Atari Pong

50.021 Artificial Intelligence Project

Clara Ho, Dhriti Wasan, Justinian Siah, Shruti Sriram

Introduction

Pong is a two-dimensional game that features two paddles and a ball. The ball is hit back and forth by the two players with the paddles until one player misses. Points are awarded to a player when the opponent misses the ball. The goal is to defeat your opponent by being the first one to get 10 points (for open AI gym, this 21 instead).

The open ai gym features the Atari 2600 version of pong, simulated through the Arcade Learning Environment. Figure 1 depicts an untrained agent (right, green) going against the internal AI model (left, orange).

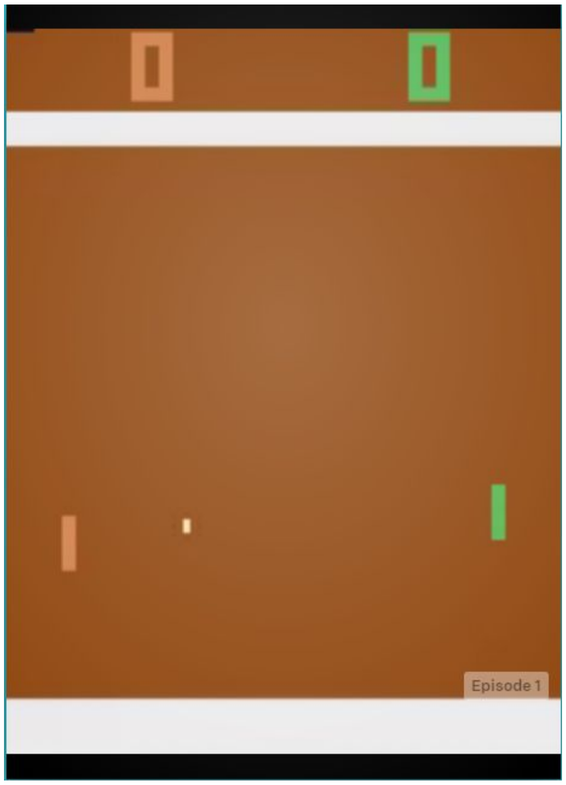


Figure 1: Frame of one game in Pong

The goal of this project is to achieve a positive average score. A positive score implies that the agent has 'won' the game—i.e. Has achieved a score of 21 points. The average score will be calculated by the moving average over the last 100 episodes and this is done to ensure that the agent's score is consistent.

Environments

The various Pong environments used in this project are:

1. **Pong-ram-v0**
   This environment provides observations of the state in the form of an array of bytes of length 128. Actions are performed for a duration of either 2, 3 or 4 frames where the number of frames are sampled randomly with equal probabilities.

2. **Pong-v0**
   This environment provides observations in the form of an RGB image of the screen. This is an array in the shape of (210, 160, 3). Similar to the RAM environment, each action is performed for 2, 3 or 4 frames which are randomly sampled.

3. **PongNoFrameskip-v4**
   This is an extension of Pong-v0. This environment also provides observations in the form of an RGB image of the screen but ensures that all frames of the environment are considered. This is different from Pong-v0 as it provides all frames for evaluation.

Learning Methods

This section highlights the various learning methodologies that were implemented to train the agent.

1. Baseline Model

A supervised learning model is used as a baseline model whose performance is later compared with the reinforcement learning models.
A simple neural network of 2 fully connected (FC) layers and Rectified Linear Unit (ReLU) and sigmoid activation functions) was used to predict the probability of actions depending on the input observation. The first hidden layer consists of 100 neurons and finally the second hidden layer (output layer) consists of a single neuron determining the probability of the action. This is depicted in figure 2.

The equation used for the final neuron is :

$$\nabla_w log_p(y = UP | x)$$

The action probability is then used for a biased coin toss in order to determine the action the agent will take.

