Artificial Intelligence Report

Chuah Min Shian Mark 1002480

Wang Yee Lin Pamela 1002367

Introduction

Goal

To train a model that can play the Atari game Breakout using the pre-processed grayscale downsampled images as its input.

How Atari Breakout Differs from LunarLander and Cartpole

- Breakout will sparse reward frequency unlike LunarLander and Cartpole which have constant feedback.
    - Gamma will have to be larger to account for points one will get later
    - The model should look more than 2 steps ahead
- Breakout has the potential of getting stuck; the ball will not appear until the 'Fire' action is executed meaning that if the machine does not it will get stuck. Compared to LunarLander and Cartpole where constantly doing the wrong actions will eventually lead to the game ending.

Current State-of-the-Art Approaches

- On the Atari gym leaderboard an A3C Model done by github user ppwwyyxx achieved a past 100 average score of 760.07
- A DQN model developed by DeepMind achieved a maximum test score of 225 with a largest past 100 average score of 168

Dataset

Environment: BreakoutDeterministic-v4. This reads the image once every four frames hence allowing the movement of the ball to be seen and hence allowing the model to better identify movement. Unlike other implementations of Breakout in the gym AI environments, BreakoutDeterministic-v4 does not require us to implement frame skipping as 4 frames are skipped each time an action is taken in this environment.

Input: N*210*160 image from the gym AI environment. We preprocess this into various forms for the different approaches, finally using a resized grayscale image matrix with multiple frames consisting of the current frame and N-1 previous frames.

Output: A probability distribution for each of the four actions allowed in Breakout's gym: NOOP, FIRE, RIGHT and LEFT.

Literature Review

Reinforcement Learning for Atari Breakout

According to the CS 221 Project Paper from the Stanford University done on Breakout, simple linear function approximators worked better than Neural Nets. Simple learning algorithms like SARSA paired with small feature sets did the best compared to more complex counterparts like replay Q-learning. Furthermore while replay and eligibility trace mechanisms offer benefits on paper, in practice they decrease performance.

Asynchronous Methods for Deep Reinforcement Learning

A study was done which compared various different methods for Deep Reinforcement Learning. The methods tested were asynchronous one-step Q-learning, asynchronous one-step SARSA, asynchronous n-step Q-learning and asynchronous advantage actor-critic.

This paper's results suggest that A3C is the most efficient in terms of training time for breakout. It also found that 16 threads lead to at least an order of magnitude speedup. In general larger epochs lead to better training efficiency. Larger numbers of threads also lead to better time efficiency.

Initial Approaches

Pre-processing

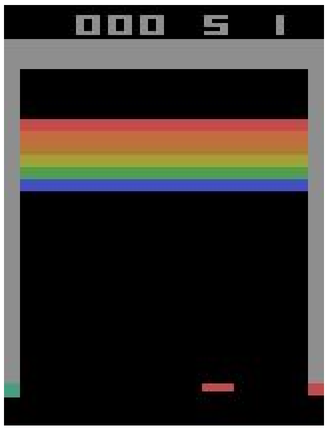


Figure 1.0.1 Original Image of Initial State

Resizing Image

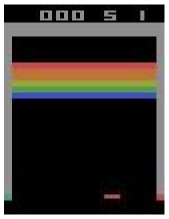


Figure 1.0.2 Resized Image of Initial State to 105*80*3

We downsampled the image to reduce the amount of memory it took up as states were saved for experience replay.

Color Conversion of Image

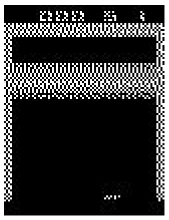


Figure 1.0.3 Black and White Image of Initial State

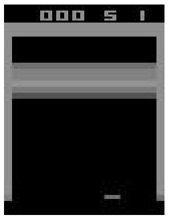


Figure 1.0.4 Grey Image of Initial State

As shown in Figure 1.0.3 & 1.0.4 above, we first attempted to reduce the amount of memory taken by the observations by reducing the color channels from the 3 channels of RGB to black and white by using PIL's convert("1") which seemed to be detrimental to the model's learning. Thus we switched to converting the image to grey which did not adversely affect learning and reduced the memory usage.

Models

All the initial ones save for A3C were ran on colab and hence uses the following packages and versions.

