Training a Tetris Agent Using Deep Q-Learning with Replay Memory

Daniel Chang

David Justice Puran Nepal

Yousaf Khaliq .

***Abstract*—Deep reinforcement learning has been used to develop superhuman agents for a variety of games. In this paper, we discuss our efforts to train a neural network to play Tetris. Currently the best Tetris agents are designed using hand-coded methods and outperform NN agents by a huge margin. We investigate the possibility of training a Tetris agent using DQ-learning with the help of replay memory and a target model strategy. Ultimately all of our experiments failed due to a bad combination of hyperparameters and the unforeseen consequences of various types of reward-shaping. However, our experiments did not exhaust the possibilities of the exploration of this agent implementation.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

The application of machine learning to the design of game-playing agents has always been popular among AI enthusiasts. It provides a fun way of learning about and testing different problem-solving strategies which might then be transferable to more useful applications. Moreover, the widespread availability of increasingly advanced technology has enabled practitioners to experiment with training methods involving neural networks which were previously considered too impractical to implement. Game-playing agents trained with deep reinforcement learning have proven capable of achieving superhuman performance in a number of games, including chess, Go, and even Starcraft II. Granted, the development of these agents leveraged the enormous computational power of Google's state of the art hardware, using tensor processing units designed to train neural networks with maximum efficiency[1]. However, even comparatively modest hardware has been adequate for training agents to play at a superhuman level in games like backgammon[2] and various classic Atari games[4]

In this paper we discuss our efforts to train a Tetris agent using Deep Q-learning. In designing our Tetris agent, we used a strategy combining deep Q-learning with replay memory. This decision was inspired by a similar approach used to successfully solve the Classic Cart-Pole Problem [1]. Cart-Pole is a simple game which requires the player to balance an upright pole on a moving cart by manipulating the horizontal movements of the cart. Our goal was to determine whether the approach used to solve this much simpler problem could be generalized to a more complicated problem like Tetris [2]. Introduction

II. Background

Tetris has been hailed as one of the greatest games of all time by Electronic Gaming Monthly and IGN and is one of the most downloaded game of all time. It is therefore somewhat surprising that the use of neural networks to play Tetris has received somewhat less attention than other games. Perhaps this is just because most attempts at implementation have yielded poor results. Moreover, the success of hand-coded Tetris agents not using neural networks has set a very high bar for possible improvement. For example, an agent created by Colin Fahey using hand-coded methods was capable of scoring a maximum of 7,216,290 lines in a single game [3]. By contrast, most agents implemented using neural networks yield mediocre performance even by human standards. The one exception we could find was the DQL agent designed by Daniel Gordon, which was not only capable of scoring an average of six hundred lines per gam but was also able to clear a majority of lines in an average game by scoring tetrises (using the long piece to simultaneously clear four lines).

It must be said, however, that the hand-coded methods maximize survivability rather than points per game. This is usually achieved by adopting a conservative strategy of clearing single lines whenever possible in order to minimize the height of the tower of pieces. However, this results in fewer points than would be achieved by scoring multiple lines. Official competitions between human players use an older version of Tetris with a limited game-length. Players are required to score more points than their opponent before reaching a maximum game level in which the pieces fall to the bottom of the Tetris board faster than the game can receive inputs from the controller. The player with the most points wins regardless of the survival time (measured as the number of pieces played). Therefore, competitors must adopt a more balanced strategy of scoring multiple lines at a time while still managing to survive for as long as possible. The use of neural networks to develop a superhuman Tetris agent which adopts a more human playing style might reveal useful strategies to be used by Professional players. Nevertheless, it remains a difficult task to train even a competent NN Tetris player, let alone a superhuman one.

