Deep Reinforcement Learning for Flappy Bird

Sudarshan Srinivasa Ramanujam, Gowtham Ganesan
Balasubramaniam Srinivasan, Charumathi Lakshmanan, Kumaran Gunasekaran

PID : A53220043, A53203033, A53210041, A53220876, A53207045
UCSD School of Computer Science

sus046@eng.ucsd.edu; gganesan@ucsd.edu;
bsriniva@eng.ucsd.edu; clakshma@ucsd.edu; kugunase@eng.ucsd.edu;

Abstract

The advent of reinforcement learning has enabled machines and software agents to automatically determine the ideal behaviour within a specific context, in order to maximize its performance. In this project, we expand upon Deepmind's work on Deep reinforcement learning on Atari games and implement similar algorithms on the popular Flappy bird game. Model free, Reinforcement Learning using Experience Replay was employed to collect samples in stochastic manner for training.Double Deep Q Learning (DDQN) as well as prioritized experience replay techniques were also employed. The method of Human teaching for populating the first few high priority experience replays was also experimented with the aim of further reducing the training time. Additionally transfer learning was explored by transferring the knowledge learned by training the Deep Q Network on Flappy bird to the Pixel Copter game and the speed and accuracy of learning was compared between transfer learning models.

1 Problem Definition

Reinforcement learning is a popular learning method used by autonomous agents to learn actions in an environment using reinforcement signals(rewards) which act as a feedback to the agent. In a real world scenario there could be several possible states of the environment and it will be very difficult to draw features out of this highly structured data. A simple and concise way to represent current state of the environment would be to capture a screen shot of the environment. For example in the DeepMind's work on Deep Q Networks(DQN), the inputs to the network are 84*84 gray scale images representing the environment at the current time step. Also they stack 4 screen shots of the network's environment to be able to learn based on the dynamics(direction,velocity) of the agent. This gives us a total of $256^{84*84*4}$ possible pixel states. Using traditional q-learning techniques (such as value iteration) to solve this problem is going to be computationally very hard and the q-table will take infinitely long to converge. To solve this problem effectively we must be able to draw good features out of the input image that compactly represent the state of the environment. This is where a neural network helps us as it learns the features in the service of the task and led to the use of Deep networks for reinforcement learning where the deep network predicts the q-value for each possible action of the agent and the agent takes the action corresponding to the maximum q-value (function approximation from state to action).

In this paper we have replicated the work done by the DeepMind in training a Deep Q Network to play Flappy Bird as a baseline and studied the effect of different heuristics on the speed of learning and the performance of the agent. Prioritized experience replay was employed to pick batches from replay memory for training the network. The effect of a human teacher during the initial phase to create experience samples was explored. The network was also trained using Double Q Learning algorithm and the performance of such a Double DQN was compared against the baseline DQN. Also the effect of transfer learning was studied on Copter(a Helicopter game) with Flappy Bird as the source task. Copter was chosen for Transfer Learning since the game states in both these games are very similar and the actions taken by the agent are also the same. We wanted to see how well a DQN trained on Flappy bird generalizes to play Copter with minimal additional training.

2 Background

The primary inspiration for this project was provided by Deepmind's 2013 NIPS paper by Mnih, et al. [4]. The follow up on this paper by Mnih, et al. [5], which had details on implementation and convolution net architectures, served as the primary source of our reference. In these papers, Deep Convolution Networks were applied successfully to beat human level performance in majority of the Atari games except for a few games like Bank Heist, River raid etc.

A few ideas implemented in our project stemmed from contributions much earlier than the work by Mnih, et al. For instance, prioritized sweeping which is used to pick experience replay samples of greater significance as opposed to random sampling was proposed as early as 1993 by Andrew W. Moore and Christopher G. Atkeson [6]. In this paper, experience replays are stored in a data structure, wherein the state of maximum interest and the states leading up to it (predecessors) are stored with their respective priorities in a priority queue. Prioritizing which transitions are replayed make the process more efficient in comparison to randomly sampling for all available replays. Prioritizations are done with the help of the magnitude of temporal difference error across transitions and they are selected stochastically along using the priorities assigned with them. The bias generated through this method is then corrected with the help of importance sampling as done by Schaul, et al [7]

The problem of Q-value overestimations in vanilla DQN was first investigated by Thrun and Schwartz [11]. They showed that when there are random errors in action values, the overestimations manifest themselves and proposed Double Q learning as a solution. Hado van Hasselt, et al [12] prevented the overestimation problem by decoupling the selection of action and evaluation of Q-value. We adopted these changes in our project work using two different weights - θ for selecting the greedy policy and $\theta^{`}$ for Q-value evaluation.

