Breakout DQN

Kyle McClintick
DS595CS525 Reinforcement Learning - 2019
Project 3
Worcester Polytechnic Institute

Fig. 1. Atari's Breakout game. The player or agent must prevent the bouncing ball from moving past the paddle while breaking bricks of the wall to increase score.

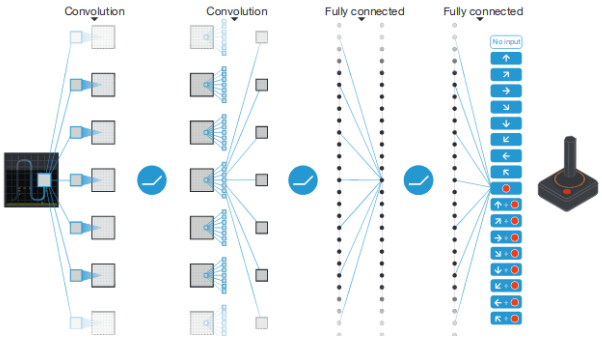

Fig. 2. Mnih et al's DQN architecture, which is composed of CNN layers with 32 square filters of size 8 and stride 4, 64 square filters of size 4 and stride 2, and 64 square filters of size 3 and stride 1, all with ReLU activation's. The model ends with a Fully Connected (FC) ReLU layer with 3136 neurons and a linear FC layer with 512 neurons, outputting 4 values for each state representing $Q(a_1|s)$ through $Q(a_4|s)$.

I. Problem Statement

Mnih et. al's Human-level control through deep reinforcement learning paper serves as the inspiration for this project, where a Deep Q Network (DQN) is used as a parametric model to estimate the goodness of state action pairs in the game of Atari Breakout (Fig 1) such that good actions can be taken and an 100-episode average of 40 reward points can be achieved using the trained DQN. Four actions exist, defined as move left, move right, stand still, and fire. States are given to the DQN in the shape $[BS, 4, 84, 84]$, where batch size is BS, four frames of the game are given such that ball velocity can be observed, and the gray scaled pixels are given as an $84x84$ matrix, as color is insignificant beyond aesthetics. The DQN is used over a tabular Q/SARSA method because the number of possible state/action pairs is very large and can be learned more quickly via a parametric approach.

II. Training

For my implementation, use of pytorch in combination with hardware acceleration proved difficult so no additional features were added. Mnih et. al's DQN architecture was used (Fig 2).

Suggested hyper-parameters were used, being those presented in table II. In addition, gradients and rewards are clipped between $\{-1, 1\}$, and states are normalized between $\{-1, 1\}$. Explore chance ϵ is decayed each time step by $\Delta\epsilon$ until $\epsilon = 0.025$. DQN weights are updated via the Adam optimizer which adjusts the learning rate of every parameter over time to aid with faster convergence. Gradients are calculated using Huber loss.

The most significant challenge encountered towards the completion of the project was maintaining correct tensor dimensions. Particularly, flipping the two size 84 state axis by using the transpose function call rather than permute. Using Google

TABLE I

DQN HYPER PARAMETERS, INCLUDING DISCOUNT FACTOR γ, LEARNING RATE α, EXPLORE CHANCE ϵ, EXPLORE CHANCE DECAY PER TIME STEP $\Delta\epsilon$, MINI-BATCH SIZE PULLED FROM REPLAY BUFFER, REPLAY BUFFER SIZE, NUMBER OF TIME STEPS SPENT FILLING REPLAY BUFFER BEFORE BEGINNING TRAINING, NUMBER OF TIME STEPS TO PASS BETWEEN EACH TARGET DQN WEIGHT UPDATE, AND NUMBER OF EPISODES SPENT TRAINING.

Hyperparameter	**Value**
γ	0.99
α	$1.5e-4$
ϵ	1.0
$\Delta\epsilon$	$1e-5$
Batch size	32
Buffer size	10,000
Train start	5,000
Update target	5,000
n_episodes	20,000

