Machine Learning Nano Degree Capstone Project: Deep Reinforcement Learning to train an agent to play Atari Pong game using DQN

Mohan Muppidi

mkumar2301@gmail.com

1 Definition

1.1 Overview

Reinforcement learning is one of the fascinating machine learning techniques which primarily focuses on playing games. There have been a lot of theoretical models proposed in the past, but due to recent advancements in computation resources and deep learning techniques, implementing reinforcement learning has become practical. Google's Deep Mind has demonstrated the potential of reinforcement learning by teaching an agent to play atari games and later they went on to defeat world's best Go players.

This project explores a deep reinforcement learning technique to train an agent to play atari pong game from OpenAI Gym. OpenAI Gym is a toolkit to develop and compare reinforcement learning algorithms. The learning agent takes raw pixels from the atari emulator and predicts an action that is fed back into the emulator via OpenAI interface. The deep reinforcement learning network used in this project is Deep Q Network (DQN), it took over 10 million episodes to perfectly play and win the game.

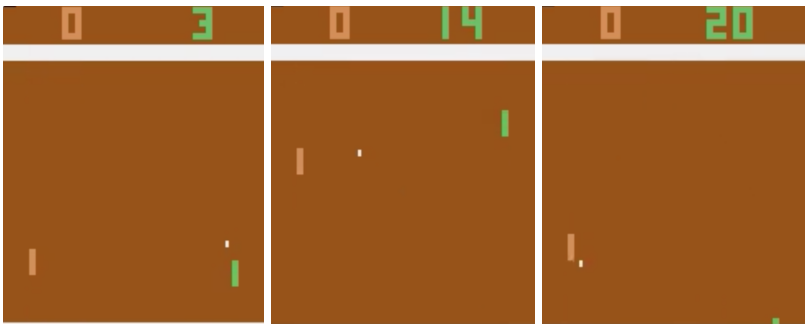


Fig. 1: Snapshots of Atari Pong game

1.2 Problem Statement

Atari pong is a simple "tennis like" game features two paddles and a ball, the goal is to defeat your opponent by being the first one to gain 20 points, a player gets a point once the opponent misses a ball [5]. One player is controlled by the traditional AI which is baked into the game itself. Usually the second player is a human, but in this project human is replaced by a deep reinforcement learning agent.

The idea is to build a deep network that can take pixel data as input and predict actions that will eventually win the game for the network. The final end goal is to win the game against traditional AI by 20-0. The deep reinforcement learning network used in this project is Deep Q Network (DQN).

1.3 Metrics

Average Reward In Atari pong game at the end of every episode the agent can receive a reward between -21 to 21 based on its performance. The best way to measure the performance of the agent is to calculate the average of all the reward it received across the episodes. In the results we use 500 episodes to determine the average reward

Best 100-episode average reward This is similar to the Average reward but instead of taking a mean of all the rewards we take rolling means [10] with length of 100 and treat the maximum among those means as a measure.

Average episode length for winning episodes Another way to measure agents performance is by evaluating the average number of steps taken by the agent to win an episode. Better agent would win a game with in a minimum number of steps. Similarly a worser agent can lose a game with in a minimum number of steps.

2 Analysis

2.1 Data Exploration and Visualization

OpenAI Gym wraps an emulator and provides us with a nice interface to extract current state, provide a action and receive a reward for the action. Different games provide different types of state information, some provide images, some provide list of numbers, some provide binary data etc. Atari pong provides images as states.

The image we extract from the Atari Pong environment is a RGB image. The RGB color model is an additive color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors [6].An RGB image is a three-dimensional byte array that explicitly stores a color

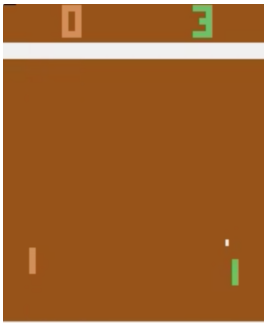


Fig. 2: Sample of data extracted from the emulator.

value for each pixel. RGB image arrays are made up of width, height, and three channels of color (red, green, blue) information [7].

In supervised learning implementations the training data has input and target output. While training generally some sort of error is calculated based on the difference between the predicted output and target output. That error is then used for training the network. In reinforcement learning implementations the network receives a reward from the environment for every action it predicted and the network uses the reward for training. Here is how the rewards received from the game would look like.

