The Analysis of different Reinforcement Learning Techniques on the Tetris Game

Amr Saad

Abstract—**This paper will discuss the different techniques of reinforcement learning that are used on Tetris. Reinforcement learning and the Tetris game will be introduced to give a better understanding of what I am looking into. An analysis of the existing research will be investigated and discussed. All of the discussed techniques in this paper have been shown to work and have their benefits. My goal is to find the most optimal strategy and propose one of my own.**

I. INTRODUCTION

REINFORCEMENT learning is a branch of artificial intelligence that focuses on achieving the learning process in the course of a digital agent's lifespan *(Carr 2005).* It allows the machine or software agent to learn its behavior based on the feedback that is received from the environment. The machine may adapt over time or may learn once and continue that behavior. A basic reinforcement learning model consists of the following: a set of environment states, a set of actions, rules to allow transitions between different states, rules that determine the amount of reward for any given transition, and finally rules that describe the agent's observations. The agent works in time steps, at each time step (decided at the developers preference) the agent calculates an observation, the observation typically gives it a reward. The agent now chooses an action from a set of those allowed, this action is then sent to the environment where the states are updated and the rewards are determined based on the transition. The idea behind the machine or software agent is that it tries to maximize the reward it receives. This can give you an idea of how many different ways there are to implement an agent. The more optimal agent will be able to effectively and efficiently calculate the long-term consequences of each action that is available to it.

The main issue faced with reinforced learning is memory management; the process of keeping the states and values requires a lot of memory. If the problem were too complex it would require looking into different value approximation techniques, which may end up giving a weak or wrong solution. The reinforcement learning problem is studied in many areas, such as control theory, information theory, robotics and control, and game theory. For the purpose of this paper the one I will be focusing on is game playing. Game playing is a big part of artificial intelligence and has always been. A lot of artificial intelligence focus has been set on two player games such as chess, checkers, mastermind, tic-tac-toe and so on. The study of these types of games is very interesting but irrelevant when it comes to reinforcement learning. Agents are much better at maximizing their reward in a fixed environment. In two player games they would be required to maximize their rewards against another player's optimal play. There are games where reinforcement learning is useful and can be used to create a well-trained agent. One such example is Tetris, in which this type of learning has been shown to work successfully. Tetris is a well-established game that was created in 1985 by Alexey Pajitnov and has been thoroughly investigated by the mathematics and artificial intelligence communities *(Carr 2005).* Al-though conceptually simple, it is NP-complete *(Breukelaar et al., 2004)* and any formalized optimal strategy would be incredibly contentious. The original version of Tetris that will be analyzed uses a 2-dimensional board that is a fixed size. In Figure 1.0 is an image of what the typical Tetris board looks like. The board is a grid of cells, having 10 columns, and 20 rows, for a total of 10 * 20 = 200 cells. Each cell can either be unoccupied (empty) or occupied (full) *(Fahey 2003).*

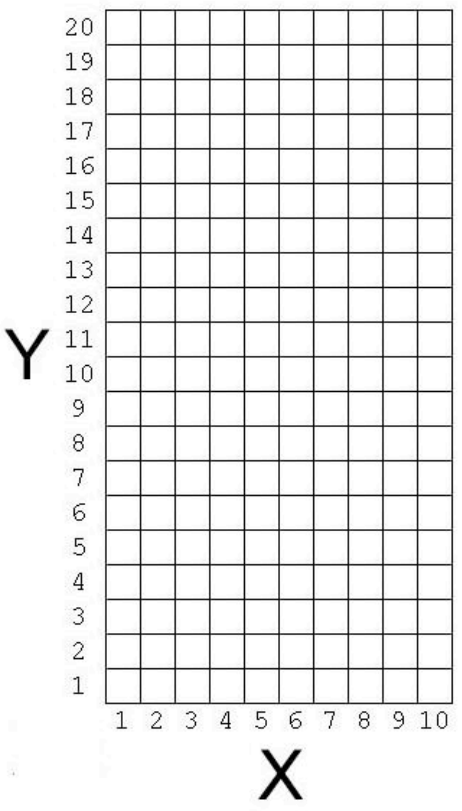


Figure 1.0: Standard Tetris board (10 columns, 20 rows) as defined by Fahey [2003]

