NTNU

Kunnskap for en bedre verden

Department of Computer Science

IDATT2502
Applied Machine Learning

# Feasibility of Tetris w/ DQN

*Author:*
Erling Sung Sletta

November, 2021

# Contents

# 1 Introduction

This report is based on a project conducted as the final examination and curriculum in the subject IDATT2502 - Applied Machine Learning with Project. Of several proposed project ideas, the one chosen for this project based itself on using Reinforcement Learning to solve a given environment. More specifically, this report will be going over the process, results and conclusion of my attempts at implementing an algorithm to train an agent that can play Tetris using Deep Q-Learning. Additionally, we will look at an alternative method using a genetic algorithm for comparison sake.

## 1.1 Personal Motivation

For our eighth obligatory exercise we used Reinforcement Learning to solve the so-called CartPole problem, in which it the goal is to prevent a pendulum attached to a cart from falling over. The concept of teaching a machine to learn and adapt to it's environment to figure out what it should do left an impression on me, so when I was then presented with the idea of attempting to solve more complex environments, like Super Mario or Tetris, I was quickly tempted. While the other topics were all quite interesting as well, my video-gaming background simply wouldn't let me pass up this opportunity.

## 1.2 Tetris (?)

[Game description]

# 2 Related Work

## 2.1 OpenAI

OpenAI is an AI research laboratory whose ”mission is to ensure that artificial general intelligence benefits all of humanity”[4]. One of OpenAI’s goals is to standardize environments that are used in AI research, and one of their contributions towards this goal is the open source Python library known as Gym. This Reinforcement Learning toolkit aims to supply users with environments that are simple to create and interact with[1]. The following bit of code shows an example of how the previously mentioned Gym CartPole environment can be setup and used.

```
import gym
env = gym.make('CartPole-v1')

# env is created, now we can use it:
for episode in range(10):
    obs = env.reset()
    for step in range(50):
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
```

Some particularly noteworthy variables here are ’observation’, ’action’ and ’reward’. The ***observation*** is often called state, as it represents the current state of the environment. In the CartPole environment these are values representing the position and velocity of the cart and pendulum. An ***action*** is one of some amount of possible actions that the agent, in other words the player, can take. In this environment it is whether the cart should move left or right. Here the action is decided randomly, but in actual implementation, policies are defined to select an action based on some algorithm. After an action has been made, the agent gets a ***reward*** based on the result of said action.

## 2.2 Reinforcement Learning

Using Reinforcement Learning to play video games is a much researched topic. Even the specific, albeit simple, Google search ”Tetris DQN” will net you a multitude of theses, GitHub repositories and so on. As part of a publication series explaining deep reinforcement learning[6], Jordi Torres describes the process of creating an algorithm to train an agent to play the classic Atari arcade game Pong[5]. Torres uses an OpenAI Pong environment that emulates an Atari 2600 environment, offering a resolution of 210x160 pixels with 128 colors. An agent does not get access to the games underlying state, but has to work with the rather complex raw pixel data. To circumvent this, Torres applies a series of processing-steps to simplify this data into smaller more practical states for us to work with, a method that greatly helped the progress of this project.

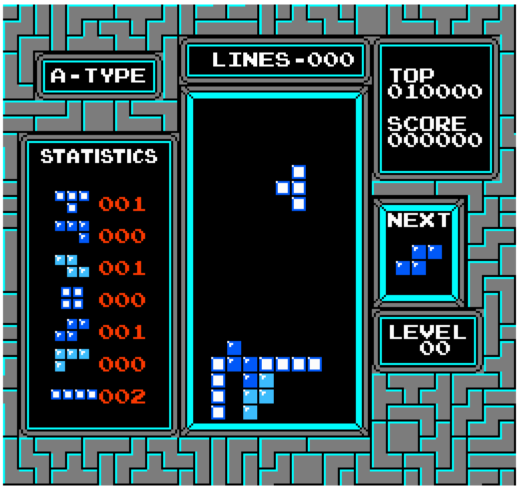
Torres also mentions a 2015 project led by DeepMind, in which a group of researchers attempt to ”create a single algorithm that would be able to develop a wide range of competencies on a varied range of challenging tasks”[3]. The group successfully developed an architecture that performs above human level in most of the tested environments, and even several times better than a professional in a large portion of them.