Reward	Cause
-1	Action resulted in user's paddle missing the ball
0	Action resulted in user's paddle hiting the ball
1	Action resulted in opponent's paddle missing the ball

Table 1: Rewards and their reasons

An environment in reinforcement learning also has an action space, that is the different actions that are accepted by the environment. Atari pong environment accepts 6 actions, here are the definitions for each action

Action	Description
0	Move up with maximum speed.
1	Move up with medium speed.
2	Move up with minimum speed.
3	Move down with minimum speed.
4	Move down with medium speed.
5	Move down with maximum speed.

Table 2: Actions and their definitions

2.2 Algorithms and Techniques

Reinforcement Learning [8]: The reinforcement learning problem is meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision-maker is called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations to the agent. These situations are called as states. The environment also gives rise to rewards, special numerical values that the agent tries to maximize over time. A complete specification of an environment defines a task, one instance of the reinforcement learning problem. Fig. 3 depicts the reinforcement learning framework.

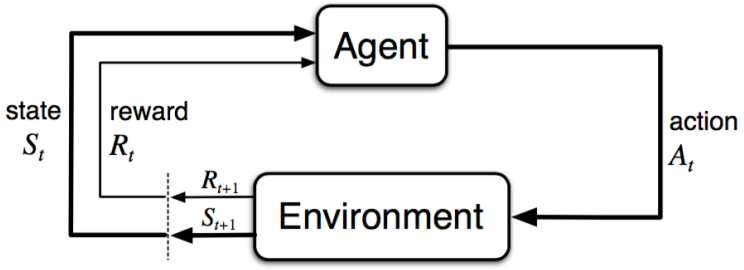


Fig. 3: The agent-environment interaction in reinforcement learning [8].

An agent is a learner and decision maker. In this project 3 agents are presented, a random agent and 2 DQN agents with replay memory. First DQN agent is trained around for 500,000 episodes and second DQN agent is trained for 1,000,000 episodes.

Random Agent: Random agent chooses an action randomly from the action space at each time step. This cannot solve the problem but it establishes a baseline.

DQN with replay memory:

Q-Learning: Q-Learning is a basic Reinforcement learning techinique. It is simple and easy to implement, especially if our states are discrete and the state space is small. To start with the agent provides random actions as input to the environment and bulids a table which eventually helps it in making the right decisions. The table is called as a Q table. Each row in the table has information about a unique state and its corresponding action Q-value pairs, for all possible

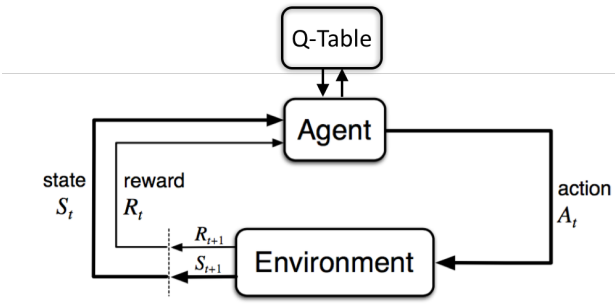


Fig. 4: The agent-environment interaction in Q-learning.

actions when the vehicle is in that state. The Q values are values that are learned by the agent from the feedback provided by the environment. The agent first looks at all the action Q value pairs for that state, then chooses the action with highest Q value and sends it to the vehicle. The vehicle then sends back a reward and next state so that the agent can update the Q value for the action in the table. The equation for updating the Q value is given by

$$Q(s,a) = R(s,a) + \gamma(\max_{a'} Q(s',a')) \tag{1}$$

The Q value learning rule or the Q-Learning rule is give by :

$$Q(s,a) := Q(s,a) + \alpha(R(s,a) + \gamma(\max_{a'} Q(s',a')) - Q(s,a)) \tag{2}$$

Where Q is Q function, R is reward function, α is Learning rate, γ is Discount factor, s is current state, s' is future state, a is current action, a' is future action.

Deep Q-Network: In Q-Learning the intelligence acquired by the agent is stored in the form of a table. This approach is not scalable and is only good for learning environment which has smaller state space. To make it scalable a neural network or a deep network can be used to replace the table.

