Assignment: Model-free reinforcement learning
Course: Reinforcement Learning, Leiden University
Written by: Daniël Pelt

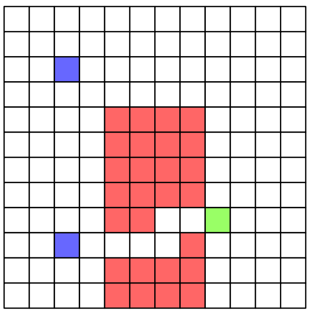
In this assignment, you will study three different model-free RL algorithms:

- Q-Learning
- SARSA
- Expected-SARSA

Your research question Your goal is to investigate these three algorithms. In particular, you will:

1. Implement these algorithms.
2. Investigate the effect of different parameters for each algorithm.
3. Compare the performance of the three methods.

The ShortCut environment We will study these algorithms on a custom environment we will call the ShortCut environment. The environment consists of a 12x12 grid, and the agent can move to adjacent squares. Actions that try to move the agent outside the 12x12 grid do not move the agent. The goal is to reach the goal square (green in the figure below). If the goal is reached, the episode ends. In each episode, the agent starts in one of two squares (blue in the figure), each with equal probability. Some squares (red in the figure) act as a 'cliff', similar to the Cliff Walking environment described on page 132 of Sutton & Barto: if the agent ends up in the cliff, it is returned to the starting position, with a reward of -100. All other actions (i.e., those that do not end up in the cliff) have a reward of -1. The RL algorithm should try to maximize the cumulative reward of each episode. An image of the ShortCut environment is shown below.



First carefully read the preparation instructions, and good luck!

Preparation

Python You need to install Python 3, the packages `Numpy`, `Matplotlib`, and `SciPy`, and an IDE of your choice.

Files You are provided with three Python files:

- `ShortCutEnvironment.py`: This file includes the ShortCut environments. Inspect the code and make sure you understand it.
- `ShortCutAgents.py`: This file contains placeholder classes for the three model-free RL algorithms you will implement: `QLearningAgent`, `SARSAAgent`, and `ExpectedSARSAAgent`. Currently, each agent randomly returns an action. Your goal is to implement the correct `init()`, `select_action()`, and `update()` methods for each class. Run the file and verify that they work for random action selection.
- `ShortCutExperiment.py`: In this file you will write your experiment code.

Handing in You need to hand in:

- A **report** (pdf) of **maximum 6 pages!!**. Be sure your report:
  - Describes your methods (include equations).
  - Shows results (figures).
  - Interprets your results.
  - Uses the following template: `https://www.overleaf.com/read/qyhkfdgvktkm#ef23c5`.
- All **code** to replicate your results. Your submission should contain:
  - The original `ShortCutEnvironment.py`
  - Your modified `ShortCutAgents.py`.
  - Your modified `ShortCutExperiment.py`, which upon execution should produce all your plots, and save these to the current folder.

Be sure to verify that your code runs from the command line, and does not give errors!

Warning: common errors (with statistical experiments).

- Average your results over repetitions (since each run is stochastic)!
- In each repetition, really start from scratch, i.e., initialize your agent from scratch.
- Do not fix any seeds within the loop over your repetitions!
- Average your curves over repetitions. If necessary, apply additional smoothing to your curves.

1 Q-learning

You first decide to study the Q-Learning algorithm for this problem. You proceed in four steps:

a Correctly complete the class `QLearningAgent()` in the file `ShortCutAgents.py`.

- Initialize the action values $Q(s, a)$ to 0.
- Implement an ϵ-greedy policy for selecting an action.
- Implement the Q-Learning update equation to update $Q(s, a)$ values after each action.

b Write a function `run_repetitions()` in `ShortCutExperiment.py`. Your function should repeatedly test the agent `QLearningAgent()` on an instance of `ShortcutEnvironment()`.

- Run a single experiment for `n_episodes=10000` episodes (you can use $\alpha = 0.1$ and $\epsilon = 0.1$ for now).
- Make a plot of the action with the maximum value for each state, and observe which path a greedy policy would take for each of the two possible starting positions. For the report, include an image of these paths. In general, any image that clearly shows the paths within the environment will be good. To help, here is some code to transform a Q-table to an image showing the greedy action for each state:

```
def print_greedy_actions(Q):
    greedy_actions = np.argmax(Q, 1).reshape((12,12))
    print_string = np.zeros((12, 12), dtype=str)
    print_string[greedy_actions==0] = '^'
    print_string[greedy_actions==1] = 'v'
    print_string[greedy_actions==2] = '<'
    print_string[greedy_actions==3] = '>'
    print_string[np.max(Q, 1).reshape((12, 12))==0] = '0'
    line_breaks = np.zeros((12,1), dtype=str)
    line_breaks[:] = '\n'
    print_string = np.hstack((print_string, line_breaks))
    print(print_string.tobytes().decode('utf-8'))
```

  One way of showing the paths in the report could simply be to make a screenshot of these greedy actions and draw the greedy paths starting from the initial positions over it using your favorite image editor.

- Run `n_rep=100` repetitions of a similar experiment, but with `n_episodes=1000` episodes. **Be sure to initialize a clean agent and environment instance for each repetition** (without fixing a seed)!
- Average the learning curves (cumulative reward of each episode) over your `n_rep=100` experiments and plot the result.