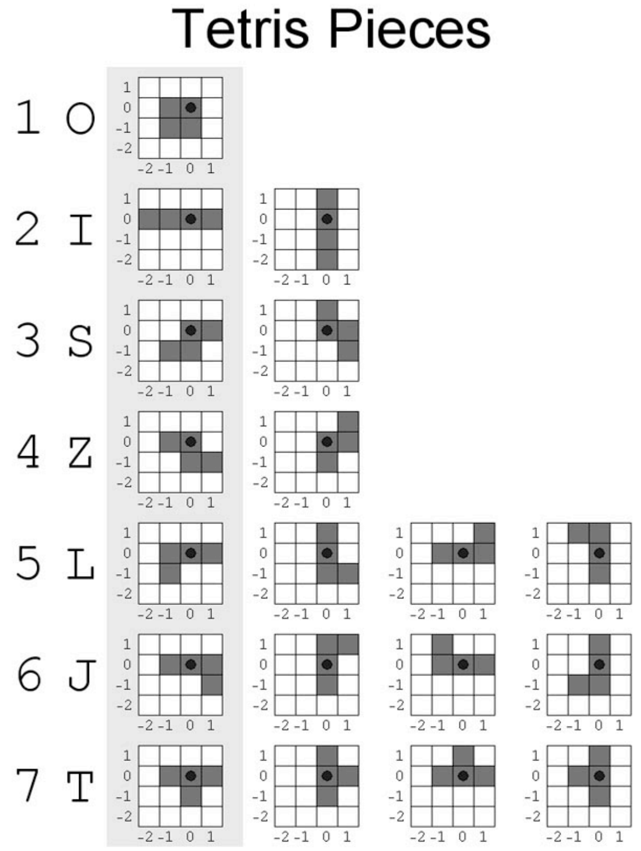


Figure 1.1: The seven Standard Tetris pieces and their "orientations" as defined by Fahey [2003].

The game consists of seven blocks that are to be placed on the board. The game begins by providing the player with a block that needs to be placed within a limited amount of time (given in time steps). Once the block is placed another block is given to player to either be stacked, or placed next to the previous block and so on. When a line is created across the board that line disappears and the user receives a certain number of points. The game continues until the blocks have stacked to the point where no other block can be supplied to the player. In other words if for any column the height is greater than the height of the board the game ends. When the game supplies the block it randomly chooses one of seven blocks that will be given to the player. In Figure 1.1 is an image of the seven types of blocks shown with their mathematical traits. The letter names are inspired by the shapes of the pieces {O,I,S,Z,L,J,T} *(Fahey 2003)*. A single Tetris block is also known as a tetromino and will be referred to as that for the remainder of this paper.

Once a tetromino has been randomly selected by the game it appears in its initial starting position in the (6,20) location on the board. The tetromino then begins to fall one space per time step until it meets an obstruction. An obstruction is either the limits of the board or another block that is in the way. As mentioned before once the block is set into a place and it completes a line, that line disappears and all rows above it move down. The specifications of formal Tetris are as follows (Fahey 2003).

- Tetris has a board with dimensions 10 by 20
- Tetris has seven distinct pieces
- The current game piece is drawn from a uniform distribution of these seven pieces
- Points are awarded for each block that is landed
- The player scores the most points for each piece by executing a drop before one or more free-fall iterations transpire.
- The game has ten different difficulty settings, which determine the period of free-fall iterations, and are applied as row completion

passes certain thresholds.
For the purposes of this paper I will consider the most optimal solution to be the one that scores the highest and learns the fastest.

II. Related Work

In order to fairly compare the related work we must establish what type of Tetris game was being studied. The types of Tetris include adversarial, one-piece, and two-piece. Adversarial Tetris is the Tetris game played between two people. The two players compete against each other to get higher scores by picking pieces for their opponents. One-piece Tetris is the standard game and does not allow the player to see the next piece that will be given to them. This version of Tetris is much simpler than the two-piece version because we only need to keep track of the current state and don't need to make calculations for both the current and next state to work together. Lastly there's the two-piece version, this version shows the current piece as well as the next piece to be given. The version of Tetris I would like to examine and compare the work of is the one-piece. This is where the majority of the study has been mainly because of the fact that it is simplest form of full Tetris that can be analyzed.