Previous efforts to create a DRL Tetris agent have relied on a hand-coded method known as reward shaping to achieve any success. Essentially this involves altering the game rewards received by the agent in such a way that it learns to mimic strategies which are intuitive to human players. This has the advantage of decreasing the training time required to achieve basic competence. However, it also has the disadvantage of limiting the agent's ability to find effective strategies which are counter-intuitive [3]. Background

III. Methods

Q-learning is a reinforcement learning algorithm which trains an agent to choose actions which are predicted to yield the highest sum of rewards from the current state onward. On each training step, the agent

performs whichever action it expects to yield the highest Q-value. Initially the agent knows nothing and performs random actions. However, through a long process of trial and error it learns to perform better and better actions by continually updating the expected Q-values for explored action-state pairs. Given enough training time and freedom to explore different possibilities, the agent will increasingly choose the optimal action in any given state. For problems involving a relatively small number of states, the solution can be found through brute force by storing the entirety of state-action pairs and their rewards in a table which is then consulted and updated after each training step. However, for most interesting problems, the number of possible states is so large that storing them all remains infeasible, so instead a Deep Q-learning neural network (DQN) is used as a Q-function approximator to map state-action pairs to Q-values.

We used the Keras functional API to build and train our DQN model and the OpenAI Gym to simulate the Tetris environment. We used a parallel convolutional network architecture in order to discern features of three different shapes: row-shape, column-shape, and 3x3. The selection of architecture was largely inspired by the one used by Daniel Gordon without understanding its specific advantages, other than the obvious fact that an agent might find it useful to be able to recognize the fullness or emptiness of each row and column. We used as inputs a picture of the current Tetris board down-sampled to a 20x10 array of 1s for occupied spaces and -1s for empty spaces. Our network did not have knowledge of the next piece.

We trained the network using an epsilon greedy policy, whereby the agent to performs random actions at a rate of epsilon, which forces exploration. Our epsilon began at 1.0 and annealed down to 0.001 with a decay rate of 0.99 per episode. The learning rate we eventually chose was 0.0003 after achieving bad results with higher rates.

To train our agent, we alternated between practice games and replay training sessions. Typically a DQ-learning model is trained using its own predictions to estimate the sum of future discounted rewards in the hope that its predictions will improve as training progresses. The purpose of using replay memory is to expedite the learning process by first acquiring correct rewards for chosen actions and using this information to create more accurate training targets.

We implemented replay memory using an array to store 500,000 of the most recent memories. The memory array was populated by experiences from practice games, during which the model weights were not actually updated. It was only between practice games that the weights were updated by replaying old experiences. Each memory consisted of a state, the subsequent state, and the action-reward pairs for both states: the values used to calculate the Q-value.

One of the ways our implementation differed from previous efforts was our use of the so-called target model strategy. This decision was based on its effectiveness in solving the previously mentioned Cart-Pole problem. The basic idea behind it is to increase learning stability during training. One of the problems with using the predictions of a DQN as its own model targets is that it tends to update predicted Q-values of actions which it has not actually explored. Given enough iterations and bad luck, this can cause meaningless predictions to exert greater and greater influence on subsequent learning. We attempted to mitigate this problem by creating a copy of our main model which was not trained but which could

