CS8803 - ACRL HW2: Tetris

Pol Llado *GTID: 903410149* William Tidwell *GTID: 901635874* Maksim Sorokin *GTID: 903364105*

1 Introduction

For this assignment we created an agent capable of clearing 10,000 lines on a simplified version of the classic video game Tetris. In our case, we only know the current piece (but not the next piece), and there are no dynamics, we simply rotate and shift the Tetrominos to place them. Throughout this process we experimented with various optimization and reinforcement learning approaches to solving this problem, but in the end it was the non-glamorous Cross Entropy Method that proved most fruitful. Our work relies on the foundations laid forth by two papers: [7], which describes the use of the Noisy Cross Entropy Method for training a Tetris agent, as well as its advantages and pitfalls, and [1], which details high quality features for board state space reduction and value evaluation.

2 Representation

We designed a deterministic agent that uses a scoring function, or controller, to guide the policy at each step. Each state-action pair is evaluated, and the action that corresponds to the highest score is the one selected. Our score is computed from a linear combination of features extracted from state-action pairs. In order to succeed with the simple architecture, we had to select an appropriate and powerful set of features to characterize our state-action pairs. More complex feature sets or policies could have been used but would have been a poor fit for Cross Entropy training methods as we seek to avoid the Curse of Dimensionality.

$$\pi(s) = \arg\max_a w^T f(s,a) = \arg\max_a \sum_{i=1}^{n} w_i f_i(s,a)$$

We decided to featurize state-action pairs directly to quantify the quality of the move itself. We featurized our 21x10 board using nine features. We used 8 of the features from [1], which themselves were introduced by [3] and [8]. These features are representative of a high quality board state, and are used to analyze the desirability of a given state-action pair. In addition to these 8, we added one of our own to indicate whether a certain move loses the game. Preliminary tests showed that, in certain situations, using only the 8 features from the literature, the highest scoring move would also be one that ended the game. By penalizing a losing move, we were able to prolong the game for at least a few more moves and, on occasion, clear a few more lines. We extracted the following features $f(s,a)$ from our 21x10 board after applying each action:

- *Landing Height* - Average height of our placed piece [3]
- *Eroded Pieces* - Cleared lines x squares from our piece in cleared lines [3]
- *Row Transitions* - Sum of alternations between filled / non filled squares in all rows [3]
- *Column Transitions* - Sum of alternations between filled / non filled squares in all columns [3]
- *Hole Counts* - Empty squares with squares filled above them [3]
- *Well Depths* - Total depth of wells (top layer dips with at least two pieces surrounding them) [3]
- *Hole Depth* - Sum of filled cells above holes [8]
- *Row Holes* - Number of rows with a hole [8]
- *Lose* - Indicator that this move would lose the game - Our Feature Addition

Because we are using the Cross Entropy Method, our reward only needs to be provided at a trajectory level, versus a state/action level. As such, our reward was simply the total number of lines cleared at the end of each episode. We experimented with other rewards such as for each piece played or clearing multiple lines, but as the final objective was lines cleared, we found that it worked well.

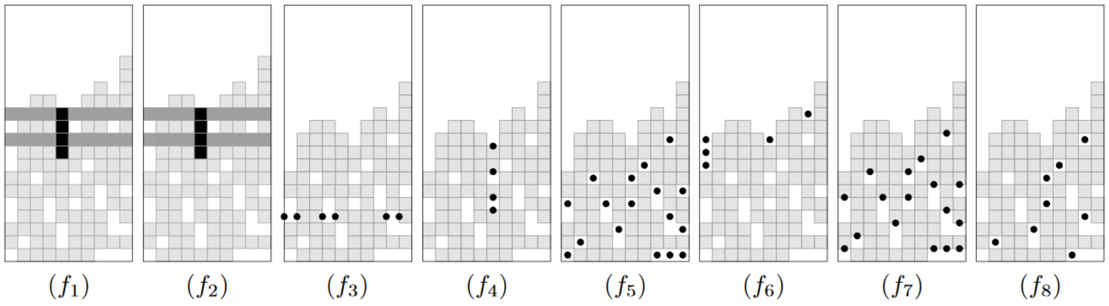

Figure 1: Features used to characterize board/action pair [1]

3 Training Methodology

In the above section, we defined an agent whose policy is based on a set of extracted features $f_i(s,a)$ and weights w_i. To formulate this policy, we had to first learn appropriate values for the weights. After much research, we decided on the Noisy Cross Entropy Method (NCEM) to learn our policy. NCEM maintains a distribution over the parameters of a model and updates the distribution based on repeated sampling of an elite population. We represented our model as a set of independent Gaussians for each parameter for simplicity. To initialize our distribution effectively, we first ran many simulations until we had 15 samples that cleared at least one line. We used the calculated mean and variance to initialize the first generation of children for our agent. This hot-start is different than the traditional algorithm, but, due to the difficulty of clearing rows randomly, it allowed us to jump start initial improvement.

