NANYANG TECHNOLOGICAL UNIVERSITY

REINFORCEMENT LEARNING FOR TETRIS

Tan Sheng Hock

School of Computer Science and Engineering

2019

NANYANG TECHNOLOGICAL UNIVERSITY

SCSE18-0161

REINFORCEMENT LEARNING FOR TETRIS

25 March 2018

Submitted in Partial Fulfilment of Requirements

for the Degree of Bachelor in Computer Science

of Nanyang Technological University by

Tan Sheng Hock

U1621606C

Supervised by Assoc Prof Xavier Bresson

Examined by Asst Prof Long Cheng

School of Computer Science and Engineering

2019

Abstract

This project presents the development and implementation of an agent that can play Tetris, which learns how to maximize along a specific dimension over many steps using reinforcement learning; in this case, maximize the points won in a game of Tetris over many moves. Like how humans learn to achieve a better score, through multiple interactions with the Tetris game environment, reinforcement learning is just a computational approach to learning from action. This project includes the development of the Tetris game environment, the development of the self-playing Tetris agent and the implementation of reinforcement learning. The model was trained using a reinforcement learning algorithm in the Tetris game environment. The model comprised of a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. The source code of the project can be found on https://github.com/JCodeSH/jcodesh.github.io.

Acknowledgment

The student would like to express his deep gratitude to Prof. Xavier Bresson for his patient guidance and support throughout the course of the project. With his expertise and useful advice, the student is able to overcome obstacles and steer the project in the correct direction

The student would also like to extend his thanks to his family, friends and loved one for their support and encouragement throughout the entire project.

Table of Contents

List of Notations

Notation	Description
$\boldsymbol{S}$	Set of possible states
$\boldsymbol{A}$	Set of available actions
$\boldsymbol{R}$	Reward function
$\boldsymbol{Q(s,a)}$	Q-Value function of Q-Learning
$\boldsymbol{s_t}$	State at time step t
$\boldsymbol{a_t}$	Action at time step t
$\boldsymbol{s_{t+1}}$	State at time step $t+1$
$\boldsymbol{a_{t+1}}$	Action at time step $t+1$

List of Figures

List of Tables

1. Introduction

1.1. Background

Machine Learning is an application of Artificial Intelligence (AI) that provides computers with the ability to learn from examples and experience without being explicitly programmed [1]. Motivated by how human beings learn from examples, experience, machine learning focuses on the development of computer programs that can teach themselves to grow from data and change when exposed to new data. In short, it is to allow machines to learn by itself without human intervention and adjust their actions accordingly.

Games are more than just only entertainment, it is commonly used as a testbed for AI, where agents are trained to outperform human players by optimizing its score. The standard Tetris is a stochastic, open-ended board game is an example of such games that is a suitable environment for such development. First developed by Alexey Pajitnov in 1984, had become popular ever since. The game being a very well-known game is now widely available on most gaming consoles and computer systems [2].

The game field is made up of 10 by 20 cells. Players control the current falling pieces randomly chosen from 7 pre-defined shapes of blocks and try to build fully occupied rows, which are removed in each turn. In addition to the current piece, the shape of the next piece is also made known to the player. The objective of the game is to place the pieces in the optimal position in the game field to best accommodate subsequent pieces and clear as many rows as possible. Wrong placement of pieces may often result in an undesirable situation, which may lead to spending additional time to manage the game field. The game ends when the top of the game field is filled, and no pieces can be placed onto the field.

One of the fields under Machine Learning is Reinforcement Learning where an agent learns by interacting with its environment, observing the results of these interactions and receiving a reward either positive or negative accordingly [3]. Reinforcement

Learning faced many challenges, and one biggest challenge is faced is the absence of labeled data [4]. Additionally, Reinforcement Learning agents learn through exploration and exploitation. Tetris involves a fair degree of strategy and the environment changes very frequently, it becomes very expensive for the agent to learn in continuous trial-and-error based learning.

1.2. Purpose and Scope

Tetris is essentially a task in visual pattern detection and objects recognition. With a recent breakthrough in deep reinforcement learning, it has shown that convolutional neural network can be trained to learn strategy and to approach this problem.

A Tetris game will be implemented to provide an environment for the agent to interact and simulate the gameplay. Reinforcement Learning technique is used to approximate the action-selection policy for any sequence of the environment states. The convolutional neural network model is trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards.

The aim of this project is to explore and implement the self-playing Tetris agent to maximize the points won in a game of Tetris over many moves. Using reinforcement learning-based algorithm, the agent is trained and will learn the optimal policy to perform the optimal set of actions to eliminate as many rows as possible in order to perform as many moves as possible to achieve a high score.