A. Reduced Tetris – Melax 1998, Bdolah and Livnat 2000

Stan Melax is a researcher who did plenty of work in AI learning particularly for Tetris. He came up with the idea of having a reduced Tetris game by minimizing the number of pieces, blocks in each piece, and the number of columns on the Tetris game board. Some of the differences in Melax's version of Tetris include the following:

- A board that is only six blocks wide
- The game was limited to dropping 10,000 tetrominoes
- A set working height of two.
- All information below the working height was thrown away.
- A set of a five tetrominoes available

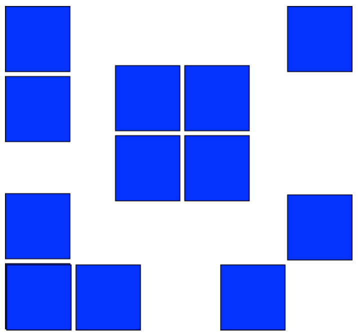


Figure 1.2: Set of tetriminoes used for the reduced version of Tetris (Melax 1998).

The reduced version of Tetris was different than a standard full game of Tetris. The game does not go until a column surpasses the height of the board because the height is theoretically infinite. Instead the game goes until ten thousand tetriminoes are given to the agent. Due to the fact that the active area of the board is only two blocks high Melax discarded anything below the active area.

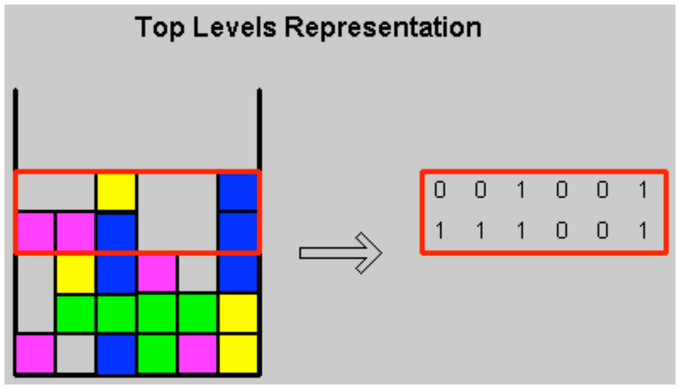


Figure 1.3: The top-level representation showing the active rows in the given state (Yael Bdolah and Dror Livnat 2000).

As you can see in Figure 1.3 the fact that the active height is only two meant the agent could not lower the block structure to complete any rows that were previously unfilled. Since the pieces are drawn stochastically and unfilled rows form an immutable blemish on the performance evaluation of the agent, this introduces a random aspect to the results (Carr 2005). The agent used a very simple reinforcement learning approach that uses a state-value table to associate a value with every state as opposed to a value for every action. This means the rewards were given upon every level that was introduced above the working height of the well instead of a reward being given for every block being placed. In Table 2.0 you can see the results of running the agent on the reduced version of Tetris for 256 games. The height of the board decreases dramatically showing that his techniques were successful.

Game	Height
1	1485
2	1166
4	1032
8	902
16	837
32	644
64	395
128	303
256	289

Table 2.0: Melax's Results (Melax 1998).

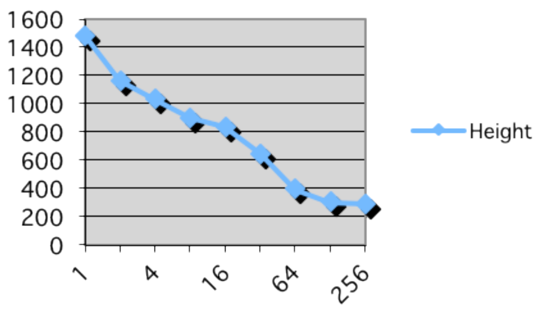


Figure 1.4: Graph of Melax's Results (Melax 1998)

With Melax's approach there are a lot of good qualities but also many things that could be done better. As we can see the agent will learn how to stack the blocks in a way that will keep the total height to a minimum. Even though the agent could make better plays than it does the results prove that reinforced learning is a good approach to solving the artificial Tetris player problem. Melax's approach inspired others to try to also apply reinforced learning to Tetris. Some people followed in his footsteps with the reduced Tetris and others worked on the full Tetris problem.

B. Reinforcement Learning and Neural Networks for Tetris (Nicholas Lundgaard and Brian McKee)

In this paper, Lundgaard and McKee conduct a project in order to record and compare various machine learning techniques for learning the game of Tetris. The focus of success for this project was to create an agent that effectively plays the game of Tetris competitively, while simulating a human player's experience. In order to do this, the following variants were placed on the gameplay: Line limits, score versus lines and competitive play. The results of the findings are then presented to address the initial hypothesis that agents trained solely based on the scoring function of lines cleared as their reward would outscore and outperform existing agents.

