Comparing Between Reinforcement learning Approach and Other Methods for Tetris AI —In Memory of Jonas Neubauer

YIRAN MENG, YICHAO ZHANG

Problem Description

After the game between GO champion and Alpha-GO, AI gaming became an interesting way to explore games. We want to find an approach for play classic game Tetris, with the deep learning technique. With reinforcement learning AI can perform better than human players. In this project, we're going to explore an approach to play Tetris.

Motivation

This project is in memory of a legendary classic Tetris player, 7 time world champion, Jonas Neubauer, who passed away January 2021 due to a medical emergency. We send our condolences to his loved ones. He did a tremendous job to bring world's attention to the game classic Tetris. His spirit will forever be remembered.

What we intend to do

What we intend to do is training a model that can play Tetris and get higher score just like human players do.

Research

Literature Survey

- Voronoi Based Multi Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning
  https://ieeexplore.ieee.org/document/9244647
- Neural Basis of Reinforcement Learning and Decision Making
  https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3490621/
- Reinforcement Learning vs Genetic Algorithm AI for Simulations
  https://medium.com/xrpractices/reinforcement-learning-vs-genetic-algorithm-ai-for-simulations-f1f484969c56
- What is reinforcement learning
  https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/
- TETRIS AI WITH REINFORCEMENT LEARNING
  https://www.slideshare.net/JungkyuLee1/tetris-automatic-playing
- Evolving Tetris AI based on genetic algorithms
  https://github.com/mzmousa/tetris-ai
- Deep Q-learning for playing Tetris
  https://github.com/uvipen/Tetris-deep-Q-learning-pytorch/blob/77445d57b67e03683563ce0a28c869da38150319/READ

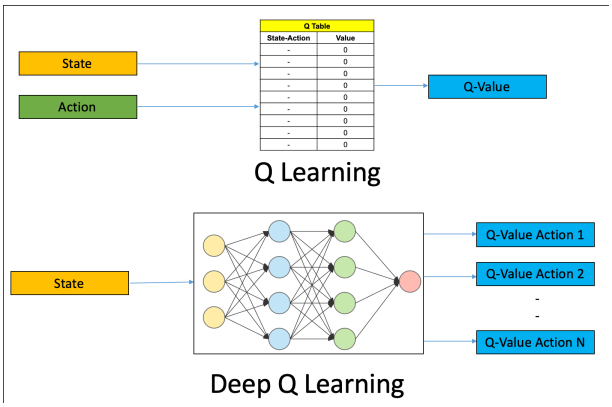
Reinforcement Learning

Reinforcement Learning(RL) is a type of machine learning technique that enables an agent learns to achieve a goal in an uncertain, potentially complex environment with trial and error using feedback from its experiences.This method assigns positive values to the desired actions to encourage the agent and negative values to undesired behaviors in order to seek long-term and maximum overall reward to achieve an optimal solution.

Deep Q-Learning

The difference between deep Q learning and Q learning is it replaces the regular Q-table with a neural network. Rather than mapping a state-action pair to a q-value, a neural network maps input states to (action, Q-value) pairs.

The reason for using Deep Q Learning instead of Q learning is that in Q learning, the agent is not aware of the environment. The agent was only awarded or punished by its actions. However, in Tetris, at different situation, the decision making policy must be different. For example, when the agent gets an I piece, if there's a chance to perform a Tetris (clear 4 rows at a time), the best option is to do so, however, when there's not, it is a horrible move to stack an I piece on top of a flat surface.



Same as Q learning, deep Q learning select the machine with the highest current average payout with probability, where the epsilon / k is the probability of the Q value.

$$p = ((1–\epsilon) + (\epsilon/k)) \times 100\%$$

So basically, there would be p percent of the chance that the agent will follow the DQN policy network. Otherwise, the agent will randomly take actions.

The loss function is mean squared error of the predicted Q-value and the target Q-value. However, we do not know the target or actual value here as we are dealing with a reinforcement learning problem. Q-value is updated equation from Bellman equation:

$$\Delta Q(S_t, A_t) = \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

$$Q^*(S_t, A_t) \leftarrow Q(S_t, A_t) + \Delta Q(S_t, A_t)$$

Technical Details

Tetris

The project applies the most classic tetris rule. There will be a board with 10 blocks width and 20 blocks height. There will be one of the 6-kind bricks drop from top randomly, the player can move the pieces laterally and rotate them until they touch the bottom of the field or land on a piece that had been placed before it. Only the whole row is fully filled, the line can be cleared. The score is counted by the blocks placement and the number of lines cleared this time $Lines$. The initial score is -2.The more lines be cleared once the more score earned, the equation is:

$$\Delta Score = Lines^2$$

Additionally, only 230 lines can be cleared, because in classic NES Tetris, it takes 230 lines to level up to level 29, where the speed gets faster than frame rate and the game becomes unplayable for any human players. This rule restricts that the game is not all about survival, the AI has to plan more carefully about each placement to get the maximum score.

