2020

Deep Reinforcement Learning: Flappy Bird Game

PRANAV LODHA, SUBARNA CHOWDHURY SOMA, JEYASRI SUBRAMANIAN

SAN JOSE STATE UNIVERSITY | 1 Washington Square, San Jose, CA

Table of Contents

1. Executive Summary

Reinforcement learning is a subfield of Machine learning which solves the problem by automatic learning and optimal decision making. With the improvement in Deep architectures in Reinforcement learning, we can solve the system involving large datasets and complex gaming applications. In this project, we have applied Deep Reinforcement Learning to Flappy bird agent.

Flappy bird was a popular game till 2015, which created in 2013. It was a highly addictive game which inspired many friendly battles to achieve the highest score. The agent is a bird that needs to continuously fly forward and avoid a set of pipes or obstacles. Hitting these pipes or ground will cause the bird to die and will end the game. If the agent successfully navigates between the pipes, the agent gets a positive reward of +1.

In this project work, we have applied Double Dueling Deep Q-Network (D3QN) with Prioritized Experience Replay to make the agent fly infinite times. We have analyzed each hyperparameter and fine-tune the results to determine the best performing model. We did a comparison study with base DQN network, outcomes and metrics from the project explained in detail in the following sections.

2. Background / Introduction

When the agent interacts with the environment, it gets a reward for the action and also the current state and the possible next state. The complexity of this optimality problem can be resolved by *Bellman Equation,* at every state the agent ends up in, it needs to select the action with the maximum expected reward for the action, which is a sum of the immediate reward and the one-step discounted long-term reward.

$$Q(s,a) = r_{s,a} + \gamma \max_{a' \in A} Q(s', a')$$

Deep Q Network

The above Q-learning method struggles to solve a complex problem involving a gaming environment or large datasets. Hence, Deep Q Network is applied to optimize the learning with the multi-layered neural network with Stochastic Gradient Descent. Such networks failed to converge due to SGD optimization with i.i.d (independent and identically distributed) data. After we accumulate a large batch of the training sample, the samples are not independent. Another limitation is, lack of i.i.d data in training leads to no difference between Q (s, a) and Q (s′, a′) in the Bellman equation, which leads to zero learning. The max operator in DQN uses the same values both to select and to evaluate an action, resulting in over-optimistic value estimates.

Double Deep Q-Network

To avoid overestimation issue in DQN, we use two separate network – source and target network. The actions for the next state computed from source network, but Q-values read from the target network.

$$Q(s_t, a_t) = r_t + \gamma \max_a Q'(s_{t+1}, \arg\max_a Q(s_{t+1}, a))$$

In weights of the target, the network copied at every tau step from the source network.

Experience Replay

To resolve the SGD Optimization issue with i.i.d training data, we use the replay buffer to store past experiences and randomly sample the mini-batches for training. Replay buffer allows us to train independent and recent data from samples. Double DQN algorithm is applied in experience replay for better optimal learning.

Prioritized Experience Replay

This method improves the efficiency of training by adding samples with priority in addition to training loss. The data with less certainty has more entropy, by adding a priority value to replay buffer provokes convergence faster. The priority of every sample in the buffer calculated as

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Buffer is updated with current state(s), action(a), reward (r), next state (s') and priority(p) and during training, we sample the experiences by underlying distribution to reduce KL divergence.

Dueling DQN

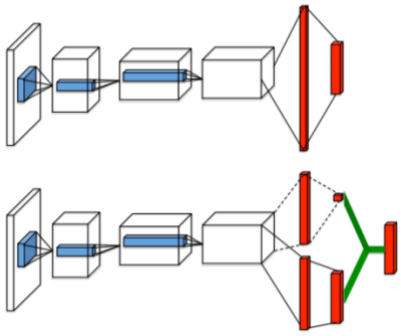


Figure 1: A basic DQN (top) and dueling architecture (bottom)

From Figure 1, dueling architecture represents two separate, fully connected layers to compute V(s) and A(s, a), which decompose to Q(s, a). The separate networks for value and action computation are the crucial difference between dueling architecture in comparison to basic DQN. In this network, we have two estimators; one estimates the score of the current state and the other estimates the action score.

$$Q(s, a) = A(s, a) + V(s)$$

V(s) represents the value of the state and A(s, a) illustrates the advantage of taking that action in that state (the value will be positive if it is a right action).

