BroadMind: A Better Platforming Agent

Abstract

Recent work in reinforcement learning have focused on building generalist video game agents, as opposed to focusing on a particular type, or genre of games. We aim to build a more specialized high-performance agent focused on the more challenging genre of platform games, which has received less attention.

1. Background

Platform games, also known as platformers, are video games in which players guide a free- running avatar to jump between suspended platforms and avoid obstacles to advance through levels in the game. As a result of the array of environments to parse and the huge decision space for strategies, these games are very difficult for machines to play well. Instead of intuiting all levels of gameplay from first principles, we aim to build an agent that breaks down the problem as a human player would.

2. Approach

2.1. Experimental Setup

2.1.1. RL-Glue

RL-Glue is a socket-based API that enables reinforcement learning projects to work across multiple languages (Tanner & White, 2009). RL-Glue applications are made up of a core communications process, an agent, an environment, and an experiment. This allows our experiments to connect reusable Python agents across many environments written in languages including C++ and Java. It also enables us to customize our experiments, such as optional game visualization, loading and saving trained policies, adjusting the game difficulty, etc.

2.1.2. Generalized Mario

Although our goal is to train agents to play Atari 2600 platforming games, it is easier to learn from a platform game that provides more layers of environment representations than raw screen pixels. To do this, we have started with the Generalized Mario environment that was contained in the 2009 AI Competition software package, and conveniently RL-Glue compatible (Togelius et al., 2010).

The Generalized Mario game has a total control input space of 12. This is made up of (-1, 0, 1) for left, none, and right motions respectively, on/off for jump, and on/off for run. We have used an encoding function that maps the integers (0-11) to and from these control input combinations.

Generalized Mario's interface provides many layers of state observation. It includes the (x,y) position and velocity of the mario actor, as well as a 22x16 tile grid semantically describing the screen space with the location of coins, pipes, blocks, etc. Separately, it has a list of all enemy positions on-screen, with similar position and velocity information as provided about Mario.

Our research aims to determine how to intelligently encode platform game state information. We are investigating ways to represent the environmental observation provided by Generalized Mario.

2.1.3. Arcade Learning Environment

We have leveraged the Arcade Learning Environment, an open-source and RL-Glue compatible framework built on top of an Atari 2600 emulator (Bellemare et al., 2013). It provides an API for interfacing with the raw pixels, the current score, and the controller input of the game. This has allowed us to use original Atari games to train and evaluate our agents. We currently support the following Atari platformers in our experimental setup:

1. Kung-Fu Master
2. Frostbite
3. Kangaroo
4. Pitfall!
5. Pitfall! 2
6. H.E.R.O.
7. Montezuma's Revenge

We plan to extend our Mario work to see if we can learn a similar state representation from the raw pixel inputs of ALE for these platform games.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2.2. Learning Algorithms

2.2.1. State Representation in Mario

It is challenging to find a powerful representation of the Mario game state that enables effective Reinforcement Learning. The environment observations provided by Generalized Mario contains a wealth of symbolic information, but there are many ambiguities that we are investigating. For example, what coordinate frame should be used?

Currently, our representation is a tiled grid of integers, 20x12 tiles in size. The relative value in each tile is given as a "measure of goodness", enemies are -2, obstacles are -1, coins are +2, etc. The grid is centered on Mario's current location. We intend to represent the state with separate substrates for each class of objects, as in (Hausknecht et al., 2013), by break it out into separate background, enemy, reward, and actor layer. We hope to find that this representation will be universal across platform games.

None of these representations encode the important velocity information about the actors of the game, which is important in platformers where the characters move with some inertia. We may attempt an alternative approach that encodes this information, which is available in the Mario environment.

2.2.2. Neural Q-Learning

Typically, Q-Learning approaches use a table representing the Q-function. This table contains a value for every combination of state and action. However, for very large state/action spaces such as platforming games, this is impractical as the space would take too many trials to explore and converge on an optimal policy. Even using optimizations such as a Nearest Neighbor algorithm ran out of memory in our approach (spaces greater than 32 are not permitted, compared to at least 1,000+ for this application).

We have implemented a neural-network based Q-learning algorithm (see Algorithm 1) to allow us to learn on large state/action spaces with reasonable memory utilization by finding a useful hidden layer. Inspired by DeepMind's approach (Mnih et al., 2013), we have avoided multiple forward propagation steps when selecting optimal actions by using only the state as the input to the network, and a weight for each action as the output. Thus, we can select an optimal action for a state by propagating once, and selecting the max argument from the outputs. Also like DeepMind, we include an Experience Replay step that stores a history of events to re-learn. This avoids overfitting to the current situation and unlearning good behavior from earlier episodes. We are actively investigating methods to store experiences more intelligently. We can optionally use the standard Q-Learning update, or the SARSA calculation.

Algorithm 1 Neural Q-Network with Experience Replay

```
Initialize Neural Q-Network with random weights
for episode= $1, m$ do
  Initialize previous state $s_0$ and previous action $a_0$ to NULL
  for $t = 1, T$ do
    Observe state $s_t$ from the emulator
    With probability $\epsilon_a$, select random action $a_t$
    Else, set $a_i = \max_a Q(s_t, a)$ by forward propagating $s_t$ through Q
    if $s_{t-1}$ and $a_{t-1}$ are not NULL then
      Observe reward $r$ from emulator
      With probability $\epsilon_r$, store the experience $\{s_{t-1}, a_{t-1}, r, s_t, a_t\}$.
    end if
    for re-experience= $1, ex$ do
      Randomly sample experience $\{s'_0, a'_0, r', s'_1, a'_1\}$
      if using SARSA update rule then
        Compute $v = r' + \gamma Q(s'_1, a'_1)$
      else
        Compute $v = r' + \gamma \max_a Q(s'_1, a)$
      end if
      Update Q through backpropagation of value v on output $a'_0$ with state $s'_0$
    end for
    Apply action $a_i$ to emulator
    Update state $s_{t-1} = s_t$ and action $a_{t-1} = a_t$
  end for
end for
```

We have found that the need to assign initially random weights to the neural network can make the initial policies very flawed. To counter this, we plan to use heavy exploration bias (high ϵ) for early episodes, while transitioning to exploitation policies (low ϵ) at later episodes.

2.2.3. Extension to Atari Platformers

We have setup the ALE environment, and connected default agents to it. However, we have yet to attempt to port our Mario agents to these problems. Initially, we will use 3 colored pixel substrates as the state representation, as in (Hausknecht et al., 2013), but we hope to reconstruct an object layer as well.

Acknowledgments

None.

References

Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment:

An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Hausknecht, M., Lehman, J., Miikkulainen, R., and Stone, P. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 2013.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2013.

Tanner, Brian and White, Adam. Rl-glue: Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10: 2133–2136, 2009.

Togelius, J., Karakovskiy, S., and Baumgarten, R. The 2009 mario ai competition. *Evolutionary Computation (CEC), 2010 IEEE Congress on.*, pp. 1–8, 2010.