Learning about reinforcement learning, with Tetris

For our final assignment for the NUS Introduction to Artificial Intelligence class (CS3243), we were asked to design a Tetris playing agent. The goal of the assignment was to get students to be familiar with the idea of heuristics and how they work, getting them to manually tune features to get a reasonably intelligent agent. However, the professor included this in the assignment folder, which made me think we had to implement the Least-squares Policy Iteration algorithm for the task.

I'll probably discuss LSPI in more detail in another post, but for now, here are the useful features we found for anyone trying to do the same thing.

Features

In our Tetris agent implementation, we included features used by Pierre Dellacherie (found on Colin Fahey's site), Lagoudakis et. al. (2002) and some features included in the handout using Least-Squares Policy Iteration (LSPI) method.

1. `DIFF_ROWS_COMPLETED`

Number rows completed by taking action a.

2. `AVG_HEIGHT, DIFF_AVG_HEIGHT`

   Average height and difference of all columns after taking action a
3. `MAX_MIN_DIFF`

   Difference between highest and lowest column after action a.
4. `SUM_ADJ_DIFF,SUM_ADJ_DIFF_SQUARED`

   Sum of differences between adjacent columns (squared)
5. `COVERED_GAPS,DIFF_COVERED_GAPS`

   Number of covered gap in the Tetris board.
6. `TOTAL_BLOCKS`

   Number of cells occupied by blocks.
7. `TOTAL_WELL_DEPTH,MAX_WELL_DEPTH,WEIGHTED_WELL_DEPTH`

   Measurement for wells that appears on the board.
8. `COL_STD_DEV`

   Standard deviation for column heights
9. `COL_TRANS,ROW_TRANS`

   Calculate the total transitions across every row and every column after action a. Transitions are the number of times there is a change from a filled cell to an empty cell. This is done row-wise and length-wise.
10. `LANDING_HEIGHT`

    The estimated height at which the current piece will drop.
11. `CENTER_DEV`

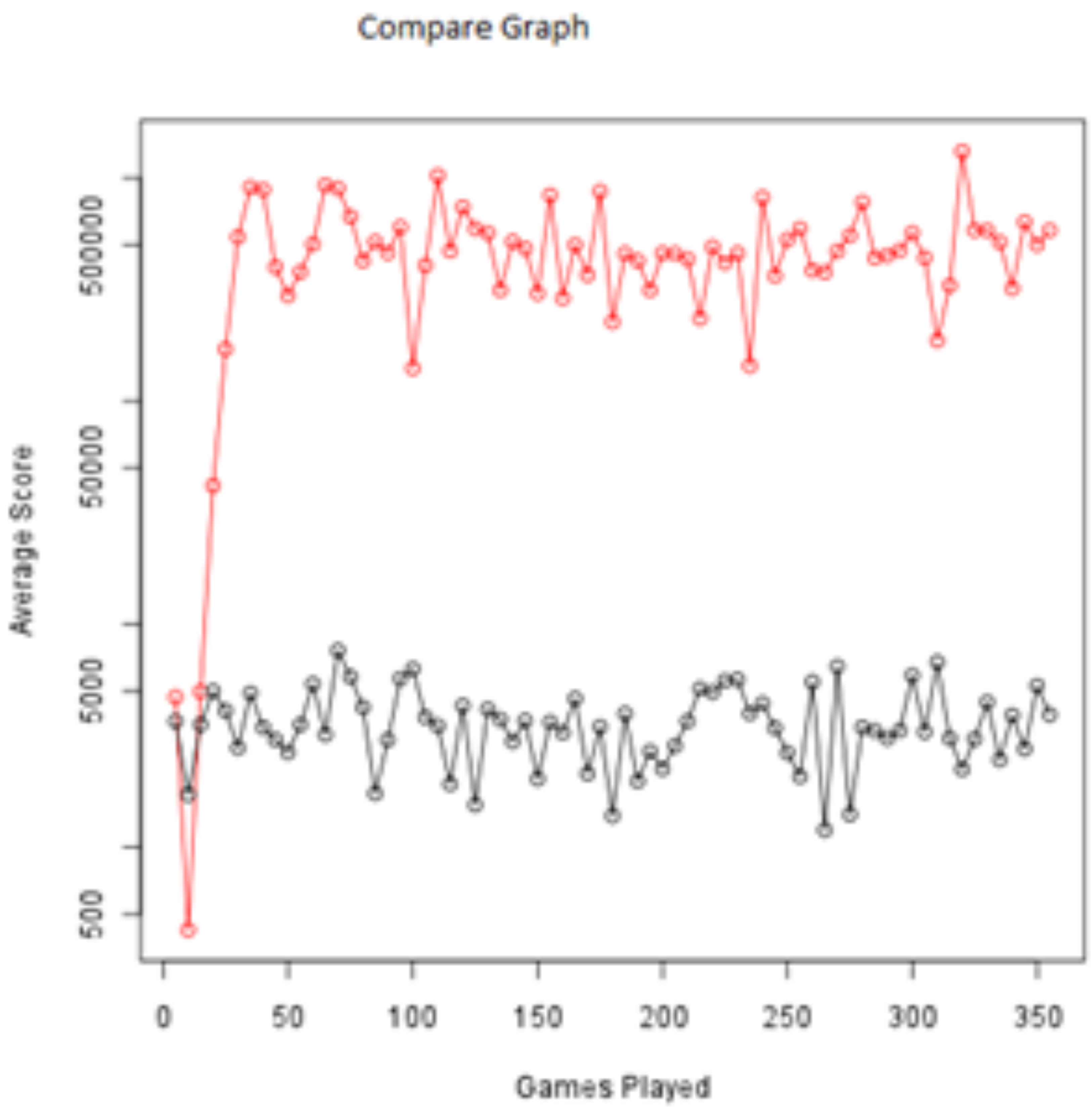
    Distance of landed piece from the center of the board.
12. `ERODED_PIECE_CELLS,WEIGHTED_ERODED_PIECE_CELLS`