- Python 3.6.7
- gym 0.10.1
- math
- random
- numpy 1.16.4

- matplotlib 3.0.3
- torchvision 1.14.0
- collections
- itertools
- pil
- torch 1.1.0
- csv
- os

	Double-Q	Reinforcement	Reinforcement with Baseline	A2C: 8-step actor critic
Env	Breakout-v0	Breakout-v0	Breakout-v0	BreakoutDeterministic-v4
Input	4*210*160	3*210*160	3*210*160	4*210*160
CNN	2d conv kernel size 10, stride 5 and outputs 6 channels Selu 2d conv with kernel size 5 and outputs 2 channels Selu 2d conv with kernel size3, stride 2 and outputs 1 channel Convert the 1*18*13 image to an array of size 234 Linear transformation with bias and array output of size 4	2d conv kernel size 5 and outputs 7 channels Leaky relu 2d conv kernel size 4, stride 2 and outputs 5 channels Max pooling with kernel size 3 2d conv with kernel size 5, stride 3 and outputs 4 Convert 4*10*7 matrix to a 4*70 matrix Linear transformation to a 4*1 array	2d conv kernel size 5 and outputs 7 channels Leaky relu 2d conv kernel size 4, stride 2 and outputs 6 channels Max pooling with kernel size 3 2d conv with kernel size 5, stride 3 and outputs 5 Convert 5*10*7 matrix to a 5*70 matrix Linear transformation to a 5 by 1 array For pi linear transform the 5*1 array to 4*1, for b linear transform to 1*1	Individual separate ones for b and pi: ● 2d conv kernel size 5 and outputs 7 channels ● Leaky relu ● 2d conv kernel size 4, stride 2 and outputs 6 channels ● Max pooling with kernel size 3 ● 2d conv with kernel size 5, stride 3 and outputs 5

				• Leaky relu Convert 5*10*7 matrix to a 5*70 matrix Linear transformation to a 5 by 1 array For pi linear transform the 5*1 array to 4*1 For b linear transform to 1*1
lr	0.01	0.01	0.01	0.001
Reduction	Halve every 50 epochs	Halve every 50 epochs	Halve every 50 epochs	Halve every 50 epochs
Optimizer	ADAM	ADAM	ADAM	ADAM
Exploration Algorithm	Epsilon greedy Epsilon = 0.9 - (0.9-0.05)*e^(-steps/200)	Epsilon soft Epsilon = 0.9 - (0.9-0.05)*e^(-steps/400)	Epsilon soft Epsilon = 0.9 - (0.9-0.05)*e^(-steps/400)	Epsilon soft Epsilon = 0.9 - (0.9-0.05)*e^(-steps/400)
Gamma	0.95	0.99	0.99	0.99
Notes	Has an artificial frameskip where after every action the next three will wait			Trains on latest, 2nd latest and best epoch
Epochs	500	500	500	500
Best Past 100 Ave	1.38 at Epoch 314	1.48 at Epoch 243	2.23 at Epoch 469	Latest only: 0.95 at Epoch 422 With 2nd Latest and Best: 1.45 at Epoch 105
Best Test	2 at Epoch 250	0	0	0

None of the above were ideal as either the training averages did not improve or the test data could not yield any score.

A3C

The Actor-Critic algorithm is a model-free, off-policy method where the critic acts as a value-function approximator, and the actor as a policy-function approximator. When training, the critic predicts the baseline and guides the learning of both itself and the actor. We do this by approximating the TD-Error using the advantage function (advantage = total return - baseline). We also incorporates asynchronous weight updates, allowing for much faster computation. We use multiple agents to perform gradient ascent asynchronously, over multiple threads.

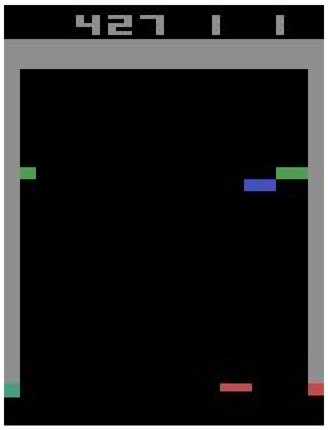
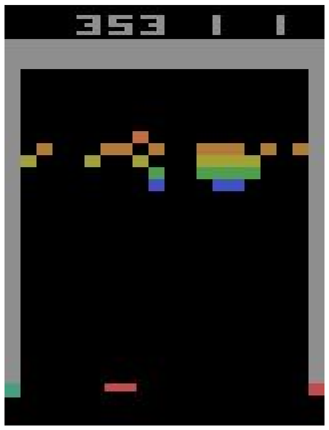
Below are the packages and versions used for our A3C. Each thread used a different exploration policy.

