Henrik Ytrehus Ågotnes

Hydropower optimization using model-based Reinforcement Learning

Masteroppgave i Master of Science in Informatics
Veileder: Helge Langseth
Juni 2023

Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Henrik Ytrehus Ågotnes

Hydropower optimization using model-based Reinforcement Learning

Masteroppgave i Master of Science in Informatics
Veileder: Helge Langseth
Juni 2023

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Abstract

One of the most effective ways to store energy is by scheduling hydropower plants to produce when it is most beneficial and reserve the water in other periods. In the Nordic power market, hydropower operators try to optimize profits.

This is a challenging problem involving non-linear dynamics, uncertainty and operating constraints. Traditional methods typically use variants of linear programming to solve the problem, but these methods can have high running times. The Nordic power market is transitioning to become closer to real-time, which increases the computational burden on traditional methods. In this thesis, we try to find alternative methods with a different trade-off between solution quality and running time. The running time needs to be lower, and ideally, the solution quality should not be adversely affected.

We make the observation that our model of the environment is differentiable with some non-continuous points, which allows us to compute the gradient of the profit with regard to the actions. We can then do gradient ascent to optimize actions for an episode. To deal with the non-continuous points we first use various techniques such as Monte Carlo tree search and the cross-entropy method in an attempt to get close to the global maximum.

We find that this method has a running time within the given time constraint and a better solution quality than some state-of-the-art methods. However, we were unable to compare the performance to production systems or the global optimum. Consequently, the exact extent of the solution's quality remains unknown.

Keywords: Model-based reinforcement learning, hydropower optimization, Monte Carlo Tree Search (MCTS), trajectory-optimization, cross-entropy method

Preface

This thesis concludes my Master of Science in Informatics degree for the Department of Computer Science at the Norwegian University of Technology and Science (NTNU). The work was performed between autumn 2022 and spring 2023. I would like to extend my gratitude to my supervisor, who provided guidance and expertise. In addition, I would like to thank employees at Trønder Energi who helped with domain knowledge, data and advice.

Henrik Ytrehus Ågotnes

Trondheim, 11th June 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We start the chapter by giving some background information about hydropower scheduling and our motivation for this project in Section 1.1. In Section 1.2 we define our research goal and three research questions that can help achieve the goal. The main contributions are listed in Section 1.3 and the structure of the thesis is laid out in Section 1.4.

1.1 Background and Motivation

Hydropower is a way to generate electrical energy from flowing water. It is the most used energy source in Norway, with 136.7 terawatt-hours of yearly production, out of a total of 156.1 terawatt-hours in a normal year (NVE, 2023). The hydropower scheduling problem involves deciding when to utilize water resources for power generation, and when to save the water for future use. The goal of the problem is to maximize profits by finding the optimal generation schedule, which specifies how much water will be released for each time step in a planning horizon. To decide future schedules operators rely on uncertain forecasts about future prices and how much water will flow into the reservoir. There are also operating constraints.

Power prices are highly volatile and fluctuate hourly, offering opportunities for hydropower operators to achieve a higher realized price. As shown in Figure 1.1, power prices in the first half of September 2022 varied widely, ranging from below €30 to over €500 per unit. This volatility can significantly impact the revenue of the operator, potentially by more than a factor of 10. Consequently, choosing an optimal generation schedule can lead to large increases in profit for the owner of the hydropower plant. Because hydropower is such a large part of Norwegian power generation, increasing the profits by a tiny factor can give large increases in profits.

Better hydropower scheduling also benefits consumers by ensuring a stable supply and price of power. This is because a high price in a particular hour signals that power is relatively scarce that hour, either because of high demand or a low supply of power. A scheduler that reacts to the high price by increasing power production will increase supply which can lead to lower prices at that hour. With intermittent power

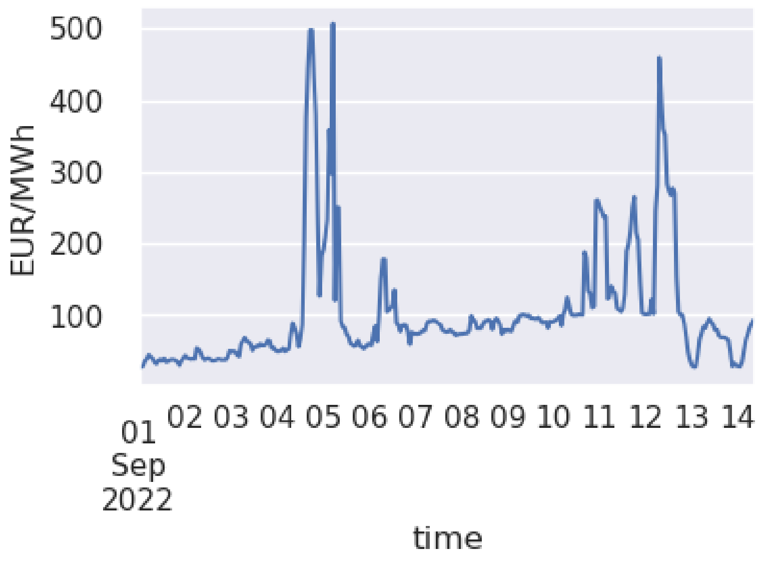

Figure 1.1: Power price in September 2022 (Trønder Energi, 2022)

sources like solar and wind projected to increase to avoid carbon emissions (IEA, 2022), the supply of power will have more volatility, which creates more volatile prices. Hydropower can provide backup power during times of low wind or solar generation, and save water during periods of high wind or solar generation. In this scenario, hydropower takes the role of a battery to create a more stable supply of power. Graabak et al. (2017) and Clarke et al. (2022) identify this approach as important to reduce reliance on fossil fuels.

Power producers currently use optimization techniques that find good solutions but have slow running times. Trønder Energi is interested in finding methods that have lower running time without substantially reducing the quality of the solutions. When the power market changes from an hourly time resolution to a new price every 15 minutes, the problem size increases exponentially. Employees at Trønder Energi worry that traditional methods might not scale to this added complexity.

1.2 Goals and Research Questions

Goal Find solutions to the short-term hydropower scheduling problem that balances computation time and solution quality

The main purpose of this thesis is to find methods from the field of Artificial Intelligence that can be used to solve the hydropower scheduling problem. The methods should approach the decision quality of traditional methods, but in addition, have a lower running time. In discussions with employees at Trønder Energi, they have stated that differences between computation times below 2 minutes do not matter. Increasing solution quality at the cost of increased running time is worthwhile when the running time is below this threshold. Consequently, we set 2 minutes as the maximum running time for our method. The methods should be compared on computation time and solution quality as measured by profit when operating over a certain time period.

Research question 1 What techniques have been used to solve continuous control problems with uncertainty?

To solve our goal, we will survey methods that have been used for similar problems. The hydropower scheduling problem is part of a class of problems called optimal control under uncertainty. The problem can be broken into subproblems such as dealing with different types of uncertainty, dealing with constraints, incorporating known dynamics of a process into the learning system and dealing with limited data. Finding relevant approaches and avoiding pitfalls is an important first step in the research process.

Research question 2 Which techniques exhibit the best fit for addressing the hydropower scheduling problem?

Based on the techniques discussed in Research Question 1, we are interested in identifying which are most suitable for the hydropower scheduling problem. To answer this question we need to discuss properties of the hydropower problem and find methods that have successfully addressed problems with similar characteristics.

Research question 3 Is it possible to create a data analysis pipeline that is competitive with state-of-the-art methods while subject to constraints on the running time?

To address this question, we need to create or choose a promising method and evaluate the technique. The evaluation should be compared to methods from State of the Art to give a reasonable baseline. In our comparison, we are interested in how much profit a method generates over a time period, and how much running time a trained system uses to arrive at its decisions. We are interested in different trade-offs between running time and solution quality, but in general, we prefer an improvement of solution quality on the cost of running time.

1.3 Contributions

We make the following contributions:

1. We survey techniques for continuous control under uncertainty. These include model-free reinforcement learning like the soft actor-critic, the cross-entropy method, heuristic search and differentiable simulators.
2. We address the suitability of these techniques for the hydropower problem.

3. We design a method to optimize the hydropower scheduling problem using a combination of heuristic search and a differentiable simulator.

4. We evaluate this approach and show that it performs better on the evaluation than a popular model-free reinforcement learning algorithm called Soft actor-critic, in addition to a model-based approach called the cross-entropy method.

1.4 Thesis Structure

This thesis is structured as follows:

Chapter 2 - Background: This chapter introduces relevant background information that can be useful to understand the rest of this thesis. Firstly, we explain the hydropower scheduling problem and introduce relevant domain knowledge. Secondly, we discuss various artificial intelligence techniques like supervised learning and reinforcement learning. Finally, we explain the motivation behind the project.

Chapter 3 Related literature: Surveys some more recent methods that build on the concepts in Chapter 2. The selection criteria for methods is that they are used on problems with similar characteristics as the hydropower problem.

Chapter 4 Architecture: Based on our problem specification and our findings from related literature, we explain our choice of methods to answer the research questions. We explain some assumptions we made when modeling the problem, show how we adapted the Soft Actor-Critic to our problem, and also explain our main contribution in this thesis, a data analysis pipeline using various ideas from the literature on reinforcement learning.

Chapter 5 Experiments and results: We show the experimental plan to answer Research Question 3, and also provide implementation details and our experimental setup. The last section gives the results of our experiments.

Chapter 6 Evaluation and Conclusion: Finally, We summarize the main points and answer our research questions. We also discuss limitations and potential future work.

Chapter 2

Background Theory and Motivation

This chapter introduces some background knowledge that will be required to understand the results of this thesis. Section 2.1 gives an introduction to domain knowledge about the hydropower scheduling problem. Then Section 2.2 discusses some relevant techniques in the field of artificial intelligence. Lastly, Section 2.3 describes the motivation behind the project.

2.1 Hydropower and the Nordic power market

This section introduces background knowledge about hydropower. Section 2.1.1 gives an overview of how hydropower plants work and how to model them. The main result from this section is the operating constraints of the hydropower plant and the relation between discharge and how much power is generated. Next, a description of how the Nordic power market sets the power price is given in Section 2.1.2. The hydropower scheduling problem is explained in Section 2.1.3. The primary takeaway from this section is the profit equation, which is the optimization target of the thesis. Lastly, in Section 2.1.4 we turn our attention to the role of uncertainty in the problem. We introduce an objective function that uses uncertain data and also explain the intuition behind multistage optimization, a more complex objective that is not used in this thesis.

2.1.1 Hydropower

Hydropower uses a turbine to convert the mechanical energy of moving water to electrical energy. Since gravity pulls on water, water has potential energy. The higher the elevation of the water and the more mass of water, the more potential energy is stored.

There are different types of hydropower plants. The most common in Norway is

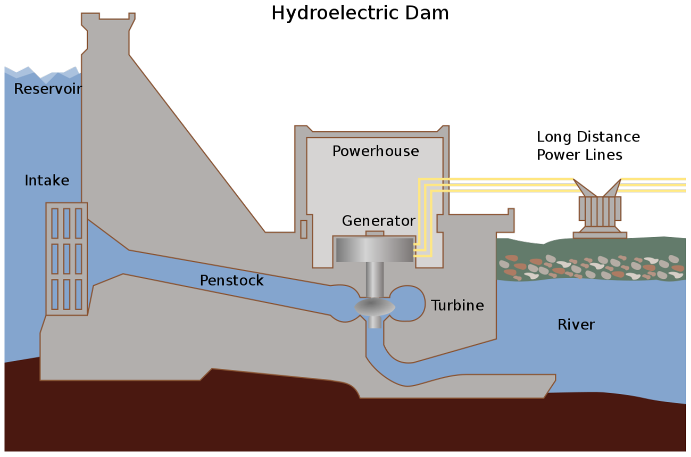

Figure 2.1: Hydropower plant (Energy Information Administration, 2022)

storage hydropower plants, which have a dam and a reservoir that can accumulate water. In this thesis, we model a hydropower plant called Mørre, which is of this type. Figure 2.1 shows some of the components of a hydropower dam. A reservoir of water can be released into the penstock to drive a turbine. A generator converts the movement of the turbine to electrical energy, which is delivered to the power grid.

The amount of power generated is measured in megawatt hours (MWh) and depends on the plant and a number of other factors. For reference, Mørre power station can produce between 6 to 14 MWh in an hour, and in 2020 produced 68 GWh during the year. The potential energy of water can be calculated by multiplying gravitational acceleration ($g = 9.81 m/s^2$), the height of the water above the turbine ($h(x_t)$), and the mass of the water. The mass of the water is given by the number of cubic meters of water flowing through each second (d_t measured in m^3/s) and the water density, ρ, around $1000 kg/m^3$.

The height of the water varies with the reservoir filling, x_t. When the reservoir water level is high, more power can be generated per cubic meter of water. This is called head and allows power producers with a relatively full reservoir to generate slightly more power per unit of water. For our model of Mørre, a full reservoir gives 6.2% more power than an almost empty reservoir.

The operators of the dam decide how much water goes out of the reservoir. Released water that spins a turbine is called discharge, d_t, and will generate electricity. This is the only mechanism for controlling the hydropower scheduling. Sometimes operators release water without generating power which is called spillage. This is not included in our equation for simplicity.

The amount of power generated does not depend linearly on the amount of discharge. The friction loss coefficient, γ, depends on the squared discharge. This friction loss makes the power generated a nonlinear function of the discharge. As discharge increases, less power is generated per cubic meter of water.

The power generated can be approximated by Equation 2.1, where w_t is the gen-

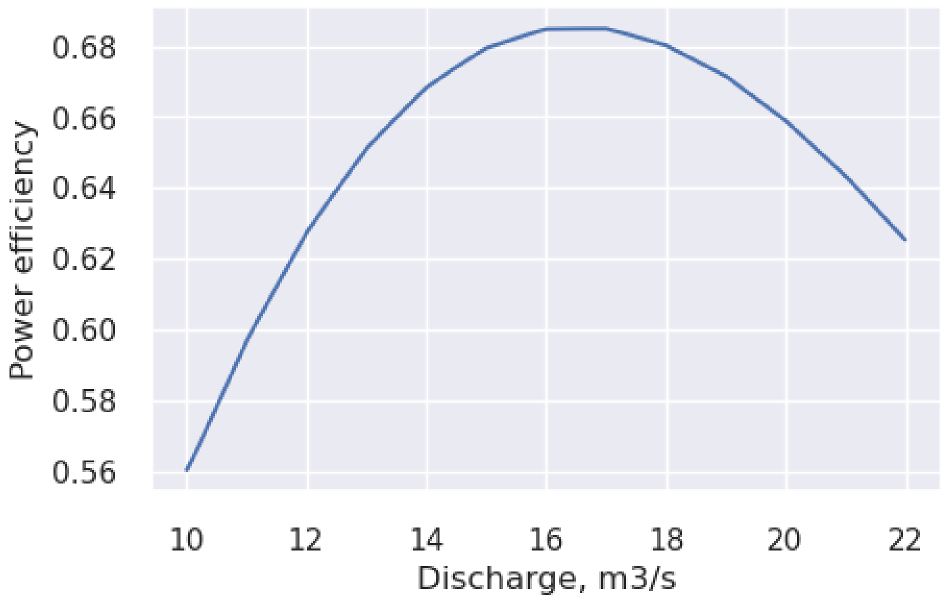

Figure 2.2: The figure shows the relation between the discharge decision, d, and the power produced per unit of water, given by w/d. The result is based on parameters specific for Mørre, and with filling at 50% of maximum. (Trønder Energi, 2022).

erated power measured in MWh. η is the turbine efficiency, which is a function of the discharge d_t. This function is based on a linear interpolation of points provided by Trønder Energi and has a maximum efficiency at around $17m^3/s$. When the turbine takes kinetic energy from the moving water, the water is slowed down. A hypothetical turbine with a 100% efficiency would stop the movement of the water, which is not possible. κ is a time step conversion factor to convert energy measured in watts to power measured in MWh, which for an hourly resolution is given by $60^2 \times 10^{-6}$. The relation between the discharge decision and the efficiency of the power production for Mørre power station is shown in Figure 2.2. The unit of power efficiency is MWH per m^3/s. We see that Mørre is efficient with discharge around 16 to 17 m^3/s, as this gives the most power generated per unit of water.

