Super Mario Bros. Meets Reinforcement Learning

CS 182 Final Project Update

Gabe Grand and Kevin Loughlin

November 23, 2016

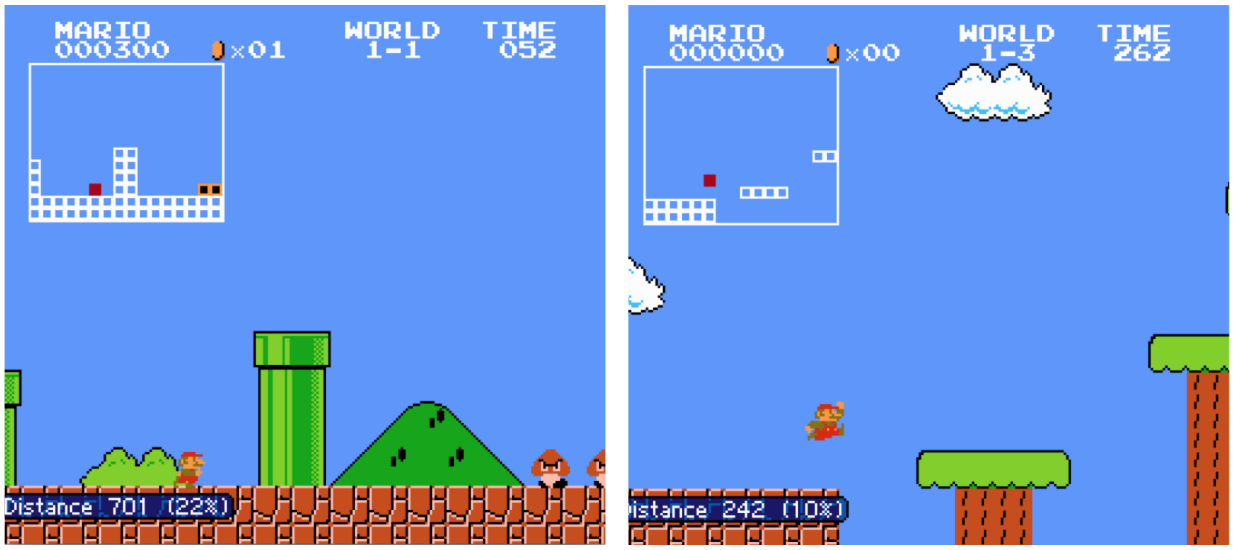
Status Update

We are on-track to give an oral presentation in class on 12/1. We are also on-track to turn in the final project on 12/12. Code is available at https://github.com/kevinloughlin/Super-Mario-RL.

Progress

We have set up the OpenAI Gym and Nintendo Entertainment System emulator environments on our machines. We have also figured out how to interface with the emulator in order to input selected actions and receive observations about the game state.

We have created a basic QLearningAgent class that uses the Bellman equations to perform Q-value updates and action selection. We have been testing our agent primarily on World 1-1 and World 1-3. We have observed that after a short amount of time, our agent learns to select actions that move Mario towards the right side of the screen. On World 1-1, Mario is able to make it about 22% of the way through the level, before getting stuck behind a tall pipe. On World 1-3, the agent has great difficulty handling two consecutive cliffs near the start of the level. As a result, Mario generally makes it through only 10% of the level.



Issues Encountered

State and Action Space

The size of the state space seems to be the main obstacle for Q Learning. The state space consists of a $16 \cdot 13 = 208$ array of square tiles representing different objects in the game. There are four object types in total:

- 0: empty space
- 1: object (e.g. platform, coins, flagpole, etc.)
- 2: enemy
- 3: Mario

Additionally, there are 6 buttons on a Nintendo controller (up, down, left, right, A, B), for a total of $2^6 = 64$ possible actions. However, we restricted the action space to include only logical combinations of 0-2 buttons (for instance, left + right is illogical). This reduces the action space to 14 actions.

In total, the size of the Q = (state, action) space is $4^{208} \cdot 14$, which is impractically large. In practice, however, many of the states are impossible (for instance, there can never be two Marios) or extremely unlikely. Additionally, the layout of the game world tends to follow certain patterns (platforms at the bottom of the map, empty space at the top), which significantly restricts the size of the state space. In 10 iterations on World 1-3, we found that Mario visited only $11,830$ unique states. We also observed that after only a few iterations, the vast majority of the actions selected had non-zero values, meaning that the agent's decision was based on (at least one instance of) previous experience.

Nevertheless, we are concerned that the large size of the state space is hindering the effectiveness of Q-Learning. One potential solution would be to switch to approximate Q-Learning using features that we compute ourselves. Of course, the challenge with this method would be hand-engineering good features. **Please let us know if you think this would be a good avenue to explore.**

Training Time

We have found training time to be another significant issue. The emulator runs in real time, so each training iteration (where 1 life = 1 iteration) can take up to 160 seconds, depending on how long Mario survives. There may be a way to increase the emulation speed, which we will look in to for the future. In the meantime, we have written code that saves the learned Q values in an efficient binary format. This method enables new instances of the learning agent to restore learned Q values, rather than start from scratch.

Emulator Freezing

On occasion, the emulator encounters an exception, which causes the game to freeze. This occurs in roughly 1 in 10 training iterations. Luckily, the OpenAI Gym environment can detect when the emulator is frozen, and can reset it automatically. When this occurs, we adjust the iteration

counter so that we always get at least N full training iterations. Emulator freezes do not reset the Q values, but they can result in "ghost" iterations, in which Q updates from a frozen iteration are incorporated into the model. This isn't necessarily an issue, but it does mean that additional learning may occur beyond the specified number of training iterations.

We are working closely on Github with the programmer who developed the Gym Super Mario Bros. environment to try to fix the issues with freezing. One solution we tried was to use a thread lock to prevent multiprocessing issues. This decreased the incidence of freezing, but under a heavy processing load, the game still occasionally freezes. Thus far, minimizing the number of active processes on the system remains the best way to prevent freezing.

Reward Function

The reward function given by the OpenAI Gym Super Mario environment is based on distance. Moving to the right results in a positive reward, while moving to the left results in a negative reward. This reward function is serviceable, but it does not incorporate other behaviors in the game that we may want to reward: for instance, when Mario stomps on a Goomba, or eats a mushroom. Information about these events is accessible through an object called `info`, which the environment compiles at every time-step. The OpenAI Gym Documentation says the following about the `info` object:

> **`info` (dict)**: diagnostic information useful for debugging. It can sometimes be useful for learning (for example, it might contain the raw probabilities behind the environment's last state change). However, official evaluations of your agent are not allowed to use this for learning.

It might be useful to incorporate some of the information from `info` into our reward function, but we do not want to "cheat" by using additional info. **Please advise us as to how to proceed.**

Future Plans

Moving forward, we will turn our attention to the POMDP component of the project. Additionally, if you think it is a good idea, we may turn to approximate inference in order to address the state space problem of Q-Learning.