Packages Used

- Python 3.7.4
- skimage
- multiprocessing
- collections
- gym 0.14.0
- numpy 1.17.0
- h5py 2.9.0
- argparse
- pathlib
- os
- statistics
- matplotlib 3.1.1
- datetime
- pandas 0.25.0
- csv
- time
- signal
- keras 2.2.4
- guizero 1.0.0

	A3C: 4 Threads	**A3C: 16 Threads**
Environment	BreakoutDeterministic-v4	BreakoutDeterministic-v4
Input	1*210*160	1*210*160
Number of Threads	4	16
Initial Learning Rate	0.001	0.001

Learning Rate Decay Rate	80000000	80000000
Minimum Learning Rate	0.00000001	0.00000001
Reduction	max(0.001*(80000000-steps)/80000000, 0.00000001)	max(0.001*(80000000-steps)/80000000, 0.00000001)
Batch Size	64	20
Swap Rate	100	100
N-Steps	10	5
Beta	0.01	0.01
Experience Queue Size	64	256
Auto Fire	True	False
Past Frame to look at	8	3
Model	2d Conv kernel size 8, stride 4 and output of 16 Relu 2d Conv kernel size 4, stride 2 and output of 32 Relu Flatten Linear transformation to output size of 256 Relu For b, do linear transformation with 1 output. For policy, do linear transformation with 4 outputs and softmax activation	2d Conv kernel size 8, stride 4 and output of 16 Relu 2d C kernel size 4, stride 2 and output of 32 Relu Flatten Linear transformation to output size of 256 Relu For b, do linear transformation with 1 output. For policy, do linear transformation with 4 outputs and softmax activation
Exploration Algorithm	Soft Policy	Soft Policy
Optimizer	RMS Prop, Initial rho 0.99, Initial epsilon 0.1	RMS Prop, Initial rho 0.99, Initial epsilon 0.1
Best Test Score with end state	427	353

image		
Average Test Score for Best Weights	350.0 for weights at 10299968 frames	271.0 for weights at 18000000 frames

Results

16 Threads

When 16 threads are used. The model learns pretty well. Average rewards after 500 epochs also exceeded 5, showing that A3C was stronger than the other four algorithms tested. However after about 2200 epochs scores of 0 start to appear very frequently and after more than 4500 epochs the score is always zero as shown in figures 1.1 and 1.2.

