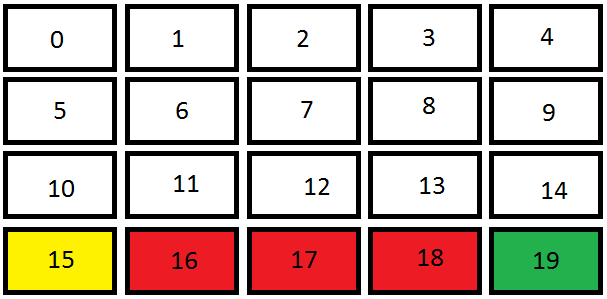
Report of implementing Q-Learning in simple game: cliff climber

Problem: control the player walk from 1 point passing the cliff and go to the destination point, avoiding to drop to the cliff.

Solution 1: e-greedy algorithm with Q-learning method is used as RL method, with epsilon value of 0.1, decay factor of 0.9 and learning rate of 0.01.

The reward is that, -100 if it falls to cliff, -1 for each movement, 100 if it gets destination.

Intuitive model is this:



Yellow: player start position

Red: cliff

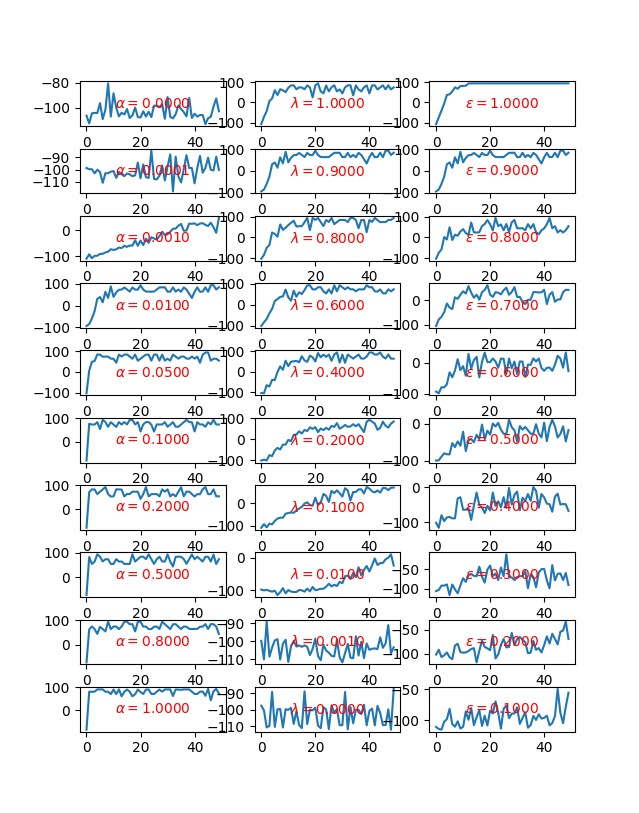
Green: destination

Possible actions are go up, down, left and right.

States are represented by the number of players’ position, 100 stands for terminal state.

The training was done in 1,000 games.

Q-learning Result by change parameters:



Default: , where is the chance to choose a random action

Solution 2: TD-gammon with 20 input nodes indicating the position of the player, 5 hidden nodes and 1 output nodes.

