

# 1BM120 - Assignment 3

## Deep Reinforcement Learning

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

May 2023

## 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 implemented in OR-GYM [1] library version 0.5.0 under the name of ‘Knapsack-v2’ [2]. 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 until the knapsack is full, or no further items can be placed.

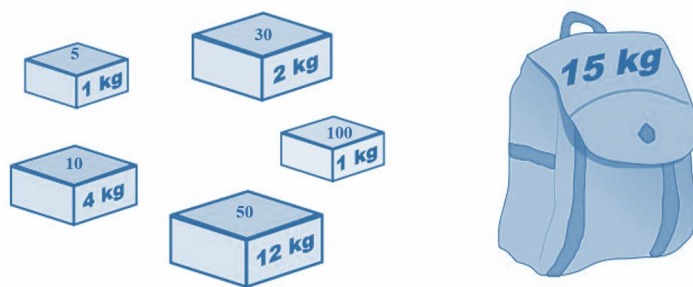


Figure 1: Figure 2: Illustration of Knapsack problem

Using the algorithm implementations provided in the **Stable Baselines3** package, you need to train a Reinforcement Learning agent that uses a neural

network as a function approximator to solve the **Knapsack-v2** environment. Typically, **this environment is considered solved when the agent obtains an average reward of 2000 (or above) over 100 consecutive episodes**. With the right algorithm and optimal parametric settings, your agent can get a reward as higher than 2600 over 100 consecutive episodes.

## 2 Work structure

To solve the proposed task you will have to follow the steps below:

- Train and test at least two different DRL agents (using two of the algorithms provided in Stable Baselines3 [3], e.g. PPO [4], DQN [5], A2C [6], etc) on the Knapsack-v2 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.

### 2.1 Submission details

The project must be implemented using Python 3.7+ and PyTorch 1.10.0+. The submission folder of the project should contain the following files:

- Python file, containing:
  - Code (well-commented) for training and testing <sup>1</sup> the DRL agents (8 points);
- Trained model weights of the two agents;
- Report file, containing:
  - Brief description of the environment including state, actions and rewards (2 point);
  - Explain the chosen algorithms and motivate your choices (2 points);
  - Details of tuned hyperparameters such as epsilon (exploration/exploitation trade-off), learning rate and discount factor and their impact on the models' performances (2 points);
  - Discussion and visualization of the agents' training over the timesteps and average accumulated reward (2 points);
  - For each agent, the indication of the highest gained rewards and number of training timesteps (2 point);
  - Explanation of which agent performed better and why (2 point).

---

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

### 3 General indications

The files described must be submitted in a zip file containing one single directory, as shown in Figure 2. The naming convention for the directory is "Groupnumber\_1BM120\_A3" (e.g. Group00\_1BM120\_A3). The maximum length of the report is **4 pages**, excluding a title page and optional references.




Name	Date modified	Type
 trained_models	23/05/2022 13:27	File folder
 Group00_1BM120_A3.py	23/05/2022 13:28	Python File
 Group00_1BM120_A3_report.pdf	23/05/2022 13:29	Adobe Acrobat Docum...

Figure 2: Guide for files names and submission directory structure

### 4 Tips for getting started

1. To install Gym and OR-Gym simply use pip:

```
pip install gym==0.21.0
pip install or-gym==0.5.0
```

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. 2023)
np.random.seed(seed)
```

3. Import the library and make an instance of the environment:

```
import or_gym
env = or_gym.make("Knapsack-v2", max_weight=300, mask=False)
```

Setting the `mask` parameter to `False` prevents the OR-GYM environment to return additional information as a part of the state, which could be confusing.

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

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

5. To install Stable Baselines3 use pip:

```
pip install stable-baselines3
```

## 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] Christian D. Hubbs, Hector D. Perez, Owais Sarwar, Nikolaos V. Sahinidis, Ignacio E. Grossmann, and John M. Wassick. Environments for or and rl research. <https://github.com/hubbs5/or-gym>.
- [3] 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.
- [4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.
- [5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fiedjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.
- [6] 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.

## A Assignment Rubric (20 points)

### A.1 Part 2 (20 points)

Training and testing code (5 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 (3 points)	The trained model weights of successful agents are given. A successful agent must achieve an average reward of 2000 in 100 executive 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 (2 points)	Plots of the average accumulated reward over timesteps are provided for both agents.
Results (2 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 (2 points)	The report discusses about the differences among the results obtained by the two agents provided in the previous sections, highlighting which one performed better and explaining why this happens.