I would like to focus specifically on neural networks. "Neural networks are machine learning tools built to simulate biological neurons. Given input, output is generated as the input flows through a set of interconnected neurons, each connection being multiplied by a certain weight to generate the values that reach the outermost output neurons." That output of the net is therefore considered the prediction. When combined with reinforcement learning, neural networks can be used in unsupervised learning where action is based on the expected reward, which is determined by the output of the net. The neural network agents are implemented on two levels of state space, a low-level neural network and a high-level neural network.

The low-level neural network had problems because it is incapable of recognizing similar states as being similar (Nicholas Lundgaard and Brian McKee). Nonetheless the results for this technique were very similar to a random agent. The agent would choose random actions and end up with higher rewards. Performance did not improve with an increase of games it stayed at a steady consistent score as can be seen in Figure 1.8.

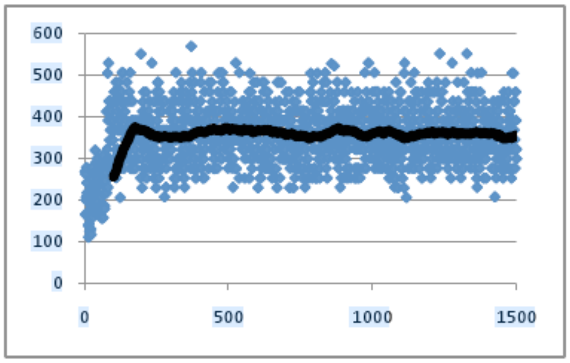


Figure 1.8: Low-level neural net performance with moving average (Nicholas Lundgaard and Brian McKee).

In the high-level neural network, the agent operates on the high-level state representation, and chooses between such high-level actions as opposed to the simple action of rotating and translating pieces. The learning results of this agent show an obvious link to the reward strategy. The high-level neural agent was trained with two types of game settings, with no limit and with line limit. Here the agent would receive rewards for choosing the maximum lines action, and then continued this action in similar situations. When there were lines to clear the misused action would lead to poor performance. The bottom line here is that when the scoring for clearing multiple lines simultaneously out-weighted the action of clearing out the board, the agent would make riskier moves, ultimately leading to a shorter average game. The below graph shows results of the agent's performance averaged over five runs of 500 games with a moving average of period 50.

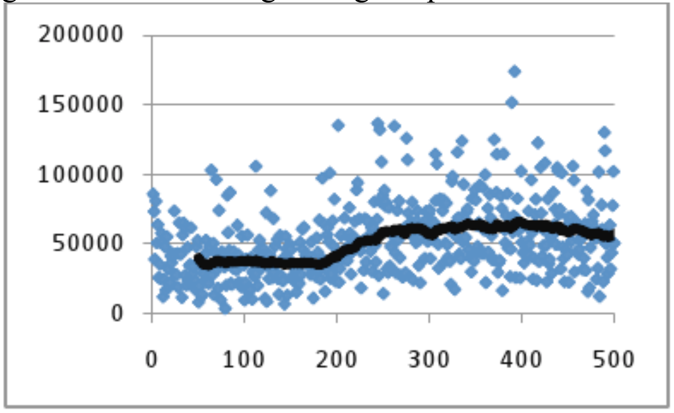


Figure 1.9: High-level neural net performance with moving average (Nicholas Lundgaard and Brian McKee).

Furthermore, below is the performance of the high-level neural net in a 500-game single run. As you can see, the agent performs poorly by implementing an action that provided a high reward early on, but was an ineffective strategy overall. With random exploration, however, the agent picks up more effective strategies.

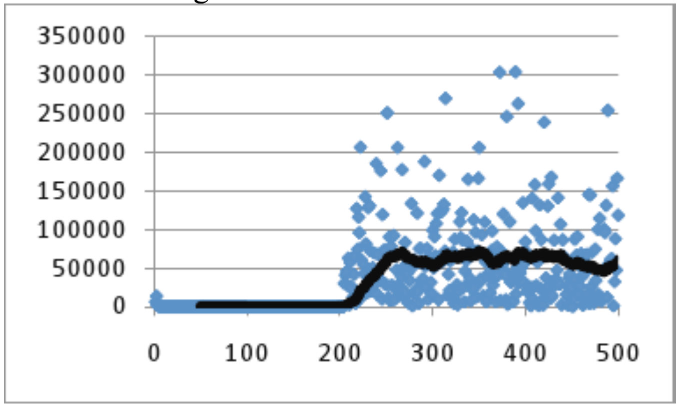


Figure 2.0: Neural net performance on single run displaying strategy shifts. (Nicholas Lundgaard and Brian McKee).

