Minimax Search and Reinforcement Learning for Adversarial Tetris

Maria Rovatsou and Michail G. Lagoudakis

Intelligent Systems Laboratory
Department of Electronic and Computer Engineering
Technical University of Crete
Chania 73100, Crete, Greece
`mariarovatsou@gmail.com, lagoudakis@intelligence.tuc.gr`

Abstract. Game playing has always been considered an intellectual activity requiring a good level of intelligence. This paper focuses on Adversarial Tetris, a variation of the well-known Tetris game, introduced at the 3rd International Reinforcement Learning Competition in 2009. In Adversarial Tetris the mission of the player to complete as many lines as possible is actively hindered by an unknown adversary who selects the falling tetraminoes in ways that make the game harder for the player. In addition, there are boards of different sizes and learning ability is tested over a variety of boards and adversaries. This paper describes the design and implementation of an agent capable of learning to improve his strategy against any adversary and any board size. The agent employs MiniMax search enhanced with Alpha-Beta pruning for looking ahead within the game tree and a variation of the Least-Squares Temporal Difference Learning (LSTD) algorithm for learning an appropriate state evaluation function over a small set of features. The learned strategies exhibit good performance over a wide range of boards and adversaries.

1 Introduction

Skillful game playing has always been considered a token of intelligence, consequently Artificial Intelligence and Machine Learning exploit games in order to exhibit intelligent performance. A game that has become a benchmark, exactly because it involves a great deal of complexity along with very simple playing rules, is the game of Tetris. It consists of a grid board in which four-block tiles, chosen randomly, fall from the top and the goal of the player is to place them so that they form complete lines, which are eliminated from the board, lowering all blocks above. The game is over when a tile reaches the top of the board. The fact that the rules are simple should not give the impression that the task is simple. There are about 40 possible actions available to the player for placing a tile and about 10^{64} possible states that these actions could lead to. These magnitudes are hard to deal with for any kind of player (human or computer). Adversarial Tetris is a variation of Tetris that introduces adversity in the game, making it even more demanding and intriguing; an unknown adversary tries to hinder the goals of the player by actively choosing pieces that augment the difficulty of line completion and by even "leaving out" a tile from the entire game, if that suits his adversarial goals. This paper presents our approach to designing a learning player for Adversarial Tetris. Our player employs MiniMax search to produce a strategy that accounts for any adversary and reinforcement learning to learn an appropriate state evaluation function. Our agent exhibits improving performance over an increasing number of learning games.

2 Tetris and Adversarial Tetris

Tetris is a video game created in 1984 by Alexey Pajitnov, a Russian computer engineer. The game is played on a 10×20 board using seven kinds of simple tiles, called *tetraminoes*. All tetraminoes are composed of four colored blocks (*minoes*) forming a total of seven different shapes. The rules of the game are very simple. The tiles are falling down one-by-one from the top of the board and the user rotates and moves them until they rest on top of existing tiles in the board. The goal is to place the tiles so that lines are completed without gaps; completed lines are eliminated, lowering all the remaining blocks above. The game ends when a resting tile reaches the top of the board. Tetris is a very demanding and intriguing game. It has been proved [1] that finding a strategy that maximizes the number of completed rows, or maximizes the number of the lines eliminated simultaneously, or minimizes the board height, or maximizes the number of tetraminoes placed in the board before the game ends is an $\mathcal{NP}$-hard problem; even approximating an optimal strategy is $\mathcal{NP}$-hard. This inherent difficulty is one of the reasons this game is widely used as a benchmark domain. Tetris is naturally formulated as a Markovian Decision Process (MDP) [2]. The state consists of the current board and the current falling tile and the actions are the approximately 40 placement actions for the falling tile. The transition model is fairly simple; there are seven equiprobable possible next states, since the next board is uniquely determined and the next falling piece is chosen uniformly. The reward function gives positive numerical values for completed lines and the goal is to find a policy that maximizes the long-term cumulative reward.