2. Literature Review

2.1. Reinforcement Learning

Reinforcement Learning is one of many areas in Machine Learning. An agent learns how to behave in an environment by taking suitable action to maximize rewards in a situation [5].

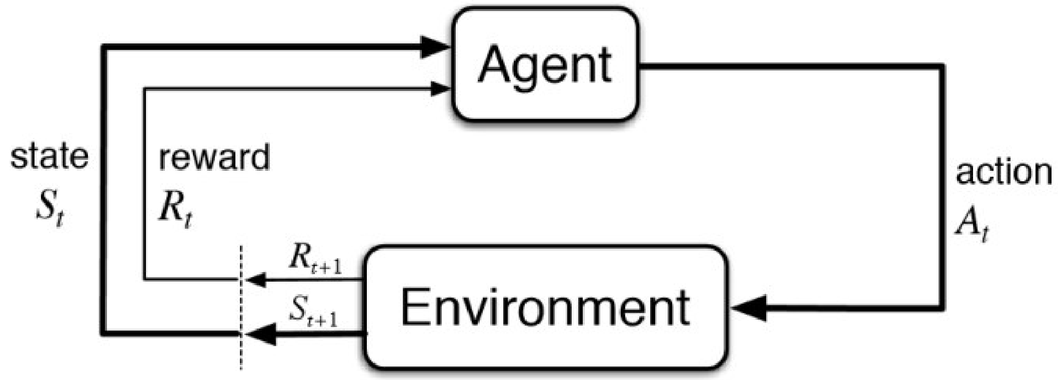


Figure 1 – Illustration of Reinforcement Learning [6]

Using Tetris as the environment which the agent is trying to learn, for each iteration, the state of the environment denoted by S_t at time t which is the frame of the Tetris grid. The agent can perform certain actions – Do Nothing, Rotate, Left, Right, Down, or Drop, denoted by A_t. The actions result in reward, R_t which can either be positive or negative. The action affects the environment and leads to new state denoted by S_{t+1} and new reward denoted R_{t+1}.

The reinforcement learning process outputs a sequence of state, action, and rewards, and repeats until termination. The goal is to learn an optimal policy mapping states to actions by maximizing the expected cumulative reward.

2.2. Q-Learning

Q-Learning is a technique of Reinforcement Learning where a table of Q values is maintained with each state, action and the reward. With referenced to figure 2 below, it shows how the data is structured, where the state refers to the frame of the Tetris grid and the actions refer to "Do Nothing", "Rotate", "Left", "Right", "Down", and "Drop".

$$\begin{array}{cc} & \textbf{Action} \\ \textbf{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \end{array}$$

$$Q = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix}$$

Figure 2 – Sample Q-Table

Q-Learning is used to find an optimal action for any given state in a finite Markov Decision Process (MDP) where it tries to maximize the value of the Q-function which represents the maximum discounted future reward when an action is performed in a state.

$$Q(S_t, a_t) = \max(R_{t+1})$$

After the Q-function is known, the action at a state with the highest Q-value is the optimal action. To solve this problem, an action with the highest predicted Q-value is chosen.

2.3. Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is a Deep Learning algorithm primarily used to do image recognition and classification [7]. It takes in an input image and makes predictions for the images it has never seen before. It sees the input images as an array of pixels and depending on the image resolution, it will see Height x Width x Dimension.

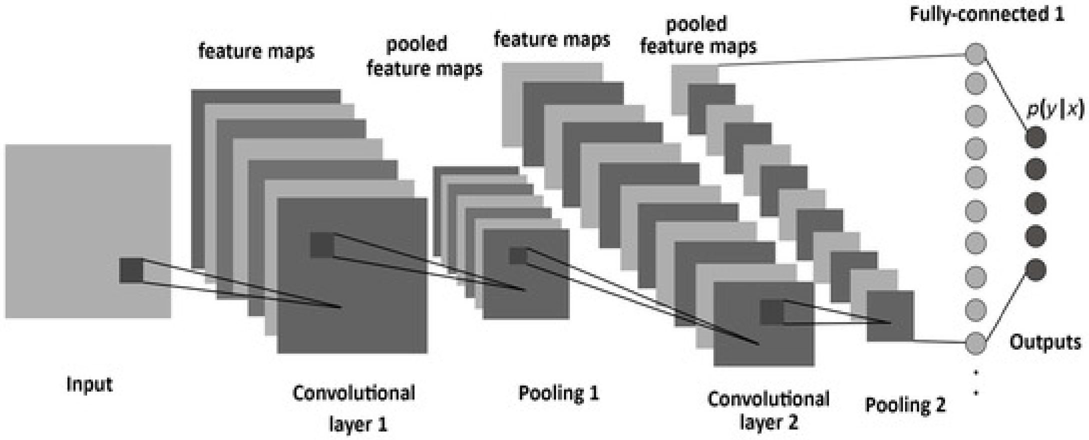


Figure 3 – Illustration of Convolutional Neural Network

Inspired by the brain [8], how CNN works is similar to how mammals perceive the world visually. With reference to Figure 3, the layers are organized in 3 dimensions in the architecture of CNN and the neurons only connect to a small region of the neurons in the next layer. The input will be reduced to the final output which is a single vector of probability scores.