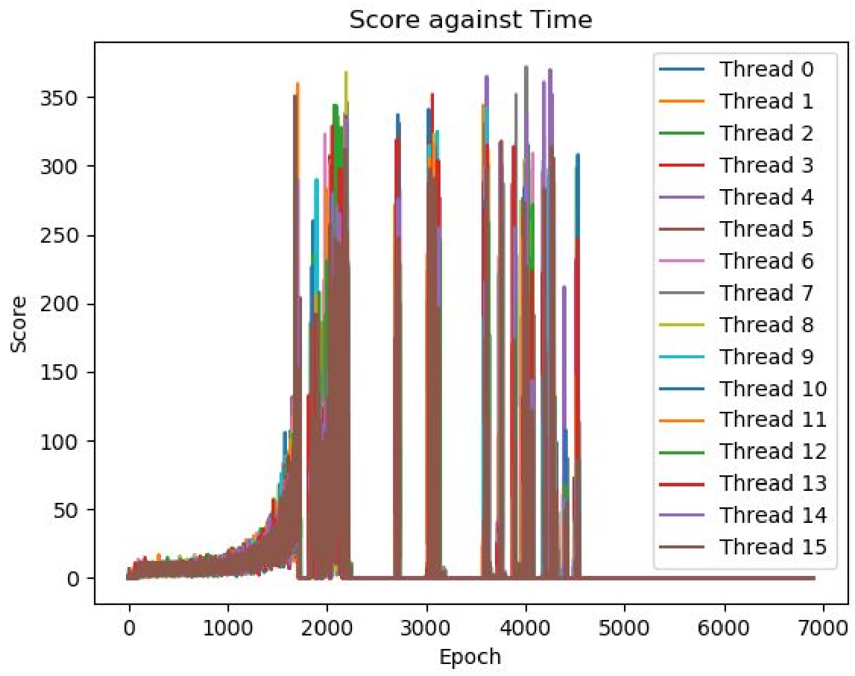


Figure 1.1: All 16 threads against their epochs

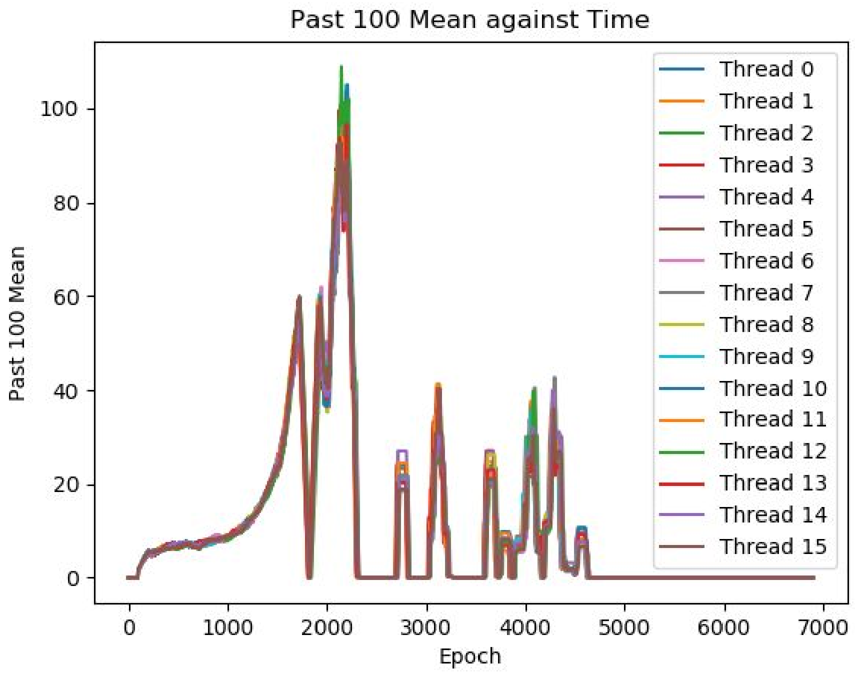


Figure 1.2: Average all epoch against time

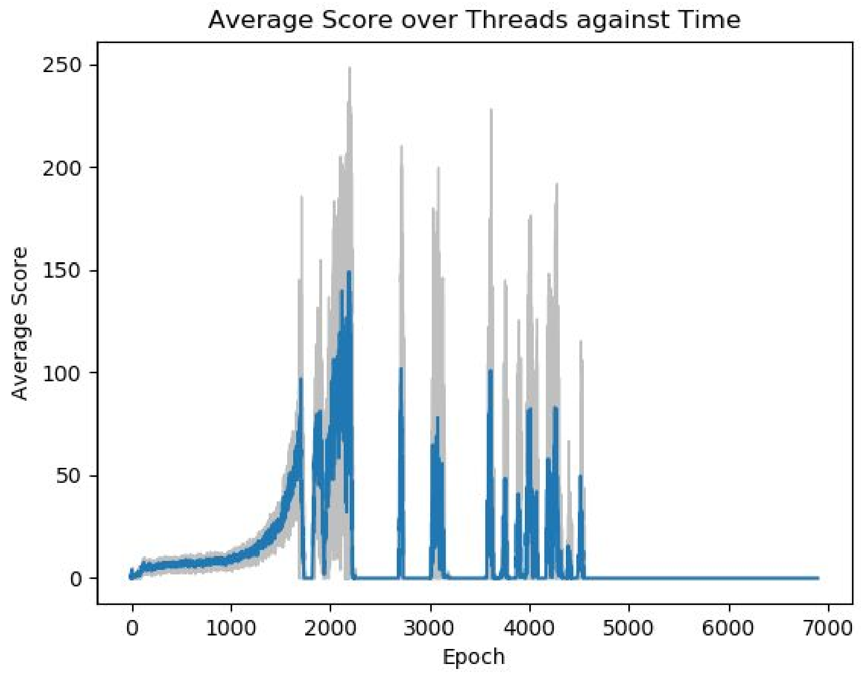


Figure 1.3: Average training score over all 16 threads against time. Grey area shows the average plus or minus standard deviation.