At each iteration, we sampled 50 children from the distribution and played 1 game with each. The 10 highest performing children were kept as the elite, and a new distribution was calculated by taking the mean and variance of this elite set. Unlike the traditional Cross Entropy Method, we added noise to our variance to ensure slower convergence and to avoid local minima [7]. 30 games were then played using the mean of the elite as the weights, and the average score of these games was used for evaluation. The process repeated until we learned values of w_i capable of clearing well over 10,000 lines of our Tetris simulation.

```
$\mathcal{D} \leftarrow \mathcal{N}(0,1)$
**while** Not Converged **do**
  **for** j = 1 to n **do**
    Sample set of weights $w_j$ from $\mathcal{D}$
    $Hist_i \leftarrow R(w_j)$ {Forward Rollout with Parameters, Return Lines Cleared}
  **end for**
  $\mathcal{E} \leftarrow \oslash$
  **for** i = 1 to elite_count **do**
    $j \leftarrow \arg\max_{jnotE} Hist_j$
    $\mathcal{E} \leftarrow \mathcal{E} \cup w_j$ {Take Set of Elite Parameters}
  **end for**
  $\mu_{new} = \mu_{w_{\mathcal{E}}}$ {Take Means of Weights in Elite Set}
  $\sigma_{new} = \sigma_{w_{\mathcal{E}}} + \epsilon$ {Variance of Elite Set + "Noise"}
  $\mathcal{D} \leftarrow \mathcal{N}(\mu_{new}, \sigma_{new})$ {Update Distribution}
**end while**
```

4 Results

We ran our Noisy Cross Entropy Method with a population size of 50 for each generation using 30 runs for evaluation purposes. Figure 2 below shows the average number of lines cleared at the end of each generation. Our final agent was able to clear over 12,000 lines on average. Given that the curve is still exponential and has not yet flattened out, we believe there is still much room for improvement. However, at this stage each generation was taking over a day due to parallelization and extremely successful games.

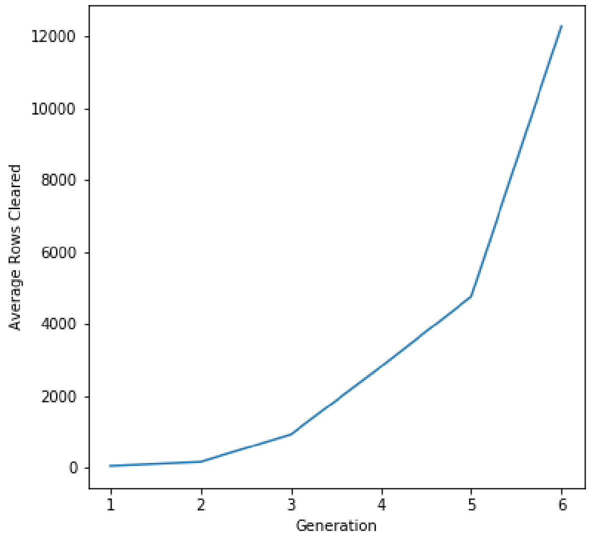

Figure 2: Average Rows Cleared by Generation

The results of a final evaluation run, along with the seed of each game, can be seen in Figure 3 below. Over 15 games, we again averaged over 12,000 lines. It is worth pointing out the wide range of the scores, with a minimum of only 529 and a maximum of 38,049. The high variability of our agent confirms that additional training may help reduce the variance in results.

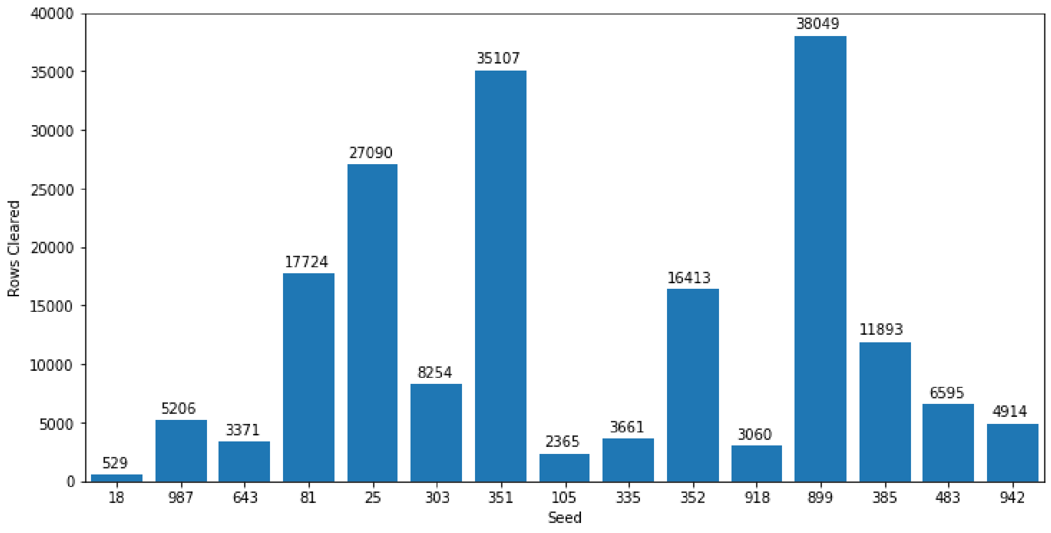

Figure 3: Rows Cleared at Final Evaluation with Seed

Figure 4 Shows the final distributions of our weights. Some features such as Wells and Landing have narrow distributions, while Lose experiences a higher variance. A possible reason is that the Lose Feature is seen less often than the others, and only when the game is almost lost. As a result it helps extend last-ditch survival, but may not have as high magnitude of an impact on average game length.

