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

Research Question

In this assignment, you will study two model-based reinforcement learning (MBRL) algorithms:

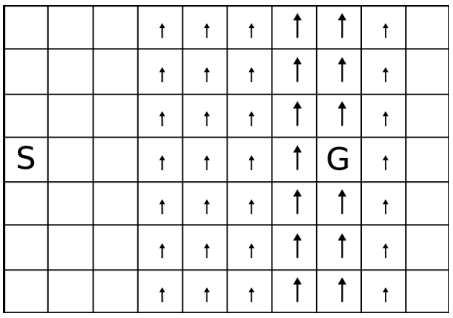
- Dyna (Sutton and Barto, Sec. 8.2) with ϵ-greedy exploration.
- Prioritized sweeping (Sutton and Barto, Sec. 8.4) with ϵ-greedy exploration.

Your goal is to investigate these two algorithms. In particular, you will:

1. Implement these algorithms.
2. For each algorithm investigate the effect of three hyperparameters:
    - `epsilon`: the parameter that scales the amount of exploration.
    - `n_planning_iterations`: the number of planning iterations each algorithm makes in between a real environment step.
    - `learning_rate`: the magnitude of each learning update step.

Environment

You will use the *Windy Gridworld* environment for your studies, as specified in Example 6.5 (page 130) of Sutton and Barto.



The environment consists of a 10x7 grid, where at each cell we can move up, down, left or right. We start at location (0,3) (we start indexing at 0, as is done in Python as well), indicated in the figure by 'S'. Our goal is to move to location (7,3), indicated by 'G'. However, a special

feature of the environment is that there is a vertical wind. In columns 3, 4, 5 and 8, we are pushed one additional step up, while in columns 6 and 7, we move up two additional steps. The reward of the agent at each step is -1, while reaching the goal gives an reward of +20, and terminates the episode.

Preparation

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

Files You are provided with four Python files:

- `MBRLEnvironment.py`: This file generates the environment. Run the file to see a demonstration of the environment with randomly selected actions. Inspect the class methods and make sure you understand them. With `render()` you can interactively visualize the environment during execution. If you provide `Q_sa` (a Q-value table), the environment will also display the Q-value estimates for each action in each state, while toggling `plot_optimal_policy` will also show arrows for the optimal policy.
- `MBRLAgents.py`: This file contains placeholder classes for your two MBRL algorithms: `DynaAgent`, and `PrioritizedSweepingAgent`. Currently, each method randomly selects and action, and does not perform any updates. 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.
- `MBRLExperiment.py`: In this file you will write your experiment code.
- `Helper.py`: This file contains some helper classes for plotting and smoothing. You can choose to use them, but are of course free to write your own code for plotting and smoothing as well. Inspect the code and run the file to verify that your understand what they do.

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.
- All **code** to replicate your results. Your submission should contain:
  - The original `MBRLEnvironment.py` and `Helper.py`
  - Your modified `MBRLAgents.py`.
  - Your modified `MBRLExperiment.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 runs is stochastic)!
- In each repetition, really start from scratch, i.e., randomly initialize a new environment, and initialize your policy from scratch.
- Do not fix any seeds within the loop over your repetitions! Each repetition should really be an independent repetition.
- Average your curves over repetitions. If necessary, apply additional smoothing to your curves.

1 Dyna

You first decide to study the Dyna algorithm. You proceed in four steps:

a Correctly complete the class `DynaAgent()` in the file `MBRLAgents.py`.

- In `init()`, initialize the means $Q(s,a)$, transition counts $n(s,a,s')$ and reward sums $R_{sum}(s,a,s')$ for each action to 0.
- In `select_action()`, implement the ϵ-greedy policy.
- Most work needs to happen in the `update()` function. Full pseudocode is available in the book, which is repeated below with more detail:

Algorithm 1: Dyna (with ϵ-greedy exploration) pseudo-code.

