Optimizing Energy Storage with Reinforcement Learning

Emile Dhifallah (14044994) Wenhua Hu (2733267) Felix Nastar (2744996)

Group 13

1 Introduction

Reinforcement Learning (RL) methodologies allow an AI system, or agent, to learn from its environment through interactive trial-and-error. Classic model-free control methods, such as Q-learning [1] use direct feedback from their environment, after performing an action, in order to update their future strategy. While classic RL research environments use simple reward definitions, it has also been established that more intricate reward structures can bring merit to the agent's performance. Reward shaping techniques such as potential-based reward shaping [2] can provide theoretical guarantees on the enhancement of the agent's performance when certain constraints are met. Moreover, it has been shown that slight changes in the environment formalization of an RL agent can influence the agent's final strategy greatly [3].

Utilizing this knowledge, we research the influence of both the reward signal and environment formalization on the performance of an agent that controls a car battery for the sake of energy storage optimization. The agent's goal is to learn a strategy that leads to maximal revenue (or minimal loss) by buying and selling energy optimally amidst varying electricity prices. It needs to do so while meeting constraints that keep the car battery charged to at least a minimum amount.

We construct tabular Q-learning agents, that through experimentation with the implementation of the agent and its reward structure, learn to respond well to the environment. The goal of our experimentation is twofold. Firstly, we are interested in the influence that different reward-shaping techniques have on our agent, and secondly, we aim to find out whether penalizing the agent up front for moves that are considered not-optimal, or illegal, helps it to learn better. Following these goals, we can formulate 2 research questions:

1. **Does potential-based reward shaping assist the learning process of our Q-agent?**
2. **Does penalization of bad actions assist the learning process of our Q-agent?**

Both factors of interest lead to an infusion of domain knowledge into the model, and as such, we hypothesize that both elements will positively enhance the agent's performance. We investigate this hypothesis through a series of ablation studies, where we let the Q-agent learn with and without the added elements. Furthermore, we perform ablations studies focused on the effect of different discretizations of the agent's state and action spaces on its performance, and the effect of the discount factor hyperparameter. By selecting the optimal set of (hyper-)parameters for our agent, its performance is enhanced by nearly 100% compared to the non-enhanced (baseline) Q-agent.

In order to grasp the implications of the experimental results, an introduction to the problem statement is appropriate. After this, the methodology and experimental setup will be described. Then, the results will be showcased and discussed, highlighting interesting or surprising findings and linking them to our hypothesis.

2 Formalization of the problem

It is crucial to clearly define the agent's environment and overall goal. The latter is to maximize the revenues of energy trading while making sure the car can drive. The former can be defined as follows.

2.1 Environment

The client has an EV battery with a usable capacity of 50kWh, which can charge or dischange with a power limit of 25kW. The battery reaches 90% efficiency during its operations, i.e. during electricity production and storage.

The agent has access to the current, and historical prices for 1MWh of energy on an (imaginary) energy trading market. It uses this to decide when to charge or discharge the EV battery.

An important constraint that we have is that, while the car is connected to the grid during each night, every morning at 8am it becomes unavailable until 6pm with a 50% chance. This represents the car driving, and upon return, it is left with 20kWh less charge. Therefore, It must have at least 20kWh charged every morning.

At the start of every hour, the agent decides to do nothing, charge or discharge the battery. The cost of buying is also constrained, as the agent pays 2 times the actual energy price as a representation of transaction costs. Selling happens at spot price. The buying action is optimally performed when electricity prices are low, and when prices are high, selling/discharging stored energy is optimal, as to create positive revenue.

The agent is encouraged to perform this described 'optimal' behavior through its reward signal. The basic signal can be formulated as follows:

$$reward_t = price_t * charge_t * E_t * T_t,$$

where $price_t$ indicates the current energy price for 1kWh, $charge_t$ indicates the amount of energy the agent decides to (dis)charge at timestep t, and E_t and T_t respectively indicate the efficiency factor and transaction cost corresponding to the action at timestep t (these are constants based on whether the agent buys or sells).