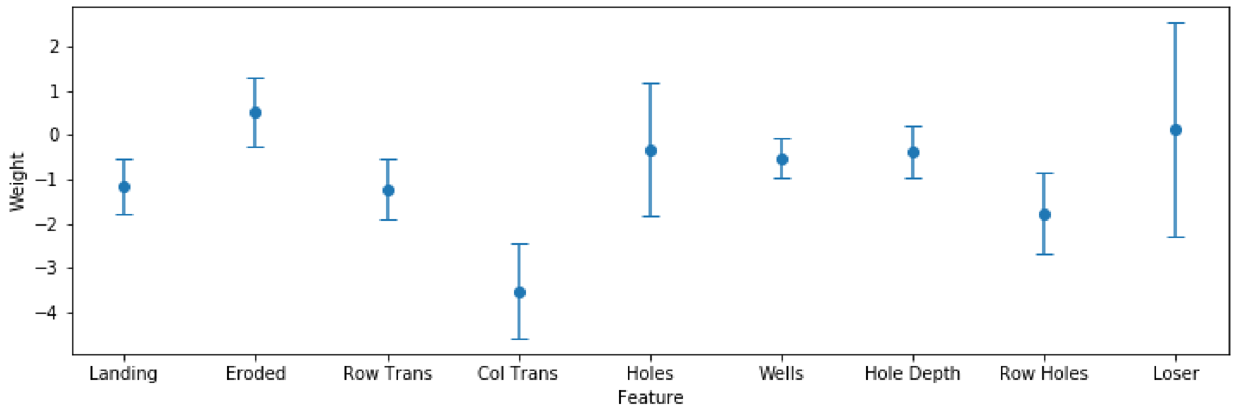

Figure 4: Learned Distribution of Feature Weights

5 Conclusion

The main lesson we all learned is that Tetris is a hard game to solve. We began by trying to implement the newer, more popular algorithms, including Deep-Q Networks, Policy Gradient, TRPO, and A2C. With many hours of debugging and training with these methods, we were only able to achieve poor results of up to 15-20 Lines. Next we introduced some hand-engineered features of higher dimensionality to our networks as opposed to relying on board state, and that improved results slightly using Policy Gradient methods, but we were unable to reach requirements. Eventually, we moved to more high order features such as the ones listed in this report, but these accrue significant compute time with the rollouts necessary for Gradient/Q based methods.

In the end, the simpler approach of CEM for optimizing a linear controller worked, clearing up to 38000 lines in a game. We're not sure if that lesson was an intentional part of the assignment, but we all learned that, often, it's best to start with the basics and to only add complexity when needed. Given more time, we'd love to try combining approaches, by using to NCEM to initialize a controller that does well, and then letting a gradient based technique fine tune the controller or add additional inputs.

References

[1] Amine Boumaza. "How To Design Good Tetris Players". In: (2013). URL: https://hal.inria.fr/hal-00926213/document.

[2] Donald Carr. "Applying reinforcement learning to Tetris". In: (Apr. 2005).

[3] C. P. Fahey. "Tetris AI, Computer plays Tetris". In: (2013). URL: https://www.colinfahey.com/tetris/tetris.html.

[4] Nicholas Lundgaard and Brian Mckee. "Reinforcement Learning and Neural Networks for Tetris". In: (Mar. 2019).

[5] Bruno Scherrer et al. "Approximate Modified Policy Iteration and its Application to the Game of Tetris". In: *Journal of Machine Learning Research* 16 (2015), pp. 1629–1676. URL: http://jmlr.org/papers/v16/scherrer15a.html.

[6] Ozgur Simsek, Simon Algorta, and Amit Kothiyal. "Why Most Decisions Are Easy in Tetris—And Perhaps in Other Sequential Decision Problems, As Well". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 1757–1765. URL: http://proceedings.mlr.press/v48/simsek16.html.

[7] Istvan Szita and Andras Lorincz. "Learning Tetris Using the Noisy Cross-Entropy Method". In: (2006). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.81.6579&rep=rep1&type=pdf.

[8] C. Thierry and B. Scherrer. "Construction dun joueur artificiel pour tetris". In: *Revue dIntelligence Artificielle* 23:387407 (2009).

[9] Christophe Thiery and Bruno Scherrer. "Improvements on Learning Tetris with Cross Entropy". In: *International Computer Games Association Journal* 32 (2009). URL: https://hal.inria.fr/inria-00418930.