3. Problem Statement

The flappy bird gaming environment has a bird agent with two possible actions 1) Flap 2) Do nothing. The bird continuously moves forward in an infinity space with random pipes, and each flap action allows the bird to avoid touching the ground. If the bird makes contact with the ground, pipes or goes above the top of the screen, it will terminate. Each state is read as 288 X 512 X 3 size RGB image and observation space is an array of float values representing bird height, bird speed, distance to next pipe (x), the height of next pipe(y), distance to the second pipe, size of the second pipe (y1) as shown in figure 2.

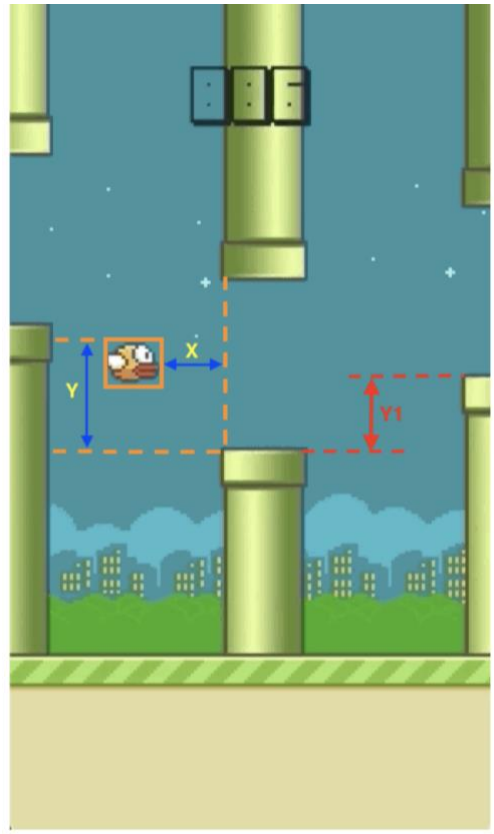


Figure 2: Flappy bird environment space

We utilized a Double Dueling Deep Q-Network (D3QN) with Prioritized Experience Replay to train this environment. This game is challenging as the agent has only two actions to perform, and also the speed of the agent increases overtime. We compared the results with other reinforcement learning algorithms and fine-tuned the parameters to improve the performance of the algorithm. In the following section, we covered the purpose, training methodology, implementation and results in comparison to our work.

4. Motivation

From the literature study and understanding of various Deep RL algorithms, we applied dueling architecture as the advantage function for action is helpful in this environment. In Dueling architecture, the Q value depends on how good in taking that action in the state. A special aggregation layer combines the estimators for V(s) and A (s, a) to get an estimate for Q (s, a). By decoupling value and action computation, the network can estimate the value without the effect of action for each state. With DQN, the value of the state measured from the action performed by the agent. This approach has a limitation, if all the actions performed by the agent in a particular state led to death, then the computation of value for that state has no relevant.

In Flappy bird environment, the agent has limited actions and the action has very high impact on the q-value. Hence applying the action advantage function is highly relevant to this environment. Prioritized experience replay keeps environment transitions and sample them proportionally to KL- divergence.

5. Differentiator / Contribution

- We have analyzed various deep reinforcement learning algorithm for the flappy bird agent
- For the D3QN algorithm, we have fine-tuned the hyperparameters and used the best performing parameter.