With a line limit of 300 max, and compared to another, manually-weighted agent, the neural net's clearing of 300 lines was inconsistent in comparison to the other. However, the neural net would earn more points than the opposing agent when it succeeded in surviving the entire game. Below is the performance of the two agents averaged over five runs of 500 games, with moving averages of period 50.

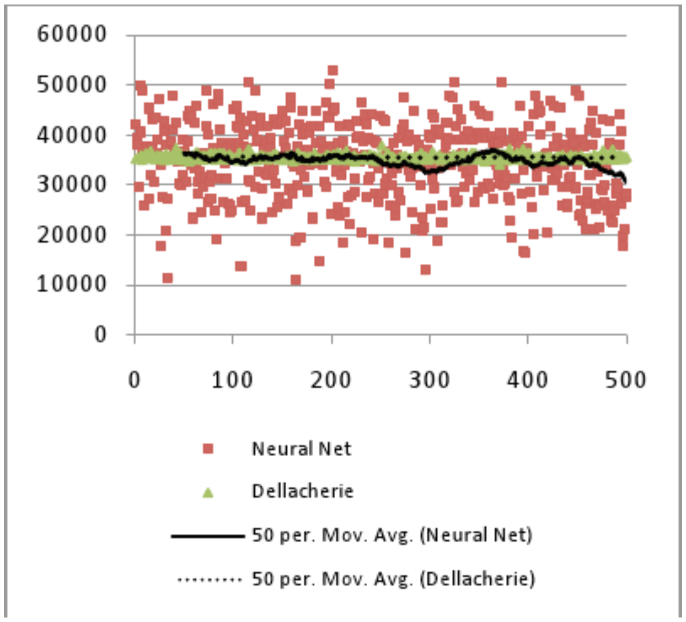
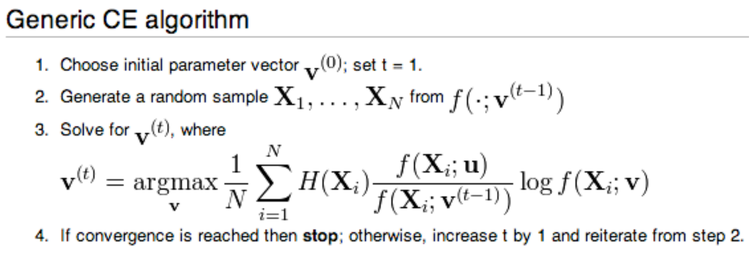


Figure 2.1: Neural net performance vs. Dellacherie over 300 lines. (Nicholas Lundgaard and Brian McKee).

C. Learning Tetris Using the Noisy Cross-Entropy Method (Istvan Szita and Andras Lorincz 2006)

This method of learning Tetris is very different than the rest because it uses a combination of reinforcement learning and the cross-entropy (CE) method. The CE method is a general method for solving optimization problems. It uses a Monte Carlo approach to accurately estimate very small probabilities. The CE method consists of two phases *(Wikipedia 2012)*:

1. Generate a random data sample (trajectories, vectors etc.) according to a specified mechanism.
2. Update the parameters of the random mechanism based on the data to produce a "better" sample in the next iteration.



Generic CE algorithm

1. Choose initial parameter vector $\mathbf{v}^{(0)}$; set t = 1.
2. Generate a random sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ from $f(\cdot; \mathbf{v}^{(t-1)})$
3. Solve for $\mathbf{v}^{(t)}$, where

$$\mathbf{v}^{(t)} = \operatorname*{argmax}_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^{N} H(\mathbf{X}_i) \frac{f(\mathbf{X}_i; \mathbf{u})}{f(\mathbf{X}_i; \mathbf{v}^{(t-1)})} \log f(\mathbf{X}_i; \mathbf{v})$$

4. If convergence is reached then **stop**; otherwise, increase t by 1 and reiterate from step 2.

Figure 1.5: The Generic Cross-Entropy Method (Wikipedia 2012)

Istvan Szita and Andras Lorincz used the cross-entropy method in addition to reinforcement learning to come up with a very optimal technique to learning Tetris. This technique outperforms other reinforcement learning techniques by almost two orders of magnitude *(Istvan Szita and Andras Lorincz 2006)*. The value function for the cross-entropy method is calculated by considering the tetromino in every available position, and after deleting any full rows if they exist. Once the values are calculated for all of the available positions of the tetromino piece the agent chooses the highest valued column and piece orientation. The cross-entropy is then applied to learn the weights of various basis functions, which are picked through an independent Gaussian distribution.

