Learning to Perform a Tetris with Deep Reinforcement Learning

Daniel Gordon *

Abstract

Video games and simulated environments have been a popular testing ground for many recent RL algorithms because of their speed, repeatability, and scalability. However many of the state-of-the-art algorithms still fail at games requiring long-term planning. In this work, I train an agent to consistently perform the eponymous Tetris (clearing 4 lines at once).

1 Introduction

Tetris has long been the subject of interest in the video gaming community. Its simple, intuitive rules result in a fun yet challenging game. Optimally playing Tetris optimally is proven to be NP-complete [3], even when the order of pieces is known in advance. Yet people are able to play Tetris remarkably well, even in very difficult scenarios (`https://youtu.be/4OFq31lek2g`,`https://youtu.be/6YOR-nAnj4I`). Thus, the game of Tetris poses several interesting RL challenges.

In Tetris, the more lines cleared with a single block placement, the higher the score. This encourages players to perform what is known as a "tetris" where four lines are cleared at once using the line piece. This requires very long-term planning because the agent must learn to fill all but one row and defer sure low scores (clearing a single row) for the possibility of a much larger score in the future. In this project, I train an A3C agent to perform as many tetrises as possible. Specifically, my task will be to maximize score for 80 line games.

2 Related Work

Ever since DQN [9], people have been obsessed with training RL agents to play old-timey video games. Yet most of the focus has been on only a handful of games including Atari, Doom, and the first Mario level[4, 8, 9, 10]. I have never really liked those games though. They tend to be boring, and often only need few step look-aheads for optimal actions. Tetris is a game I actually like, and I'm pretty good at it (figure 1). Tetris is also a more interesting game because an agent can easily learn to clear single lines indefinitely (`https://youtu.be/z04CitKSoeg`), but will have a much harder time learning the long-term planning required for performing tetrises. After an extensive literature review (Google search), I have found no tetris programs that learn to consistently perform tetrises, and the few that are able to do tetrises use hand-crafted features.

Stevens and Pradhan attempted to train a DQN agent for tetris, but they had difficulty training it due to the long-term planning nature of the problem [12]. Their deep agent did not surpass the performance of agents given hand-defined features. Other explorations into using reinforcement learning to train tetris agents use explicit features such as board contours [2], holes and valleys [7], and aggregate height [5]. Stevens and Pradhan's pixel-based DQN approach additionally required prioritized experience replay, multiple action grouping and hand-crafted reward functions. In contrast, I use GA3C [1], the GPU-accelerated Asynchronous Advantage Actor Critic algorithm, and do not require experience replay or multiple action grouping. I did use a modified reward function which was crucial for success, but the network still operates on pixels and must learn to predict the reward and understand the features that contribute to it rather than being explicitly given the features. Plus, if you watch their agent play, it kind of sucks (`https://youtu.be/uzXMFUtvAlY`).

*`xkcd@cs.washington.edu`

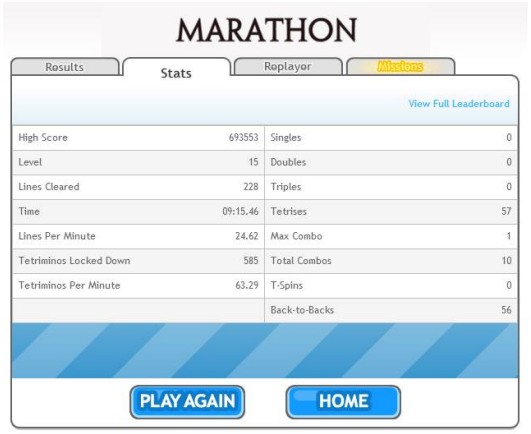

Figure 1: Results of a run I performed consisting entirely of tetrises.

2.1 Contributions

In summary, I differ from previous approaches in that I train an end-to-end system using only pixels as input (as well as a 1-hot vector for the next piece), and my agent is able to learn behavior that performs remarkably well, clearing an average of 53 of the 80 rows using tetrises. When left indefinitely, the agent performs an average of 600 lines, clearing 64% during tetrises.