Replay Memory: In practice, DQN algorithm stores the last N experience tuples in the replay memory, and samples uniformly at random from the memory when performing updates. This approach is in some respects limited because the memory buffer does not differentiate important transitions and always overwrites with recent transitions owing to the finite memory size N. Similarly, the uniform sampling gives equal importance to all transitions in the replay memory.A more sophisticated sampling strategy might emphasize transitions from which we can learn the most, similar to prioritized sweeping.[1]

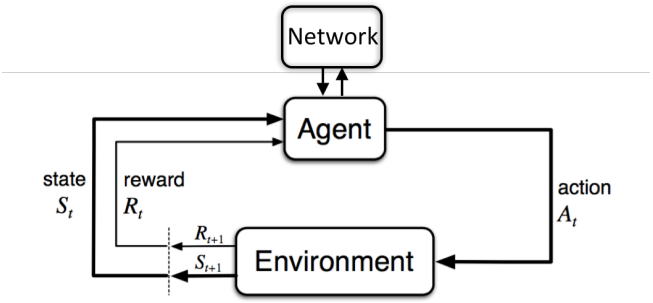


Fig. 5: The agent-environment interaction in Q-Network.

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Fig. 6: Algorithm for DQN with Replay memory [1].

2.3 Benchmark

The benchmark for this project would be achieving the maximum reward possible. In case of Atari pong the maximum reward we can achieve is 21 per episode. Deepmind's DQN paper [1] has shown that they have achieved an average reward of 18.9. I am currently using Open AI Gym PongNoFrameskip-v3 environment which doesn't have a public score board, but Open AI Gym Pong-V0 environment has many scores above 20 in the score board [9].

3 Methodology

3.1 Data Preprocessing

As mentioned in Section-2, the state we receive from the environment is a RGB image with size 210x160x3. Fig-7(a) shows an example RGB image received from the environment. Reducing the input data while preserving the features will help in building efficient models which will run faster and yield results. Converting a RGB image to a Gray scale image reduces the number of data points by 3 times while preserving the features in the image. Gray scale conversion can be achieved in many ways [11], in this project we use the following formula for the conversion.

$$Gray = (Red * 0.299 + Green * 0.587 + Blue * 0.114) \tag{3}$$

Fig-7(b) shows the Gray scale image achieved by applying the formula-3 on Fig-7(a). To reduce the size further the image resized using OpenCV resize function. The resized image of Fig-7(b) is Fig-7(c). The final size of the image is 84x84x1. These images are added to the replay memory as mentioned in section-2.2.

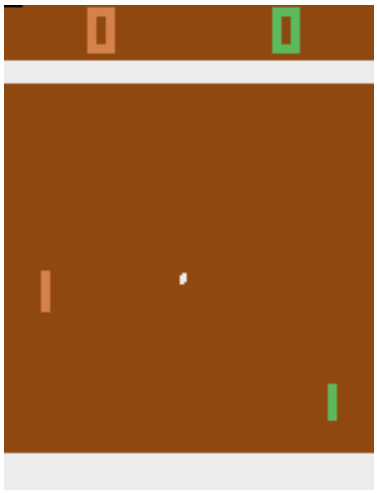
As mentioned in section-2.2 the images are processed and added to a replay memory. At every step the input to the deep network is 4 recent entries of the replay memory stacked on top of each other as shown in Fig-7(d)

3.2 Implementation and Refinement

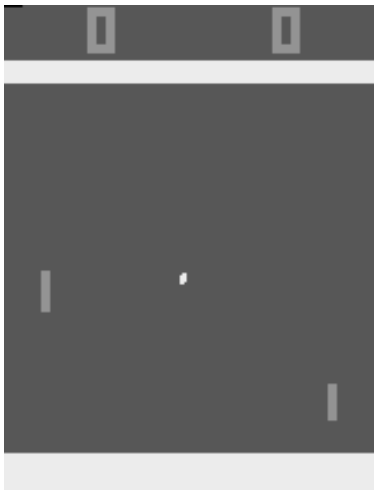
For the implementation of the agent I have followed Deepmind's DQN paper [1], lectures and homework from CS 294: Deep Reinforcement Learning course [3]. I have obtained the stater code from the homework 3 of [3].

