Course Project

Reinforcement Learning

Saarland University – Winter Semester 2021/22

rl-w21-tutors
rl-w21-tutors@mpi-sws.org

1 Introduction

1.1 Solving Tasks in the Karel Environment with Basic Actions

For the course project, we will continue to use the block-based visual programming paradigm where one has to put together a sequence of commands (code blocks) to solve a visual task as introduced in Section 3 of the Week 2 assignment. As before, the commands (code blocks) will be limited to basic actions without involving programming constructs such as conditionals or loops. The elements of a Karel programming task and its command set is exactly the same as described in Section 3 of the Week 2 Assignment, under the paragraph headings "Description of visual task" and "Description of Karel commands".

In this project, we will limit the tasks to 4×4 grids. Figure 1 shows a few examples of such Karel tasks. To solve these tasks, the Karel AVATAR has the following commands in its command set: {`move`, `turnLeft`, `turnRight`, `pickMarker`, `putMarker`, `finish`}.

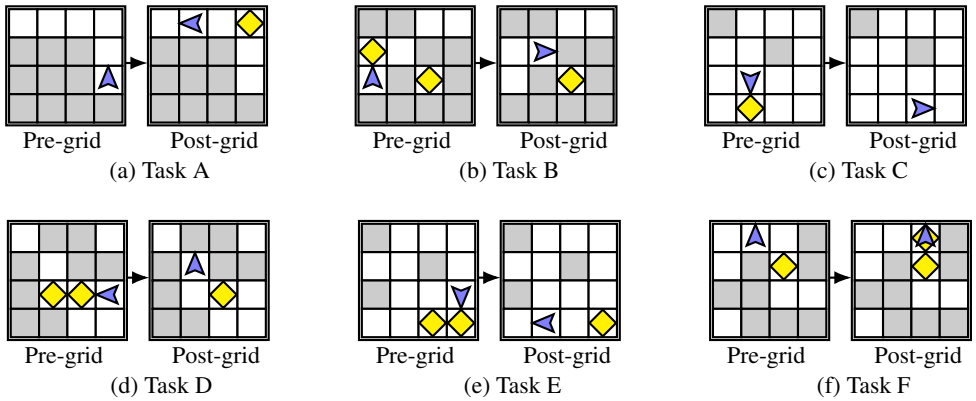

Figure 1: Karel tasks with 4×4 grid.

Solving a Karel task. To solve a given task, one has to use a sequence of commands that transforms the Pre-grid to Post-grid; this sequence must end with the `finish` command and there should be no termination via "crashes" during the execution. Furthermore, the objective is to use small number of commands to solve the task. For instance, Figure 1a shows a Karel task that can be solved with a minimal-sized command sequence: {`move`, `move`, `turnLeft`, `putMarker`, `move`, `move`, `finish`}.

1.2 RL Agent's Objective

In all the course assignments, the RL agent's objective was to solve a **fixed** Karel task (referred to as T_1 or T_2 in assignments). To solve a fixed task, we considered the tabular setting and designed an agent that learns a tabular policy.

In this project, the objective of the RL agent is to learn a policy that can solve **any** Karel task with 4×4 grid provided as input to the policy; Figure 1 shows a few examples of such Karel tasks. In this case, the state space becomes intractably large, and we will consider an agent with neural policies (i.e., a policy represented using a neural network). The idea is that the agent should be able to learn from experience on tasks seen during the training phase and then generalize its experience to new unseen tasks during the deployment phase. More concretely, we have the following two phases:

- **Training phase:** To train the agent's neural policy, we are given a set of Karel tasks $S_{\text{init}}^{\text{train}} \subset S$ (i.e., the "train" dataset); each task is represented by a tuple (Pre-grid, Post-grid).
- **Deployment phase:** After the training phase, the agent's neural policy will be evaluated on a set of new unseen Karel tasks $S_{\text{init}}^{\text{test}} \subset S$ (i.e., the "test" dataset); each task is represented by a tuple (Pre-grid, Post-grid).