In order to devise a control strategy, the agent trains on 3 years of past electricity price data. The training dataset consists of hourly prices for the years 2007, 2008, and 2009. The agent's strategy will be evaluated on a validation dataset consisting of 2 years (2010 and 2011) of hourly electricity prices.

Next to price data, the agent has 2 more variables present in its state. These are the battery's current charge level, and the time of the day, providing a temporal signal to the agent. This means that, although the agent has 3 environment variables available to train on, there is no long-term temporal information in its state, which might hinder long-term success in its strategy formulation.

Energy prices contain seasonality on a daily basis, but also contain more long-term seasonality as prices vary between summer and winter months. Exploiting this variability is essential to learning a successful control strategy. Furthermore, the agent needs to balance battery level and trading, ensuring the EV's readiness for daily use. The presence of hourly seasonality in the data is showcased in Figure 1, where we see the average price per hour over 2 years of validation data. There is a visible trend in the fluctuation of prices, where prices are lower during nighttime, when people are asleep, and higher during daytime. Not plotted here, but also present in the data, are fluctuations between higher prices during fall and winter compared to summer, because these months are darker and colder with increased energy demand.

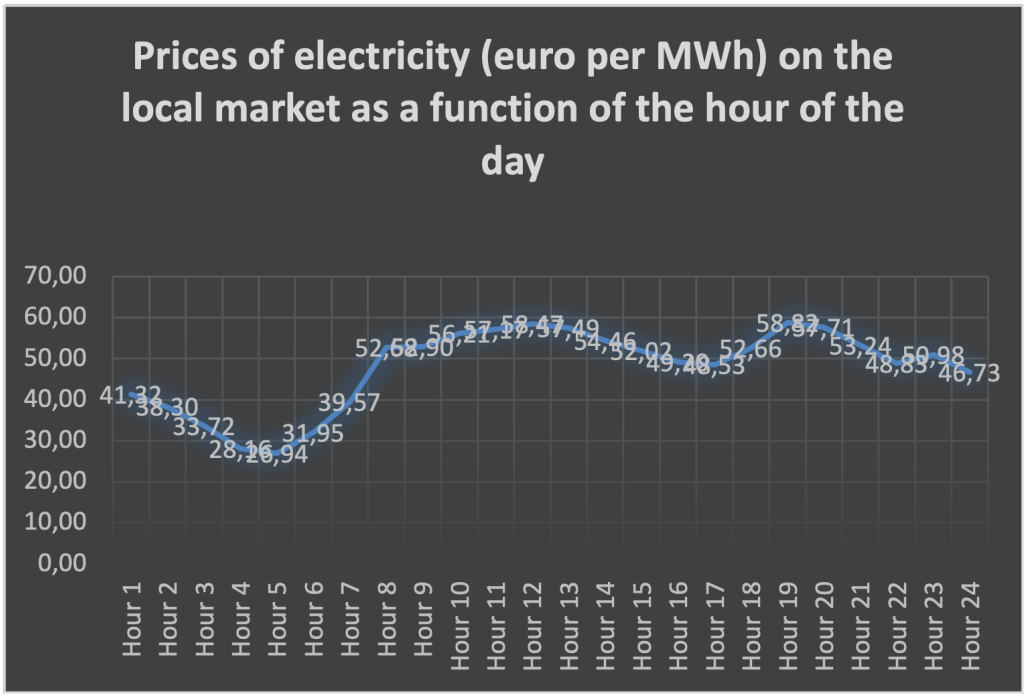

Figure 1: Price of electricity over hours

Besides seasonality, stochasticity in the availability of the car for energy trading combined with the 20 kWh charge minimum adds complexity agent's learned strategy. It poses a challenges to work under constrained choices, while still exhibiting beneficial behavior. This combined with the stochasticity of future energy prices implies that The agent needs to achieve some measure of robustness and generalization when learning to navigate its environment. While conventional RL methods are powerful in varying scenarios, it is interesting

to see the quality of such methods, like Q-learning, learn in an environment like this. We furthermore try to aid the model in this process by approaching the problem using more innovative approaches and more complex formalizations, which may help it navigate the dynamic scenario at hand effectively.