Input: Exploration parameters ϵ, number of planning updates K, learning rate α, discount parameter γ, maximum number of timesteps T

Initialization: Initialize $Q(s,a) = 0$, $n(s,a,s') = 0$, $R_{sum}(s,a,s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$.

for $t = 1...T$ **do**

- $s \leftarrow$ current state — `/* Reset when environment terminates */`
- $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ — `/* Sample action */`
- $r, s' \sim p(r, s'|s, a)$ — `/* Simulate environment */`
- $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ — `/* Update model (Alg.2) */`
- $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s', a') - Q(s,a)]$ — `/* Update Q-table */`
- **repeat** K **times**
  - $s \leftarrow$ random previously observed state — `/* Find state with` $n(s) > 0$ `*/`
  - $a \leftarrow$ previously taken action in state s — `/* Find action with` $n(s,a) > 0$ `*/`
  - $s', r \sim \hat{p}(s', r|s, a)$ — `/* Simulate model */`
  - $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s', a') - Q(s,a)]$ — `/* Update Q-table */`
- **end**

end

Algorithm 2: Estimate model $\hat{p}(s', r|s, a)$, split up as a tabular dynamics model $\hat{p}(s'|s, a)$ and deterministic reward function $\hat{r}(s, a, s')$.

Input: Maximum number of timesteps T.

Initialization: Initialize $n(s,a,s') = 0$ and $R_{sum}(s,a,s') = 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$

repeat

- Observe $\langle s, a, r, s' \rangle$ — `/* Observe new transition */`
- $n(s,a,s') \leftarrow n(s,a,s') + 1$ — `/* Update transition counts */`
- $R_{sum}(s,a,s') \leftarrow R_{sum}(s,a,s') + r$ — `/* Update reward sums */`
- $\hat{p}(s'|s,a) = \frac{n(s,a,s')}{\sum_{s'} n(s,a,s')}$ — `/* Estimate transition function */`
- $\hat{r}(s,a,s') = \frac{R_{sum}(s,a,s')}{n(s,a,s')}$ — `/* Estimate reward function */`
- $\hat{p}(s,a|s') = \frac{n(s,a,s')}{\sum_{s,a} n(s,a,s')}$ — `/* Reverse model (only for prior. sweeping) */`

until -;

Verify that your code works by running `MBRLAgents.py`. You can manually execute every step by pressing 'Enter', or complete the full run by pressing 'c'. Look at how the agents learns,

how the $Q(s, a)$ value estimates change, and how the optimal policy changes. Is it easy for the agent to learn?

b Write a function `run_repetitions()` in `MBRLExperiment.py`. Your function should repeatedly test the `DynaAgent()` on an instance of `WindyGridworld()`. **Switch plotting off for your repetitions, plotting will slow the run down a lot**.

- Run `n_rep=10` repetitions, with `n_timesteps=10000` steps for each repetition. **Be sure to initialize a clean policy and environment instance for each repetition** (without fixing a seed)!
- Average the learning curves over your `n_rep=10` repetitions, smooth your curve with `smoothing_window=101`, and plot the result.

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

c You decide to run three more structured experiments:

- Test `epsilon`: `[0.01,0.05,0.1,0.25]`, for `learning_rate=0.5` and `n_planning_updates=5`.
- Test `learning_rate`: `[0.1,0.5,1.0]`, for `epsilon=0.05` and `n_planning_updates=5`.
- Test `n_planning_updates`: `[1,5,15]`, for `epsilon=0.05` and `learning_rate=0.5`.

Run your function from 1b for these different experiments, where you again average over `n_rep=10` repetitions of `n_timesteps=10000` steps, and smooth your curve with `smoothing_window=101`.

Make a nice plot for each of the three experiment, where you compare the learning curves for the different hyperparameter settings. **Add a legend, and label the x and y-axis appropriately**. You could use the `LearningCurvePlot()` class for this.

d Write the first section of your report. Describe:

- Your methodology (with equations).
- Your results (graphs).
- Interpret the results, give possible explanations.

2 Prioritized sweeping

You decide to repeat the above procedure with a different idea, prioritized sweeping, which also plans in the reverse direction to spread the information more quickly over the state space. You should largely be able to reuse your code from the previous experiment.

a Correctly complete the class `PrioritizedSweepingAgent()` in the file `MBRLAgents.py`.

