AI for Tetris

Hongchen Cao
2019533114
caohch1@shanghaitech.edu.cn

Chenbo Xi
2019533113
email address

Ziang Geng
2019533112
gengza@shanghaitech.edu.cn

Jiawen Yang
2019533175
yangjw@shanghaitech.edu.cn

Fuyi Yang
2019533170
yangfy@shanghaitech.edu.cn

***Abstract*—Artificial intelligence (AI) techniques, especially machine learning and deep learning, have shown amazing potential in the field of decision making (e.g., Go, poker). Compared with traditional board and card games, video games have more complex rules and larger state spaces, which makes it challenging to develop effiective AI agents for video games. Tetris is a classic and popular video game. In this paper, we design three AI agents for Tetris, including heuristics algorithm, genetic-beam search, and deep Q-learning, and conduct a comprehensive evaluation. The Comparison experiments show that the Q-Learning Agent has the best performance in general, which arrives 21887.584 for average scores, and 934.67 for average cleared lines by fixing the number of Tetrominoes to 2500.**

***Index Terms*—Reinforcement learning, Video game, Heuristic algorithm**

I. Introduction

Nowadays, with the deepening of research, artificial intelligence technology has provided some conveniences for people's daily life. Opportunities for use of this technology are numerous, such as computer science, finance, medicine, diagnostics, and more.

Amazon's many department are inclined to AI, such as unmanned stores, drone delivery of goods, and Alexa[1] voice assistants. For Microsoft, they use AI to focus on real time translating[2]. What's more, AI can also play an important role in games, the current most popular AI agent which is called AlphaGo [3] has won many professional players.

Tetris is one of the most popular video games. It is difficult to quantify and model for a satisfying strategy for human players, but the AI agent can be designed in multiple ways for this game.

In this work,we explore three Tetris AI agents:

•**Heuristic Agent:**As a classic method dealing with search problem, it provide us with a basic but effective idea to get a higher score in Tetris

•**Genetic-Beam Agent:**An algorithm simulates the natural evolution process to find the optimal solution.

•**Q-Learning Agent:**An reinforcement learning based algorithm using Deep Q-Network (DQN).

We also conduct a comprehensive evluation for each agent to test their performance. For Heuristic Agent and GeneticBeam Agent, we explore the effect of their hyper-parameters on performance; for Q-Learning Agent, we explore the effect of the model complexity on performance. The code is open-source and available at our Github [4].

II. Background

Tetris agent has become a classic application of reinforcement learning technology in the field of game agents. As a long-standing game, Tetris has a clear definition of state space and a relatively simple game mechanism.

Actions. The action in Tetris consists of two parameters: the falling position and the number of rotations. The falling position determines the final position of the currently falling tetromino, and the number of rotations determines the final orientation of the tetromino.

States. Similar to most video games, it's hard to accurately quantify and model the state of Tetris. The simplest and clearest method to represent the game state in Tetris is using a $N \times M$ grid. The empty position (i.e., no block occupies) of the grid is represented by 0, and the occupied position is represented by other numbers(e.g., 1, 255).

If the height of a column exceeds N after the block falls, the game ends and the final scores of the game are evaluated. The final score of the game is divided into three parts: the tetrominoes score, the cleared lines score, and the composite score. The details of how each score is calculated are described in Sec. IV.

III. Methodology

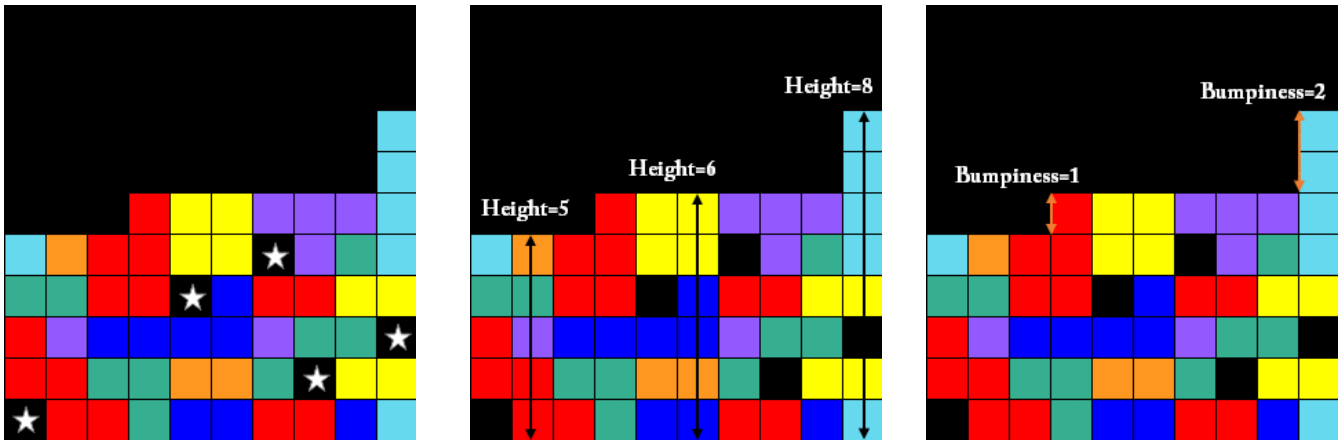
A. Notations and Definitions

The notations used throughtout following sections are shown in Table. I.

