AI GAMER

Teaching AI to Play Tetris

By: Ian Taylor
Jake Backues
David Moyes
Jacob Van Skyhawk

Problem Statement

We want to explore AI's ability to learn and play video games, focusing on the classic puzzle game, Tetris.

Executive Summary

- **OBJECTIVE:**Develop models capable of playing Tetris with a degree of proficiency and understand how AI gains knowledge and improve performance without an explicit dataset
- **APPROACH:** We used reinforcement learning models and learning techniques to train our models to play Tetris
- **FINDINGS:** Training the model to play with proficiency proved to be challenging. Several models were trained, but none were able to play with a significant degree of skill
- **NEXT STEPS:** Further work and research is needed into the requirements for effectively training these models i.e. computing power, time, parameter tuning

Tetris

Tetris is a puzzle game, created by Alexey Pajitnov, where the player attempts to manipulate falling blocks called “tetrominoes” to complete horizontal lines with no gaps. As the player clears lines, they score points and level-up, which makes the pieces drop faster and the game harder.

We are playing classic Tetris, which, unlike modern Tetris, does not allow piece holding, hard drops, T-spins, and other advanced moves, putting the emphasis on basic strategy and positioning.

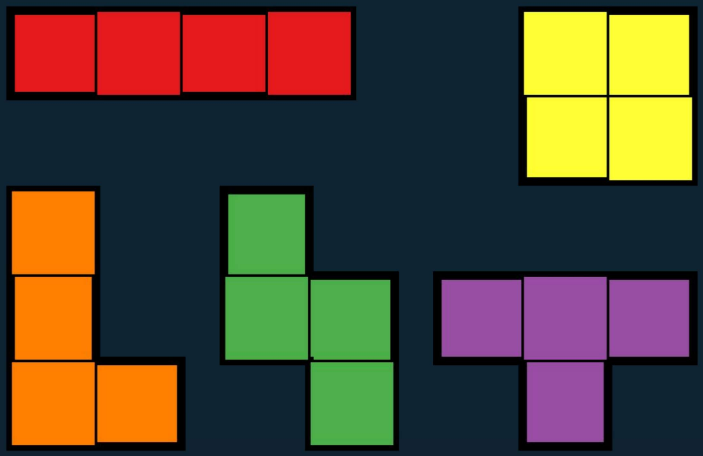


Image from wikipedia.org/wiki/Tetris

Reinforcement Learning

A type of machine learning (ML) that trains itself through positive feedback (rewards) and negative feedback (punishments) based on the Markov Decision Process. The system tries to maximize rewards and minimize punishments while solving problems.

- Training dataset built by the AI model
- Discrete environment and actions
- Transformations can still be done
- Learning Policy (π)
- Observation
- Feedback (rewards and punishments)

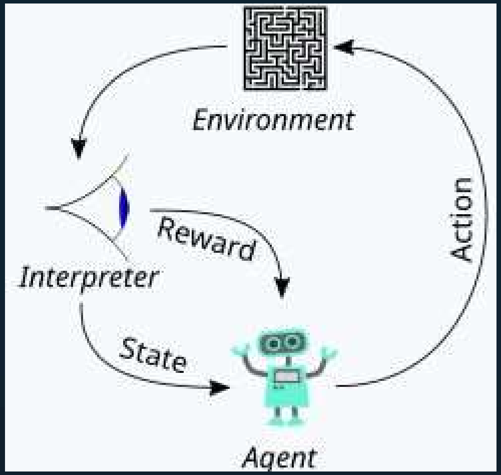


Image from wikipedia.org/wiki/Reinforcement_learning

Emulator

A pseudo game console. Allows us to run games, like Tetris.

We used programmatic emulator, Pyboy & Arcade Learning Environment (ALE, to help us see and interact with the Tetris game environment.

Environment

A replacement for the interface of a specific games, like Tetris.

We used the Atari Tetris game wrapper from Gymnasium and their implementation of Tetris.

Models and Policies

D

Deep Q-Network

This model views the state as the input & then outputs Q-values for all possible actions.

P

Proximal Policy Optimization

A policy gradient method that balances the benefits of on and off-policy methods

M

Multilayer perceptron

A policy that uses multiple layers of neurons to process inputs and compute actions

C

Convolutional Neural Network

A policy that uses convolutional layers to automatically detect and learn patterns in spatial data

Challenges & Results

TIME & COMPUTE

- Even small training models take no less than 30 min to fully process
- High RAM and disk space are needed to train each model (locally or Colab).

HYPERPARAMETERS

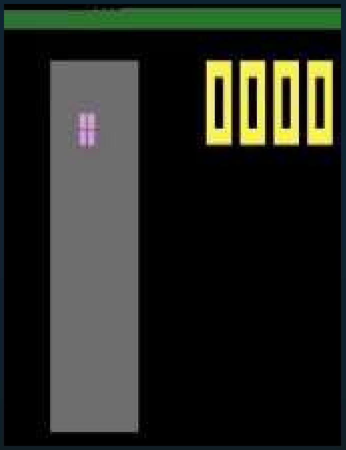
- Adjusting hyperparameters is a lot of trial and error
- Hard to evaluate hyperparameters until the model finishes training

REWARD STRUCTURE

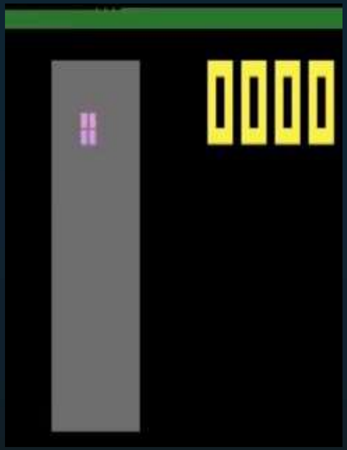
- Difficult to standardize rewards and penalties
- Custom rewards can behave unexpectedly and result in suboptimal behavior