We have shared the hyperparameter tuning and comparison of various deep RL algorithms in the following sections

6. Methodology

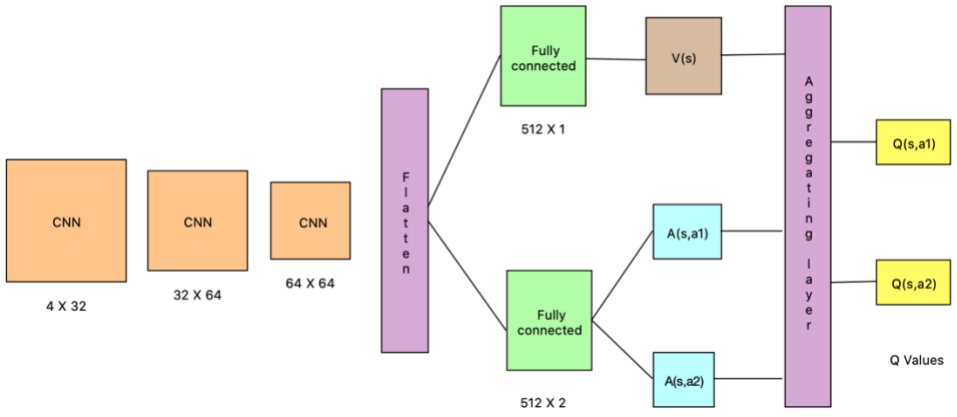


Figure 3: Dueling Deep Q-Network architecture for Flappy bird agent

Figure 3 represents the dueling deep Q-network architecture for flappy bird agent. As this agent has two actions, the second fully connected layer returns two advantage action values. In the aggregation layer, if we combine V and A values directly, there will be *issue of identifiability* during training. Given a value of Q(s, a), the system cannot determine V(s) and A(s, a) in the initial step of backpropagation. To avoid this problem, we force advantage function estimator to have zero advantage at a chosen action. Hence the formula for Q(s, a) during implementation will be

$$Q(s,a;\theta,\alpha',\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{\mathcal{A}}\sum_{a'} A(s,a';\theta,\alpha))$$

Table 1: D3QN Parameters

Layer	Size	Stride	# of output features
Conv 1	4 X 32 X 4	2	53792
Conv 2	32 X 64 X 3	2	25600
Conv 3	64 X 64 X 2	1	23104
FC value 1	512	-	512
FC advantage 1	1	-	1
FC value 2	512	-	512
FC advantage 2	2	-	2

The training is broken up into episodes, where an episode has the current state, reward, action and next state of the environment. During the episode the agent receives feedback from the environment in the form of depth images and direction to the goal, and it determines the action it needs to take to move toward the goal. At each training time-step, the agent accumulates a new experience for the experience replay buffer, and samples a minibatch of experiences from the buffer to train the network on. The buffer is rolling such that a fixed number of previous experiences are stored. The policy the agent follows is changing each time step during the episode. At every step, the mean reward is computed, and the experiment is repeated for the maximum mean reward of 1.2.

The training is conducted on Nvidia-GeForce GTX 100 machine with 1 GPU and 4 cores for multiple hyperparameter metrics evaluation. Each experiment took three hours for the network to converge. On an average, the experiment took 8000- 1000 episodes to converge. With the decay in epsilon value, the agent explores the environment with random actions in the initial steps.

6.1 Algorithm

Algorithm Double Dueling Q network with Experience Replay

Initialize replay memory D to capacity N

Initialize dueling Q-network with random weights θ

Initialize target dueling Q-network with weights $\theta^{-} = \theta$

for episode $= 1$, M **do**

$t = 0$

while s_t is not terminal **do**

With probability ϵ select a random action a_t

otherwise select $a_t = \text{argmax}_a \, Q(s_t, a; \theta)$

Execute action a_t and observe r_t and state s_{t+1}

Store transition (s_t, a_t, r_t, s_{t+1}) in D

Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D

if s_{j+1} is terminal **then**

$y_j = r_j$

else

$y_j = r_j + \gamma Q(s_{j+1}, \text{argmax}_{a'} \, Q(s_{j+1}, a'; \theta); \theta^{-})$

end if

Perform gradient descent on $\mathbb{E}_{(s_j, a_j, r_j, s_{j+1}) \sim \, U(D)}[(y_j - Q(s_j, a_j; \theta))^2] \; w.r.t. \; \theta$