The agent is built using the Tensorflow library. The agents network is a CNN and is trained using the ADAM optimizer. The agent is trained for 9.5 million steps. While training, snapshot of the agent is saved at 5260000, 8090000 and 9500000 timesteps. The performance of all 3 are presented in results section. Table. 3 shows the list of hyper parameters used for training the agent.

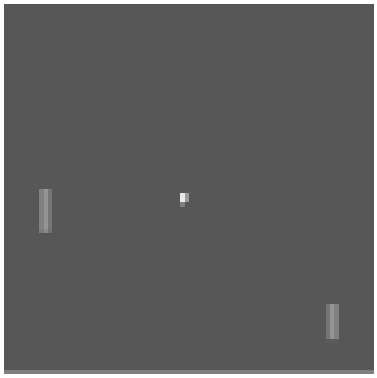
Fig. 8-12 show different parameters behaved throughtout the learning period. Fig. 8 shows how the average reward gained by the agent during the past 100 episode varied between 0 and 10 million steps. Similarly Fig. 9 shows best average



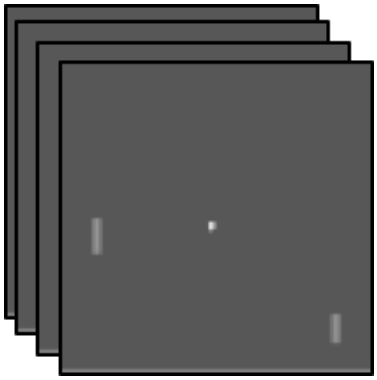
(a) Original Image



(b) Grayscale version of original



(c) Resized and cropped version of grayscale image



(d) 4 recent images of type (c) stacked on top of each other

Fig. 7: Outputs after each preprocessing step

Parameter	Value	Definition
batch size	32	How many transitions to sample each time experience is replayed.
replay memory size	1000000	How many memories to store in the replay memory
agent history length	4	How many past frames to include as input to the model.
target update frequency	10000	How many experience replay rounds to perform between each update to the target Q network
discount factor	0.99	discount factory gamma used Q-learning update
exploration	A linear scheduler	schedule for probability of chosing random action.
learning starts	50000	Step at which the learning starts
learning frequency	4	How many steps of environment to take between every experience replay

Table 3: List of hyper-parameters and their values

reward gained by the agent during the past 100 episode against the training steps. Fig. 10 shows how the exploration factor has been changed during the training. Fig. 11 shows how the learning rate has been changed during the training.

During training the exploration factor is varied in such a way that, the value is high initially, so that the agent can explore more of the environment and then it is reduced gradually. Exploration factor determines the probability at which actual agent's decision is used to control the environment. As the exploration factor decreases the actual decisions from the agent are fed into the environment more frequently. This influence of the exploration factor on performance of the agent can clearly seen from Fig. 8 and Fig. 10. Initially when exploration factor is above 0.1 the performance of the agent is bad, but once the exploration factor value is less than 0.1, the agent started to perform well.

High learning rate ensures high exploration and low learning rate ensures convergence. So the learning rate is initially high and then it is slowly reduced to reach a smaller value. Fig. 12 shows that around 4000 episodes where run during the 10 million steps of training. It can be observed that the rate at which the episodes are completed was initially high, this is because the agent is not trained well yet and also the exploration factor is high, this means the agent was sending a lot of random inputs to the environment. The environments AI can easily win when proper decisions are not being made by the agent. Then the rate decreases as the agent is getting better and so playing more timesteps before the environments AI or agent win the episode. Finally the rate increases again as agent is now better than environment AI and is capable of winning an episode quickly.

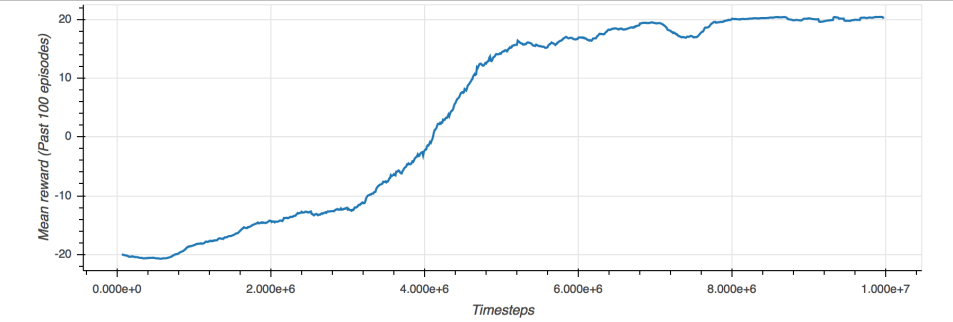


Fig. 8: Average reward gained by the agent during the past 100 episode.

