Independent Study

Report-2, 31st January

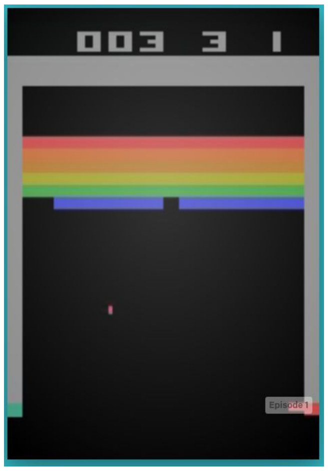
Submitted by: Vaibhav Garg(20171005), Kartik Gupta(20171018)

Chosen RL Problem

We searched for our RL problem mainly with two goals in mind:

- The problem should be fairly popular within the RL research community with an existing State of The Art "deep-net/black-box" model solution which can serve as our neural oracle in the NDPS algorithm.
- The problem should be simple enough so that the generated program is easier to interpret, not very long, and does not lead to very complicated search optimization.

After an extensive search and discussion, we decided to go ahead with the seven classic Atari games { *B. Rider, Breakout, Enduro, Pong, Q*bert, Seaquest, S. Invader* }. Reference paper for their benchmarks: http://www.cs.toronto.edu/~vmnih/docs/dqn.pdf. These games are fairly simple with easily available environments in Python.



To begin with, we plan to focus only on the Breakout game, the goal in this game is to break all of the bricks without letting the ball fall away from the board. There are only two actions available for the agent, either to move to the left or to move to the right.

Observations

Our agent will operate with the following observations

1. Positions of the bricks as 2 vectors (x, y coordinates)
2. Position of the ball (x, y coordinates)
3. Position of the paddle (x coordinate)

Actions

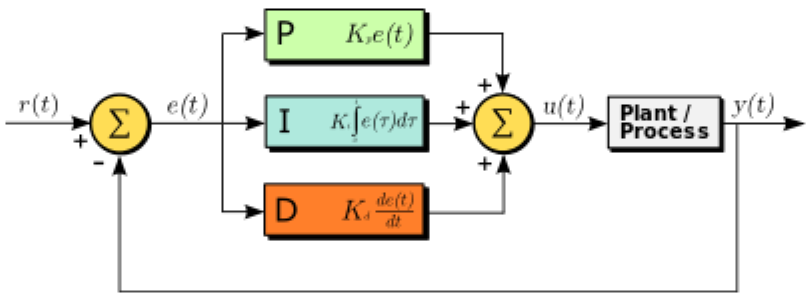
Our agent can perform the following actions:

1. Move left
2. Move right

Since we only have 2 actions, we use a boolean output to control it.

Sketch

We have decided to use PID based control systems as the basis for our program generation.



They're used widely in industries and are often a good and simple enough starting point for any feedback based control loop.

DSL

Our DSL consists of

1. **Atoms**
   a. Scalar or Vector constants
   b. Scalar of Vector variables
   c. Scalars can be floats or booleans
2. **Operators**
   a. +, -, *, /
   b. and, or, not
   c. >, <, >=, <=, ==
   d. = (assignment operator)
3. **Higher Order Combinators**
   a. Map(f, [e_1, e_2, ...]) = [f(e_1), f(e_2), ...]
   b. Filter(f, [e_1, e_2, ...]) = [e_{f1}, e_{f2}, ..] where f(e_{fi}) is true
   c. Reduce(f, [e_1, e_2, …]) = f(e_1, f(e_2 f(e_3 …)))
4. **Access Operations**
   a. Head([e_1, e_2…e_k]) = e_1
   b. Tail([e_1, e_2…e_k]) = [e_2, e_3, ...e_k]

Some semantic rules are:

1. All scalars are essentially 1 element vectors
2. Type checking to ensure that correct operators apply on applicable operands.

Context Free Grammar

We came up with the following rough CFG for our expected solution, based on the PID controller structure:

```
P -> Head(eps - h_i)
I -> Reduce(+, h_i)
D -> Head(h_i) - Head(Tail(h_i))
C -> c_1*P + c_2*I + c_3*D > 0
B -> c_0 + c_1*head(h_1) + c_2*head(h_2) ... c_k*head(h_k) > 0
      | B and B | (not B) | B or B
E -> C | if B then E else E
```

h_i represents the history of observations from the ith sensor.
Since the output is binary, we return a boolean by comparing the PID sum with 0. If this is +ve, the agent will move right, else left.

Oracle

We will be using a DQN as an oracle since DQN provides some of the best performance in the given environment. Since our observation space is different from standard implementations, which use images of the game as the input, we will be training a model from scratch.

Next Phase

Our aim for the next phase is

1. Obtain a decently performing oracle for the given problem.
2. Look into alternative controllers from control theory, for suggesting more sketch alternatives.