2.2 Reward shaping

Reward shaping in RL refers to the technique of modifying the reward function to aid the agent in learning a well-performing control strategy for its specific environment. Because reward shaping is usually inherently based on domain knowledge, the technique can furthermore enhance a strategy's robustness.
Potential-based reward shaping [2] involves adding supplementary rewards to the original reward signal, to guide the learning agent towards desired behavior or away from undesired behavior. It can be formulated as follows:

$reward_t = reward_t + F_t$,

where F_t represents an additional reward factor dependent on the current state and action chosen by the agent.

This technique can be particularly useful in complex environments where the agent receives very sparse or delayed feedback, or where desired behavior is not clearly defined. In our situation

In our environment, reward shaping can be put to use by steering the agent towards high charging levels in low-price situations, and vice versa. We formulate an initial reward shaping mechanism as follows:

$F_t = (price_t - price_{t-1)*(battery_t - battery_{t-1})}$.

The specific goal of this addition is firstly to encourage the agent to buy more after price decreases, and encourage selling after price increases. It should also encourage selling energy when the battery level is high, and vice versa.

2.3 Reward penalization

Another form of reward shaping is penalization, where the agent gets a high negative reward when it either chooses a bad action or is in a bad state. The quality of an action or state is again decided based on domain knowledge. We try to penalize the agent with a negative reward of -1000 (equal to e.g. buying 100kWh at a price of 10) whenever it buys energy when the battery already contains maximum charge, or attempts to sell energy when the battery is already (almost) empty. This addition is aimed at helping the agent avoid certain pre-determined bad courses of action.

3 Methods

3.1 Experimental setup

We implement the environment as described above. We use the following settings:

1 Data Source and Variables

- The agent has access to *train.xlsx*, which contains hourly electricity prices and dates;
- The variable *car_available* is encoded in the environment but not part of the state, and represents whether the car is currently connected to the grid.
- Environment variables, such as max power, battery charge/discharge efficiency, battery capacity and minimum morning charge, are encoded in the environment as constants.

2 Action space:

- The action space is a continuous range between -25 and 25, which is implemented as a normalized range between -1 and 1. An action value of -1 indicates selling full capacity, 1 indicates buying full capacity, and 0 represents taking no action (the actual amount of energy bought/sold is determined by scaling the action value with the power limit);
- The action space is discretized into either 3 or 5 bins, representing the actions -1/0/1 or -1/-0.5/0/0.5/1 respectively.

3 Observation/state space:

- *battery_level*: the current charge level of the battery, taking a value between 0 and 50. Battery levels are discretized in either 6 or 11 bins, representing interval lengths of 5 of 10 kWh respectively;

- *price*: The current electricity price per kWh, discretized into either 3 or 5 bins, which are determined by the corresponding quantiles, given by [1.0e-02 4.3e+01 1.5e+02] for the 3-valued version;
- *hour*: The current hour of the day, either discretized into 3 bins (each bin reprsenting 8 hours), or simply represented as 24 bins of each 1 hour.

4 Environment dynamics:

- The '*reset*' method initializes the environment to a starting state for each episode, which is at the start of the train/test timeseries. This method sets the initial time, day, battery level, and price;
- The '*step*' method updates the environment's state in response to an action taken by the agent. This includes updating the battery level, financial statistics, reward, and moving to the next time period.

3.2 Baseline

To establish a baseline for our model, we first created a *global* baseline which is based on random actions, and hovers around a total reward of -8200 for training and -5200 for testing. The other baseline we have is a simple Q-learning strategy, learned from the train set using a discount factor of 0, and no reward shaping or penalization. We furthermore use a default configuration to compare our modified experiment configurations against. This configuration is formulated below:

```
{
    "bin_size": {
        "battery": 11,
        "price": 3,
        "hour": 3,
        "action": 3
    },
    "properties": {
        "reward_shaping": 1,
        "penalties": 0,
        "nr_simulations": 400,
        "discount_rate": 0.95
    },
    "learning_rate": 0.10,
    "adaptive_epsilon": 1
}
```

3.3 Tabular Q-learning

Tabular Q-learning, known for its simplicity and effectiveness in environments with discrete, finite states, involves learning the value of actions in given states to maximize the total reward over time, and finding an optimal policy.