3. Project Timeline

The project was worked on from August 2018 to March 2019. Work progress, tasks, and findings were documented and also presented during meetings and discussions with the supervising professor.

3.1. Estimated Timeline

Table 1 – Estimated Project Timeline

Task	Duration
Understand the theory behind Deep Reinforcement Learning and Convolutional Neural Network for Visual Recognition	From Aug 2018 To Sep 2018
Design and implement a Tetris game environment to simulate and visualize the actual gameplay	From Sep 2018 To Nov 2018
Learn how to implement Convolutional Neural Network for Visual Recognition to allow the agent to recognize the environment	From Nov 2018 To Nov 2018
Design and implement an agent that can interact with the Tetris game environment	From Dec 2018 To Jan 2019
Reinforcement Learning techniques will be used to train the agent to learn how to behave in the environment	From Jan 2019 To Feb 2019
The agent is required to learn the optimal policy and select the optimal action to perform which maximizes the score of the Tetris game	From Jan 2019 To Feb 2019
Final Year Project Report Submission	Mar 2019
Amended Final Year Project Report Submission	Apr 2019
Final Year Project Oral Presentation	May 2019

3.2. Actual Timeline

Table 2 - Actual Project Timeline

Task	**Duration**
Understand the theory behind Deep Reinforcement Learning and Convolutional Neural Network for Visual Recognition	From Aug 2018 To Sep 2018
Design and implement a Tetris game environment to simulate and visualize the actual gameplay	From Sep 2018 To Nov 2018
Learn how to implement Convolutional Neural Network for Visual Recognition to allow the agent to recognize the environment	From Nov 2018 To Dec 2018
Design and implement an agent that can interact with the Tetris game environment	From Jan 2019 To Feb 2019
Reinforcement Learning techniques will be used to train the agent to learn how to behave in the environment	From Feb 2019 To Feb 2019
The agent is required to learn the optimal policy and select the optimal action to perform which maximizes the score of the Tetris game	From Feb 2019 To Mar 2019
Final Year Project Report Submission	Mar 2019
Amended Final Year Project Report Submission	Apr 2019
Final Year Project Oral Presentation	May 2019

4. Algorithm

This project used reinforcement learning as its learning approach. It interacts with the game environment and uses the images of the game screen as well as the information of the next active block as the input. The possible available actions are Do Nothing (**Action 0**), Rotate Clockwise (**Action 1**), Left (**Action 2**), Right (**Action 3**), Down (**Action 4**) and Drop (**Action 5**).

Training in Reinforcement Learning is very unstable without any labeled data. The model will play the game randomly with each state, action, and reward recorded. The model will then be trained on randomly selected batches from these experiences.

The experience at time t contains state s_t, action a_t, the subsequent block n_t, the reward which refers to the difference between the score at time t and $t-1$, r_t, the state at time $t+1$, s_{t+1}, and a terminal which shows if the game crashed or not. The experience is appended to replay memory D, and when the memory is full, the oldest experience is removed.

For the agent to experience more possible states, epsilon-greedy is implemented to maximize the exploration which could give better rewards. When the probability is lesser than the current epsilon-greedy, an action is randomly chosen. Otherwise, by using Q-Learning and approximate a special function which drives the action-selection policy for any sequence of environment states in order to choose an action with the highest predicted Q-value. The epsilon-greedy exploration parameter is gradually decayed with each time step to reduce the randomness as it progresses.