    Number of cells removed that belonged to the most recently dropped piece after action a.

We achieved an average of about 500,000 rows completed, with a maximum of 3.8 million rows with weights using LSPI, and hand-coded weights by Pierre Dellacherie as a starting point.

Experimental Results & Analysis

We made our AI agent play the game 350 times. The graph below shows the number of games played against rows completed (y-axis uses logarithm scale). The red line depicts the learning agent while the black line shows the hand-coded features. The hand-coded features averaged at around 5000 lines.



The trend of the red graph shows an average improvement of performance by the learning agent . This was expected as the learning agent's weights get better with the amount of games played. The graph also shows that the aggregate performance of the learning agent is far better than the hand-coded agent. An important reason for this is that our learning agent assigns a weight to a particular feature after taking into consideration what other features are available; it might assign a higher weight to a feature x in the presence of a feature y because both features complement each other. For example, `AVG_HEIGHT` and `DIFF_AVG_HEIGHT` are features that complement each other. With both features present, the weights assigned by learning agents were 2 and 19 respectively. After `AVG_HEIGHT` was removed, `DIFF_AVG_HEIGHT` was assigned a higher weight of 22 as it was the only feature left to account for the average height. For a hand-coded agent, we can not adjust the hand-coded weights for complementary features as accurately as the learning agent.