We borrow the following ingredients of the problem setup from the Week 5 and 6 assignments:

- The RL agent's objective is to solve any given Karel task with 4×4 grid using a short sequence of commands.
- The environment operates in an infinite horizon setting with a discount factor γ. An episode starts by sampling a starting state (i.e., a task) and the agent's interaction will continue forever, until the program termination happens (i.e., until the AVATAR "crashes" or the `finish` command is sent).
- The agent does not have access to fully specified transition dynamics P or reward function R; instead, the agent can "query" the environment with a state-action pair (s, a); in turn, the environment will provide "feedback" in the form of a reward value r and next state s'.
- During the training phase, your environment's set of initial states is given by $S_{\text{init}}^{\text{train}}$ as discussed above. When resetting and starting an episode, a state $s \in S_{\text{init}}^{\text{train}}$ will be chosen, i.e., the agent starts solving a fresh task given by a tuple (Pre-grid, Post-grid). Your algorithms might need thousands (or possibly millions) of episodes before convergence.
- Your algorithms might be stuck in an infinite loop if no crash happens in an episode. In your implementation of the agent's algorithms, you should add a hyperparameter H denoting the maximal length of a policy rollout in an episode, after which you will stop and reset the episode. For your implementation, you should add this hyperparameter H as part of the agent's algorithms and not as part of the environment.

1.3 Project Milestones

We have divided the project into three milestones where each milestone comprises a part of the project grade as described below. *REMARK: We have added submission due dates per milestone to ensure that you can seek timely feedback. It is important to note that the implementation component in Project Milestone#2 and Project Milestone#3 will take substantial time, and we suggest to not wait for the submission due date of Project Milestone#1 before starting the implementation.*

- **Project Milestone#1: DESIGN.** This comprises 40% of the grade points for the project and has a submission due date of 16 Jan 2022, 4pm CET. In this milestone, you will work on conceptual elements of the project that are needed to complete it; this milestone does not involve any implementation.
- **Project Milestone#2: TRAIN.** This comprises 50% of the grade points for the project and has a submission due date of 30 Jan 2022, 4pm CET. In this milestone, you will implement and train the RL agent to achieve its objective.
- **Project Milestone#3: DEPLOY.** This comprises 10% of the grade points for the project and has a submission due date of 6 Feb 2022, 4pm CET. In this milestone, you will deploy your RL agent's trained neural policy and evaluate its performance on Karel tasks in the "test" dataset.

In the next sections, we present the details of each milestone. It is important that you read the entire project PDF (i.e., description of ALL the milestones) before starting to work on the project.

2 Project Milestone#1: DESIGN

2.1 Exercise

In this milestone, you will work on conceptual elements of the project that are needed to complete the project. To structure this milestone, we have put together several questions below to guide the design process of the project. You should answer these questions in the PDF submission file – see instructions in Section 2.2.

(E.1) Design an MDP that captures the RL agent's objective and describe it in detail.

(E.2) Given that the state space of the MDP is intractably large, what are possible feature representations you would use for training neural policies?

(E.3) What policy gradient algorithms would you use to train the RL agent? What are potential challenges you expect to encounter while training the agent? Describe any techniques that you would use to mitigate these challenges and accelerate the training process.

HINT: As potential techniques to accelerate training, you can think of curriculum design and reward design as we briefly discussed during the course. Furthermore, during the training phase, you could also leverage the optimal sequences of commands for tasks in the "train" dataset – this technique is also known as Imitation Learning or Learning-from-Demonstrations.

2.2 Submission Instructions

Please submit the following file:

- **<Lastname>_project1_design.pdf**. This PDF file will contain submissions to the following exercises: E.1, E.2, E.3. The PDF file should be generated in latex using NeurIPS 2021 style files; see further formatting instructions below. The PDF file size should **not exceed 5mb**. Please use the following naming convention: Replace <Lastname> with your respective Lastname with initial letter capitalized (e.g., Singla_project1_design.pdf).

Please use the following formatting instructions for the PDF submission:

- The PDF file should be generated in latex using NeurIPS 2021 style files available at https://neurips.cc/Conferences/2021/PaperInformation/StyleFiles.
- Inside the .tex source, use the option: *\usepackage[preprint]{neurips_2021}*.
- Inside the .tex source, use **"Project Milestone#1: DESIGN"** in the title field, and your Lastname / email in the author's field.
- For ease of reading, we recommend that you create different sections (e.g., one per exercise).