The Q-learning strategy is based on learning the values of this Q-table. Doing so, we form an agent that can map from any (discretized) state space to the optimal action to take in the next timestep. The agent learns by using a calculated balance between exploring new actions and exploiting known ones to maximize rewards.

Critical to this approach is state space discretization, i.e. discretizing of the *price*, *batterylevel*, and $hourof the day$ variables. Different configurations of discretizations are explored that all ensure efficient encoding inside of the Q-table, i.e. we avoid huge Q-table sizes.

Thus, The discretization into bins we set up for battery levels, prices, and hours simplifies the complex space into manageable segments, enabling efficient learning and decision-making by the RL agent.

3.4 Hyperparameters

We define an adaptive learning rate, based on the Adam optimizer [4], which proved to work better than linear learning rate schedulers in initial experiments. The discount factor, determining the importance of recent and long-ago timesteps respectively, plays a big role in our environment, as it proved influence learned strategies significantly. The discount rate directly influences the rate at which the agent updates Q-values based on new information and the extent to which it values future rewards over immediate ones. We vary the discount rate between 0 and 1. The default value is set at 0.95 during other experiments, which proved to be successful after initial experimentation. We furhermore set the same seed during all of our experiments to enhance

reproducibility of our experiments. Lastly, initial results (up to 2000 episodes) showed that the agent stops learning after around 300 simulations, and hence all experiments are capped at 400 simulations.

By iteratively refining these hyperparameters and the agent's exploration strategy, we experiment with the agent's abiulity to adapt to its environment optimally. This ultimately resulted in a policy that combines optimal setting of all available parameters and discretization options.

4 Results and Analysis

4.1 Influence of hyperparameters and discretization

Figure 4 in Appendix represents the influences of all listed hyperparameters and states discretizations. We can see at first glance that the agent abruptly stops learning, around simulation 200, after quickly improving rewards in the first 200 episodes.

Firstly, we can see that using more price bins benefits the agent greatly. At the same time, using 5 action bins instead of 3 action bins or 11 instead of 6 battery bins, does not help learning significantly. Even more so, using 24 hour bins instead of 3 negatively impacts performance greatly. This goes against our expectation, as we expected the number of bins to be positively correlated with total training reward.

Next, it can be seen that reward shaping leads to significant performance enhancement in the early stages of training, however once the agent finishes learning, differences in total reward between the techniques stagnate mostly. Comparing the effect of using penalties during training is difficult, as -1000 penalties naturally influence the training score of the agent greatly, as can be seen in the respective subplot. For this comparison, we need to therefore look at the test performances (see next section).

Lastly, we can see that the discount factor has a big impact on learning, with lower discount factors performing better than higher ones, especially in later stages of training. While this might seem surprising, a reason for this could be that the agent lacks long-term temporal state information, and thus benefits most from recent observations. Another reason could be that the reward shaping pushes the agent to depend on the previous state more heavily, which means that the agent benefits more from learning from short-term temporal information (i.e. the previous timestep).

4.2 Test performance

In Table 1, we can investigate the final test for every experiment configuration. As can be seen, all Q-agents perform superiorly to the random agent, and most also superiorly to the Q-learning baseline. Surprisingly, on the test set, higher discount factors lead to better performance, contrary to the training scenario. Both reward shaping and penalties benefit test performance, with the former having a bigger positive impact. Contrary to training, on the test set the agent benefits from more bins for both the battery and price level. Increasing action and hour bins here negatively impacts the agent a lot. We can also see that our best result across all variations is a total reward of €-485.10.

