# $\lambda$-Policy Iteration in Long Episodes of Tetris

Spencer Graves (SUID 066005103)

## Abstract

This paper examines the $\lambda$-policy iteration algorithm and the worsening performance observed throughout training when the algorithm has been applied to Tetris. The same application is examined, the behavior of note is produced, and a variation of the algorithm is demonstrated to provide monotonic increasing performance instead of experiencing a dip. An interpretation of the phenomenon is provided that attributes the problem to episode length increasing to the point that states may recur during a single episode.

## 1 Introduction

This paper examines the game of Tetris, a game with some history of study in reinforcement learning. Tetris is an interesting case study for examining problems in which the state space is too large for tabular methods, and which must instead be handled with function approximations in order to construct value functions or to parameterize policies. It also has the property that rewards are received as a result of the combined effect of many past actions.

What follows will concern the application of the $\lambda$-policy iteration method. An initial implementation by Bertsekas demonstrated an early dip in performance for which Scherrer's work gave a well-evidenced and plausible explanation [1, 2]. The implementation from Scherrer et al., used as a comparison to further work on policy iteration [3], can be seen to also exhibit performance that decreases over time past some point. This second dip has not so far been explained. These results are approximately replicated, with some important distinctions, and then a modification is introduced that is seen to resolve the issue of decreasing performance.

## 2 Related Work

Two past implementations of $\lambda$-policy iteration have already been mentioned. An excellent survey from Algorta and Şimşek details where these attempts fit into the application of other algorithms to Tetris [4]. Of these, particularly relevant here are direct search methods and the work by Gabillon, Ghavamzadeh, and Scherrer, which are noteworthy not only for the algorithms developed but also for demonstrating that linear estimators can achieve performance of many millions of lines [3]. This last work is also the source of the corroborating plots that show the performance dip of $\lambda$-policy iteration which became the focus of this investigation.

A deep neural network approximating architecture comprises the current state-of-the-art on the problem of Tetris [5]. To understand the success of this algorithm, A3C, it seems prudent to attempt to understand the influence both of the choice of learning algorithm and also the choice of approximating architecture. By addressing the dip in performance of $\lambda$-policy iteration , some insight might be gained into what constitutes good training data for other approximating architectures, and also into the limitations of a particular feature set. These insights may have practical benefit when working with other and more recent methods.

## 3 Features and Data

The Berstekas features have been alluded to, but will be explained in detail here. They are used to construct a linear estimator of the value function, and are comprised of the constant value one; the height of each column; the forward difference between each column height and the next; the maximum height of any column; and the to-

tal number of holes below all the columns. The original definition of a hole states that it is an empty space fully enclosed by game pieces; in this work it is treated as the sum of all empty spaces that fall anywhere in a column below the column height, including those that are not enclosed at the sides. This seemed a reasonable and computationally expedient departure to make, since the typical game rules for computer Tetris do not allow sliding pieces or rotating them as a human player might to fit the piece below an overhang, and so holes as defined here indicate squares which are not accessible to the algorithm until a line is cleared. This approach seems to have been taken in other literature but the distinction from the original meaning is not always made explicit.

The dataset is generated using a C♯ program executing on a CPU to track the board state and to supply pieces, using Python for the regression step. An intuitive implementation of Tetris requires branching, looping, and comparison operators that do not translate well to GPU computing, which introduces some difficulty in generating large numbers of long games. This motivated organizing the algorithm into operations on batches of not more than a few million moves.

It is important to mention that the Tetris simulator was custom-written for this task because, as was seen during development, any deviation from the intended Tetris ruleset can have dramatic consequences on the value functions learned. This should be seen as a caution to implementers.

# 4 Methods

Pseudo-code is provided for the $\lambda$-policy iteration algorithm and for a new variant parameterized with two terms.

