Breakout's agent-driven control to outperform a normal human score.

Keywords—Deep Q-Learning, Breakout.

# INTRODUCTION

Artificial intelligence researchers are focusing their attention on reinforcement learning (RL). It's a method for an agent to learn how to attain rewards ‘r’ through interactions with its surroundings. When dealing with a dynamic environment, such as that seen in real-world applications like robotics and self-driving automobiles, reinforcement learning is an obvious choice. An RL agent learns a mapping of states ‘s’ to optimal actions ‘a’ and creates a policy to obtain long-term rewards through exploration and exploitation of learned situations. However, the agent has a number of difficulties in learning this policy. Furthermore, each action taken by the agent has an unpredictable consequence on what is best later on. Even if the agent learns a policy that allows it to obtain rewards, we don’t know whether the policy is optimal or not. As a result, the agent must choose between investigating possibly suboptimal actions in the hopes of discovering a more optimal approach and exploiting its current policy. Learning to control agents directly from high-dimensional sensory inputs like vision and speech has long been a challenge in reinforcement learning (RL). A combination of hand-crafted features and linear value functions or rules has been used in many successful RL applications in various industries The quality of a system's feature representation has a direct impact on its performance with these systems. There are several drawbacks to reinforcement learning with deep learning. A substantial amount of training data that has been manually labelled is required for the most successful deep learning applications to date Additionally, RL algorithms must be able to learn from reward signals that are sparse, noisy and delayed. Unsupervised learning's immediate connection of inputs and goals looks immensely overwhelming with millions of time steps between actions and rewards. Many deep learning algorithms, however, presume that data samples are independent, but in reinforcement learning sequences of strongly connected states are more common. RL's data distribution may vary as a deep learning system learns a new behavior, which can be troublesome for such systems. We will develop deep q-learning using Breakout's replay memory.

# LITERATURE SURVEY

Several research groups have looked at how reinforcement learning may be used to arcade games like Flappy Bird, Tetris, Pacman, and Breakout. The article published by Google DeepMind in 2015, in which an agent was trained, is perhaps the most relevant and well-known to our research to play Atari games solely using visual input. While past implementations of reinforcement learning to arcade games relied mainly on hand-crafted feature sets, DeepMind decided to extract essential information from the game's video input using convolutional neural networks. The novel concept of a deep Q-network, which combines Q-learning with neural networks and experience replay to decorrelate states and update the action-value function, was a major contribution of the DeepMind project to the field of reinforcement learning. The DeepMind agent outperformed humans on nearly 85 percent of Breakout games after being trained with a deep Q-network.

TD-Gammon is another well-known reinforcement learning application. Without knowing the game rules, an agent learnt to play this extremely stochastic game in 1995. To estimate the value function, TD-Gammon employed a unique encoding to describe the state of the backgammon board and coupled TD-learning with a multilayer neural network. This strategy, however, did not translate well to other games, possibly because the unpredictability inherent in backgammon rules inevitably leads to adequate exploration of the state space.

The work of DeepMind and TD-Gammon influenced our decision to choose Breakout as our issue. To train an agent, both DeepMind and TD-Gammon used a non-linear function approximator in conjunction with an RL algorithm.

**GAME DESCRIPTION:-**

Break-Out is a game in which a board moves along the bottom of the screen returning a ball that will destroy the bricks at the top of the screen. The agent must learn to control the board by moving left and right returning the ball and removing all the bricks.

Brick Wall:

There are six rows of bricks. The color of a brick determines the points you score when you hit it with your ball.

* Red - 7 points
* Orange - 7 points
* Yellow - 4 points
* Green - 4 points
* Aqua - 1 point
* Blue - 1 point

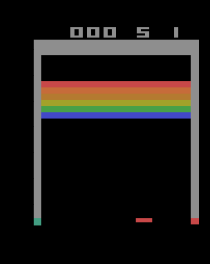


Fig 1

This is how the game looks like which can be seen in the above figure and that is the environment of our Deep Q-Learning.

