CS182 Project Update

Nikhil Suri and Soumil Singh

November 2018

1 Progress So Far

So far, our project is going well. We have made considerable progress in training an exact Q-learning agent for playing Tetris, and are currently testing this agent and brainstorming different features that we could use for an approximate Q-learning agent.

We model the state for our exact Q-learning agent in the following way: it is a 3-tuple consisting of 1. the normalized top line of the game board (more on this later), 2. the shape of the piece currently falling, 3. the shape of the next piece that will fall. There are 7 different shapes in Tetris, so there are $7 * 7 = 49$ possible combinations of (fallingPiece, nextPiece) pairs. We have defined the "topline" of a game board to be all of the free spaces at the very top of each column. These are the spaces in which we would be able to place a falling piece. By "normalized" topline, we mean that, given a top line, we shift its rows down so that its lowest row is at row 0. For example, a top line of (row, column) pairs of $((4,0),(3,1),(1,2),(4,3),(4,4))$ becomes $((3,0),(2,1),(0,2),(3,3),(3,4))$. This helps reduce the state space, and is also valid in practice because piece actions are determined independently from what we determine our top line to be.

We model the actions we can take in a certain state as (rotation, column) pairs. The rotation represents the numerical rotation of a piece (for example, a number from 0 through 3). The column represents the column which we will drop the piece down (the point of reference is the top left-hand corner of the piece we are dropping). We determine what the valid actions for a piece are by checking which columns we can drop it down from the top of the board, for every rotation.

How does our agent do on exact learning compared to random?

We experimented with our agent by seeing the performance of our exact Q-learning agent in two different conditions. The first condition was using an agent which had undergone Q-learning for 800000 simulations of the game (where one game is defined by the agent starting the game, and playing until it 'dies'/fails), and the second condition was using a random agent i.e. an agent which hadn't undergone any exact Q-learning. Both qualitatively and quantitatively, it was clear that the agent which had undergone Q-learning performed significantly better than the random agent. Qualitatively, we noticed that the trained agent effectively chose moves that enabled it to clear lines and fill up empty spaces while minimizing height. Quantitatively, the results were clear: Testing over 10000 games revealed that on average the trained agent had a score of about 1000 points higher (1 'line-clear' higher) than that of the random agent. We conducted these experiments on a 5x8 board.

Of course, we would like our agent to achieve scores that are much better than this. Since the state space for exact Q-learning is so huge (and therefore training an agent for 800K iterations took a very long time), we hope that we will be able to effectively train an approximate Q-learning agent.

How are we going to implement Approx learning with features? We are yet to implement feature based Q-learning agent. In our discussion of possible features we might want to consider, we had various ideas for features listed below:

1. Average height of the 'stacks' in the game (we expect states with lower average stack height to be better, as this means the agent is farther away from dying).

2. Number of lines which would be cleared with the next action (we expect a state with more possible line-clears to be better).

3. Number/size of 'holes' on the board, where a hole is defined as an unfilled space on the board which has been blocked off by other pieces. (we expect more/bigger 'holes' on the board to be worse, as this limits the agents actions and means it might be closer to dying).

4. Height of the highest stack on the board (similarly to 1. we expect states with a lower highest stack height to be better).

We would appreciate any feedback from you regarding whether these seem like reasonable features to use.

We are also considering investigating these Q-learning agents on variations of the Tetris game, for example, when we incorporate the notion of a 'reserve piece'. However, we would appreciate any suggestions you might have for extensions of this project, in terms of the Q-learning aspect. We will investigate how the use of different features for approximate Q-learning affects the performance of our agent, and are looking for ways to make this exploration as interesting and complex as possible.