Deep Q-Net

In this project, we used the Deep Q Network as our method. The network consists of two hidden layers with hidden dimension of 64, and ReLU activations. Output dimension is set to 1, which is the Q value of the current state of the Tetris board. The state of the Tetris board is determined by the number of lines cleared, holes created, bumpiness, and current height. Back propagation of the Deep Q Network only happens when a game is over and when the number of states has stacked up to a certain amount. To evaluate the performance of the network, a randomly drafted batch (in our case, Batch size is 8192 and the memory size is 30000, so 8192 states are drafted from 30000 logs) of the states and their corresponding next state (after current action taken) will be used as input of the DQN model to generate the Q value for current state and next state. Then a y value will be calculated as below:

$$y = R + \gamma \times Q_{next}$$

,

If the current state does not end the game. The γ value is selected to be 0.99. Otherwise, $y = R$, which R is the addition of score of the current action.

Then the Q value and y value will be compared through Mean Squared Error loss function:

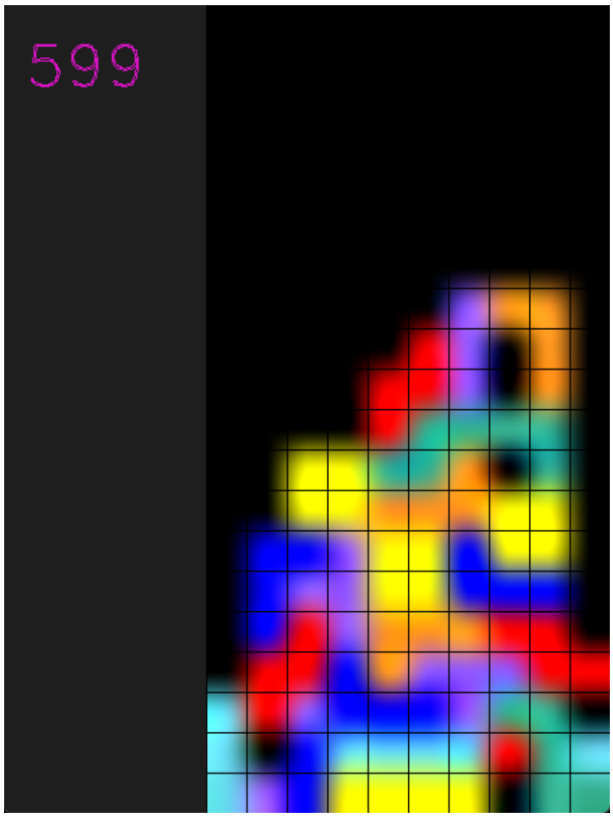
$$L = \sum_i (y_i - Q_i)^2$$

Note here, epoch 1 does not start from the first game because there needs to be enough states stored in the memory to start making any meaningful training.

Result

Final Result

The AI cleared 230 lines, the maximum number lines we set and it is able to find the way to get more score. It will place the bricks in shape and wait for the I brick to achieve a 4 lines clear.

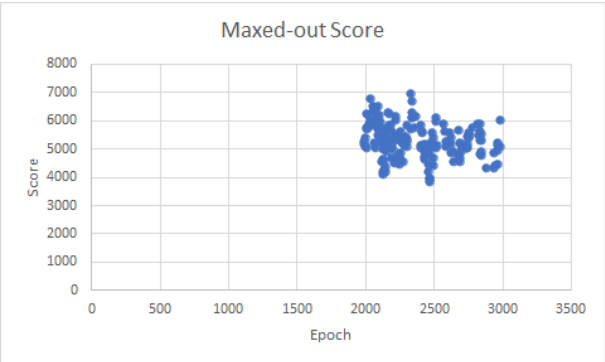


What we intend to do What we intend to do is training a model that can play Tetris and get higher score just like human players do.

What we have accomplished We have accomplished a AI that can play Tetris and clear the max lines and earn more score than we did. The AI can get about 6000+ score.