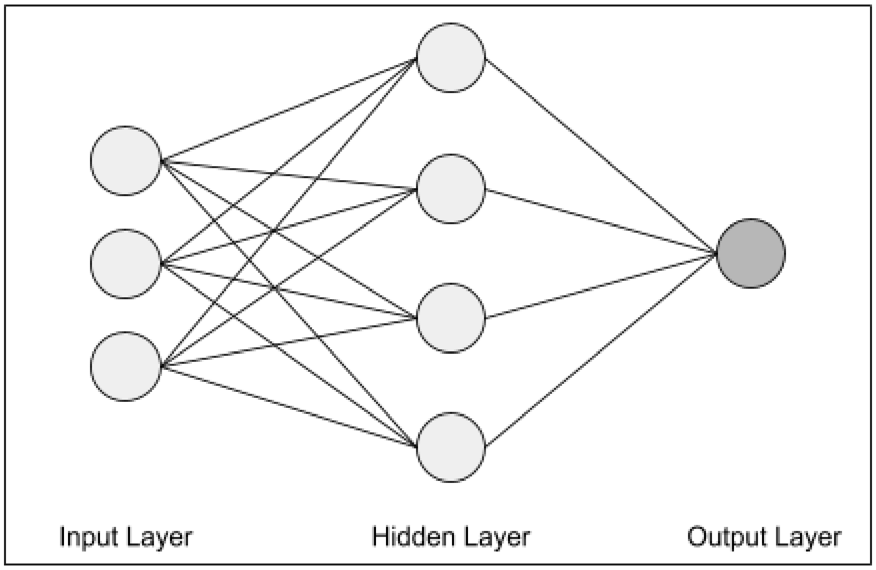


Figure 2: Structure of the baseline model

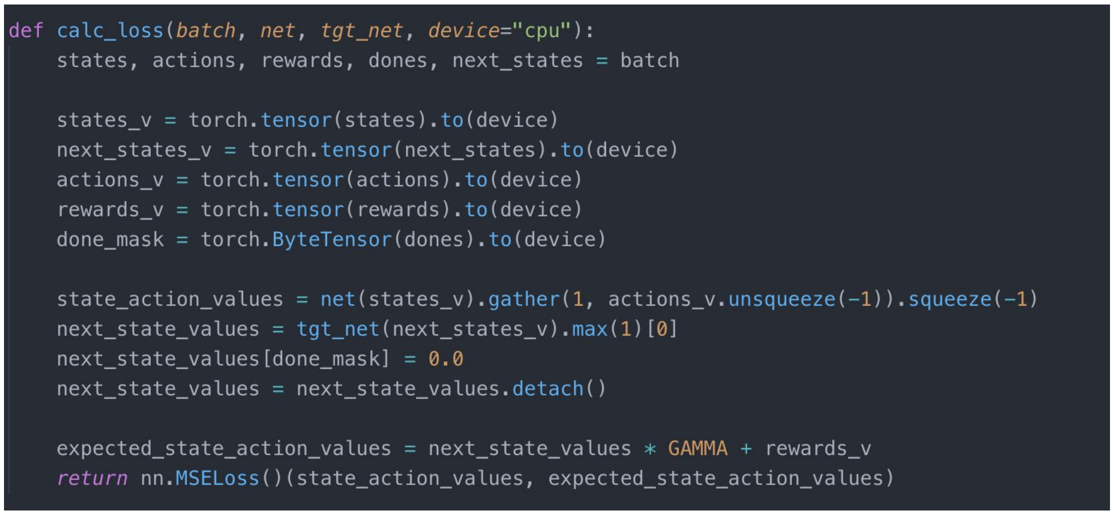
2. Deep Q-Learning Networks (DQN)

In deep Q-learning, a neural network is used to approximate the Q-value function from traditional Q-learning. The network we used has 3 convolution layers and 2 fully connected layers that are separated by ReLU nonlinearities. The output of the model are the Q-values for every action.

For the loss calculation,

$$L = (Q_{s,a} - r)^2$$

The target network uses Bellman approximation to calculate the next state values and this is compared (using MSELoss) with the training model to obtain gradients.



```python
def calc_loss(batch, net, tgt_net, device="cpu"):
    states, actions, rewards, dones, next_states = batch

    states_v = torch.tensor(states).to(device)
    next_states_v = torch.tensor(next_states).to(device)
    actions_v = torch.tensor(actions).to(device)
    rewards_v = torch.tensor(rewards).to(device)
    done_mask = torch.ByteTensor(dones).to(device)

    state_action_values = net(states_v).gather(1, actions_v.unsqueeze(-1)).squeeze(-1)
    next_state_values = tgt_net(next_states_v).max(1)[0]
    next_state_values[done_mask] = 0.0
    next_state_values = next_state_values.detach()

    expected_state_action_values = next_state_values * GAMMA + rewards_v
    return nn.MSELoss()(state_action_values, expected_state_action_values)
```

Figure 3: The function used to calculate loss

3. Advantage Actor Critic (A2C)

A2C is a variant of the policy iteration method of reinforcement learning. Policy iteration methods were also used in the Cartpole and Lunar Lander homework which implemented REINFORCE with baseline. The code used for the homework, along with some additional modifications, is used to test on the various Pong environments.

The A2C models consist of 2 separate neural networks which output a probability distribution and a value. Figure 4 gives a visual representation of the neural network.

- Common Layer: Fully-Connected Layer with 100 neurons with ReLU as the activation function.
- Actor : 6 output values, Activation function - Softmax
- Critic: 1 output value