Model	Discount	Shaping	Penalty	Battery	Price	Hours	Actions	Reward
Qlearning	0.95	Yes	No	6	3	3	3	-659.31*
Discount	0.5	Yes	No	6	3	3	3	-586.17
	0.1	Yes	No	6	3	3	3	-607.84
	0.0	Yes	No	6	3	3	3	-1534.80
Shaping	0.95	No	No	6	3	3	3	-876.49
Penalty	0.95	Yes	Yes	6	3	3	3	-688.34
Battery	0.95	Yes	No	11	3	3	3	-578.41
Price	0.95	Yes	No	6	5	3	3	-566.298
Hours	0.95	Yes	No	6	3	24	3	-1253.62
Actions	0.95	Yes	No	6	3	3	5	-1018.91
Misc.	0.5	Yes	No	6	5	24	5	**-485.10**
	0.5	No	No	6	3	3	3	-923.74
	0.0	No	Yes	6	3	3	3	-837.86
Q-basel.	0.0	No	No	6	3	3	3	-949.46
Random	0.0	No	No	6	3	3	3	-5226.48

Table 1: Test Rewards for every combinations of experiment setting on the Q-learning agent. Discount indicates the discount rate of future reward; Shaping indicates whether reward shaping is used or not; Penalty indicates the use of reward penalties; Battery/Price/Hours/Actions indicate the number of bins used to discretize space of the respective state/action variable; Rewards gives the total reward over the test set (i.e. sum of all rewards).
** This represents the default experiment configuration; use this to compare to other experiments.*

4.3 Further improvements

Leading from our ablation experiments above, we configured experiments where we combine optimal settings for all customizable parameters but the penalties, also including increased bin sizes for all state discretizations. This agent, inside of the *Misc.* category of Table 1, performs superior to all previously seen agents. What exactly contributes to the success of this agent in particular is difficult to dissect, but it seems to be at least partially traceable to a balanced discount rate of 0.50, as 0.95 worked less well in an ablated experiment. Furthermore, its sucess can traced back to increased bin sizes too, as the combination of increased bins for every state and action variable achieved best performance overall. Surprising is that turning off penalties in this configuration further helped the agent to perform better on the test set, even though it consistently showed merit for total training reward.

4.4 Behavior of learned agent

In Figure 2, we see the behavior of the optimal learned agent as a function of the price and chosen actions. Over a subset of the test set, we can see slight patterns emerge, where the agent prefers to sell charge at high prices, and buy charge at low prices. This is what we expect from an optimal agent in our environment. However, another finding of this graph is that our agent is slightly "lazy" when the prices are in a mid-range, meaning that it does not perform a buy or sell action, when in optimal conditions we would want it to trade energy to increase revenue.

Lastly, we want to see the effect of training on the energy distribution in the agent's battery. In Figure 3, the battery level is plotted over the last 3 days of the training dataset, before and after training (i.e. during simulation 1, and 390). The actions taken are indicated by the color of the line. At first glance, both before (top subplot) and after (bottom subplot) might appear to feature approximately random action choices. However, we can actually see that the trained agent has higher overall charge during the night and early mornings, whereas the untrained agent has less high battery levels during these periods. This behavior is also visible through the rest of the trained agent's timesteps. This could be attributed to the agent learning that there needs to be sufficient charge during morning periods. Besides the temporal dimension, not much differentiation can be made between the agents, as both buy and sell at frequently throughout the day.