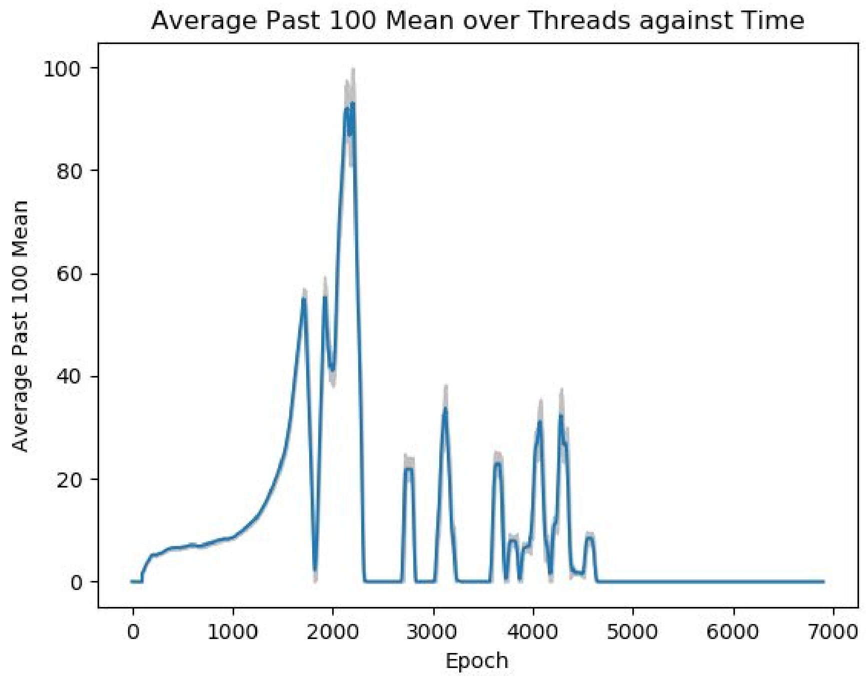


Figure 1.4: Average of past 100 averages against time. Grey area shows the average plus or minus standard deviation.

The 16 threads A3C model only looks at the past 3 frames and considers less n-steps than the 4 threads A3C model, it learns to hit the ball consistently but never learns to tunnel consistently. It de-learns to not fire at the start.

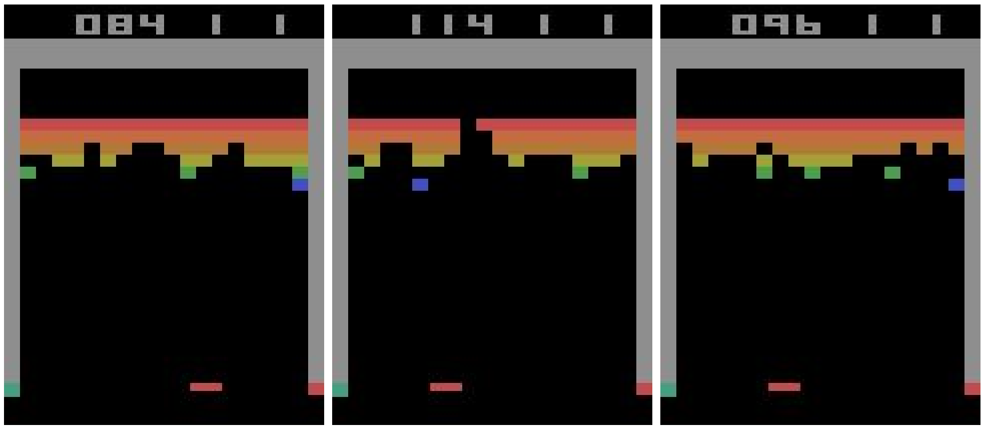


Figure 1.5: End States of 16-thread A3C Model

4 Threads

With 4 threads initially the average reward for the first 500 epochs were still around 1 or 2. However it manages to pick up later on. The model stops yielding good results after around 5200 epochs indicating that less threads can lead to better learning in the long run as it did not unlearn as quickly. It also learns the tunnelling strategy, indicating that looking at more past frames helps the model learn an actual strategy.