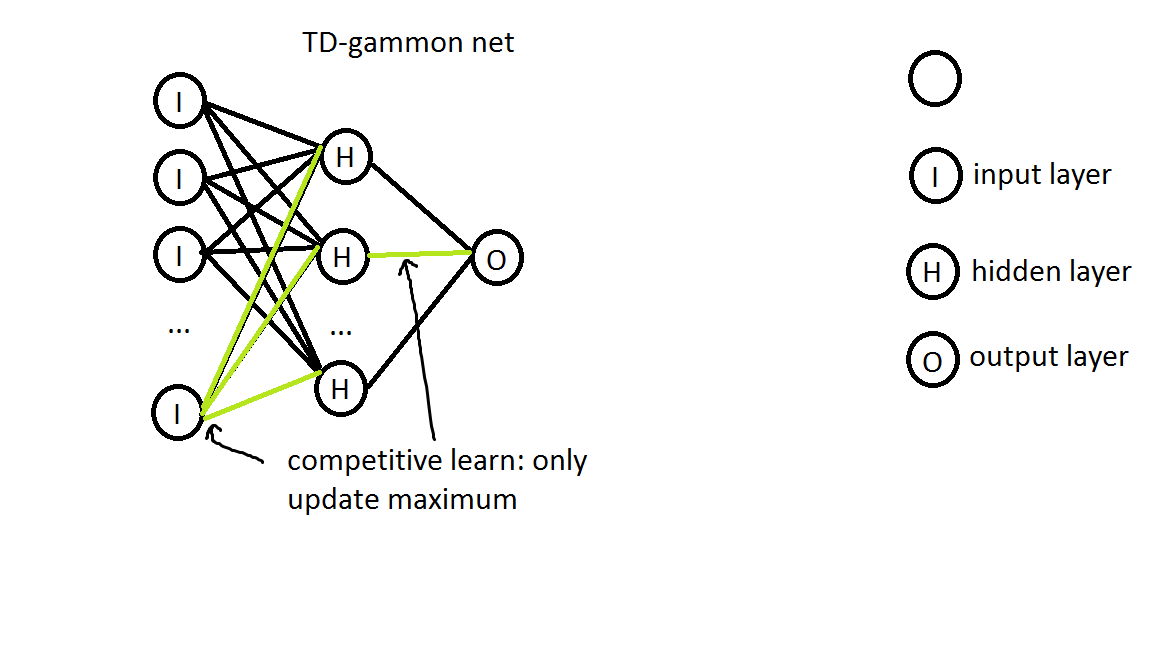
Hidden nodes and output nodes calculation method:

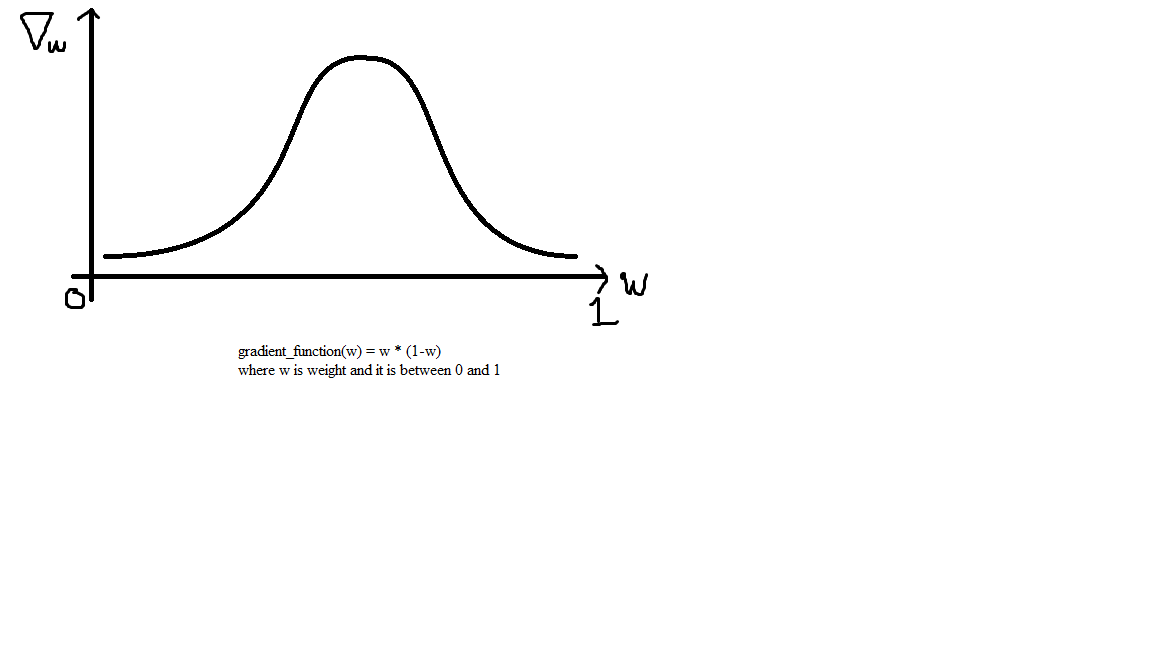
where is the value of i-th input, is the weight of its connection to j-th hidden unit.