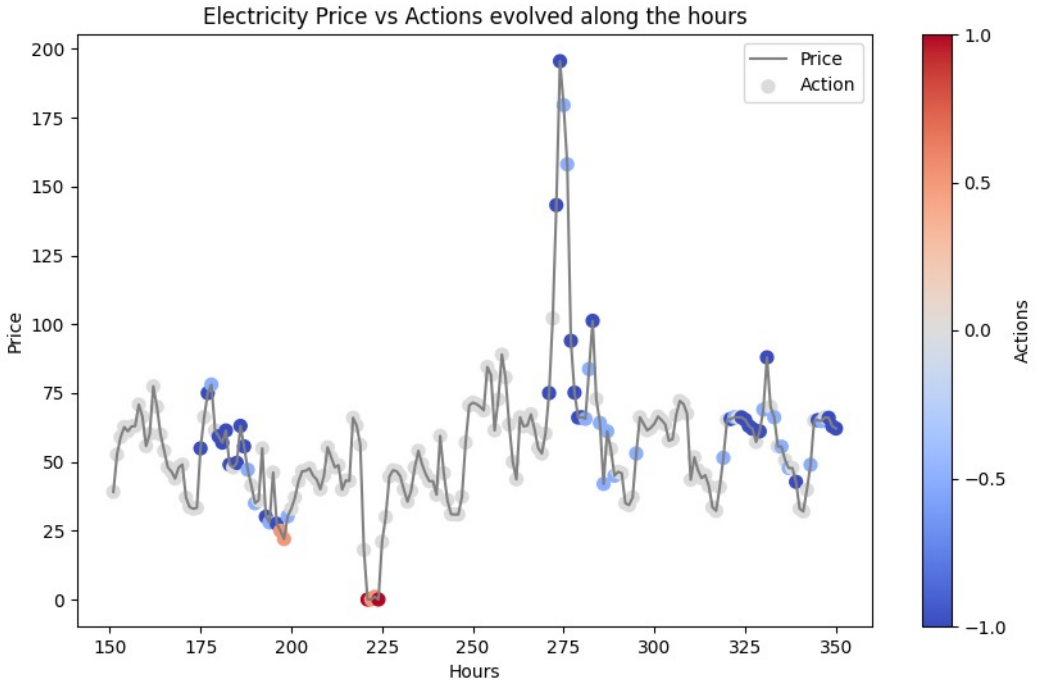

Figure 2: Illustration of the sequence of actions and of electricity prices with time (hours). A dot is associated to every market period and its color refers to the action taken.