TABLE I: Notations

Notation	Illustrations
$\boldsymbol{A}$	action space
$\boldsymbol{a}$	a choosen action
$\boldsymbol{S}$	current state
$\boldsymbol{S'}$	possible next state space
$\boldsymbol{S'(S, a)}$	next state for excute $\boldsymbol{a}$ in $\boldsymbol{S}$
$\boldsymbol{Q(S,a)}$	Q-value for excuting $\boldsymbol{a}$ in $\boldsymbol{S}$

Before diving into the detail of three AI agents, we briefly describe three definition used in the following algorithms.



(a) Holes (b) Block Height (c) Bumpiness

Fig. 1: Description of Holes, Block height and Bumpiness

•**Holes.** We define an empty position as a hole if it is surrounded by non-zero numbers or the border of the board.(See Fig1(a))

•**Block height.** The distance from the lowest non-zero number to the highest non-zero number of each column is the height of the column.(See Fig1(b))

•**Bumpiness.** The list of difference between the column heights of two adjacent columns.(See Fig1(c))

B. Heuristic Agent

The heuristic algorithm is a classic method dealing with search problem, which is similar to our Tetris problem. Thus, it has provided us a basic but effective idea to get a higher score in Tetris, including two steps. Step one, calculate h_value for every action of the dropping tetromino, by our heuristic function. Step two, choose the best action according to h_value, so that we can make our agent maximize its score. The heuristic function is designed as following:

$$h(f_{1:8}) = \sum_{i=1}^{i=8} w_i * f_i \tag{1}$$

where f_i is defined in Table. II To get a better heuristic function, 8 parameters are chosen based on two parts: properties of the game board and ***S'(S, a)***.

TABLE II: Parameters

ID	Parameter name	Illustrations
1	***board_maxHeight***	The max height of the columns in ***S'***
2	***board_avgHeight***	The average height of the columns in ***S'***
3	***board_hole_num***	The number of holes in ***S'***
4	***board_max_diff***	The max difference of height between columns
5	***action_cleared_line***	Whether the action clears a line
6	***action_hole_num***	The number of holes in ***S'(S, a)***
7	***action_bumpiness***	The sum of bumpiness as defined in Sec. III-A
8	***action_maxHeight***	The max height of the columns in ***S'(S, a)***

C. Genetic-Beam Agent

Genetic-Beam search is an algorithm which searches for the optimal solution by simulating the natural evolution process. Inspired by this algorithm, we have conceived our Genetic-Beam agent for the Tetris game, which converts the solution process into a process similar to the crossover and mutation of the chromosomes in biological evolution. The main process of the Genetic-Beam Agent is as follows:

- Step 1: Randomly generating 7 chromosomes(following uniform distribution) and denote them as the first generation.
- Step 2: Taking current generation as heuristic function(see in Alg. 1) of the system and run the game. For each piece falling, using the generation to get the best action. When the game finishes, recording the score.
- Step 3: Applying crossing chromosome(see in Alg. 3) and crossing genetic beam algorithm(see in Alg. 4) to the generation so as to form new generation. During crossing chromosome, each chromosome gene has α chance comes from one of its parent and $1-\alpha$ chance comes from an average of the two parents' respective genes. During crossing genetic beam, the new generation contains β better chromosomes from the previous generation and $1-\beta$ one point crossing between the best chromosomes of the previous generation. Meanwhile, at this stage there is γ chance of gene mutation(see in Alg. 2).
- Step 4: Repeatedly performing Step 2 and Step 3 to get several rounds of evolution for the generation. After comparing the score each generation has got, choosing the best generation and use the chromosomes as the final heuristic function parameters to run the Tetris(see in Alg. 5).