3 Methods

To train the agent, I employed various standard techniques that were crucial to its success. In order to extract useful features for Tetris, I created a custom network architecture. The agent was trained with the GA3C algorithm [1]. To encourage certain behaviors, I employed reward shaping, which was crucial for teaching it to perform tetrises.

3.1 Custom Network Architecture

The standard DQN architecture [9] was not effective for training my tetris agent for several reasons. Firstly, the standard architecture begins with an 84×84 input and is eventually reduced to 8×8 in the final convolutional layers because of strided convolutions. This makes sense for Atari games where graphics use sprites and thus not every pixel is necessary to express the game state. However, in Tetris, the board state begins at 10 , so using their architecture the final layer would be 1 . Simply rescaling the Tetris board to be 84×84 does not make much sense either because this would distort certain rows arbitrarily. My network design eventually scales the initial input down by 8 but also includes skip connections to ensure the higher resolution feature maps are available to the later stages of the network. Specifically, after every pooling stage, the features are saved and concatenated before the first fully connected layer.

I also used domain knowledge about Tetris to craft convolutional layers which made more sense for the task. Specifically, I employed full row/column convolutions at each scale where a full row convolution of a $M \times N$ board is a convolution with a kernel shaped $M \times 1$, and a full column is shaped $1 \times N$. These can be helpful for extracting features such as if a column is entirely empty or if the column has any holes as well as if a row is nearly full and at which position a piece would need to be to fill the row. Also, the first convolutional layer has a kernel of size 5×5 because each tetris piece lies within a 4×4 square, but all remaining square convolutions are of size 3×3. Finally, each convolutional block consists of a 3×3 layer followed by a 1×1 downprojecting layer followed by a 3×3 layer inspired by Network In Network architectures [6]. A full network diagram from Tensorboard is shown in figure 2.

The input to the network consisted of images with 10 columns, 22 rows, and 6 channels (two timesteps concatenated). Each tetris board state was encoded into a 3-channel input of the currently placed pieces, the current active piece and a projection of where the piece would land if it were dropped at the current moment (often called the ghost piece). Additionally, a 1-hot encoding of the next piece to

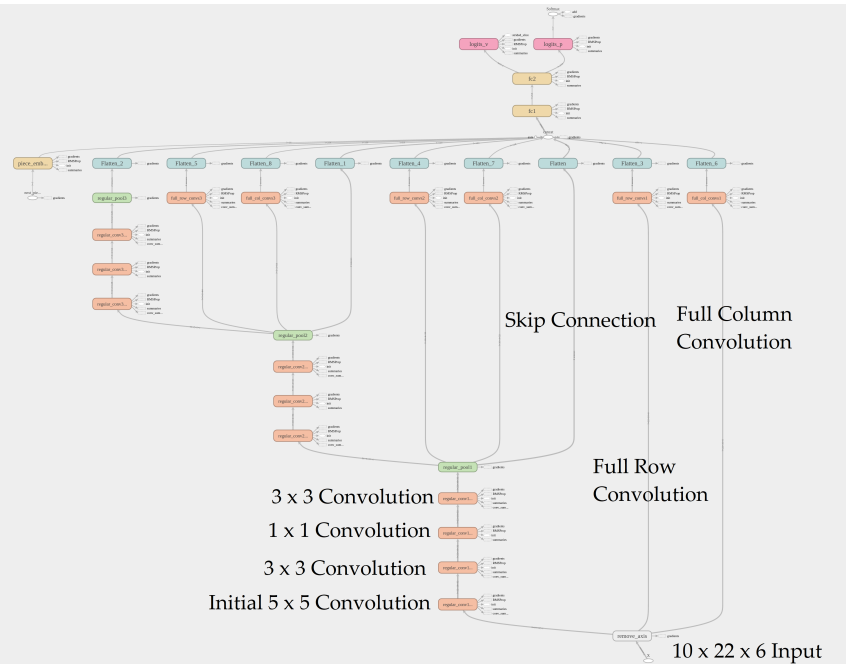

Figure 2: Network Architecture visualized by Tensorboard.

be placed was input to the fully-connected layers. The network outputs a Value function estimate and a distribution over the probabilities of the 7 possible actions (do nothing, move left, move right, rotate left, rotate right, move down 1, drop to bottom).

