Lewis Stuart

Notes

It is important to add that around 10000 episodes normally need to be played to produce significantly impressive results; for most of the results only 1000 will be played as computation times are too large

Attempt #1

Parameters	Values
Learning Rate	0.001
Epsilon values	Start: 1, End: 0.1, Decay: 0.0025
Discount	0.999
Image input resize	80-50
Crop	Width: 0.05-0.95, Height: 0.15-0.95
Screen process type	Append
State queue size	4
Learning technique	Deep Q-Learning
Network type	CNN
Batch size	256

Game: Deterministic Breakout V4

Neural net parameters:

- Kernel_sizes: [8, 4, 3]
- Strides: [4, 2, 1]
- Neurons per layer: [32, 64, 64]

Results:

- Average reward: 4
- Highest reward: 17
- Number of episodes: roughly 1000

Conclusions:

A good start to learning, the AI is clearly processing the state information correctly and producing coherent actions that lead to high scoring outcomes. In addition, the AI was moving consistently while the game was taking place, hitting the ball back pretty consistently anywhere in the environment

The problems arise with computation time, it took around 12 hours to reach 1000 episodes on the computer; to reach any sort of conclusions (needs around 14,000 episodes) is not really possible on this implementation. Hence, some changes need to be made to make the AI faster, this can be done by decreasing the size of the input to the neural network, and reducing the among of neurons in each layer. This is the next course of action

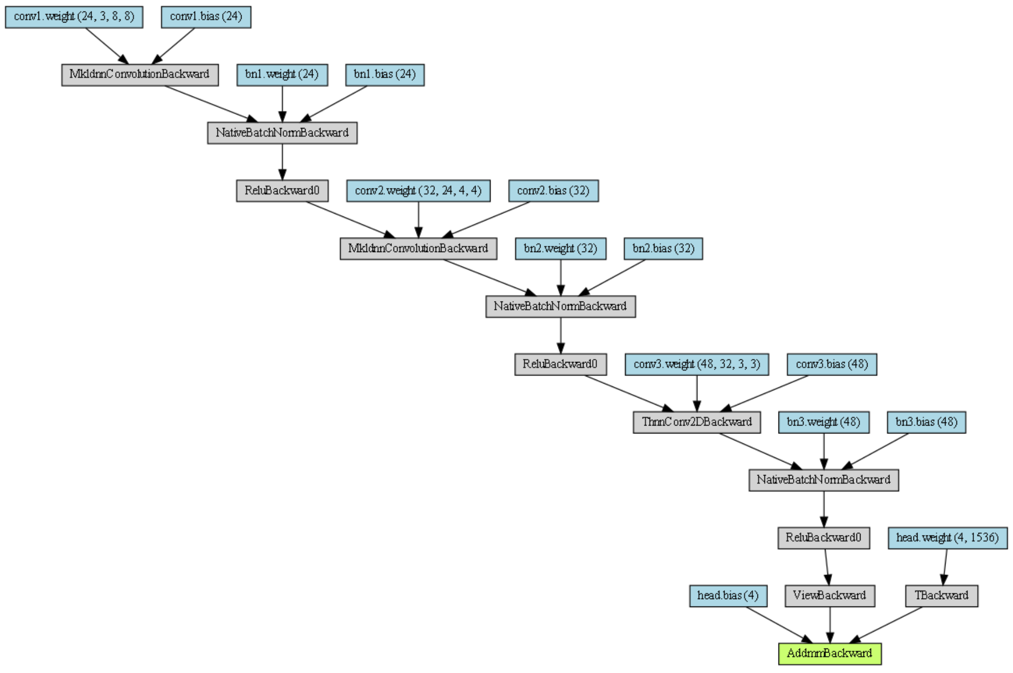
Lewis Stuart

Attempt #2

Game: Deterministic Breakout V4

Parameters	Values
Learning Rate	0.001
Epsilon values	Start: 1, End: 0.1, Decay: 0.0005
Discount	0.999
Image input resize	72-40
Crop	Width: 0.05-0.95, Height: 0.25-0.95
Screen process type	Append
State queue size	4
Learning technique	Deep Q-Learning
Network type	CNN
Batch size	256