5. Input Processing

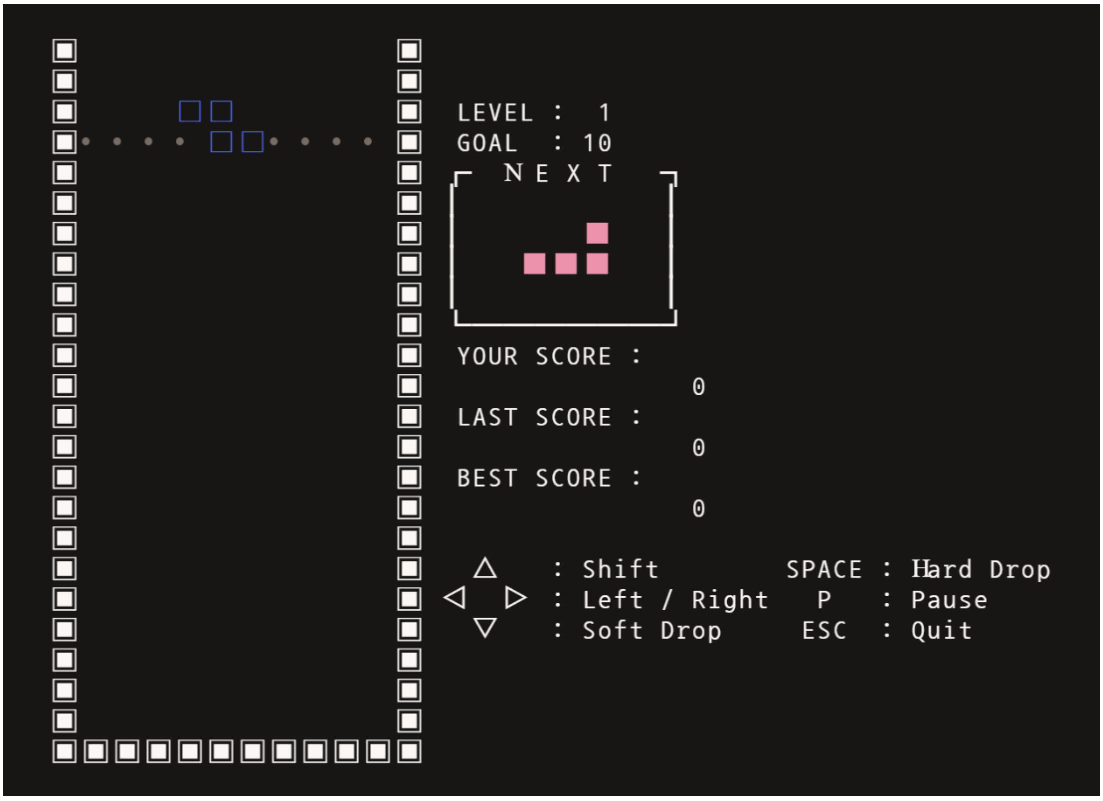


Figure 4 – Raw Image Captured

As shown in figure 4, the raw image captured from the game has a resolution of around 1260 x 910 pixels with 3 (RGB) channels. The model will receive 4 consecutive screenshots as input, which makes it computationally expensive and not all portion of the image is useful for playing the game. Hence the image is reduced to a dimension of 100 x 50 pixels and single (grey scale) channeled as in Figure 5.

