

1BM120 - Assignment 3

Deep Reinforcement Learning

Luca Begnardi, Dr. Zaharah Bukhsh, Dr. Laurens Bliet

May 2024

1 Problem Description

The Bounded Knapsack problem is a combinatorial optimization problem. It requires the user to select from a range of goods of different values and weights to maximize the value of the selected items within a given weight limit. The bounded implies that each item can be selected a limited number of times. *For this assignment, you will train a reinforcement learning agent to solve the Bounded Knapsack problem with 200 items.* The environment is adapted from the OR-GYM [1] library and is based on the Gymnasium framework. The **RL model's state** is defined as a concatenation of vectors containing items weight, items value, items limit, as well as the knapsack's current load and maximum capacity. The number of actions is equal to the number of items available. The **reward** is the total value of all items placed within the knapsack. At each timestep, the agent can select only one item. An episode ends when the knapsack is full or no additional items can be placed.

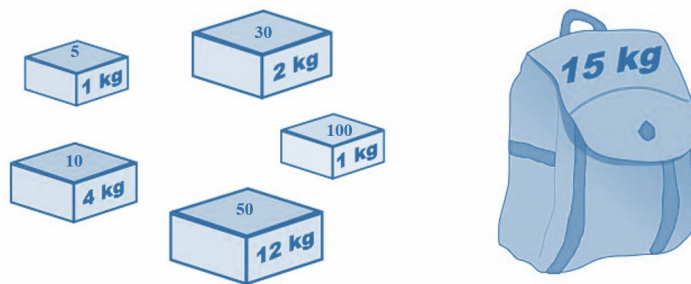


Figure 1: Illustration of Knapsack problem

The assignment has two parts.

Part 1: Basic agent

Using the algorithm implementations provided in the **Stable Baselines3** package, train a Reinforcement Learning agent that uses a neural network as a function approximator to solve the **Bounded Knapsack** environment. With the right algorithm and optimal parametric settings, **your agent can get an average reward between 1300 and 1500 over 100 episodes.**

Part 2: Invalid action masking

In the previous part, your agents have learnt how to exploit the information contained in the environment state to decide the next item to pack. Since the episodes end every time the agent selects an item which does not fit the knapsack or which is not available anymore, the neural network has to implicitly learn not to select those items. However, given the state of the environment at every step it is easy to compute whether an item (or action) can or cannot still be selected. The BoundedKnapsack environment offers an action mask that is automatically updated after every step and can be accessed through the method `env.get_mask()`.

This information can be used in policy gradient algorithms to:

- avoid that an agent selects an invalid action when inferred
- improve the training process, recognizing that certain actions should not be selected in a certain state; the agent can learn how to correctly assess those actions left more easily.

In particular, the process of action masking for policy gradient algorithms consists in setting the logits (obtained from the output layer of the policy network) to minus infinity (or the lowest possible value) for the values corresponding to the invalid actions such that the related probability will be 0 after applying the softmax function.

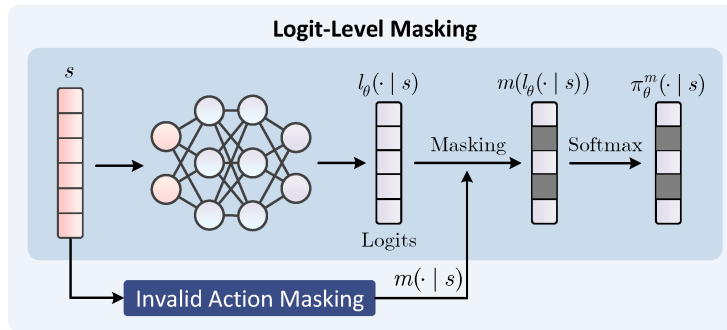


Figure 2: Overview of logit-level masking

Using the algorithm implementation provided in the **SB3 Contrib** package, train a MaskablePPO agent **that can reach an average reward above 2000 over 100 episodes**.

2 Work structure

To solve the proposed task, the following steps must be undertaken:

Part 1

- Train and test at least two different DRL agents (using two of the algorithms provided in Stable Baselines3 [2], e.g. PPO [3], DQN [4], A2C [5], etc) on the BoundedKnapsack environment;
- Experiment with different neural network architectures;
- Tune the algorithms hyperparameters (by hand);
- Evaluate the agents and compare the best results obtained using the different algorithms.

Part 2

- Enabling action masking, train and test a MaskablePPO agent from SB3 Contrib [6] on the BoundedKnapsack environment;
- Experiment with different neural network architectures and tune the algorithm hyperparameters (by hand);
- Evaluate the agent and compare the best results obtained with those of the best agent from Part 1.

Refer to Rubric given in appendix A for further details.