D. Q-Learning Agent

Mnih et al. [5] combine deep neural networks and Q-learning to purpose Deep Q-Network (DQN) and apply it

ALGORITHM 1: Heuristic

Input: generation
Output: h
$h1 \leftarrow generation[1] * action\_cleared\_line$
$h2 \leftarrow generation[2] * board\_hole\_num$
$h3 \leftarrow generation[3] * board\_block\_num$
$h4 \leftarrow generation[4] * board\_maxHeight$
$h5 \leftarrow generation[5] * board\_avgHeight$
$h6 \leftarrow generation[6] * action\_bumpiness$
$h7 \leftarrow generation[7] * board\_max\_diff$
$h \leftarrow h1 - h2 - h3 - h4 - h5 - h6 - h7$
return h

ALGORITHM 2: Mutation

Input: chromosome
if $randInt(1, 10) == \gamma$ **then**
  if $chromosome < certain\_range$ **then**
    return $-0.1 * randInt(1, 5)$
  else
    return $0.1 * randInt(1, 5)$
  end if
else
  return 0
end if

ALGORITHM 3: CrossingChromosome

Input: chrom1, chrom2
Output: NewChrom
if $randInt(1, 10) == \beta 1$ **then**
  $NewChrom \leftarrow Mutation(chrom1)$
else if $randInt(1, 10) == \beta 2$ **then**
  $NewChrom \leftarrow Mutation(chrom2)$
else
  $NewChrom \leftarrow (chrom1 + chrom2)/2$
end if
return $NewChrom$

ALGORITHM 4: CrossingGeneticBeam

Input: generation
Output: new_generation
$ordered\_generation \leftarrow sorted(generation, key = score)$
$length = len(ordered\_generation)$
for i = 1 to length*α **do**
  $new\_generation \leftarrow ordered\_generation$
end for
for i = length*α to length **do**
  $new\_generation \leftarrow$
  $CrossingChromosome(generation, ordered\_generation)$
end for
return $new\_generation$

ALGORITHM 5: GeneticBeam

Input: round
Output: best_generation
$generation \leftarrow UniformDistribution(chrom)$
for i = 1 to round **do**
  $score \leftarrow Tetris(generation, Heuristic(generation))$
  $new\_generation \leftarrow CrossingGeneticBeam(generation)$
  $generation \leftarrow new\_generation$
end for
$best\_generation \leftarrow sorted(generation, key = score)$
return $best\_generation$

to classic Atari 2600 games. Inspired by them, we design a DQN Agent for Tetris. Similar to most video games, it's hard to accurately quantify and model the state of Tetris, so we carefully select four features to represent the game state ***S'(S, a)***:

- cleared lines: the number of lines to be cleared after excute action ***a*** in state ***S***;
- holes: the number of holes defined in Sec. III-A;
- bumpiness: the sum of bumpiness as defined in Sec. III-A;
- block height: the sum of block height of each column.

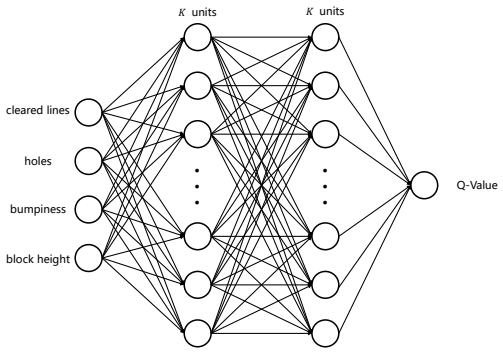


Fig. 2: MLP for Q-Learning

We build a simple MLP (See Fig. 2) to predict the Q-Value accorading to the four input features. The MLP has two hidden layers, each of both has K units. We explore the effect of K on performance in Sec. IV-C.

IV. Evaluation

Experiment settings. To cater for the needs of our methods, we modify and integrate some Tetris implementations on GitHub [6], [7], [8], [9]. To evaluate the three methods of Tetris agent, we set $N = 10$, $M = 20$ (i.e., a 10×20 grid) as our test environment.

Evaluation metric. In order to evaluate the efficiency of methods implementing Tetris agent, we choose three scoring criteria to quantify the performance of different methods. The detailed definition of the three metrics are as follows:

•**Tetrominoes.** The Tetrominoes is the number of tetrominoes fallen in the whole game.