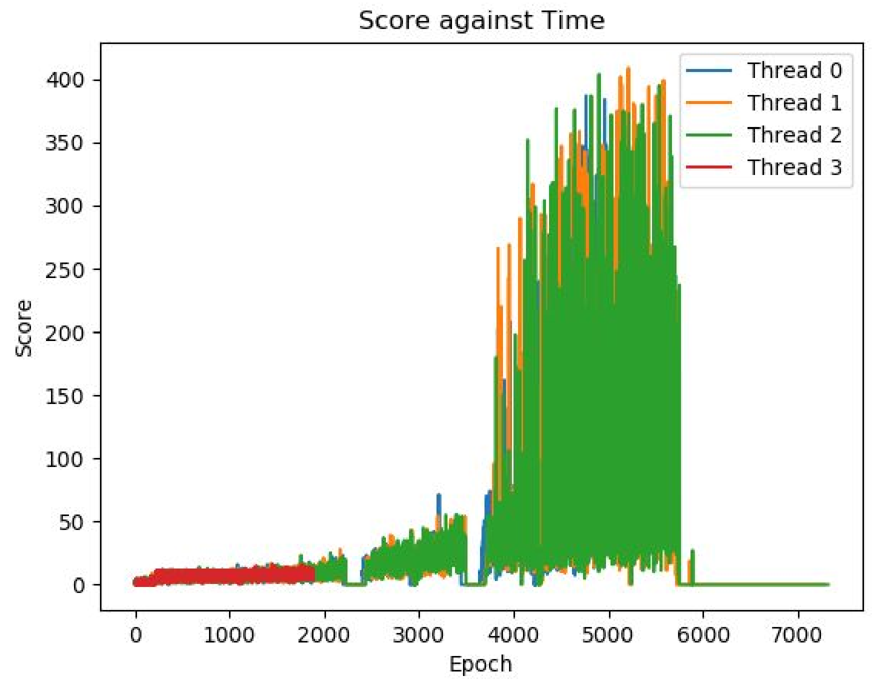


Figure 2.1: Training scores all 4 threads against epochs

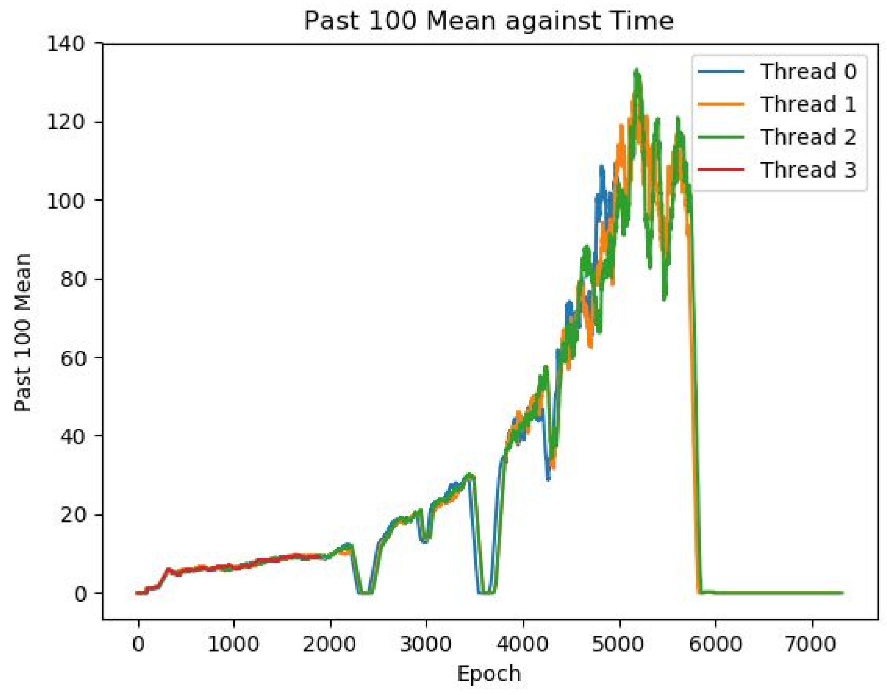


Figure 2.2: Average past 100 training scores against epochs

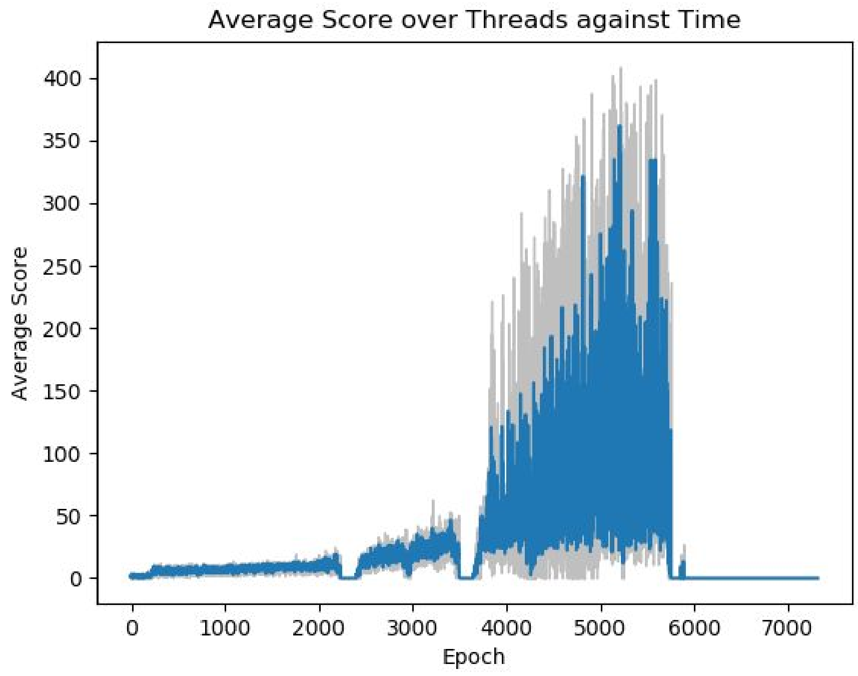


Figure 2.3: Average training scores over threads against epochs. Grey area shows the average plus or minus standard deviation.