3.2 GA3C

GA3C [1] is a variant of the Asynchronous Advantage Actor-Critic algorithm which utilizes GPU hardware more effectively than the standard A3C. In this variant, each parallel environment performs actions and gathers observations independently, and each sends the experiences to an experience buffer. When the buffer is full, it is fed through the network and the weights are updated. When the independent environments need to make decisions using the network, they do the same thing except to a separate buffer which only performs the forward pass. This allows for better parallelism due to larger batch sizes and more stability during training. Using this algorithm I was able to train using 64 threads and a single Titan X GPU.

3.3 Reward Shaping

Effective reward shaping was crucial for training the agent to perform tetrises. The two largest modifications I made to the intrinsic Tetris reward function were as follows. Firstly, tetris normally rewards clearing lines proportionally to the square of the number of lines cleared simultaneously (1, 4, 9, 16). I modified this by changing the tetris reward to 64, thus dramatically encouraging tetrises over other options if possible, even in the long-term. Secondly, To encourage good wall construction, I added a penalty of (highest filled block) - (lowest unfilled block) per row which not only discourages holes, but discourages burying those holes, which makes it harder to recover from previous mistakes.

3.4 Implementation Details

The Tetris simulator I used was MaTris, an python based open-source Tetris clone (`https://github.com/SmartViking/MaTris`). I modified its main code to provide direct access to the board state rather than use a higher-resolution game image. By doing this, I was able to increase the speed of the simulator to roughly 1000 FPS on a single thread. Additionally, I modified the rules of the simulator to use the Tetris Grand Master randomizer as opposed to a simple random choice for both the initial piece as well as the next piece selection (`http://tetris.wikia.com/wiki/TGM_randomizer`). This results in more interesting games and fewer degenerate cases of "bad luck." The GA3C

implementation was provided by its authors via `https://github.com/NVlabs/GA3C` which I modified to work with MaTris instead of OpenAI Gym. All deep learning processing was done using TensorFlow. One thing I probably should have done was record the number of games/frames of experience, but I forgot to. Such is life.

4 Experiments

My experiments all revolve around fixed-length games. Namely, the agent plays until it completes 80 lines. This decreases outlier effects on the total population statistics because no games will go on indefinitely. It also acts as a natural normalization over things such as number of pieces, total number of moves, and maximum possible score.

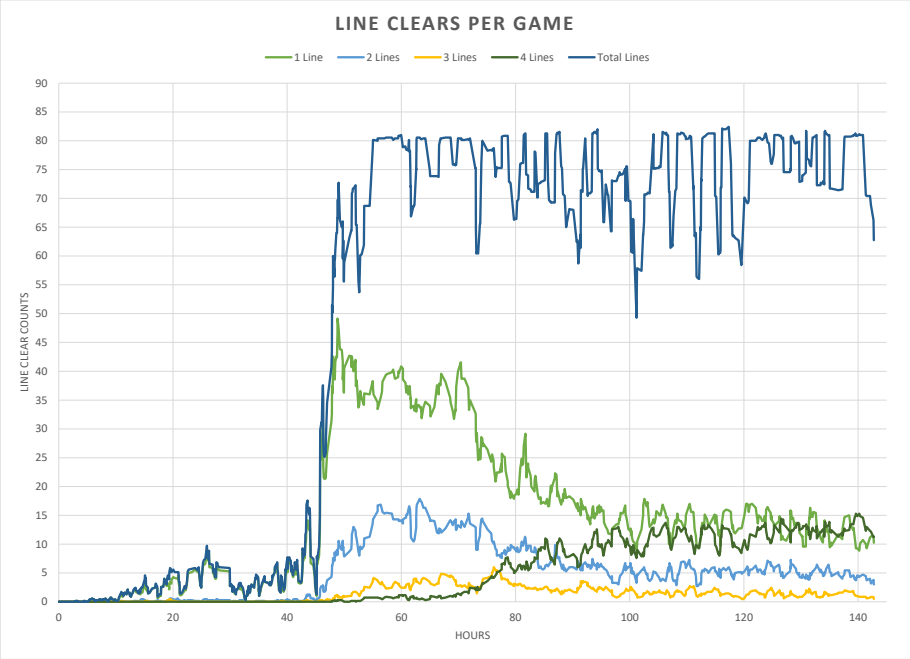

Figure 3: Count of simultaneous line clears per game. Note that 20 consecutive 4-lines corresponds to a perfect game.