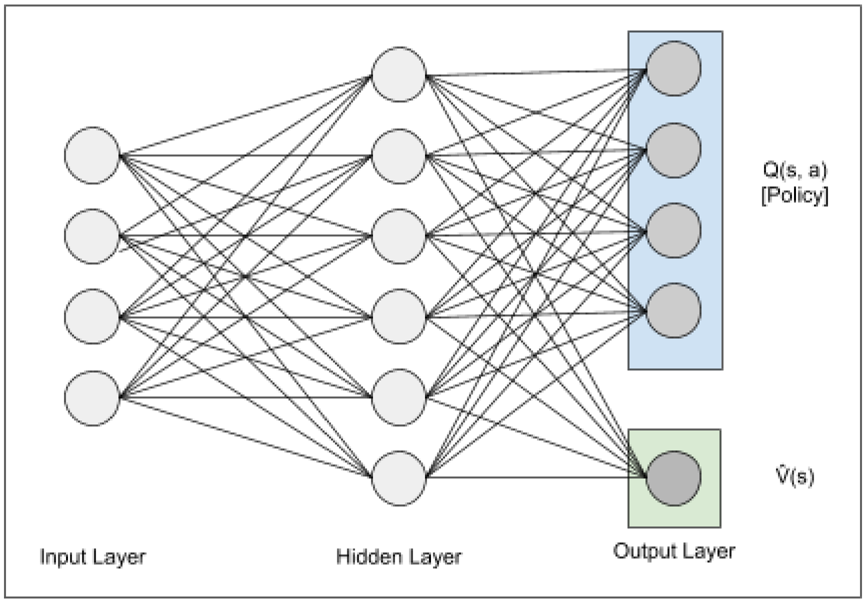


Figure 4: A2C Neural Network (self drawn)

In addition to the neural network the following parameters were changed and tested accordingly:

1. **Loss functions**

   The loss function consists of 2 components, one corresponding to the actor network and the other to the critic network. Based on the policy gradient method, the model can be classified as REINFORCE, Q-Actor Critic or Advantage Actor Critic (A2C) as shown below [5].

$$\nabla_\theta J = E_{\pi_\theta}[\nabla_\theta log\pi_\theta(s,a)v_t] \ (REINFORCE)$$
$$\nabla_\theta J = E_{\pi_\theta}[\nabla_\theta log\pi_\theta(s,a)\ Q^w(s,a)] \ (Q\ Actor - Critic)$$
$$\nabla_\theta J = E_{\pi_\theta}[\nabla_\theta log\pi_\theta(s,a)\ A^w(s,a)] \ (Advantage\ Actor - Critic)$$

   The Critic loss can be a simple L1 loss or Huber loss.
   The loss function is given by $Loss = Actor\ loss + Critic\ Loss$.

2. **Optimizer and learning rate:** The learning rate parameter of the optimizer, RMSProp or Adam, which determines the speed of convergence.

3. **Exploration Method:**

   a. **Entropy:** Entropy is given by the following formula:

   $$H(\pi_\theta) = -\Sigma\pi_\theta log(\pi_\theta)$$

   Adding an entropy term, multiplied with the hyperparameter **alpha** to the loss function encourages exploration [4]. The loss function is then given by:
   $Loss = Actor\ loss + Critic\ Loss + \alpha H(\pi_\theta)$ where $\alpha H(\pi_\theta)$ is the entropy term.

   b. **E-greedy:** Alternatively, the epsilon greedy method can also be used to select actions.

Results

RAM

Baseline model

The baseline model for RAM performed poorly, with a stable average score of -20 as seen in Figure 5.

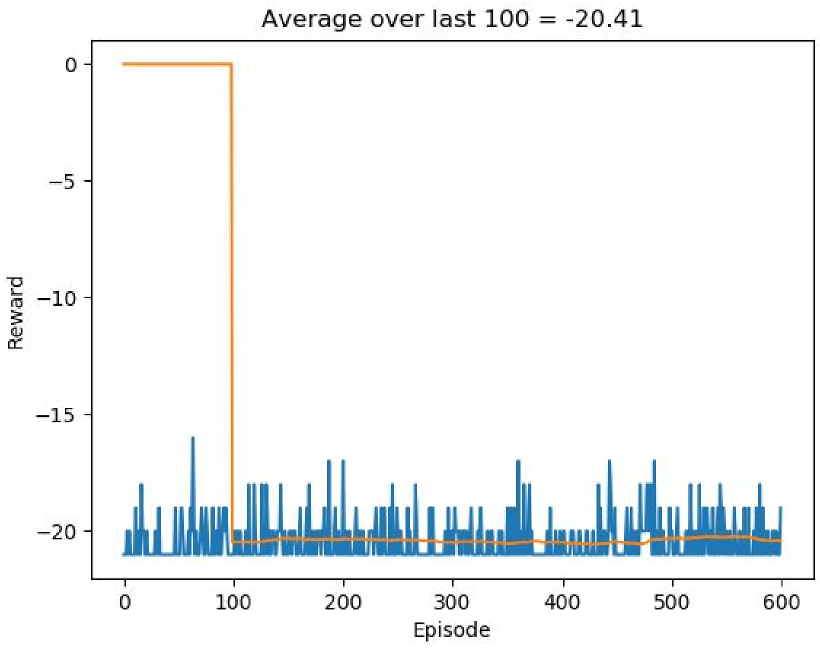


Figure 5: Performance of baseline model in Pong-ram-v0

A2C model

The A2C model performed better than the baseline with a stable reward of -15 after 600 episodes . While this is an improvement over the baseline, the agent plateaued past 1000 episodes as seen in figure 6.