# 3 Method

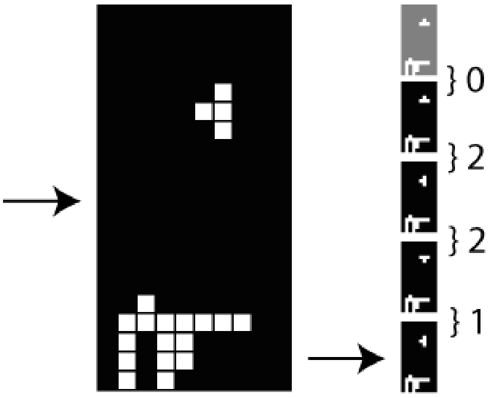
## 3.1 Environment

### 3.1.1 Specifics

The environment used for this project is an OpenAI Gym interface that emulates classic Tetris on the NES, developed by Christian Kauten[2]. This particular environment is similar to the previously referenced Pong environment, in the sense that it does not let our agent directly view the games underlying state, with a few exceptions. The state returned by the environment has the shape 240 x 256 x 3, representing a 240x256 pixel image with RGB color. This is a rather complex structure that would be very taxing to process, which is why a method to reduce the resolution drastically is used. The method is heavily inspired by Torres' Pong solution[5], and reduces the state to some number of 10x20 images, each representing the Tetris board at sequential time steps.



(a) The original state



(b) Processing steps
(image upscaled)

### 3.1.2 Processing

In figure a we see the visual representation of the state that the environment initially returns. This initial state would be very inefficient to work with, as it is not only very complex; but also has a lot of redundant noise. For example, the agent does not need to see the tetrominoes scattered around in the background, or the various texts around the board. Not only do these details increase the state complexity, but they could also interfere with our learning algorithm.

For these reasons, the image is processed to simplify the structure and crop out the noise, as we see in the first step of figure b. The image is then resized into a 10x20 image, represented by an array of ones and zeroes. Since the agent would have no dynamic piece knowledge with just a single layer, we keep each processed state in a buffer so we can pass this on to the agent. In the figure we see the newest image being appended to a buffer with a capacity of four states, as well as the actions that were taken between each state. Below is a table showing which action each number represents.

| Action Space | |
|---|---|
| # | Description |
| 0 | Do nothing |
| 1 | Rotate piece clockwise |
| 2 | Rotate piece counter-clockwise |
| 3 | Move piece right |
| 4 | Move piece left |
| 5 | Move piece down |

## 3.2 Deep Q-Learning

### 3.2.1 Q-Learning

Deep Q-learning (henceforth DQN) is a variation of a reinforcement learning algorithm known as Q-learning; an algorithm that doesn't use a model of the environment, but instead creates a table, often called a Q-table, to keep track of each state + action combination, and a value that decides how good the given action is for that state. The agent then uses this table as the basis for an algorithm to be used in the policy.

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg( \underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \bigg)}^{\text{temporal difference}}$$

Figure 2: Q-learning algorithm[7]

The algorithm above is used to calculate how good an action $a_t$ is given when in state $s_t$. The learning rate $\alpha$ decides how much should be learnt at each step, that is how quickly the Q-value should change. The discount rate $\gamma$ decides how 'greedy' the agent is, meaning how much it values immediate gain over long-term gain.

### 3.2.2 DQN In-Depth

DQN replaces the Q-table from Q-learning with a convolutional neural network. This network takes the state and "analyses" it to find and learn patterns in which actions lead to what rewards.

Experience replay

## 3.3 Genetic Algorithm

Arguably not machine learning, but interesting to look at nonetheless

### 3.3.1 Heuristics

Uses general but concrete definitions for what a "good" game state looks like.

### 3.3.2 Evolution

Used to optimize heuristic weights.