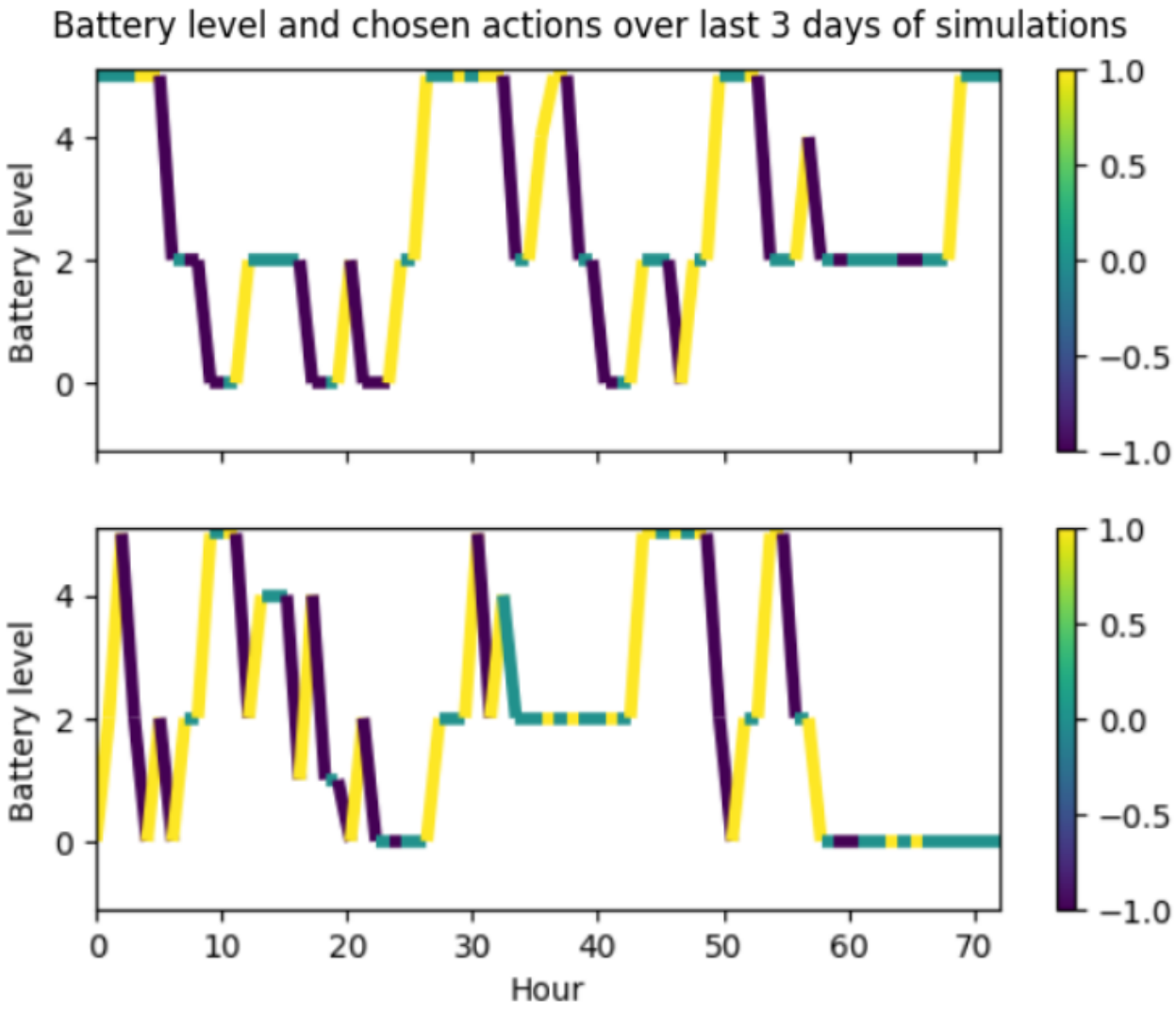

Figure 3: Visualization of the actions the agent chooses plotted against current battery level. The top plot is taken from the first simulation, where behavior is more or less random. The bottom plot is taken from simulation 390, when the agent is supposed to have learned until convergence. As we can see, the battery is unloaded more often, while there is also more variability in battery level overall.

5 Discussion

While some of the learned behaviors of the agent were in line with our expectations, other were not. We will discuss some of these, and other considerations that need to be taken when building RL agents like the one at hand, including stability of results, and potential improvements.

5.1 Answers to research questions

Coming back to the questions that lie at the basis of our experimentation, we can verify that a Q-learning agent, trained until convergence in the battery storage environment as formulated above, indeed benefits from potential-based reward shaping. Furthermore, reward shaping in the form of heavy penalization for unwanted states and actions benefits the Q-agent. These claims are based on the ablated experiment configurations, where we have seen that both elements by themselves improve the default agent. This is in line with our hypothesis. However, what turned out to be surprising, is that the combined configuration, where non-optimal settings like big bin sizes and no penalization, led to the best performing Q-agent among all configurations. While not in line with training runs, we did expect bigger bin sizes for discretization to benefit the agent, due to having more flexibility for calculating next timesteps, e.g. not being constrained to either buying or selling the full 25kW at once.

Despite some shortcomings due to computing efficiency, version control troubles and other hindrances, we managed to learn how a Reinforcement Learning model can be set up and improved iteratively through ablation analysis and various experiments and tweaks.

Deep Q-Networks (DQNs) could potentially perform better than traditional Q-learning here due to their ability to handle large or continuous state spaces through function approximation. DQNs use neural networks to approximate the Q-value function, allowing them to generalize over a much wider range of states than tabular Q-learning, which is limited by the need to discretize the state space. This makes DQNs more suitable for complex environments where accurately representing the state space and the action value function with a simple table is impractical.

It could be useful to run the algorithm with less or more features and see how the agent reacts. 3-4 meaningful features make up smart feature engineering. It could also be beneficial to have a forecast of the prices via Deep Learning, the agent would then be able to make a more informed decision.

RL algorithms can learn and adapt strategies over time to maximize rewards, such as profit from buying and selling electricity. Given the fluctuating nature of energy prices and consumption patterns, RL is suitable for developing policies that can anticipate and react to these changes efficiently, optimizing energy usage and financial returns based on real-time data and long-term outcomes.

Other methods besides Reinforcement Learning can be used including traditional optimization techniques, rule-based algorithms, and machine learning models such as regression, decision trees, and neural networks. These methods can analyze historical data to predict energy prices and consumption patterns, optimizing decisions accordingly. However, they might not adapt as dynamically to new information or optimize sequential decisions over time as effectively as RL.

References

[1] Christopher Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8:279–292, 05 1992.

[2] E. Wiewiora. Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19:205–208, September 2003.