5 Initial Results

RL works for a change! Without too much hand-tuning, I was able to train a super-human Tetris bot which can perform tetrises a majority of the time. Figure 3 shows the performance of the network over time. Initially, it barely clears any lines. Eventually (around 50 hours), it learns how to consistently clear single lines at a time and quickly reaches the 80 line limit. However, as training proceeds, it performs fewer single line clears, and more 2-line clears, then 3-lines, and finally tetrises. When watching the final agent play, I observe very human-like actions such as leaving an open column all the way on one side, forgoing 3-lines now for 4-lines soon, and blocking the last row rather than creating a hole. This can be seen in Figure 4 by the fact that the agent performs different actions given similar setups. The policy predictor also produces very peaked distributions, indicating that it has converged to a deterministic solution. More results can be seen at `https://youtu.be/iI6TQOQ_Ccc` (note, in this version I coded up the "hold" action, so it does even better than the results presented here.

5.1 Ablations

Once the initial agent was working, I tried various ablations to see which of the numerous parts were necessary and to see if I could speed up training. Firstly I tried simpler network designs without the skip connections which failed to learn a policy. Secondly, I tried removing the full row and full column convolutions. This resulted in slightly worse performance, but was still super-human.

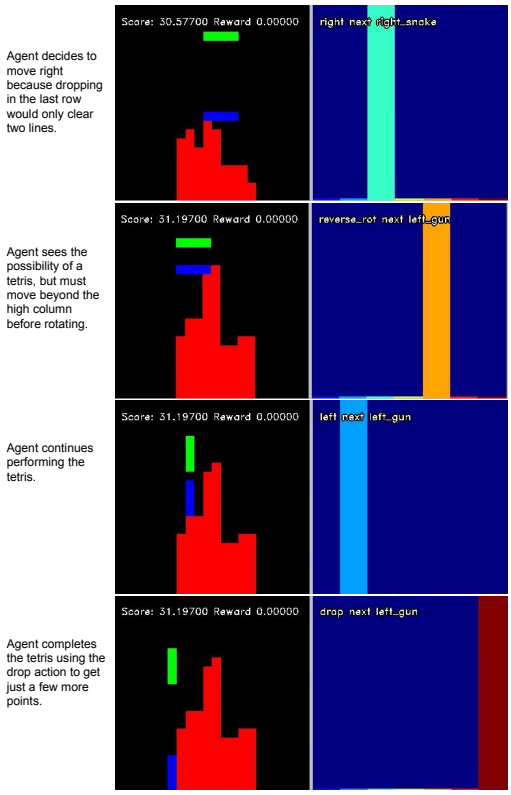

Figure 4: Qualitative results of various points in a single game run.

Thirdly I tried training a network with fewer channels in each layer to see if the network was truly using its capacity. This small network was learning, but at a very slow rate, and never matched the performance of the wider networks. Finally, I tried a network with no convolutional layers (4 fully connected layers) just to see if it could work. It showed very poor performance likely due to the overparameterization of the problem and the failure to reuse spatially similar features like convolutions are made for.

I also tried to bootstrap the network performance via behavioral cloning and imitation learning via DAgger [11]. Initially, I played 100 games to act as a training set. Given this training, the network was able to learn to consistently clear 4-10 lines, but very quickly ended up in unknown situations and immediately failed. Next, I tried using DAgger defining the expert policy as the already trained agent. This resulted in a new agent which could clear about 20 lines, but then fell into similar issues. I then tried fine-tuning these pretrained networks using RL and only updating the final fully connected layer, but this too resulted in poor performance. I never really figured out a good way of bootstrapping for Tetris.

