Learning to Play Tetris with Big Data!

Deng Yue A0164440M, Li Xiang A0115448E, Wu Shuang A0133926A, Zhou You A0133976U

I. Introduction

TETRIS is a video game played on a two-dimensional grid of size 20 rows by 10 columns in this project. Each square in the grid can be full or empty. Each falling object can be moved horizontally and can be rotated by the player in all possible ways. There are 7 possible falling object shapes in this project as shown in Figure 1.

The aim of this project is to create a competent utility-based agent to maximize the score attained (total number of rows removed) up to termination of the game.

In the remainder of this report, we first explain our strategy of the agent and selection of feature functions. Then we describe and evaluate two learning methods in detail. Next we show the scalability to big data. Finally, we conclude this project with discussion and possible future work.

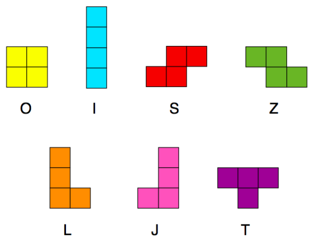


Fig. 1: The 7 Tetris object shape

II. Agent's Strategy

Given board position i and shape of current falling object y, the agents strategy for playing the Tetris game is defined as follows:

$$\mu(i,y) \triangleq argmaxr(i,y,a) + \nu(f(i,y,a))$$

where $r(i,y,a)$ is the reward (i.e., number of rows removed) and $f(i,y,a)$ is the resulting new board position when action a is applied to the falling object y in board position i. ν is the heuristic function. We explain it in detail in the following section II-A.

A. *heuristic function ν*

We construct a heuristic function that evaluates a Tetris board position based on some characteristic features of the position. Consequently, the agents utility function can be approximated by a heuristic function that comprises a linear weighted sum of features

$$\nu(i) = \omega(0) + \sum_{k=1}^{n} \omega(k)\phi_k(i)$$

where n is the number of features, $\omega = (\omega(0), \omega(1), ..., \omega(n))$ is the weight vector, and $\phi_k(i), k = 1, ..., n$, are the feature functions for board position i. We describe the selected feature functions in section III.

We search for the optimal ω using our learning methods. The learning methods are explained in details in section IV.

III. Feature Functions

In the existing literature, Dellacherie introduced six feature functions [2] and Thiery and Scherrer introduced other two feature functions [3] . Recently, Boumaza uses a union of these two sets of feature functions and achieves fairly good results [1]. In this project, we follow Boumaza's approach and use eight feature functions. The details are listed in figure 3.

Other feature functions are also experimented on. However the additional feature functions slows down the player and learner without apparent improvement on the program performance. Thus we choose to follow the literacy and use the aforementioned 8 feature functions.

IV. Learning Method

We try two learning method: genetic algorithm and least-square policy iteration. In this section, We explain the methodology, report the performance and describe our observation and analysis for each learning method in details.

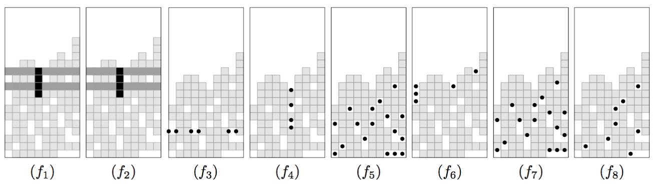


Fig. 2: Feature functions. (f1) Landing height: the height at which the current piece fell. (f2) Eroded pieces: the contribution of the last piece to the cleared lines time the number of cleared lines. (f3) Row transitions: number of filled cells adjacent to empty cells summed over all rows. (f4) Column transition: same as (f3) summed over all columns (f5) Number of holes: the number of empty cells with at least one filled cell above. (f6) Cumulative wells: the sum of the accumulated depths of the wells. (f7) Hole depth: the number of filled cells above holes summed over all columns. (f8) Row hole: the number of rows that contains at least one hole.

A. Genetic Algorithm

1) Methodology: Genetic Algorithm (GA) is an adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics [4]. It is based on an analogy with the genetic structure and behavior of chromosomes within a population of individuals.

During learning stage, our initial individuals are all set to zero values. After about two hundred generations, the weights converges to value listed in appendix VII-A.

2) Performance: To measure its performance, we have used the weights to run PlaySkeleton for one hundred times. The result is listed in figure 3 and table I.