The reservoir water volume, x_t, will change by the difference between inflow and discharge as seen in Equation 2.2. q_t is the inflow, which is the amount of water flowing into the reservoir, and can come from sources such as rain, rivers and melting snow. We also need to convert inflow and discharge from m^3/s to the time resolution of the decision, so we use the time conversion factor δ. In the case of an hourly resolution, $\delta = 60^2$. The Norwegian Water Resources and Energy Directorate constrains operation rules for hydropower plants to prevent environmental damage (NVE, 2022). Plants consequently have a minimum and maximum reservoir water level that should not be violated. Equation 2.4 expresses this constraint, where x_t is the filling of the reservoir. The consequences for breaking the minimum reservoir level are worse, as this can have environmental damage. If the reservoir level exceeds the maximum threshold, $x_t > x_{\max}$, then the excess water is lost as spillage. Because both reservoir water level and the amount of discharge affect the power efficiency of a unit of water, there is an opportunity for maximizing the power produced. Much of the literature described in Section 3 tries to achieve this goal while ignoring prices.

Operators are also limited in how much discharge they can release, which leads to the constraint in Equation 2.3. For Mørre, the power operator can either choose 0 discharge, or between $10m^3/s$ and $22m^3/s$.

The equations were adapted from Matheussen et al. (2019), with some modifications.

$$w_t = \kappa\eta(d_t)\rho g(h(x_t) - \gamma d_t^2)d_t \tag{2.1}$$

$$x_t = \min(x_{t-1} + \delta(q_t - d_t), x_{\max}) \tag{2.2}$$

$$\begin{cases} d_{min} \leq d_t \leq d_{max}, \\ d_t = 0 \end{cases} \tag{2.3}$$

$$x_{min} \leq x_t \leq x_{max} \tag{2.4}$$

Where:

w_t**:** Power produced (MWh) at time t

d_t**:** Discharge m^3/s

q_t**:** Inflow m^3/s

x_t**:** Filling in reservoir m^3

κ**:** Unit conversion, $60^2 \times 10^{-6}$ for hourly resolution

η**:** Turbine efficiency

ρ**:** Water density, $1000kg/m^3$

γ**:** Friction loss coefficient

g**:** Gravity 9.8 m/s^2

$h(x_t)$**:** Height of water over turbine, m

δ Conversion factor, 60^2 to convert from cubic meters per second (m^37s) to cubic meters per hour.

2.1.2 Nordic power markets

The power market is designed to set a price that matches the amount of power produced to the amount of power consumed. Nord Pool organizes the power exchange and enforces rules and regulations to ensure an efficient power market.

The following procedure is used to determine the spot price. Producers in the power market submit how much power they want to sell at different prices, and consumers

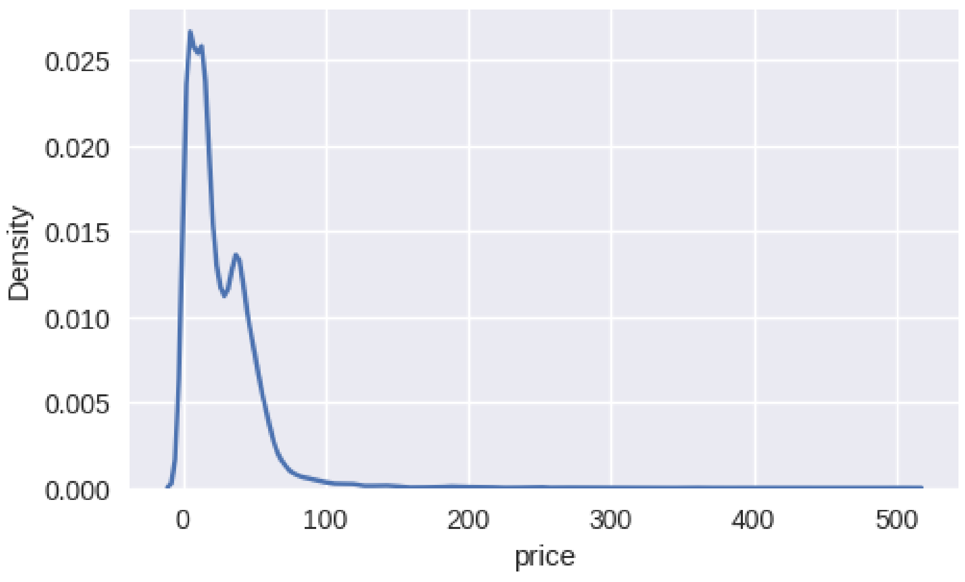

Figure 2.3: The distribution of the power price from September 2019 to September 2022, in NO3. The price is measured in EUR/MWh (Trønder Energi, 2022)

submit how much they want to buy at different prices. The market-wide supply and demand are found by aggregating the individual bids. The power price is where demand equals supply, and all actors who were willing to buy above this price or sell below this price will have their orders accepted.

When a hydropower producer decides how much power to sell, he must consider future prices. Analysts and domain experts try to forecast power prices. The forecast uses many uncertain factors like demand patterns, the price expectations of other producers, weather and other things. Power companies like Trønder Energi rely on an ensemble of possible future prices, which are called scenarios. These scenarios can be produced in-house or bought from companies such as SKM Market Predictor which specializes in forecasts and analysis of the power market.

The price can be modeled using a stochastic, heavy-tailed distribution. Figure 2.3 shows the distribution of the power price over three years. The tail of the distribution is very long, which means that the probability of sampling values very far from the mean is very large compared to a normal distribution. The unbiased skewness is around 4.5, which also indicates that the distribution is highly asymmetrical. This heavy-tailed distribution is a common feature of power prices Powell, 2014.

There are a few different power market exchanges. In the day-ahead market, actors submit bids for each hour on the next day. Other exchanges such as intraday and the regulating market allow bidding closer to delivery time and can have different prices to the day-ahead market. This can affect the optimal scheduling, but for simplicity, this thesis only considers optimizing profits for the day-ahead market. The power market is also divided into many areas, where Norway has 5 areas, and the area of Mørre power station is called NO3.

Changing the scheduling can affect the price. If a power producer decides to produce more power in a specific hour, the supply of power increases and more buy bids for lower prices can be met. The price a firm gets could therefore decrease as a consequence of their bidding choices. It is difficult to model how much the prices will change, so we make the assumption that the power producer is a price-taker, meaning that a single producer's schedule does not affect the price. This assumption is more reasonable when the hydropower plant has a small share of market production, which is the case for Mørre.

In summary, the power price depends on a complex interplay between many actors and uncertain factors.

2.1.3 The hydropower scheduling problem

A schedule is a sequence of discharge decisions, one for each time step in the planning horizon T. The schedule, S, is given by $S = (d_t)_{t=1}^T$.

The objective of the deterministic optimization problem is to find a schedule that maximizes the profit V, given by Equation 2.5:

$$V(\vec{d}) = \sum_{t=1}^{T}(p_t w_t - c_t) + I(x_{T+1}, e_{T+1}) \tag{2.5}$$

$$c_t = \begin{cases} 10^6, & \text{if } x_t < x_{min} \\ 110, & \text{else if } d_{t-1} = 0 \wedge d_t > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

For each time step in the planning horizon, T, the profit is given by p_t, the power price, times the quantity of power produced, w_t. The relation between dt and w_t was given in Equation 2.1, and how d_t affects the reservoir level was given in Equation 2.2. An explanation of how the power price, p_t is formed was given in 2.1.2. In addition, there is a cost, c_t, if the generator was previously turned off or the constraints for x_t were broken. We also remove the revenue for time steps where the lower reservoir was broken, and only the punishment -10^6 is given for that time step. Values specifically for Mørre power plant are shown in Equation 2.6.

In addition, the value of the remaining water at the end of the planning horizon is given by I, which depends on the amount of water left in the reservoir and the expected price of this water in the future. If a rest price was not included, then a profit-maximizing agent would always want to empty the reservoir at the end of the planning horizon, which would not be desirable. Trønder Energi uses a medium-term model which plans with a longer time horizon to determine the value of the rest of the water. This model outputs cut files that contain estimates of the value of the remaining water.

To give some intuition for the relationship between actions and profit, we will now consider Figure 2.4. The figure shows how changing a single action affects the profit. The red points and the blue line are all legal decisions. In this case, the price is 10% higher than the rest price, and we see that to achieve the highest profit, the model should produce at around $17m^3/s$. The shape of the curve between 10 and

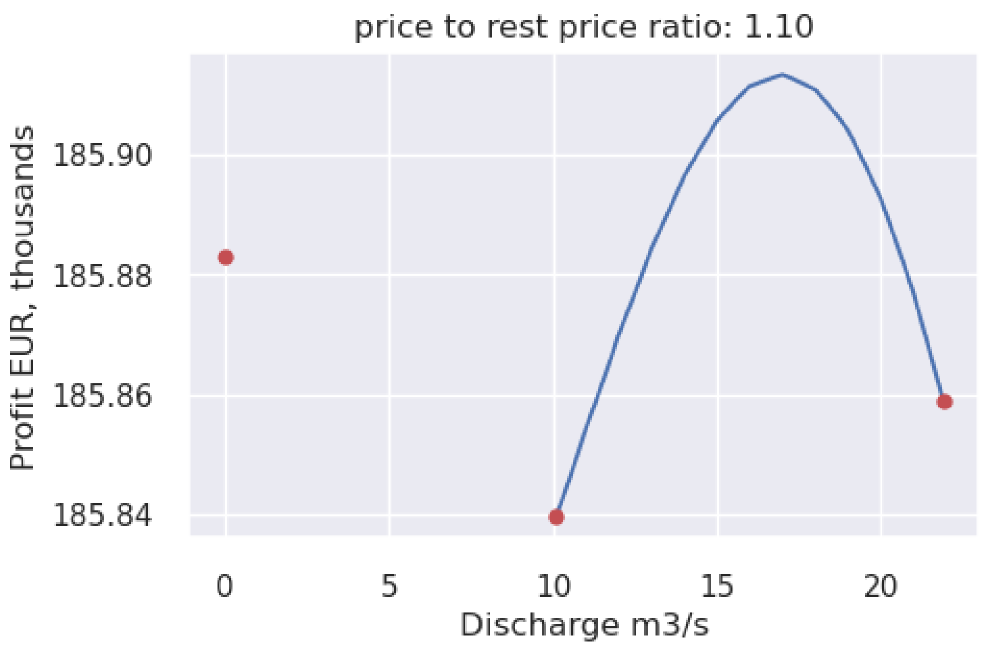

Figure 2.4: The figure shows the profit for various discharge decisions. The discharge decision is for one specific timestep, and all other decisions are constant. The price is 10% higher than the rest price.

22 is mostly due to the power efficiency at various levels of discharge, illustrated in Figure 2.2. Turning on production, from the first red point to the second, leads to a reduction in profit because power production is very inefficient at $10m^3/s$, so the revenue from this production is lower than the revenue lost from the rest value. The same is true for discharge decisions close to the maximum, where the efficiency falls again. Another small effect is that the lost water reduces head for all future timesteps, which gives some reduced profit. In addition, the startup cost reduces the profit by €110 in this figure because the previous action was nonzero. When the price to rest price ratio is lower, it is often beneficial to not produce at all. In the opposite case, when the price is much higher than the rest price, the slope of the curve changes, and it may be optimal to use maximum discharge. Finally, if the single action can lead to a violation of the reservoir filling constraints, then there can be discontinuities in the curve.

The planning horizon, T, is usually around 12 days. Bidding in the day-ahead market only requires submitting a schedule for the next 24 hours, but the optimal decision depends on future decisions, which is why a longer planning horizon is used. For instance, if prices are expected to be high for the next two weeks, an optimal schedule for the first 24 hours would be more likely to save water. For a planning horizon of 12 days and an hourly resolution, $24 \times 12 = 288$ discharge decisions have to be considered. A brute force search with the discharge discretized to n values would require checking n^{288} combinations. When the power market changes to 15 minutes resolution, the problem becomes even more computationally heavy, with $n^{288\times4}$ possible combinations.

2.1.4 Modeling uncertainty

We get a stochastic optimization problem by introducing uncertain inflow and price forecasts. The true p_t in Equation 2.5 and the true q_t in Equation 2.2 are both unknown at the time of planning, so the planner must instead rely on uncertain forecasts of these variables. We model the forecasts as scenarios, where a sequence of future prices from time $t = 0$ to $t = T$ occur with some probability. These predictions are made using information available at time f. The notation $\hat{p}_{t,s,f}$ means a forecast of the price in time t. s is the scenario if we have many different forecasts, made using information up to time f. $\hat{q}_{t,s,f}$ is the equivalent for inflow. The objective is now to maximize what we will refer to as the matrix profit computed over the scenarios.

One possible interpretation of scenarios is that they are a discrete probability distribution over all possible prices or inflows. It could either be over all possible prices, or over the sequences of prices. In reality, this modeling choice does not accurately represent reality, as the actual price will not be exactly equal to any of the forecast scenarios. Instead, the actual values will likely come close to some scenarios. There are two main reasons Trønder Energi optimizes for the discrete case. Firstly, the forecasts arrive as discrete, possible worlds based on weather simulations and different assumptions. Secondly, the discrete probability case is required to solve the problem with stochastic programming, the current model used by Trønder Energi. In general, it makes it easier to create an objective function.

One issue with our modeling of the inflow scenarios as a discrete probability distribution is that the true inflow might be lower than the lowest inflow scenario. This can have large consequences for constraint violation, as each inflow scenario might barely avoid breaking constraints, but the true inflow is more extreme than either of the scenarios and leads to a constraint violation. One simple way to deal with this would be to add soft constraints, like being a certain distance from the maximum or minimum reservoir level, and get punished for coming closer. This requires reasoning about the forecast error and what risk we are willing to accept. Because we lack domain knowledge about the inflow forecast error and also do not have enough information to quantify the damage of emptying the reservoir, this is not further developed in this thesis. We only use soft constraints of a value that seems to work well on the simulation.