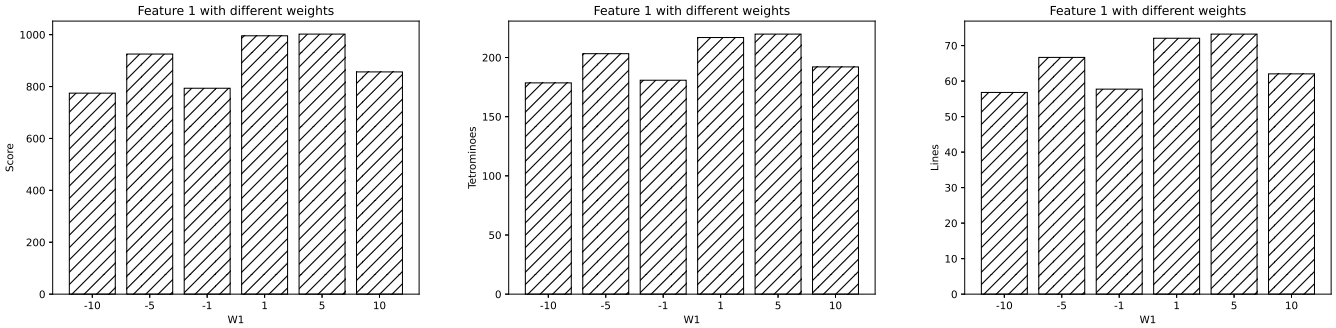
•**Lines.** The lines is the number of rows been removed in the whole game.

•**Scores.** To calculate the final scores, for each falling block, we calculate a current score using the following method:

$$currentScore = 1 + C^2 * N \tag{2}$$

where C represents the rows being cleared in the current step and N represents the width of the grid. After getting the score for each step, we calculate the Scores by the following equation:

$$Scores = \sum_{i=1}^{gameOver} currentScore_i \tag{3}$$



(a) Scores (b) Tetrominoes (c) Lines

Fig. 3: Heuristic appearance for ***board_maxHeight*** with different $w1$ on Scores, Tetrominoes, and Lines

A. Heuristic Agent

To maximize the performance of the agent, we experiment different weight sets for all eight features defined in Table. II. For each weight, we run 300 turns of game and average its game scores, tetrominoes and lines. Due to page limitation, all detailed performance are shown in Fig. 9 and Fig. 10. (See Sec. VI)

As shown in Fig. 3, there is not much difference in the performance of Heuristic Agent with feature1 given different weights. Other features related to ***S'*** are similar. No matter how we choose the weights of feature 1 to 4, the game score fluctuates around 950 (average is 937.6458). However, compared with features related to ***S'***, features correlative to ***S'(S, a)*** have greater impact on the performance of the Heuristic Agent. Unlike board-related features (i.e. feature 1-4), there is a vast difference among the different weight of action-related features (i.e. feature 5-8). Feature5 *action_cleared_line* performs an average score of 945.475 at positive weight but poorly at negative weight. Considering the property that f_5 is permanently larger than 0, the possible reason is that the positive weight and make f_5 have a large proportion in the calculation of heuristic function. It makes agent prefer to clean lines as many as possible. If it has a large negative weight, the tendency of the agent is not to eliminate lines. Similarly, the agent perform very well when it tends to eliminate more holes on the board and keeps the block flat. (Feature *action_hole_num* and *action_bumpiness*)

B. Genetic-Beam Agent

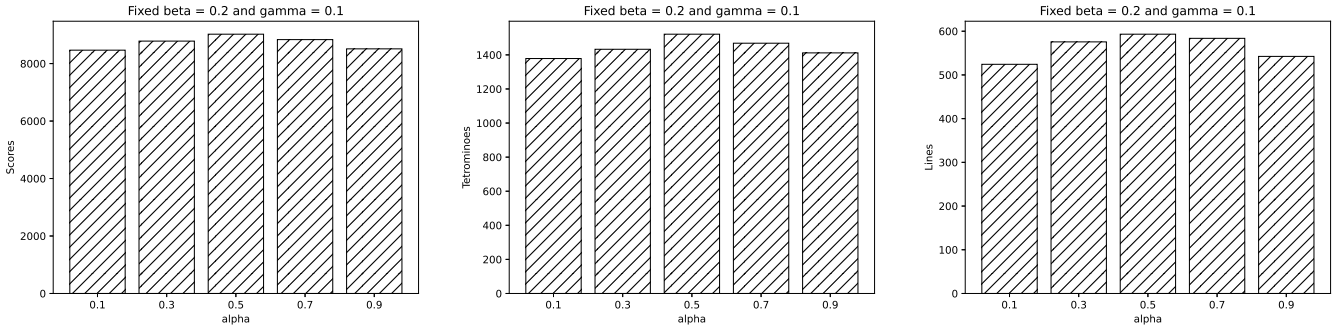
We evaluate the performance of Genetic-Beam agent with different selection of α, β and γ. For each case, we add a time limitation of 5 minutes to run the game. We run 300 turns of game for each case and average its score at last. From our experiment, we find that for fixed β and γ, the best scores, lines and tetrominoes come when α = 0.5 and decrease as α goes higher or lower(shown in Fig. 4). The possible reason for this result is that α determines the rate of better generation and crossing chromosomes, in that case it will be better for 50 % chance to do crossing chromosomes. For fixed α and γ, the best scores, lines and tetrominoes come when β = 0.2 and decrease as β goes higher(shown in Fig. 5). The possible reason for that result is that β decides the rate of chromosome coming directly from its parent, therefore the smaller β is, the higher chance that chromosome is the average of the two parents' respective gene. For fixed α and β, the best scores, lines and tetrominoes come when γ = 0.1 and decrease sharply when γ goes higher(shown in Fig. 6). The possible reason for this may be that γ represent the rate of mutation, temperate mutation may get better chromosome, however when the rate is high, the chromosomes tend to be perform randomly and badly.

