

CS 533 Final Project: Learning to Play Tetris

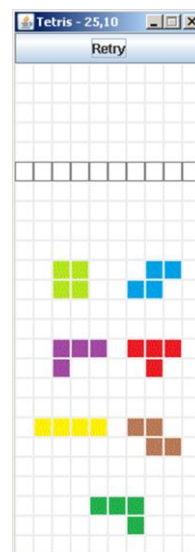
Using TD-Q Learning with Linear Approximations

Shubhomoy Das
(dassh@eecs.oregonstate.edu)
Oregon State University

Problem Definition

Tetris has the following rules and objectives:

- 20x10 board
- Seven blocks – one block at a time falling one row per time step
- **Actions:** Move Left/Right, Rotate Clockwise/Anti-Clockwise, Drop
- Game up once the highest row is reached
- **Goal:** complete as many rows as possible
- Completed rows get removed to create more room at the top
- Total **state space** at least $\sim 2^{200}$



Obviously, considering the state space, it is impractical to save properties of each state. Other researchers have tried to tackle this problem by reducing the state space using various reduction strategies or by simplifying the shape of blocks. In this project, we develop an agent that learns to play the full version of the game using one of the standard reinforcement learning techniques – Q-learning with TD updates. This approach has been shown to yield reasonable results (Farias, Van Roy 2006).

Q-Learning with Linear Approximation

The standard Q-Learning algorithm with linear function approximation is defined as (Fern A. 2011)

$$\hat{Q}_{\theta}(s, a) = \theta_0 + \theta_1 f_1(s, a) + \theta_2 f_2(s, a) + \dots + \theta_n f_n(s, a)$$

$$\theta_i \leftarrow \theta_i + \alpha \left(R(s) + \beta \max_{a'} \hat{Q}_{\theta}(s', a') - \hat{Q}_{\theta}(s, a) \right) f_i(s, a)$$

The θ_i values are feature weights for the features $f_i(s, a)$. These weights need to be learnt dynamically by exploration and TD-updates. This involves simulating the future allowable actions before deciding on one based on the explore/exploit policy. We use GLIE-I (epsilon) exploration strategy. One of the most crucial decisions is regarding the definition of features. We look at this next.

Feature Definition

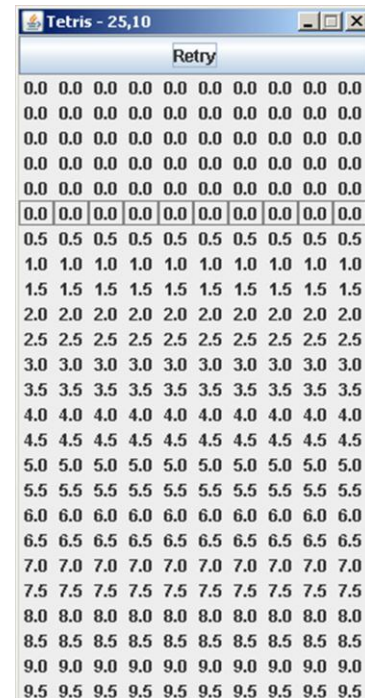
Some good suggestions for features come from (Farias, Van Roy 2006). We use 33 features some of which are similar to theirs but not the same.

- One bias term – Always 1
- Depth of topmost occupied cell in each column starting from the current block's present location (10 features)
- Difference between two successive depths computed above (9 features)
- Current block's row and column (2 features)
- Block type – only one among seven features turned on (1) corresponding to the block type (7 features)
- Block Orientation – only one among four features turned on corresponding to the rotation angle (4 features)
- All feature vectors are normalized so that the Q-values do so get so high as to become untenable

Rewards

We setup rewards for each board position to encourage the agent to gravitate towards the more favorable states. The cell rewards that act as the basis for all reward computations are shown in the figure on the right. The intuition behind this setup was to help the agent to:

- go to lower states
- look for good fit
- complete rows whenever possible
- penalize on reaching the highest row in proportion to the number of unoccupied cells