We denote the set of price scenarios and inflow scenarios with P_f and Q_f. Assume that each scenario has an equal probability of occurring. A complete realization of the stochastic elements in the problem is then given by an element in the Cartesian product of P_f and Q_f, $P_f \bigtimes Q_f$. If we assume that both Q_f and P_f are independent and uniformly distributed, then each of the pairs of price and inflow scenarios occurs with equal probability given by $(|Q_f||P_f|)^{-1}$.

A natural way to estimate the matrix profit is then to use Equation 2.1, 2.2 and 2.5, but replace the deterministic price and inflows with each pair of price and inflow scenarios from $P_f \bigtimes Q_f$, and calculate the average profit.

For ease of use, we here define the deterministic objective function by:

$$J(\vec{d}, \vec{p}, \vec{q}, x_0, e_T), \tag{2.7}$$

where the vectors $\vec{d}$, $\vec{p}$ and $\vec{q}$ are the discharge decisions, prices and inflows for all time steps $t \in (1, 2, ..., T)$, and x_0 and e_T are the scalar values for the initial reservoir filling and the rest price. This objective function outputs the profit, given by Equation 2.5.

In the uncertain case, $\vec{p}$ and $\vec{q}$ are unknown, so we instead optimize the objective:

$$\frac{1}{|\mathcal{C}_f|} \sum_{\tilde{s}=1}^{|\mathcal{C}_f|} J(\vec{d}, \vec{p_s}, \vec{q_s}, x_0, e_T), \tag{2.8}$$

where $\mathcal{C}_f = P_f \bigtimes Q_f$. The only change compared to the deterministic objective is that the true price and inflow are replaced with scenarios. We then compute this profit for each pair of possible price and inflow scenarios, and take the average of these profits. This is our proxy metric for how good a trajectory is and can be interpreted as the expected profit of the trajectory $\vec{d}$ if we assume all scenarios occur with uniform probability and are independent.

However, while we optimize Equation 2.8 in this thesis, this is just a lower bound of the optimal solution. Because we are given more information in the next 24 hours, in the form of more accurate forecasts, combined with the ability to reschedule discharge decisions after this time, it is possible to do better. This is a class of problems called recourse problems with multiple stages, while Equation 2.8 can be seen as a one-stage optimization problem. While we only use one-stage optimization for simplicity in this thesis, we elaborate some more on multi-stage optimization in the rest of this section.

We will now show a simple example of how multiple stages can change the optimal schedule compared to a one-stage optimization problem. Assume that we must make a plan for some planning horizon. Within this planning horizon, we know that we get more information and can reschedule our production based on this information in what we will call stage 2. In stage 1 we must decide what production to use now, which depends on what is optimal in stage 2. Let us consider a simple problem with a planning horizon of three steps where we only have enough reservoir water to produce at a single time step. There is a 50% chance that the price sequence $\vec{p_a} = (60, 0, 100)$ occurs, and a 50% chance that $\vec{p_b} = (60, 100, 0)$ happens. In the one-stage problem, we have to commit to one action now, without getting more information later. The optimal decision would then be to produce at the first step where the price is 60. Producing at step 2 or 3 gives an expected price of 50, which is lower than 60. However, this changes if we consider a two-stage optimization problem where we get to know which scenario occurred after step 1, and we can also update our plan according to which scenario occurred. We should then not produce at step 1 but save the water and produce at step 3 in scenario A and step 2 in scenario B, for an expected profit of 100. This shows how the multistage problem changes the optimization problem, and how a one-stage optimization procedure gives a sub-optimal production schedule. The negative side of multi-stage optimization is that solving it requires so much computation and memory that it is sometimes not possible to fit all scenario-dependent solutions in memory. It also requires modeling exactly what information is revealed in the stages. One way to solve the multistage problem is discussed in Section 2.2.1.

2.2 Background Theory

This Section gives background knowledge on relevant methods in the field of artificial intelligence. Section 2.2.1 explains the intuition behind traditional optimization methods used by industry. These are linear programming and stochastic programming. The next parts are related to reinforcement learning, and start off by giving an introduction to reinforcement learning in Section 2.2.2. We also mention the difference between model-based reinforcement learning and model-free reinforcement learning. In model-based reinforcement learning, there are many ways to plan ahead, by simulating many options. One specific way to plan on a model is Monte Carlo tree search, which we discuss in Section 2.2.3. Another way to plan is the cross-entropy method which we give some relevant background material on in Section 2.2.4. Supervised learning, a key component in many reinforcement learning algorithms is discussed in 2.2.5. Finally, we end by explaining Markov chains and how they can be used to generate unlimited amounts of data in Section 2.2.6.

2.2.1 Linear programming and stochastic programming

Traditionally the hydropower industry uses methods from the field of operations research to optimize scheduling. Linear programming can maximize a linear objective function of the form $z = c_1x_1 + c_2x_2 + ... + c_nx_n$ given a number of linear constraints. Linear programming only works on deterministic problems, since it requires all data to be known. Stochastic programming is a way to solve uncertain problems. Trønder Energi uses the software Sharm, based on stochastic programming (Sintef, 2022). These methods can get very close to the optimal solution but have a high running time when there are many price and inflow scenarios. When the power system changes to a 15-minute resolution, the running time will increase further.

To get an intuition for how stochastic programming works, we first need to explain linear programming, which we will base on Matouek and Gärtner (2006). The canonical form of linear programming is given by:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{m} w_j \ x_j \\ \text{subject to} \quad & A\vec{x} \leq \vec{b} \\ & \vec{x} \geq 0 \end{aligned}$$

The goal is to find a vector, $\vec{x}$, that maximizes a linear objective. The maximization is subject to linear constraints, given by the elements of the matrix A. Consider a matrix row $A_j, * = (1, 2)$ and vector element $\vec{b}_j = 25$. This gives the constraint $x + 2y \leq 25$. The linear constraints define a feasible region, where Figure 2.5 shows an example with four constraints. A feasible solution is a point within the dark area.

Consider an example where the objective is given by $z = 2.25x + 1.5y$. We can visualize multiple levels of the objective line in Figure 2.6. A single line shows all points that give the same value z. In this case, lines further to the right have higher objective values.

The fundamental theorem of linear programming states that at least one optimal solution occurs at a vertex point. A vertex point is an intersection between two or more linear constraints. In Figure 2.5 we have four vertex points. Consider a point in the interior of the feasible region. A better solution can always be obtained by moving along x_j in a that increases the objective function. On a point at the edge of a single linear constraint, one can move along the linear constraint. If the linear constraint is not parallel with the objective function, there will be one direction that increases the objective function. In some cases, there are an infinite amount of optimal solutions if the objective function is parallel with a constraint, but in that case, the point can move along this constraint without changing the objective value until another constraint is reached. Figure 2.7 shows the objective lines and the constraints in the same figure. The pink line with $z = 40$ seems close to the optimal value.

One possible algorithm for finding the optimal solution is then to enumerate all possible vertex points and compute the objective value for each of them, and then keep the optimal solution. However, the number of vertex points can be very large if there are many constraints and dimensions. The main idea of the simplex algorithm is to traverse these intersections. It starts at a vertex point and searches in the directions where the objective function has the steepest ascent. There are also many edge cases handled in specific ways and other optimization tricks that are not described here. In the worst case, the simplex algorithm must visit all vertex points, which is exponential in the number of constraints, given by $O(2^n)$. Intuitively, each new constraint can intersect all other constraints and double the number of vertex points. However, the average case has a polynomial complexity in the number of constraints for constraint matrices sampled from most probability distributions.

Stochastic programming introduces uncertainty into the linear programming problem specification (Higle, 2005). Recourse models are a type of stochastic programming where some decisions, x, has to be made at the start. Then at a later stage, some information is revealed and parts of the decision, y can be adapted to the specific scenario. Decision y specific to scenario $\tilde{w}$ is given by $y_{\tilde{w}}$. In our hydropower scheduling problem, the problem is to decide the 24 first hours, while keeping in mind that our forecasts will improve before we make the next 24-hour decision.

A two-stage recourse problem is given by Equation 2.9. The formulation is the same as for linear programming, except that the term $E(h(x, \tilde{w}))$ is added to the objective. This is a sub-problem to be solved, where new information according to scenario $\tilde{w}$ is given. Equation 2.10 shows the sub-problem. The parameters of the objective function, $g_{\tilde{w}}$ and the linear constraints defined by $W_{\tilde{w}}$ and $r_{\tilde{w}}$ are all specific to the

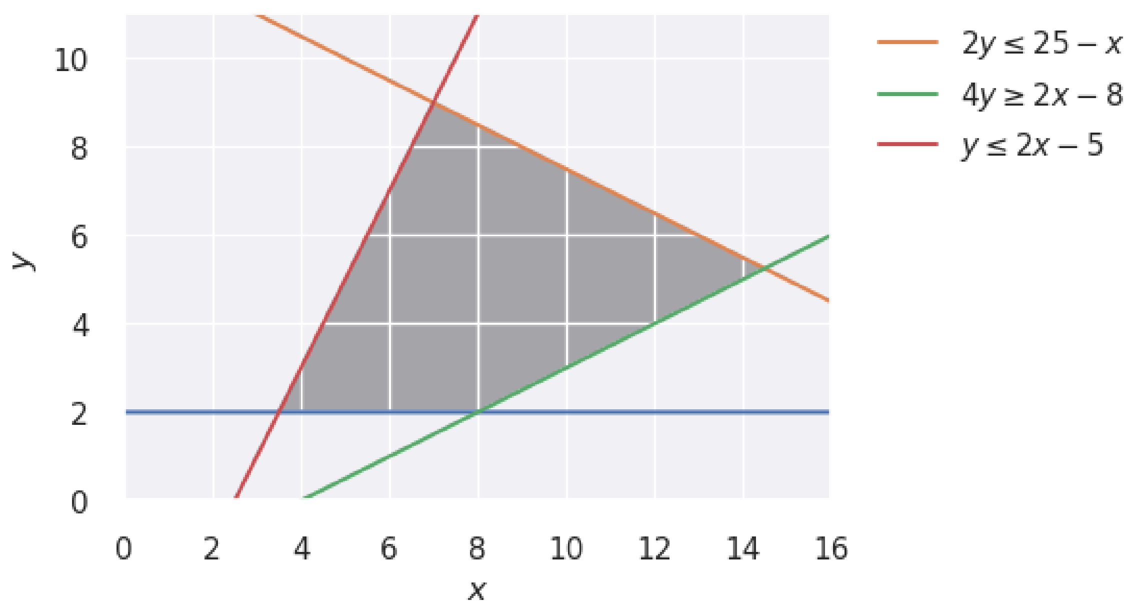

Figure 2.5: The feasible area of a linear programming problem with two variables

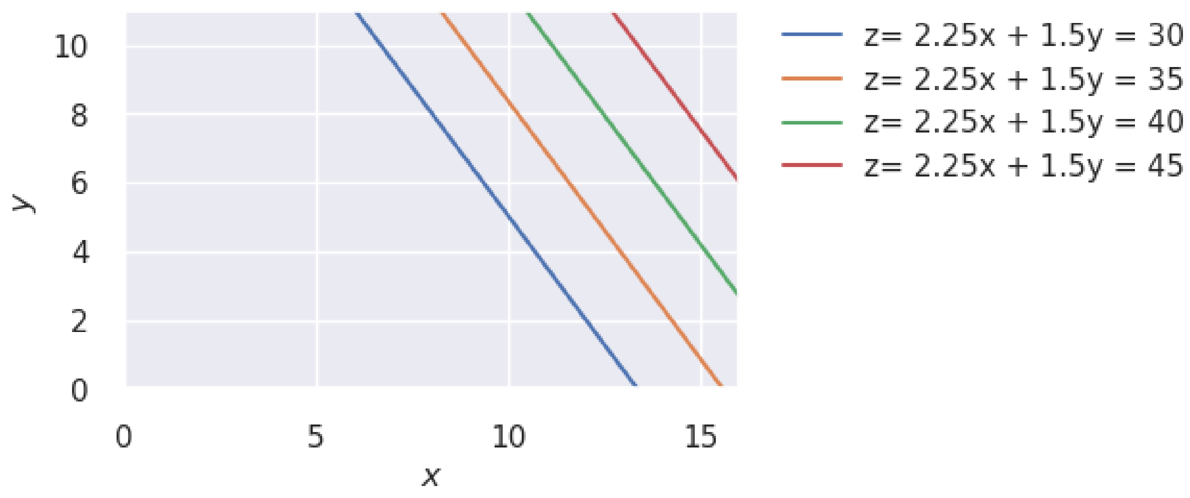

Figure 2.6: The objective lines, $z = w_1x + w_2y$ for different values of z. A single line show all values of x and y that give a specific value z.

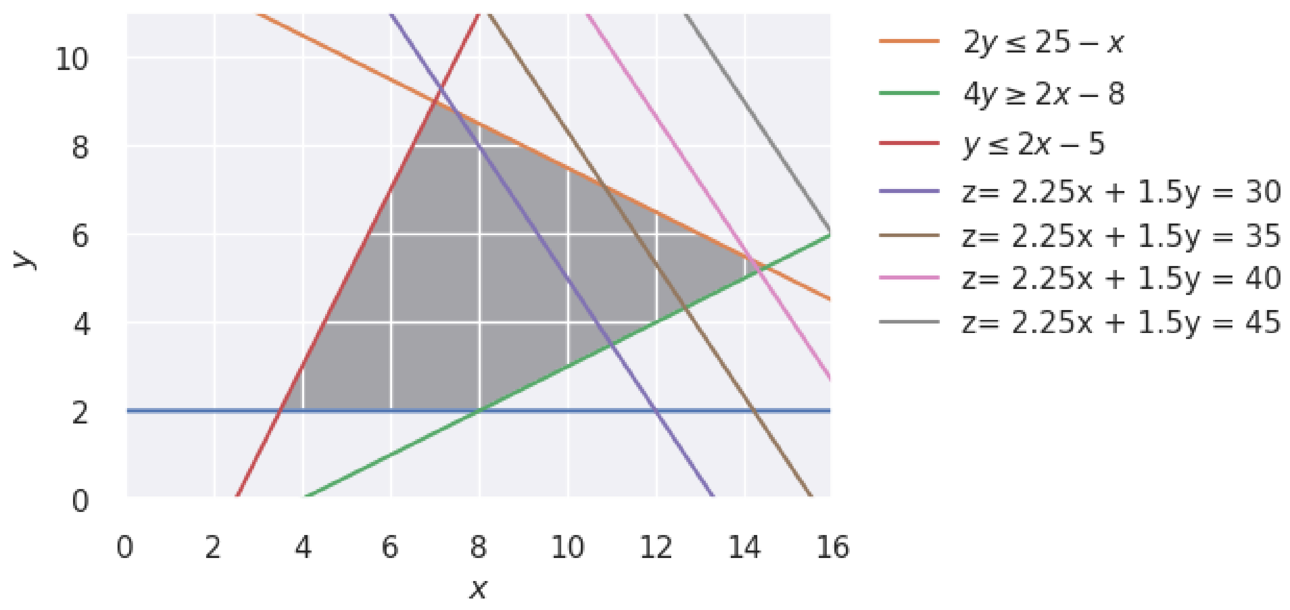

Figure 2.7: The solution to a linear program. The pink line is approximately the highest line that intersects with the feasible area.

scenario $\tilde{w}$. The linear constraints also depend on the first stage decision x, given by $T_{\tilde{w}}x$.