**RL Formulation:​**

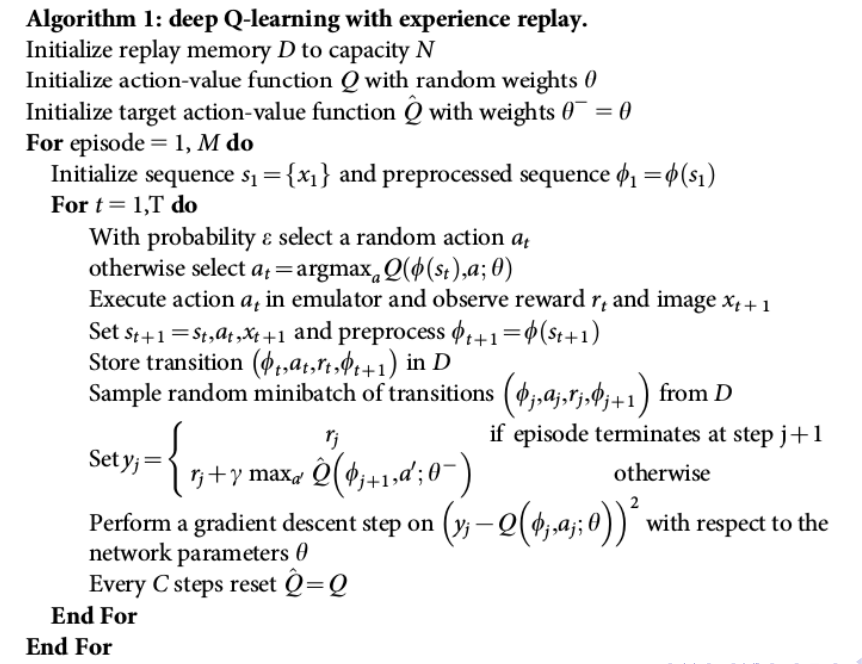
* States : A stack of four consecutive grey-scale images
* Agent : DQN
* Actions : Four discrete actions
  + - NOOP
    - FIRE
    - LEFT
    - RIGHT
* Rewards: +1 for hitting the brick and zero in all other cases.

# METHODS

**Deep Q-learning:**

In Deep q-learning, it combines the concept of Reinforcement Learning and Deep Neural Networks. To decide what action an agent should take in the environment deep q-learning will first sample a state from the environment. In our project the state is a stack of four consecutive grey-scale images. The current reward of a state and the expected reward from taking an action are combined in a weighted algorithm to create a q value of each action, the action with the highest q value will be the one taken at that step in the game and this process repeats for the entire game. Deep Q-Learning uses a neural network to find an approximation  of .θ is the hyper-parameter of the neural network.

Algorithm:



**Replay Memory:**

Across all episodes played by the agent, the replay memory records all of the agent's experiences at each time step. Actually, we'll normally see the replay memory set to a finite capacity limit in practice, so it'll only store the most recent encounters. We'll randomly take some samples from this replay memory to update the weights of the network.

When we take an action and complete a step in order to collect a reward, the network does not learn from this final step, instead we will add the experience to the replay memory. By selecting a random mini batch from the replay memory we will perform back-propagation using a gradient descent step.

# IMPLEMENTATION

Pre-processing:

It can be computationally challenging to work with raw Atari frames, which are 210 by 160 pixel graphics with a 128 color palette, therefore to reduce the input dimensionality we do a basic pre-processing step. The raw frames are pre-processed by down sampling to a 110 x 84 picture and transforming their RGB representation to grayscale. The final input representation is created by cropping an 84 by 84 portion of the image that approximately captures the playing area.



Fig 2

The above figure is the input for Deep-q-network which is a stack for consecutive frames.

Model:

Now we'll go through the precise design of the Breakout game:

* An 84 x 84 x 4 input is sent into the neural network.
* The input is convolved using 32 x 8 x 8 filters with stride 4 and a rectifier nonlinearity is applied in the first hidden layer.
* The output from previous layer is convolved using 64 4 × 4 filters with stride 2, and a rectifier nonlinearity is applied in the second hidden layer.
* The output from previous layer is convolved using 64 3 × 3 filters with stride 1, and a rectifier nonlinearity is applied in the third hidden layer.
* The fourth hidden layer is flatten layer.
* The last hidden layer is a fully-connected layer and consists of 512 neurons.
* The output layer is a fully-connected linear layer with four neurons which are the possible number of valid actions. The single output from each neuron estimates the Q-value for performing an action.