TABLE I: Performance summary for 100 games with weights learned by GA

Number of Games	Min	Max	Median	Mean
100	38,812	5,394,253	977,628	1,226,003

3) Observation and analysis: When generating genes, we need to set upper and lower boundary. We research in different values, and find that when the genes are generated without boundary, the weights could become infinity or negative infinity in a few generations. When the boundary is set too low or too high, the results are also not so good. Lastly, we find that one million is a good value and obtain weights in appendix VII-A.

The fitness function of our GA is: for one particular set of weights, we run it only once and the score attained(total number of rows removed) is the returned score of fitness function. However, GA sometimes might reach the local maximum value. Therefore, our final weights learned by GA are not the optimal weights.

B. Least square policy iteration

1) Methodology: Least Square Policy Iteration (LSPI) is one of the implementations of Least Square Temporal Difference (LSTD) and it is a form of supervised learning. It can be used to solve learning problems which can be approximated by linear architectures to evaluate feature functions values. It is chosen due to its simplicity of implementation, debugging and suitability in the Tetris problem in which weight can be modeled in a linear architecture.

We implement the optimized version of LSTD implemented, which is shown in the figure 4.

2) Performance: While greater number of states learned means better result, the LSPI learner is able to generate fairly good result after learning a small number of states. It is able to clear average of more than 1 million rows after 20 iterations and learning 100,000 states per iteration with random generated initial weight.

In order to generate a set of good weights, learner is fed with 7,000,000 states for each iteration and the weight obtained is shown in appendix VII-B. 100 games is played with this set of weights and the result is shown in Figure 5 and table II.

TABLE II: Performance summary for 100 games with weights learned by LSPI

Number of Games	Min	Max	Median	Mean
100	6,104	8,528,014	1,123,703	1,633,164

3) Observation and analysis: In LSPI, it is required to explicitly define the rewarding policy and the discount factor. In the case of our project, these values are chosen based on the proven success in previous Tetris projects and research done about Tetris game. Different parameters is also experimented on and the current set used in our project is the best of what we found so far.

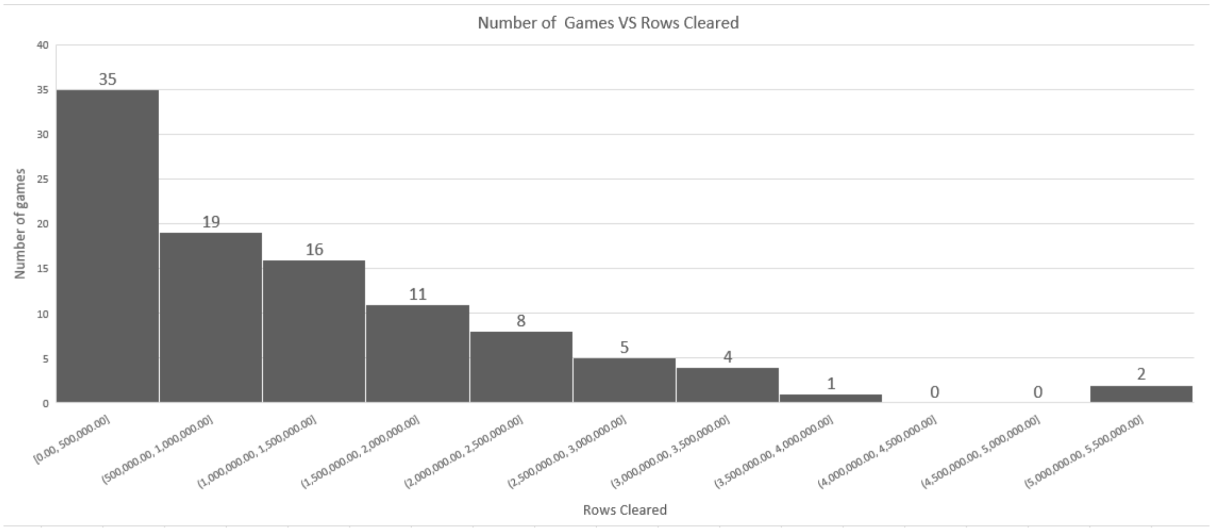
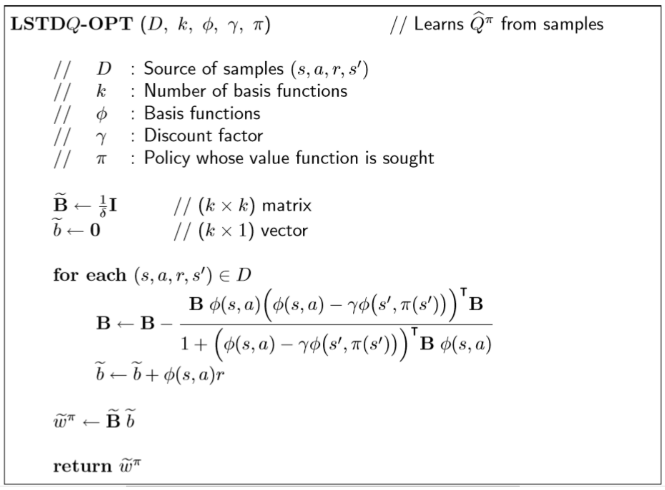