Note that the optimal solution to the sub-problem depends on the first-stage decision x, and the choice of x depends on the solution to the sub-problem. The problem is solved by formulating it as a linear programming or mixed-integer linear programming problem, though the objective function and the number of constraints are much larger than the deterministic version. This makes the running time much higher.

$$\begin{aligned} \text{maximize} \quad & cx + E(h(x,\tilde{w})) \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{2.9}$$

Where $h(x, \tilde{w})$ is given by

$$\begin{aligned} \text{maximize} \quad & g_{\tilde{w}}y \\ \text{subject to} \quad & W_{\tilde{w}}y \leq r_{\tilde{w}} - T_{\tilde{w}}x \\ & y \geq 0 \end{aligned} \tag{2.10}$$

2.2.2 Reinforcement learning

Reinforcement learning is an area of artificial intelligence where agents learn by interacting with the environment. Figure 2.8 shows the interaction. For each time step, the agent is given a state (S_t) and a reward (R_t). Given this information, the agent must decide which action (a_t) to take. The action affects the environment, and the agent should learn to select actions that maximize the future rewards.

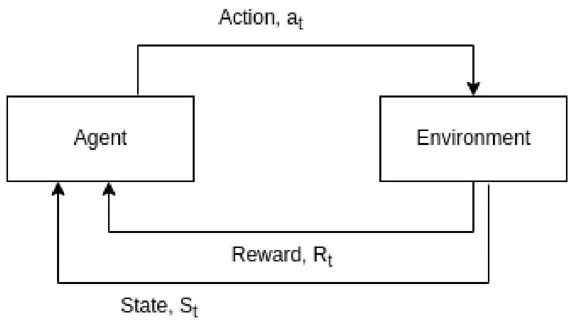

Figure 2.8: Agent environment interaction

Reinforcement learning is typically formalized as a Markov decision process. The sets of all possible states, rewards and actions are given by the 3-tuple $(\mathbb{S}, \mathbb{R}, \mathbb{A})$. The state and the reward are random variables that only depend on the preceding state and action. The probability of transitioning to state s' and receiving reward r when in state s and selecting action a, is given by Equation 2.11. This probability function describes the dynamics of the environment. A state, therefore, gives all information that is relevant to the future of the environment. This is is called the Markov property: $S_{t+1} \perp\!\!\!\perp \{a_0, s_0, a_1, s_1, ..., a_{t-1}, s_{t-1}\}|S_t$. State S_{t+1} has conditional independence from earlier states and actions, conditioned on the previous state, S_t.

$$p(s', r|s, a) = Pr(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a) \tag{2.11}$$

The goal is to maximize expected future rewards. G_t is the discounted future reward, shown in Equation 2.12, where γ $(0 \leq \gamma \leq 1)$ is the discount rate and T is the episode length. In the case of an environment with an infinite possible number of time steps, $T = \inf$, the discount rate makes the sum of future rewards finite. In the case of a finite episode length, the discount rate can be equal to 1 or excluded.

$$G_t = \sum_{k=0}^{T} (\gamma^k R_{t+k+1}) \tag{2.12}$$

Policy $\pi : s \to a$

A policy is a mapping from state to action, or possibly to a probability distribution over possible actions. In the latter case, $\pi(a|s)$ can be read as the probability of selecting action a when in state s. Reinforcement learning normally iteratively improves the policy so that it selects actions that maximize G_t in Equation 2.12. The policy can be represented as a table, a neural network, or another other function approximator.

Value function $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$

A value function is a mapping from a state to the expected value of a state following a policy *pi*. The expected value is the total accumulated reward that can be expected from being in a state. The future rewards depend on the decisions of the agent, so a value function can be on-policy, meaning the value represents future rewards given the current policy, or off-policy when the value is based on some other policy. Q-values are often used when referring to the value of a state-action pair and have the same definition as the value function but conditions on a state and an action.

Optimal policies and value functions π^*, $v*$

A policy π is defined to be better than or equal to another policy π' if $v_\pi(s) \geq v_{\pi'}(s), \forall s \in \mathbb{S}$. That is the expected value for the policy is greater or equal for all states s. This gives an ordering of policies, and the optimal policies will have no other policies that are greater and are denoted π^*. The optimal value function is v^* and is the value function corresponding to the optimal policy. For complex problems, the optimal policy is generally not possible to find, and the goal is to find a good approximation of the optimal policy.

An important property is the Bellman equation (Sutton and Barto, 2018, p. 59):

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + G_{t+1} | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a)[r + \gamma v_\pi(s')] \end{aligned}$$

The equation states that the value of state s equals the value of the next expected state added to the next reward (s', r). Each possible (s', r) are weighted by their probability of occurring. The expression is also weighted by the probability of each action occurring, as given by the policy. Many reinforcement learning algorithms use some modification of the Bellman equation to update the value function to improve the estimates of the value of a state.

The value iteration algorithm changes the Bellman equation into an update rule and is guaranteed to approach the true value function.

$$v_{k+1}(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_k(s')], \tag{2.13}$$

for all states s ($\forall s \in \mathbb{S}$). v_k is the kth iteration of the value function. A difference to the Bellman equation is that the action that maximizes the expected future rewards is selected. The updates can be run iteratively and will formally not guarantee true values until after an infinite number of iterations. The time complexity is $O(|\mathbb{S}|^2 |\mathbb{A}|)$ for a single iteration. When the set of states and actions is large, value-iteration might not be feasible. In addition, the transition model must be known.

TD-learning is an example of an algorithm that does not require the transition model to be known. It uses the update rule $v(s) \leftarrow (1 - \alpha)v(s) + \alpha(r + \gamma v(s'))$. Since the transition model is not known, TD-learning updates the value estimation based on actually observed rewards and states. Since there could be many possible unobserved transitions from s, not just s' and r, the estimation is slowly updated using a learning rate, α. Value iteration and TD-learning are examples of model-based and model-free algorithms, respectively.

In general, a model-based agent needs access to a model of the environment to map a state and an action to the next state and reward, given by $\tilde{f}(s_{t+1}, R_{t+1}|s_t, a_t)$. This transition model can either be given or learned using function approximation such as a deep neural network. The transition model allows agents to plan ahead, and simulate the outcome of different actions. There exist several planning methods other than value iteration. Some common examples used in this thesis include Monte Carlo tree search, discussed in Section 2.2.3, and the Cross-entropy method discussed in Section 2.2.4. It is also possible to use conventional model-free algorithms to do many simulations with the transition model, and then select the best trajectory.

The Actor-critic is a popular model-free algorithm that builds on TD-learning. It consists of an actor, that learns a probability distribution over actions, and a critic, which approximates the value function. To update the actor and critic, the algorithm computes the TD-error, which measures how wrong the estimate of $v(S_t)$ was. For the true value function, the value of the current state should be equal to the next reward and the discounted value of the next state. The difference between these two is the TD-error which is equal to $\delta = R_{t+1} + \gamma v(S_{t+1}) - v(S_t)$. This TD-error is then used to update both the actor and the critic. The critic simply updates its value function to $v(s) = v(s) + \alpha\delta e(s)$. Intuitively, if the TD-error is negative, the action led to a state with an unexpectedly low value, and that action is made less likely to be selected in the future.

There are a number of properties that can make reinforcement learning problems hard to solve. If the set of possible actions or the set of possible states is large, then exploring all state-action pairs and making good value estimations will take a long time. Getting the best policy from limited experiences is referred to as sample efficiency, and is a key objective in state-of-the-art reinforcement learning. Function approximation using deep learning can allow an agent to generalize its experiences to unseen states and actions. Finding good representations of states and actions is also important to get good generalizations.

Another problem is that of temporal credit assignment. If an agent receives a high reward, there is a chain of earlier states and actions leading up to the high reward that could be responsible. Perhaps some actions have contributed positively, while others might have had a negative influence. Deciding which state-actions should be credited is called the credit assignment problem. One common way to handle this is to give each state-action a discounted reward, which means that state-actions in the more distant past get less credit. Giving all earlier state-actions some credit will make some bad actions seem good, and some good actions seem bad. This can be handled by sometimes picking actions that seem worse to explore different actions. Problems where good actions immediately get good rewards are easier to solve. If possible, systems should be designed to immediately give rewards that reflect how good the action was.

2.2.3 Monte Carlo Tree Search

Monte Carlo Tree Search is a heuristic search algorithm that can be used to solve a wide range of sequential decision problems, such as games, planning and optimization. At a high level, the algorithm builds a tree where each node is a state and each edge is a possible action, leading to a successor node and state. It then searches the tree for good actions by prioritizing unexplored actions and actions associated with high rewards.

The algorithm starts by creating an initial node corresponding to the current state. Then the following four steps are repeatedly carried out. Firstly, a tree policy selects the best actions repeatedly until a leaf node is reached, that is the end of the tree. There are many ways to select actions, but most prioritize selecting actions that on average received a high reward and actions that have been less explored. A common tree policy is Upper confidence bounds for Trees (UCT). This policy selects the action that maximizes the following equation:

$$\text{UCT} = Q(s,a) + c\sqrt{\frac{N_i}{\log n_i}},$$

where $Q(s,a)$ is the estimated value of taking action a in state s, N_i is the number of times the parent node corresponding to state s has been visited and n_i is the number of times the child node has been selected. The last term is the exploration bonus which increases when the parent node has been visited many times relative to the child node. The parameter c determines the prioritization of exploration relative to exploitation.

After using the tree policy to reach the last node of the tree, the leaf node, it is possible to expand the leaf node by adding edges leading to new leaf nodes. However, at a certain distance from the root node, the algorithm will often stop creating child nodes to save memory and computation and instead go to the third phase, the simulation phase. No nodes or statistics about actions are kept, and a different policy, the rollout policy, is used to reach the end of the episode so the final reward is known. Finally, the cumulative reward is propagated through all the visited nodes to improve node selection for future iterations. The four steps are summarized below.

1. Selection: Start with the root of the tree and select actions with the tree policy until a leaf node is reached.
2. Expansion: Expand a leaf node.
3. Simulation: Use the rollout policy to simulate the rest of the episode.
4. Backpropagation: Return the score to the nodes from the selection phase.

Browne et al. (2012) gives an overview of several extensions and variations of Monte Carlo tree search. One such extension is the idea of seeding, which initializes nodes with useful statistics, perhaps from heuristics. Another extension to Monte Carlo tree search is history heuristics. As the program runs, some results are saved and used to inform either the heuristics or other things. If many trajectories broke the lower reservoir constraint, this could be a signal to be more conservative with power.

One of the main advantages of Monte Carlo tree search is that it is highly parallelizable. This means that the algorithm can take advantage of multiple processors or even distributed computing resources to explore different parts of the search space simultaneously. By dividing the search space into smaller subproblems and assigning them to different processors, the algorithm can significantly reduce its running time, making it practical for solving large-scale problems. There are many ways to parallelize Monte Carlo tree search. It is possible to run almost the same program in multiple processes and occasionally exchange information. Another way is to use the tree policy in a single process, and when a leaf node is reached, run many rollouts in parallel, which can yield a more accurate estimation of the true value of selecting a certain action.

2.2.4 Cross-entropy method

The cross-entropy method is an iterative optimization technique that has been successfully applied in model-based reinforcement learning. The main idea is that we sample potential solutions from an initial probability distribution, and then change the sampling distribution towards more promising areas based on the sample solutions. This section is based on Boer et al. (2005).

Consider the task of optimizing a function $f(\vec{a})$ with respect to the vector $\vec{a}$. In the context of reinforcement learning, $\vec{a}$ will be a vector that represents a sequence of actions $(a_0, a_1, ..., a_t)$ from time step 0 to t. The cross-entropy method starts by defining an initial probability distribution $H(x; v_0)$, where v_0 is a vector of parameters for the probability distribution. We can then sample n trajectories $X_i \sim H(x; v_0)$, and evaluate them on our objective function, $Y_i = f(X_i)$. We then rank the results by the objective function score and select the m best trajectories as an elite set. The elite set is used to update the sampling distribution. There are many ways to update the probability distribution, but intuitively, the update should increase the probability of sampling trajectories that are similar to the elite set, thus focusing the optimization on more promising areas. One way to achieve this is to take the maximum likelihood of observing the elite set. We set the new probability distribution parameter as follows:

$$v_{\text{new}} = \underset{v}{\operatorname{argmax}}\, L(v|X_1, X_2, .., X_n), \tag{2.14}$$

The new probability distribution $H(x; v_{\text{new}})$ will have the parameter v_{new} that maximized the likelihood of observing the elite set. Consequently, when we sample new trajectories in the next iteration, the samples will be closer to the high-performing elite set of the previous iteration.

In practice, updating the sampling distribution as described in Equation 2.14 can make the procedure converge on a local maximum too quickly, without thoroughly exploring the search space. Especially if the dispersion of the probability distribution is reduced too quickly, then large parts of the solution space will never be explored. To tackle this, a common approach is to update the sampling distribution more gradually, for example using the update $v_{\text{new}} = \alpha v_{\text{MLE}} + (1-\alpha)v_{\text{old}}$. Here α is a parameter that tunes how much we weigh the parameters suggested by maximum likelihood estimation for the elite set relative to the parameters of the old sampling distribution. A low value of α will prioritize exploration over exploitation. The sampling and updating of the sampling distribution are carried out for many iterations until a stopping criterion is met, for example, when a maximum number of iterations is reached. The trajectory with the highest score on the objective function among the sampled trajectories is selected as the output.

While this update is applicable to various probability distributions, it works especially well when the maximum likelihood estimate has an analytical solution, such as a multivariate Gaussian distribution. The update of the distribution will then be faster and simpler than more complex distributions where a numeric solution to the maximum likelihood estimate is needed. Another limitation of this method is that it is sensitive to the choice of the initial distribution. It can be beneficial to start with the center of the distribution close to the global optimum. Like most methods, the cross-entropy method will also struggle more when $\vec{a}$ is large in size, as the solution space increases exponentially. In these cases, increasing the number of iterations or samples per iteration might be needed to find an adequate solution, though this increases the running time.

2.2.5 Supervised learning

Supervised learning is a class of algorithms that learns to map inputs to outputs given example data. This mapping can then be used to predict outputs on new, unseen inputs. The training data are tuples of inputs and outputs $(x_1, y_1), ..., (x_n, y_n)$, where x_i is the input and can be a vector of relevant values. y_i is the output and can be a continuous value for regression tasks or a discrete class label for classification. x_i and y_i are related by an unknown function f, and the goal is to find a hypothesis h_θ that approximates f. h_θ is a function with a vector of parameters θ. To find the optimal parameters, we define a loss function $L(h_\theta(X), Y)$. The loss function defines a measure of the distance between the predicted output and the true label. Our goal is then to find the parameters θ that minimize the average loss. This function approximation can then predict the output of x in cases where y is unknown.

We will now give a brief high-level description of some supervised learning methods used in this thesis, starting with neural networks.