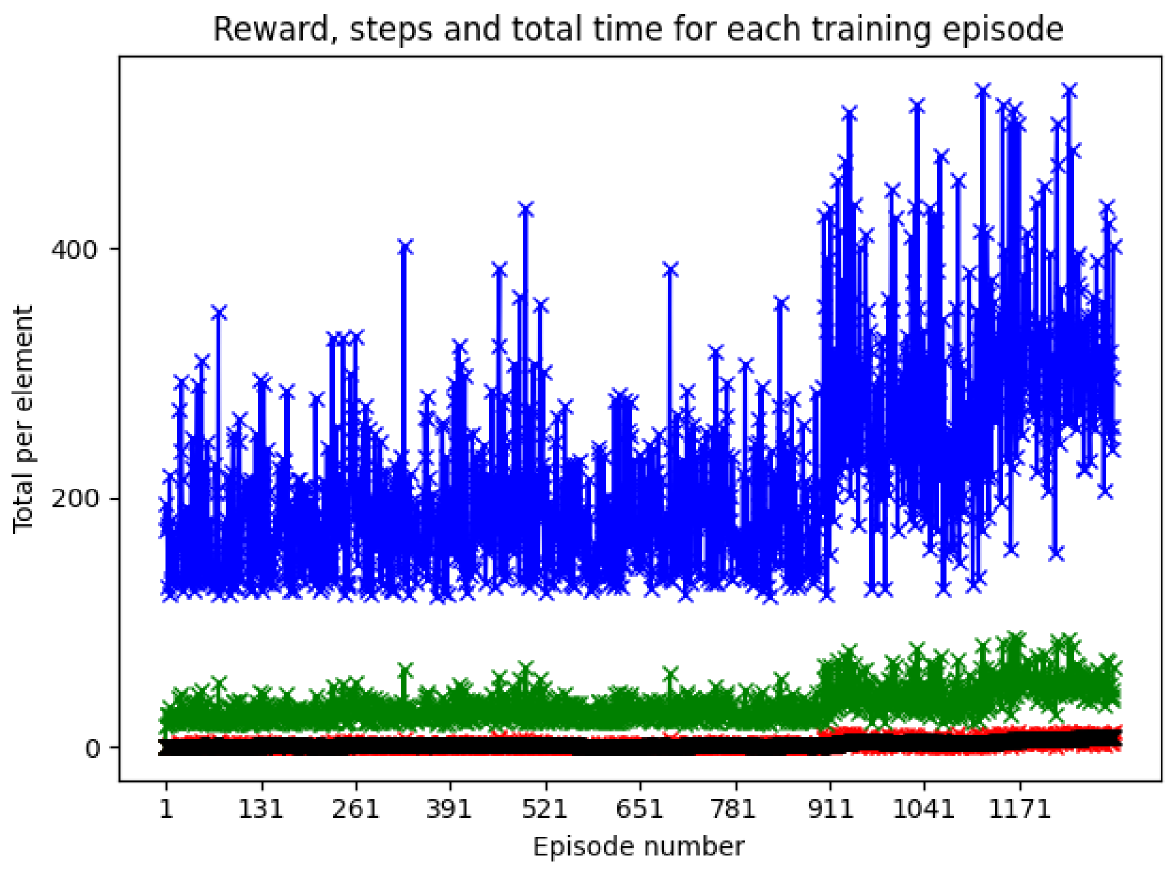
Neural net parameters:



- Kernel_sizes: [8, 4, 3]
- Strides: [4, 2, 1]
- Neurons per layer: [24, 32, 48]

Results:

- Average reward: 7.5
- Highest reward: 12
- Number of episodes: roughly 1200

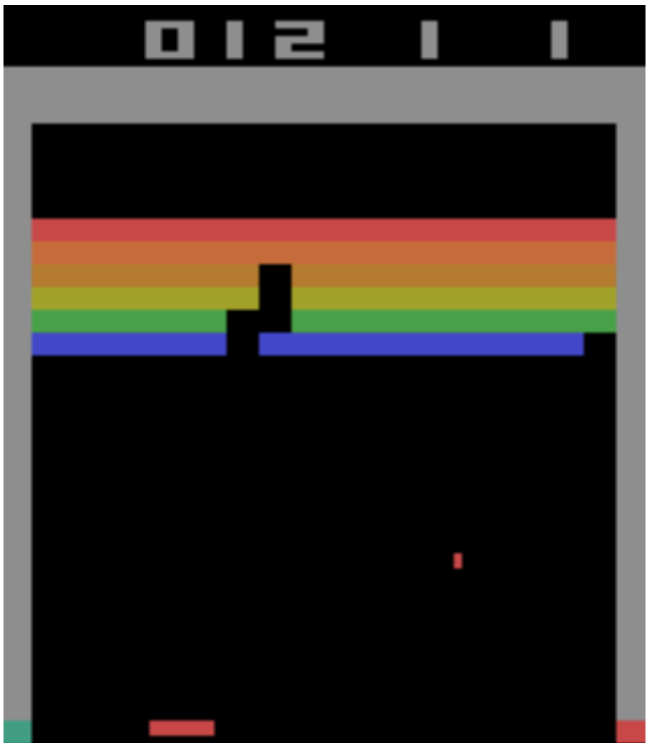


Conclusions:

Firstly, the number of neurons has been reduced, as well as increasing the amount of cropping and reducing the resize values for the input image. While this will reduce the complexity of the agent, and theoretically make the agent worse, it almost doubled the speed of the agent, making training times for the agent much lower

While this agent has consistently better scores than the previous attempt, it suffers from some major issues; the main one being that it follows the same pattern every episode.