$s_t \leftarrow s_{t+1}$

$t \leftarrow t + 1$

$\theta^{-} \leftarrow \theta$ every C steps

end while

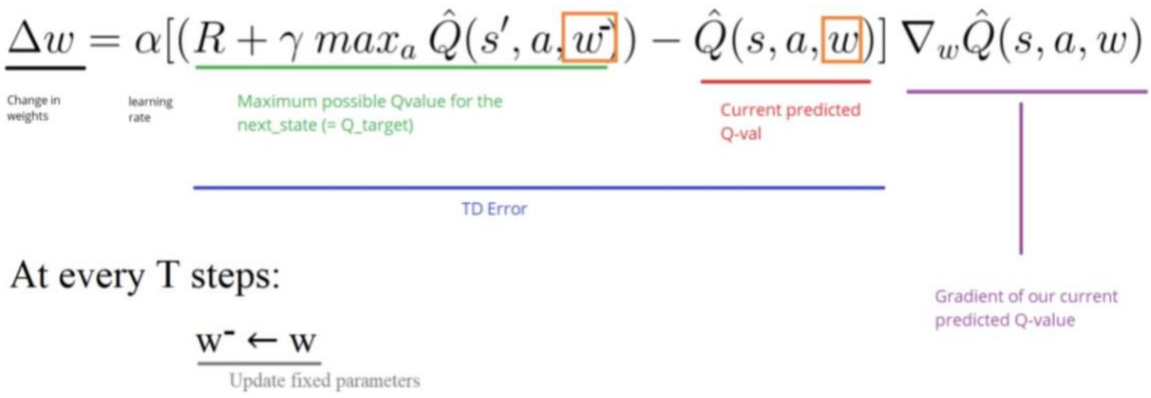
end for

Dueling Double Deep Q- Network has the following hyperparameters

Table 2: D3QN hyperparameters

Parameter	Description
Skip frame	An agent takes action at every k-frame instead of every frame
Gamma (γ)	Discount factor
Alpha (α)	Learning rate
Experience_buffer_size	Size of the experience replay buffer
Sync target frame (τ)	Synchronize source network to target network
Batch size	Neural network mini-batch size for backpropagation

Double Dueling DQN which has two estimators, the source network estimates the score of the current state and the target calculates the action score. Prioritized experience replay stores the reward, state, action values along with priority in memory. The memory size varies as per the experience buffer size value of the network, at every τ steps, the weight of the source and target networks synced in the training process. The temporal difference error computed as a difference between the Q- value from the current network and the maximum possible Q-value from the target network.



Prioritized experience replay computes the frequency of occurrence of the state-action pair in the provided state dimension. We sample the prioritized experience and match the experience from underlying distribution for training. Since the randomness is highly abundant, we have a chance of bias and overfitting. We use importance sampling (IS) to update the weights of often seen samples.

7. Implementation / Results

The following graphs from tensor board represent the hyperparameters verified, and the models are stored to validate for best performance.

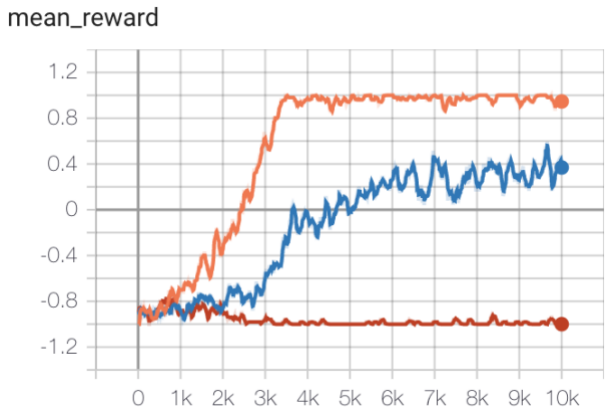


Figure 4: Learning rate hyperparameter for the value $1e^{-6}$, $1e^{-4}$,$1e^{-2}$