- In `init()`, initialize the means $Q(s,a)$, transition counts $n(s,a,s')$ and reward sums $R_{sum}(s,a,s')$ for each action to 0, and initialize an empty priority queue PQ.
- In `select_action()`, implement the ϵ-greedy policy.
- Most work needs to happen in the `update()` function. Full pseudocode is available in the book, which is repeated below with more detail:

Algorithm 3: Prioritized sweeping (with ϵ-greedy exploration) pseudo-code.

Input: Exploration parameters ϵ, number of planning updates K, learning rate α, discount parameter γ, maximum number of timesteps T, priority threshold θ.
Initialization: Initialize $Q(s,a)=0$, $n(s,a,s')=0$, $R_{sum}(s,a,s')=0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$, and prioritized queue PQ.
for $t = 1...T$ **do**
  $s \leftarrow$ current state /* Reset when environment terminates */
  $a \sim \pi_{\epsilon\text{-greedy}}(a|s)$ /* Sample action */
  $r, s' \sim p(r, s'|s, a)$ /* Simulate environment */
  $\hat{p}(s', r|s, a) \leftarrow \text{Update}(s, a, r, s')$ /* Update model (Alg.2) */
  $\mathbf{p} \leftarrow |r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a)|$ /* Compute priority p */
  if $\mathbf{p} > \theta$ **then**
    Insert (s, a) into PQ with priority $\mathbf{p}$ /* State-action needs update */
  end
  /// **Start sampling from** PQ **to perform updates**
  repeat K **times**
    $s, a \leftarrow$ pop highest priority from PQ /* Sample PQ, break when empty */
    $s', r \sim \hat{p}(s', r|s, a)$ /* Simulate model */
    $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a)]$ /* Update Q-table */
    /// **Loop over all state action that may lead to state** s
    for *all* $(\bar{s}, \bar{a})$ *with* $\hat{p}(\bar{s}, \bar{a}|s) > 0$ **do**
      $\bar{r} = \hat{r}(\bar{s}, \bar{a}, s)$ /* Get reward from model */
      $\mathbf{p} \leftarrow |\bar{r} + \gamma \cdot \max_a Q(s, a) - Q(\bar{s}, \bar{a})|$ /* Compute priority p */
      if $\mathbf{p} > \theta$ **then**
        Insert (s, a) into PQ with priority $\mathbf{p}$ /* State-action needs update */
      end
    end
  end
end

Verify that your code works by running `MBRLAgents.py`. Look at how the agents learns, how the $Q(s,a)$ value estimates change, and how the optimal policy changes. Is it easy for the agent to learn?

b Modify `run_repetitions()` to work for your `PrioritizedSweepingAgent` as well. **Switch plotting off for your repetitions, plotting will slow the run down a lot**.

c Run the same structured experiments as before:

- Test `epsilon`: `[0.01,0.05,0.1,0.25]`, for `learning_rate=0.5` and `n_planning_updates=5`.
- Test `learning_rate`: `[0.1,0.5,1.0]`, for `epsilon=0.05` and `n_planning_updates=5`.
- Test `n_planning_updates`: `[1,5,15]`, for `epsilon=0.05` and `learning_rate=0.5`.

Run your `run_repetitions()` for each of these experiments, where you again average over `n_rep=10` repetitions of `n_timesteps=10000` steps, and smooth your curve with `smoothing_window=101`.

Make a nice plot for each of the three experiment, where you compare the learning curves for the different hyperparameter settings. **Add a legend, and label the x and y-axis appropriately**. You could use the `LearningCurvePlot()` class for this.

d Write a second section of your report. Describe:

- Your methodology (with equations).
- Your results (graphs).
- Interpret the results, give possible explanations.

3 Reflection

Write a short reflection on both algorithms:

- What could be a strength and weakness of model-based RL compared to model-free RL (like Q-learning)?
- How do you compare Dyna and Prioritized Sweeping? Which approach performed better (you may add a plot where you compare the best performing models)? Which idea do you like better? Could both be combined?
- All our model-based reinforcement learning experiments did not vary action selection: we always used ϵ-greedy exploration. Do you think ϵ-greedy is a smart exploration approach? Could you think of other approaches? Could you think of an exploration method that would also use the dynamics models you learned in this assignment?