16-899C: ARCL - Homework 1

Chris Cunningham, Daniel Silva, Bhaskar Vaidya

1 Introduction

Tetris is a complex game that involves dropping differently configured tetrominos into a pit, where completely filling a horizontal line erases it, and allows the game to progress for a longer time. The goal of the game is to obtain as many points as possible, where a point is obtained when a line is cleared. The game is stochastic, meaning that the next piece that drops is selected randomly. The game is also unwinnable, as there exists an adverse sequence of tetromino drops that will result in guaranteed death. The stochasticity and enormous atate-space of the problem make it difficult to do well for both human players and learned/generated policies.

2 Strategies

Strategies for creating a good policy for a tetris agent are widely varied. As tetris is a stochastic game, where using the same policy in two different games could have two different results, we consider mean scores as the benchmark for evaluating the effectiveness of a policy. Hand coded policies and genetic algorithms have been shown to produce mean scores above half a million, while cross-entropy methods are able to get around 350,000 [2]. As we wanted to implement some type of learning algorithm, we chose the cross-entropy method for the promising results seen in [2]. Additionally, we explored using a policy gradient method (Reinforce), but had poorer results than those using the cross-entropy method.

3 Cross-Entropy Method

The cross entropy method is a black-box stochastic optimization method that tries to optimize a set of parameters to maximize a reward function. For tetris specifically, the reward that needs to be maximized is the score (the number of lines cleared). We implemented an algorithm based upon the one in [2]. We selected a set of 8 features, as listed below, and used the max of the linear combination of the features with our weight vector to select the next action [1].

Features
Tetromino Landing Height
Eroded Pieces
Row Transitions
Column Transitions
Number of Holes
Cumulative Wells
Hole Depth
Rows With Holes

At every step of the algorithm, we initialized a set of 75 weight vectors by sampling the distribution found at the end of the previous iteration. For each of these weight vectors, 10 simulations were run, and the final scores were averaged to obtain a total score. This averaging was done to reduce the effect of stochasticity on the overall algorithm, so that a single very good or very bad run could not affect it drastically. Additionally, these multiple independent runs were parallelized to take advantage of a multi-core architecture. Of these 75 final total scores, we selected the top 10, and then fitted a Gaussian distribution to the corresponding feature vectors. During the next iteration, new feature vectors would be sampled from this distribution. Additionally, to prevent early convergence to a single point, we added noise to the standard deviations of the distribution; this noise was decreasing over time [2].

```
Cross-Entropy
Initialize Gaussian distribution: $(\mu, \sigma^2) = (0, 100)$
while $(\mu, \sigma^2)$ does not converge do
    Sample 75 weight vectors from the current distribution
    Run each sample 10 times and average score to account for stochasticity
    Select top 10 final scores
    Refit Gaussian distribution with the top 10 weight vectors
    Add time-decreasing noise to the new variance
end
return $(\mu, \sigma^2)$
```

Algorithm 1: Cross entropy algorithm used

We originally attempted to use the 22 features from [2], but the results improved drastically when we switched to using only the 8 features from [1]. This may be due to the fact that the chosen feature-set can more accurately describe the problem.

4 Reinforce

We also implemented Reinforce, a policy gradient method that takes advantage of the structure of the policy during optimization. We implemented the iterative version of the gradient descent algorithm (GDA) seen in class. We used the Boltzmann distribution as the assumption for the conditional probability for the policy.

$$\pi(a|s) = \frac{exp(\theta^T f(s,a))}{\sum_{a'} exp(\theta^T f(s,a'))}$$

The iterative algorithm steps are:

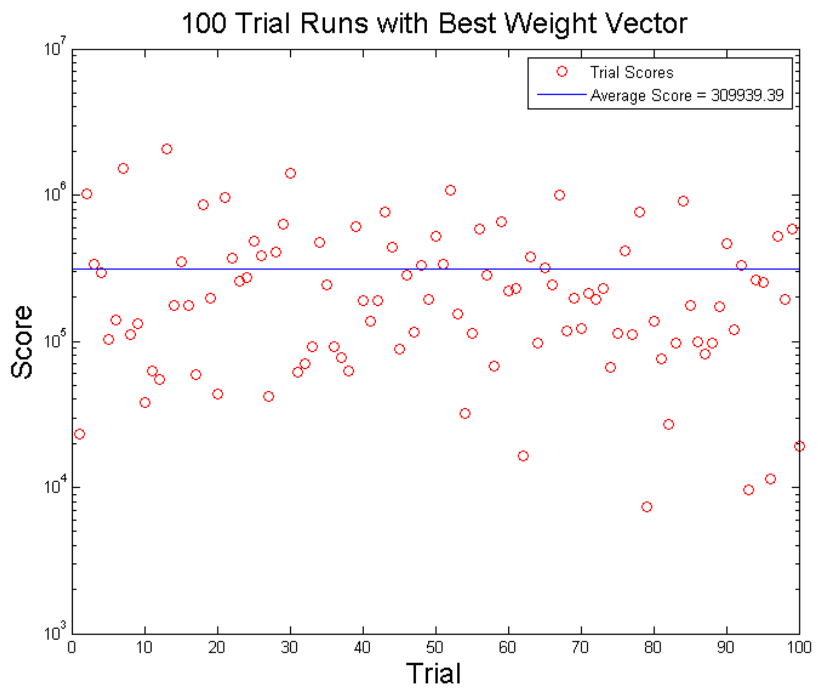
$$\begin{cases} z_{t+1} = \gamma z_t + \frac{\nabla \pi(a_t|s_t)}{\pi(a_t|s_t)} \\ \\ \Delta_{t+1} = \Delta_t + \frac{1}{t+1}(r_{t+1} z_{t+1} - \Delta_t) \end{cases}$$

Above, z_t is the discounted estimate of the gradient of the log-likelihood of the policy at time t and Δ_t is the estimate of ∇J for the gradient descent algorithm at time t. For the feature vector, we used the 22 features from [2]. We implemented the algorithm with a few different approach variations. We started by selecting the next action by sampling one action from the current policy distribution. Another approach was to select the action with maximum likelihood based on the current policy distribution. We also experimented with different discount factors, learning rates for the GDA (including time-decreasing learning rates for better convergence).

The best results were obtained for the algorithm where the action was selected by maximising the log-likelihood of the current distribution and by using a time-decreasing learning rate for GDA. Nevertheless, the results were significantly worse than the ones obtained by the cross-entropy method. Our best scores (number of cleared lines) were on the order of a few thousand lines.

5 Results

We ran multiple trials of the cross-entropy method, as some trials got stuck in local minima. Additionally, we varied the number of times a single sample was run, in an effort to obtain a tradeoff between reliability and running time. Our best training run resulted in an average score of $\boxed{309{,}939.39}$ over a hundred trial runs. We had a maximum score of 2,047,679 and a minimum score of 7435. Our final weight vector is shown below:



Final Weight Vector:

$$\begin{bmatrix}-47.4343 & 26.1263 & -57.652 & -109.3106 & -229.7158 & -68.4304 & -10.0172 & -159.8211\end{bmatrix}$$

6 Improvements

We saw a definite improvement when we switched features to the set of 8, as well as when we parallelized the concurrent sample runs. However, there may be additional features that we could implement to improve the performance of cross-entropy. Additionally, a Gaussian distribution is probably too much of a simplification for how the optimal weight vectors are distributed. As the problem is stochastic, something like a mixture of Gaussian distributions may have led to better results, and prevented the algorithm from getting stuck in low local minima. Cross entropy also has a number of tunable parameters, such as the number of samples drawn per iteration, the number of samples kept for forming the new distribution, as well as the noise added on to keep the algorithm from converging too fast. We did not attempt to tune these parameters, as optimization of them would have resulted in very, very long running times. Such optimizations may lead to further improvement if carried out, however. Increasing the number of samples, in addition to the number of trials run per sample, would have both increased the chance of converging to a higher score, as well as increasing reliability in the face of the stochastic problem.

For the reinforce algorithm, a major improvement that we could have implemented would have been to estimate the gradient using many sample runs, instead of just a few, due to the fact that the simulations are fairly noisy. However, since cross-entropy was working much better than Reinforce, we decided to optimize what we could for the cross-entropy algorithm.

References

[1] Amine Boumaza. How to design good tetris players. 2014.

[2] Andras Lorincz Istvan Szita. Learning tetris using the noisy cross-entropy method. *Neural Computation*, 18:2936–2941, 2006.