We refer to convolutional networks trained with our approach as Deep Q-Networks (DQN). We created two models. In which, the first model makes the predictions for Q-values which are used to make an action. And the second model is the target model which is trained using the weights generated by first model after some number of iterations.

Hyper-parameter Tuning:

The hyper-parameters used in the training are:

* Discount factor (): Discount factor gamma is used in Q-learning update.
* Initial exploration (): Initial value of in -greedy exploration.
* Final exploration (): Final value of in -greedy exploration.
* Final exploration frame: The number of frames over which the initial value of exploration value () is gradually decayed to final value.
* Replay start size: Before estimating the Q-values we first save the previous transitions of this size. Which can be used as experience replay before learning starts.
* Replay memory size: Size of the previous calculated transitions.
* Batch size: Size of random number of transitions taken from replay buffer which is used for updating weights.
* Max Steps per episode: Maximum steps of an episode.
* Update frequency: The frequency with which the model is trained.
* Target network update frequency: It specifies how frequently the target network is updated.
* Learning rate: The learning rate used by Adam.

Hyper-parameter tuning is being done with three different values of gamma, Learning rate and update\_after\_actions.

For gamma hyper-parameter, values taken are 0.1, 0.5, and 0.99. The plots of number of episode vs rewards received are shown in the graphs below:

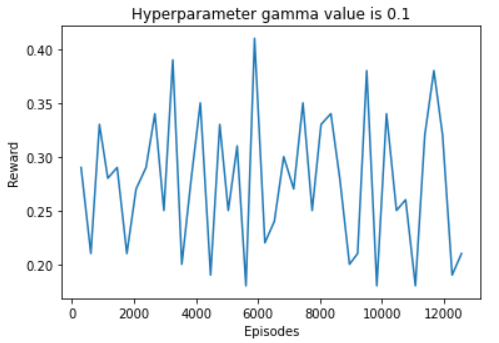


Fig 3a

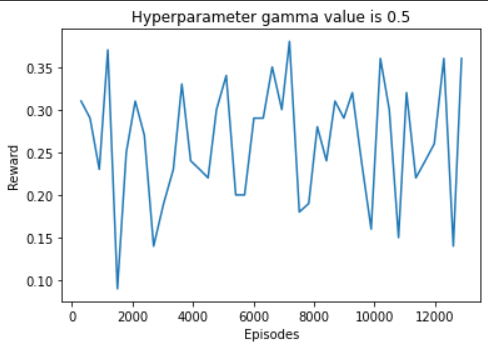


Fig 3b

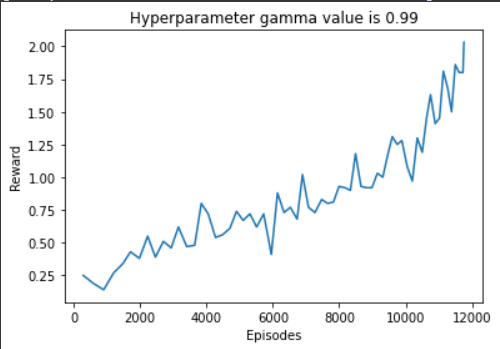


Fig 3c

From the figures 3a, 3b, 3c we can see that we get better reward progress as the number of episodes increases. So we can observe that setting the hyper-parameter gamma value to 0.99 we get better progressive rewards as number of episodes increase in training the DQN. So from now we will fix this hyper-parameter and tune other hyper-parameters.

For Learning Rate hyper-parameter, values taken are 0.01, 0.0025, 0.001, and 0.00025. The plots of number of episode vs rewards received are shown in the graphs below:

