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

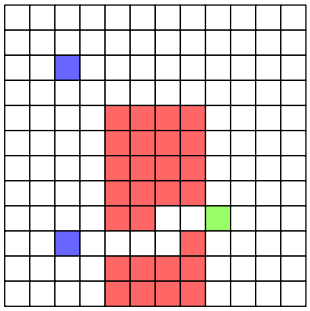
In this assignment, you will study four different model-free RL algorithms:

- Q-Learning
- SARSA
- Expected-SARSA
- n-step 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 quantitative and qualitative performance of the 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. In particular:
  - `reset()` resets the environment for a new episode
  - `state()` gives you the current state
  - `step(action)` lets you take an action and observe a reward
  - `done()` allows you to check whether the episode has terminated (so you should move to the next one).
  - `render()` allows you to visualize the current state of the environment.
  - `render_greedy(Q)` allows you to visualize the greedy policy based on a given $Q(s, a)$ array.

  You can run the file to see a demonstration of the rendering functions.
- `ShortCutAgents.py`: This file contains placeholder classes for the three model-free RL algorithms you will implement: `QLearningAgent`, `SARSAAgent`, `ExpectedSARSAAgent` and `nStepSARSAAgent`. Currently, each agent randomly returns an action. Your goal is to implement the correct `init()`, `select_action()`, and `update()` and `train()` methods for each class. All these methods have some starting arguments, but (especially for the `update()` method) you should sometimes augment these with variables that are required in that method.
- `ShortCutExperiment.py`: In this file you will write all your experiment code (running repetitions, smoothing, plotting, etc.). This file should produce the graphs in your report.

Handing in You need to hand in:

- A **report** (pdf) which:
  - Describes your methods (include equations).
  - Shows results (figures).
  - Interprets your results.
- 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 environment step (Eq. 6.8 from Sutton and Barto).
- Implement a full Q-learning training loop, where you return a vector with the obtained cumulative reward per episode.

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, there is a `render_greedy(Q)` method in the environment that prints the greedy policy for a given $Q(s, a)$ array for you.
  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 within the repetition loop)!
- 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. **Apply smoothing if appropriate. 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 environment step (Eq. 6.7 from Sutton and Barto).
- Implement a full training loop, where you return a vector with the obtained cumulative reward per episode.

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 Q-Learning, 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 Q-Learning. 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. **Apply smoothing if appropriate. 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 Q-Learning 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 Q-Learning. 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 environment step (Eq. 6.9 from Sutton and Barto).
- Implement a full training loop, where you return a vector with the obtained cumulative reward per episode.

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. **Apply smoothing if appropriate. 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 and SARSA.

5 n-step SARSA

You decide to try yet another algorithm on the original ShortCut enviroment: n-step SARSA. Now your agent will start to make updates width a depth above one. You follow the same scheme as for your previous experiments.

a Correctly complete the methods of `nStepSARSAAgent()` 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 n-step SARSA update equation to update $Q(s, a)$ values, based on Equations 7.4 and 7.5 from Sutton and Barto. Note that you now need to feed a sequence of states, actions and rewards to the update function. **Although the book does show a full algorithm on page 147, we advice to ignore it: it is correct, but quite complex and cluttered with all the indexing. Instead, just implement the below logic**:
  - At the start of an episode, you first need to take $n(+1)$ actions before you can perform your first update (to the first state-action of the episode).
  - Then, after every next action you take, you can also perform one update (for the state-action pair n steps ago).
  - Once your episode terminates, you still need to update all last state-action pairs in the episode (which are less than n steps before the end and were still waiting for an update).

  (Also note: if your episode terminates before the first n actions, you still need to update the observed state-actions, i.e., you jump to the third item of the above list.)

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

- Write a completely new function (e.g., `run_repetitions_nstepsarsa()`) 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'` and `agent_type='nstepsarsa'`.

Again, first do a single experiment with `n_episodes=10000` episodes. 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 different values of `n`: `[1, 2, 5, 10, 25]` (where you use an appropriate value of `alpha`, which you may base on your previous results).
- Run your function from 1b for these different values of `n`, 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. **Apply smoothing if appropriate. 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 explain the potential benefits and problems of multi-step ($n > 1$) updates. What happens if we set $n \to \infty$?

6 Comparison and conclusions

Finish your report by comparing all algorithms you implemented with eachother.

- Compare the best version of every algorithm in a single graph and/or table.
- In your discussion of these results, first focus on the different versions of 1-step updates (Q-learning, SARSA, Expected SARSA). What are their strengths and weaknesses? Refer back to the qualitative preferred paths each algorithm finds as well.
- Then present a similar discussion for the difference between single-step and multi-step methods.
- Conclude with a short overall reflection: what did you learn about model-free RL algorithms? Do you see a possible next step for these experiments?