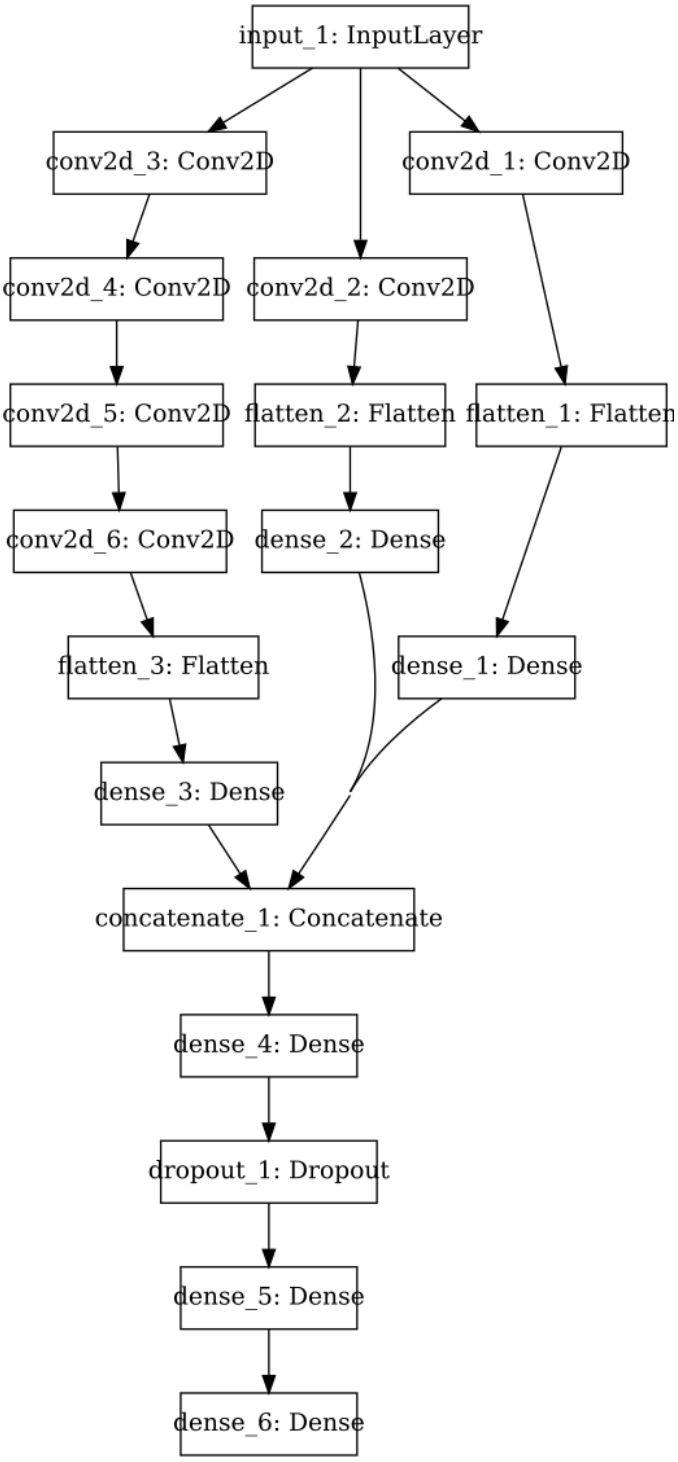


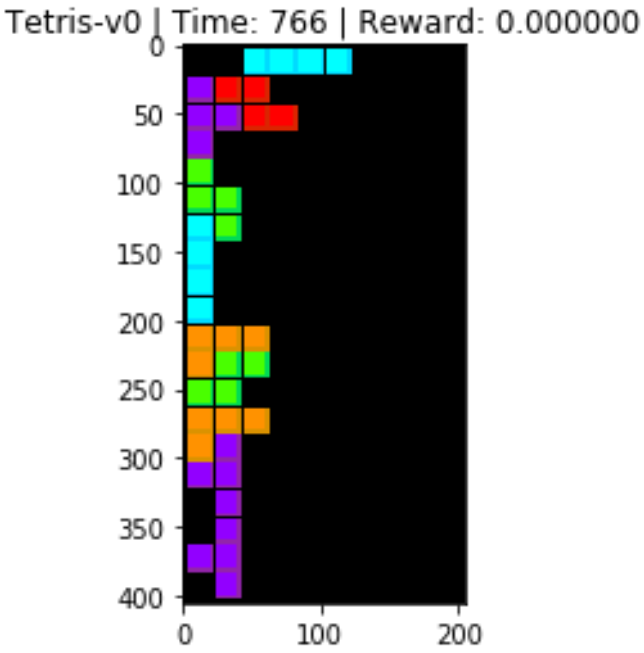
Fig. 1. Tetris Architecture!

continually generate predictions to use as targets for the main model.

