Temporal Difference Learning in Tetris

Lou Brand and Alan Yeung
Department of Mathematics and Computer Science
Colorado College
Colorado Springs, CO 80903
`lodewijk.brand@coloradocollege.edu`

Abstract

Creating bots to play games has been an intense focus of AI researchers for many years. From checkers and Othello to more complicated games like chess and backgammon, the focus of AI researches has been to design algorithms that can play games as well as, or better, than humans. This project adds to that list of games that can be intelligently played by computers. The main focus of this project was to create an agent that could play Tetris intelligently. This paper focuses on a reinforcement learning algorithm that uses a neural network to approximate the score of a particular Tetris move. In addition to creating a bot that can play Tetris, another challenge addressed in this project is how to design a learning algorithm that trains an agent without the use of human-crafted heuristics.

1 Introduction

Tetris is an infinite game centered around not losing. The player lowers an infinite stream of different shaped blocks, made up of four individual cells, onto a board. The goal is to complete lines on the board. Once a line on the Tetris board is filled it gets cleared and the remaining blocks fall down. Our goal was to create a Tetris agent that could play Tetris effectively without any encoded heuristics. The only time the agent receives information is when it loses the game. The theory behind the agent used to play Tetris stems from two branches of AI combined together: reinforcement learning, and neural networks.

2 Related Work

The algorithm behind this project is based on an agent created by Gerald Tesauro [2]. Tesauro's agent was designed to play backgammon. His implementation used reinforcement learning, paired with a neural network's function approximation of the board, to choose the next best possible move. The final agent created by Tesauro "TD-Gammon 3.0" was able to play the game of backgammon just as well, if not better than, the best backgammon players in the world. What makes his temporal difference learning algorithm so unique is that it employed no human encoded heuristics. The agent learned entirely from the rules of the game. In the same vain, our Tetris agent will only be allowed to know when it has lost the game.

3 Methodology

The Tetris agent uses a basic reinforcement learning technique to determine what move to perform next. In order to determine what the best move is, a neural network, using a temporal difference algorithm, is trained and queried via a feature vector at every step. The temporal difference algorithm used for the neural network in this agent is similar to backpropagation, but over a long period of time.

When the agent loses, the temporal difference algorithm effectively backpropagates the negative consequence to the game states that came beforehand.

3.1 Feature Encoding

The Tetris board fed into the neural network for evaluation was represented by a feature vector that represented the entire board square for square. Each square was either on, or off, depending on whether or not it contained a Tetris piece. Seven more nodes were added to end of the Tetris board vector. One of these seven nodes was turned on; this node represented the next piece that was going to enter the game.

3.2 Reinforcement Learning

As with all reinforcement learning algorithms, a certain level of exploration is needed in order to get an idea of what move to preform. This exploration process of game states occurs with a certain probability: $P(\epsilon)$. As time moves forward $P(\epsilon)$ gets smaller and smaller and the algorithm begins to choose the best move more often. Our agent decides to make a given move based on the score a neural network gives it. If the neural network gives a higher relative score, compared to other moves, for a given legal move, the agent will choose that move.

3.3 Temporal Difference Algorithm

The temporal difference learning algorithm takes into account information gained over time. Once a game of Tetris is finished the algorithm is able to apply this reward (or lack therof) to future training. The algorithm is described mathematically (with help from Bonde and Sutton [1]) below:

$$w_{jk}^{t+1} = w_{jk}^{t} + \beta E_k e_{jk}^{t}$$

$$v_{ij}^{t+1} = v_{ij}^{t} + \alpha E_k e_{ijk}^{t}$$

$$E_k = r + \gamma y_k^{t+1} - y_k^{t}$$

w_{jk} is the weight from hidden node j to output node k
v_{ij} is the weight from input node i to hidden node j
E_k is the error between the future output (times a scaling factor γ) and the current output
e_{jk} and e_{ijk} are the eligibility traces at time t

The eligibility traces e_{jk} and e_{ijk} take into account how to apply rewards from one training cycle to another. They can be derived from backpropagation:

$$e_{jk}^{t+1} = \lambda e_{jk}^{t} + (y_k^t(1 - y_k^t)h_j^t)$$

$$e_{ijk}^{t+1} = \lambda e_{ijk}^{t} + y_k^t(1 - y_k^t)w_{jk}^t h_j^t(1 - h_j^t)x_i^t)$$

λ is a decay parameter
y_k^t is the value of the output node k at time t
h_j^t is the value of the hidden node k at time t
x_i^t is the value of the input node k at time t

4 Experiments

In order to test our temporal difference learning algorithm with Tetris we decided to see how well the algorithm performed on a four-by-four board. A small board allowed us to see improvements quickly, and efficiently, and helped us determine whether or not our algorithm was working. We tested to see whether or not the agent could place more pieces as time went on. Additionally, we tested to see whether the error of the temporal difference neural network went down over time.

5 Results

We tested the temporal difference learning algorithm by letting our bot play Tetris on a four-by-four game board. The average number of pieces placed (per game) was graphed with respect to the number of games played (Figure 1).

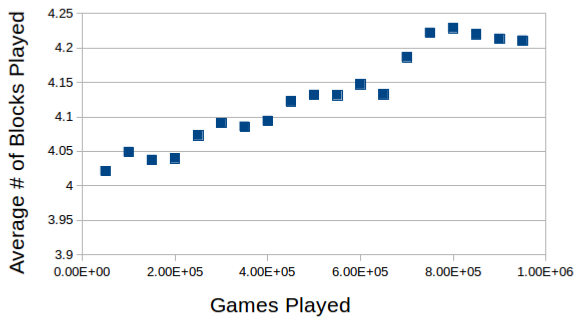

Figure 1: Average Blocks Played (per game)

Next we graphed the average error in our neural network with respect to the total number of pieces placed on the board (Figure 2). We decided to change the x-axis from "games played" to "the number of moves made" placed because the neural network updates its weights after every move.

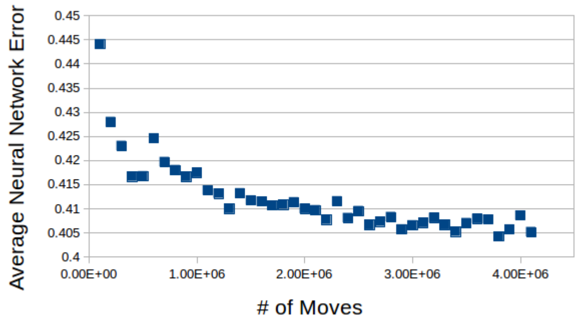

Figure 2: Average Neural Network Error (per move)

6 Conclusion

After looking at both of the graphs produced, we came to the conclusion that our temporal difference algorithm works relatively well for a four-by-four board. Although, we do see a leveling off of the neural network error and the number of blocks played. In further simulations it would be interesting to see what happens to the neural network error, and the number of blocks played, if the simulation ran longer. If the "flattening out" trend continued, it would likely represent a local minimum was found in the neural network's error landscape.

However, it is significant that the algorithm used to teach this Tetris agent did not use any human defined heuristics to learn. The agent was only notified when it lost the game. Given a significant amount of time playing Tetris, paired with a way to avoid local minima, this Tetris agent could theoretically learn to play Tetris optimally, without human oversight.

References

[1] Bonde, A. & Sutton R. (1993). Nonlinear TD/Backprop Pseudo C-code. *GTE Laboratories Incorperated*

[2] Tesauro, G. (1995). Temporal Difference Learning and TD-Gammon. *Communications of the ACM* **38**(3), 58-68.