The cost function featured in the pseudo-code is given as

$$J(\theta; a) = \sum_{m=0}^{M} \sum_{i=0}^{N_m - 1} \left(v(\theta, x_{m,i}) - a_{m,i}\right)^2$$

**Policy evaluation**
**for** $M$ games of length$N_m$ **do**
  sample $(T^{\pi_k})^{N_m} v_k$
  **for** $i \in [0, N_m - 1]$ **do**
    collect $a_{m,i} = v_i + \sum_{j=i}^{N_m - 1} (\lambda\gamma)^{j-i} \delta_j$
  **end**
**end**
**Policy improvement**
$\pi_{k+1} \leftarrow \arg\min_\theta J(\theta; a)$

**Algorithm 1:** $\lambda$-policy iteration

where the $a_{m,i}$ are defined as

$$a_{m,i} = v_{m,i} + \sum_{j=i}^{N_m} (\lambda\gamma)^{j-i} \delta_{m_j}(x_{m,j}, x_{m,j+1}; \theta_k).$$

and the $\delta$ are the temporal differences. This geometric averaging of the temporal differences has a similar effect to eligibility traces of propagating rewards to adjacent states [6], and is the central feature of $\lambda$-policy iteration . The minimization is done by least-squares regression. Here $\pi_k$ denotes a policy that is greedy with respect to the value function $v_k$ parameterized by $\theta_k$. Some liberty has been taken in the pseudo-code in substituting subscripts which are indices into generated series for arguments of the function that is sampled by that series where this meaning was thought to be clear.

Two modifications are made in the choice of policies for both algorithms considered here. Rather than have the policy be exactly greedy, let the action set first be restricted to the set of moves which do not terminate the game. Secondly, use a function $Q$ instead of $V$ which is aware of reward to be received by transitioning to the next state - this information is available naturally in the implementation, as cleared lines must be discovered to generate the next state. This allows value functions that effectively compare non-scoring moves but poorly estimate the final score to perform well initially, which will be seen to have two interesting effects. Both of these modifications are thought to aid in exploration. The policy is precisely defined as

$$\pi_k(x_t) = \max_{x_{t+1} \in \mathcal{X}_{w_t, x_t}} r(x_t, x_{t+1}) + \gamma V(x_{t+1}; \theta_k).$$

Here $\mathcal{X}_{w_t,x_t}$ is the set of non-terminal moves given the next piece $w_t$ and board state $x_t$.

One other significant modification was made additionally. Despite having verified the convergence of the algorithm both to TD(0) and to policy iteration as $\lambda$ goes to zero or one, respectively, observed value functions even of very successful controllers did not always correctly give a roughly decreasing value as the board height increased, when operating within this regime where $\lambda$-policy iteration sees decreasing performance. Examining the numbers lead to the following proposed algorithm, which takes only a tail of fixed length $\eta$ of the last weighted TD targets of each run.

To deal with long game lengths, the exponentiated term in the inner sum of the provided cost function was truncated when the result of exponentiation dipped below a certain threshold, 1E-3 in this case. $\lambda$ was fixed at 0.5 throughout.

To deal with high variance in the estimate of the application of the Bellman operator observed around the peak performance of the first algorithm, the second algorithm also applies the learning rate, $\alpha$, to set the weights between their previous values and those that exactly optimize the cost function. This same modification was used by Bertsekas and Tsitsiklis with smaller batches, but was seen here as necessary with larger batches as scores improve [2].

**Policy evaluation**
**for** $M$ games of length$N_m$ **do**
  sample $(T^{\pi_k})^{N_m} v_k$
  **for** $i \in [N_m - 1 - \eta, N_m - 1]$ **do**
    collect $a_{m,i} = v_i + \sum_{j=i}^{N_m} (\lambda\gamma)^{j-i} \delta_j$
  **end**
**end**
**Policy improvement**
$\pi_{k+1} \leftarrow (1-\alpha)\pi_k + \alpha \arg\min_\theta J(\theta; a)$

**Algorithm 2:** $\lambda$-policy iteration with two additional parameters.

## 5 Experiments

The $\lambda$-policy iteration algorithm described conforms very closely to the plots from Scherrer et al [3] (Figure 1). An initial spike is visible, and then the performance settles to score on average 5,000 lines per game. The first rise that decays down to zero occurred when the Bertsekas parameters are supplied with the traditional defaults developed by Van Roy [7], which produce a negative value function for all boards initially. The weights must cross zero before the second peak occurs and they ultimately settle. This zero-crossing is reflected also in the work from Scherrer [1], without an initial peak. The use of a function $Q$ which is aware of the reward seems to allow the parameters to make initial progress towards their opposites, where Scherrer reports the performance decaying to zero immediately. The first peak in his work coincides with the second observed here.