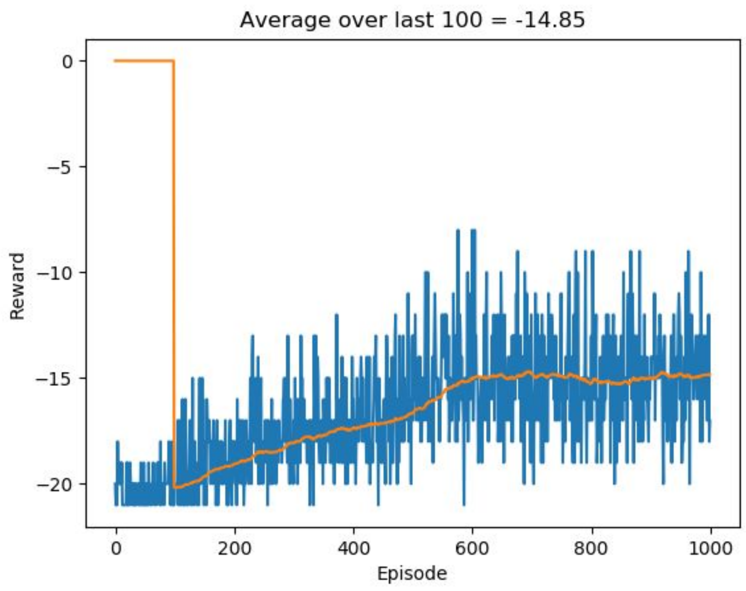


Figure 6: Performance of A2C model in Pong-ram-v0

DQN

Similar to the baseline model, the DQN model did not perform very well as seen in figure 7, showing no increase in mean rewards past -20.6 even after 600 episodes.

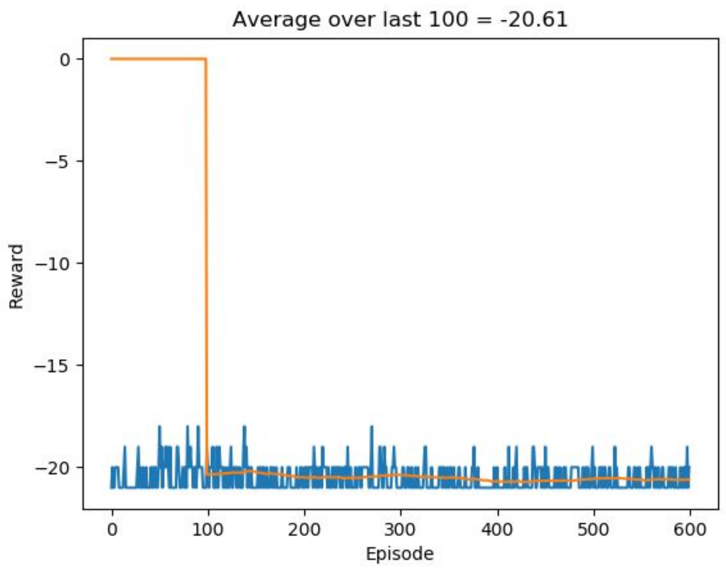


Figure 7: Performance of DQN model in Pong-ram-v0

Images

Preprocessing

Before the training, some preprocessing was done on the image observation of the environment. This is mainly to crop and resize the images to reduce the number of input dimensions into the neural networks. The two types of pre-processing used are listed as follows:

- Method 1: Vector Conversion
  This method reduces the input size from 210*160*3 to a vector of size 6400 (80*80 grayscale). It involves cropping, downsampling and erasing the background layers [3]. This method was applied on the baseline A2C agents as they did not have convolution layers.

- Method 2: Wrappers
  This method uses the wrappers obtained from the OpenAi repository to reduces the input from 210*160*3 image to a 84*84 grayscale image. The wrappers and their description is found in Appendix A. This method was applied to the DQN agents.

Baseline

The baseline model for Images performed poorly, with a stable average score of -20 as seen in Figure 5.

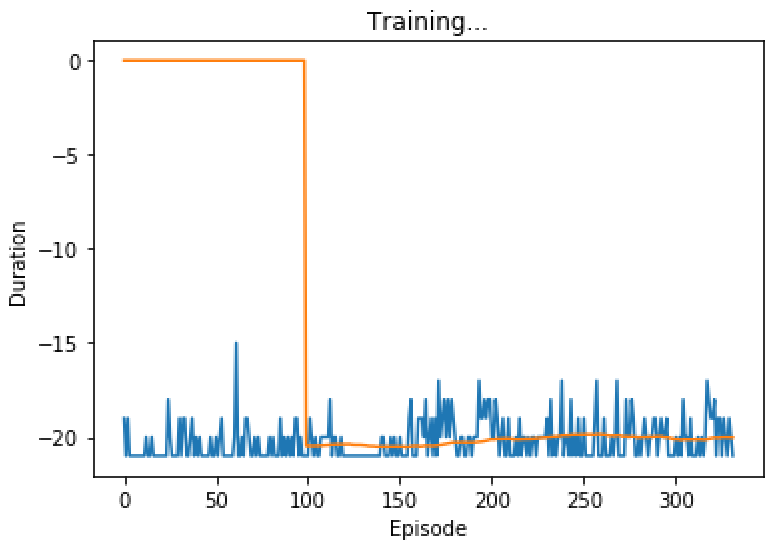


Figure 8: Performance of Baseline model on Pong-v0

A2C

The A2C image model was trained with different combinations of parameters. The maximum mean reward of some of these models is listed below in Table 1.

