Task Sheet 4 (Capstone Task): Reinforcement–Learning Dispatch of a 1 MWh Battery

June 25, 2025

Motivation

High shares of variable renewable generation create pronounced and sometimes negative intraday price swings. Grid–scale batteries can earn revenue by *arbitraging* these spreads—charging when prices are low and discharging when prices are high. Reinforcement learning (RL) has become a promising technique for this sequential decision problem. In this project you will build, improve and benchmark an RL agent that operates a realistic lithium–ion battery on historical spot prices.

Scenario and Data

For this task, you are the operator of a 1 MW battery who wants to make use of the vast price differences observed in electricity markets. As you are situated in Austria, you will be using the Austrian 15–minute day–ahead electricity prices from **2019–2024** for Austria located in the **Central European Time (CET)** zone. The file format is

```
DateTime(UTC),Price[Currency/MWh]
2024-12-31 22:45:00,20.32
2024-12-31 22:30:00,44.97
```

As you are eager to know whether it is economically viable to operate your battery, you have made yourself familiar with the following constraints of your battery:

Parameter	**Value**	**Meaning**	**Rationale**
Usable energy	1 MWh	The battery cannot have more than 1 MWh stored	Typical containerised Li–ion system
Power limit	0.5 MW charge/discharge	You can only move up to 0.5 MW in an hour	2 h duration; cap drives timing decisions
Round–trip efficiency	90 %	A charge/discharge consumes/produces only 90 percent of its bought/sold energy	Includes inverter and thermal losses
Degradation cost	20 EUR/MWh throughput	Charging/Discharging one MWh throughput costs 20 EUR	$\approx 5\,\%$ of CAPEX over 10 000 cycles
State–of–charge	0–100 %	The battery cannot have less than 0 percent energy or more than 100 percent	Soft boundary in environment

Given all of this information, you would like to know the ebst strategy when to turn your battery on to buy and store electricity and when it is time to sell it. Hence, you will program a RL agent using Q-Learning for a time frame of the whole data of your choice.

Your Tasks

T1. Baseline agent. Use and adjust the DQN skeleton of previous tasks to train a battery–trading policy on the historical price stream (train–test split and time window of your choice)[1]. Track the agents' profit over each episode and explain why it develops like that.

[1] You do not need to use the whole five year period, in fact it is highly recommended to shorten this period significantly to make sure you can train your model in time. Eventually, you should have at least 10000 observations in total.

T2. Benchmarks. Benchmark your model on the test set with the following two models:

(a) Random Buy and Sell. (Random Policy)

(b) Buying when market is negative, selling when positive.

Both benchmarks must also incorporate the constraints.

T3. Evaluation. Evaluate in-depth and over time, when your agent decides to take what action, what the battery status is and what the current cumulated profit is using the test data. Interpret your findings. Compare this with the results from your benchmarks. Why is your agent better or worse in terms of overall profit (for the test data)?

T4. Mandatory enhancement. Try to improve your model by understanding and implementing one of the following and report its impact:

- Double–DQN (DDQN) *or*
- Prioritised Experience Replay (PER) *or*
- Inclusion of a sophisticated and realistic forecasting model[2].

You do not need to repeat tasks 2 and 3, just give information on what you have done and why it helped (or did not).

T5. Store your weights. Save the trained network parameters of T1 and T4 to an external file (`model_weights.h5` or `.rds`).

Deliverables (one `zip` per group)

1. **main.R** (code for solving tasks).
2. **model_weights.*** (saved weights, one for T1, one for T4).
3. **Report.pdf** (verbal answers, relevant code chunks, illustrations and tables for the tasks, ca. 8 pages).

Important Remarks

Solving this practical application of RL to electricity prices will come with some unexpected challenges. Electricity prices are a time series and should be treated as such. This is important especially when you plan to somehow alter the series itself. Also, computing this DQN on a larger dataset will cause complications on RAM, VRAM and CPU or GPU time. Make sure that your code is written efficiently and does not crash a computer. We can offer each group up to 24 hours of training time on our RTX 4090 at the department (24 GB VRAM). To do so, please send your code to steinert@europa-uni.de. We can only accept code which runs without adjustments and does not cause any issues, such as memory leaks. A code, which runs for a while but fails mid training will still be deducted from your 24 hour time window. Hence, make sure to test your code if it can run on other computers. Our system: Ubuntu 22.04.4 LTS, RTX 4090, 32 GB RAM, GPU Driver: 535.183.01, CUDA Version: 12.2, Tensorflow (R) 2.16, Tensorflow (Python) 2.13.

[2] Do not download external data like solar power generation etc. for that. You need to build the forecasting model using the data given.