2.1 Submission details

The project must be implemented using Python 3.11 and PyTorch 2.0+. The submission folder of the project should contain the following files:

- Python file, containing:
 - Code (well-commented) for training and testing ¹ the DRL agents;
- Trained model weights of the three agents;
- Report file, containing:

¹For testing, we should be able to load and execute your trained agents weights/policy.

- Brief description of the environment including state, actions, rewards and mask;
- Explain the chosen algorithms and motivate your choices (only for Part 1);
- Details of tuned hyperparameters such as epsilon (exploration/exploitation trade-off), learning rate and discount factor and their impact on the models’ performances;
- Discussion and visualization of the agents’ training over the timesteps and average accumulated reward;
- For each agent, the indication of the highest gained rewards and number of training timesteps;
- Explanation of which agent performed better and why.

3 General indications

The files described must be submitted in a zip file containing one single directory, as shown in Figure 3. The naming convention for the directory is “Groupnumber_1BM120_A3” (e.g. Group00_1BM120_A3). The maximum length of the report is **6 pages**, excluding a title page and optional references.





Name	Date modified	Type
 trained_models	23/05/2022 13:27	File folder
 Group00_1BM120_A3_part1.py	23/05/2022 13:28	Python File
 Group00_1BM120_A3_part2.py	23/05/2022 13:28	Python File
 Group00_1BM120_A3_report.pdf	23/05/2022 13:29	Adobe Acrobat D...

Figure 3: Guide for files names and submission directory structure

4 Tips for getting started

1. To install Gymnasium, simply use pip:

```
pip install gymnasium
```

2. To ensure the reproducibility of your code (for assessment), add the following lines at the beginning of your file:

```
import numpy as np
seed = (number of your choice, e.g. 2024)
np.random.seed(seed)
```

3. Download the provided knapsack_env.py file. Import the environment from the provided file:

```
from knapsack_env import BoundedKnapsackEnv
```

4. Create an instance of the environment with the following parameters:

```
env = BoundedKnapsackEnv(n_items=200, max_weight=200)
```

- (a) For Part 2, you will first need to enable the mask:

```
env = BoundedKnapsackEnv(n_items=200, max_weight=200, mask=True)
```
- (b) Then, you need to wrap the environment with the ActionMasker:

```
env = ActionMasker(env, mask_fn)
```

Note that the implementation of the mask_fn function is left to you.

5. Inspect the state space and action spaces as follows:

```
state_space = env.reset()  
action_space_size = env.action_space.n
```

6. To install Stable Baselines3 and SB3 Contrib use pip:

```
pip install stable-baselines3  
pip install sb3-contrib
```

5 References

- [1] Christian D. Hubbs, Hector D. Perez, Owais Sarwar, Nikolaos V. Sahinidis, Ignacio E. Grossmann, and John M. Wassick. Or-gym: A reinforcement learning library for operations research problems, 2020.
- [2] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.
- [3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.
- [4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dhharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.
- [5] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.
- [6] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Sb3 contrib. <https://github.com/Stable-Baselines-Team/stable-baselines3-contrib>.

A Assignment Rubric (20 points)

A.1 Part 1 (14 points)

Training and testing code (3 points)	The code is functional and well-documented for training and testing two different agents to solve the Bounded Knapsack Problem, using Stable Baselines3 library.
Model weights (2 points)	The trained model weights of successful agents are given. A successful agent must achieve an average reward between of at least 1300 over 100 episodes.
Environment (2 point)	The report introduces the environment of the Bounded Knapsack Problem.
Choice of learning algorithm (2 points)	The report clearly describes the learning algorithms and why these specific algorithms were chosen.
Hyperparameters (2 points)	The report provides the details of tuned hyperparameters such as epsilon (exploration/exploitation trade-off), learning rate and discount factor and their impact on the models' performances.
Learning visualization (1 points)	Plots of the average accumulated reward over timesteps are provided for both agents.
Results (1 points)	For each agent, the highest obtained reward is presented in a tabular form, along with the number of needed timesteps and the related configuration of hyperparameters.
Agents comparison (1 points)	The report discusses about the differences between the results obtained by the two agents provided in the previous sections, highlighting which one performed better and explaining why this happens.

A.2 Part 2 (6 points)

Training and testing code (2 points)	The code is functional and well-documented for training and testing a MaskablePPO agent to solve the Bounded Knapsack Problem, using SB3 Contrib library.
Model weights (1 points)	The trained model weights of successful agents are given. A successful agent must achieve an average reward of at least 2000 over 100 episodes.
Learning visualization (1 points)	A plot of the average accumulated reward over timesteps is provided.
Results (1 points)	The highest obtained reward is presented in a tabular form, along with the number of needed timesteps and the related configuration of hyperparameter.
Agents comparison (1 points)	The report discusses the differences between the results obtained by the new agent and the best one found in Part 1, highlighting which one performed better and explaining why this happens.