The recent Reinforcement Learning (RL) Competition [3] introduced a variation of Tetris, called *Adversarial Tetris*, whereby the falling tile generator is replaced by an active opponent. The tiles are now chosen purposefully to hinder the goals of the player (completion or lines). The main difference in the MDP model of Adversarial Tetris is the fact that the distribution of falling tiles is non-stationary and the dimension of the board varies in height and width. Furthermore, the state is produced like the frames of the video game, as it includes the current position and rotation of the falling tile in addition to the configuration of the board and the player can move/rotate the falling tile at each frame. The RL Competition offers a generalized MDP model for Adversarial Tetris which is fully specified by four parameters (the height and width of the board and the adversity and type of the opponent). For the needs of the competition 20 instances of this model were specified with widths ranging from 6 to 11, heights ranging from 16 to 25, and different types of opponents and opponent's adversity.

3 Designing a Learning Player for Adversarial Tetris

Player Actions. In Adversarial Tetris the tile is falling one step downwards every time the agent chooses one of the 6 low-level actions: move the tile left or right, rotate it clockwise or counterclockwise, drop it, and do nothing. Clearly, there exist various alternative sequences of these actions to achieve the same placement of the tile; this freedom yields repeated board configurations that lead to an unnecessary growth of the game tree. Also, playing at the level of the 6 low-level actions ruins the idea of a two-player alternating game, as the opponent's turn appears only once after several turns of the player. Lastly, the branching factor of 6 would lead to an intractable game tree, even before the falling tile reaches a resting position in the board. These observations led us to consider an abstraction of the player's moves, namely high-level actions that bring the tile from the top of the board directly to its resting position using a minimal sequence of low-level actions planned using a simple look-ahead search. The game tree now contains alternating plies of the player's and the opponent's moves, as a true two-player alternating game; all unnecessary intermediate nodes of player's low-level actions are eliminated. The actual number of high-level actions available in each state depends on the width of the board and the number of distinct rotations of the tile itself, but they will be at most $4 \times wb$, where wb is the width of the board (wb columns and 4 rotations). Similarly, the opponent chooses not only the next falling tile, but also its initial rotation, which means that he has as many as $4 \times 7 = 28$ actions. However, not all these actions are needed to represent the opponent's moves, since in the majority of cases the player can use low-level actions to rotate the tile at will. Thus, the initial rotation can be neglected to reduce the branching factor at opponent nodes from 28 to just 7. In summary, there are about $4wb$ choices for the player and 7 choices for the opponent.

Game Tree. The MiniMax objective criterion is commonly used in two-player zero-sum games, where any gain on one side (Max) is equal to the loss on the other side (Min). The Max player is trying to select its best action over all possible Min choices in the next and future turns. In Adversarial Tetris, our player is taken as Max, since he is trying to increase his score, whereas the adversarial opponent is taken as Min, since he is trying to decrease our player's score. We adopted this criterion because it is independent of the opponent (it produces the same strategy irrespectively of the competence of the opponent) and protects against tricky opponents who may initially bluff. Its drawback is that it does not take risks and therefore it cannot exploit weak opponents. The implication is that our agent should be able to play Tetris well against any friendly, adversarial, or no-care opponent. The MiniMax game tree represents all possible paths of action sequences of the two players playing in alternating turns. Our player forms a new game tree from the current state, whenever it is his turn to play, to derive his best action choice. Clearly, our player cannot generate the entire tree, therefore expansion continues up to a cut-off depth. The utility of the nodes at the cut-off depth is estimated by an evaluation function described below. MiniMax is aided by Alpha-Beta Pruning, which prunes away nodes and subtrees not contributing to the root value and to the final decision.

