Reinforcement Learning in Large State Spaces and Tetris

Nikhil Suri [suri@college.harvard.edu]
Soumil Singh [soumil_singh@college.harvard.edu]
CS182: Artificial Intelligence

Fall 2018

For our project, we are studying the application of Reinforcement Learning in very large state spaces. The specific game that we are training an agent for is Tetris. Using Exact Q-Learning and Approximate Q-Learning with various heuristics and experimenting with different hyper-parameters, we hope to see which methods provide the best results and generalize well to larger state spaces. Based off of these results, we also hope to explore how well our agents perform in a more complex environment.

Introduction

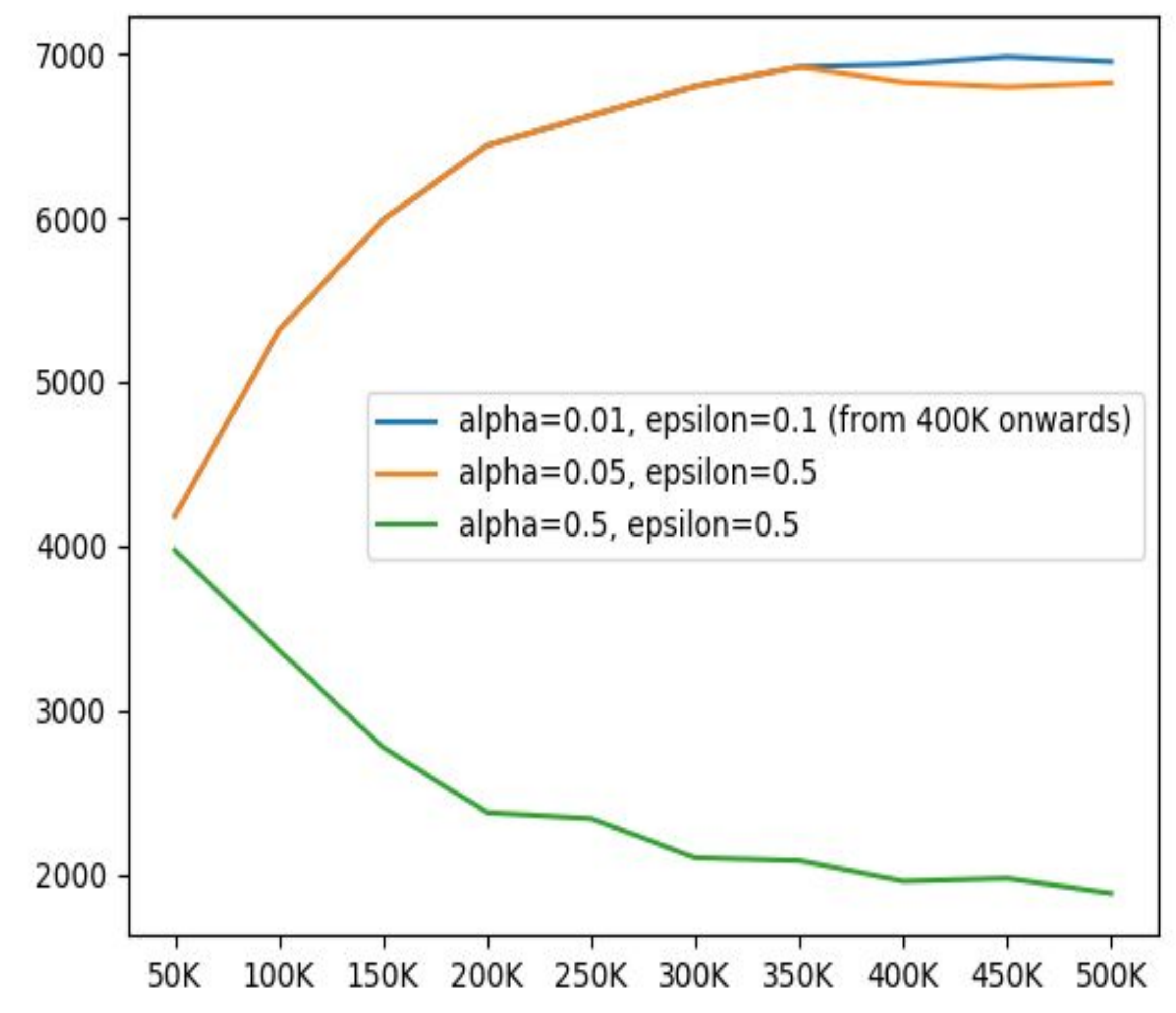
- Tetris is a game in which the player is presented with a grid, and one ‘piece’ in each time step. The player gets limited time to place the piece on the grid; if he/she completes a line (row), then the pieces occupying that line are removed from the board, and the stack reduces. If the piece ever breaches the top of the board the player loses.
- We investigate how effective applications of Q-learning are in producing agents to play this game
- We utilise both exact and approximate Q-learning methods (with various heuristics) and compare the effectiveness of both for differently sized tetris boards

Approach

- State Space: We define a state to be the configuration of the ‘top line’ of the shape stack (i.e. the arrangement of the highest blocks in each column) and the piece we are presented with. We originally thought about modelling the states as the entire board configuration + the current piece, but found we could reduce the state space using our current state definition whilst retaining most of the important information.
- Actions: Our actions are defined by a ‘rotation’ of the piece, and a column in which we ‘drop’ the piece.
- Approx Learning heuristics: We have implemented an Approximate Q-Learning Agent in addition to our exact Q-learning agent. Some features our Approximate Agent utilises are the average difference in height between columns (i.e. the ‘jaggedness of our board), the highest column on our board, and the number of blank spaces on our board within our shape stack. We aim to incorporate other features including, for example the sum of the squared sizes of the ‘holes’ inside the shape stack.

Results

- We tested our exact Q-learning agents with different learning hyperparameters. We observed the ‘average rewards’ the agent received when playing the game after a certain amount of iterations of training (for a particular hyperparameter), and plotted the ‘average rewards’ below. The average rewards refer to the ‘score’ the agent received when playing - a score of ‘7000’ would approximately correspond to the agent having cleared ‘7’ lines in the game.
- In the graph below, the x axis refers to the number (in thousands) of iterations of training. The y axis refers to the ‘average rewards’ the agent received with the given number of training iterations.
- We notice that when we set our alpha and epsilon learning parameters both to 0.5 our average rewards/scores reduced with the number of iterations (as illustrated by the green line). However, when we used lower values for our parameters (alpha at 0.05 and 0.01 and epsilon starting at 0.5 and decreasing to 0.1), we noticed that our rewards/scores increased with the number of iterations of training with some element of decay (as illustrated by the orange and blue lines).



Conclusions

- RL is very applicable to Tetris. Upon running enough iterations of the game, our exact Q-learning agent performs well. However, we suspect there is still much fine-tuning that we can do by experimenting with different values for our learning parameters (alpha and epsilon).
- The success of the approximate Q-learning agent is highly dependent on picking the most representative features. There is much more experimentation that we can do here. We hope to find the best combination of features to use, and the best way to scale these features so that we obtain a very effective Tetris-playing agent..
- Once we have some more preliminary results using approximate Q-learning, we would like to add more complexity to our model. For example, in actual tetris, you can drop piece and shift it over instead of only dropping down a column. You can also hold a reserve piece for later use. Such an alteration would make our state space more complex, and would require a re-examination of which heuristics are best for our approximate agent.

Citations and Links

- Tetromino (a Tetris clone). By Al Sweigart. al@inventwithpython.com. http://inventwithpython.com/pygame. Released under a "Simplified BSD" license.