[3] Theresa Eimer, Carolin Benjamins, and Marius Lindauer. Hyperparameters in contextual rl are highly situational, 2022.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

6 Appendix

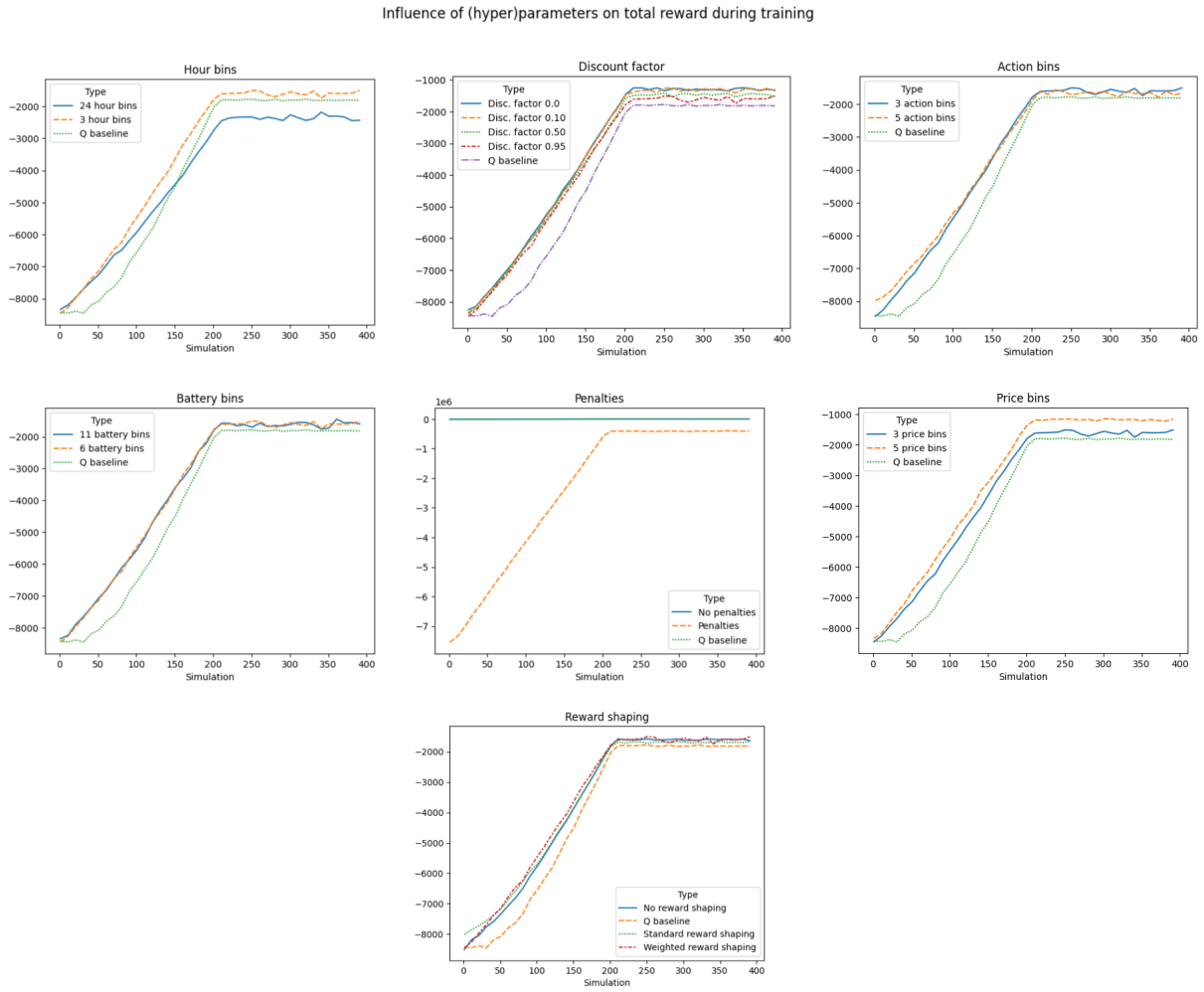

Figure 4: Window containing subplots showing the influence of 7 different environment choices and hyperparameters. All experiments ran for 400 simulations, and average total reward is shown per 10 simulations.