# 4 Result

## 4.1 DQN Models

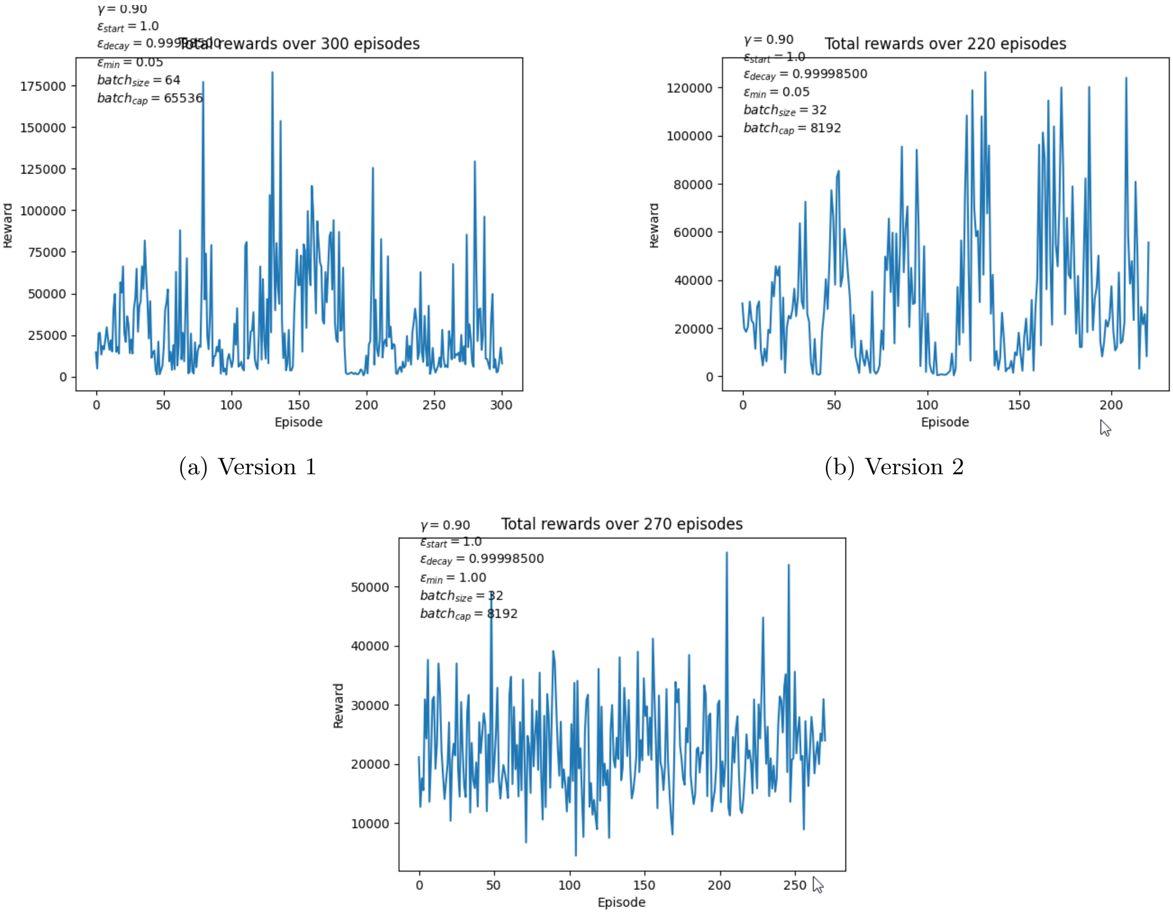
Different iterations, will go over some

### 4.1.1 Informational

No state... current piece, next piece etc.
Reward graph
Network illustration?

### 4.1.2 State

Figured out state, processing
Videos
Reward graph, stay alive
Network illustration?