3 Project Milestone#2: TRAIN

3.1 Exercise

In this milestone, you will implement and train the RL agent to achieve its objective. To train the agent's neural policy, we are given a set of Karel tasks corresponding to $S_{\text{init}}^{\text{train}} \subset S$ (i.e., the "train" dataset) as described in the next subsection. Specifically, answer the following questions:

(I.1) Implement the MDP designed in Project Milestone#1, along with RL algorithms and specific acceleration techniques to train the agent to solve any Karel task; by default, we recommend using the Python programming language for implementation. Your submission must also contain a README. This should be submitted in the ZIP file – see instructions in Section 3.3

(I.2) Describe the implementation details including the following: (a) the state feature representation; (b) the details of the RL algorithm used during training; (c) the network architecture used for the RL agent's neural policy; (d) the agent's hyperparameters. This should be answered in the PDF submission file – see instructions in Section 3.3.

(I.3) Describe any specific techniques (e.g., curriculum design, reward design, Learning-from-Demonstrations) that you used to accelerate the training process. Discuss the effect of these techniques in terms of speedup or other performance metrics (e.g., per episode reward or % number of tasks solved). You can report these findings through tables or plots. This should be answered in the PDF submission file – see instructions in Section 3.3.

(I.4) Report the final performance of your RL agent based on your best performing configuration (i.e., hyperparameters, algorithmic choices, and any specific techniques). Here, you will report performance of the agent on tasks in a validation dataset (henceforth, referred to as the "val" dataset) – this validation dataset serves as a proxy for the "test" dataset that will be made available to you as part of Milestone#3. Tasks in the "val" dataset corresponds to set of states $S_{\text{init}}^{\text{val}} \subset S$. Report the final performance in terms of the following two metrics: (a) % number of tasks solved for tasks corresponding to $S_{\text{init}}^{\text{val}}$; (b) % number of tasks solved with minimal-sized sequences for tasks corresponding to $S_{\text{init}}^{\text{val}}$. As described in the next subsection, we have provided you with minimal-sized sequences for these tasks in the folder `datasets/data/val/` that can be used to evaluate these metrics. This should be answered in the PDF submission file – see instructions in Section 3.3.

3.2 Datasets Description

"train" dataset tasks: Folder `datasets/data/train/task/` contains 24000 Karel tasks – see Figure 1 for a few examples. Each task in this folder is saved as a JSON file, and contains the task's configuration specifying the tuple (Pre-grid, Post-grid). When generating the dataset, we used the following coordinate system: a location $[r, c]$ corresponds to r^{th} row and c^{th} column; the "origin" $[0, 0]$ corresponds to the grid cell in the top-left corner. As a concrete example, the JSON representation of Task C shown in Figure 1c is:


```
{
    "gridsz_num_rows": 4,
    "gridsz_num_cols": 4,
    "pregrid_agent_row": 2,
    "pregrid_agent_col": 1,
    "pregrid_agent_dir": "south",
    "postgrid_agent_row": 3,
    "postgrid_agent_col": 2,
    "postgrid_agent_dir": "east",
    "walls": [[0, 0], [1, 2]],
    "pregrid_marker": [[3, 1]],
    "postgrid_markers": []
}
```

REMARK: You are not limited to this "train" dataset, and are free to generate more Karel tasks for the training phase to improve the agent's performance.[1]

"train" dataset minimal-sized sequences: Folder `datasets/data/train/seq/` contains the minimal-sized sequences for each of the Karel tasks in the folder `datasets/data/train/task/`. Each sequence in this folder is saved as a JSON file with same ID as the corresponding task. As a concrete example, the JSON representation of the minimal-sized sequence for Task C in Figure 1c is: {"sequence": ["move", "turnLeft", "pickMarker", "move", "finish"]}. These minimal-sized sequences can be used to evaluate the performance of the RL agent during the training phase, and may also be useful for accelerating the training process via the technique of Learning-from-Demonstrations as mentioned in Section 2.1.

"val" dataset: Folder `datasets/data/val/` contains a validation dataset with the same format and structure as the "train" dataset. This dataset can be used as a proxy for the "test" dataset and you are asked to report the performance of the agent's trained neural policy on the "val" dataset in the question I.4 above.

3.3 Submission Instructions

Please submit the following two files:

- **<Lastname>_project2_train.zip**. This ZIP file will contain submission of the following exercise: I.1. Importantly, the ZIP file should unzip to a folder named **<Lastname>_project2_train**. Inside this folder, please add minimal code files (e.g., only specific Python scripts and a README). The ZIP file size should **not exceed 5mb**. Please use the following naming convention: Replace <Lastname> with your respective Lastname with initial letter capitalized (e.g., Singla_project2_train.zip).
- **<Lastname>_project2_train.pdf**. This PDF file will contain submissions to the following exercises: I.2, I.3, I.4. The PDF file should be generated in latex using NeurIPS 2021 style files; see further formatting instructions below. The PDF file size should **not exceed 5mb**. Please use the following naming convention: Replace <Lastname> with your respective Lastname with initial letter capitalized (e.g., Singla_project2_train.pdf).

