Training a Tetris Agent Using Deep Q-Learning with Replay Memory

By Daniel Chang, David Justice, Puran Nepal, and Yousaf Khaliq

Introduction/Background

- Invented by Alexey Pajitnov on June 6, 1984
- Hailed as one of the greatest games of all time and the most downloaded game of all time
- Deep reinforcement learning has been applied to games such as chess, go, and Startcraft II but less so to Tetris
- Grid 10 blocks wide by 20 blocks high
- Each shape consists of four blocks: "I", "O", "T", "S", "Z", "J", and "L“
- The goal is to create “lines” using the shapes as they land at the bottom of the grid
- The game ends when the shapes reach the top of the grid
- Based on the rules and mechanics of the game, we decided to use Deep Q-Learning to find a solution

Introduction/Background (Cont.)

- Our aim was to design and train a successful neural network to play the game of Tetris
- We approached the problem in a similar fashion as the cart-pole problem using Deep Q-Learning
- The goal was to expand the cart-pole solution to the more complex problem of Tetris
- Currently, hand-coded solutions to Tetris yielded the most impressive scores and neural network agents proved less successful
- We obtained inspiration from Daniel Gordon's paper where he designed a DQL agent capable of scoring an average of 600 lines per game
- Hand-coded methods maximize survivability rather than points per game. Human competitions focus on points per game so it was our goal to focus on this, as well

Introduction/Background (Cont.)

- Four objectives in determining the computational complexity of Tetris:
  - Maximizing the number of rows cleared while playing the given piece sequence
  - Maximizing the number of pieces placed before a loss occurs
  - Maximizing the number of simultaneous clearing of four rows
  - Minimizing the height of the highest filled grid square over the course of the sequence
- We focused on the second objective
- It was proven that all four objectives are NP-complete
- Even if a computer solves Tetris, the game will ultimately end if the pieces fall truly randomly as there will eventually be a sequences of “S”’s and “Z”’s that will make the game impossible to continue

Methods/Results

- Used a Deep Q-Learning neural network with replay memory
- Replay memory was used to expedite the learning process and create "targets" for training and updating the weights
- Utilized the OpenAI Gym to simulate the Tetris environment
- Memory consisted of state, action, reward, and next state
- The model's input was the state
- Actions: left, right, down, rotate left, and rotate right
- Epsilon greedy policy: began at 1.0 and annealed down to .2
  - Random actions were taken more towards the beginning of training to encourage exploration so as not to repeat patterns the model became used to
- Replay memory varied. We tried 500,000 at first and ended up using only 15,000 recent memories. Our logic was to avoid patterns that may have had lower Q values

Methods/Results (Cont.)

- Most strategies that exist to solve Tetris have had varying degrees of success or lack thereof
- Shrinking down the size of the input vastly reduced training times: 430 x 330 down to 20 x 10
- Various training regimes and architectures used (as Daniel will discuss)

Contributions

- Yousaf – Administrator, organized and assigned project deliverables, presentation, reviewer
- Daniel – Code and reviewer
- David – Paper and reviewer
- Puran – Demo and reviewer

Demo Time!