Reinforcement Learning Methods for Tetris NES

Noam Manaker Morag Itai Bear

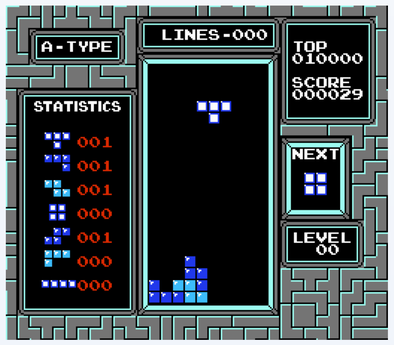
Department of Computer Science, University of Haifa

ABSTRACT

This project aims to train an AI agent to play the NES version of Tetris using Reinforcement Learning (RL) methods. We focus on two gameplay scenarios: a 'Real-Time Agent' that controls every move and a 'Per-Tetrimino Agent' that decides the final position of each tetrimino. We reimplemented an existing Particle Swarm Optimization (PSO) method to successfully train the Per-Tetrimino Agent. However, the Real-Time Agent did not achieve consistent success in line-clearing despite applying methods like Deep Q-Learning, Reward Shaping, Curriculum Learning, and Imitation Learning. We also trained a Value Network that estimates the best board state based on different tetrimino placements, using a reward function derived from the PSO algorithm. Additionally, we updated the gym-tetris environment to meet modern gymnasium standards.

TETRIS NES

In Tetris NES, players arrange falling geometric blocks, known as tetriminos, within a grid. The objective is to fill complete horizontal lines, which then disappear, earning points and freeing up space.

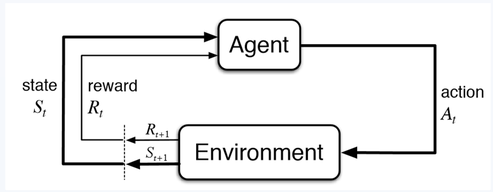


Screenshot of Tetris NES gameplay

The NES (Nintendo Entertainment System) version of Tetris includes several nuances that must be accounted for, such as a Delayed Auto Shift (DAS) mechanism. DAS initially shifts the active Tetrimino one cell horizontally upon pressing either the Left or Right buttons. Holding down these buttons results in the game automatically shifting the tetrimino every 6 frames, but only after an initial delay of 16 frames. For the real-time case, we addressed the DAS challenge by allowing the agent to act every other frame, essentially mimicking the 'drumming' technique used by pro players to mitigate the DAS effect.

REINFORCEMENT LEARNING

Reinforcement Learning (RL) is a type of machine learning paradigm where an agent learns to make decisions by interacting with an environment. The agent takes actions based on its current state and receives rewards or penalties, facilitating the learning of optimal behavior.



The Reinforcement Learning cycle

METHODS - Real-Time Gameplay

Deep Q-Networks (DQN)

DQN is an extension of Q-Learning, a model-free reinforcement learning algorithm. The Q-function quantifies the quality of taking a specific action from a given state, based on the expected cumulative reward from taking that action from the state. DQN employs neural networks to approximate this Q-function, allowing the agent to generalize across a high-dimensional and complex state spaces. DQN's have been shown to be successful in other retro games like Atari.

Reward Shaping

Another method used is Reward Shaping, where we refine the reward structure to guide the agent more effectively. By providing additional, immediate rewards or penalties, the agent gains quicker feedback on its actions, accelerating the learning process and aiding in quicker discovery of the most effective strategies.

We experimented with various reward functions. Some of them include the in-game score, number of lines cleared, number of holes, bumpiness of the board, and many more.

Hyperparameter Search

This method is used to find the best hyper-parameters which govern a model's learning process. These are not learned from the data but are set prior to the learning process and remain constant during it.

A Search was conducted with an aim to find a good balance between the reward functions as well as optimizing the learning process. Key hyper-parameters included the weights assigned to the various reward functions. Other hyperparameters included network architecture, learning rate, etc.

Curriculum Learning

Curriculum In order to learn a technique, an agent must first master simple tasks before moving on to more challenging ones. This strategy mimics how people typically learn, moving from easier to harder tasks. In our project, we used this technique by exposing the agent initially to just one type of tetrimino, which is easier to play, and then gradually exposing him to more kinds. As a result, the agent can master one tetrimino type before moving on to another. It also makes it easier for the agent to clear lines early on, allowing him to receive positive feedback and understand the game's purpose more quickly.

Imitation Learning

Imitation Learning involves training an agent by observing human or expert behavior. Unlike standard reinforcement learning that explores an environment to find an optimal strategy, Imitation Learning uses real-world demonstrations to learn the underlying policy.

We employed a state-of-the-art method, Inverse Q-Learning, which is sample-efficient and can adapt to sparse data.

Initially, we used human demonstrations as the learning basis for our agent. Since humans don't have perfect response time, most of the data were noop actions. To mitigate this, we filtered the data to include only active actions followed by a movement. However, inconsistencies and hesitance due to human behavior remained in the data.

Later, we leveraged our successful agent in the Per-Tetrimino gameplay to gather expert data, upon which we reran our experiments.

RESULTS – Real-Time Gameplay

Although our extensive research and experimentation with many different methods, none of the agents trained were successful in clearing consistently even a single line.

In the imitation method however we did find the agent to pack the board more densely and clear a line more often.

We suspect the primary challenge lies in the agent's ability to plan far ahead, a skill essential for success in Tetris.

METHODS – Per-Tetrimino Gameplay

For the Per-Tetrimino gameplay we reimplemented a PSO based algorithm from Java in Python.

The algorithm is as follows:

1. When a new tetrimino spawns, a BFS search is conducted over the entire state space to find all possible lock-in locations for the tetrimino. The actions leading to each state are also tracked. The algorithm also accounts for gravity as well as specific Tetris NES behaviors, such as DAS, by inserting 'no-operation' (noop) actions between consecutive shift actions.
2. Each possible final state is evaluated using a value function composed of various weighted sub-functions.
3. The set of actions leading to the state with the highest value function score is executed

To find the optimal weights for the value function, Particle Swarm Optimization (PSO) is employed.

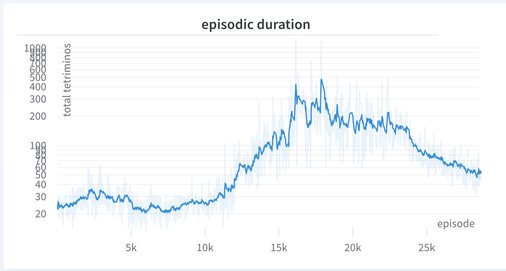
Extension to Reinforcement Learning

Extending the PSO-based algorithm, we utilized its optimized value function as the reward function for training a specialized neural network. This network, unlike Q-Networks, solely takes the board state as input. After a tetrimino spawns and a BFS search is conducted, the potential lock-in states are fed into the network. The network then evaluates these states instead of the value function.

This serves as a proof of concept that RL can effectively solve the per-tetrimino problem. However, future work can enhance the pretrained model using a more score-oriented reward function.

Results – Per-Tetrimino Gameplay

Both methods were highly successful, essentially completing the game by reaching level 30.



Training graph showing the amount of tetriminos taken to complete an episode; Y-axis in log scale

Contact

Itai Bear – itai.bear1@gmail.com

Noam Manaker Morag – noam.school.only@gmail.com