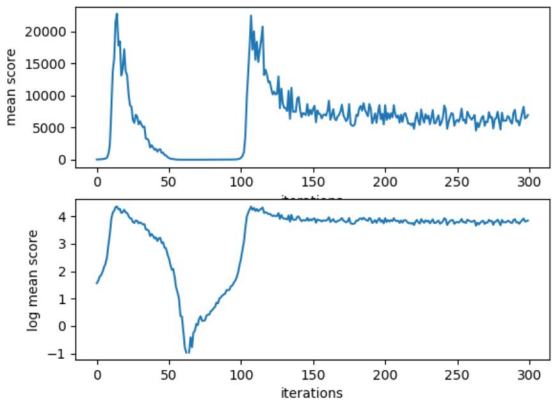


Figure 1: The traditional algorithm with its unmodified update rule and characteristic overshoot. After the initial false start, attributed to the substitution of $Q(x, \pi(x))$ for $V(x)$, it converges to around 6,000 lines scored on average in agreement with past implementations in other studies.

These results will be compared to the modified algorithm having two additional parameters, the learning rate $\alpha$ and the tail length $\eta$, that may be adjusted throughout training in order to make the most expedient progress. For the results presented here, a flat learning rate of 0.1 was used in order to more directly compare the effect of varying the tail length. Tails of length 500 and 300 were taken, in addition to taking all of the

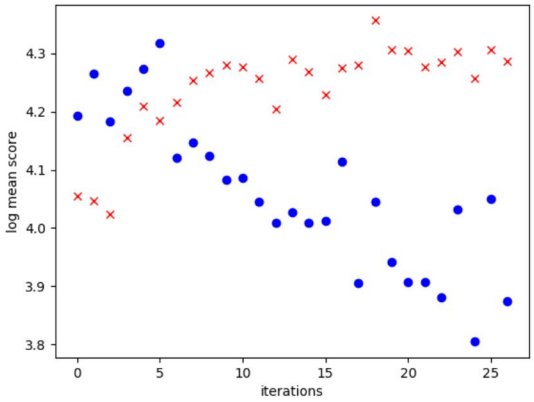


Figure 2: The red series demonstrates continued improvement by learning on only the past 300 examples, and the blue series follows the unmodified algorithm. Both series start with the same initial weights, with their initial separation giving an indication of the variance of the algorithm.

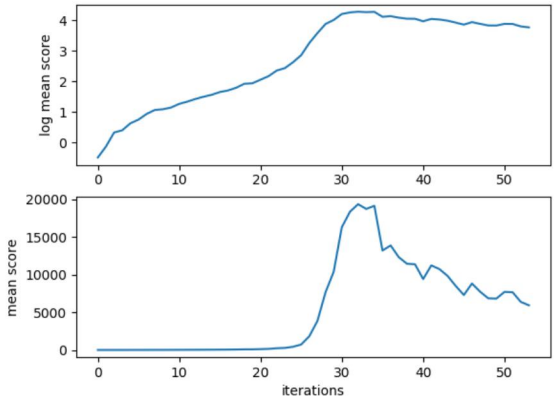


Figure 3: The modified $\lambda$-policy iteration algorithm with $\alpha$ of 0.1 and $\eta$ equal to 500, still seen to overshoot but without a false start.

usual input to the regressor from the original algorithm but still applying the learning rate $\alpha$. The discount rate was taken to be close to one, with little difference if a small discount was applied.

The optimization of the value function was done using least-squares regression and all scores given are averages over 128 games. A cross-entropy search was also implemented, both to provide a benchmark to further test the correctness of this implementation by examining discrepancies with [3]. This algorithm behaved as expected, scoring a few hundred lines, and was quickly outpaced by the other methods explored here.

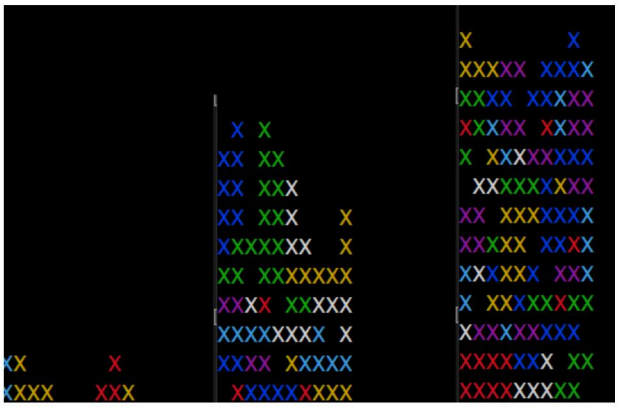

Figure 4: A few board samples from the middle of play. From left to right, board values are 33, 19, and 3, showing the correct tendency of the values to decrease as the game concludes. In general the values need only vary in proportion to the expected final score. See also the remarks on the substitution of $Q$ for $V$.

