Replacing Plumbers with AI:

Using Q-Learning in *Super Mario Bros.*

Marilu Bravo | Sean Decker | CS221 | Stanford University

abstract

The stout, red-hatted plumber Mario has since 1981 monopolized the industry of saving Princesses from overly intelligent turtles/apes while collecting oversized coins. This monopolization has gotten so out of hand that Mario's parent organization, Nintendo, is today Japan's third largest industry and also a major shareholder in the MLB's Seattle Mariners. In any such overly dominated industry, there is possibility for disruption, thus this project aims to disrupt the savior plumber industry through the introduction of AI, to see if the job of Mario could be automatized. More specifically, we are using an ϵ-greedy Q-learning algorithm with feature extraction and SARSA to train our agent to travel the furthest distance in *Super Mario Bros.*

overview

- We are using an ϵ-greedy Q-learning algorithm with feature extraction and a SARSA algorithm in order to learn to play Mario
- We model the game of *Super Mario Bros.* as a state-based model with defined actions and rewards that incentivize certain actions over others

model

- We abstracted the game of *Super Mario Bros.* as an MDP where nodes represent positions Mario takes in the game and edges represent Mario's possible actions

STATES

- Each state node is described by
  - a 16x13 array of tiles (figure 1) representing the current screen frame of the game
    - Each tile holds one of the following values:
      0: empty, 1: object, 2: enemy, 3: Mario

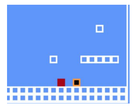

figure 1

ACTIONS

- An action is represented by a length-6 array of booleans:
  Action = [up, left, down, right, A, B]
  where A and B are controls on the NES controller that allow Mario to jump and dash/throw fireballs, respectively
- Buttons can be pressed simultaneously, giving 2^6 = 64 possible actions
- We have 2 different models for actions:
  - No bias - all actions are allowed in this model
  - Right bias - this is a relaxed version of the game. Because Mario in general wants to move to the right, this model improves the score of the game (at least in level 1)

REWARDS

- Mario is rewarded for positive change in distance from the start

algorithm

SARSA

- We choose an algorithm for SARSA based initially on randomized actions, then it fine tunes its policy using:

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta[r + \gamma\hat{Q}_\pi(s',a')]$$

- SARSA then chooses the best policy out of the initial randomized actions
- Feature extraction is not used. States action pairs are completely described by the layout of the tiles and Mario's position

algorithm

Q-LEARNER

- We are using an ϵ-greedy Q-learning algorithm with feature extraction
- The Q-value is updated by definition as:

$$Q_{opt}(s_t,a_t) \leftarrow (1-\eta)Q(s,a) + \eta(r + \gamma max_{a\prime \in Actions(s\prime)}(Q_{opt}(s\prime,a\prime)))$$

- At each state, the agent will choose actions governed by the following policy:

$$\pi_{\text{act}}(s) = \begin{cases} \arg\max_{a\in\text{Actions}} \hat{Q}_{\text{opt}}(s,a) & \text{probability } 1-\epsilon, \\ \text{random from Actions}(s) & \text{probability } \epsilon. \end{cases}$$

- Our weight vector, *w*, will be initialized as 0 at the beginning, and will be updated by the definition:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\hat{Q}_{\text{opt}}(s,a;\mathbf{w}) - (r + \gamma\hat{V}_{\text{opt}}(s'))]\phi(s,a)$$

- φ(s,a) represents an abstraction of the state of the 16x13 array of tiles based on Mario's relative distance to tiles

results

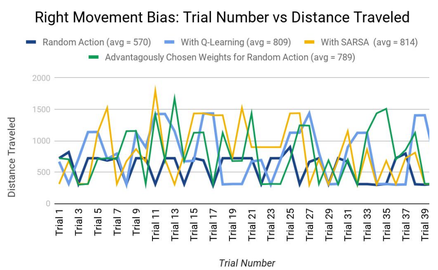

figure 2

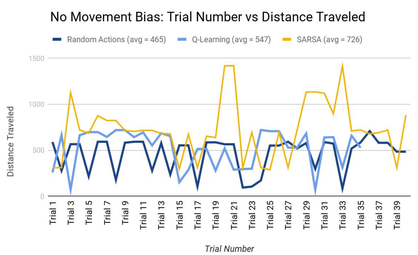

figure 3

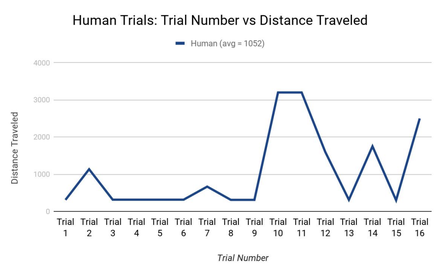

figure 4

Controls:

- Random action policies
- An Advantageously Weighted Action Policy (an attempt to hard code a good policy for Super Mario)
- Human Players

discussion

- We see that our Q-Learning Algorithm performs better than random actions, with and without a right lean on Mario's movements
- Even compared to a policy that has been hard coded to be optimized for level 1 (i.e. Figure 2: Advantageously Chosen Weights in Right Lean), Mario performs best following a policy dictated by its Q-Learner
- SARSA does much better than random actions or Q-Learning with feature extractions in the non-bias runs and better in right-bias runs.
  - This may be because SARSA overfit its policy to the specific course we ran it in
  - This may have to do with how we didn't have it use feature extraction and because it sticks to optimizing one policy

future directions

- Currently our approaches aim to maximize the distance Mario travels. Future implementations could take into account
  - the importance of killing enemies (goombas/turtles),
  - collecting coins,
  - and gathering useful items (mushrooms/wildflowers)
- Our current feature extractors only take into consideration linear features instead of the relationship between features
  - A neural network may be used to better reason through the relationship between features.
- We were restricted to using only the first level of *Super Mario Bros.* because of our gym environment
  - Our Q-learner though is not optimized specifically for level 1 and would most likely generalize well

references

D, Soren. "Teaching a Neural Network to Play a Game Using Q-Learning." *Practical Artificial Intelligence*, 4 Sept. 2017, www.practicalai.io/teaching-a-neural-network-to-play-a-game-with-q-learning/.

ehrenbrav. "Teaching Your Computer To Play Super Mario Bros. – A Fork of the Google DeepMind Atari Machine Learning Project." *Obiter Dicta | The Blog of Ehren J. Brav*, 25 Aug. 2016, www.ehrenbrav.com/2016/08/teaching-your-computer-to-play-super-mario-bros-a-fork-of-the-google-deepmind-atari-machine-learning-project/.

Paquette, Philip. Gym-Super-Mario. (2017). GitHub Repository. https://github.com/ppaquette/gym-super-mario.