Playing Tic Tac Toe using SARSA and Q-Learning

Vismantas Dilys

Project contains the following file:

- deepTic.py: main module, containing the algorithm
- play.py: script for a human to play against a compute opponent, which can either be trained by the script, or a file, containing pre-trained state-action values, can be specified when running the script with "—brain" parameter
- deepTicTests.py: unit tests for deepTic.py
- experiments.py: script that runs various experiments and outputs the results in the expResults folder.
- drawGraphs.py: script that draws the graphs used in this report based on data in expResults folder.
- expResults: folder contains the results of the experiments, produced by experiments.py file
- graphs: folder contains the images used in this report. Can be populated using drawGraphs.py
- brainy.brain: pre-trained state-action pair values. Can be used with 'play.py' script.

Main design decisions:

- The algorithm works by calculating state-action pair values.
- The state is represented as a tuple of 9 digits - each one represents a position of the board. 0 denotes an empty square, 1 is for 'X' and 2 is for 'O'.
- The reward for any action is 0 except for actions that lead to a terminal state. In a case when the action leads to a terminal state, agent receives 1 for winning, -1 for loosing and 0 for a tie.
- Terminal states have 0 values.
- As the game always terminates in a finite number of steps (at most 9), no discounting of future rewards is used.
- Agents use ε-greedy strategy when training.
- Code to handle symmetries of the game has been implements - agents can be switched to take advantage of this, or not. This is controlled by setting 'useSymmetry' attribute.
- When playing against agents, they can be switched to pure greedy strategy (by setting attribute 'competitionMode' to True.

Experiments

I carried out a few experiments, to compare the performance of SARSA and Q-learning.
Here is the setup of the experiments:

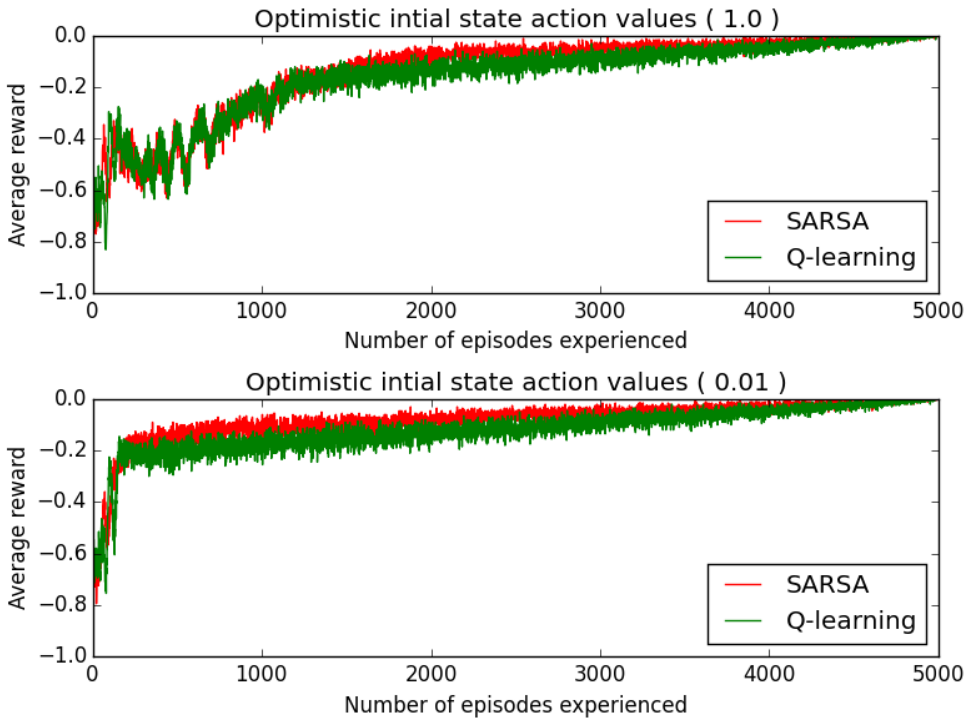
- A 'canonical' agent has been trained using SARSA and playing against it self for 2 million episodes, to make sure that it learns the best strategy.
- For each experiment an appropriate new agent instance with different parameters was instantiated. This agent would play against the canonical agent. For each experiment I averaged out the results of 200 agents, each one of which was trained against the 'canonical' agent for 5000 episodes.
- The results of all 200 runs of different agents are then averaged out, to produce a vector of length 5000 that contains average return for that episode.
- The agents were using 0.2 learning rate.
- The ε parameter was set to 0.2 on the first episode and then reduced linearly ε * (episodeIndex / nOfEpisodes)