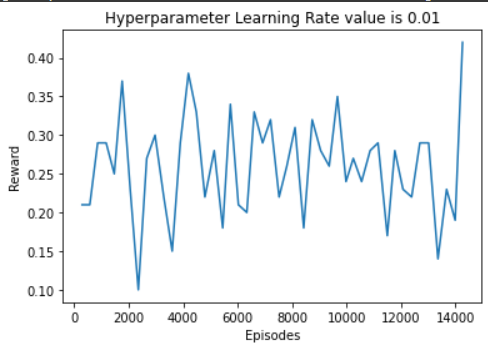


Fig 4a

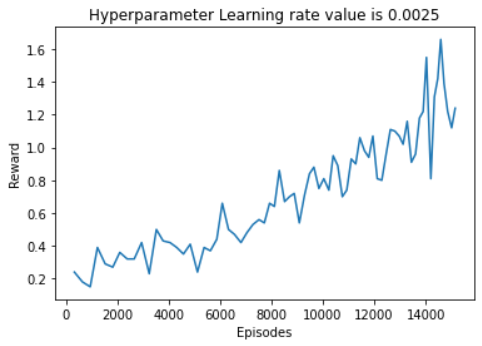


Fig 4b

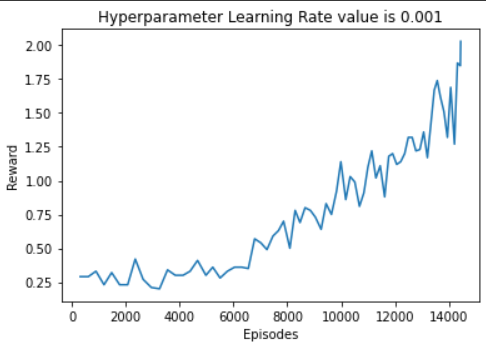


Fig 4c

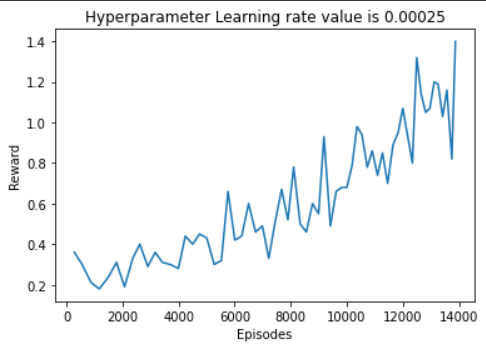


Fig 4d

From the figures 4a, 4b, 4c and 4d we can see that we get better reward progress as number of episodes increases. So we can observe that setting the hyper-parameter learning rate value to 0.001 we get better progressive rewards as number of episodes increase in training the DQN. So from now we will fix this hyper-parameter and tune other hyper-parameters.

For update\_after\_actions hyper-parameter values taken are 10, 20, and 30. The plots of number of episode vs rewards received are shown in the graphs below:

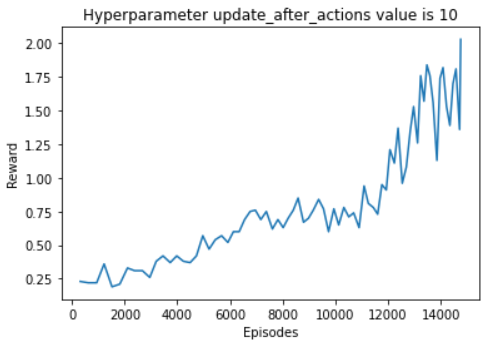
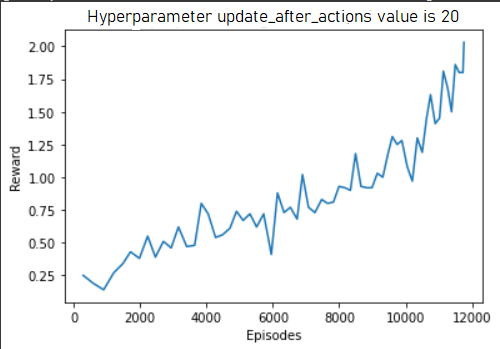


Fig 5a



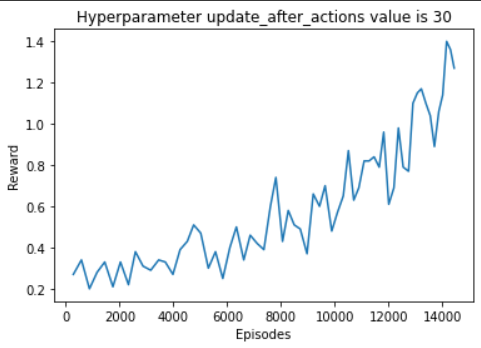


Fig 5c

From the figures 5a, 5b, 5c we can see that we get better reward progress as number of episodes increases. So we can observe that setting hyper-parameter update\_after\_actions value to 10 we get better progressive rewards as number of episodes increase in training the DQN.