Fig. 3: Rows cleared with weights learn t by genetic algorithm



```
LSTDQ-OPT ($D$, $k$, $\phi$, $\gamma$, $\pi$)          // Learns $\widehat{Q}^\pi$ from samples

    // $D$ : Source of samples $(s,a,r,s')$
    // $k$ : Number of basis functions
    // $\phi$ : Basis functions
    // $\gamma$ : Discount factor
    // $\pi$ : Policy whose value function is sought

    $\widetilde{\mathbf{B}} \leftarrow \frac{1}{\delta}\mathbf{I}$      // $(k \times k)$ matrix
    $\widetilde{b} \leftarrow \mathbf{0}$      // $(k \times 1)$ vector

    for each $(s,a,r,s') \in D$
        $\mathbf{B} \leftarrow \mathbf{B} - \frac{\mathbf{B}\,\phi(s,a)\Big(\phi(s,a) - \gamma\phi\big(s',\pi(s')\big)\Big)^{\mathsf{T}}\mathbf{B}}{1 + \Big(\phi(s,a) - \gamma\phi\big(s',\pi(s')\big)\Big)^{\mathsf{T}}\mathbf{B}\,\phi(s,a)}$
        $\widetilde{b} \leftarrow \widetilde{b} + \phi(s,a)r$

    $\widetilde{w}^\pi \leftarrow \widetilde{\mathbf{B}}\,\widetilde{b}$

    return $\widetilde{w}^\pi$
```

Fig. 4: An optimized implementation of the LSTDQ algorithm.

V. Discussion

In the case of our project, the best weight set generated by LSPI learner performs better than GA learner. As we can see from Table I and Table II, the average number of rows cleared by GA is about 1.2 million while that by LSPI is about 1.6 million.

With regard to stability, LSPI consistently outputs similar results with the same parameters while the weight generated by GA learner can differ greatly. Since GA depends largely on mutation chance, there is no guarantee that the weight generated will be better than the previous weight. In comparison, LSPI is more deterministic and the learned result is generally improved when more states are learned. The randomness of the states generated for LSPI also affects the LSPI learner less than that of the mutation chance affects GA learner.

Regarding learning time, LSPI is very suitable for the linear architecture we adopted. Our heuristic function ensures that LSPI is able to learn efficiently within a short period of time. If we change the heuristic function into polynomial with larger degree, GA may perform much better.

In terms of the scalability, we believe Genetic Algorithm can perform better as more genes and populations can be provided for evolving. However, multiple LSPI learners can hardly contribute to each other learners' weights since there is no obvious relationship unless extra steps are carefully introduced to consolidate learning results.

VI. Scalability to big data

Our learning method are designed to be capable of learning a proficient Tetris agent using millions of Tetris objects in a sufficiently short time. To achieve this, we develop multi-threaded learner program. Each Tetris learner node runs on a single thread and at the end the learning results are consolidated.