Neural networks

Artificial neural networks are a type of supervised learning that can approximate a wide range of functions. A neural network consists of layers with artificial neurons, each neuron receiving some input and computing an output that can be fed into the next layer of neurons. Using many layers allows the network to represent very complex functions. The most basic type of layer is the fully connected layer. A single neuron in a fully connected layer computes the weighted sum of all the inputs and then applies a non-linear function. This is shown in Equation 2.15, where $\theta^{(l)}$ are the parameters for layer l, $x^{(l)}$ is either the input or output of layer l and b is the bias. Neural networks can be built from many layers, each consisting of many neurons. The neurons in later layers compute the weighted sum of the neurons in the previous layer. It can be proven that neural networks with at least one hidden layer are universal function approximators(Cybenko, 1989), with the caveats that the function is continuous and that the result is an approximation that can be made more accurate by increasing the number of hidden neurons. This means that there exist weights that can learn almost any mapping.

$$a^{(l+1)} = g(\theta^{(l)}x^{(l)} + b^{(l)}) \tag{2.15}$$

Finding good weights requires learning, which is done by minimizing a loss function. By forward propagating a sample input through the network, a prediction $\hat{y}$ is acquired. How close the prediction was to the true value y is measured by a loss function. The goal is then to minimize the loss, which is typically done by a variation of the gradient descent algorithm. It is possible to calculate the derivative of the loss with regard to each weight in the neural network. The vectors of partial derivatives are called gradients, and point in the direction of the steepest ascent. Intuitively, the local minimum of the loss function can be found by going in the opposite direction of the gradient. This gives the following update rule:

$$\theta_j = \theta_j - \alpha \frac{\partial L}{\partial \theta_j}$$

Where θ_j is weight j, α is the learning rate and L is the loss. This update is done for all weights j and will approach a local minimum for the loss function.

The key challenge is to learn a neural network that generalizes to unseen examples. When a network learns to perform well only on data used for training, the network has been overfitted. A number of techniques exist to improve the generalization of neural networks, such as penalizing large weights to rely less on some features or dropping some random features to make the network more robust. There are also other layers than the fully connected layer. If the data has a spatial structure, then convolutional layers can perform well. Transformers or LSTM layers are good at capturing temporal dependencies in data.

Gaussian Naive Bayes

The Gaussian Naive Bayes can learn to map continuous features to classes given two assumptions. Firstly, we assume that each feature follows a Gaussian distribution, and secondly that each feature is independent.

The likelihood of observing the feature x_j given the class C is given by:

$$P(x_j|C) = \frac{1}{\sqrt{2\pi\sigma_C^2}} \exp(-\frac{(x_j - \mu_C)^2}{\sigma_C^2}), \tag{2.16}$$

where $\mu_C)^2$ and σ_C^2 are the sample mean and sample variance for feature j within class C. That is, for all features in samples that belong to class C, we compute the sample mean and variance. With these parameters, we have a Gaussian distribution that shows the probability of observing some feature x_j given that it belongs to the class C.

To use this to classify samples with unknown classes, we use Bayes rule to invert the probability. The probability of the sample belong to class C given the features $(x_0, X_1, ..., x_n)$ is given by $P(C|x_0, x_1, ..., x_n$. Because all the features are assumed to be independent, we can rewrite this to $P(C)P(x_0|C)P(x_1|C)...P(x_n|C)$. Note that the normalization factor has been dropped because it is the same for each class. We compute this metric for each class C and classify the sample to the class with the highest probability. This method is relatively fast and simple, with the trade-off that the assumptions are typically not correct. However, the method can give reasonable results even when the assumptions are not true.

Linear Discriminant Analysis

Linear Discriminant Analysis is a classification method that finds the linear combination of features that best separates the different features. Linear discriminant analysis can only learn linear decision boundaries and make some often unrealistic assumptions about the data. The advantages are that the model is fast, and because the hypothesis is simple, it often avoids overfitting. It can also get decent performance even if the assumptions do not hold.

Linear Discriminant Analysis works by reducing the dimensions of the features in a way that maximizes the separability between classes. To do this, we first compute the mean feature vector for each class. We also compute something called the within-class scatter matrix, measuring the spread of data within the class, and the between-class scatter matrix which measures the separation between classes. The latter is calculated by taking the deviation between the in-class mean and the mean for all classes, and multiplying this by its transpose. We then find a projection that maximizes the ratio between the between-class scatter and the within-class scatter. In the projected space, the features in the same class will be close together, and the dispersion between classes will be maximized, which allows for easier separation. To infer classes for unseen data, we project the features with our learned projection matrix and use a simple method like a threshold or a nearest neighbor to classify the points.

Gradient boosting

Gradient boosting is an ensemble method that combines the predictions of many simple models. Decision trees are often used as the simple model. The decision tree algorithm builds a tree by selecting the features that best splits the data, aiming to maximize the separation between different classes. Child nodes are created for each of the splits, and the data samples corresponding to that split are used to consider the next split. This is done recursively until some stopping criterion is met.

Gradient boosting starts by training a decision tree or other model to predict the label. Subsequent models then try to predict the error of the previous model. This is repeated for a certain depth and can lead to better predictions. To infer with a trained ensemble, all individual models make predictions that are added together to produce the final prediction. There are numerous implementation details to deal with continuous values, different losses, optimization and other topics we will not go into here. A popular implementation used later in this thesis is LightGBM, described by Ke et al. (2017).

2.2.6 Markov chains

Markov chains are a type of mathematical model that can be used to generate sequences of events. The model is made up of a set of states and a transition model. The transition model assigns probabilities to the possible transitions between states. Formally, this can be expressed as $P(X_t = x_i | X_{t-1} = x_j)$ for all x_i, x_j in the set of states. There are two important assumptions in Markov chains. The first is called the Markov assumption which states that the next state depends only on the current state. This simplifies calculation since conditioning on the previous state is enough to get the new probability distribution ($P(X_t|X_{t-1})$. The second assumption is that the transition model is stationary, meaning that the transition model does not change for any times t.

Markov chains can be applied to both discrete and continuous state spaces. In the case of continuous state spaces, the states can be converted to discrete ones by grouping the data into bins.

2.3 Motivation

Optimizing hydropower scheduling can lead to large increases in profits for hydropower operators. The main mechanism is that power prices are highly volatile, so producing when prices are high can give increases in revenue. Additionally, the operators can optimize the total amount of power produced by using head variation and reducing discharge friction loss, which for a given unit price will increase profits. In addition, there are also operating constraints that must be followed.

In addition, hydropower optimization carries broader benefits to society as a whole. Hydropower serves a key role in ensuring a stable supply of power by producing power during periods of low load. Both the usage and production of power fluctuate, and this can cause periods where there is not enough power production and other periods where there is too much power production. Hydropower scheduling is a way of storing power for use when it is most beneficial. This will be especially important with the growing usage of intermittent renewable energy sources, such as solar and wind. By ensuring a stable supply of power, effective hydropower scheduling can reduce the reliance on fossil fuels and facilitate the transition to a more sustainable economy.

We have discussed stochastic programming, which is a traditional method to solve the hydropower optimization problem. While this method can give nearly optimal solutions, they need large amounts of computation and time to converge. This will become more problematic when prices and discharge decisions will change from an hourly resolution to a 15-minute resolution (Statnett, 2022). The number of steps in the planning horizon will quadruple, which can create problems with computation. Employees at Trønder Energi think that the current software might not scale to this added complexity. Our overall goal in this thesis is therefore to find an alternative method that gives a good trade-off between running time and the quality of the solution.

Furthermore, the hydropower scheduling problem is theoretically interesting since the problem shares many characteristics with other important problems. The most similar problems involve some form of energy storage, like battery arbitrage, demand response, and other problems described later in Section 3.1. Methods that work for hydropower optimization might also be beneficial for these problems. In addition, addressing the challenge of large state spaces is crucial for applying artificial intelligence methods to more complex problems. The hydropower problem also deals with uncertainty arising from many different sources, like markets, weather and human behavior. Finding artificial intelligence methods that solve this problem can have broader implications for the fields that study decision-making under uncertainty.

In recent years, many advancements have been made in the field of artificial intelligence. Reinforcement learning, for example, addresses Markov decision problems, which is a very general problem formulation that also includes hydropower scheduling. By surveying ideas and tricks from the literature on reinforcement learning, we can get inspiration to tackle our challenges. This is the motivation behind Research Question 1. Applying reinforcement learning ideas to the hydropower scheduling problem can also aid in identifying the strengths and weaknesses of the methods, which can lead to further improvements in the field.

Chapter 3

Related literature

In this chapter, we will discuss state-of-the-art techniques. Because the literature on hydropower optimization is limited, we need to look at problems with similar characteristics. We therefore discuss the structure of the hydropower problem in Section 3.1, and introduce problems with similar properties. In Section 3.2 we look at different methods that have been used to solve similar problems.

3.1 Overview of relevant literature

There are a few papers using methods from artificial intelligence to solve the hydropower scheduling problem. One limitation of these papers is that they often use different problem specifications or assumptions, making it difficult to evaluate the effectiveness of the algorithms for this problem. Additionally, the authors do not always provide a thorough evaluation of the algorithms, such as a comparison with traditional methods or baselines. This makes it challenging to assess the feasibility and potential benefits of using the proposed techniques.

It is therefore useful to also examine problems that have similar characteristics. The hydropower scheduling problem is an instance of optimal control under uncertainty. This is a well-studied problem in fields like artificial intelligence, operations research and control engineering. Examining such research can help to gain valuable insights and potentially adapt or extend existing approaches to better solve the hydropower scheduling problem.

There are six important characteristics of the hydropower scheduling problem. Firstly, the problem involves a resource that can be managed. This resource could be energy that can be used now or saved for later, like in hydropower. Secondly, the evaluation of the decision relies on exogenous, uncertain information such as prices and inflow, that are not known when taking the decision. Uncertainty can have different distributions and modeling choices. Thirdly, we have access to many estimates of the future trajectory of these uncertain exogenous variables. These estimates allow for better decisions when included in the state. However, with 30 price scenarios and 51 inflow scenarios over 12 days, the forecast uses 23328 real numbers. This makes the state space very large and hard to explore fully. This property is less studied than the previous two.

Our fourth characteristic is that the cumulative reward, the profit, is differentiable with regard to the actions. Work using this property is discussed in Section 3.2.4. Another characteristic is that the action space is continuous. The last characteristic is that the dynamics of the system have non-linearities. Some dynamics are easily modeled, such as the water level being the difference between inflow and outflow.

Environment	Manageable resource	Uncertainty	Forecasts	Differentiable objective	Continuous	Non-linear dynamics
Hydropower scheduling	✓	✓	✓*	✓	✓	✓
Unit commitment problem	✗	✓	✓	✗	✗	✓
Battery arbitrage	✓	✓*	✓*	✗	✓	✓
HVAC	✓	✓	✗	✗	✓	✓*
Inventory management	✓	✓	✓	✓	✓	✓
RL benchmarks	✗	✓*	✗	✗	✓	✓

Table 3.1: Overview of which characteristics apply to each environment. In environments where a characteristic sometimes applies, but not always, ✓* is used.

Method	Research paper	Environment
Deep learning 3.2.1	Matheussen et al., 2019	Hydropower scheduling
	Niu et al., 2019	Hydropower scheduling
	Dariane and Moradi, 2016	Hydropower scheduling
Model-free RL, 3.2.2	Riemer-Sørensen and Rosenlund, 2020	Hydropower scheduling
	Xu et al., 2020	Hydropower scheduling
	Haarnoja et al., 2018	RL benchmarks
	Kuznetsov et al., 2020	RL benchmarks
	Luo et al., 2022	HVAC
	Li et al., 2023	NA
CEM 3.2.3	Chua et al., 2018	RL benchmarks
	Pinneri et al., 2020	RL benchmarks
Differentiable simulator 3.2.4	Madeka et al., 2022	Inventory management
Heuristic search 3.2.5	Powell, 2014	Various energy-related
	Mars and O'Sullivan, 2022	Unit commitment problem

Table 3.2: Overview of research papers reviewed in this chapter and their corresponding environments and methods used.

Some problems that share the three first characteristics include battery arbitrage, demand response, the unit commitment problem, HVAC and the use of reinforcement learning for financial applications involving prices. Battery arbitrage involves buying power for storage when the price is low and selling power when the price is high. Similarly, demand-response is a class of problems where power consumers try to use power whenever the price is low and reduce power usage whenever the price is high. The unit commitment problem involves discrete actions, where many units of gas generators can be either turned off or on. Turning on a unit has some startup costs and there might be constraints.

HVAC (Heating, ventilation and air conditioning) involves trying to keep temperature, air quality, or humidity within a comfortable range while minimizing electricity usage or costs. These variables can be controlled by changing the intensity of heating and cooling units, though the exact effects and nature of these units vary from building to building. The problem contains non-linearities.

However, all of these problem domains have many modeling choices where the literature diverges. This is similar to the literature on hydropower scheduling. For instance, some do not include price forecasts in the state, and others do not include prices at all and instead aim for targets such as reducing power usage in HVAC, or matching supply and demand in the unit-commitment problem. This means that parts of the literature on these problems are not relevant. But a subset of the literature on these problems shares all characteristics mentioned earlier, except the unit commitment problem where the action space is discrete.

The inventory management problem tries to keep a sufficient supply to match the demand for different products. Keeping a larger stock of products increases storage costs while having less inventory than demand will result in a part of that demand being lost. Both demand and vendor lead times, the time from ordering an additional product until it arrives in the inventory is stochastic.

Most literature on reinforcement learning methods tests their algorithms on common benchmark problems. For continuous control, these include the half-cheetah problem, continuous cart pole and many others. These environments are similar in that they have continuous states and actions and have complex, non-linear dynamics. They differ in the other characteristics mentioned earlier, but the work is still worth reviewing because some of the methods do well across a large variety of tasks.

For a complete overview of which characteristics apply to each environment, see Table 3.1. The characteristics that only sometimes are included in a problem are indicated by ✓*. Table 3.2 gives an overview of each paper reviewed in Section 3.2 and the corresponding environment and which method was used to solve the problem.

3.2 Methods

This section looks at what methods have been used to solve the relevant environments in Table 3.1. Attempts have been made to use supervised learning to directly map a state to the next action, which we review in Section 3.2.1. Section 3.2.2 introduces two state-of-the-art reinforcement learning methods and looks at some applications to relevant problems. Usage of the cross-entropy method for model-based reinforcement learning is discussed in Section 3.2.3. We then turn to a paper that uses the property of a differentiable simulator in Section 3.2.4. Finally, some applications of heuristic tree search are reviewed in Section 3.2.5.

3.2.1 Supervised learning

It is possible to map a state directly to a discharge decision, using supervised learning. Matheussen et al. (2019) tries to learn the mapping from an ensemble of possible price and inflow scenarios to a discharge decision.They fit a neural net on training data, mapping inflow and price to an optimal discharge. A second neural net maps a reservoir level to the expected profit given the decisions found by gradient ascent. When making a decision in the case of an ensemble of scenarios, the neural net predicts the best decision for each scenario. The second neural network calculates the expected profit for all suggested decisions across all scenarios. The decision that maximizes this profit is selected.

