Toward a Deep Reinforcement Neural Network Capable of Playing Breakout

1st Adam Mischke
Computer Science Undergraduate
Middle Tennessee State University
Murfreesboro, TN, USA

2nd Patrick Howell
Computer Science Undergraduate
Middle Tennessee State University
Murfreesboro, TN, USA

3rd Eric Sciullo
Computer Science Undergraduate
Middle Tennessee State University
Murfreesboro, TN, USA

4th Kevin Lutz
Computer Science Undergraduate
Middle Tennessee State University
Murfreesboro, TN, USA

5th Benjamin H. Buehler
Computer Science Undergraduate
Middle Tennessee State University
Murfreesboro, TN, USA

6th Garrett Coulter
Computer Science Undergraduate
Middle Tennessee State University
Murfreesboro, TN, USA

***Abstract*—Since Google DeepMind conquered Atari video games have become a standard environment to measure a Deep Q reinforcement learning network agent. We implement a DQN agent to play the classic Atari game breakout. Unlike the similar the Google DeepMind team used, in which an agent was taught to play multiple Atari 2600 games from a few training games, we failed to achieve human-like performance. We believe this to be due to a combination of factors including the time which the network trained and the computational resources at our disposal.**

***Index Terms*—deep learning, Atari, Breakout, reinforcement learning, neural nets, DQN Agent, Q function, artificial intellegence, AI, python, Open AI Gym, Arcade Learning Environment**

I. Introduction

The goals of this project were to build and train a neural network capable of playing the Atari 2600 game, Breakout, and to observe the learning process of the network. Videogames are a well known and appreciated topic in the computer science community and they are also an excellent platform for exploring the topic of training neural networks. [1] [2] These are the most likely reasons for the existence of the wealth of knowledge and tools available on the topic of teaching neural networks to play videogames in partially obstructed environments - i.e., the game learns from the same data a human player would have while playing. Training networks to play in these partially obstructed environments affords the opportunity to watch a network learn to do a task with which the creators are probably intimately familiar. Familiarity with the task makes the learning process significantly less abstract, especially when compared against something like learning to sort pictures or recognize language. The videogame playing network also makes it easier to explain the process to new or even non computer scientists, because the typical language of reward and punishment, or any of the other human related motivational terms typically used to explain neural networks start to make more sense. For example, describing a network, playing a shooter, as avoiding the suffering of death, or negative reward values, by running away from enemies, until it learns it has a weapon it can use. Because we understand the process of learning how to play a game, we can better understand the process a neural network goes through when learning to play a game.

II. Background

To develop this neural network we used the OpenAI Gym toolkit for integrating our reinforcement learning with Breakout. We used OpenAI Gym because it has a common interface for Atari games. [3] Reinforcement learning takes a game state and predicts which action is optimal given previous experience, trying to optimize a reward function. [4] The Reinforcement learning algorithm's goal is to increase the reward to the agent as it interacts with the environment it is placed in. [4]

Because previous experience determines the expected reward and unexplored game states are assumed to lead to a reward of zero, an agent could easily get stuck in a loop if it were to pursue the maximum expected reward for every state. [4] The agent must have some mechanism to break out of this loop. [4] This problem has been called the multi-armed bandit problem. [4] Supposing an agent is at a casino in front of a row of slot machines with n quarters with the probability of payout for each machine unknown, what should a rational agent do? [4] An agent maximizing the expected reward would pick a random slot machine until one of the machines paid out and would then play that machine until they lost more money than their initial first attempt. [4] Similarly, an agent picking random slot machines could be picking the 'duds' more often than necessary. [4] Our solution to this problem, commonly used by implementers of reinforcement learning algorithms, was an ϵ-greedy approach. [5]

The learning the agent goes through can be broken down into sections or episodes. [4] This is known as episodic reinforcement learning. With this the agents is placed at an initial state and proceeds until it reaches a terminal state or the end. [4]

Both our group and Google DeepMind used a form of called DQN reinforcement learning. [5] A Q^{π} function, also called an action-value function for policy π, maps from the cross product of the set of all possible states in the problem and the set of possible actions at that state to the expected reward when the policy (rule for picking actions) π is followed. [4] Q^{*}, also called the optimal action-value function has the same domain and codomain but returns the expected reward if the agent followes the optimal policy throughout the rest of the episode. [4] The function of the neural network in DQN reinforcement learning is to use its experience to approximate this optimal action-value function. [5] Once an approximation is found (assuming the number of possible actions at any given time is low) picking the correct action for the agent is trivial because all that is needed is to pick the action a' for which the Q^{*} value is highest for every state s. [5] [4]

In our case the environment our agent was placed in was the Atari 2600 game Breakout. Breakout is a game that was inspired by Pong. In this game there are several layers of bricks on the top portion of the screen and a moveable paddle on the bottom of the screen. The agent releases the ball allowing it to bounce accordingly from the paddle to the walls and hitting a brick. When a brick is hit it is destroyed and gives the agent points. The agent loses a life when the ball misses the paddle and lands at the bottom of the screen. When the agent loses all lives the game ends.