In the paper by Du, et al. [1] the concept of transfer learning for deep reinforcement learning algorithms have been explored by applying them to cart-pole and Atari games without breaking the network structure. In our we work we have tried using transfer learning (3 variants) by using parameters learnt for the flappy bird game to the copter game.

We have also experimented using human teaching for the initial observation phase as had been done by Thomaz, et al. [10]. This work demonstrates the importance of understanding the human-teacher/robot- learner system as a whole in order to design algorithms that support how people want to teach while simultaneously improving the robot's learning performance.

3 Network Model and Methodology

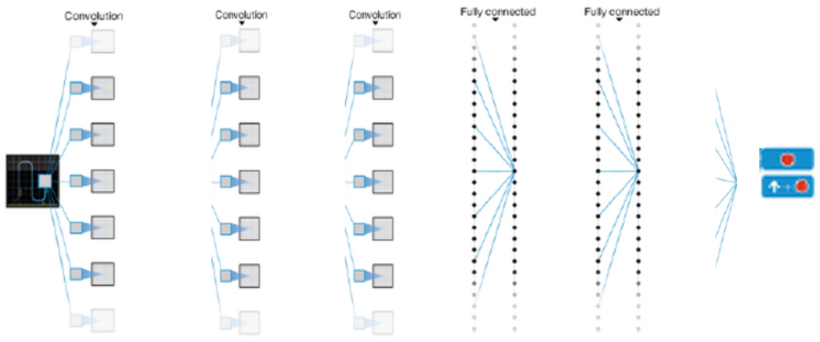
3.1 Network Model

In this paper, the same architecture used by DeepMind was used.

The exact architecture is as following :
1. To estimate velocity and direction of the bird we stacked 4 image frames at each time as input to the convolution layer.
2. The input to the neural network consists of (288x512x3) images reshaped to (84x84) gray scale images. The first hidden layer consists of 32 filters of size 8 x 8 with stride 4 and

applies ReLU activation function.
3. The 2nd layer consists of 64 filters of 4 x 4 with stride 2 and applies ReLU activation function.
4. The 3rd layer consists of 64 filters of 3 x 3 with stride 1 and applies ReLU activation function.
5. This is followed by a Dense layer of 512 units with relu activation and a final Dense output layer with 2 units for each action. This final layer has linear activation where the output represents the Q-value for each action.

Figure 1: Network Architecture of Convolution Deep Net used [5]



The network shown in Fig.1 was used for training the Flappy bird network. The last layer was replaced with 2 output cells as the game only has 2 actions.