Challenges & Results

Base Model



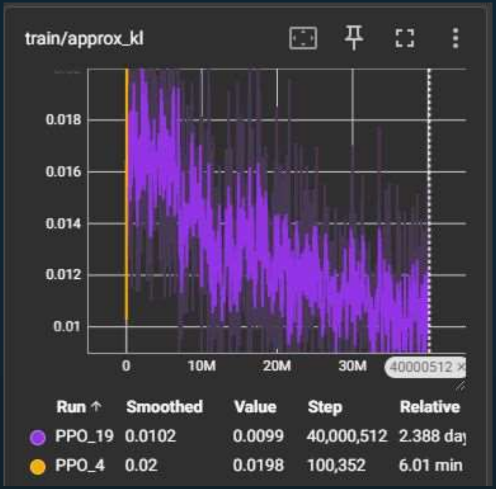
Spinning



High Stack

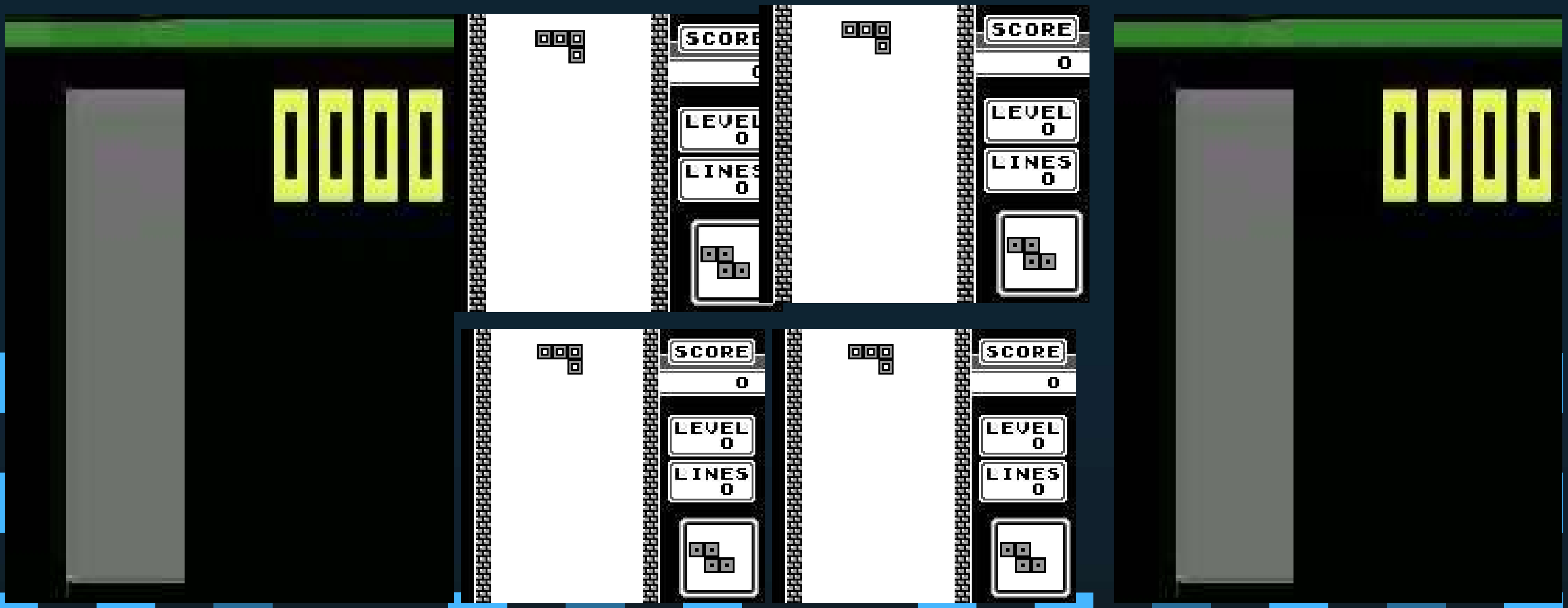
Interpreting Results

- Watching the game play
- Score, Level, Lines Cleared, Mean Rewards
- Logs - i.e. TensorBoard
  - **Kullback_Leibler(KL) Divergence** - indicator of how random the model is behaving. Should decrease over time
  - **Clip-fraction** - how many results are being dropped (clipped) to not influence the model to heavily - related to gradient-loss
  - **Entropy-loss** - How much the model is exploring
  - **Loss** - How well the models actions have the predicted outcome



Run	Smoothed	Value	Step	Relative
PPO_19	0.0102	0.0099	40,000,512	2.388 day
PPO_4	0.02	0.0198	100,352	6.01 min

Showcase



How to Improve

Hyperparameter Tuning

It’s important to have an understanding of all the hyperparameters. Misuse of them can cause suboptimal model behavior.

Adaptive Exploration

Gradual decay of the exploration rate to allow the model to continue learning until it reaches a certain mean reward

Gradual Training

Start out with simple pieces and gradually add more and speed up the pieces falling

Rewards and Penalties

Implement correct coefficients for rewards and penalties

Conclusion

Randomness

Randomly filling in a line of tetris is not as easy as you think it is

Reward Structures

Defining and tuning the rewards is the name of the game

Compute

Training large reinforcement learning models takes a lot of time and computational power

Time of training

Training for a longer amount of time doesn’t necessarily the model will be better or continue learning

THANKS

Questions?