Szita and Lorincz run into the problem of converging to a single point too fast when using the CE method. The fix for this early convergence is adding noise to the distribution at every iteration. The next thing of note is that the scoring rules that are used for this experiment are different than Melax's method. The agent registers 1 point for each row that is cleared and has no bonus points for pieces and or extra rows.

The samples were drawn from playing a single game that uses a corresponding value function. The first test was run without any kind of noise added to the distribution. As expected the average performance for the Tetris trials was roughly 20,000 points. In the next experiment there was a constant noise applied to the distribution. This time the problem was a lack of convergence because the noise was too high. The most optimal scores when keeping a constant noise rate that did not diverge ended up being an average of 70,000 points. This led to the idea used for the third experiment, which was a decreasing amount of noise starting at the max noise that did not cause any divergence. With this approach there was a dramatic increase in results. The average scores exceeded 300,000 points and the maximum score was more than 800,000 points. In Figure 1.6 shows the results of all three experiments.

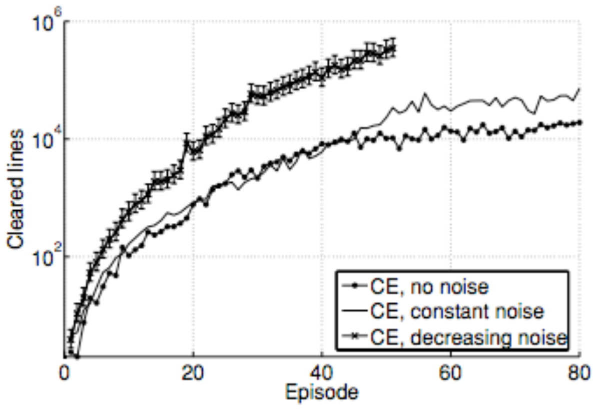
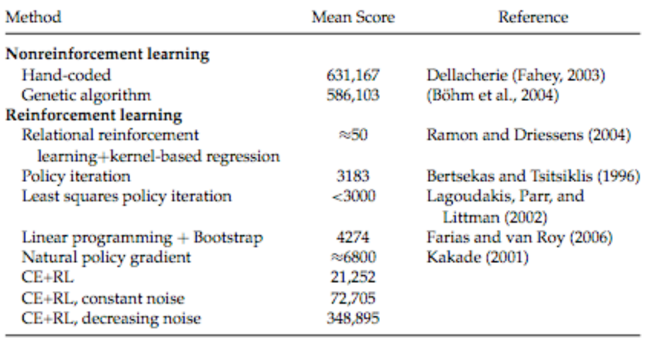


Figure 1.6: Tetris scores on logarithmic scale. Mean performance of 30 evaluations versus iteration number. Error bars denoting 95% confidence intervals are shown on one of the curves (*Istvan Szita and Andras Lorincz 2006).*

In comparison to the other work presented in this paper this method has the most successful reinforcement learning results. Szita and Lorincz present a chart comparing the majority of results that have been run on the Tetris problem. Table 1.1 presents these results as presented by I.Szita and A.Lorincz.



Method	Mean Score	Reference
Nonreinforcement learning		
Hand-coded	631,167	Dellacherie (Fahey, 2003)
Genetic algorithm	586,103	(Böhm et al., 2004)
Reinforcement learning		
Relational reinforcement learning+kernel-based regression	≈50	Ramon and Driessens (2004)
Policy iteration	3183	Bertsekas and Tsitsiklis (1996)
Least squares policy iteration	<3000	Lagoudakis, Parr, and Littman (2002)
Linear programming + Bootstrap	4274	Farias and van Roy (2006)
Natural policy gradient	≈6800	Kakade (2001)
CE+RL	21,252	
CE+RL, constant noise	72,705	
CE+RL, decreasing noise	348,895	

Table 1.1: Average Tetris Scores of Various Algorithms (*Istvan Szita and Andras Lorincz 2006).*

III. Future Work And Experiments

A. *Accuracy of Comparison*

For the future work and analysis I would like to make a better comparison of the different methods. Adjusting the tested methods to all follow the same rules and scoring systems would result in much more accurate conclusions. As of right now all of the comparison is measuring performance on different implementations.