Weights update rule is:

Where is the vector of eligibility trace corresponding to the weight vectors; and it is calculated as:

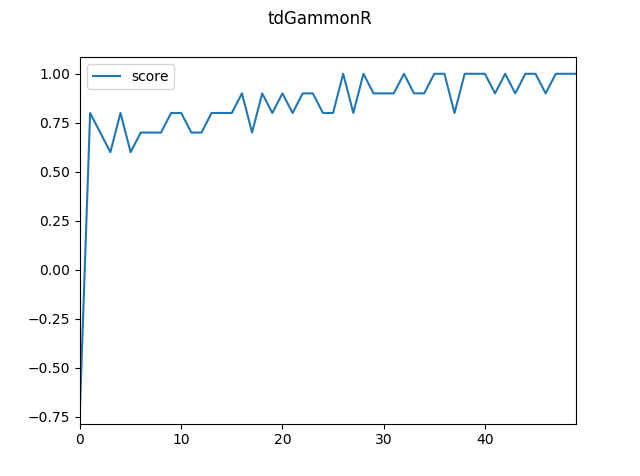
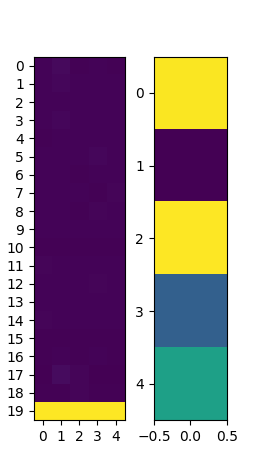
Reward structure is -1 if fall off the cliff, 1 if go to the destination

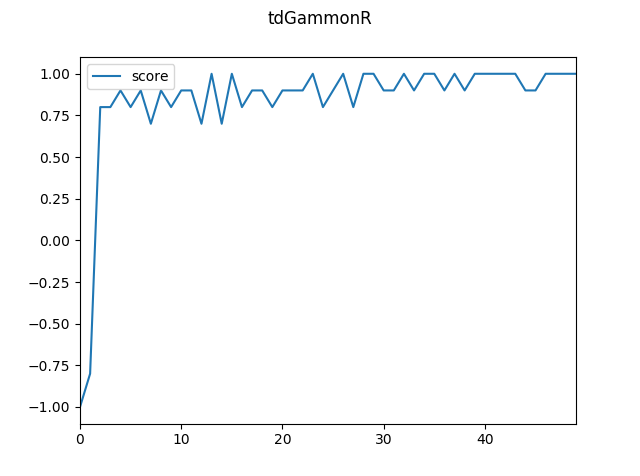
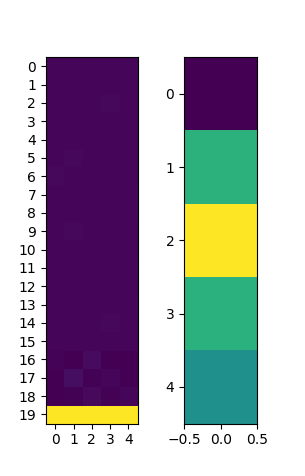


