Playing Pong with Deep Reinforcement Learning

Bobby Dorward, Ryan Wilson, and Eric Bell

December 15, 2016

1 Introduction and Background

The history between artificial intelligence and gaming is long standing because short term choices give rise to long term sophistication. For example, the moves possible in the game of Go are few, yet it took one of the most sophisticated machine learning algorithms to date to be able to perform sufficiently at this problem. We aimed to revisit this relationship between machine learning and gaming in order to gain facility with how machine learning, specifically deep neural networks, are used.

We have trained our neural network through reinforcement learning, specifically through an implementation of Q-learning. [BOBBY PLS WRITE THIS SECTION SO ITS THE MATHINESS YOU WANT]

Our algorithm interacts with the game through the OpenAI Gym, an environment that allows the programmer to interact with Atari games (as well as a few other simulations) such that a machine learning algorithm can be trained to play the game ("solve" the problem). What this entails is that the Gym allows us to extract pixel data from the game, signal actions for the AI to take on a frame-by-frame basis, and collect feedback used to train the algorithm. Our original goal when starting this project was to apply this network to the "DoomBasic-v0" but due to limitations, we decided to start with the simpler "Pong-v0" environment. This environment runs a typical game of pong: a ball bounces between two paddles and one player scores when the ball hits the opposing side of the screen. The two players continue until someone scores 21 points, after which the game ends.

2 Data

As previously mentioned, the data we used as input into our deep network was the pixel data of the environment. Each pixel is represented as an array of three elements, representing the three channels of an RGB pixel. Neural networks by nature are memory-intensive data structures due to the large amount of nodes and connections they contain. This is especially true when one uses a more complex derivative of a neural network, such as the deep neural networks we have implemented. Therefore, despite our best efforts to make our implementation generic and applicable to many different environments, we had to make pong-specific alterations to our data to streamline the size of the network and make it runnable on the resources we had available to us.

The first change we made was to grayscale the pixel values, thereby cutting the data we need to put into our network by three. Essentially what this means is that we have averaged the three values of the RGB channel into one value for each pixel. Next, we cropped the images being given to us by the OpenAI Gym in order to exclude all the pixels on the screen that weren't directly relevant to the frame-by-frame progression of the game: the scoreboard, the white edges, and the padding behind each paddle was cropped to minimize the input count of the network. (Downsampling/Pooling?) Through this preprocessing, we were able to reduce the amount of inputs being taken for each frame to 20,800.

Finally, since the game of pong does not consist of still images, but rather gains meaning through the motion of the game objects (two paddles and the ball), we needed to capture this motion using the pixel data. We achieve this by extrapolating a difference frame between each subsequent pair of images, in which we subtract the first image from the second image. Through this, we can train the algorithm with data that represents where the game objects have been and where they're going, zeroing out any unmoving (and thus irrelevant) pixels in the process. It is because of these difference frames that we have limited the action that our AI can take; instead of the 5 actions that are available by default to the algorithm, we've limited it to only moving left and right. This way, the algorithm is always able to see its own paddle because it's constantly in motion and therefore creating data on the difference frame.

3 Implementation

Line by line explanation of what the code does and looks like, how we realized Qlearning (decaying rate, epsilon, etc.)

4 Results and Discussion

Talk about topology/talk about problems the algorithms having and why that might be the case

5 Conclusion