C. Q-Learning Agent

As shown in Fig. 7, we evaluate the performance of MLPs with different units number K. For each model, we run 3000 turns of game and average its scores. However, the MLP32 achieve a such great performance that it hardly reaches game over conditions declared in Sec. II so we define it as *inf*. All MLPs that more complex than MLP32 have worse performance, and the performance decreases as the model complexity increases. The possible reason is that the input feature is only a four-dimensional vector, and the complex models meet over-fitting so the performance get worse. MLP16 performs bad since it has too few parameters to fit the states and actions of Tetris.

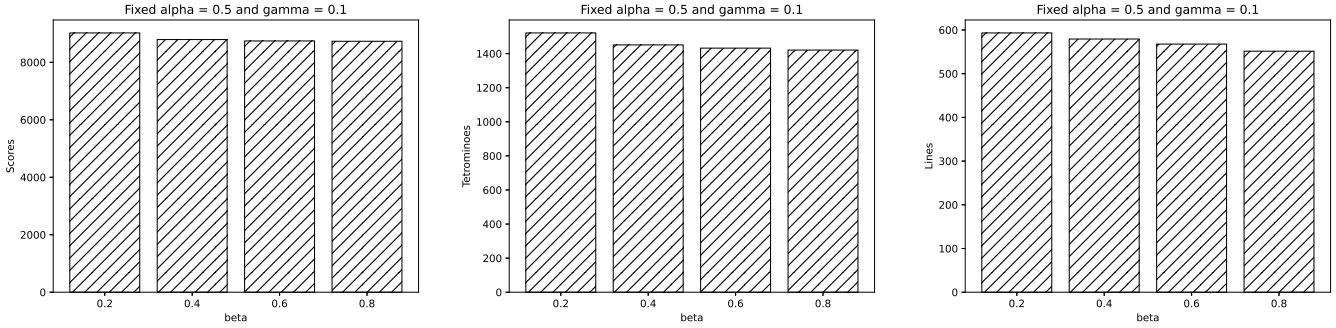
D. Comparison

We evaluate the performance of Three agents by fixing the number of falling tetrominoes. For each agent, we run 500 turns with 500, 1000, 1500, 2000, 2500 tetrominoes respectively to get the average score. As shown in Fig. 8, the Q-Learning agent generally gets higher scores than the other methods. As the number of tetrominoes grows, the gap of scores between Q-Learning and the other agents becomes more perceptible. However, when the number of tetrominoes reaches 1500 or more, there is an obvious difference of the average completed tetrominoes and eliminated lines between the Heuristic agent and the other agents. It shows that the Heuristic agent is not efficient enough to accomplish the requirements. The Genetic-Beam agent usually survives the longest time and eliminates more lines, though its score