Please use the following formatting instructions for the PDF submission:

- The PDF file should be generated in latex using NeurIPS 2021 style files available at https://neurips.cc/Conferences/2021/PaperInformation/StyleFiles.
- Inside the .tex source, use the option: *\usepackage[preprint]{neurips_2021}*.
- Inside the .tex source, use **"Project Milestone#2: TRAIN"** in the title field, and your Lastname / email in the author's field.
- For ease of reading, we recommend that you create different sections (e.g., one per exercise).

[1] We have also provided you with two simpler variants of `datasets/data/`: `datasets/data_easy/` and `datasets/data_medium/`. These two simpler datasets consist of tasks which require only up to three commands to solve the tasks; moreover, `datasets/data_easy/` does not have any MARKER in the grids. These simpler variants of the datasets could be useful to debug the training process of your RL agent.

4 Project Milestone#3: DEPLOY

4.1 Exercise

In this milestone, you will deploy your RL agent's trained neural policy and evaluate its performance on new unseen Karel tasks in the "test" dataset – these tasks correspond to the set of initial states $S_{\text{init}}^{\text{test}} \subset S$. The "test" dataset will be released after the submission deadline for Milestone#2; meanwhile, you can use the "val" dataset discussed in the previous section as a proxy for the "test" dataset.

The goal of this milestone is to evaluate if the RL agent is able to learn from experience on tasks seen during the training phase and then generalize its experience to new unseen tasks during the deployment phase. Importantly, we will be using rollouts of the trained neural policy to solve tasks in the "test" dataset, instead of computing a sequence of commands for a task from scratch. More concretely, when starting an episode with $s \in S_{\text{init}}^{\text{test}}$, use the trained policy to rollout an episode; then, actions taken during this rollout is the generated sequence of commands for the task corresponding to state s. The agent's generalization performance will be measured by the quality of these generated sequences using the following two metrics as discussed earlier: (a) % number of tasks solved for tasks corresponding to $S_{\text{init}}^{\text{test}}$; (b) % number of tasks solved with minimal-sized sequences for tasks corresponding to $S_{\text{init}}^{\text{test}}$. For the "test" dataset, these metrics will be computed by tutors based on your submission; you will not have access to minimal-sized sequences for these tasks. For this milestone, submit the files as specified in the following two questions:

(I.1) Submit a script along with model parameters of the RL agent's trained policy. When this script is executed from the command line, it should take as input any Karel task represented in the JSON format as described above, and should print the generated sequence of commands. By default, we recommend using the Python programming language for implementation. Your submission must also contain a README. This should be submitted in a ZIP file – see instructions in Section 4.2.

(I.2) In addition to submitting the trained policy, execute the policy on all tasks in the "test" dataset and save the generated sequences of commands for each task as a separate JSON file. This should be submitted in a separate ZIP file – see instructions in Section 4.2

4.2 Submission Instructions

Please submit the following files:

- **<Lastname>_project3_deploy_policy.zip**. This ZIP file will contain submission for the following exercise: I.1. Importantly, the ZIP file should unzip to a folder named **<Lastname>_project3_deploy_policy**. Inside this folder, please add minimal code files (e.g., only specific Python scripts, model parameters, and a README). The ZIP file size should **not exceed 5mb**. Please use the following naming convention: Replace <Lastname> with your respective Lastname with initial letter capitalized (e.g., Singla_project3_deploy_policy.zip).
- **<Lastname>_project3_deploy_testseqs.zip**. This ZIP file will contain submission for the following exercise: I.2. Importantly, the ZIP file should unzip to a folder named **<Lastname>_project3_deploy_testseqs**. Inside this folder, please add each generated sequence in a separate JSON file similar to the format of sequences presented in the folders `datasets/data/train/seq/` or `datasets/data/val/seq/`. The filename for each generated sequence must be the same as its corresponding task ID in the "test" dataset. The ZIP file size should **not exceed 5mb**. Please use the following naming convention: Replace <Lastname> with your respective Lastname with initial letter capitalized (e.g., Singla_project3_deploy_testseqs.zip).

Based on template from https://neurips.cc/Conferences/2021/PaperInformation/StyleFiles