	Actor Loss, Critic loss	Exploration	Learning rate	Mean reward
Model 1	Q, L1 loss	Entropy, a = 0.001	RMSprop(0.001)	-20.96
Model 2	Advantage, L1 loss	E-greedy, e = 0.05	Adam (0.001)	-2.17
Model 3	Advantage, L1 loss	None	Adam(0.001)	3.11
Model 4	Advantage, Huber loss	Entropy, a = 0.001	Adam(0.001)	7.29
Model 5	**Advantage, L1 Loss**	**Entropy, a = 0.01**	**Adam(0.001)**	**13.47**

Table 1: Performance of models with varying parameters on Pong-v0

The models were run for at least 2500 episodes and then left to plateau. The best performing agent was model 5 (depicted in figure 9) attaining an average reward of 13.47 after 10000 episodes.

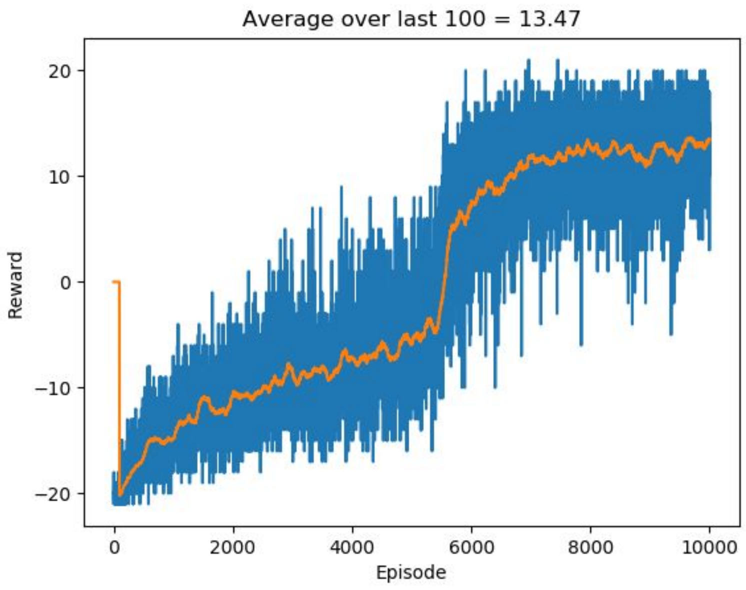


Figure 9: Performance of A2C model with the best reward on Pong-v0

DQN

The DQN Image model faired better than its RAM counterpart, reaching an average score of -15 relatively quickly. However, its performance stalled over the remaining episodes, reaching a maximum average reward of -11 and stabilising at an average reward of -11.55.

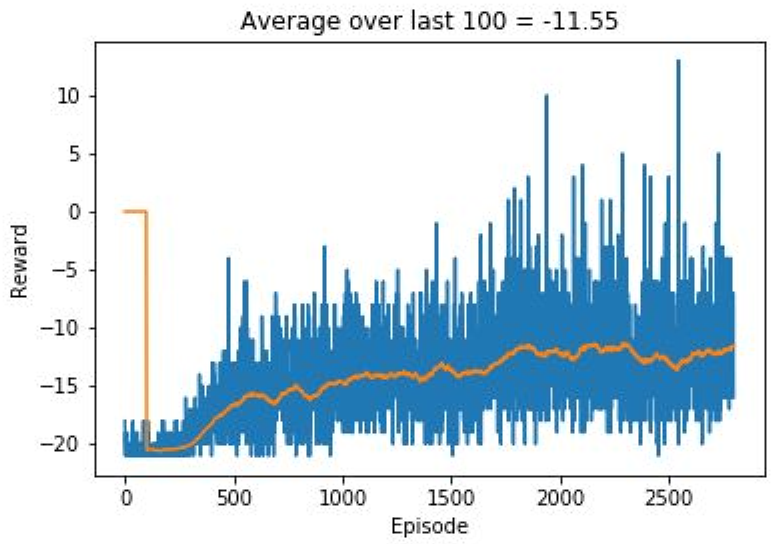


Figure 10: Performance of DQN model on Pong-v0

Images (No Frameskip)

For this environment, only A2C and DQN models were run due to time constraints. Baseline was left out due to poor performance in the other environments. Since PongNoFrameskip-v4 is an extension of Pong-v0, it is expected that A2C performs well here too.

A2C

The A2C model increased relatively fast in this environment, reaching an average score of -15 in just 500 episodes, however it seemed to have converged at a local optimum, remaining relatively constant and increasing only slightly to -13.24 at 1000 episodes. Figure 11 compares the A2C agents from both environments: the left image shows model 5 from above captured at 1000 episodes whereas the right shows. As shown, the previous model on the Pong-v0 environment is still increasing at 1000 episodes while the model on PongNoFrameskip-v4 seems to be plateauing.