We use LSPI to demonstrate the scalability to big data. At the end of each iteration, the learned weights are consolidated by Stochastic Gradient Ascent algorithm. As shown in table III, With 1,000,000 sample size and 20 iteration, x4.34 speedup can be achieved. [1]

[1]Macbook Air. Processor:1.3 GHz Intel Core i5. Memory: 8 GB 1600 MHz DDR3

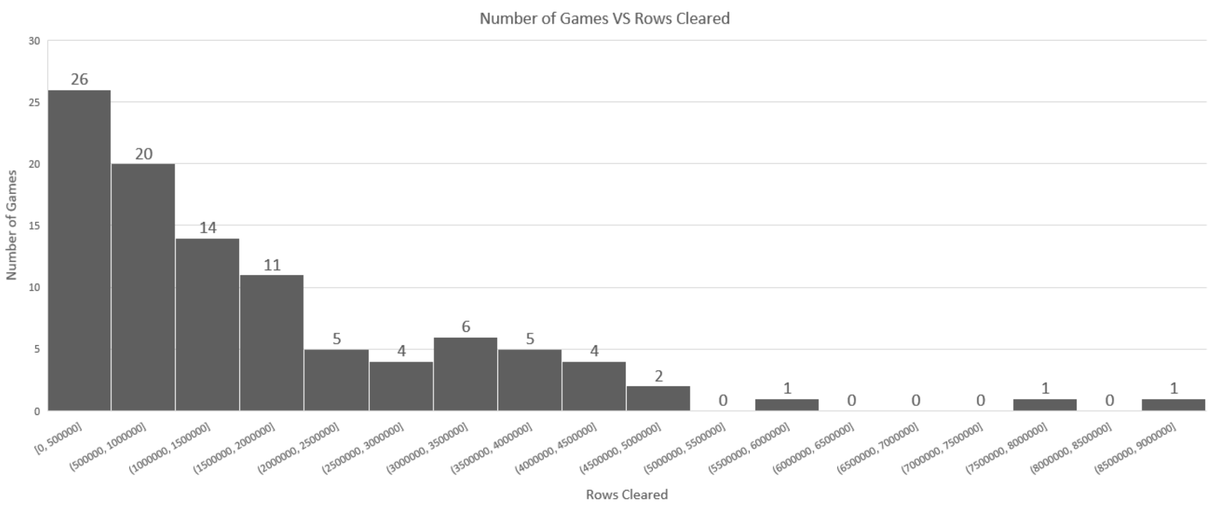


Fig. 5: Rows cleared with weights learnt by LSPI.

TABLE III: Time to learn with 1 million samples and 20 iterations

1 Thread	4 Threads	Speed-up
1,480s	341s	4.34

VII. Conclusion

We experiment on two learning algorithms and in the case of our project, LSPI is proven to perform better than Genetic algorithm. The result is fairly satisfactory as average number of rows clear is more than 1.6 million.

The learning method we adopted can also be easily scaled up to big data to improve its performance further. In accordance to the recent trend of big data, our learning method can indeed be run in a parallel fashion to achieve much better speed and performance.

In the future, we can try different combination of feature functions and explore other learning method such as Particle Swarm Algorithm and Ant Colony Optimization.

References

[1] A. Boumaza. How to design good Tetris players. 2013. hal-00926213

[2] C. P. Fahey. Tetris AI, Computer plays Tetris. 2003. On the web http://colinfahey.com/tetris/tetris_en.html.

[3] C. Thiery. B. Scherrer. Construction dun joueur artificiel pour Tetris. 2009.Revue des Sciences et Technologies de lInformation - S erie RIA : Revue dIntelligence Artificielle, Lavoisier, 2009, Mod elisation et d ecision pour les jeux, 23 (2-3), pp.387-407. . .

[4] Introduction to Genetic Algorithms(1996). Introduction to Genetic Algorithms [Online]. Available: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/hmw/article1.html.

Appendix

A. *Weights Generated by Genetic Algorithm*

The initial weights are all set to zero. These weights are generated by Genetic Algorithm after about two hundred generations.

Feature	Weight
landing height	-471,940.8723823464
radio segment size	232,255.79893859173
row transition	-626,283.2377368517
column transition	-844,431.1082631972
number of holes	-991,800.4234056593
Well sum	-157,305.5448753127
hole depth	-746,378.8844266993
avg. height of columns	-895,861.1996044472

B. *Weights Generated by LSPI*

This weight is generated by LSPI after learning 7,000,000 states per iteration and going through 20 iterations. The discount factor is set to be 0.9 and the winning reward is set to be 1,000,ll000.

Feature	Weight
landing height	-16442.074156262446
radio segment size	6696.8428940551885
row transition	-28190.894358102174
column transition	-34818.68429996376
number of holes	-17374.73731900446
Well sum	-4867.585098512209
hole depth	-50817.08977970609
avg. height of columns	-45345.976872643456