Because this game is deterministic, the ball will always enter the scene from the middle heading down to the left. Hence, if the AI always went left, it will always hit the ball and get at least one point for hitting it into a brick. However, the game breakout rewards more points for hitting bricks that are deeper into its brick structure. Therefore, the AI prioritised this by consistently hitting the ball into the same spot every time, 'drilling' a singular hole deeper into the brick layer:



This was the agent's best performance, getting a total amount of points of 12

The issue is that the agent does not prioritise staying alive: it will hit the ball upwards, the ball hits a brick, and then the ball drops into the pit and the agent loses a life. This is not the point of the game; the game should be about staying alive and keeping the ball in play as long as possible, and thus hitting as many bricks as possible. Getting a large reward for hitting deeper bricks is an inherent problem for this AI, as it will always prioritise this over staying alive and hitting a series of bricks on the lower layer (and getting less points).

A few methods that I believe could solve this issue:

- Restarting the AI, hopefully, will result in the agent not getting fixated on this specific pattern of input, and maybe lead it down a path of better scores
- Prioritise staying alive, this can be done in two ways:
    - Producing a negative reward if the agent loses a life
    - Give the agent a small positive reward for every step it is still in play

  The former seems more viable, however due to the environment, it will involve writing custom code to detect from the pixel data when the agent has lost a life, which defeats the purpose of creating an AI that can play the majority of Atari games
- Changing parameters could result in a different outcome, since the previous AI didn't suffer from this issue, reverting to similar parameters will theoretically null this issue
- Try running the agent on a non-deterministic game. The game will always start with the ball going to the left (from the middle), which produces bad habits of camping in the left corner, as the agent will always hit the ball. Hence, if the ball entered the game from any angle, the agent will have to learn to adapt to different scenarios, and thus stop sitting in the right corner (and hopefully follow the ball while it traverses the environment)

Attempt #3

Game: Deterministic Breakout V4

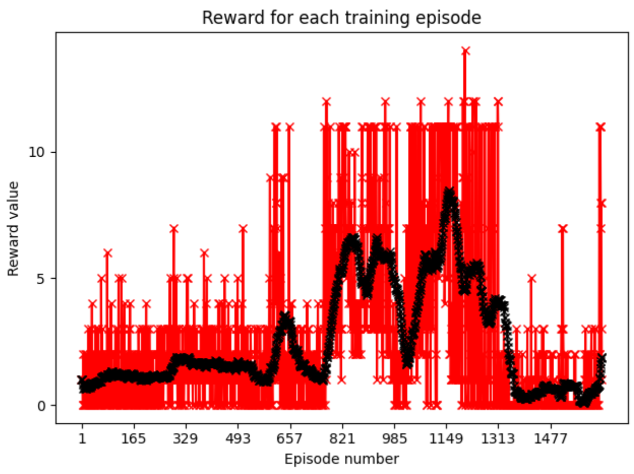
Parameters	Values
Learning Rate	0.001
Epsilon values	Start: 1, End: 0.1, Decay: 0.0005
Discount	0.999
Image input resize	72-40
Crop	Width: 0.05-0.95, Height: 0.25-0.95
Screen process type	Append
Colour type	Grayscale
State queue size	4
Learning technique	Deep Q-Learning
Network type	CNN
Batch size	256

Neural net parameters:

- Kernel_sizes: [8, 4, 3]
- Strides: [4, 2, 1]
- Neurons per layer: [24, 32, 48]

Results:

- Average reward: 5
- Highest reward: 14
- Number of episodes: roughly 1800



Conclusions:

The main changes between this attempt and the previous, was that the image data was converted to grayscale (only one colour channel) rather than using all RGB values. Ultimately, this didn't really have much effect on the agent or the performance, but since colours are not important for this game it's good practice to only evaluate pixel intensity.

Similarly, to the previous attempt, the agent preferred to just hit the deeper bricks and didn't seem to care about whether it lost a life or not; this was how it hit its high scores. I noticed there were some attempts to continue after the deepest brick had been hit, but ultimately the agent didn't seem to improve beyond this point. Since the none of the previous potential improvements that had been stated were implemented, this was to be expected (but it is clear restarting the agent won’t fix this issue)

In addition, the interesting point was that after a certain amount of time the agent suddenly performed much worth, like it had forgotten its progress, and regressed to hitting an average of 0 scores (between episodes 1300-1500) before reverting back to its old formula of drilling into the layer to retrieve the maximum amount of points. This has lead to the conclusion that the learning rate may potentially be too large, hence this will be halved in the next attempt

In addition, the batch size that is analysed is around 256 states, this is around the amount of steps the agent takes per episode. It would be in the best interest to reduce this to increase computation times, only time will tell whether this will actually impact the time taken to learn

For the next attempt, the learning rate will be reduced, and the epsilon values increased to hopefully make the agent choose more actions that do not lead to its drilling deadlock