Feature evaluation

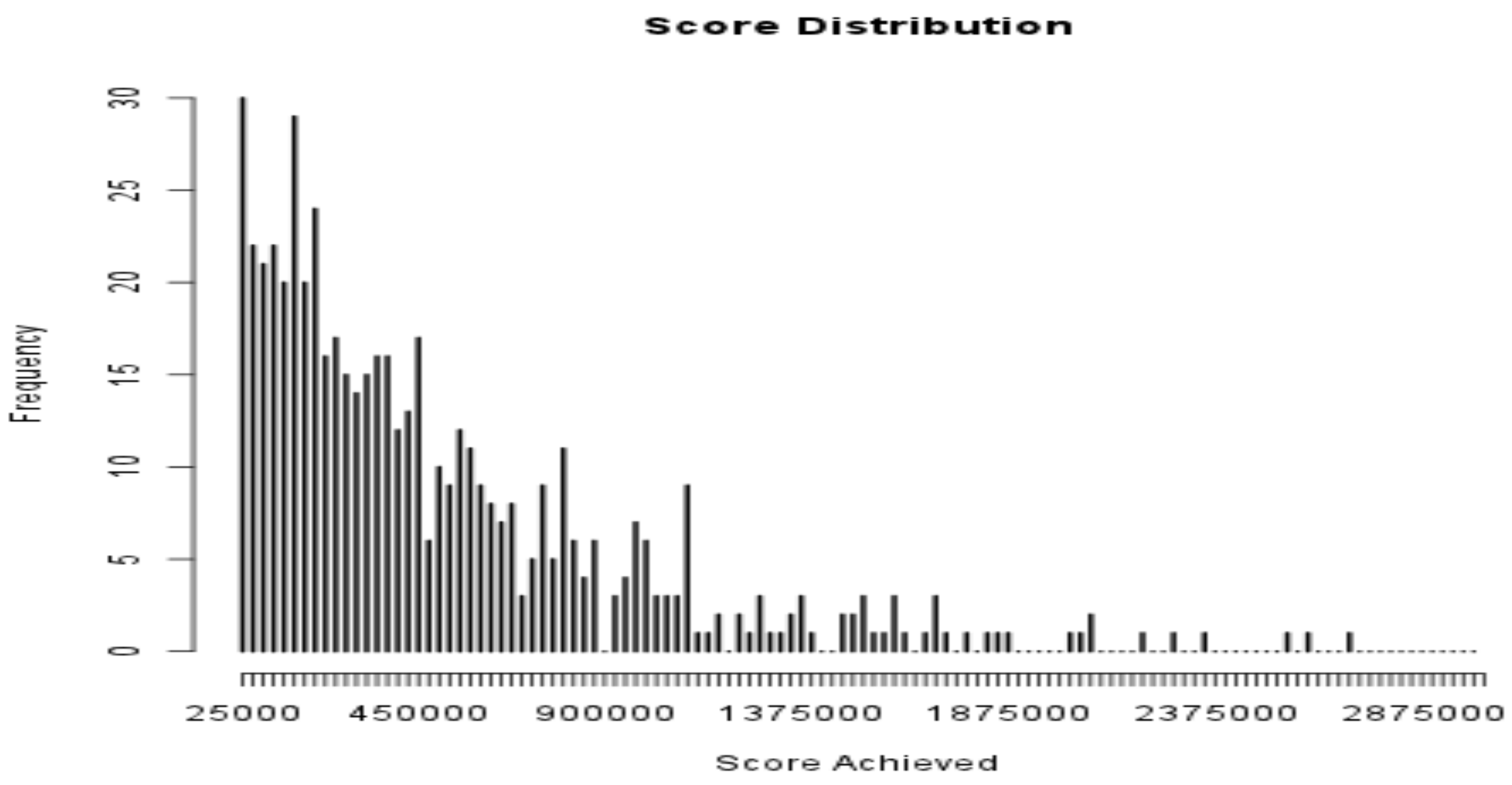
For certain features, the magnitudes of certain weights were smaller than 0.001

So we ran 2 versions of the agent, one before, and the other after the removal of these features, for 10 games, and computed the average score for each version. After removing these features we attained a higher average score. Before removing, our average was 531931.70 after the removal, it was 564613.38. This suggests that these features were indeed unnecessary and could be dropped. For similar reasons, we also removed `SUM_ADJ_DIFF`, `CENTER_DEV`, `ERODED_PIECE_CELLS`, `WEIGHTED_ERODED_PIECE_CELLS`.

We tried implementing some features which were not in the related works and these yield higher number of rows completed. For example, applying `SUM_ADJ_DIFF_SQUARED` resulted in higher number of completed rows when compared with using `SUM_ADJ_DIFF`. We conclude that the change in feature makes our agent more sensitive to height differences as each height difference value is amplified by squaring it. For instance, the difference between [latex]h_i=2[/latex] and [latex]h_j=1[/latex] compared to the difference between [latex]{h_i}^2=2^2=4[/latex] and [latex]{h_j}^2=1[/latex]. This encourages the agent to play in the way such that the top surface is smoother, which implies higher frequency of row completion.

Similarly, `COL_STD_DEV` encourages smooth play by keeping deviation of column heights small. Intuitively it means an agent would tend to evenly distribute the blocks instead of stacking them at one side of the game board.

The most important features were definitely `DIFF_ROWS_COMPLETED` and `DIFF_AVG_HEIGHT` as the weights assigned to them were very high; 20 for `DIFF_ROWS_COMPLETED` and 19 for `DIFF_AVG_HEIGHT`. If we removed these features, then the performance dropped to an average of 1000 lines.



Our Tetris agent gives a highest performance of 3.8 million rows completed and an average of around

500,000 rows completed. A distribution of our agent's performance over 475 games is seen in the adjacent diagram.

The Java code is here.

Also read...

86 Neural Turing Machines – Copy Task

46 Neural Turing Machines – A First Look

46 Neural Turing Machines – Implementation Hell

39 Recursive Auto-encoders: An Introduction

39 Recursive Auto-encoders: Example in Theano

Leave a Reply

Comment

Post Comment

← READ MORE