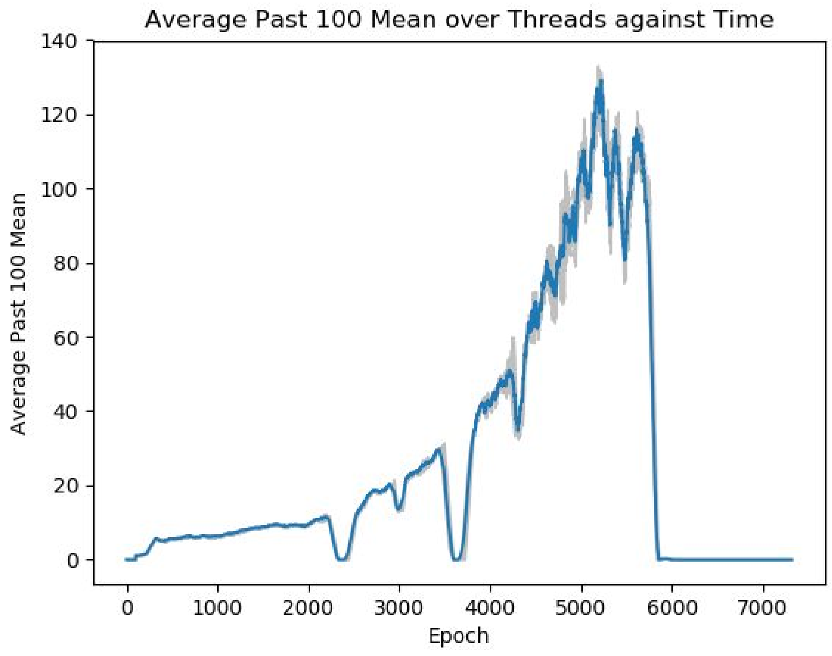


Figure 2.4: Average past 100 mean over threads against epochs. Grey area shows the average plus or minus standard deviation.

The A3C model with 4 threads that looks at 8 past frames with a larger n-step learns to tunnel through the center to earn more rewards which is a strategy that yields greater rewards than the other models. Although this model does not manage to hit the ball as consistently as the 16 threads model, it tends to have higher rewards.

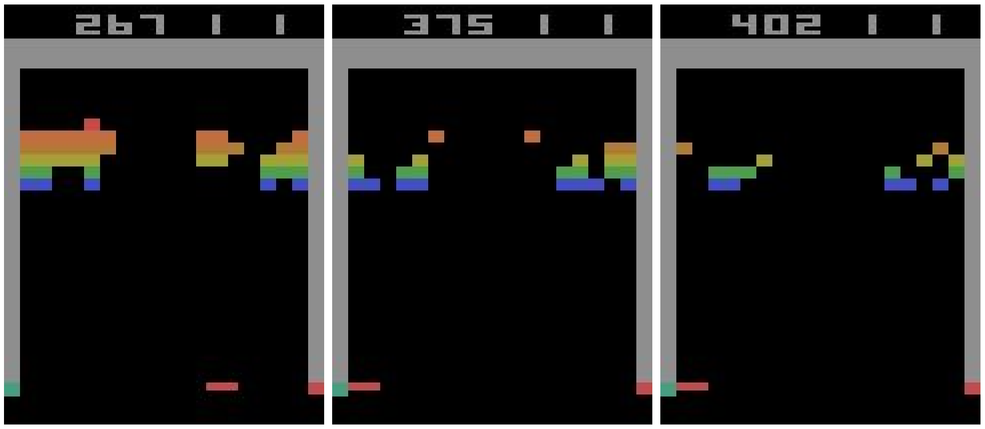


Figure 2.5: End State of 4-Thread Model

Furthermore, the largest average of the past 100 means for 4 threads is over 120 which is much larger than the largest for 16 threads which is less than 100. Hence showing that 4 threads is better. Not to mention it saves space as well due to having less environments.

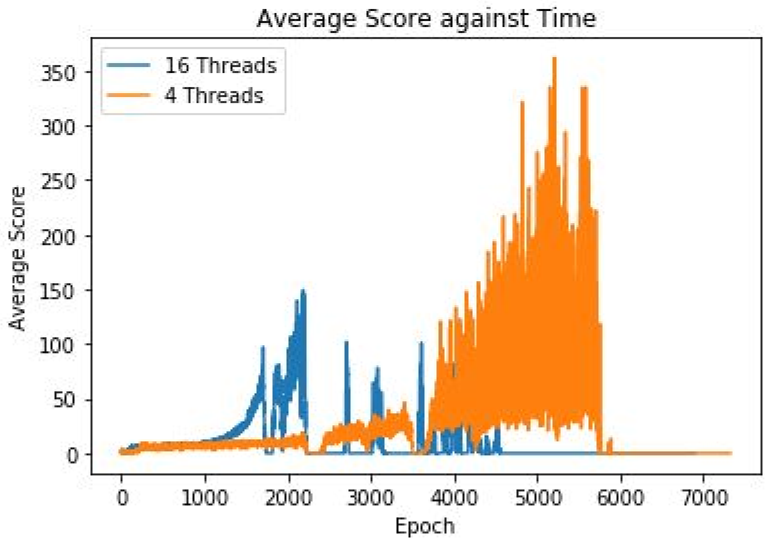


Figure 3.1: Raw average of training scores over the threads against Epochs. 4 Threads did better in the long run while 16 threads initially increases faster.