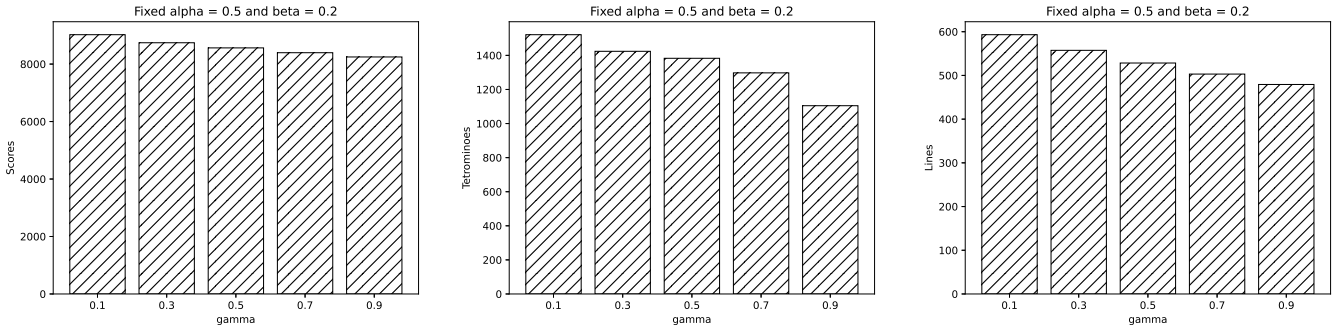
(a) Scores (b) Tetrominoes (c) Lines

Fig. 4: Scores, Tetrominoes, and Lines when β and γ are fixed



(a) Scores (b) Tetrominoes (c) Lines

Fig. 5: Scores, Tetrominoes, and Lines when α and γ are fixed



(a) Scores (b) Tetrominoes (c) Lines

Fig. 6: Scores, Tetrominoes, and Lines when α and β are fixed

is generally lower than the Q-Learning's. According to our definition of *Scores*, the possible reason is that the Genetic-Beam agent prefers to eliminate lines in each step while the Q-Learning agent prefers to store the tetrominoes then eliminate multiple lines in one step to get a higher score.

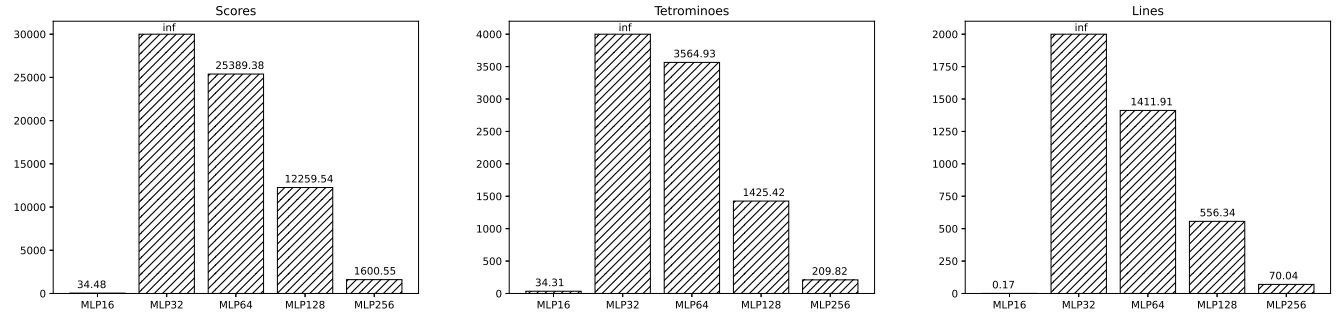
V. Conclusion

We design three AI agents for Tetris, including heuristics algorithm, genetic-beam search, and deep Q-learning, and conduct a comprehensive evaluation for the hyper-paramaters of each agent. We also explore the potential reasons for the effects of the hyper-parameters. Among there agents, both Genetic-Beam Agent and DQN Agent can achieve a great performance that hardly meet game over with proper hyper-parameters. The Comparison experiments show that the Q-Learning Agent has the best performance in general, which arrives 21887.584 for average scores, and 934.67 for average cleared lines by fixing the number of Tetrominoes to 2500.

References

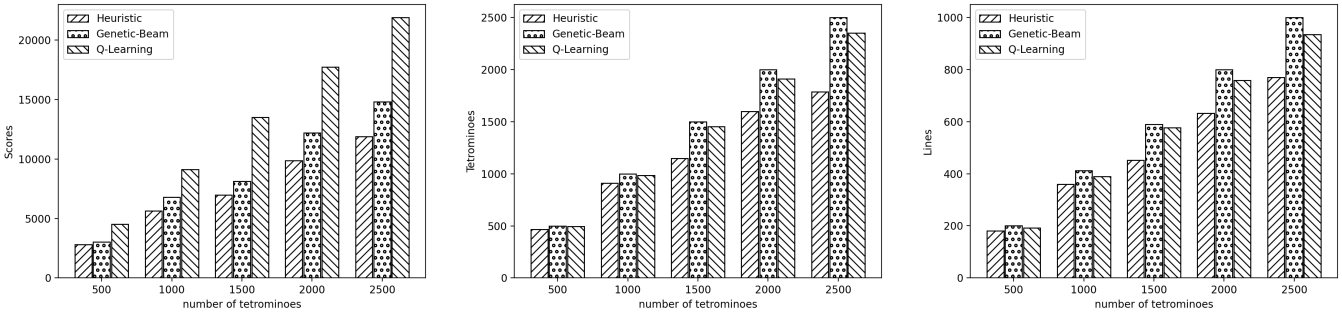
[1] "wiki for amazon alexa," 2022. [Online]. Available: https://en.jinzhao.wiki/wiki/Amazon_Alexa

[2] "wiki for microsoft translator," 2021. [Online]. Available: https://en.jinzhao.wiki/wiki/Microsoft_Translator



(a) Scores (b) Tetrominoes (c) Lines

Fig. 7: Effect of units number K in MLP on Scores, Tetrominoes, and Lines



(a) Score (b) Tetrominoes (c) Lines

Fig. 8: Three Agents' Performance with different condition

[3] "wiki for alphago," 2021. [Online]. Available: https://en.jinzhao.wiki/wiki/AlphaGo

[4] "Ai-tetris agent," 2022. [Online]. Available: https://github.com/caohch-1/AI-Project

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nat.*, vol. 518, no. 7540, pp. 529–533, 2015.