## 6 Results

Training was halted when an average score of 29,647 was achieved over the last five iterations, well above the peak overshoot. This is still some orders of magnitude below the best known linear estimators in the Bertsekas features [3], and so it is plausible that this estimator could be arrived at. The modifications are seen to produce consistent growth for 30 iterations past the overshoot

of the unmodified algorithm and see a doubling of performance.

Since the learning rate differs between the two plots of figure 2, it is not quite fair to compare the number of iterations and the trends directly. Figure 3 is helpful in that regard for showing that the same learning rate and even an adjustment due to the parameter $\eta$ still produce overshoot and a long-term decreasing trend when $\eta$ is large enough for the board to be cleared and filled several times. The theory proposed is that the estimate of the $N_m$ applications of the Bellman operator is no longer an unbiased estimator for high values of $\eta$. Combined with function approximation and bootstrapping, this leads to instability until the game length dips.

It should be noted, however, that $\lambda$-policy iteration is a stochastic algorithm, with not all curves exhibiting the same properties. The baseline implementation demonstrates with its initial false start that qualitatively different behavior can be achieved by different curves - the main risk in this analysis is that it may be necessary to examine many more runs in order to say that the behavior observed is representative.

# 7 Discussion

The value of the tail length $\eta$ was selected based on the observation that high-performing estimators nearly clear the board every few hundred moves. A good estimator, producing phase portraits that overwhelmingly reside in the states that describe relatively empty boards, might be able to revisit states with some considerable likelihood. The cumulative results obtained from those states would then be correlated. These correlated observations would comprise an increasing percentage of the data fed to the regressor. Taking the tail of the target set aims to reduce the likelihood that a state is visited multiple times, while keeping the terminal state information. Selecting $\eta$ random samples of the regression targets instead of the last $\eta$ produced results much like those of the original algorithm.

The result of this change also has the effect of causing the value function to behave sensibly, and to once again give lower values at the end of the game than at the start. Figure 4 shows this property of the value function.

# 8 Conclusion

The value of these findings likely lies outside of this particular algorithm and application. Modern data-intensive, high-performance methods are generally structured around asynchronous updates as a way of dealing with the extremely long natural episode length, and so have their own ways of dealing with correlated observations. It nonetheless seems valuable to understand the nature of the training data regardless of the choice of approximating architecture, and in that respect these findings may have some use.

# 9 Future Work

If the interpretation give for the observed overshoot of unmodified $\lambda$-policy iteration is correct, then the same effect should be seen when the algorithm is applied to tasks with discrete state spaces that see this lengthening of episodes, and not when applied to other problems. A more thorough treatment of this phenomenon would include this exension to other tasks.

It may also be possible and fruitful to attempt to quantify the error in the policy evaluation step. Existing work has examined how the propagation of errors through the $\lambda$-policy iteration algorithm in the step that estimates the Bellman operator translate to error in the final policy [1]; quantifying the error that is claimed to be introduced here may allow for an analytic description of the difference in long-term behavior of these two algorithms.

Lastly, if $\lambda$-policy iteration is to be explored in more extended training sessions than were conducted here, there remain substantial speed and efficiency improvements to make. Only 0.3% of the moves taken were used in regression at the end of training. Addressing this fact and moving to a GPU implementation would be recommended.

## References


[1] B. Scherrer, "Performance bounds for $\lambda$ policy iteration and applications to the game of tetris," *Journal of Machine Learning Research*, 2014.

[2] D. Bersekas and J. Tsitsiklis, *Neuro-Dynamic Programming.* Athena Scientific, 2016.

[3] M. G. V. Gabillon. and B. Scherrer, "Approximate dynamic programming finally performs well in the game of tetris," *Neural Information Processing Systems (NIPS)*, 2013.

[4] S. Algorta and . Şimşek, "The game of tetris in machine learning."

[5] V. M. et al., "Asynchronous methods for deep reinforcement learning," *International conference on machine learning*, 2016.

[6] A. Barto and R. Sutton, *Reinforcement Learning.* The MIT Press, 2018.

[7] B. Van Roy, "Feature-based methods for large scale dynamic programming," *Lab. for Info. and Decision Systems Reports LIDSTH-2289*, 1995.