B. *Close to Game Over*

There is a small problem when it comes to how the agents view the game when it is approaching the end. The agent has one of two options to take when it is choosing actions for the last few moves. The agent can either pick the highest evaluation even if that means a path to game over or choose the option to make it survive longer. In most cases although not obvious to the agent, if the game goes on longer the final score will be higher than if picking the more obvious higher evaluation. For example, in a test run done on Dellacherie's algorithm the difference of scores was remarkable. When eliminating paths that lead to game over the algorithm completed 5,200,000 rows on average as opposed to 850,000 rows on average. This is just one of the difficulties that are faced when comparing algorithms based on different implementations.

C. *Tetris Settings and Rules*

Just like a simple game ending adjustment can change comparison results so can Tetris settings and rules. One big difference is the comparison between the reduced Tetris game and the full Tetris game as well as the game board sizes. Most implementations end the game when a piece goes over the maximum height of the board. So in a 10 x 20 board if the piece overflows into the 21st row the game is finished. If this is compared to the techniques used by Bertsekas and Tsitsiklis (1996), they call the game over when any block is in the 20th row. Christophe Thiery and Bruno Scherrer note that in their findings changing the rule on Dellacherie's experiment changed the results from an average of 5,200,000 rows to 2,500,000 rows. With this in mind we can see that some techniques can be greatly underrated or overrated with respect to the others based on how the results are calculated. On the next page in Figure 1.7 you can see the different features that can impact the results (*Christophe Thiery and Bruno Scherrer).*

D. *Future Work*

For future work it would be nice to expand on a few things discussed in this paper. Mainly the areas of expanding would include standardizing the experiments in order to retrieve viable results, using pre-taught agents that continue to learn, and finding optimizations to allow for further look ahead to learn in more advanced Tetris games.

Standardizing the experiments is a necessary step if I plan to get results that can be compared. In an ideal situation the experiments will be run on the same Tetris game using the different analyzed algorithms. The number of trials per algorithm would need to be significantly high to accommodate for the randomness that is included in the game. To be even more precise it would be nice to include confidence intervals for the results to know the distribution. The algorithm with the most average points after a sufficient number of runs under the same rules could easily be deemed the best algorithm for the task at hand.

I would also like to try many of these algorithms with some human help, meaning pre-determined weights to help the learning process. If we were able to limit the choices of the agent to moves that we know will score higher the overall outcome would be positive. As you saw earlier in Table 1.1 the hand-coded method gave the best results. If we could work on finding away to combine the hand-coded perks with the learning perks it may eventually help.

I think the most interesting future work could be from looking into the different Tetris problems. I had mentioned a few of them earlier in the paper but Tetris problems such as the following:

- Two-Piece Tetris (One piece look ahead)
- Adversarial Tetris (Competitive 2-player)
- Multiple-line bonus points
- Survival Tetris (Last as long as possible)
- Sprint Tetris (Clear 20 lines as fast as possible)

It would be interesting to study all of these different versions of Tetris to see if reinforcement learning is suited well for them. Part of studying these other versions of the game would also require very optimal algorithms to be developed. For example one-piece Tetris has a maintainable state space when trying to calculate all of the possible moves and possible rotations for each piece. When you start looking into multiple line bonuses or more than one piece look ahead the state space becomes much larger because now planning is involved. All of these areas discussed are definitely worth taking a deeper look at.

IV. Conclusion

In conclusion out of the methods previously discussed I believe that the cross-entropy method has been shown to be the most successful and easiest to expand on. Although the number of lines completed in the reinforcement learning trials is nearly half of those that are hand-coded, there is still plenty of potential for adjustment.