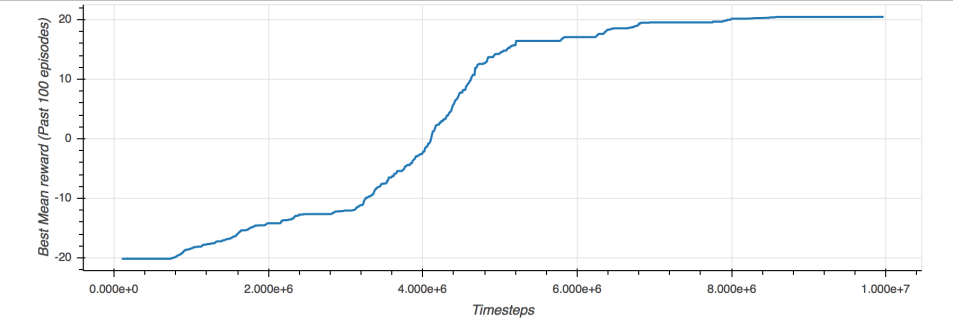


Fig. 9: Best average reward gained by the agent during the past 100 episode.

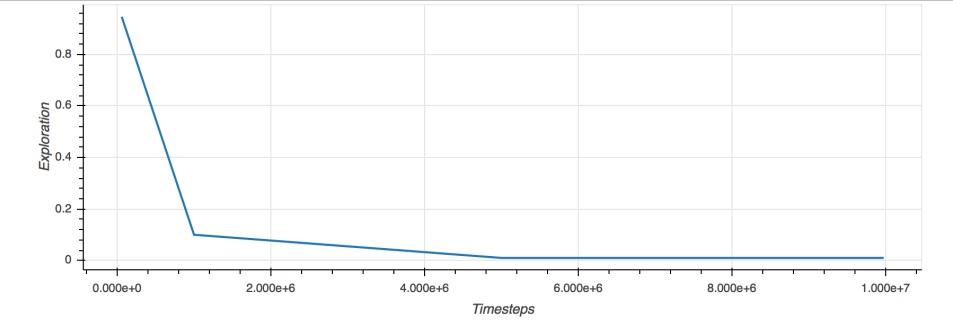


Fig. 10: Exploration factor value throughout the training.

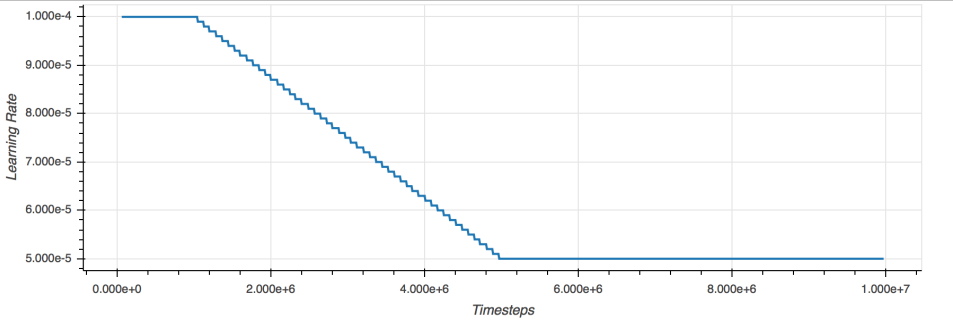


Fig. 11: Learning rate value throughout the training.