To compare the two algorithms I carried out the following experiments:

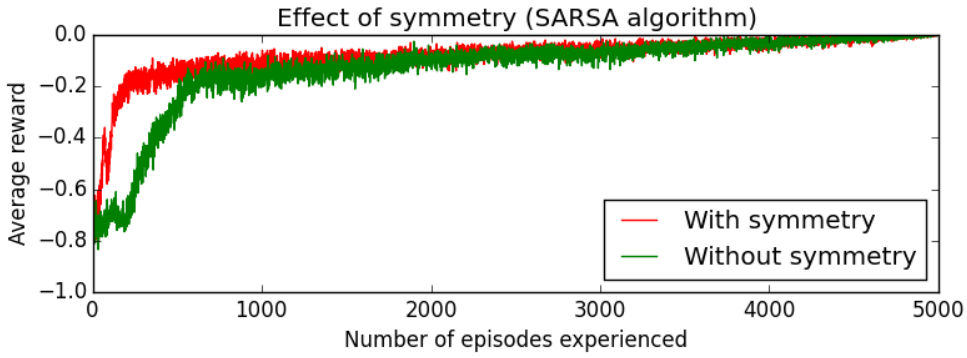
- SARSA algorithm trained with initial Q values set to 0.01 and using symmetry awareness.
- Q-learning algorithm trained with initial Q values set to 0.01 and using symmetry awareness.
- SARSA algorithm trained with initial Q values set to 1 and using symmetry awareness.
- Q-learning algorithm trained with initial Q values set to 1 and using symmetry awareness.
- SARSA algorithm trained with initial Q values set to 0.01 and not using symmetry awareness.
- Q-learning algorithm trained with initial Q values set to 0.01 and not using symmetry awareness.

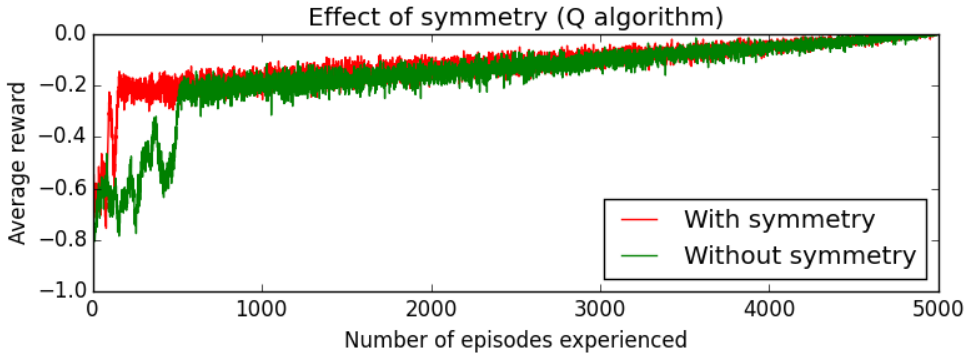
Analysis

The first graph shows the performance difference between SARSA and Q-learning on the tic tac toe game. It can be seen, that both algorithms perform pretty much the same. Setting different initial state-action pair values affects the amount of exploration, done by the agents. Setting the values too over-optimistic (to 1) slows down the learning of the agents, because while learning agains it self the agent learns optimal strategy, which always leads to a tie (reward 0), so it takes time to reduce the suboptimal state-action pair values to the true values. In the bottom part of the graph we see that agents learn faster because the state-action pair values are still initialised optimistically, but to much more reasonable values (0.01).
The similarity between SARSA and Q-learning in this case could be explained by the fact, that the game is very small, has small number of states and is deterministic (except the actions of the second player - but the opponent is also learning optimum strategy, so this becomes more deterministic as well).



The second graph show the effect of adding symmetry awareness to the algorithm. Tic tac toe game has 8 symmetries. By adding symmetry awareness, the total number of states, seen by the agent can be decreases 8 times. Also experienced learned in one state, would be immediately transferred to the other states. For these reasons we would expect an increased learning rate of the agents that are symmetry aware vs the ones that do not. The graphs show that the





expectations are correct and that both SARSA and Q-learning algorithms learn much quicker with symmetry awareness than without it.

It can be seen that the agents lear optimal strategy before the 1000th episode - after that we still see some errors (as for optimal strategy the returns should always be 0) but that is because of the ε parameter, and basically the graphs track the ε as it decreases with the number of episodes.