Model Training

We trained 3000 epochs. At the first couple hundreds of epoches, because the p value is still low, most likely the agent will randomly drop the pieces. So the agent barely clears any lines. After some time of training, it started to realize that it's a bad decision to stack the piece on top of each other when there's available space on the right or left. So it will decide to move the pieces left or right to stack evenly to stack more pieces because each piece drop will add 1 point to the reward. Throughout the process, it will randomly clear a line or two. When this happens more often, the model realizes clearing lines will add tremendously more score than droping pieces alone. But so far, it hadn't found out the more effective way to do so. In the next few hundreds of epoches, it will slowly learn its way to stack more uniformly so it can clear more lines. It will mostly clear about 5-7 lines for a few hundred epoches. At around 1900 epochs of training, the AI can clear 230 lines, which is the maximum number of lines it could clear before the game ends. At this point, the model has already learned how to survive continuously. When this happens multiple times, the model slowly starts to find ways to clear more lines at a time for higher reward. Finally, at line 2331st epoch, the model scored 6995 points, which is the best model.The diagram below shows the heat map of the scores when the model cleared the maximum amount of lines.



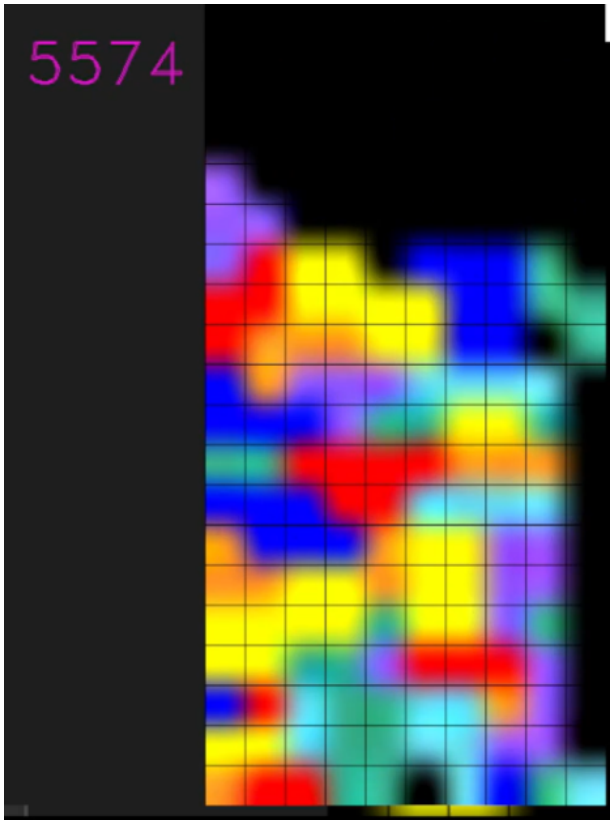


Figure 1: best model demo

Compare with Other's Work

Other Deep Q Learning projects In this project, the AI learns to decide the piece's pose and position to drop, then calculated the closed-form control sequence consists of moving, rotating and T-spinning to make that drop happen. So in our model, each action is a tuple of 2 values, the position and orientation of the drop. However, others have tried other methods. In Jungkyu Lee's project TETRIS AI WITH REINFORCEMENT LEARNING, he trained the deep Q Network with similar structure and parameters, but he used a queue of keyboard movement as actions, instead of a tuple of position and rotation. Also, in rewards, he enabled fast dropping as a rewarding factor. His model takes longer to train because the AI struggles to find the pattern of each key operation.

By comparing to his work, it raised a question. If a closed form solution is available, is there still needs to train an AI? In this case, if there's a way to compute the movement for each desired piece drop, why do some projects still train actions from scratch? Lee approached to this problem with another perspective. In Lee's model, when the model "sees" the current state of the game, it does not respond with where it wants the next piece to drop, instead, it responds with a series of actions. It is optimal if the game gets more complicated, such as combative games where there's no closed form solution to achieve the temporary goal because the situation varies continuously. In this setting, lower level training might give a better solution.

Genetic Algorithm Genetic algorithm is another approach other people have used to make a Tetris AI. In Zuhry Mousa's project Evolving Tetris AI based on genetic algorithms, an iterative evolving batch of genomes are used to train the Tetris AI. Instead of using a neural network, Mousa tried to implement a linear function to get the reward value from each piece drop. Each linear function is a genome. In the first generation, the values are randomized for each genome, then the genomes which scored among the highest will be the parents for the next generation. Their children mutate from the parents by adding or subtracting mutation rate for each linear coefficient of the genome. All possible combinations of mutations will fill the next generation. By 25 generations, Mousa is able to get a much higher score.

In genetic algorithm, the mutation rate is a parameter that needs attention. Tuning mutation rate is way harder than tuning learning rate in traditional machine learning. Sometimes such mutation will gain a small reward, but mutated the opposite way from the optimal model. Because of the ability to add pooling layers and dropout layers, these issues might not be common in traditional machine learning approaches.

CODE

https://github.com/Baturhoo/dl2021f