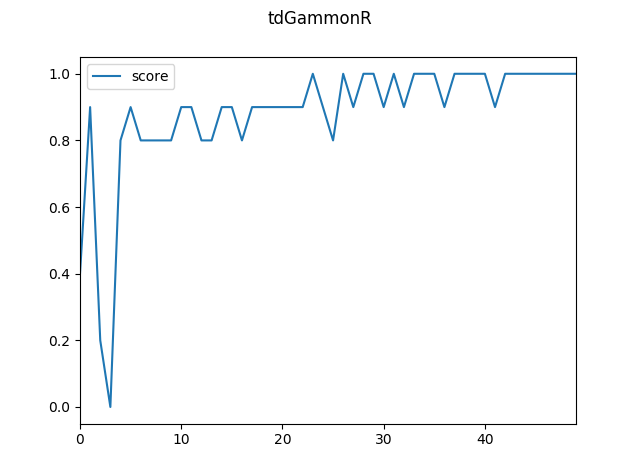
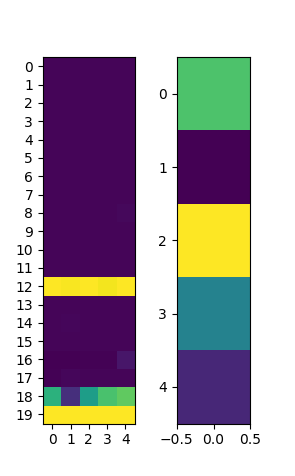


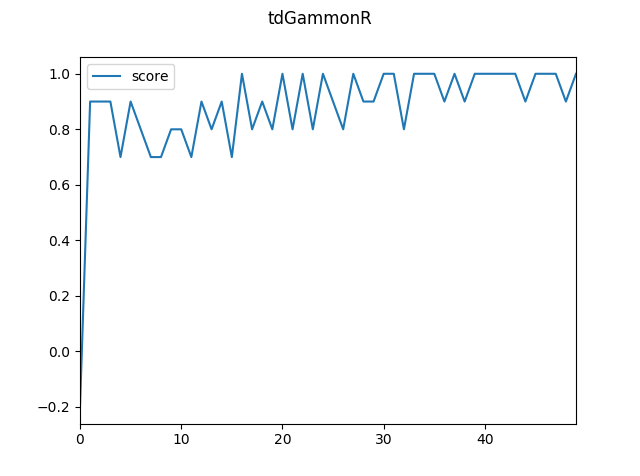
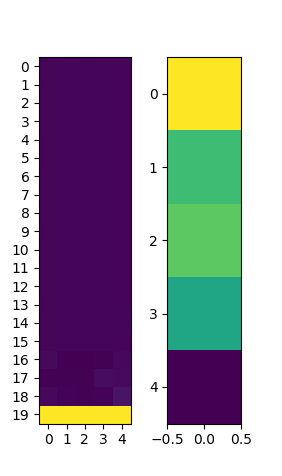
A model is used to simulate the game (exactly the game logic)

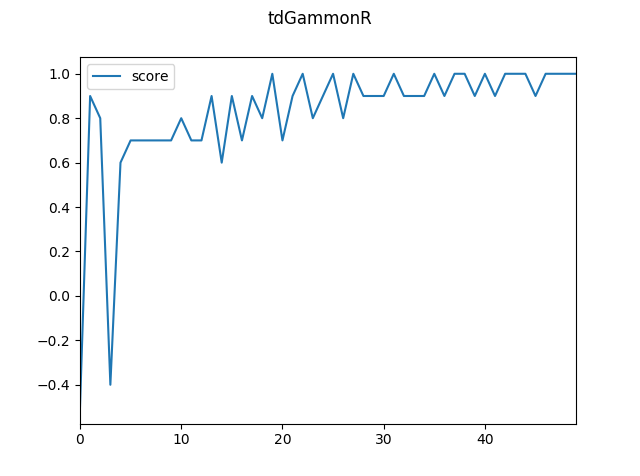
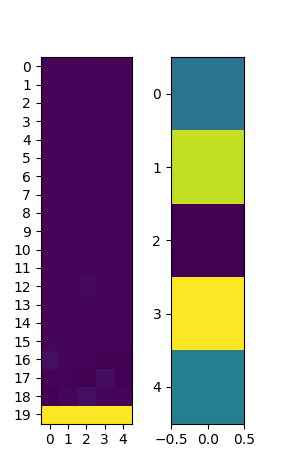
Result:



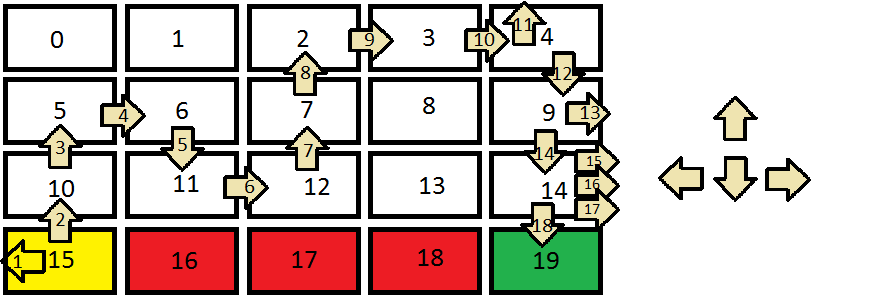




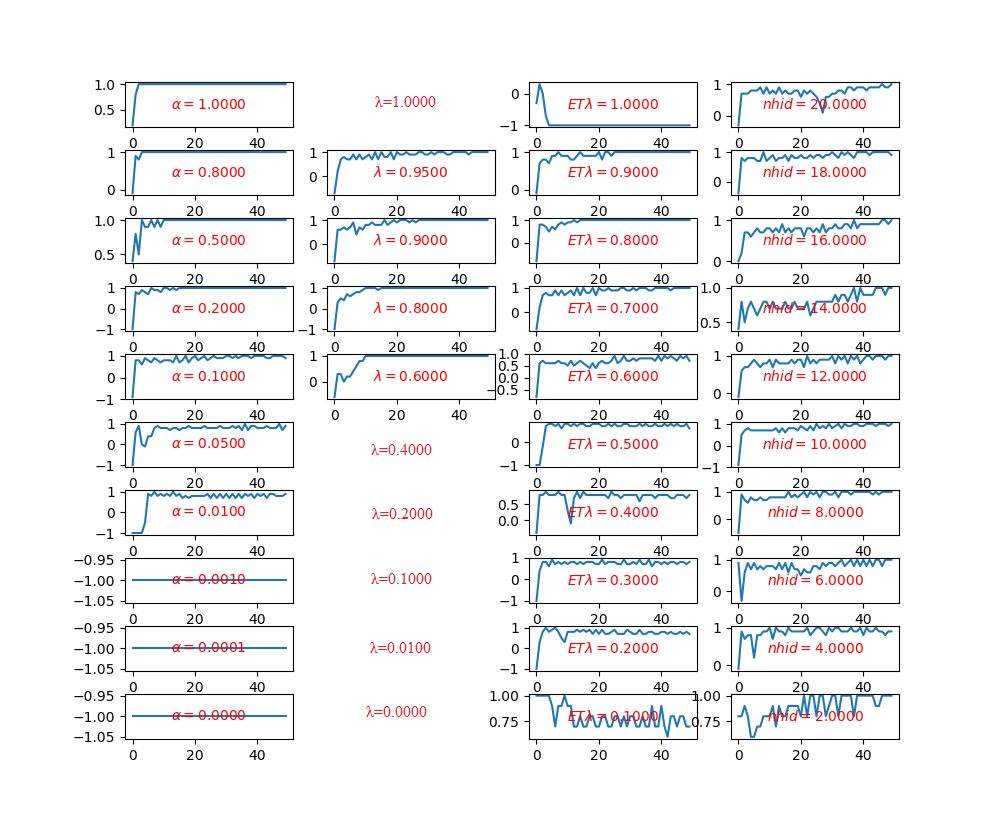




Example step log:



TD result by change params:



Where empty figure represents the program with that params runs forever.