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 84, 84, 4)     0
____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 21, 21, 32)    8224        input_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 11, 11, 64)    32832       convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 11, 11, 64)    36928       convolution2d_2[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 7744)          0           convolution2d_3[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 512)           3965440     flatten_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 2)             1026        dense_1[0][0]
====================================================================================================
Total params: 4,044,450
Trainable params: 4,044,450
Non-trainable params: 0
____________________________________________________________________________________________________
```

3.2 Methodology

The algorithm that is being implemented is known as the ”Model Free Deep Q Learning” in the literature. Since the number of states is exponentially large in the game, it is impossible to take a model based reinforcement learning approach. Hence a model free approach is applied for this problem wherein a convex loss function is defined and mini batch gradient descent is applied to minimize loss.

3.2.1 Dataset Description

For this project, there is no need of external datasets. Screenshots from the game and the game states (reward, termination state) are retrieved from the game API and stored in a replay memory from which training samples are drawn stochastically.

3.2.2 Pygame API

API for Flappy Bird and Pixel Copter were used from [8] and [9] respectively.

The game API (after few customizations) returns the following game state information to our application:

1. Reward for current step
2. Terminal information (whether the game is over or still in progress)
3. RGB Image frame of current game state

3.2.3 Detour to Deep-Q-Learning

The premise of the deep Q learning algorithm comes from the bellman equation, where a framework can be extracted to maximize future rewards using a greedy strategy.

Lets denote r_t as the reward obtained for timestep t when transitioning from state s_t to state s_{t+1}. The total future reward from timestep t onwards can be expressed as

$$R_t = r_t + r_{t+1} + r_{t+2} + ... + r_n \tag{1}$$

Since the environment is stochastic one can never be certain that we will continue to get the same rewards in the future. For this reason, we add in a discount factor for future rewards.

$$R_t = r_t + \gamma \left(\sum\nolimits_{i=t+1}^{i=n} r_i \right) \tag{2}$$

Typically as discount factor is low, it means that the algorithm is short-sighted (we are relying only on immediate rewards). Typically the discount factor is set to 0.9 to look more into maximizing future rewards.

Q_t value is defined as the the maximum discounted future reward when we perform an action 'a_t' in state 's_t', and continue optimally from that point on.

$$Q\left(s_t, a_t\right) = maxR_{t+1} \tag{3}$$

Since we cannot see far into the future from our current state, the way we can maximize future rewards is by employing a greedy strategy. We take the action 'a' that maximizes reward at the next time step. The proof of convergence by employing this strategy can be seen in the paper on Q learning by Christopher J. C. H. Watkins and Peter Dayan [13]. The bellman equation for one transition $(s, a, r, s')$is given by:

$$Q\left(s, a\right) = r + \gamma max_{a'} Q\left(s', a'\right) \tag{4}$$

In the network model shown in Fig.1, the two outputs denote the Q values for the 2 actions possible. The network is learn a function optimizer which is the Q-value for the two possible actions. The loss function that we are trying to optimize is given by the following equation.

$$L = \frac{1}{2}\left[r + \gamma max_{a'} Q\left(s', a'\right) - Q\left(s, a\right)\right]^2 \tag{5}$$

where s' correspond to image that appears after an action a is taken at state s. In this equation the term on the left side of the difference is the actual Q-value while the term on the right side is the Q-value predicted by the DQN. In our implementation we keep a set of 4 images in every state. In every time step one old image is removed and a new game state image is appended to the state list.