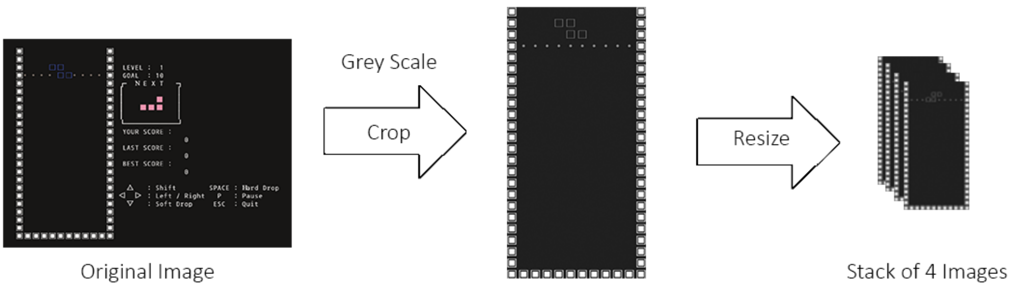


Figure 5 – Image Processing Flow

6. Model Architecture

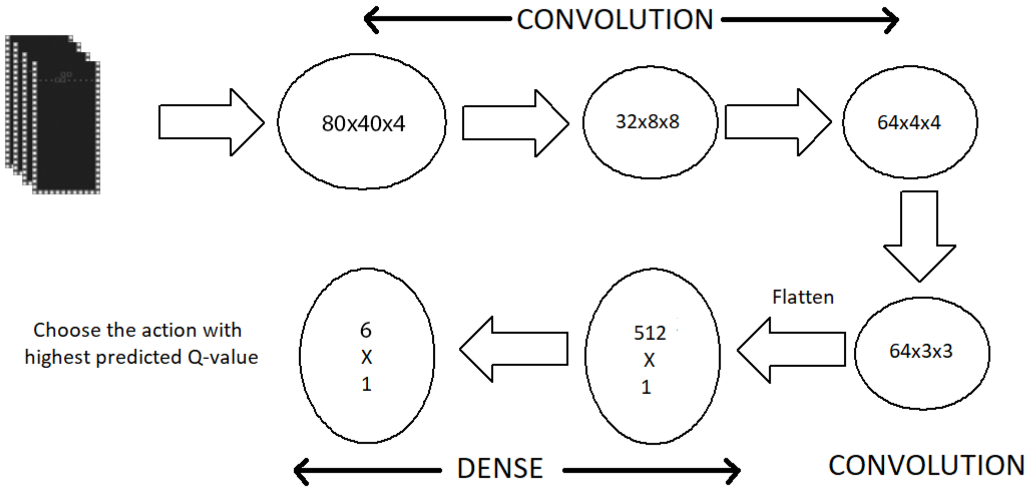


Figure 6 - Convolutional Neural Network Architecture

A series of three Convolution layers is used before it is flattened into dense layers and an output layer. Max Pooling layers are used as they can significantly improve the processing of a dense feature set. The output layer consists of six neurons, where each of them represents the maximum predicted reward for each action. The action with the maximum reward or Q-Value is chosen as the next action to take.

7. Experiments

The Tetris game is implemented in JavaScript on ChromeDriver and the model is written in python. By using a browser automation tool Selenium, the model is able to send actions to the browser and also get different information such as the current score and the next block from the game. This interface also allows the model to capture the game screen without affected by the screen resolution and window location by getting a base64 formatted image from the HTML Canvas using JavaScript. Similar to the actual Tetris gameplay in figure 7, the game environment allows the agents to interact and train.

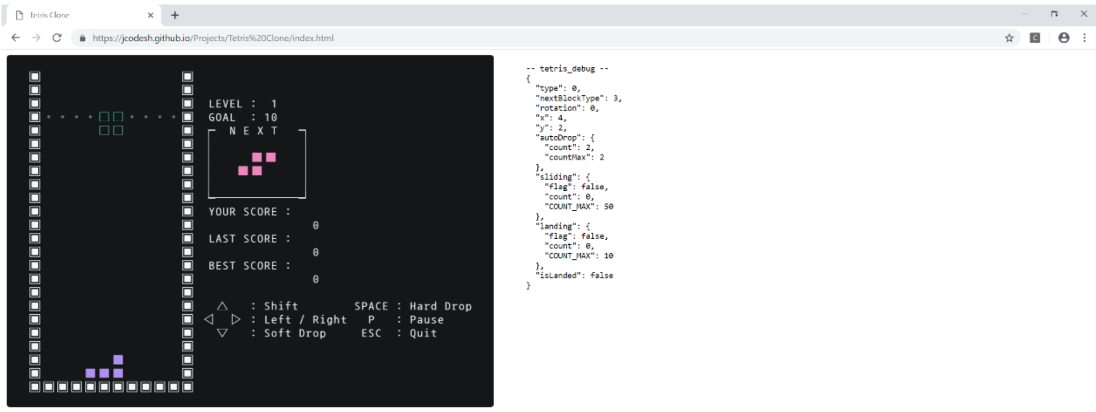


Figure 7 – Screenshot of Tetris Environment in ChromeDriver

Table 3 – Available Action and their Value

Action Name	Action Value
Do Nothing	Action "0"
Rotate Clockwise	Action "1"
Left	Action "2"
Right	Action "3"
Down	Action "4"
Drop	Action "5"

Table 4 – List of Possible Next Block and their Value

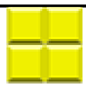
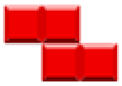
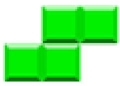
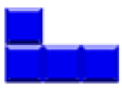
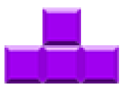
Block Name	Next Block Value
 Shape O	Next Block "0"
 Shape I	Next Block "1"
 Shape Z	Next Block "2"
 Shape S	Next Block "3"
 Shape L	Next Block "4"
 Shape J	Next Block "5"
 Shape T	Next Block "6"

Table 5 - List of Parameters and their Value

Parameter	Value	Description
Gamma	0.99	Decay rate of past observations
Observation	100	Number of observations to observe before training
Explore	100000	Number of frames to anneal epsilon
Initial Epsilon	0.1	Starting value of epsilon
Final Epsilon	0.0001	Final value of epsilon
Replay Memory	100000	Number of previous experiences to remember
Batch	16	Size of minibatch selected for each iteration
Learning Rate	1e-4	Weights of network with respect to the loss gradient