Evaluation Function. The evaluation of a game state s whether in favor or against our agent is done by an evaluation function $V(s)$, which also implicitly determines the agent's policy. Given the huge state space of the game, such an evaluation function cannot be computed or stored explicitly, so it must be approximated. We are using a linear approximation architecture formed by a vector of k features $\phi(s)$ and a vector of k weights w. The approximate value is computed as the weighted sum of the features, $V(s) = \sum_{i=1}^{k} \phi_i(s)w_i = \phi(s)^\top w$. We have issued two possible sets of features which will eventually lead to two different agents. The first set includes 6 features for characterizing the board: a constant term, the maximum height, the mean height, the sum of absolute column differences in height, the total number of empty cells below placed tiles (holes), and the total number of empty cells above placed tiles up to the maximum height (gaps). The second set uses a separate block of these 6 features for each one of the 7 tiles of Tetris, giving a total of 42 features. This is proposed because with the first set the agent can learn which boards and actions are good for him, but cannot associate them to the falling tiles that these actions manipulate. The same action on different tiles, even if the board is unchanged, may have a totally different effect; ignoring the type of tile leads to less effective behavior. This second set of features alleviates this problem by simply weighing the 6 base features differently for different falling tiles. Note that only one block of size 6 is active in any state, the one corresponding to the current falling tile.

Learning. In order to learn a good set of weights for our evaluation function we applied a variation of the Least-Squares Temporal Difference Learning (LSTD) algorithm [4]. The need for modifying the original LSTD algorithm stems from the fact that the underlying agent policy is determined through the values given to states by our evaluation function, which are propagated to the root; if these values change, so does the policy, therefore it is important to discard old data and use only the recent ones for learning. To this end, we used the technique of *exponential windowing*, whereby the weights are updated in regular intervals called *epochs*; each epoch may last for several decision steps. During an epoch the underlying value function and policy remain unchanged for collecting correct evaluation data and only at the completion of the epoch are the weights updated. In the next epoch, data from the previous epoch are discounted by a parameter μ. Therefore, past data are not completely eliminated, but are weighted less and less as they become older and older. Their influence depends on the value of μ which ranges between 0 (no influence) to 1 (full influence). A value of 0 leads to singularity problems due to the shortage of samples within a single epoch, however a value around 0.95 offers a good balance between recent and old data with exponentially decayed weights. A full description of the modified algorithm is given in Algorithm 1 (t indicates the epoch number). In order to accommodate a wider range of objectives we used a rewarding scheme that encourages line completion (positive reward), but discourages loss of a game (negative reward). We balanced these two objectives by giving a reward of $+1$ for each completed line and a penalty of -10 for each game lost. We set the discount factor to 1 ($\gamma = 1$) since rewards/penalties do not loose value as time advances.

Algorithm 1. LSTD with Exponential Windowing

```
$(w^t, A^t, b^t)$ = LSTD-EW$(k, \phi, \gamma, t, D^t, w^{t-1}, A^{t-1}, b^{t-1}, \mu)$
  if $t == 0$ then
    $A^t \leftarrow 0$;   $b^t \leftarrow 0$
  else
    $A^t \leftarrow \mu A^{t-1}$;   $b^t \leftarrow \mu b^{t-1}$
  end if
  for all samples $(s, r, s') \in D^t$ do
    $A^t \leftarrow A^t + \phi(s)\big(\phi(s) - \gamma\phi(s')\big)^\top$;   $b^t \leftarrow b^t + \phi(s)r$
  end for
  $w^t \leftarrow (A^t)^{-1} b^t$
  return $w^t, A^t, b^t$
```

Related Work. There is a lot of work on Tetris in recent years. Tsitsiklis and Van Roy applied approximate value iteration, whereas Bertsekas and Ioffe tried policy iteration, and Kakade used the natural policy gradient method. Later, Lagoudakis et al. applied a least-squares approach to learning an approximate value function, while Ramon and Driessens modeled Tetris as a relational reinforcement learning problem and applied a regression technique using Gaussian processes to predict the value function. Also, de Farias and Van Roy used the technique of randomized constraint sampling in order to approximate the optimal cost function. Finally, Szita and Lörincz applied the noisy cross-entropy method. In the 2008 RL Competition, the approach of Thiery [5] based on λ-Policy Iteration outperformed all previous work at the time. There is only unpublished work on Adversarial Tetris from the 2009 RL Competition, where only two teams participated. The winning team from Rutgers University applied look-ahead tree search and the opponent in each MDP was modeled as a fixed probability distribution over falling tiles, which was learned using the cross entropy method.