3.2.4 Experience Replay

As seen in the previous section, the difference between the Q values at state s' and state s is taken as the error that is back propagated to the network. We keep an experience replay memory in the form of a queue that stores the tuple (s_t, a_t, s_{t+1}, r_t). Initially we let the bird run without training the network for 2500 iterations and store the sampled images in the replay memory. In order to perform the mini batch gradient descent for this problem samples are drawn from the replay memory. In this report we explore 2 variations in which we draw samples from the replay memory.

In method 1, 32 unique samples are drawn at random from the replay memory following a uniform distribution.The drawback with this approach is that all the experiences are given equal probability of being replayed regardless of how important they are. It has often been observed in reinforcement learning that certain experiences contribute more to learning than certain experiences.

To fix this drawback, in method 2, an extra parameter called priority was added to each element being stored in the replay memory $(s_t, a_t, s_{t+1}, r_t, p)$. Priority p is defined according to the following equation:

$$\Delta = |max_{a'} Q\left(s', a'\right) - Q\left(s, a\right)| \tag{6}$$

where Δ is called the temporal difference or in other words the error in the prediction.

$$p_i = \Delta_i + \epsilon \tag{7}$$

where p_i is the priority of the i^{th} element, Δ_i is the temporal difference and ϵ is a bias(to ensure a non-zero priority even for samples with zero temporal difference).

$$P_i = \frac{(\Delta_i + \epsilon)^\alpha}{\Sigma_k (\Delta_k + \epsilon)^\alpha} \tag{8}$$

where P_i is the probability of the i^{th} element being picked from the replay memory and α is the amount of prioritization. When alpha=0, it reduces to the uniform random sampling described in method 1. This idea of prioritized random sampling is called prioritized sweeping in literature.

Candidates for replay are chosen from the replay memory by a stochastic multinomial distribution driven the probability P_i. Every time an experience is chosen, it's priority is annealed by a multiplicative factor of 0.1 so as to reduce the selection bias introduced by high priority experiences.

3.2.5 Double Deep Q Learning

Deep Q learning tends to over estimate the state value, resulting in over optimistic value estimates. This is counter productive and does not assist in learning. The idea of double Q learning is to reduce overestimations by decomposing the max operation in the target into action selection and action evaluation[12].

We use two networks to implement the Double Deep Q learning: the prediction network and the actual network. The third term in eqn 5 (Q(s,a)) is still going to remain the same. The first term however is modified in the following manner:

1. Run forward pass on state s' with the predict model
2. Find out which action maximizes Q value (argmax a)

3. Run forward pass on s' with the actual model Choose the Q value of the action index obtained from point 2 to compute the squared loss
4. Backpropogate error to train the predict model
5. After N iterations (1000 for this project), we assign the weights of the actual model to the predict model

By implementing the above steps we ensure that co-relation and over estimation of Q values is controlled resulting in faster convergence.

3.2.6 Exploration-Exploitation

The initial actions taken by the DQN are pretty much random since the weights of the network are randomly initialized.This could be called as an exploratory phase in the algorithm. However as the Q-function converges,the network settles to more predictable actions and the exploration ceases gradually.This phase is called exploitation. In other to avoid settling in a local maxima it is necessary to keep exploring frequently.