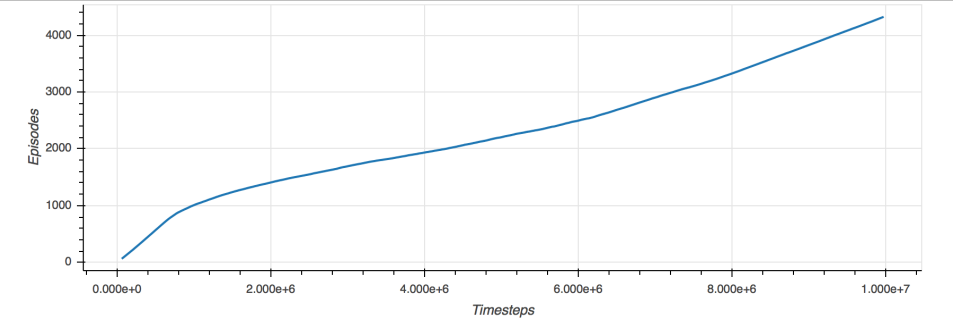


Fig. 12: Number of episodes vs timesteps

4 Results

In this section performance of above mentioned agents are compared with each other. In order to compare the agents I ran each algorithm on the pong environment for 500 episodes. Graph in Fig. 13 shows the total reward received in each episode. As expected the random agent received the least total reward per episode. The DQN trained for 9.5 million time step received the highest reward. It can also be observed that as the number of timesteps per training increased the performance of agent increased.

Fig. 13: Episode number vs Total reward gained during the episode

Graph in Fig. 13 shows the Average reward gained by the agent during the past 100 episode. This graph is less noisy when compared to Fig. 13 and shows the clear distinction between performance of different agents.

Fig. 14: Episode number vs Average reward gained during the past 100 episode

Table. 4 shows average reward and average episode length. From the table it can be observed that on an average the random agent takes less number of timesteps to complete an episode. This is because the AI of the Atari pong game

is capable of winning the game against random agent quickly. It can also be observed that as the DQN is trained for more steps the average timesteps to complete an episode decreased.

Agent	Average reward	Best rolling 100-episode average reward	Average episode length
Random agent	-20.33	-19.00	3691.13
Agent after 5260000 steps of training	12.04	14	14512.00
Agent after 8090000 steps of training	20.08	20.5	8716.08
Agent after 9500000 steps of training	21.00	21.00	7432.27

Table 4: Agents performance comparision

In an Atari Pong game the maximum reward a player can achieve is 21 and the minimum reward a played can achieve is -21. The random agent, I ran to establish the base line received an average reward of -21. The DQN agent I trained for 9.5 million timesteps has achieved the maximum reward possible.

5 Conclusion

In this project I wanted to train a reinforcement learning agent which can play Atari pong game and win against the default game AI. To establish a baseline I first ran the game with a random agent, as expected the average reward gained by the agent is -21, which is the minimum reward possible. Then I trained a DQN agent with a replay memory for 9.5 million timestepss. While training I have saved the snapshot of the agents network at 5260000, 8090000 and 9500000 timesteps. In the results section I have presented how these 3 networks performed over 500 episodes. The agent trained for less number of steps on an average gained less reward and took more timesteps to finish the game. The agent trained for 9.5 million timesteps on an avergae gained a reward of 21, which is the maximum possible reward for this environment and it took less number of steps to finish than the other two.

At the end of the project I achieved what I was hoping at the beginning. I trained an agent which is capable of winning against the default Atari Pong AI. In future I would like to investigate and train other types of reinforcement learning agents like policy gradinents and Double DQN.

References

1. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S.,

2015. Human-level control through deep reinforcement learning. Nature, 518(7540), pp.529-533.
2. Andrej Karpathy, http://karpathy.github.io/2016/05/31/rl/
3. CS 294: Deep Reinforcement Learning, Spring 2017, http://rll.berkeley.edu/deeprlcourse/
4. Udacity MLND Reinforcement Learning course, Charles Isbell and Michael Littman. https://www.udacity.com/course/reinforcement-learning--ud600
5. Pong Game http://www.ponggame.org/
6. RGB color model Wikipedia https://en.wikipedia.org/wiki/RGB_color_model
7. RGB Image Organization http://northstar-www.dartmouth.edu/doc/idl/html_6.2/Indexed_and_RGB_Image_Organization.html
8. Sutton, Richard S. and Barto, Andrew G., 1998, ISBN: 0262193981, Introduction to Reinforcement Learning,
9. https://gym.openai.com/envs/Pong-v0
10. https://en.wikipedia.org/wiki/Moving_average
11. http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/