8. Results

8.1. Scores

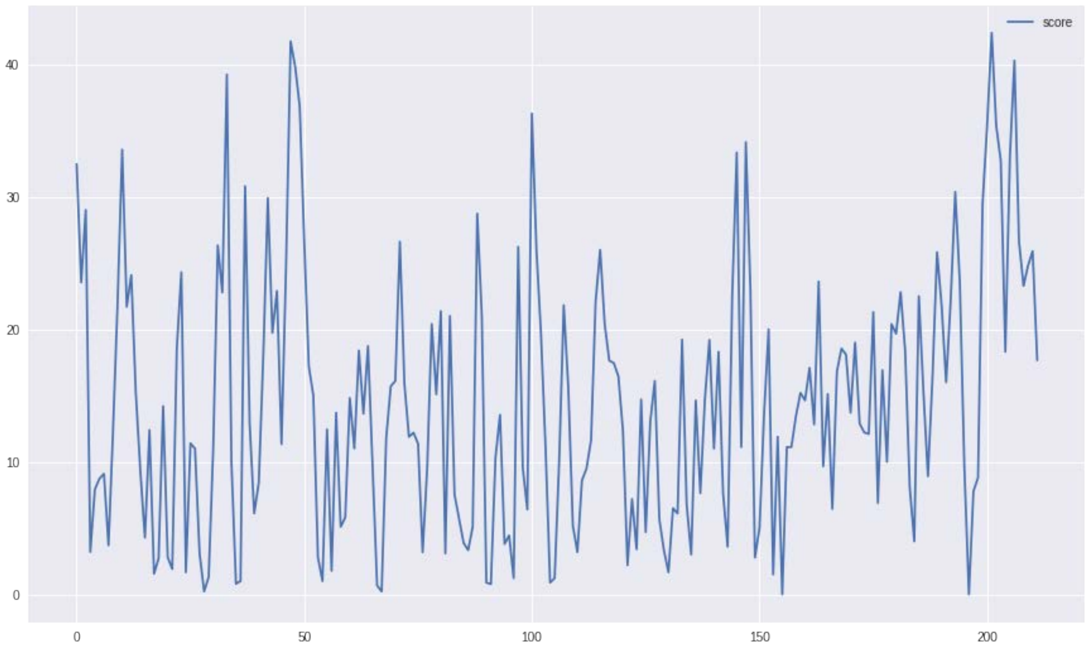


Figure 8 – Average Score every 10 games

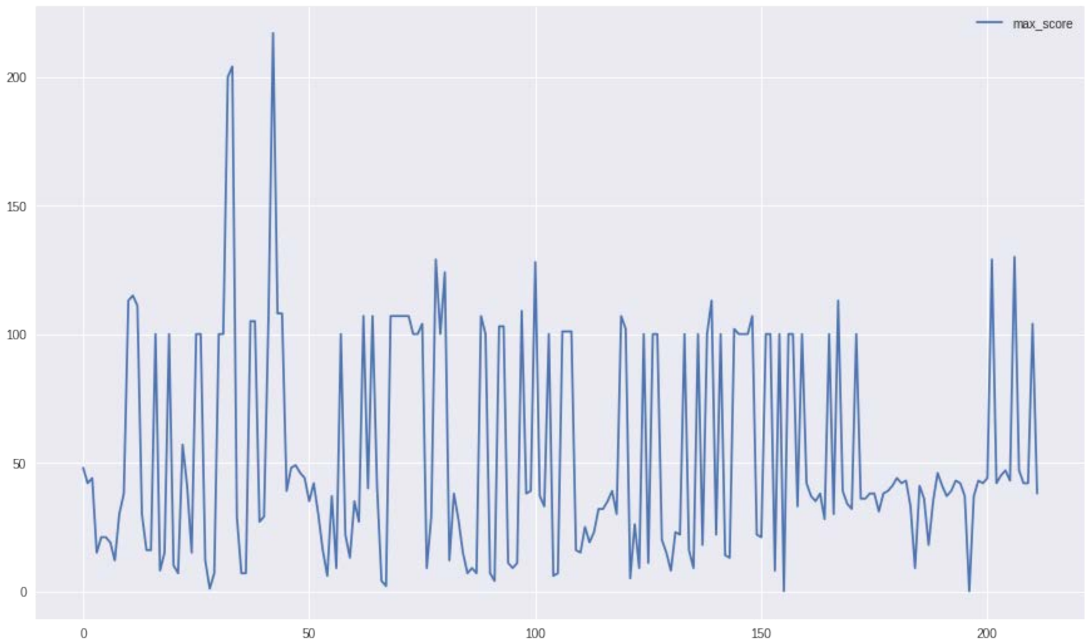


Figure 9 – Max Score every 10 games

The scores of games are the total score for each game the agent plays. According to figure 8, the scores of the model tends to fluctuate but according to figure 9, the model did manage to achieve a score of more than 100 points for some of the games. As the actual Tetris gameplay has varying speeds, with speed increasing as the level progresses, the Tetris environment is modified to allow the model to train at a constant speed.