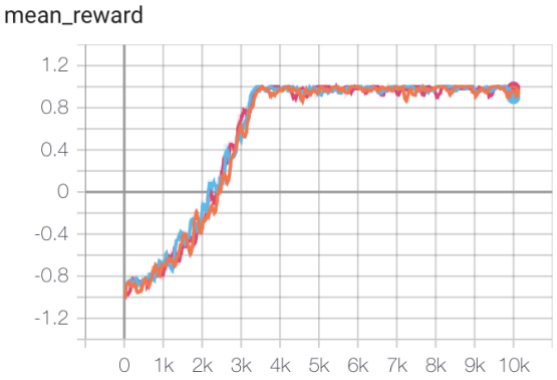


Figure 5: discount factor values 0.99,0.95,0.97

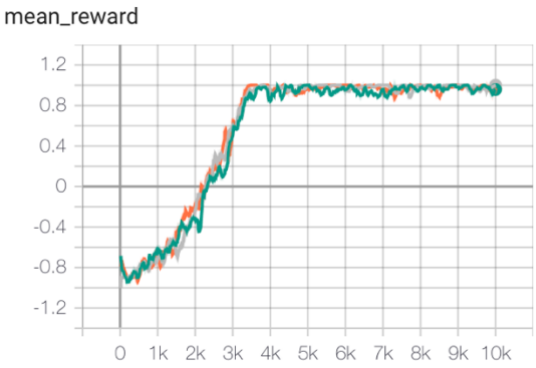


Figure 6: Sync target frame (τ) for the values 15,60,90

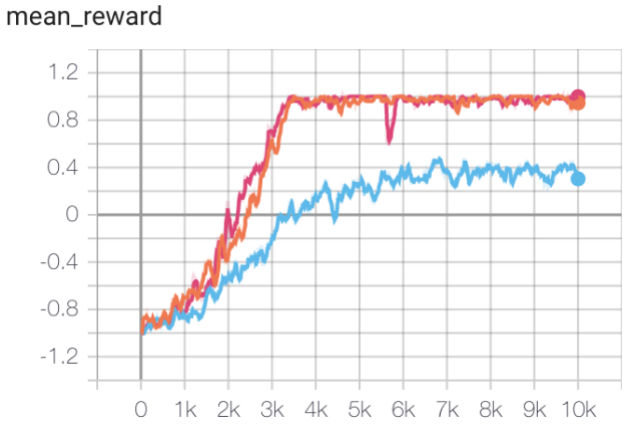


Figure 7: skip frame values 2,5,1

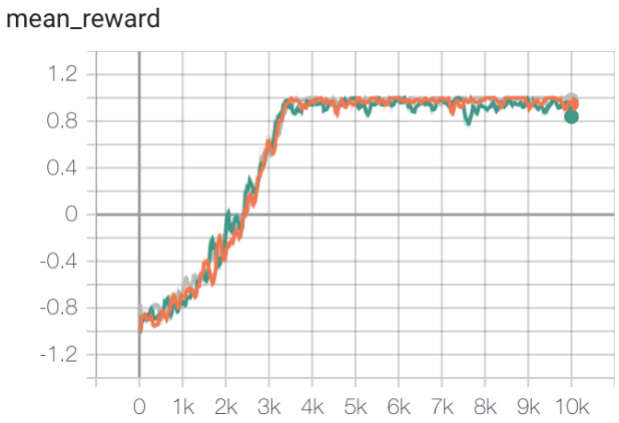


Figure 8: Experience buffer size values 200, 500, 8000

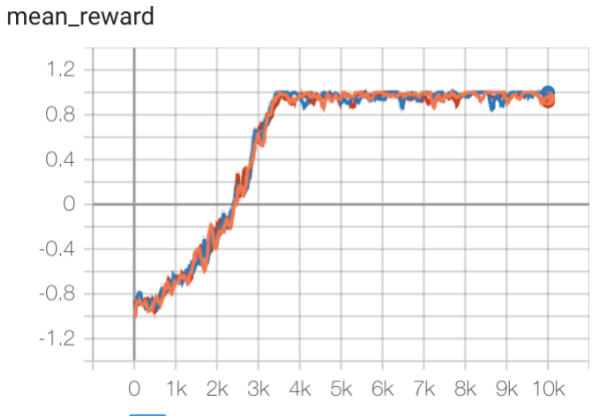


Figure 9: Batch size values 32, 16, 64

From the above graphs, we could find a significant impact with learning rate and skip frame hyperparameters. We have introduced a new metric to measure the performance by playing the game continuously for 3 minutes. In those three minutes metric, some of the parameters failed to play the game without failures. We have recorded the metrics in the below table. The column "attempts required in 3 minutes" represent the number of attempts the AI system requires to play the game.

Table 3: hyperparameter metric for infinity play

No	**Hyperparameter**	**Value**	**Attempts Required in 3mins**
1	Sync_target_frames	15	2
2	Sync_target_frames	60	0
3	Sync_target_frames	90	10
4	Discount factor	0.95	0
5	Discount factor	0.98	2
6	Experience_buffer_size	500	8
7	Experience_buffer_size	8000	3
8	Learning_rate	1e-6	0
9	Learning_rate	1e-2	0
10	Skip_frame	5	0