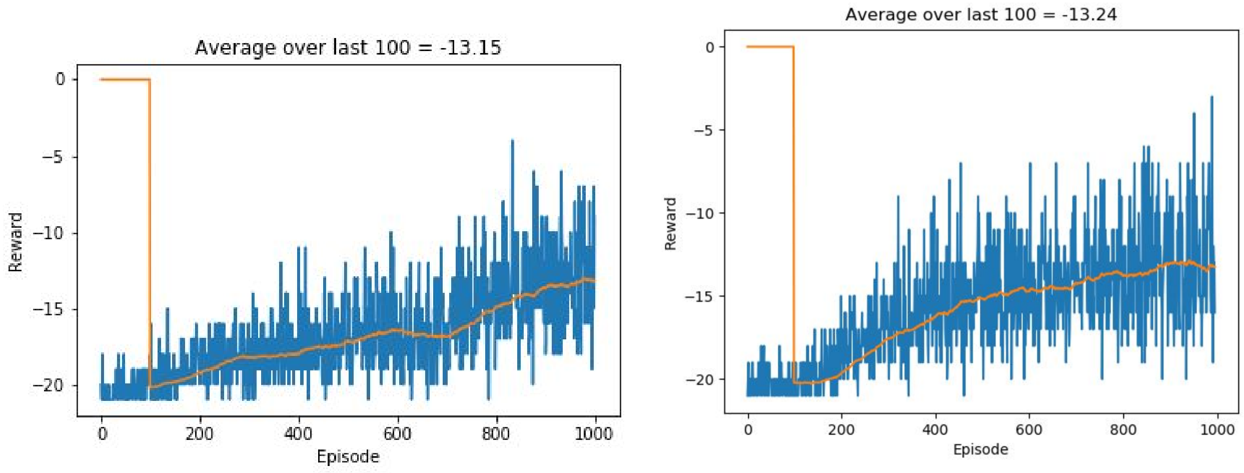


Figure 11: A2C model on Pong-v0 (left) vs PongNoFrameskip-v4 (right)

DQN

The DQN performed extremely well in the no frame skip environment (PongNoFrameskip-v4), reaching a positive average reward in just 150 episodes. After running for 227 episodes, the average rewards surpassed the A2C model on Pong-v0 of +13.47. The average rewards started to taper off at about 500 episodes, at +17.35. Further training was considered, but due to time constraints, it was decided that there was no need for an extended run as this was the best model by a great margin.

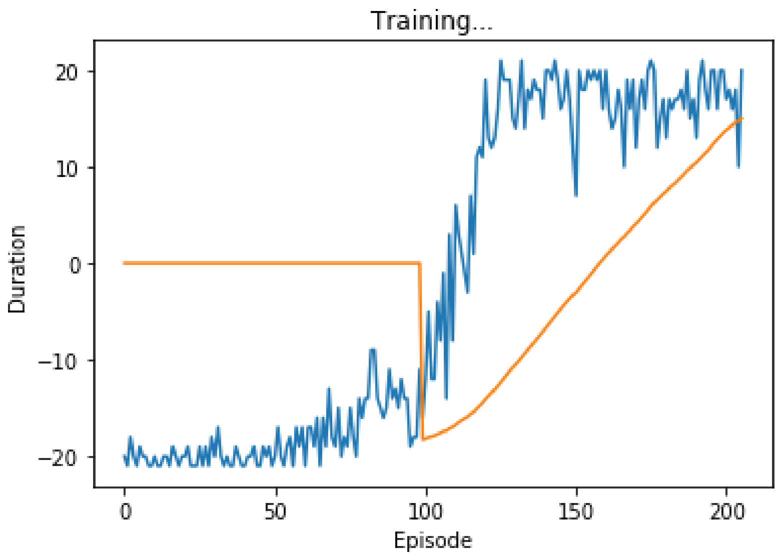


Figure 12: Performance of DQN on PongNoFrameskip-v4

Evaluation

Atari Pong is a difficult game to train since its rewards are very sparse. In all environments, the agent does not receive a reward until the end of a match (i.e. when a player gains a point), this makes experience based learning quite inefficient. For instance, memory based algorithms like DQN are unable to make use of the ability to learn from past experiences if the rewards only comes once in every 5000 frames or so. This could explain why DQN performed worse than A2C for pong-v0.

The RAM models performed significantly worse than their image-based counterparts. This could be due to the RAM state being harder for the neural network to interpret. Despite having a smaller input size of 128 as compared to the images, 6400, the RAM state does not directly correspond to any object in the game, rather it is just the state of the RAM of that instance in time, which may have overlapping representations [6].

Generally with “Pong-v0” environment, the RL model acts only once every 4 frames. This allows the model to train and run faster (x4 faster) than “PongNoFrameskip-v4”, minimally affecting the effectiveness of the model. While making use of the no frame skip environment, the model experienced a massive increase in performance, thereby yielding a stable mean reward value of 17. This increase can be attributed to the increase in the number of frames evaluated per episode.