In our case when the agent loses a life/the ball hits the bottom of the screen it is reset while it is rewarded only when a brick is destroyed. There is no punishment for this cycle, as our intentions were for it to learn to keep the ball in play. The longer that it did so, meant that the ball would learn to continue to do so, only being rewarded for good episodes. The Open AI Gym wrapper was observed to only return a reward of 1 when a brick was destroyed.

III. Methods

A. Environment

We used the BreakoutNoFrameskip-v4 in Open AI Gym, and implemented our own frameskipping, repeating the same action 4 times in a row for those skipped frames as shown in Figure 2 (**??**). We then also tryed another environment, BreakoutDeterministic-v4 that does the same, except implemented by Open AI Gym. [3] The Atari environments in Open AI Gym use the Arcade Learning Environment. [3] [6] Open AI Gym and the Arcade Learning Environment provide a useful way of accessing game data as input for a neural network. [3] [6] We accessed only pixel data from this environment and not internal game states.

B. Input Processing

We downsampled the images to 84x84 then converted to grayscale (0-255) from a 210x160 rgb image in a similar manner to what was done by Mnih et al. for their DQN network. [5] We would train on normalized (0.0-1.0) float32 image data, but would store the images in as uint8 to reduce memory footprint, as 1,000,000 size numpy arrays are huge. With discrete framing, meaning 2 seperate 4 frame windows of array size (1,000,000, 84, 84, 4) for current and next frame, the memory taken up would be around 65Gb. As for a more efficient frame sliding technique, where we store one (1,000,000, 84, 84, 5) array with the first four channels being current frame and the next four being next frame, the memory taken was around 45Gb. Additionaly, with frame skipping, which is a common way to decrease training time in video game playing agents. [5] [1] We discretize the input by packaging four frames into the neural networks input data structure as discussed by Max Lapin in his Medium Article Speeding up DQN on PyTorch: how to solve Pong in 30 minutes. [7]

C. Topology of the Net and Loss Functions

Our neural net uses three convolutional layers to process visual features in the input pixels. They have a kernel of 8x8, 4x4, and 3x3 respectively. The net is then flattened out and two dense layers follow with 512 neurons and 3 neurons respectively. ReLU is used for all the layers except for the last which is linear for the 3 output buttons. The three most commonly used loss functions were implemented: the Logarithm of the Hyperbolic Cosine, Mean Squared Error, and Huber loss.

D. Reinforcement Learning Algorithm

We used a reinforcement learning approach to create our agent. This involved using DQN networks to approximate a Q^{*} function in a similar manner to Mnih et al.. [5] Additionally, we employed an ϵ greedy exploration approach. Our agent uses an offline learning method with replay memory in order to prevent catastrophic forgetting.

Training

We were able to train the agent for almost 82 hours (294661 seconds). This was well after ϵ reached its minimum during training. Because as ϵ approaches 0 the probability of choosing the action with the highest Q value increases, exploration had diminished by this point. [4] We implemented a linear annealing ϵ that started at 1.0 and ended at 0.1 for 1 millions frames as per the Deepmind paper. [5]

.. code:: ipython3

IV. Results

After training for over 3 days and 1.75 million frames the network was still unable to do much more than track the ball from the start position. Observing the data from different training sessions, one may note the lack of consistency. This is due to the ϵ value starting at 1, or more exploration, and slowly decreasing to 0.1, or less exploration, over the course of the first million frames. This can be seen in Figure 1 (III-D) in how there is a great degree of variation, by design, from the beginning of the training session up to around 4500 episodes. The first episode to go over a million frames is episode 4289. One may observe that shortly after that the core of the reward values tightens up, and while the core, or most frequent values

