CS221 Project Proposal: Breakout Bot

Priyanka Rao, Reid Pryzant, Vincent-Pierre Berges

Introduction

The challenges of applying reinforcement learning to modern AI applications are abundant, particularly in unknown environments in which there are delayed rewards. In this project, we aim to teach a computer agent how to play a custom implementation of the popular arcade game Breakout, in which a paddle is moved horizontally to bounce a ball and break bricks. Breakout has a rich state space and is aptly suited for a reinforcement learning (RL) approach. By applying variants of the well-known Q-learning algorithm, we seek to deepen our understanding of the problem and solution space.

Input / Output / Scope

The input provided to the agent is a vector representation of the game state. This may consist of features like the following:

- paddle position
- ball position
- distance between the ball and paddle
- position of remaining bricks
- paddle direction
- ball direction
- indicators on the existence of every brick
- number of bricks on the board
- ball velocity
- paddle velocity
- current game score
- time game has been in session

Refer to the appendix for an example state vector. The values for the features above can be extracted from a series of frames capturing the state of the board at small differentials in time. Given a state representation of the board, the agent will output the optimal horizontal movement of the paddle with the goal of maximizing the future game score. At this time, we intend to train the agent to play a constrained game of breakout with one life. Time permitting, we hope to introduce random placements and varying numbers of bricks on the board. We may also increase the number of lives allowed in the game.

Evaluation Metric

The primary metrics that we will use are total score at the end of a game and the time taken to achieve that score. We will compare the score and time taken for a game to that of an oracle, baseline, and previous games. We may also make our evaluation metric a function of both score and elapsed time in an effort to encourage speedy play.

Baseline and Oracle

The current baseline follows a single rule: move the paddle towards the ball. Given that the speed of the paddle is limited, this solution is sub-optimal and we expect to surpass it with a learned agent. The oracle operates similarly to the baseline but its speed is not rate-limited - its x coordinate is set exactly to that of the ball. The oracle cannot lose and will eventually destroy all bricks, but makes no “smart” decisions like aiming the ball. Though both the baseline and oracle fail to strategize, there exists a gap in performance: the baseline smashes only a few bricks while the oracle wins every time. Our goal is to learn an agent which can close this gap to match or surpass the performance of the oracle. Please refer to the Appendix for examples of inputs and outputs to the agent.

Approach

We plan to take a model-free, off-policy RL approach because any breakout-playing agent lacks explicit access to the game's reward and transition functions. The agent is tasked with maximizing its expected utility by taking actions but has no clue as to the implications of these actions and lacks any representation of the environment it inhabits. We choose model-free because our environment may be known (we implemented the game so we know all its rules), but is too large to model directly. On-policy methods like $SARSA(\lambda)$ offer a tantalizing middle ground between the high variance model-free MC estimate and biased bootstrap estimates like $SARSA$. However, many on-policy methods are prohibitively slow in our setting and off-policy methods give us the ability to reuse experience generated by old policies. Additionally, though we could use on-policy TD learning because games' state transitions are deterministic, we wish to build a more general-purpose learner that could adapt to modified rules (e.g. if we injected some randomness into ball deflection).

The primary goal of our project is to implement and compare several variations of Q-learning. This is a model-free, off-policy approach where the estimated utility of being in a state and taking an action is updated towards a bootstrap estimate of the true utility. There are some issues with this algorithm. If we always take actions on a greedy basis, than we will fail to adequately explore the state space. We ameliorate this by introducing a ϵ-greedy acting policy. Second, our state space is too big to feasibly explore. We will explore two solutions to this challenge. We will discretize our state features and use these values as a reduced resolution state that is feasible for traditional Q-learning, and we will use function approximation to adapt the algorithm to continuous state features.

Q-learning with function approximation will be the main thrust of our project. We will compare two different function appropriators: neural networks and logistic regression. Because both are differentiable, we may use stochastic gradient descent (SGD) to update our parameters at every step. There remain two challenges to overcome. First, game trajectories (our "training data") are highly correlated. We correct for this by making use of experience replay. This involves caching a record of agent experience and sampling from this experience at each time step. This both decouples game trajectories and increases our data efficiency. Second, Q-learning uses the freshest weights to calculate its bootstrap target, which is highly unstable. We will correct for this by periodically freezing an auxiliary set of weights and using these for target calculation.