Experiment a bit with varying your hyperparameters: `epsilon`, `n_episodes`, and `alpha`. Get a feeling for how they affect your results.

c You decide to run a more structured experiment.

- You will test the following values for `alpha`: `[0.01, 0.1, 0.5, 0.9]`.

- Run your function from 1b for these different values of alpha, where you again average over `n_rep=100` repetitions of `n_episodes=1000` episodes.
- Plot the learning curves for each setting of alpha in a single graph. **Add a legend, and label the x and y-axis appropriately**.

d Write the first section of your report. Describe:

- Your methodology (with equations).
- Your results (learning curve graph and greedy paths).
- Interpret the results, give possible explanations.

2 SARSA

You decide to try another algorithm: SARSA. You follow the same scheme as for your previous experiments.

a Correctly complete the methods of `SARSAAgent()` in the file `ShortCutAgents.py`.

- Initialize the action values $Q(s, a)$ to 0.
- Implement an ϵ-greedy policy for selecting an action.
- Implement the SARSA update equation to update $Q(s, a)$ values after each action.

b Test your `SARSAAgent()` agent over repetitions, like in question 1b. You have two options:

- Write a completely new function (e.g., `run_repetitions_sarsa()`) in `ShortCutExperiment.py`.
- Modify your previous `run_repetitions()` to take an argument `agent_type`, to make it work for either `agent_type='qlearning'` or `agent_type='sarsa'`.

Again, first do a single experiment with `n_episodes=10000` episodes, and make a plot of the action with the maximum value for each state, and observe which path the greedy policy takes for each of the two starting positions. Compare this result with the result from QLearning, and explain any differences in your report. Run `n_rep=100` repetitions of a similar experiment, but with `n_episodes=1000` episodes, average the learning curves (cumulative reward of each episode) over your `n_rep=100` experiments and plot the result. Again, compare with the results from QLearning. Can you also explain any differences you observe with the results of the Cliff Walking experiment shown on page 132 of Sutton & Barto?

c Run a structured experiment like question 1c.

- You will test the following values for `alpha`: `[0.01, 0.1, 0.5, 0.9]`.
- Run your function from 1b for these different values of alpha, where you again average over `n_rep=100` repetitions of `n_episodes=1000` episodes.
- Plot the learning curves for each setting of alpha in a single graph. **Add a legend, and label the x and y-axis appropriately**.

d Write a results section in your report with:

- Your methodology (with equations).
- Your results (graphs and greedy paths).
- Interpret the results, give possible explanations. Be sure to compare with QLearning.

3 Stormy weather

Suddenly, the wind picks up! The environment changes to a WindyShortCutEnvironment. The setup is very similar to the original ShortCut environment, but it includes a stochastic wind: after each action, there is a 50% chance that the agent is moved to the square below.

a Add code to `ShortCutExperiment.py` that does the following:

- Run a single WindyShortCutEnvironment experiment with QLearning for `n_episodes=10000` episodes, using $\alpha = \epsilon = 0.1$.
- Make a plot of the action with the maximum value for each state, and observe which path the greedy policy takes for each of the two starting positions.
- Do the same with SARSA instead of QLearning. Observe any differences between the greedy policy for both methods, and the differences between these experiments and the similar experiments for the original ShortCut environment. Can you explain the difference between the results?

b Write a results section in your report with:

- Your results (greedy paths).
- Interpret the results, give possible explanations. Be sure to compare with similar earlier experiments. Can you predict what would happen if the wind would come from a different direction?

4 Expected SARSA

You decide to try another algorithm on the original ShortCut enviroment: Expected SARSA. You follow the same scheme as for your previous experiments.

a Correctly complete the methods of `ExpectedSARSAAgent()` in the file `ShortCutAgents.py`.

- Initialize the action values $Q(s, a)$ to 0.
- Implement an ϵ-greedy policy for selecting an action.
- Implement the expected SARSA update equation to update $Q(s, a)$ values after each action.

b Test your `ExpectedSARSAAgent()` agent over repetitions, like in question 1b. You have two options:

- Write a completely new function (e.g., `run_repetitions_expectedsarsa()`) in `ShortCutExperiment.py`.
- Modify your previous `run_repetitions()` to take an argument `agent_type`, to make it work for either `agent_type='qlearning'`, `agent_type='sarsa'`, or `agent_type='expectedsarsa'`.

Again, first do a single experiment with `n_episodes=10000` episodes, and make a plot of the action with the maximum value for each state, and observe which path the greedy policy takes for each of the two starting positions. Compare this result with the result from QLearning and SARSA, and explain any differences in your report. Verify that your code runs and produces reasonable curves, and play around with a few settings of `alpha`, `epsilon`, and `n_episodes`.

c Run a structured experiment like question 1c.

- You will test the following values for `alpha`: `[0.01, 0.1, 0.5, 0.9]`.
- Run your function from 1b for these different values of alpha, where you again average over `n_rep=100` repetitions of `n_episodes=1000` episodes.
- Plot the learning curves for each setting of alpha in a single graph. **Add a legend, and label the x and y-axis appropriately**.

d Write a results section in your report with:

- Your methodology (with equations).
- Your results (graphs).
- Interpret the results, give possible explanations. Be sure to compare with QLearning and SARSA.

5 Comparison and conclusions

Finish your report by comparing the three algorithms you implemented with each other. What are their strengths and weaknesses? Which parameter choices are optimal for each algorithm? Conclude with a short reflection: what did you learn about model-free RL algorithms?