To get the necessary training data, they use a Markov chain learned from historical data to generate an arbitrary amount of price and inflow data. They then find an optimal or at least good discharge decision for each of the price and inflow scenarios using gradient ascent.

Niu et al. (2019) tries to find the schedule that optimizes power generated given inflow and initial water level. The authors compare traditional methods with a neural network, support vector machine and other machine learning methods. They find that dynamic programming performs better than machine learning methods. Dariane and Moradi (2016) compares reinforcement learning to an artificial neural network in maximizing power production. The neural network is evaluated on a simulation. Since gradients are not available from a simulation, a variant of particle swarm optimization is used to optimize the weights. This is compared to Q-learning, and the authors conclude that the neural network outperforms reinforcement learning.

Some papers use machine learning for hydropower tasks not directly related to scheduling. Sauhats et al. (2016) use deep learning to forecast the future price and inflow, and then use traditional methods such as dynamic programming to schedule.

3.2.2 Model-free reinforcement learning

There are several model-free reinforcement learning algorithms that have been shown to perform well on a wide variety of problems.

The Soft Actor-Critic architecture is a popular model-free reinforcement learning algorithm that has achieved good performance on a diverse set of problems (Haarnoja et al., 2018). It builds on the Actor-Critic model, which we discussed in Section 2.2.2. The Soft Actor-Critic supports continuous action by making the policy a continuous probability distribution over the actions. It balances exploration and exploitation by adding an entropy term to the objective function. The normal objective is to maximize the sum of future rewards, the augmented objective is expressed in Equation 3.1. The function H is the Shannon entropy of the probability distribution of the policy π over the actions a, given by Equation 3.2. In general, the maximum entropy would be a uniform probability distribution, making each action equally likely. The effect of the entropy regularization term is then to increase the probabilities of actions that would otherwise be low, and decrease the probabilities of actions that would otherwise be much higher. This makes the policy more random and exploratory. The trade-off between exploration and exploitation can then be tuned by the parameter α, where higher α gives a more random policy. According to the authors, this entropy term often leads to faster convergence than the conventional reinforcement learning goal.

$$G_t = \sum_{t=0}^{T} (R_t + \alpha H(\pi(s_t)) \tag{3.1}$$

$$H(\pi(s_t)) = -\int (\pi(a|s) log(\pi(a|s))da \tag{3.2}$$

Riemer-Sørensen and Rosenlund (2020) uses the Soft Actor-Critic on the hydropower problem, though the environment specifications are quite different from the problem in this thesis. The authors optimize profit where a time step is for a week, and there is no forecast. They define a state to consist of week number, storage, the price for the week, inflow for the week and number of weeks before the storage is empty. The objective is to map this state to the discharge that maximizes profit. Since there is no information about future prices, the model essentially uses experience from past years to decide if the current price is good. For instance, the model might remember that weeks around winter typically have higher electricity prices, so it makes more sense to save water in the autumn. There is no good evaluation of the approach, so how well the Soft Actor-Critic performs compared to traditional methods is not known. The authors state that the model is not ready for deployment.

In most literature applying reinforcement learning to hydropower scheduling, the objective does not involve prices and instead only aims to optimize energy production subject to constraints. Energy production can be optimized by taking into account that efficiency varies with the amount of discharge and the amount of water in the reservoir. In addition, the reservoirs have constraints on how much water can be stored and discharged. These papers, therefore, solve a similar, but simpler optimization problem than described in this thesis. Xu et al. (2020) uses deep Q-learning to solve a problem with cascading hydropower plants. Cascading hydropower plants mean that the discharge of an upper reservoir feeds into a downstream reservoir. When the action space consists of multiple discharge decisions, one for each reservoir, the possible combinations of decisions increase exponentially. Two techniques are used to overcome this problem. The first is to use an aggregation-disaggregation model to combine multiple reservoirs into one. This reduces the state- and action space. In addition, the action space is discretized, changing it from discrete time and continuous action to discrete time and discrete action. They use a negative reward for breaking constraints.

Li et al. (2023) reviews the use of transformers in reinforcement learning. The use of transformers has seen success on sequential data, and applying it to reinforcement learning might seem promising. They find several challenges, notably that transformers typically have high latency and memory requirements, which increases training time. There are also problems when the policy changes during play, which introduce non-stationarity. This non-stationarity affects transformers more than other shallower neural networks, according to the authors. In cases where offline reinforcement learning can be used, transformers have achieved state-of-the-art performance. The advantage of transformers is that they can capture complex temporal relations, but they have a tendency to capture patterns that are just noise, which leads to over-fitting. In our application, the temporal relations are not very complex, and simpler architectures are likely to do better. These are all arguments for not using transformers for the hydropower scheduling problem.

A more recent state-of-the-art reinforcement learning algorithm is the Distributional Quantile Critics (Kuznetsov et al., 2020). The work outperforms the Soft Actor-Critic on most continuous benchmarks. The authors introduce three main ideas. Firstly, the critic estimates the distribution of the cumulative reward using quantile regression, instead of estimating the expected cumulative reward which is more common for critics. Secondly, to prevent overestimation bias the critic's highest quantiles are truncated. How much of the distribution is truncated can be controlled to balance between overestimation and underestimation. Thirdly, the value distribution estimates are improved by using an ensemble of critics. Because this model has faster convergence than the Soft Actor-Critic on a wide variety of problems, it might also perform better on the hydropower scheduling problem. As we will see in the next paragraph, incorporating uncertain estimates and ensembles to model-free reinforcement learning has given decent results on the HVAC problem.

Luo et al. (2022) looks at the HVAC problem, which involves keeping the temperature of a building within an acceptable range while minimizing power usage. This paper does not include power prices in their problem. The authors propose a solution using model-free reinforcement learning but find a number of ways to make use of domain knowledge. They deal with a large action and state space by discretizing the continuous action space to a large, discrete one. The discretization is non-uniform to ensure critical intervals are more granular. Close to actions that were previously chosen, the agent gets many candidate actions to choose between. Further away, the candidate action selector is coarser. Action pruning avoids the curse of dimensionality. The hydropower scheduling problem has parts of the action space that is unlikely to be optimal, for example releasing discharge close to the minimum possible value is very inefficient.

They further improve performance by using an ensemble of neural networks to predict state-action values. The mean of the predictions is used as a final prediction, but the standard deviation is also used as a measure of uncertainty. During exploitation, the agent prefers high mean and low standard deviation and picks actions it is confident will be quite good. During exploration, the actions that have a high predicted standard deviation are more likely to be selected. The model also has separate networks that predict if actions are likely to violate constraints. Actions with high predicted risk are then pruned based on the probability of constraint violation. An optimal neural network policy would need to model the risk of constraint violation implicitly, but explicitly predicting risk can lead to convergence with less data. The experimental results show that their solution performs better than common model-free reinforcement learning algorithms. The solution was also deployed to real buildings, which saved power compared to the older control system.

Because the problem formulation does not include prices or forecasts, the problem is quite different compared to the problem formulation in this thesis. However, the usage of action pruning to deal with large action spaces, discretization and ensemble networks is relevant.

The model-free algorithms discussed here would, after training, have more than good enough query time. However, even though Soft Actor-Critic and Distributional Quantile Critics are considered sample efficient, this is compared to other model-free reinforcement learning algorithms. Compared to many physical models or methods that incorporate domain knowledge, both methods will likely require much more samples and computation time. Some literature reviewed later that deals with very large state and action spaces finds that model-free algorithms do not perform great and propose more sample-efficient methods that outperform model-free algorithms. Because implementations of both algorithms are available, using these methods as a benchmark could be useful.

3.2.3 Cross-entropy method for continuous control

Pinneri et al. (2020) uses the cross-entropy method, discussed in Section 2.2.4, to tackle continuous reinforcement learning problems. The authors evaluate their methods on several common continuous control benchmarks. The environments involve manipulating multiple joints and handling contact with surfaces to achieve some objective. In Humanoid Standup, the goal is to move the limbs of a human-like creature in order to get from a lying position to a standing position. Another example is the Half Cheetah environment, where a cheetah learns to move its legs to maximize velocity. Both environments include Gaussian noise for friction, meaning the environment is not deterministic.

The authors assume that the model of the environment is available for planning, and define the objective function $f(\vec{a})$. This is the accumulated reward when applying the actions in the trajectory $\vec{a}$ to a starting state. The cross-entropy method is used to try thousands of trajectories and select the first action of the trajectory with the best performance. A more detailed explanation of cross-entropy was given in Section 2.2.4, but we reiterate the basic idea here. Pinneri et al. (2020) samples action trajectories from a multivariate Gaussian probability distribution. A multivariate Gaussian is parameterized by a mean and a standard deviation for each action a_t in the trajectory, $N(\mu_t, \sigma_t^2)$. To optimize a function, they sample n action trajectories $X_1, X_2, ..., X_n$ from the initial distribution. X_i is here a trajectory of actions $\vec{a}$ that can be used to simulate the environment for T time steps. Each action trajectory X_i is then evaluated on the objective function, $f(X_i)$, and the m best samples are chosen as an elite set. This elite set is used to update the sampling probability distribution to improve future sampling. To update the probability distribution, they calculate the mean of the elite set, and set the mean of the new distribution as the weighted average of the old distribution and the elite mean, as shown in Equation 3.3. This means the probability distribution center will slowly move towards areas associated with high-performing samples. To update the dispersion, they use Equation 3.4 which similarly to the mean update is a weighted average of the old standard deviation and the standard deviation of the elite set. α can be tuned to get faster changes to the normal distribution, though a low α increases the risk of getting to a local optimum. When the algorithm has finished its iteration, the first action of the best trajectory encountered is returned.

$$\mu_t^{\text{new}} = \alpha \mu_t^{\text{old}} + (1 - \alpha)\mu^{\text{elite-set}} \tag{3.3}$$

$$\sigma_t^{\text{new}} = \alpha \sigma_t^{\text{old}} + (1 - \alpha)\sigma^{\text{elite-set}} \tag{3.4}$$

In addition, they test a number of smaller improvements, such as keeping some of the elite set for the next iteration, and if they are good enough, they might be selected to the elite set again, and give a better update. They also find decaying the number of samples n in later iterations can reduce running time without substantially impacting solution quality. Because the dispersion decreases over time, there is less need for huge sample sizes in the later iterations. They point out that an important condition for this method to be effective is that the computation time of evaluating a trajectory x on $f(x)$ needs to be low because f needs to be run perhaps millions of times to get good results. This is also an any-time algorithm, meaning it can return after any iteration, but the solution may get better with more run-time.

Another application of the cross-entropy method is by Chua et al. (2018), who learns a transition model during training and then applies the cross-entropy method to the learned dynamics model. They compare their method with model-free algorithms like the Soft Actor-Critic and find that their model is more sample efficient and converges faster than the Soft Actor-Critic. They use the same continuous control problems discussed earlier in this section. The main contribution of this work over Pinneri et al. (2020) is not so much the planning, but the learning of the dynamics model. They separate uncertainty inherent to the environment and uncertainty stemming from too little experience. The latter can be corrected by sampling more experience. They train a transition model by doing real trajectories and collecting data $\{(s_t, a_t, s_{t+1}), ...\}$. They then train an ensemble of neural networks that predict a probability distribution over the next states given the previous state and action, $\tilde{f}(s_{t+1}|s_t, a_t)$. They then plan on the model similarly to Pinneri et al. (2020), and even with the learned model, this is much more sample efficient than models like the Soft Actor-Critic.

Finally, we note that the cross entropy method has been used for more than planning in reinforcement learning. Some examples include approximating solutions for the traveling salesman and combinatorial optimization for relevant problems like the unit commitment problem (Ernst et al., 2007). The latter is a very simplified version of the unit commitment problem, but they state that the cross-entropy method almost always finds the optimal solution with quite few iterations and samples per iteration.

3.2.4 Differentiable simulators

Madeka et al. (2022) tries to model the inventory management problem. Keeping insufficient inventory results in missed demand, while keeping an excessively large inventory increases storage costs. They use the reward function $R_t = p_t \min(D_t, I_t) - c_t a_t$. The revenue is given by the price times the demand D_t, or if there is not enough inventory to cover the demand, the price times the inventory I_t. Both I_t and D_t are random variables, where I_t depends on earlier actions, which involve ordering items from vendors. Ordering items from vendors has a unit cost of c_t, and takes a random amount of time steps to arrive. The authors prove that the accumulated reward, $E[\sum R_t]$, and the transition function are differentiable. Instead of directly optimizing the goal with regard to the actions, they instead try to train a policy using the following procedure. Consider the policy, $\pi_\theta(s_t) = a_t$, where θ is the parameters of the policy, for example, weights in a neural network. They can then use this policy and an initial sampled state to roll out an episode and compute the gradient of the objective function with regard to the parameter θ. They then update the policy with the gradient, $\theta_b = \theta_{b-1} + \alpha \nabla_\theta J_b$, where α is the learning rate and $\nabla_\theta J_b$ is the gradient of the objective function with regards to the parameters θ. This approach outperforms model-free reinforcement learning. The authors test the model by using a randomized control trial and find that the method outperforms baseline methods by 12%.

However, the hydropower scheduling problem has discontinuous points, meaning that it is not possible to take the derivative at all points. This technique would therefore not necessarily converge to a global optimum, due to the discontinuous points. Another way to use the differential property could therefore be to use a non-differential method to get close to the global maximum and use the differential property to fine-tune from this starting point.

3.2.5 Tree search and rollouts

Powell (2014) reviews a number of problems like battery arbitrage, unit commitment problem and hydropower scheduling. The author suggests using look-ahead policies when many uncertain forecasts are available. In the case of deterministic forecasts, doing rollouts gives us results and we can select the trajectory with the best solution quality that does not violate any constraints. In the case of uncertain forecasts, one can either sample a possible outcome or try to calculate the expected outcome from a rollout. The author claims that simulating rollouts often performs well when forecasts are available, but without pointing to any specific experimental results.

Mars and O'Sullivan (2022) tries to solve the unit commitment problem. Given a number of gas generators that can either be switched on or off, the agent must try to match supply to demand, handle constraints and minimize the cost. The generators have different costs and characteristics, which means that turning on generator x instead of generator y does not necessarily give the same result. Switching off a generator involves some downtime, so the generator can't be used for a period of time. There is also a start-up cost for turning on a generator, fuel costs and a cost for if demand is lower than supply. With n generators and t time steps to plan, each state has 2^n possible actions, which gives 2^{nt} possible outcomes. The demand and supply are unknown but uncertain forecasts are available to the model.

The author uses tree search with heuristics to reduce the branching factor to get a good trade-off between solution quality and running time. The nodes of the search tree are the state, and the edges are actions leading to the next states. They estimate the weight of each edge, which is the estimated cost of taking that action. These weights are estimated by doing many rollouts, where each rollout samples a possible forecast. The result is the expected cost of taking the action in that particular state. The problem can then be seen as finding the lowest cost path through the search tree.