8.2. Training Loss

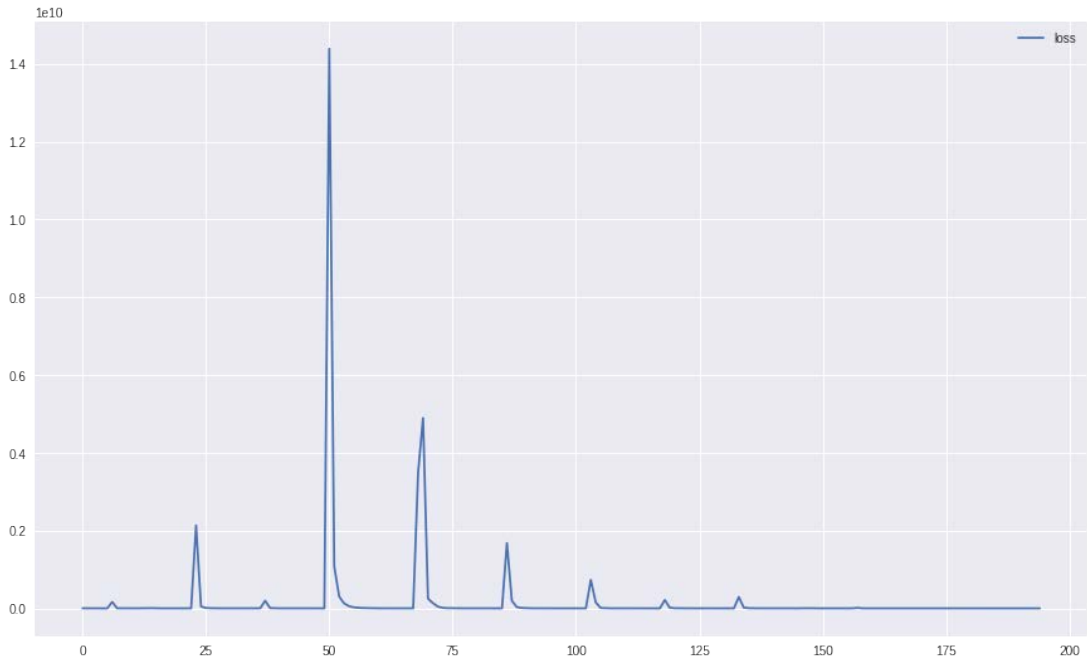


Figure 10 – Mean of Training Loss every 10 games

Training loss shows how closely a model behaves with the environment. According to figure 10, we can observe that training loss decreases as the number of training increase and has gradually stabilized and stays low with minute fluctuations.

9. Discussion

We can see from the results of the experiments, achieving convergence will be difficult given the current setup. Looking at the actions taken for particular gameplay, it can be observed that the agent takes similar action repeated, causing the pieces to either rotate clockwise or it will be moved to a particular side while the game slowly lowers it. A similar pattern can also be observed in many other gameplays resulting in not able to achieve convergence.

Additionally, some gameplay that managed to achieve a significantly high score after similar action is caused by performing the action “Hard Drop” which adds an additional bonus score.

10. Conclusion

This paper showed the implementation of deep learning model for reinforcement learning Tetris game, where it uses only raw pixels as input. The goal is to maximize the points won in a game of Tetris over many moves. Q-Learning techniques along with mini batches of experience replay to ease the training of the agent. However, this approach failed to converge as it is possible that this game is more reflex-based compared to strategy-based. Although Tetris requires a certain degree of strategy, for example, increasing the number of cleared lines for additional scores. Hence, by predicting actions without any heuristic input might cause the agent to perform unnecessary or unmeaningful actions.

11. Future Work

Below are several suggestions for future research in this field:

1. **Experimenting with human boosted data**: Comparison of time-wise progression and improvement between the agent that learn with and without human boosted data [9]. Further exploration in this aspect could determine if providing information to the agent would be able to achieve similar results in a shorter time or possibly produce an agent that can perform significantly better.

2. **Tuning of hyperparameters and architecture of the model**: Possible extensive experiments can be performed to find the optimal hyperparameters values as well as finetuned the architecture of the model.