Feature	Description	Tsitsiklis *et al.*, 1996	Bertsekas *et al.*, 1996	Llima, 2005	Lagoudakis *et al.*, 2002	Fahey, 2003	Dellacherie (Fahey, 2003)	Ramon *et al.*, 2004	Böhm *et al.*, 2005	Thiery *et al.*, 2009
Max height	Maximum height of a column	×	×	×	×			×	×	
Holes	Number of empty cells covered by a full cell	×	×	×	×	×	×	×	×	×
Column height	Height of each column		×					×		
Column difference	Height difference between each pair of adjacent columns		×					×		
Landing height	Height where the last piece is added			×			×		×	×
Cell transitions	Number of full to empty or empty to full cell transitions			×						
Deep wells	Sum of well depths, except for wells with depth 1			×						
Embedded holes	Sort of weighted sum of holes (not precisely documented)			×						
Height differences	Sum of the height differences between adjacent columns				×					
Mean height	Mean height of columns				×			×		
Δ max height	Variation of the maximum column height				×					
Δ holes	Variation of the hole number				×					
Δ height differences	Variation of the sum of height differences				×					
Δ mean height	Variation of the mean column height				×					
Removed lines	Number of lines completed at the last move				×	×			×	
Height weighted cells	Full cells weighted by their height					×			×	
Wells	Sum of the depth of the wells					×		×	×	
Full cells	Number of occupied cells on the board					×			×	
Eroded piece cells	(Number of rows eliminated in the last move) × (Number of bricks eliminated from the last piece added)						×			×
Row transitions	Number of horizontal cell transitions						×		×	×
Column transitions	Number of vertical cell transitions						×		×	×
Cumulative wells	$\sum_{w \in \text{wells}} (1 + 2 + \cdots + \text{depth}(w))$						×			×
Min height	Minimum height of a column							×		
Max − mean height	Maximum column height − Mean column height							×		
Mean − min height	Mean column height − Minimum column height							×		
Mean hole depth	Mean depth of holes							×		
Max height difference	Maximum difference of height between two columns								×	
Adjacent column holes	Number of holes, where adjacent holes in the same column count only once								×	
Max well depth	Maximum depth of a well								×	
Hole depth	Number of full cells in the column above each hole									×
Rows with holes	Number of rows having at least one hole									×
Pattern diversity	Number of different transition patterns between adjacent columns									×

Table 1.1: Features mentioned in Tetris publications (*Christophe Thiery and Bruno Scherrer)*

V. References

Periodicals:

[1] Boer, P. de, Kroese, D., Mannor, S., and Rubinstein, R. (2004). A tutorial on the cross-entropy method. Annals of Operations Research, Vol. 1, No. 134, pp. 19–67.

[2] Szita, I. and Lo˝rincz, A. (2006). Learning Tetris Using the Noisy Cross-Entropy Method. Neural Computation, Vol. 18, No. 12, pp. 2936–2941.

[3] Thiery, C. and Scherrer, B. (2009). Construction d'un joueur artificiel pour Tetris. Revue d'Intelligence Artifi- cielle, Vol. 23, pp. 387–407.

[4] Thiery, C., Scherrer, B.: Building Controllers for Tetris. International Computer Games Association Journal 32, 3-11 (2009)

[5] Driessens, Kurt, and Sašo Džeroski. "Integrating Guidance into Relational Reinforcement Learning." *Machine Learning* 57.3 (2004): 271-304.

Books:

[6] Bertsekas, D. and Tsitsiklis, J. (1996). Neurodynamic Programming. Athena Scientific.

[7] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. The MIT Press, Cambridge, MA, 2002. URL http://www.cs.ualberta.ca/sutton/book/ebook/index.html.

Technical Reports:

[8] Carr, D. (2005). Applying reinforcement learning to Tetris. Technical report, Computer Science Department of Rhodes University.

[9] Kurt Driessens. Relational Reinforcement Learning. PhD thesis, Catholic University of Leuven, 2004. URL: http://www.cs.kuleuven.ac.be/~kurtd/PhD/.

[10] Lundgaard, Nicholas, and Brian McKee. *Reinforcement Learning and Neural Networks for Tetris*. Tech. N.p.: University of Oklahoma, n.d. Print.

Papers Presented at Conferences / Workshops:

[11] Demaine, E. D., Hohenberger, S., and Liben-Nowell, D. (2003). Tetris is hard, even to approximate. Proc. 9th International Computing and Combinatorics Conference (COCOON 2003), pp. 351–363.

[12] Ramon, J. and Driessens, K. (2004). On the numeric stability of gaussian processes regression for relational reinforcement learning. ICML-2004 Workshop on Relational Reinforcement Learning, pp. 10–14.

Websites:

[13] Fahey, C. P. (2003). Tetris AI, Computer plays Tetris. http://colinfahey.com/tetris/tetris_en.html.

[14] Yael Bdolah and Dror Livnat. Reinforcement learning playing Tetris. 2000. URL: http://www.math.tau.ac.il/~mansour/rl-course/student\proj/livnat/tetris.html.

[15] Stan Melax. Reinforcement learning Tetris example. 1998. URL http://www.melax.com/Tetris/.