5.2 Feature Maps

As part of my ever-growing personal TensorFlow suite, I always visualize the convolutional filters to see if there are any interesting patterns. It turns out that this time there were. These are shown in Figure 5. It clearly seems to learn filters that look like the line piece, T piece, L piece, square piece, and somewhat the S and Z piece at various orientations. Secondly, the square filters show lots of green and purple. Green blocks indicated that the kernel will activate wherever the current piece is (the second input image channel), and purple combines the first and third channel, essentially saying "what would the board look like if the current piece dropped right now." I thought it was particularly interesting that the network makes no distinction between the currently placed pieces (red channel) and the ghost piece (blue channel). Additionally, the first layer row and column maps have some interesting patterns which probably identify things like completely empty columns (useful for saving up to perform tetrises later) and nearly full rows (useful for identifying at which position a hole needs to be filled).

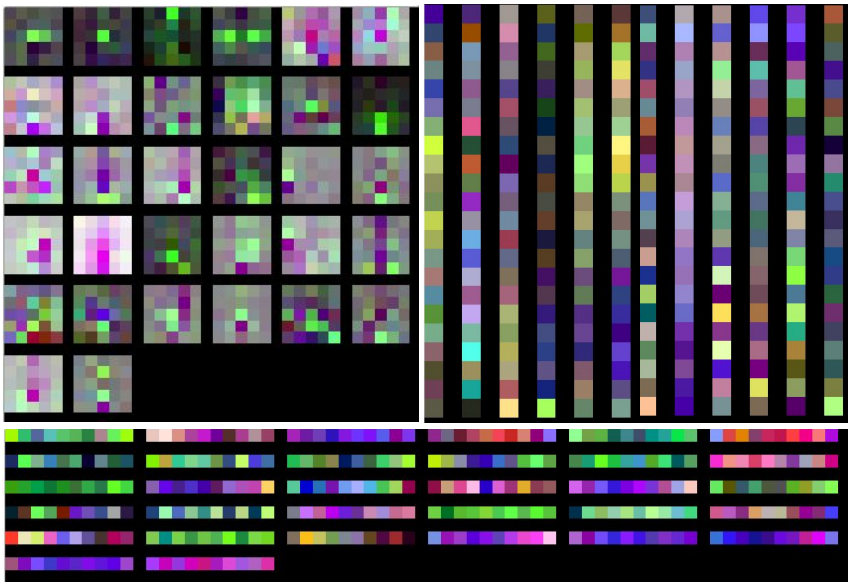

Figure 5: First layer square (left), column (right), and row (bottom) convolutional filters learned by the RL agent. Best viewed not as a JPEG but I was lazy.

6 Conclusion and Future Work

In this work, I trained a super-human tetris agent which learns the long-term planning problem of performing tetrises consistently. One minor issue is that the agent will eventually die, as it has been trained to maximize tetrises and be more risky than the single-line-clear agents. In future work, I would like to experiment with adding more penalty for dying to create an agent that never dies. I would also like to add the "hold" move to allow the network to get out of difficult-to-place pieces, and introduce an even longer term planning problem. Overall, I am happy with the performance of the agent, and could watch it play for a long time.

References

[1] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz. Reinforcement learning thorugh asynchronous advantage actor-critic on a gpu. In *ICLR*, 2017.

[2] D. Carr. Applying reinforcement learning to tetris. *Department of Computer Science Rhodes University*, 2005.

[3] E. D. Demaine, S. Hohenberger, and D. Liben-Nowell. Tetris is hard, even to approximate. In *International Computing and Combinatorics Conference*, pages 351–363. Springer, 2003.

[4] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pages 1–8. IEEE, 2016.

[5] Y. Lee. Tetris ai – the (near) perfect bot.

[6] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[7] N. Lundgaard and B. McKee. Reinforcement learning and neural networks for tetris. *University of Oklahoma, Website: http://www. mcgovern-fagg. org/amy/courses/cs5033 fall2007/Lundgaard McKee. pdf*, 2006.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[10] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

[11] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[12] M. Stevens and S. Pradhan. Playing tetris with deep reinforcement learning.