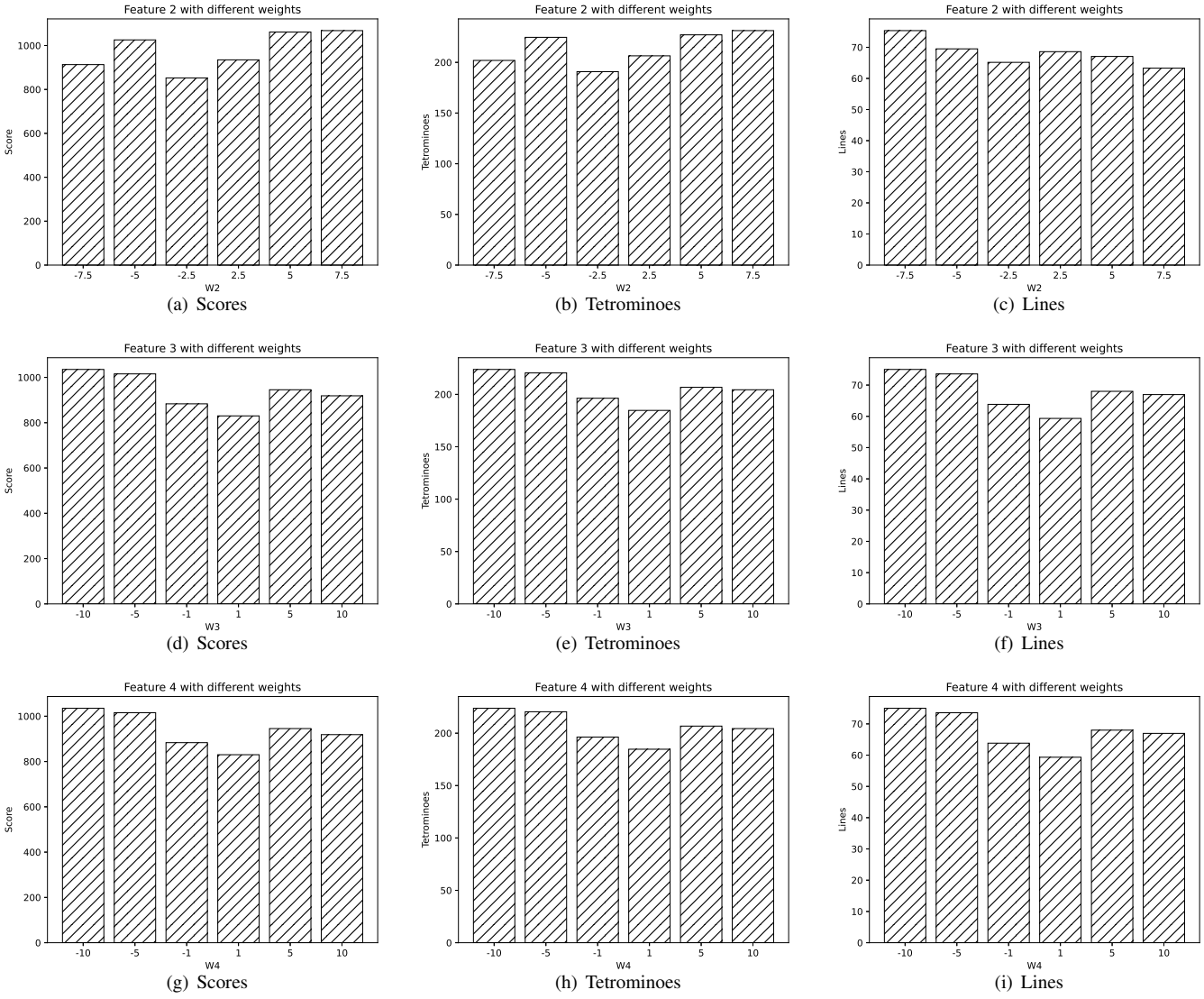
[6] "Tetris game," 2018. [Online]. Available: https://github.com/LoveDaisy/tetris_game

[7] "Tetrisrl," 2020. [Online]. Available: https://github.com/jaybutera/tetrisRL

[8] "tetris," 2020. [Online]. Available: https://github.com/yoshiki-kokubo-kbt/tetris

[9] "Tetris-deep-q-network," 2021. [Online]. Available: https://github.com/william-r-austin/Tetris-Deep-Q-Network