One way to fix this problem is by choosing a greedy exploration strategy wherein a random action is chosen intermittently. The probability of choosing a random action is driven by ϵ, the exploration factor. In our methods we have chosen an exploration factor of 0.1 and have used the same value throughout the training process without annealing.

3.2.7 Transfer Learning with Pixel Copter

The trained network for the Flappy bird was used to evaluate the performance of transfer learning to the pixel copter game. The API of the pixel copter game returned and accepted the same values as that of the Flappy bird game.

Three strategies for transfer learning were tried out and evaluated :

1. Initialize network with weights of the network trained on Flappy bird and re-train all layers of the network
2. Initialize network with weights of the network trained on Flappy bird and re-train all layers except the first convolution layer
3. Initialize network with weights of the network trained on Flappy bird and train all layers except the first 2 convolution layers

One challenge seen in transferring knowledge from Flappy Bird to Pixel Copter is that in Flappy bird there is no decision that has to be made since there is only one way the bird can move at any time step. However in Pixel Copter the learning agent has to make a choice about the direction it has to fly since the multiple ways to get a short term reward,not all of which lead to a maximized long term reward . Hence, we believe transferring knowledge from Pixel Copter to Flappy Bird will be more efficient in terms of the training time.

3.2.8 Meta parameters

SNo	Meta-parameter	Value
1	Learning rate for Adam optimizer	$1e^{-6}$
2	Mini batch size	32
3	Replay memory	10000
4	Exploration rate	0.1
5	Discount factor γ	0.99
6	Prioritized sweeping ϵ	0.001
7	Prioritized sweeping α	0.7

Table 1: Meta parameters

3.2.9 Credit Assignment

We used rewards of -3 when the bird collides with the top and bottom of the screen and a reward of -2 when the bird collides against the pipe. As for positive rewards we give the bird with +2 points when it crosses a pipe successfully and +0.5 points while moving through the passage between the pipes. At every other step the bird receives a positive reward of 0.1.

4 Results and Discussion

An episode in the below plots is defined as a collection of 50 continuous games.
Score in the below plots is defined as the accumulated total reward per game (before the bird hits an obstacle and dies)

4.1 Training with Uniform Random Sampling from Experience Replay

Figure 2: Max Q value vs Episode

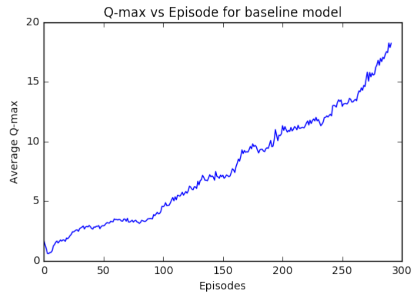
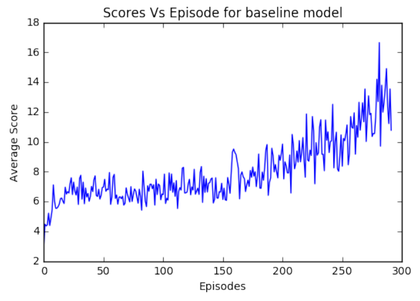


Figure 3: Score vs Episode



In Fig.2 and Fig.3 we can see that both the scores and the max Q value keep increasing along with increase in time steps. This clearly is due to the fact that the rewards kept growing with number of time steps which is tantamount to the bird progressing further in the game.

The baseline Flappy bird model was trained for a period of 5 days(600,000 timesteps) in total on a local machine, after which the bird is currently able to cross 25 pipes on an average on testing.

4.2 Training with Prioritized Sampling from Experience Replay

In this section, the mini-batch elements (s_t, a_t, s_{t+1}, r_t) for every time step are chosen as per the priority values described in section 3.2.4 (method 2). With prioritized sweeping the training time has to decrease and our results are coherent with this idea.

Figure 4: Q-max Comparison between random and prioritized Sampling

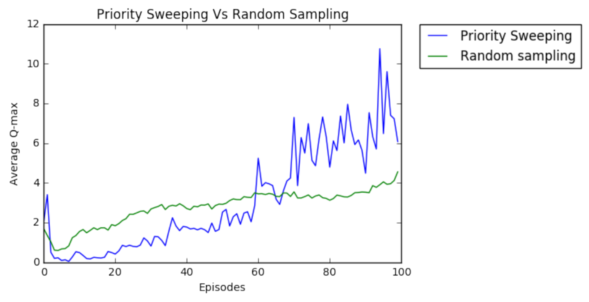
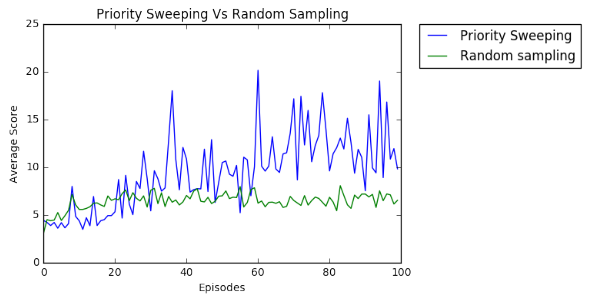


Figure 5: Scores Comparison between random and prioritized Sampling



From Fig.4 and Fig.5, we can clearly see that the prioritized sampling method maximizes both the actual(scores) and expected(q-max) rewards and learns at a much faster rate as compared to the random sampling method.

With prioritized sweeping method, the Flappy bird was able to cross a maximum of 10 pipes(and average of 5 pipes) in just 50,000 iterations which is a huge gain in the training time compared to the uniform sampling approach.

4.3 Double DQN