Conclusion

In this project we evaluated 3 different reinforcement learning methods in 3 different environments for the atari game, pong. For Pong-ram-v0, A2C was the best model with a stable average reward of -15; For Pong-v0, A2C performed better than the DQN and the baseline with a stable average reward of +13.47; And for PongNoFrameskip-v4, DQN was the best model with an average reward of +17.35.

The initial goal was to achieve a positive average score, implying that the agent has 'won' the game. This was achieved in the Pong-v0 and PongNoFrameskip-v4 environments with the A2C and DQN models respectively.

In order to improve training speed and faster convergence, models like Asynchronous Advantage Actor Critic (A3C) which involves using multiple agents can be used. For the DQN model, increasing the Replay Memory size may yield better results. In addition, the performance of a model is attributed to many factors including pre-processing methods, parameter tuning and the type of environment chosen.

References

[1] Atari Pong Environment. (n.d.). Retrieved from https://www.endtoend.ai/envs/gym/atari/pong/

[2] Trazzi, M. (2019, May 06). Spinning Up a Pong AI With Deep Reinforcement Learning. Retrieved from https://blog.floydhub.com/spinning-up-with-deep-reinforcement-learning/

[3] Karpathy, A. (n.d.). Deep Reinforcement Learning: Pong from Pixels. Retrieved from http://karpathy.github.io/2016/05/31/rl/

[4] Ziebart, B. D. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. Carnegie Mellon University, 2010. Retrieved from https://arxiv.org/pdf/1801.01290.pdf

[5] Omerbsezer. (2019, January 22). Omerbsezer/Reinforcement_learning_tutorial_with_demo. Retrieved from https://github.com/omerbsezer/Reinforcement_learning_tutorial_with_demo

[6] AwokeKnowing (2017, August 11). How to interpret the observations of RAM environments in OpenAI gym? Retrieved from https://stackoverflow.com/questions/45207569/how-to-interpret-the-observations-of-ram-environments-in-openai-gym

Appendix

A- Wrappers

The following wrappers were obtained from the OpenAI baselines repository that implements methods and algorithms for use in Atari games for Reinforcement Learning.

1. **FireResetEnv** - In some games, such as Breakout and Pong, the game begins when FIRE was pressed. Therefore, the wrapper in question initiates the game when reset is called by providing the FIRE action and following it with a RIGHT action to make the paddle go up.

2. **MaxAndSkipEnv** - This compresses the actions and pixels in the next “K” consecutive frames.

3. **ProcessFrame84** - This wrapper pre processes the image from the monitor that are originally 210x160 resolution with all three RGB channels to a grayscale 84x84 resolution image.

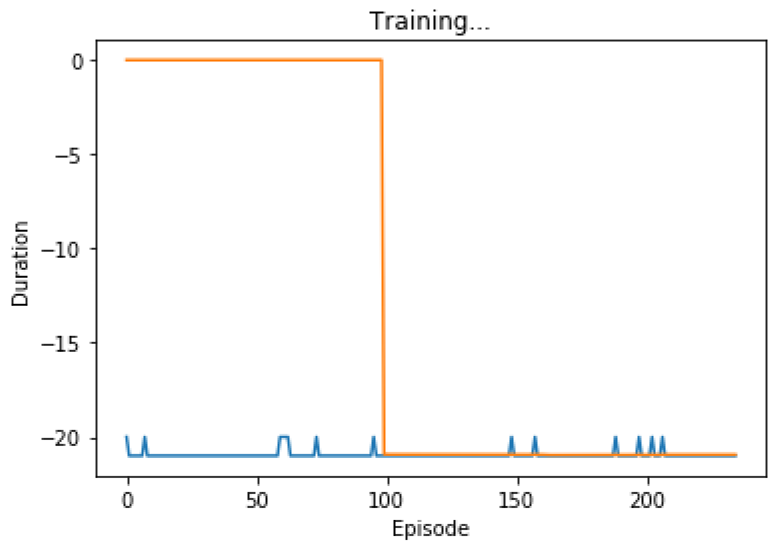
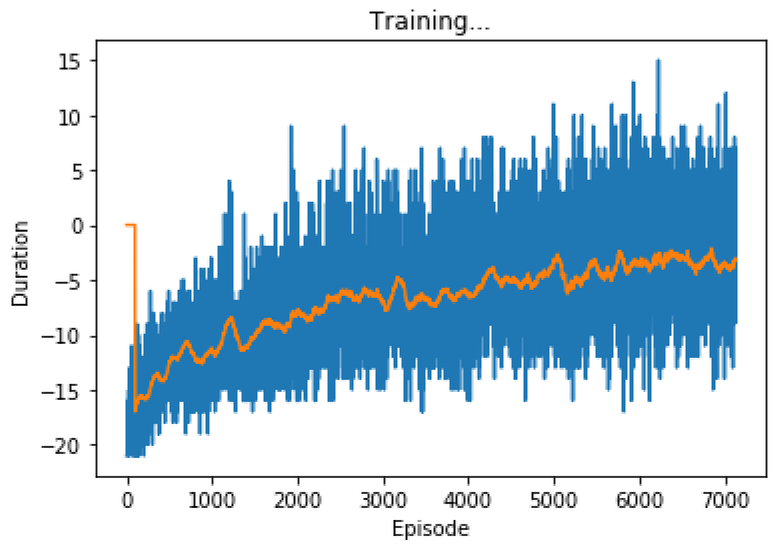
4. **ImageToPyTorch** - This wrapper changes the format of the input image from HWC to the required CHW.