Related work

Several research groups have explored the application of the RL approach to games such as Flappy Bird, Tetris, Pacman, and Breakout. Perhaps most well-known is the project undertaken by Google DeepMind in 2015, in which a convolutional neural network was combined with deep Q-learning and experience replay to train an agent how to play Breakout [1]. The state representation used by DeepMind was based on pixel values and the current game score; this is different from the state representation we intend to use. While DeepMind demonstrated that an agent can successfully learn to play and even beat humans, there is no existing comparison of the challenges and performance benefits of multiple RL approaches for Breakout.

Appendix

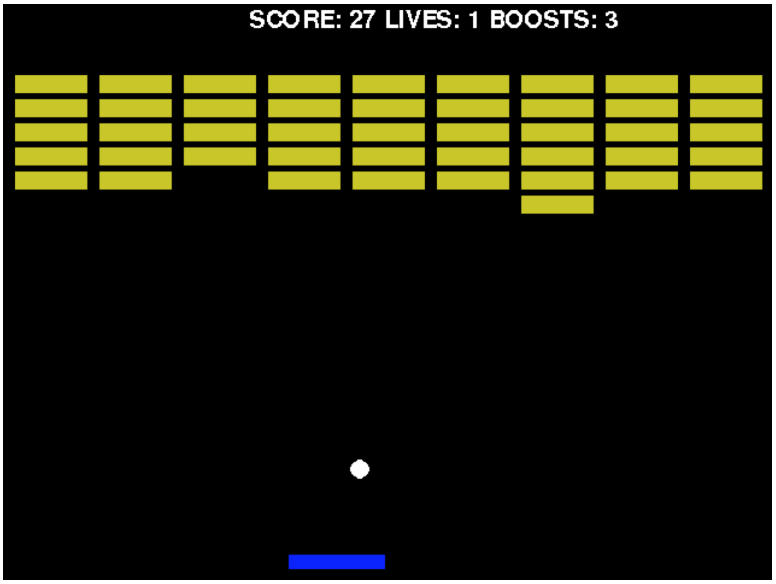

Figure 1: Oracle playing

With the current implementation of the game, the baseline scores 15.0 points in 316 frames and leaves 49 bricks. The oracle scores 162 points in 1416 frames and destroys all the bricks. And example of state in the game could look similar to this :

```
state = {
    'game_state': 'STATE_PLAYING',
    'ball_x':112,
    'ball_y':30,
    'ball_vel_x':5,
    'ball_vel_y':-8,
    'paddle_x':18,
    'time': 28,
    'n_bricks': 2,
    'd_ball_paddle':98.67
    'score': 167
    'bricks_positions':[(1,2),(3,6)]
    }
```

In this situation, the ball is going up and the ball is on the right of the paddle. The response of the agent could be *[INPUT_R]* which means the paddle will then move right. In this configuration, the baseline agent would indeed move the paddle right since $ball\_x > paddle\_x$.
In another situation :

```
state = {
    'game_state': 'STATE_BALL_IN_PADDLE',
    'ball_x':45,
    'ball_y':0,
    'ball_vel_x':0,
```

```
    'ball_vel_y':0,
    'paddle_x':45,
    'time': 0,
    'n_bricks': 10,
    'd_ball_paddle':0
    'score': 167
    'bricks_positions':[(0,0),(0,1),(0,2),(0,3),(0,4),(1,0),
              (1,1),(1,2),(1,3),(1,4)]
    }
```

In this situation, the ball has not left the paddle yet and the game consist of 10 bricks. The baseline model and the oracle would both launch the ball, so the response is *[INPUT_SPACE]* for both agents. One last example :

```
state = {
    'game_state': 'STATE_PLAYING',
    'ball_x':62,
    'ball_y':16,
    'ball_vel_x':1,
    'ball_vel_y':3,
    'paddle_x':62,
    'time': 23,
    'n_bricks': 1,
    'd_ball_paddle':16
    'score': 167
    'bricks_positions':[(1,3)]
    }
```

In this situation, the ball is right above the paddle. The speed of the ball is quite low so the baseline could follow and remain under the ball. The same goes for the oracle but it could have achieved the same result regardless of the speed of the ball.

References

[1] Mnih, Volodymyr et al. "Human-Level Control Through Deep Reinforcement Learning". Nature 518.7540 (2015): 529-533. Web.