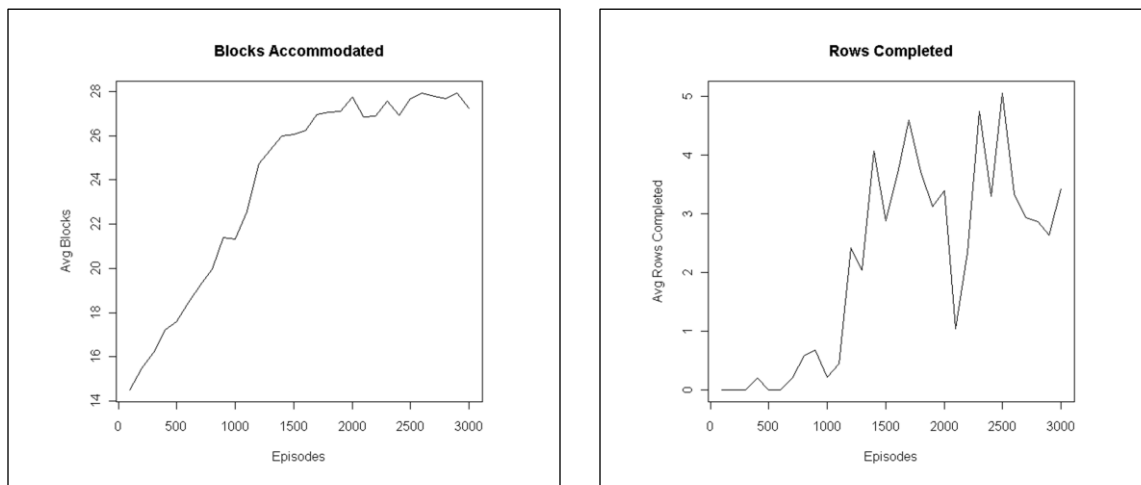


| | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 |
| 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2.5 | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 |
| 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 |
| 3.5 | 3.5 | 3.5 | 3.5 | 3.5 | 3.5 | 3.5 | 3.5 | 3.5 | 3.5 |
| 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 |
| 4.5 | 4.5 | 4.5 | 4.5 | 4.5 | 4.5 | 4.5 | 4.5 | 4.5 | 4.5 |
| 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 |
| 5.5 | 5.5 | 5.5 | 5.5 | 5.5 | 5.5 | 5.5 | 5.5 | 5.5 | 5.5 |
| 6.0 | 6.0 | 6.0 | 6.0 | 6.0 | 6.0 | 6.0 | 6.0 | 6.0 | 6.0 |
| 6.5 | 6.5 | 6.5 | 6.5 | 6.5 | 6.5 | 6.5 | 6.5 | 6.5 | 6.5 |
| 7.0 | 7.0 | 7.0 | 7.0 | 7.0 | 7.0 | 7.0 | 7.0 | 7.0 | 7.0 |
| 7.5 | 7.5 | 7.5 | 7.5 | 7.5 | 7.5 | 7.5 | 7.5 | 7.5 | 7.5 |
| 8.0 | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 |
| 8.5 | 8.5 | 8.5 | 8.5 | 8.5 | 8.5 | 8.5 | 8.5 | 8.5 | 8.5 |
| 9.0 | 9.0 | 9.0 | 9.0 | 9.0 | 9.0 | 9.0 | 9.0 | 9.0 | 9.0 |
| 9.5 | 9.5 | 9.5 | 9.5 | 9.5 | 9.5 | 9.5 | 9.5 | 9.5 | 9.5 |

Results

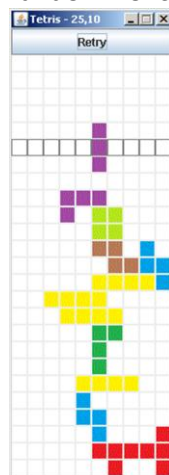
The standard criteria for measuring Tetris performance in machine learning literature is the number of rows completed in an episode. However, at this point the features, rewards and the learning & decay parameters need to be tuned further to achieve any competent level of expertise. We instead compare the average number of blocks accommodated on the board per episode after training to that achieved by a random policy. We find that while the random policy accommodates only 12 blocks, our TD-Q learning agent does 27. The random policy never completes any row whereas the TD-Q learner averages 2-4 per episode after training. Below is the graph for both the number of blocks and rows completed.

The results are very sensitive to the learning rate and decay parameters. In case the learning rate is too high, it leads to a greedy policy that is extremely bad.

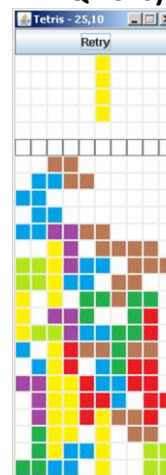


Typical Screen shots for Random and Trained policies towards the end of one episode are shown below.

Random Policy



TD-Q Policy



Conclusion

Q-learning with function approximation works quite well for large state-space problems like Tetris. It is quite sensitive to the decay and learning rate parameters. Since a lot of state spaces need to be visited and some might be bad, we need to keep the learning rate very low (we used 0.0001 for our case) else the algorithm tends to become greedy.

References

Fern., A, 2011. Fall 2011 Lecture Notes: Reinforcement Learning for Large State Spaces

Farias., V., and Van Roy, B., 2006. Tetris: A Study of Randomized Constraint Sampling