Implementation Details

Equations to implement for temporal different learning for tetris. The equations are for a linear feature-based approximation architecture using approximate and optimistic λ-policy iteration.

Variable:

w : wall width

r : A feature vector of (2w + 2) entries. The first entry is a constant. (2w + 1) entries are values of features listed below.

i : A board state in the game

Features:

h_k : The height of the kth column of the wall, k is from 1 to w

$|h_k - h_{k+1}|$: The absolute different between the heights of the kth and the (k + 1) column

$max_k h_k$: The maximum wall height

L : The number of holes in the wall

Utility function:

$$J(i, r) = r(0) + \sum_{k=1}^{w} r(k)h_k + \sum_{k=1}^{w-1} r(k+2)|h_k - h_{k+1}| + r(2w)max_k h_k + r(2w+1)L$$

Vector r can be assigned with random values first. For each iteration of the algorithm we play M (in the order of 100) games. Suppose after t iterations the weights vector is r_t. We can find r_{t+1} by the following equation:

$$r_{t+1} = argmin_r \sum_{m=1}^{M} \sum_{k=0}^{N_m} \left(J(i_{m,k}, r) - J(i_{m,k}, r_t) - \sum_{s=k}^{N_m - 1} \lambda^{s-k} d(i_{m,s}, i_{m,s+1}) \right)^2$$

where
$(i_{m,0}, i_{m,1}, ..., i_{m,N_m - 1}, i_{m,N_m})$ is the sequence of board states comprising the mth game in the iteration and i_{m,N_m} is the terminal state

$d(i_{m,s}, i_{m,s+1}) = g(i_{m,s}, \mu_t(i_{m,s}), i_{m,s+1}) + J(i_{m,s+1}, r_t) - J(i_{m,s}, r_t)$ is the temporal difference

where $g(i_{m,s}, \mu_t(i_{m,s}), i_{m,s+1})$ is the number of rows clear of going from board state $i_{m,s}$ to board state $i_{m,s+1}$ using action $\mu_t(i_{m,s})$
$J(i_{m,N_m}, r_t) = 0$
λ is a chosen learning factor
μ_t is the policy function

The vector *r* in the equation above can be found my using a number of methods: stochastic gradient descent which is great for parallel and distributed platforms but is only available in libraries as part of a supervised learning routine. Other optimization methods are readily available in libraries: Nelder-Mead Simplex algorithm, Broyden-Fletcher-Goldfarb-Shanno algorithm, Newton-Conjugate-Gradient algorithm.

For some algorithms, the matrix form and the derivative of the above equations are also required.

<u>Interface Between Player and Learner:</u>

The Player java program accepts as input (as command line argument) a maximum number of games to play and a string of 22 space separated floating point numbers corresponding to the 22 entries of the weight vector.

e.g. *java PlayerSkeleton 100 “1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22”*

The java player then plays the specified number of games unless the total board states has reached more than ***2000*** in which case the player will terminate and discard data from any incompleted game. The player writes the output data to the ***standard output*** under the following format:

- Different games are separated by a line consisting of the character '#'
- Each game consists of a series of rows separated by new line each row records data for a board state.
- Each row contains a space separated set of numbers starting with the constant 1 followed by 21 features value with the same order as in the features list mentioned above followed by the reward (number of rows clear) from moving from the previous state to the current state reflected by the current row. (thus the reward of the first row/state is 0).

e.g. 2 games, each game has 3 states, total number of states is 6

```
1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 0
1 0 18 13 11 1 16 5 20 18 20 19 2 16 19 17 8 2 6 19 9 12 2
1 9 5 3 12 1 3 21 0 11 14 6 10 18 10 5 2 6 1 3 3 1 1
#
1 10 2 4 17 21 6 7 11 6 15 11 11 6 1 6 4 13 11 16 10 10 0
1 13 15 4 10 14 13 13 6 9 0 4 8 9 7 3 11 17 10 12 8 18 4
1 7 18 10 3 11 1 0 8 17 0 4 17 13 0 17 6 16 8 0 16 8 0
```