In an attempt to guide the network in the right direction, we adjusted the rewards output by the game. We added a steep penalty for dying (-25.0), as well as punishing actions which led to an increase in the height of the tower of pieces. We also rewarded the agent for the number of occupied squares in the bottom four rows to coax the agent toward actions which would distribute the pieces more evenly and which might potentially clear lines. We terminated games after the height of the tower of pieces exceeded fifteen to speed up training time so that the agent did not waste time thinking about the best action in a hopeless position. [4]. Methods

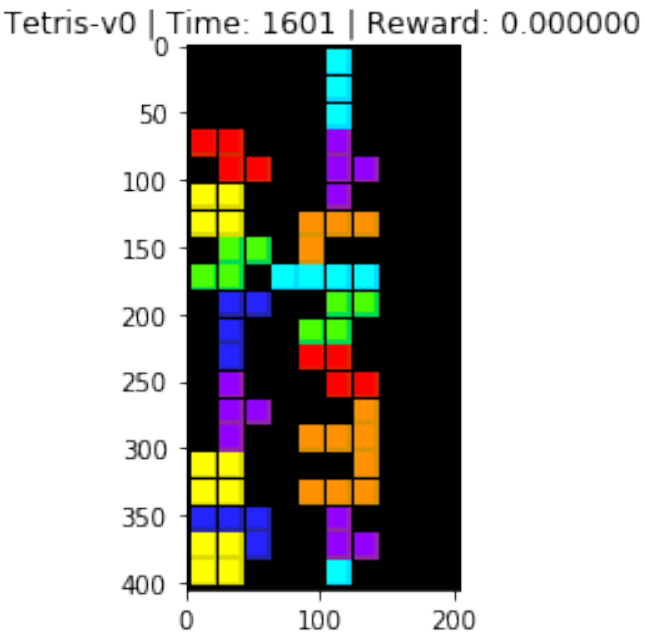
IV. Results

We experimented with different network architectures, hyperparameters, and reward shaping heuristics, but none of our agents achieved basic competence. During one of our earlier experiments, we used a learning rate of 0.001, which we thought was low. At the same time, we used a reward function which rewarded every single action by a small amount, thinking that the agent would learn to play moves which would allow it to survive longer. This was the result:



We speculate that, early in training when epsilon was close to 1.0, the agent learned to use all non-random actions to move pieces to the left of the board in order to survive more random actions per game. Then, even after epsilon had annealed down to the minimum of 0.001, the agent continued to adopt the same strategy. This indicates overfitting due to a poor combination of hyperparameters.

The figure below shows the performance improvement gained by selecting a lower learning rate: the agent actually learns to explore the rest of the board. Obviously it still has a quite a lot of room for improvement. Unfortunately we did not have enough time to attempt to train a better agent using a more appropriate learning rate. [5]. Results



V. Conclusion

Our original aim was to determine the effectiveness of using DQ-learning along with replay memory and a training model strategy to implement a Tetris agent. Our results definitely do not indicate success; however, they do not necessarily indicate failure either. Many alternative combinations of network architecture, hyperparameters, and reward shaping could have been tested with the same general algorithm. One of the biggest lessons we learned was how well intentioned reward-shaping can have drastically unfavorable consequences. It is very difficult to predict how different hand-coded rewards will effect the outcome.

Further work might investigate the development of a DQL Tetris agent while limiting the hand-coded manipulation of rewards as much as possible. The most interesting implementation of a Tetris agent would be one that uses no reward-shaping at all and simply allows its imagination to run wild. [1]. Conclusion

References

[1] J. Phillips, "Q-learning with feed-forward neural networks," *J. Name Stand. Abbrev.*

[2] S. Brown, "The evolution of alphastar."

[3] D. Carr, "Applying reinforcement learning to tetris," 2005.

[4] D. Gordon, "Learning to perform a tetris with deep reinforcement learning."

[5] e. a. V. Mnih, "Playing atari with deep reinforcement learning," *DeepMind Technologies*, 2013.