(a) Version 1

(b) Version 2

(c) Random

Figure 3: DQN w/State rewards

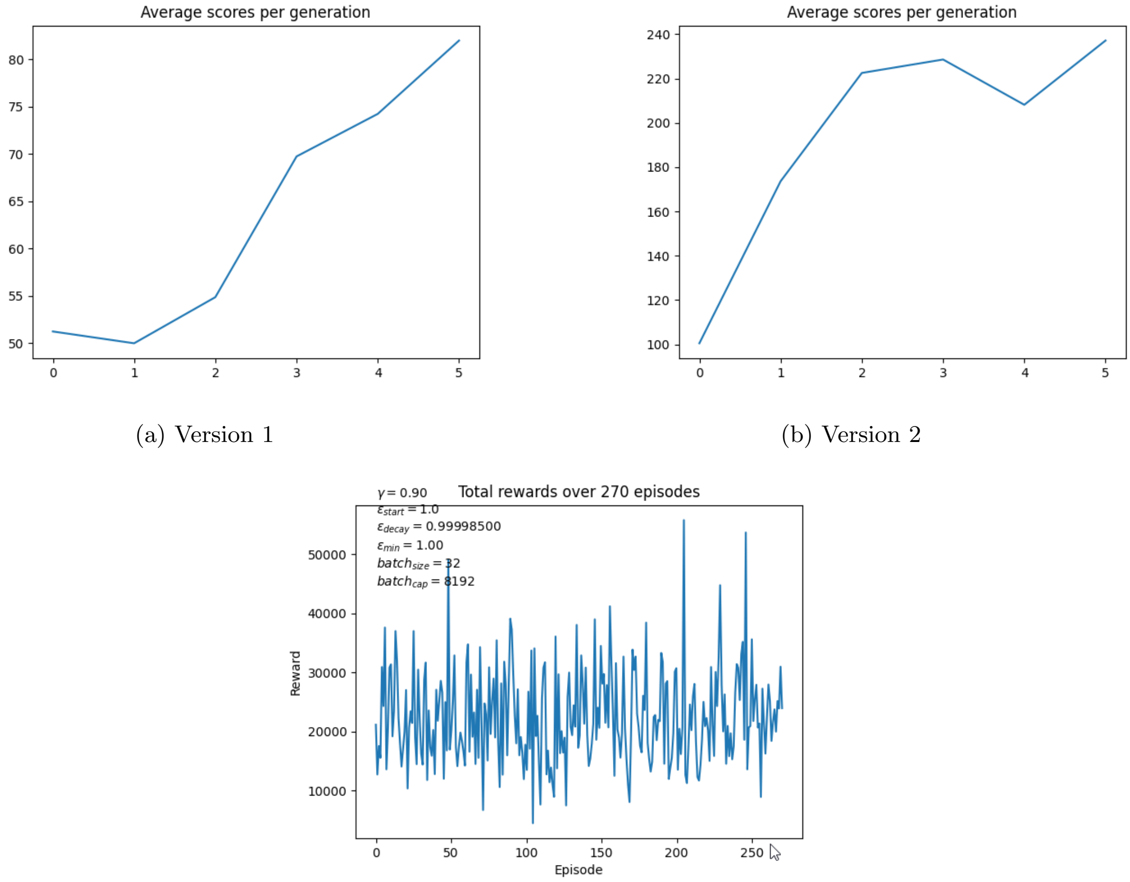
## 4.2 Genetic Algorithm

### 4.2.1 Random Heuristics

Random heuristics
Reward graph
Individual performance

### 4.2.2 Evolutionary Heuristics

Evolution to optimize weights
Generational reward graphs
Video



(a) Version 1

(b) Version 2

(c) Random

Figure 4: Genetic algorithm rewards

Version 1 had 5 generations with a population of 8, each playing 5 games up to 50 moves.

Version 2 had 5 generations with a population of 12, each playing 5 games up to 50 moves.

## 4.3 Comparison

Comparison graphs, DQN vs Genetic

## 5 Discussion

Tetris very complex
DQN difficult cause nothing concrete to work with, only a line clear every blue moon, too many states.
First move is 7 * 4 * 9 possible permutations.

# 6 Conclusion

As shown in the previous sections, performance is much better when the machine is supplied with concrete general definitions for what a good move looks like. Theoretically, it should be possible to have an agent know every single possible permutation state within Tetris, and thus every optimal move. Considering the complexity of Tetris, however, it is simply not feasible to create an algorithm that can play Tetris effectively at a high level of play with current-day technology.

# References

[1] Greg Brockman et al. *OpenAI Gym.* 2016. eprint: arXiv:1606.01540.

[2] Christian Kauten. *Tetris (NES) for OpenAI Gym.* GitHub. 2019. URL: https://github.com/Kautenja/gym-tetris.

[3] Volodymyr Mnih et al. 'Human-level control through deep reinforcement learning'. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

[4] OpenAI. *OpenAI Charter.* URL: https://openai.com/charter/ (visited on 24th Nov. 2021).

[5] Jordi Torres. *Deep Q-Network (DQN)-I.* URL: https://towardsdatascience.com/deep-q-network-dqn-i-bce08bdf2af (visited on 16th Nov. 2021).

[6] Jordi Torres. *Deep Reinforcement Learning Explained.* URL: https://towardsdatascience.com/tagged/deep-r-l-explained (visited on 16th Nov. 2021).

[7] Wikipedia contributors. *Q-learning — Wikipedia, The Free Encyclopedia.* [Online; accessed 25-November-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Q-learning&oldid=1056775822.