The hyper-parameters which performed well are gamma = 0.99, learning rate = 0.001, and update\_after\_actions = 10.

The Hyper-parameter values used in training the DQN are given below:

|  |  |
| --- | --- |
| Hyper-parameters | Values |
| Discount factor () | 0.99 |
| Initial exploration () | 1 |
| Final exploration () | 0.02 |
| Final exploration frame | 1000000 |
| Replay start size | 50000 |
| Replay memory size | 50000 |
| Batch size | 32 |
| Max Steps per episode | 10000 |
| Update frequency | 10 |
| Target network update frequency | 10000 |
| Learning Rate | 0.001 |

Target network and Parameter Update:

We randomly pick a mini-batch of size 32 before changing the network's parameters. A gradient descent step is performed: The main network looks at state and estimates the - values that say how good each action is. The - values, on the other hand, should follow the Bellman equation we discussed before. As a result, we use the Bellman equation to compute the -values (as we want them to be) and then compare the estimates to the targets. Values depend on the current states which are picked from the mini-batch and on the hyper-parameters of the DQN that estimates it.

The   value is computed using Bellman equation. It is the sum of the immediate reward r received for performing action ‘a’ in state ‘s’ (action and state from the mini-batch) and the maximum Q-value over all possible actions a′ in s′ (new state from the mini-batch):

The issue is that if just one network is employed, both and are dependent on the same parameters theta. Because the "target is shifting," this might cause instability while regressing to . We achieve a "fixed target" by introducing a second network that calculates the target Q-values and has fixed and only infrequently modified parameters.

As a result, one network was used to forecast the - value, while the other fixed network was used to predict the -value. During the gradient descent phase, the main network is optimized, and the parameters of the main network are transferred to the target network every 10000 steps. Keep in mind that the number of chosen actions/frames viewed determines the network update frequency.

Demo-Code**:**



Fig 6

In this code the base-line agent has been implemented and at last we plotted episode vs reward. Here one episode is considered as one life.

# RESULTS

Even though we tuned the hyper-parameters and the took the best the highest score that we were able to reach is 41.Overall we trained the final model for over 1 hour and the model was still not efficient as possible with more training time we believe this model can effectively be a game of breakout. The agent was not able to learn to maximize rewards over 14,000 episodes. Video of the agent playing the game has also been generated which helps us to visualize the gameplay.

# CONCLUSION

In this paper, for the agent to play Breakout we used Deep Q-Learning. We used four consecutive pre-processed frames in a stack as an input to DQN in this work. We used this deep Q-Network as a agent for reinforcement learning and verified its ability to estimate the Q-values for the Breakout game. Replay Memory benefitted our model and helped to perform well.

Finally, we discuss what improvements could be made to better the agent in the future. We proposed different possible techniques from recent studies: Double Q-learning, Prioritized Experience Replay, and Parallel Threads.

# References

[1] Mnih, Volodymyr, et al. “Playing atari with deep reinforcement learning.” arXiv preprint arXiv:1312.5602 (2013).

[2] Adam, Sander, Lucian Busoniu, and Robert Babuska. “Experience replay for real-time reinforcement learning control.” IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42.2 (2012): 201-212.

[3] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In Proceedings of the 12th International Conference on Machine Learning (ICML 1995), pages 30–37. Morgan Kaufmann, 1995.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[5] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” Journal of Artificial Intelligence Research, vol. 47, pp. 253–279, Jun. 2013.

[6] C. J. C. H. Watkins, “Learning from delayed rewards,” King’s College, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/∼chrisw/new thesis.pdf

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, “Playing atari with deep reinforcement learning,” CoRR, vol. abs/1312.5602, Feb. 2013. [Online]. Available: <http://arxiv.org/abs/1312.5602>

[8] Karpathy, Andrej, “Deep Reinforcement Learning: Pong From Pixels”. karpathy.github.io. N.p., 2016. Web. 7 Dec. 2016.

[9] Kuan, C-M., and Kurt Hornik. “Convergence of learning algorithms with constant learning rates.” IEEE Transactions on Neural Networks 2.5 (1991): 484-489.