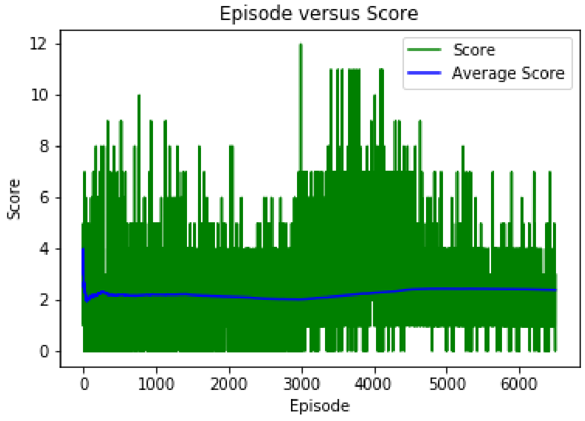
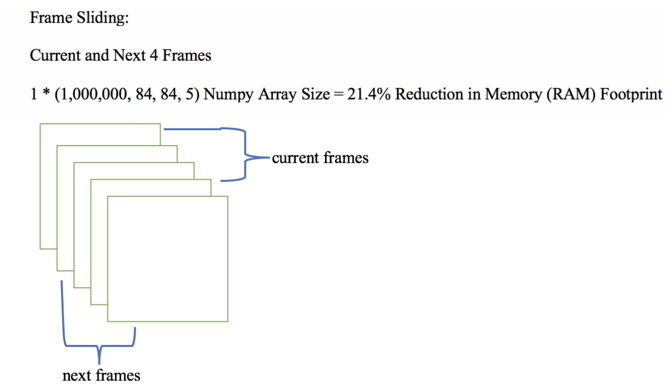


Fig. 1. Training Data


Frame Sliding:

Current and Next 4 Frames

1 * (1,000,000, 84, 84, 5) Numpy Array Size = 21.4% Reduction in Memory (RAM) Footprint

Fig. 2. Custom frame skipping method.

of the graph decrease, they become more consistent. In the last third of the graph the bulk variation goes from between 1 and 7 for episode 3000 to 4500, to 1 and 4 after. This shows that the network is acting in a more consistent manner. In Human-level Control through Deep Reinforcement Learning, the network was trained for 38 days worth of playing time and 50 million frames. [5] While that network was being designed to learn how to play any Atari 2600 game, given the relatively large amount of variance caused by the speed of the ball based on what surfaces it contacts and the variation in the angle of the bounce off the paddle based on the position of the paddle the ball strikes, 1.75 million frames is a fraction of the necessary training time required to train a network to play at a human level. Even when considering that our network is only training to play one game.

V. Discussion

While expected, the results of training the neural network were quite interesting to view first-hand. We understood that the rewards would be much more randomly distributed for the first million frames, but when ones prior experiences include mostly training lasting minutes, one tends to expect results when the network is given a training period spanning multiple days. The amount of time our network trained to gain what little proficiency it had, draws attention to how incredible it is that our minds are able to take our previous experiences and the output of the game and solve the problem so quickly. We as humans are able to draw on our previous experiences and even to apply them holistically. We do these studies to better understand understand not only how teach a network, but also to better understand how humans try to solve problems at a fundamental level. Having the technology to teach a neural network to play a game at a superhuman level, with enough training, is the first step in teaching the network to apply what it knows about one game to playing another game, or eventually, teaching it to apply skills that are only tangentially related to new problems. However, within the scope of this project, what might be an interesting direction would be growing the capabilities of the network to eventually include games beyond the Atari 2600. Obviously, changes would need to be made to the way input is received and processed, but what if the network could recognize patterns in similar game design and apply it to more modern games?

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, December 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[2] A. Agarwal, "Explained simply: How deepmind taught ai to play video games," article, August 2017. [Online]. Available: https://medium.freecodecamp.org/explained-simply-how-deepmind-taught-ai-to-play-video-games-9eb5f38c89ee

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. MIT Press, February 1998, oTHER URL https://mitpress.mit.edu/books/reinforcement-learning. [Online]. Available: http://incompleteideas.net/book/ebook/the-book.html

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling", ""the arcade learning environment: An evaluation platform for general agents"," *CoRR*, vol. abs/1207.4708, 2012. [Online]. Available: http://arxiv.org/abs/1207.4708

[7] M. Lapan, "Speeding up dqn on pytorch: how to solve pong in 30 minutes," article, November 2017. [Online]. Available: https://medium.com/mlreview/speeding-up-dqn-on-pytorch-solving-pong-in-30-minutes-81a1bd2dff55