As described in the previous section we explored the concept of double Q-learning which separates the action selection and action evaluation process in the Deep Q Networks and prevents the network from overestimating the action values while still maximizing the rewards in a stable manner.

Figure 6: Average Q-max vs Episode - DDQN

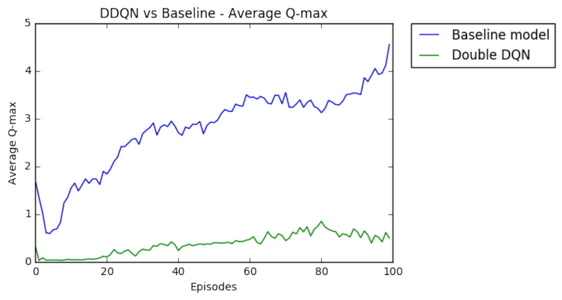


Figure 7: Average Score vs Episode - DDQN

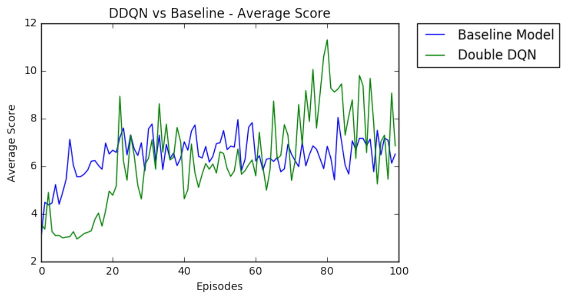
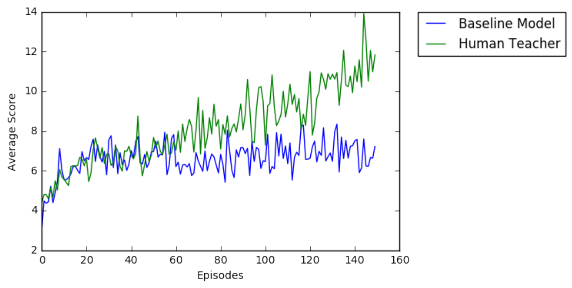


Fig.6 matches our expectation as we see that the Q values are lesser than the Q values for the baseline model while the scores obtained by both the models is comparable as evident from Fig.7. These results prove that the DDQN prevents overestimation of action values and predicts values that are more closer to reality.

4.4 Experience Generation with Human Teacher

In our baseline model we had populated the replay memory by taking a series of random actions during the initial phase of the learning process. While this worked well, we contemplated achieving quicker learning by using a human teacher to populate the initial replay memory as we are not creating more relevant samples for the network. As shown by Fig.8, the model with a human teacher has scored more points in the same number of timesteps as the baseline model.

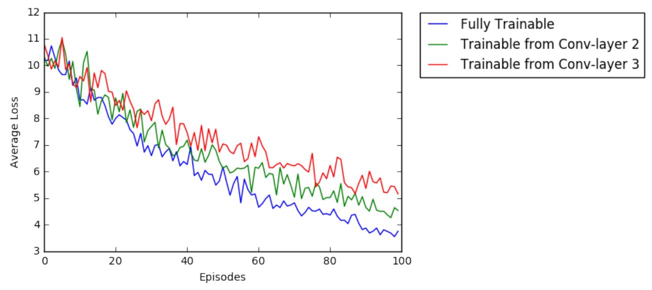
Figure 8: Average Score vs Episode - Human Teacher



4.5 Transfer Learning to Pixel Copter

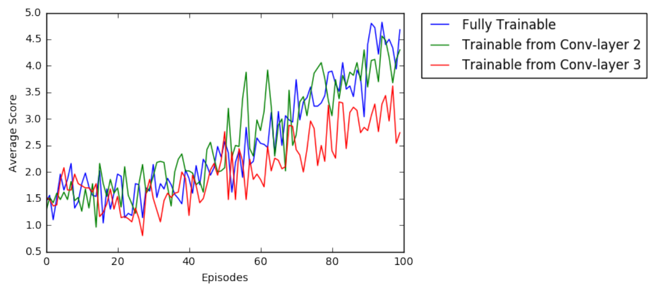
Three variants of transfer learning were implemented as discussed in the previous section. We trained one model with all the network layers trainable and one with the first convolution layer non-trainable and an other model with the first two convolution layers non-trainable. Since the initial layers of a CNN extract basic features out of an image and the two games in our consideration have some basic similarities, we hypothesized that the model with the first convolution layer non-trainable should be able to learn quicker that the other two models and also believed that the model with all the layers trainable should eventually perform better than the other models.

Figure 9: Average Loss vs Episode



However as evident from the loss plot shown in Fig.9 we found that the model with all the layers trainable was able to learn faster than the other two models. This could be due to the considerable differences in the terrain of the two games. Also the model with two non-trainable layers learned the slowest which is understandable since we are making the high level feature maps fixed and overload the fully connected layers to learn more features.

Figure 10: Average Score vs Episode



From Fig.10 we can see that the performance of the model with all the layers trainable is comparable to the one with the first layer non-trainable. Again we see that the model with the first two layers set as non-trainable, performs the worst.

4.6 Filter Visualization

Figure 11: Flappy Bird Original Game Image

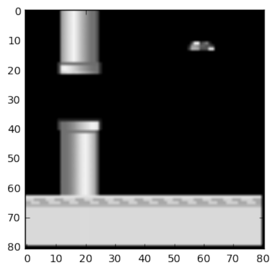


Figure 12: First Layer Filter Activations for Fig.11

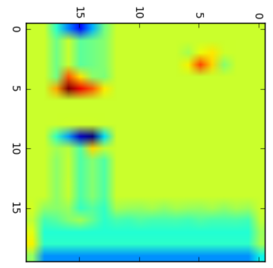


Figure 13: Pixel Copter Original Image

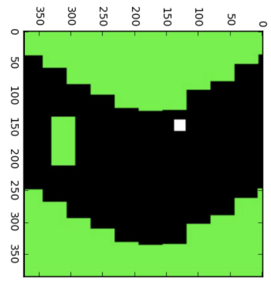
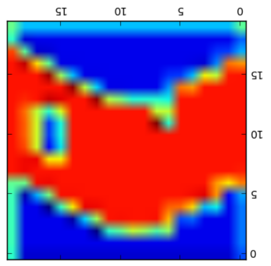


Figure 14: First Layer Filter Activation for 13



The colors in the above images represent heat maps. The behavior of the first layer of filters was as expected. The edges in the image are detected in the first convolution layer. The obstacles are clearly differentiated from the path which the copter and flappy bird can take in the arena.

5 Conclusion

We successfully implemented the baseline model as given in the Human Level Control through Deep Reinforcement Learning work. We followed this up with the implementation of Double Deep Q Learning and the results obtained were substantially better for the same number of timesteps. We then implemented a prioritized experience replay version to further reduce the training time required to achieve the same score as the previous models (baseline and DDQN). The effect of having human teacher signals during the observation phase was also successfully studied. Finally we also performed transfer learning (3 variants) by using the weights learned from the flappy bird game to the pixel-copter (helicopter) game.