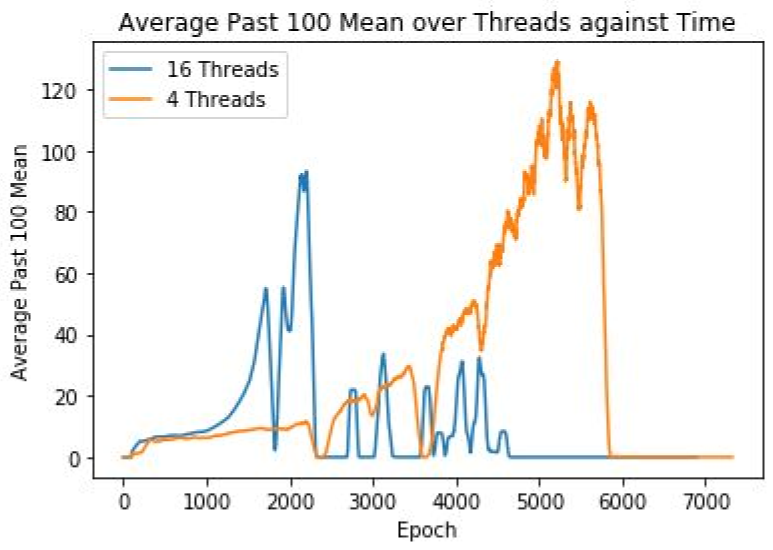


Figure 3.2: Average past 100 mean over the threads against time. Once again in the long run 4 threads did better than 16 threads.

Conclusion

Hypothesis on Initial Test Performance

One reason the initial tests did not perform as well as the A3C Models could be that they utilised too few channels in each layer. This in turn meant that there were too few values and hence less accurate predictions.

Second could be due to the lack of additional threads. The lack of threads results in less stable learning hence there was a greater chance of de-learning.

The Double-Q may have yielded better test scores even though it does not look as many steps forward because of it's exploration policy. Double-Q uses epsilon greedy which selects the most probable action with a random probability of epsilon of a random action. Actor-critic, reinforcement and reinforcement with baseline utilise epsilon soft which selects randomly from a returned probability distribution. This means Double-Q is explores more thus it may learn a more accurate model than the rest.

At the same time, it must also be noted that the initial tests were done with only 500 epochs which is too little to see any significant changes.

A3C Models

The A3C model with 4 threads that looks at 8 past frames with a larger n-step and a smaller batch size outperforms the other models attempted such as the A3C model with

16 threads that only looks at 3 past frames. This indicates that Breakout benefits from looking at past frames in order to learn a better playing strategy due to the sparse score frequency requiring one to consider many frames ahead. However measures need to be taken to ensure that the model does not de-learn to not fire at the start or to only choose to go left as both models have shown a tendency to do so.

We also found that increasing the batch size helped control the stability of learning, this was our measure to preserve stability when we decreased the number of threads used from 16 to 4 threads as more threads make the learning more stable but running more threads also slows down learning in terms of computational speed.

References

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. NIPS Deep Learning Workshop 2013. Retrieved from https://arxiv.org/abs/1312.5602

(2019, August 8). Leaderboard openai/gym Wiki. Retrieved from https://github.com/openai/gym/wiki/Leaderboard

(2019, January 28). Deep-RL-Keras/a3c.py at master germain-hug/Deep-RL-Keras. Retrieved from https://github.com/germain-hug/Deep-RL-Keras/blob/master/A3C/a3c.py

(2017, April 15). AI-blog/CartPole-A3C.py at master jaromiru/AI-blog. Retrieved from https://github.com/jaromiru/AI-blog/blob/master/CartPole-A3C.py

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., … Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. ICML 2016. Retrieved from https://arxiv.org/abs/1602.01783

Berges, V.-P., Rao, P., & Pryzant, R. Reinforcement Learning for Atari Breakout. Stanford University CS 221 Project Paper. Retrieved from https://cs.stanford.edu/~rpryzant/data/rl/paper.pdf