The heuristics drastically decreases the required running time to get a good solution quality. The main heuristic is to rank order the generators according to their marginal fuel costs, and then apply the generator with the lowest marginal fuel cost until demand is met. This is not necessarily optimal because the generators with the lowest marginal fuel cost might be unavailable if turned off at a later point, but relaxing this temporal complexity can still yield decent results and much better running time. They also set a maximum branching factor, meaning no more than n actions can be tried for each state. In addition, they set a maximum depth, H and solve the sub-problem of minimizing cost starting from the initial state s_t until horizon H instead of the full planning horizon. They then select the best initial action, apply it and solve the minimization problem starting from s_{t+1} until horizon $H+1$. This is done iteratively until a plan for the full planning horizon is finished. The method has a much lower run time compared to uniform cost search, but trivially higher costs.

Chapter 4

Architecture

In this chapter, we explain our architecture. Our main concern is to discuss how we adapted the methods to our problem, what informed our design choices and the theoretical strengths and weaknesses of different choices. Implementation details such as hyper parameters and training time are in the next chapter (Section 5.2). In Section 4.1, we explain the assumptions in our model of the environment. Next, how we adapted the Soft Actor-Critic to the hydropower problem is discussed in Section 4.2. Finally, we explain the main contribution of this thesis in Section 4.3. This is a pipeline of methods we have called Heuristic tree search with gradient fine-tuning (HG).

4.1 Hydropower environment assumptions

This section gives an explanation of how we modeled the hydropower problem. We list the assumptions of our model of the environment and discuss how these assumptions could lead to a performance gap between our experiments and actual deployment.

The implementation of the environment has several assumptions that could make the simulated experience different from real-world experience:

1. Price-taker. Production decisions do not affect prices.
2. Prices and inflow are independent, $p_t \perp\!\!\!\perp q_t, \forall t \in T$
3. The first-order Markov assumption holds for prices and inflow
4. Inflow and prices are stationary processes
5. The relation between the true price or inflow and their respective forecasts is modeled correctly.
6. The relation between the rest price and future prices is modeled correctly.

7. Forecasts for time t with a given issue date f become less accurate for a greater difference between the issue date of the forecasts and the time t. We can then expect the following to hold: $(p_{t,s,f} - p_t)^2 < (p_{t+a,s,f} - p_{t+a})^2$

Assumption 1, that the choice of production does not change prices is likely approximately correct. The maximum hourly production is 14 MWh for the Mørre power station, which likely does not make a significant difference. On the other hand, demand for power is known to be inelastic, meaning that small changes in quantities can lead to large changes in prices. When optimizing larger powerplants or groups of them, this assumption could be more problematic. However, modeling the causal effect of production level on prices is difficult, so we do not have much choice.

The second assumption, that inflow and price are independent, might not be entirely true. Data from late 2019 to the middle of 2022 shows a -0.13 correlation between price and inflow to Mørre, indicating that when the inflow is large, we more often observe low prices. One way to solve this could be to model the Markov chain so that the next state of the price depends on both its previous state and also the state of the inflow. However, that would require a bigger transition matrix and more data to get good estimates of each transition probability, so this idea was scrapped.

The assumption that price and inflow are first-order Markov chains means that they have conditional independence to earlier states given their previous state. Again, this might be a good approximation, but not entirely true. Consider an example where some event like melting snow will create a very large inflow for some period of time. When this event occurs, we know that the future inflow for many time steps will not be as great, which could be one example of the first-order Markov assumption not being true.

The most problematic assumption is likely about the stationarity of the processes. Price data has daily seasonality, weekly seasonality and yearly seasonality. The autocorrelation for the price data shows that the correlation between a point and the lag that is a multiple of 24 is generally greater than other lags, which indicates daily seasonality. The inflow has yearly seasonality. In our historical inflow data, the month of March has almost three times higher average inflow compared to February. In addition, patterns have changed over time. Price volatility was much lower in 2020 than in 2021, with the former having a standard deviation of 6.9 compared to 27.2. The average price has also changed. The mean price in 2020 was 9 EUR/MWh and the mean price in 2021 was 41 EUR/MWh. This indicates that the synthetic data encountered in our simulation may be very different compared to the real world.

Another concern is that the relation between the forecasts and the true values is not modeled correctly. Some examples could be that the real-world forecast errors are much greater than in our simulation, in which case our model would perform worse. The distribution of the forecast error also matters, for instance, if it is mostly very close to correct but sometimes has a very large error.

The sixth assumption is false because the rest price is generated randomly and does not actually represent the marginal water value. This can create problems for how we evaluate agents. In production, agents plan for the period T but only commit decisions for the next day. In an episode, the first planning session might be for the period $t = 0$ to $t = T$, the second for $t = 24$ to $t = T+24$, and the last for $t = T-24$ to $t = 2T-24$. Because at time T we end the episode and give a reward for using the rest price, the planning involving data after this point will not be optimal for maximizing profit in the simulation. For example, if the price forecasts after T are very high, the agent might save water, even though when optimizing for $t = 0$ to T the optimal would be to produce. This effect can be so large that we chose to not replan every day, but instead, plan at time $t = 0$ and submit all actions until the episode ends, as we mention in Section 5.2.

The seventh assumption is that forecasts are more accurate when the issue date is close to the date to be forecasted. For instance, one would expect a forecast about the inflow tomorrow to be more accurate than a forecast about the inflow in 12 days. We have modeled this in our synthetic data generation, however, there is not much thought into exactly how much forecast accuracy decreases. If the forecast accuracy is much lower after 12 days, then our choice to use one-stage optimization is more problematic.

4.2 Model free reinforcement learning for hydropower optimization

This section details our main design choices to adapt the Soft Actor-Critic to our problem. In Section 4.2.1 we explain how we represented the state for the Soft Actor-Critic agent. The action space is explained in Section 4.2.2, and finally we discuss how we used reward shaping to speed up learning in Section 4.2.3.

4.2.1 State-space representation

The state representation provided to the Soft Actor-Critic algorithm is designed to be effective with a shallow neural network. To avoid issues such as exploding gradients and certain features overpowering others, all values in the state are normalized within the range of $[-3, 3]$, or sometimes within a tighter bound.

One challenge with our problem formulation is that including all price and inflow forecasts would result in 23328 features. The state space would be very large and to explore it enough to create a reasonable mapping to good actions would require a very large training set. We deemed that the required computation and training time to get reasonable results was not feasible. Instead, the features below try to compress the most relevant information to just 10 features. The first two features show the startup cost if the agent chooses to produce at this timestep. The second shows the startup cost relative to the rest price because if prices are very high, the startup cost does not affect the decision as much. The third and fourth feature indicates how close the agent is to break the maximum or minimum reservoir constraints. The reservoir filling is a number between 0 and 1, which correspond to the maximum and minimum reservoir levels. In addition, we include a measure of the risk of overflow, which roughly measures how much time it would take for an overflow to occur if all actions selected were 0. This feature is an interaction between the reservoir filling and the inflow. Lastly, features 5 and 6 deal with prices. The fifth shows the price quantile of the forecast of the current price, relative to all later prices. 6. is really 5 features, which are the prices relative to the rest price for the next 5 time steps. If the current price forecast is higher than the rest price, this is a good indicator of whether or not the agent should produce. If the price forecasts for the next few steps are also high, then selecting a non-zero action is more tempting because one can avoid paying the startup cost for the next actions.

1. Shows if a startup cost is needed if the agent chooses a nonzero action.
2. Startup cost for producing next timestep, relative to the rest price.
3. Reservoir filling, between 0 and 1.
4. A measure of overflow risk, which is an interaction between the reservoir filling and the inflow.
5. The price quantile, which shows how high the current price forecast is relative to other prices in the forecasting horizon.
6. The next 5 average price forecasts relative to the rest price.

4.2.2 Action space

The legal actions are $a \in 0 \cup [10, 22]$, but we make some adjustments because we sample actions from a normal distribution. We want to accomplish three things. Firstly, we want to normalize the action space to be between $[-1, 1]$, because this can give greater stability to the learning process. Secondly, we want to avoid illegal actions between 0 and 10. Thirdly, we want a high probability of choosing the action 0, because otherwise, the agent will often break the lower constraint. To allow the agent to select the action 0 more often, we map an interval to the action 0. Setting an interval to map to zero increases the likelihood of choosing zero. As a result, our action space is $[-1, 1]$ where $[-1, -0.8]$ map to 0 and $(-0.8, 1]$ map to $[10, 22]$. The latter interval is scaled linearly.

4.2.3 Reward function design

In an attempt to speed up training, we use reward shaping. The idea is that we want the agent to quickly understand that. Consider two identical states except for the rest price. In the case where the rest price is very low, producing large amounts of power is much more beneficial than in the case where the rest price is much higher than the current price. If we just gave the reward $w_t p_t - c_t$ the agent would not get the feedback to understand this until the end of the episode. Consequently, we use the following rules to give rewards to the agent:

$$\lambda R_t = \begin{cases} w_t(p_t - e_T) - c_t & \text{No constraints broken} \\ -10^6 & \text{Lower constraint broken} \\ I(x_t, e_T) + \sum w_t e_T & t = T \end{cases}$$

The main adjustment is that we subtract the rest price, e_T from the current price. This gives the agent an indication of whether or not the benefit of producing now outweighs the lost rest revenue. However, we do not know the exact reduction in future rest revenue because the rest revenue is not linear as a function of the amount of water left. To correct this and make sure the agent gets the correct accumulated reward for an episode, we return the correct rest revenue at the end of the episode and add the incorrect rest revenue to cancel the penalties that were given during the episode. This is given by $I(x_t, e_T) + \sum w_t e_T$. This ensures that the agent gets both more immediate feedback and that the accumulated reward over an episode is correct.

Lastly, we scale the reward by a constant λ. Large rewards of a million can create inefficient usage of neural network parameters and very large gradients that lead to large updates of the neural network parameters. This can create instability in the learning process and inefficient usage of parameters.

4.3 Heuristic tree search with gradient fine-tuning

Heuristic tree search with gradient fine-tuning (HG) is the main method developed in this thesis. It consists of a pipeline with four methods that progressively improve our solution. The first three methods can handle discontinuous points and can be seen as an exploration of the solution space using heuristics. The last phase uses gradient ascent to fine-tune the best solutions found by the earlier phases. In Section 4.3.1, we give an overview of the pipeline and a high-level explanation of our approach. Next, we explain the objective function used by all phases in the pipeline in Section 4.3.2. We then go through each phase in order, starting with seeding (4.3.3), then Monte Carlo tree search (4.3.4), the cross-entropy method (4.3.5) and finally gradient ascent (4.3.6).

4.3.1 Overview of pipeline

There are four phases in the pipeline, where the first 3 methods can handle discontinuous points, and the last is a fine-tuning of the best trajectories found earlier using gradient ascent. The main reason we chose this structure is that the last method does not handle discontinuities and therefore can not reach most points from a given starting trajectory. Gradient ascent works by simulating our environment with an action trajectory and then computing the gradient of the objective function with regard to the action trajectory. We can then optimize our actions by increasing the actions where the derivative is positive and decreasing our actions where the derivatives are negative. However, this update does not work in discontinuous points

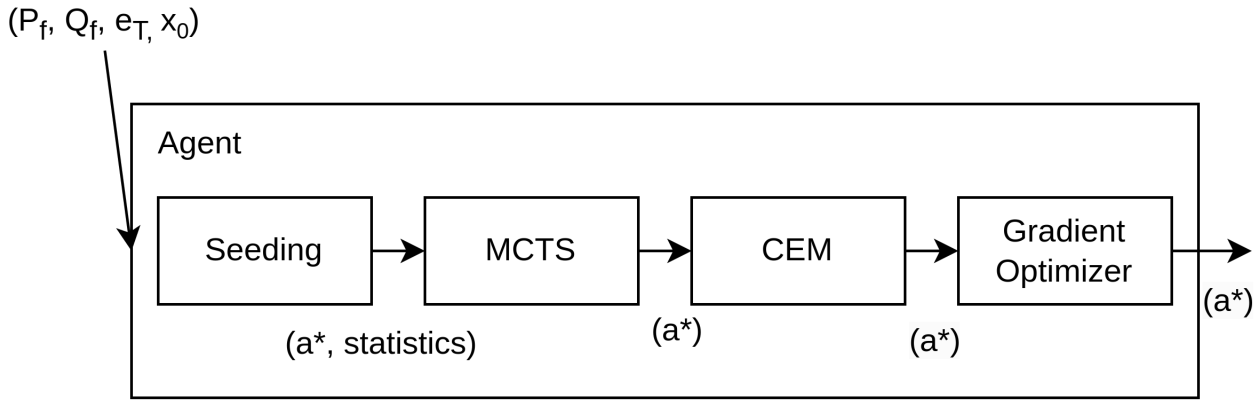

Figure 4.1: Agent pipeline

which means gradient ascent is susceptible to local maxima and not guaranteed to converge to the global maximum. Our first three phases, therefore, use methods that are not vulnerable to discontinuities to try to solve the problem of getting close to the optimal maximum, so that the last phase can reach the global optimum. Figure 4.1 shows the flow of data in the agent. First, the price forecasts, inflow forecasts, the rest price and the initial filling are entered into the system. This information is used to compute features and evaluate suggested trajectories with our objective function. Then each stage of the method finds the n best action trajectories, a^* and feeds them into the next phase as a starting point. The seeding phase in addition feeds some statistics to the Monte Carlo tree search phase to initialize the search tree. The last method, the gradient optimizer returns the best trajectory which is the final trajectory.

The four methods listed below are in decreasing order of coarseness of the action space and use the trajectories of earlier phases as a starting point.

1. Seeding
2. Monte Carlo tree search with heuristics
3. Cross entropy method
4. Gradient optimization

4.3.2 Objective function

Each step in the pipeline tries to find trajectories that score well on the objective function. When comparing two trajectories, we prefer the one that scores the highest on this objective function. To accomplish this, we implemented the function $f(\vec{a})$ which computes this objective, but with some changes to increase performance. Because this function will be called many times, its running time is critical. As a basis for the objective function, we use Equation 2.8 as specified in Section 2.1.4.

The function is reiterated here:

$$\frac{1}{|\mathcal{C}_f|}\sum_{\tilde{s}=1}^{|\mathcal{C}_f|} J(\vec{d}, \vec{p_s}, \vec{q_s}, x_0, e_T),$$

where $\mathcal{C}_f$ is the set of all pairs of price and inflow scenarios, $\mathcal{C}_f = P_f \bigtimes Q_f$.

It can be proven that in this specification, the uncertainty in price does not affect the profit. We show this in Equation 4.1. Because the prices only interact linearly with the decision, it does not matter if we compute the profit with the average price or if we take the average profit over all possible price scenarios. Because summation is commutative, we can move the summation over the price scenarios to the price, and replace the price with the average price over the scenarios. The result will be equivalent to computing the profit for each price scenario and taking the average of the profit. The difference is that computing the average price is faster and allows us to represent future prices more efficiently.

$$\begin{aligned} V(d_t) &= \frac{1}{|Q_f||P_f|}\sum_{\tilde{s}=1}^{|Q_f|}\sum_{\tilde{s}=1}^{|P_f|}\sum_{t=1}^{T}(p_{t,\tilde{s}}w_t - c_t) \\ &= \frac{1}{|Q_f|}\sum_{\tilde{s}=1}^{|Q_f|}\sum_{t=1}^{T}\left(\left(\frac{1}{|P_f|}\sum_{\tilde{s}=1}^{|P_f|}p_{t,\tilde{s}}\right)w_t - c_t\right) \\ &= \frac{1}{|Q_f|}\sum_{\tilde{s}=1}^{|Q_f|}\sum_{t=1}^{T}(\bar{p}_t w_t - c_t) \end{aligned} \tag{4.1}$$

However, uncertainty in inflow does matter because of the non-linear relation between inflow and profit. Firstly, a different inflow changes the power production through the head of all future time steps, and is thus non-linear, as described in Section 2.1.1. In addition, the inflow also changes the rest value which is also a non-linear function of the reservoir-filling. We discussed the rest-value in Section 2.1.3. The effects of inflow uncertainty are small, except when one scenario breaks the reservoir constraints. In this case, uncertainty in inflow has a large effect on the profit.