VI. Appendix

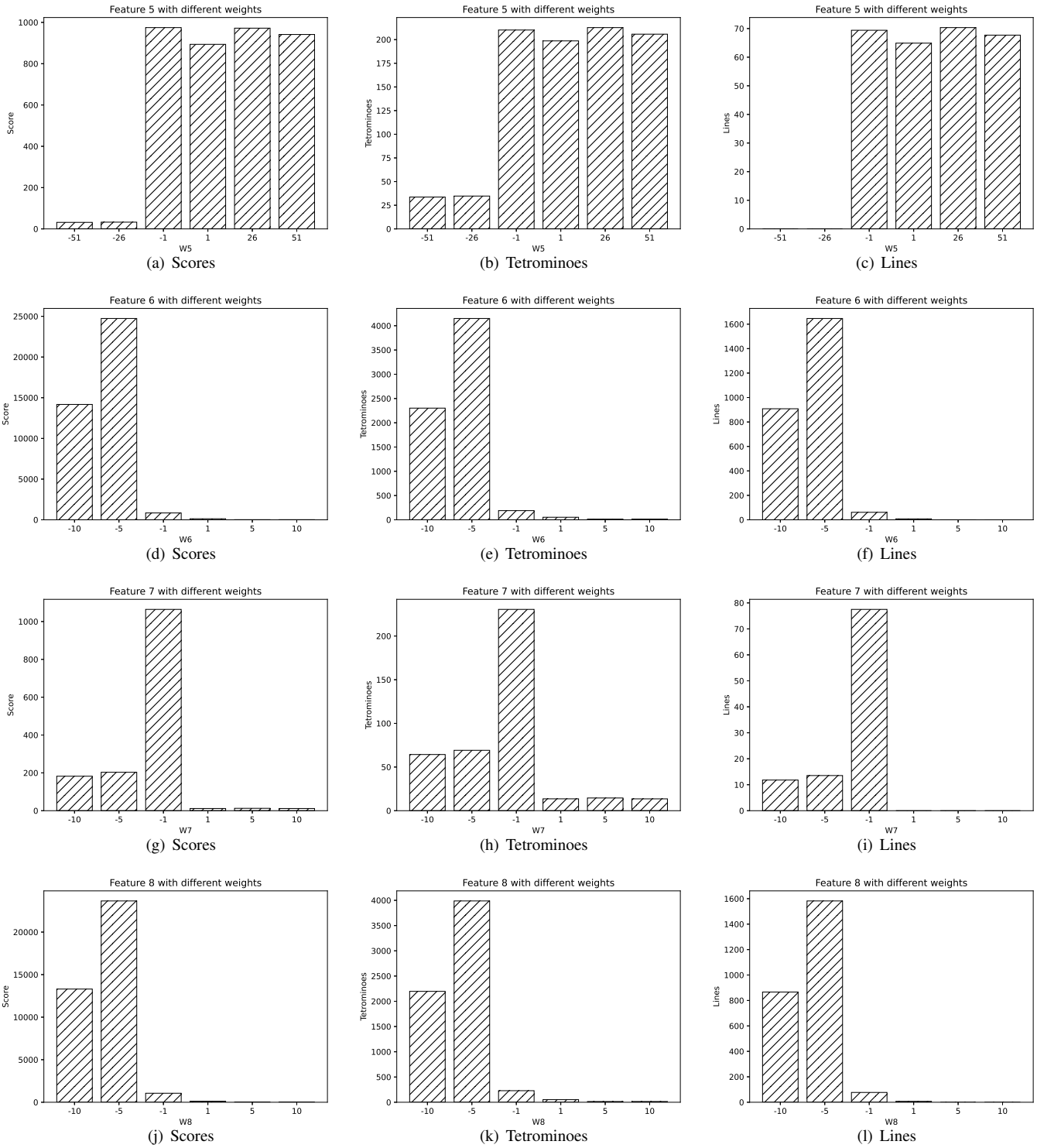


(a) Scores (b) Tetrominoes (c) Lines

(d) Scores (e) Tetrominoes (f) Lines

(g) Scores (h) Tetrominoes (i) Lines

Fig. 9: Heuristic appearance for fi with different Wi for i=(2,3,4) on Scores, Tetrominoes, and Lines



(a) Scores (b) Tetrominoes (c) Lines

(d) Scores (e) Tetrominoes (f) Lines

(g) Scores (h) Tetrominoes (i) Lines

(j) Scores (k) Tetrominoes (l) Lines

Fig. 10: Heuristic ppearance for fi with different Wi for i=(5,6,7,8) on Scores, Tetrominoes, and Lines