3. **Include Heuristics and Feature Selection**: Possible implementation of a better scoring function that gives meaningful rewards depending on the action and how it affects the overall state of the game.

12. Appendix

12.1. Project Setup

Setup Virtual Machine

The project is setup in a virtual machine off Google Compute Engine running Ubuntu 16.04.6. The virtual machine is running a **complete desktop environment** so that screenshots can be capture and utilize.

Installing Dependencies

Selenium *pip install selenium*

OpenCV *pip install opencv-python*

Download Chromedrive from http://chromedriver.chromium.org

Running the project

The project source file can be retrieved from:

https://github.com/JCodeSH/jcodesh.github.io/tree/master/Projects/Tetris%20Agent

tetris.py

For the first run, uncomment *init_cache()*

Else, leave *init_cache()* commented

13. References

[1] "Expert System," [Online]. Available: https://www.expertsystem.com/machine-learning-definition/.

[2] "Wikipedia, The Free Encyclopedia," [Online]. Available: https://en.wikipedia.org/wiki/Tetris.

[3] "Medium," freeCodeCamp, [Online]. Available: https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419.

[4] A. Samal, "Reinforcement Learning : Its necessity and challenges," Medium, 2 May 2017. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-its-necessity-and-challenges-febef1470e9a.

[5] Bonsai, "Medium," [Online]. Available: https://medium.com/@BonsaiAI/deep-reinforcement-learning-models-tips-tricks-for-writing-reward-functions-a84fe525e8e0.

[6] "skymind," [Online]. Available: https://skymind.ai/wiki/deep-reinforcement-learning.

[7] "Convolutional Neural Networks with Reinforcement Learning," Packtpub, 6 April 2017. [Online]. Available: https://hub.packtpub.com/convolutional-neural-networks-reinforcement-learning/.

[8] D. Corenlisse, "An intuitive guide to Convolutional Neural Networks," Medium, 25 April 2018. [Online]. Available: https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050.

[9] "Improving reinforcement learning with human input," BOREALIS AI, 10 July 2018. [Online]. Available: https://www.borealisai.com/en/blog/early-careers-reinforcement-learning/.

[10] V. Mnih, K. Kavkcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, Playing Atari with Deep Reinforcement Learning, DeepMind Technologies.

[11] M. Stevens and S. Pradhan, Playing Tetris with Deep Reinforcement Learning, Stanford.

[12] S. Y. "Github," [Online]. Available: https://github.com/songyanho/Reinforcement-Learning-for-Self-Driving-Cars.

[13] G. Surma, "Atari - Solving Games with AI 🤖 (Part 1: Reinforcement Learning)," Medium, 2 October 2018. [Online]. Available: https://towardsdatascience.com/atari-reinforcement-learning-in-depth-part-1-ddqn-ceaa762a546f.

[14] M. Comi, "How to teach AI to play Games: Deep Reinforcement Learning," Medium, 15 November 2018. [Online]. Available: https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a.

[15] G. Surma, "Cartpole - Introduction to Reinforcement Learning (DQN - Deep Q-Learning)," Medium, 26 September 2018. [Online]. Available: https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288.

[16] S. Weidman, "The 3 Tricks That Made AlphaGo Zero Work," Medium, 8 January 2018. [Online]. Available: https://hackernoon.com/the-3-tricks-that-made-alphago-zero-work-f3d47b6686ef.

[17] A. Tankala, "Build your First AI game bot using OpenAI Gym, Keras, TensorFlow in Python," Medium, 19 October 2018. [Online]. Available: https://medium.com/coinmonks/build-your-first-ai-game-bot-using-openai-gym-keras-tensorflow-in-python-50a4d4296687.

[18] A. L. Ecoffet, "Beat Atari with Deep Reinforcement Learning! (Part 1: DQN)," Medium, 22 August 2017. [Online]. Available: https://becominghuman.ai/lets-build-an-atari-ai-part-1-dqn-df57e8ff3b26.

[19] R. Munde, "How I built an AI to play Dino Run," Medium, 1 May 2018. [Online]. Available: https://medium.com/acing-ai/how-i-build-an-ai-to-play-dino-run-e37f37bdf153.

[20] R. Munde, "Build an AI to play Dino Run," PaperspaceBlog, 25 May 2018. [Online]. Available: https://blog.paperspace.com/dino-run/.

[21] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way," Medium, 16 December 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[22] P. "Understanding of Convolutional Neural Network (CNN) — Deep Learning," Medium, 4 March 2018. [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148.

[23] H. Pokharna, "The best explanation of Convolutional Neural Networks on the Internet!," Medium, 29 July 2016. [Online]. Available: https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8.