4 Results and Conclusion

Our learning experiments are conducted over a period of 400 epochs of 8,000 game steps each, giving a total of 3,200,000 samples. The weights are updated at the end of each learning epoch. Learning is conducted only on MDP #1 (out of the 20 MDPs of the RL Competition) which has board dimensions that are closer to the board dimensions of the original Tetris. Learning takes place only at the root of the tree in each move, as learning at the internal nodes leads to a great degree of repetition biasing the learned evaluation function. Agent 1 (6 features) learns by backing up values from depth 1 (or any other odd depth). This set of features ignores the choice of Min and thus it would be meaningless to expand the tree one more level deeper at Min nodes, which are found at odd depths. The second agent (42 features) learns by backing up values from depth 2 (or any other even depth). This set of basis functions takes the action choice of the Min explicitly into account and thus it makes sense to cut-off the search at Max nodes, which are found at even depths. The same cut-offs apply to testing.

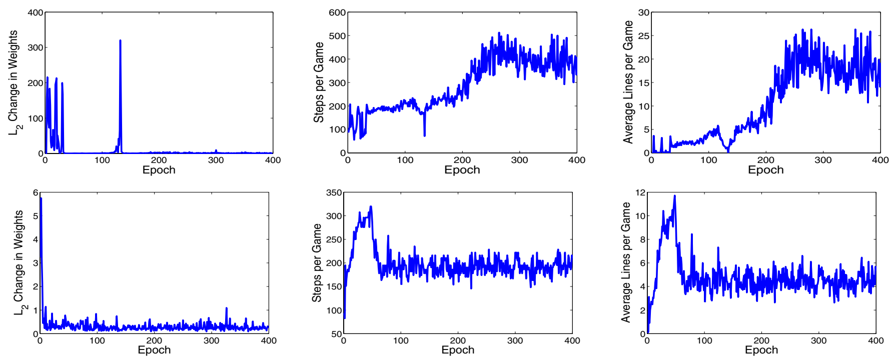

Fig. 1. Learning curve, steps and lines per update for Agents 1 (top) and 2 (bottom)

Learning results are shown in Figure 1. Agent 1 clearly improves with more training epochs. Surprisingly, Agent 2 hits a steady low level, despite an initial improvement phase. In any case, the performance of the learned strategies is way below expectations compared to the current state-of-the-art. A deeper look into the problem indicated that the opponent in Adversarial Tetris is not very aggressive after all and the MiniMax criterion is way too conservative, as it assumes an optimal opponent. In fact, it turns out that an optimal opponent could actually make the game extremely hard for the player; this is reflected in the game tree and therefore our player's choices are rather mild in an attempt to avoid states where the opponent could give him a hard time. Agent 1 avoids this pitfall because it goes only to depth 1, where he cannot "see" the opponent, unlike Agent 2. Nevertheless, the learned strategies are able to generalize consistently to the other MDPs (recall that training takes place only on MDP #1). For each learned strategy, we played 500 games on each MDP to obtain statistics. Agent 1 achieves 574 steps and 44 lines per game on average over all MDPs (366 steps and 16 lines on MDP #1), whereas Agent 2 achieves 222 steps and 11 lines (197 steps and 5 lines on MDP #1). Note that our approach is off-line; training takes place without an actual opponent. It remains to be seen how it will perform in an on-line setting facing the exploration/exploitation dilemma.

References

1. Breukelaar, R., Demaine, E.D., Hohenberger, S., Hoogeboom, H.J., Kosters, W.A., Liben-Nowell, D.: Tetris is hard, even to approximate. International Journal of Computational Geometry and Applications 14(1-2), 41–68 (2004)
2. Tsitsiklis, J.N., Roy, B.V.: Feature-based methods for large scale dynamic programming. Machine Learning, 59–94 (1994)
3. Reinforcement Learning Competition (2009), `http://2009.rl-competition.org`
4. Bradtke, S.J., Barto, A.G.: Linear least-squares algorithms for temporal difference learning. Machine Learning, 22–33 (1996)
5. Thiéry, C: Contrôle optimal stochastique et le jeu de Tetris. Master's thesis, Université Henri Poincaré – Nancy I, France (2007)