11	Skip_frame	1	4
12	Batch_size	16	2
13	Batch_size	64	2

7.1 Best Model:

Following table represents our hyperparameter values for our best performing model. We have chosen the best metric from the above experiment. The flappy bird is flying for infinite time for this model.

Table 4: Best model hyperparameter

Hyperparameter	**Value**
Experience_buffer_size	2000
State_dim	4
Actions	[0,1]
Discount factor	0.99
Epsilon decay frames	(10**4)/3
Mean_goal_reward	1.0
batch_size	32
Sync_target_frames	30
Learning_rate	$1e^{-4}$
Skip_frame	2

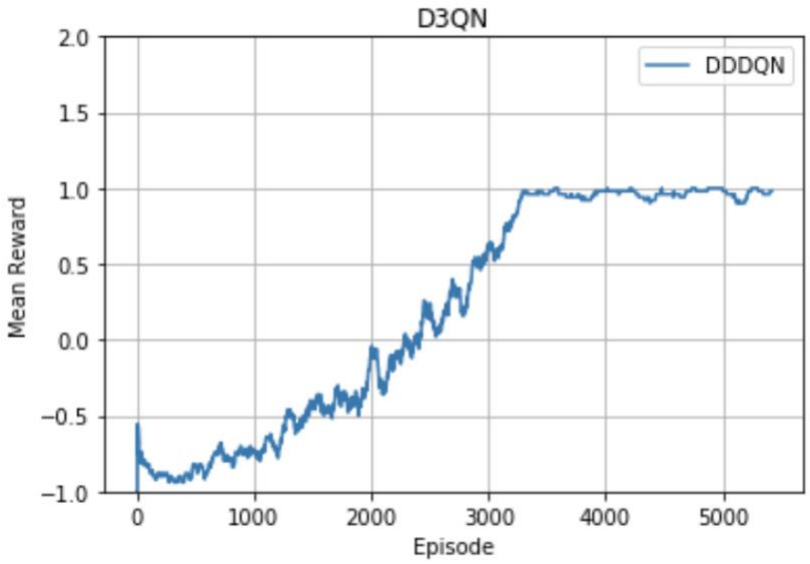


Figure 10: Mean reward plot for the best model

Figure 10 represents the mean reward for the best model. The model is verified to play continuously for more than 10 minutes without any failure. In the initial hyperparameter experiment on various values for each metric, doesn't provide much meaningful plot for all the values. Hence, we extended this duration to play and verified the model. We did a comparison study of the performance of this model with other policy-based reinforcement algorithms and DQN.

7.2 Comparison Study

We have experimented Flappy bird agent with other reinforcement algorithms such as Asynchronous Actor critic (A2C), Deep Q-network (DQN) and Proximal policy optimization (PPO) algorithms.