6 Project Innovations

We have made some improvements over the baseline model with the following innovations:

1. Applied prioritized experience replay to the Flappy bird game and used an annealing method to the re-prioritize already drawn experiences as opposed to the importance sampling method proposed in literature[7].
2. Applied transfer learning to the copter game using three different approaches - one where the parameters from the Flappy bird game are used as the initial parameters and the model is re-trained completely. Second, using the parameters from the trained Flappy bird as initial weights and re-training from the second convolution layer.Third, using the parameters from the trained Flappy bird as initial weights and re-training only from the third convolution layer.
3. Human Teacher was used to populate the initial experience replay memory to further speed up the training time.
4. Experimented changing the learning rate(0.0025) proposed by DeepMind to $1e^{-6}$ since using a learning rate of 0.0025 made the algorithm converge to a local maxima.

7 Roadblocks

For our project, we were unable to successfully use AWS. Since the display was to be captured for each timestep and the lack of a powerful integrated display device with the Xeon cores it was much slower on AWS. In fact, the application was running faster in our personal laptops than in the GPU which was something unprecedented for us. We also tried to use "compute optimized amazon instances" which was still slow. A possible reason could be due to presence of a stronger integrated display device in the CPU's of our personal laptops rather than in the AWS instance(s).

Since the training takes a long time in itself, the problem was compounded since we do not have faster compute speeds. Nevertheless, we have managed to show more than a proof of concept in our work which can be seen in the results.

8 Individual Contributions

We worked as a team to understand about Deep Q networks and revised our knowledge of reinforcement learning. We collectively discussed the problem and decided on the innovations and experiments we wanted to do on the baseline model. The first and second authors worked on developing the baseline model and human teacher based reinforcement learning. The third author worked on prioritized experience replays and Double DQN. He previously worked on setting up the AWS. The fourth and fifth authors initially contributed to the baseline model and later worked on Transfer learning. They also worked on customizing the pygame API to work well with our code.

9 Acknowledgements

We would like to thank Professor Garrison W. Cottrell and his teaching assistants for their valuable inputs over the different stages of the project. We would also like to show our appreciation towards Tambet Matiisen for his blog titled "Demystifying Deep Reinforcement Learning" [3] which served as the primary reference for our baseline model. We would also like to thank Ben Lau for his blog [2] which helped us solve implementation issues while building the baseline model.

References

[1] Yunshu Du, V Gabriel, James Irwin, and Matthew E Taylor. Initial progress in transfer for deep reinforcement learning algorithms.

[2] Ben Lau. Using deep q-network to play flappybird.

[3] Tambet Matiisen. Demystifying deep reinforcement learning.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[6] Andrew W Moore and Christopher G Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

[7] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[8] Norman Tasfi. Flappy bird api.

[9] Norman Tasfi. Pixel copter api.

[10] Andrea Lockerd Thomaz, Cynthia Breazeal, et al. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Aaai*, volume 6, pages 1000–1005, 2006.

[11] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

[12] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[13] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.