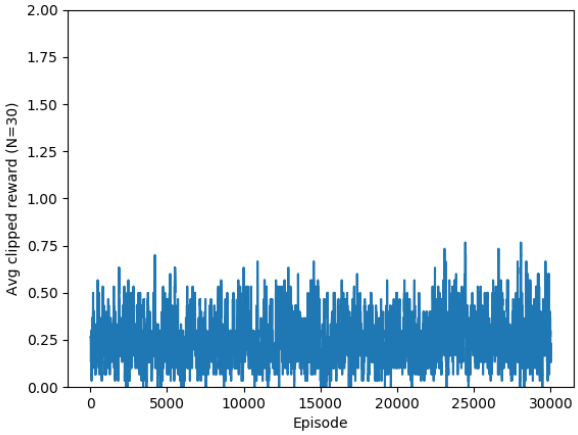

Fig. 3. Average clipped training rewards obtained by decaying $\epsilon-$greedy policy.

```
Episode:  158  of:  20000
Total time steps:  5000.0
Epsilon (chance to explore):  0.9500000000002276
Average reward/episode last 30 episodes: 0.16666666666666666
Episode:  319  of:  20000
Total time steps:  10000.0
Epsilon (chance to explore):  0.9000000000004551
Average reward/episode last 30 episodes: 0.26666666666666666
Episode:  464  of:  20000
Total time steps:  15000.0
Epsilon (chance to explore):  0.8500000000006827
Average reward/episode last 30 episodes: 0.23333333333333334
Episode:  599  of:  20000
Total time steps:  20000.0
Epsilon (chance to explore):  0.8000000000009102
Average reward/episode last 30 episodes: 0.3
Episode:  766  of:  20000
Total time steps:  25000.0
Epsilon (chance to explore):  0.7500000000011378
Average reward/episode last 30 episodes: 0.36666666666666664
Episode:  916  of:  20000
Total time steps:  30000.0
Epsilon (chance to explore):  0.7000000000013653
Average reward/episode last 30 episodes: 0.3
Episode:  1061  of:  20000
Total time steps:  35000.0
Epsilon (chance to explore):  0.6500000000015929
Average reward/episode last 30 episodes: 0.3333333333333333
Episode:  1212  of:  20000
Total time steps:  40000.0
Epsilon (chance to explore):  0.6000000000018204
Average reward/episode last 30 episodes: 0.23333333333333334
Episode:  1348  of:  20000
Total time steps:  45000.0
Epsilon (chance to explore):  0.550000000002048
Average reward/episode last 30 episodes: 0.36666666666666664
Episode:  1507  of:  20000
Total time steps:  50000.0
Epsilon (chance to explore):  0.5000000000022755
Average reward/episode last 30 episodes: 0.2
Episode:  1660  of:  20000
Total time steps:  55000.0
Epsilon (chance to explore):  0.4500000000022255
Average reward/episode last 30 episodes: 0.3
Episode:  1825  of:  20000
Total time steps:  60000.0
Epsilon (chance to explore):  0.4000000000021755
Average reward/episode last 30 episodes: 0.26666666666666666
Episode:  1959  of:  20000
Total time steps:  65000.0
Epsilon (chance to explore):  0.3500000000021255
Average reward/episode last 30 episodes: 0.43333333333333335
Episode:  2107  of:  20000
Total time steps:  70000.0
Epsilon (chance to explore):  0.3000000000020755
```

Fig. 4. The print statements for 2,000 episodes of training, at the same frequency as target DQN weight updates. ϵ decays as expected, while episode reward averages vary between $\{0.1, 0.7\}$.

Cloud, a TeslaK80 GPU, 4 vCPUs, and 15 GB memory, training was executed over 12 hours (Fig 3).

At every target DQN weight update, the current ϵ, episode, time step, and average reward over the past 30 episodes is displayed to aid in diagnostics (Fig 4).

III. TESTING

In testing, the chance to explore is set to zero $\epsilon = 0.0$ such that a greedy policy is executed, maximizing rewards in the 100-episode duration by leveraging the learned weights. The constructor of the Agent_DQN class loads the trained weights to the policy model, and the make action method is called over and over for 100 episodes, yielding the average reward per episode of 0.18, well below the required 40.

IV. CONCLUSIONS

The cause of the low performance is unknown, however it appears that the rewards vary in a way that suggests actions taken are random. Verification of all objects, tensors, and methods was performed, which leads me to the conclusion that the values returned by the DQN forward pass are random in some way, or at least never improve with training, as the make action method is functioning

as intended, and ϵ decays as intended. This is confusing, as all reccommended hyper parameters were used. Some variations were made in the time given, including changing the optimizer from Adam to SGD, but non yielded an increasing trend to reward averages in training or test rewards.