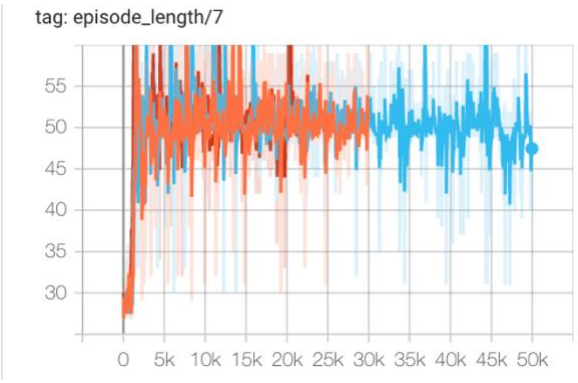


Figure 11: Q- value for 50k episodes from DQN, PPO and A2C

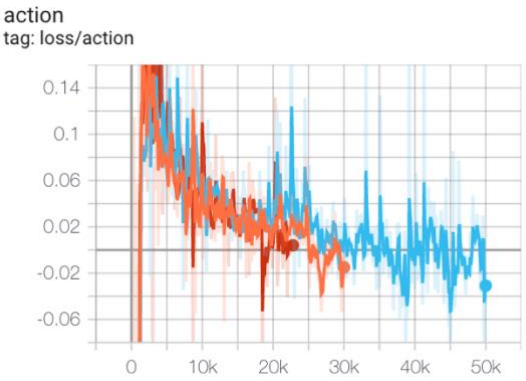


Figure 12: Loss values for 50k episodes from DQN, PPO and A2C

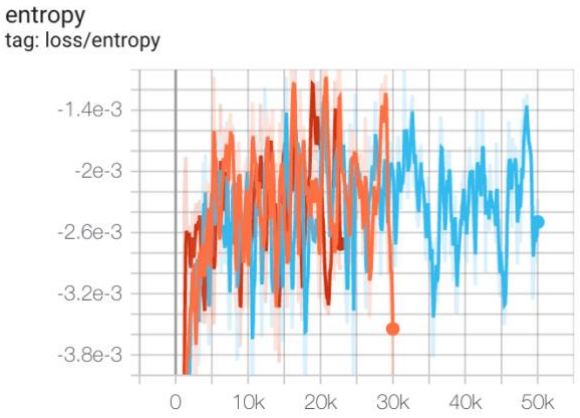


Figure 13: Entropy loss for episodes from DQN, PPO and A2C

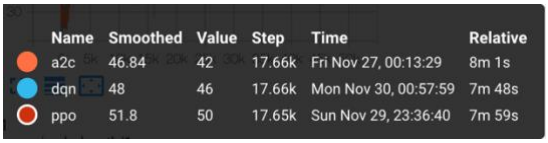


Figure 14: Legend /color code representation for A2C, DQN and PPO

From the above comparison study, we have noticed that policy-based algorithms are not converging even after 50k episodes. DQN performance was lacking compared to all the other algorithms.

8. Contribution

Name	Id	Contribution
Pranav Lodha	009468121	• Literature study on various DQN • Understanding prioritized experience replay • Inference pipeline with the best model • Experiments with hyperparameters for learning rate and experience buffer size
Subarna Chowdhury Soma	014549587	• Literature study on D3QN • Exploring various research works in D3QN • Study of the hyperparameters • Experiments with the hyperparameters for discount factor, sync target frame, batch size and skip frame
Jeyasri Subramanian	014510132	• Exploring various DQN architecture and design architecture • D3QN environment setup • Tensor board setup for hyperparameter experiments • A comparison study with DQN, A2C and PPO
Team		• Report • Presentation • GitHub

9. Conclusion

We explored the application of deep reinforcement algorithms to the problems of agent navigation in a minimal action space environment like Flappy bird game. The application of deep Q-learning is capable of determining best action from high-dimensional inputs such as RGB image from the game environment. We have experimented various hyperparameter in D3QN neural network architecture and propose the best value for playing the game without failure. The prioritized experience buffer helps to reduce RL-divergence and converges this network in less than 5k episodes.