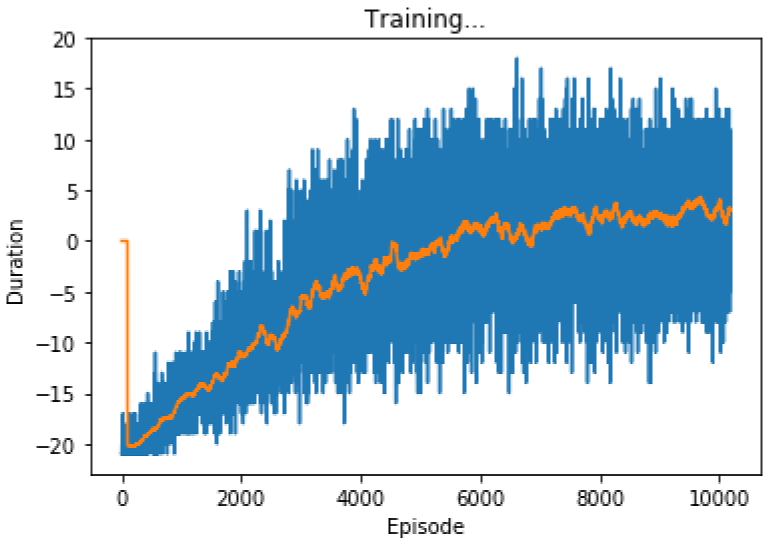
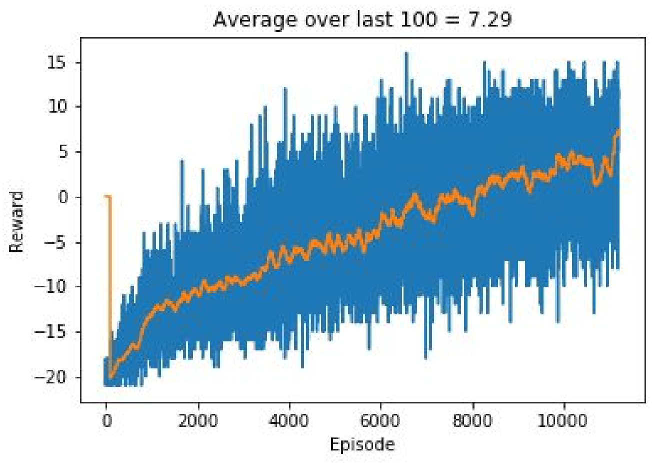
5. **BufferWrapper** - This wrapper maintains a stack of all frames to keep track of the dynamics of the game. This is especially so that enemy speed and direction in understood and the information is preserved.

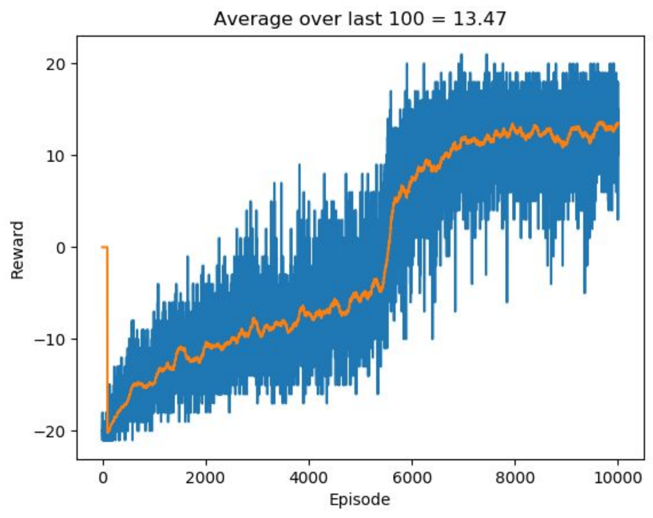
6. **ScaledFloatFrame** - This wrapper scales the pixel values to the range (0-1).

B- A2C Models

	Actor Loss, Critic loss	**Exploration**	**Learning rate**	**Mean reward**
Model 1	Q, L1 loss	Entropy, a = 0.001	RMSprop(0.001)	-20.96

				
Model 2	Advantage, L1 loss	E-greedy, e = 0.05	Adam (0.001)	-2.17
				
Model 3	Advantage, L1 loss	None	Adam(0.001)	3.11

				
Model 4	Advantage, Huber loss	Entropy, a = 0.001	Adam(0.001)	7.29
				
Model 5	**Advantage, L1 Loss**	**Entropy, a = 0.01**	**Adam(0.001)**	**13.47**



C- Github link

This is the repository where all of the code for this project is uploaded to.
https://github.com/dhritiwasan14/AI-Project