We also included a comparison study of various policy-based algorithms to conclude how D3QN is best suitable for this problem. From this work, D3QN is capable of flying flappy bird agent in an infinite space, and the proposed hyperparameters are the best values to converge network within a short time. The ability of theD3QN architecture to learn navigation policies and generalize to new environments suggests that the future of robot navigation in cluttered environments is promising.

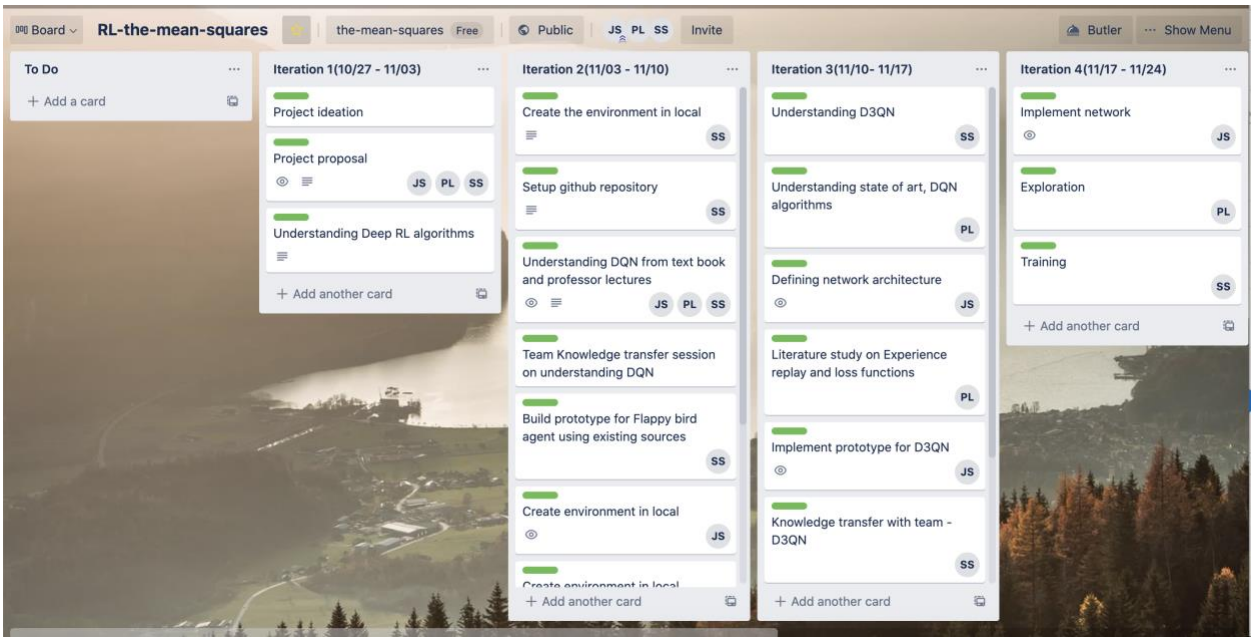
10. Appendix

10.1 GitHub

GitHub: https://github.com/s-c-soma/RL_Project_FlappyBird_DQN_DDPG

10.2 Trello

Trello: https://trello.com/b/8CrK9MHe/rl-the-mean-squares



10.3 Reference

1. Kk. (2020, April 04). Using Dueling DQN to Play Flappy Bird. Retrieved November 28, 2020, from https://www.fromkk.com/posts/using-ddqn-to-play-flappy-bird/
2. Xu, T. (2020, January 22). Use reinforcement learning to train a flappy bird NEVER to die. Retrieved November 28, 2020, from https://towardsdatascience.com/use-reinforcement-learning-to-train-a-flappy-bird-never-to-die-35b9625aaecc
3. Gite, S. (2017, April 11). Practical Reinforcement Learning - 02 Getting started with Q-learning. Retrieved November 28, 2020, from https://towardsdatascience.com/practical-reinforcement-learning-02-getting-started-with-q-learning-582f63e4acd9