We can then restate our objective function as Equation 4.2, where $\bar{p}_t = \frac{1}{|P_f|}\sum_{\tilde{s}=1}^{|P_f|} p_{t,\tilde{s}}$. Because each price scenario occurs with uniform probability, we simply take the average price over the price scenarios. We then compute this profit for each inflow scenario, and take the average of these profits. This is our metric for how good a trajectory is, and gives the expected profit of the trajectory $\vec{d}$ across all scenarios with uniform probability.

$$\frac{1}{|Q_f|}\sum_{\tilde{s}=1}^{|Q_f|} J(\vec{d}, \bar{\mathbf{p}}, \vec{q_s}, x_0, e_T) \tag{4.2}$$

The only two changes compared to the deterministic objective is that the actual price trajectory is replaced with our expected price trajectory, that is $\bar{\mathbf{p}} = \frac{1}{|P_f|}\sum_{\tilde{s}=1}^{|P_f|}\vec{ps}$.

Here, $\vec{ps}$ is a vector with prices for each time step for a scenario s. Because each price scenario occurs with uniform probability, we simply take the average price over the price scenarios. We then compute this profit for each inflow scenario, and take the average of these profits. This is our metric for how good a trajectory is, and gives the expected profit of the trajectory $\vec{d}$ across all scenarios with uniform probability.

In addition to this change, we also make some changes to speed up the code. If we break the lower constraint for a single inflow scenario, we immediately return the punishment without computing revenue, startup cost and other things. The reasoning is that an optimal strategy should never break the lower constraint, so we would rather quickly return a negative value without bothering to get the exact estimate. There are also some changes to how overflow affects the objective value, which can lead to some divergence between the true accumulated reward and the objective value. The changes in the case of overflow are discussed in Section 5.2.

Our last change is that we introduce a buffer reservoir filling because it is possible to get a true inflow that is more extreme than any of the scenarios. We do this by defining $x_{\min}$ and $x_{\max}$ to be slightly larger and smaller respectively, resulting in a more restrictive constraint in order to reduce the chance of breaking constraints.

4.3.3 Seeding

We mentioned the concept of seeding as an initialization of Monte Carlo tree search in Section 2.2.3. The main idea is to initialize the tree search with useful values to speed up the search. Because a trajectory of discharge actions can be evaluated very fast, one option is to evaluate many trajectories generated from simple, fast rules like producing at a pre-specified level if and only if prices exceed a certain threshold. The evaluation of the trajectory can give useful statistics like if the reservoir constraints were violated, and in which direction, and how much profit certain actions led to. We can save computation by avoiding the traversal of a tree where we need to decide a_t before a_{t+1}. This allows us to test more trajectories in the same running time.

The seeding phase uses fast heuristics to test many promising strategies. In less than a second, 3×288 trajectories can be evaluated, much faster than when actually traversing a tree. Most of these results are saved and used to initialize the search tree.

We make an approximate rank ordering of the potential of each action. This was inspired by the work of Mars and O'Sullivan (2022) discussed in Section 3.2.5. It is very fast and finds reasonable, but often sub-optimal solutions. To carry out this method, we first compute the gradient of each action with regard to the profit at the same discharge level. We then sort the actions in descending order of the gradient, establishing an order that approximately reflects the benefit of producing at that time. The average price will be the main driver of the gradient, so this is mostly an ordering of actions by price, but it also takes into account that all later actions will get less reservoir height, which slightly reduces the power. We start by evaluating a trajectory where all discharges are set to 0 on the objective function. We then progressively add $17m^3/s$ discharge to the most promising action until all actions are nonzero. In addition, to avoid unnecessary start-up costs, we also try to remove all actions that are nonzero but where the previous and next action is zero, which results in an extra startup cost. In cases where two nonzero discharges are separated by a time step with zero production, we also try to set that middle time step to $10m^3/s$ in order to avoid an unnecessary startup cost. In total, $3T$ trajectories are evaluated, and we return the best trajectory.

The usage of gradients in this way does not cause problems with discontinuous points, because the gradients are all computed at the same discharge level and at a reservoir level which ensures we do not break any constraints. This heuristic is approximately the same as using prices but also incorporates some more information, which we discuss in Section 4.3.6.

The two main reasons for using this method are that the method is very fast and that the heuristic works quite well. In a typical optimal solution, the time steps with the highest price will likely have nonzero production. By heavily restricting our possible actions and ignoring many complexities of the problem we can get some reasonable solutions fast, even though the solution probably will not be close to a global maximum. Most discharges are set to $17m^3/s$ in this method because this is what maximizes the power per unit of water. This makes our method vulnerable to local maxima, which is a disadvantage of using this simple method.

4.3.4 Monte Carlo tree search with heuristics

Our second phase uses a modified version of Monte Carlo tree search, which we discussed in Section 2.2.3. To Adapt Monte Carlo tree search to the problem in this thesis, some issues must be addressed. The actions are continuous for our problem but need to be discrete for Monte Carlo tree search. Consequently, we discretize our actions into four actions: $\{0, 10, 17, 22\}$. $17m^3/s$ is the most efficient discharge level per unit of water, and 10 and 22 are the minimum and maximum possible discharges. With 4 possible actions at each step, we get a large search space of 4^{288} due to a planning horizon of 288 steps. This is impractically large for a complete search. Consequently, employing a naive UCT search is not possible, and we instead rely on the tree policy in Equation 4.3:

$$\begin{aligned}\pi_{\text{tree}}(a_i) &= \text{max value bonus} + \text{exploration bonus} + \text{model bonus}\\ &= \frac{v_i}{v^*} + c_1\sqrt{\frac{\ln N_i}{n_i}} + c_2\pi_{\text{model}}(a_i|X) \qquad (4.3)\end{aligned}$$

The tree policy selects the action with the highest value in this equation. The first term for choosing action a_i is given by a max value bonus, which is the best accumulated reward encountered after selecting that action. This value is normalized by dividing by the best accumulated reward found during the seeding phase, and so v_i/v^* will be greater than 1 if this trajectory has led to a greater reward than the best trajectory from the seeding phase. In normal UCT, an average accumulated reward is used instead of the maximum. This would not be useful in our approach, because if just a single trajectory breaks the lower constraint then the large negative profit would dominate the average value. By using the maximum, we don't penalize risky but high-reward trajectories. This is especially important because in many cases, the optimal solution will be on the brink of violating constraints in our environment. In the second term which is the exploration bonus, N_i is the number of times the parent node has been visited, while n_i is the number of times the child node has been visited. If the ratio of visits to the parent node compared to the child node grows, the exploration bonus increase for that child node. The constant c_1 can be increased to make the tree search more exploratory. Lastly, the model bonus is given by predicting the probability of choosing that action with the same models used during the rollout phase, which we will now discuss.

During the rollout phase we either use a random model to classify the action, or stop the rollout early and use the end of the best trajectory to complete the simulation. We use the classifiers Lightgbm, linear discriminant analysis and naive Bayesian classification. These models were chosen because they are fast to train and infer with. Their hypothesis space is quite simple, which means that they can get reasonable results with limited data. The following features were designed to have a simple relation to the optimal action, and are used as input to the models:

1. The average price to rest-price ratio for this time step
2. Average inflow for this time step
3. The gradient scaled by dividing with the maximum gradient for this time step.
4. The price quantile for this time step compared to other prices in the trajectory.
5. The distance from violating the lower constraint given 0 discharge.
6. The distance from violating the upper constraint when choosing no discharge for all actions.

These features are mapped to the actions of the best trajectory encountered during Monte Carlo tree search. Because the training set for the models retains data from earlier searches, the models should learn to generalize and become better over time.

When the rollout phase has decided on its last action, we use the objective function to compute the profit for that trajectory and propagate the result back to the nodes. We also maintain a max heap with the four best trajectories which are candidates for multi-start gradient. So this phase gives some extra robustness by adding many start alternatives in addition to the extra exploration of the solution space.

An advantage of Monte Carlo tree search is that it offers us a way to try the actions that seem best first, before progressively trying less promising paths. As the most promising actions are tested, the exploration bonus for other actions will increase, so some alternative trajectories are also tested. This is an anytime algorithm, so we can decide how long we want the search to be and likely return a decent solution quite fast. Furthermore, by using trainable models, the method should get better over time.

A disadvantage is that the exploration is mainly focused at the start of the trajectory, as this is where we build the search tree. In the rollout phase, the algorithm does not have access to statistics about node visits and so is more exploitative. This means that the later action possibilities are less explored, which can be problematic because the optimal choice of the actions at the beginning of the episode depends heavily on the choices in the later parts of the trajectory.

Another disadvantage compared to traditional methods is that choosing the heuristics for one hydropower model might not fit other hydropower models or market conditions. This method might therefore require laborious experimentation and domain knowledge in order to adapt it to other power plants.

We also have a restricted action space where the optimal solution might not be present, just like the seeding phase. Because we have later phases that further improve the solutions from this phase, the main question is not if this method can find the optimal solution, but whether or not the optimal solution is reachable from the returned solutions using gradient ascent. For this to be the case, those actions that are nonzero in the optimal solution should be nonzero in the proposed solution. This is not guaranteed with only the four actions above. Methods like value iteration or the Soft Actor-Critic have the property that given the standard assumptions of reinforcement learning, the methods should eventually converge to an optimal solution. The problem sizes are often so large that convergence is infeasible, but with infinite computation, the methods should eventually get the optimal solution. This is not always the case for HG with just four actions, which is a weakness of this approach. We could increase the number of actions available to the tree search at the cost of a much larger search space.

4.3.5 Cross-entropy method

The third phase of the model uses the cross-entropy method to optimize, using the best trajectory from Monte Carlo tree search as a starting point. If CEM manages to improve this trajectory, then the improved trajectory is passed to the next phase. We discussed CEM in Section 2.2.4 and 3.2.3. We use the method as described in 3.2.3 without the suggested improvements. The only adaptation to this environment is that we allow the agent to sample negative values and clip them to zero. The intention is to increase the probability of sampling actions equal to zero, as the model would otherwise often break the lower reservoir boundary.

This method can technically represent the whole action space, unlike our two preceding methods which can only use a limited subset of the action space. The aim of including this method in the pipeline was to give HG the opportunity to fully explore the action space in the hope of avoiding ending in local maxima.

4.3.6 Gradient optimization

In the last phase of the HG method, we use multi-start gradient ascent starting from the n best trajectories. Gradient ascent run in parallel in separate processes to optimize the n trajectories. The best-performing trajectory is returned and used as a final output of the system.

We defined our objective function earlier in Equation 4.2. We will refer to it as $f(\vec{d})$ in this section. We want to compute the gradient of this objective function with regard to the trajectory of discharge decisions:

$$\nabla_{\vec{d}} f(\vec{d}),$$

The resulting gradient points in the direction of the steepest ascent, meaning the direction that will lead to the greatest increase in profits. With this information, we can optimize our trajectory by increasing the discharges with positive derivatives and decreasing our discharges with negative derivatives. The standard update equation is given by:

$$\vec{d} = \vec{d} + \alpha \nabla_{\vec{d}} f(\vec{d}),$$

where α is the learning rate.

We will now explain how the gradient can be computed. Equation 4.4 shows the derivative of the objective function with regards to the discharge decision at step i.

$$\frac{\partial f(\vec{d})}{\partial d_i} = \frac{1}{|Q_f|} \left(\sum_{s}^{Q_f} \left(\bar{p_i} \frac{\partial w_{i,s}}{\partial d_i} + \sum_{t=i+1}^{T} \left(\bar{p_t} \frac{\partial w_{t,s}}{\partial x_{t,s}} \frac{\partial x_{t,s}}{\partial d_i} \right) \right) + \frac{\partial I(x_{T+1}, e_{T+1})}{\partial x_{T+1}} \frac{\partial x_{T+1}}{\partial d_i} \right) \tag{4.4}$$

Intuitively, the effect of increasing d_i works through three mechanisms. Firstly, increasing d_i leads to more direct revenue through $\bar{p_i} w_i$. Here, the power produced, w_i, increases as a function of d_i, but the increase is diminishing: $\frac{\partial \bar{p_i} w_i}{\partial d_i} > 0$, $\frac{\partial^2 p_i w_i}{\partial^2 d_i} < 0$. The relation between discharge and power output was illustrated earlier in Figure 2.2. Assuming positive prices, $\bar{p_i} w_i$ is always non-negative. Increasing the amount of discharge gives more power and more revenue but with diminishing return because $\partial^2 w_t / \partial^2 d_t < 0$.

Secondly, using more water now means we have less water later which gives us less rest revenue through $I(x_{T+1}, e_{T+1})$. Increasing d_i reduces all subsequent reservoir levels $(x_{t+1}, x_{t+2}, ..., x_T)$ by $\delta = 60 \times 60dt$. We reiterate that the function I is a slightly concave function of x_{T+1}. I can be described by $I(x_{T+1}, e_{T+1}) = e_{T+1}(a x_{T+1} - b x_{T+1}^2)$, where a and b are constants, the latter being a very small constant. This gives us the derivative $e_{T+1}(-a\delta + 2b\delta x_{T+1})$, which is negative because $a > 2b x_{T+1}$ for all legal values of x_{T+1}. This mechanism, therefore, leads to reduced profit when increasing d_i, but by how much depends on the rest price.

Thirdly, the change in all subsequent reservoir levels also reduces the head for all future production. Less reservoir filling means that the height in the reservoir decreases and that $(w_{t+1}, w_{t+2}, ..., w_T)$ all decrease, which gives less revenue for future production. This is expressed by the term $\sum_{t=i+1}^{T} \left(\bar{p}_t \frac{\partial w_{t,s}}{\partial x_{t,s}} \frac{\partial x_{t,s}}{\partial d_i} \right)$ in Equation 4.4. This effect is typically quite small and also depends on future production and prices.

These three mechanisms summarize how the discharge affects the profit. When we add uncertainty in inflow by including $|Q_f|$ scenarios we also need to keep track of the reservoir levels for all $|Q_f|$ different inflows, because they give different heads, rest revenue and perhaps constraint violations. This means that the derivative of the objective function with regards to d_i goes through $|Q_f|$ different scenarios, and we take the average over the scenarios. This explains the sum over inflow scenarios s.

Our objective function has two types of non-continuous points where the derivative is undefined. Firstly, there is a discontinuity for $d_t \in \{0, 10, 22\}$. We handle this by masking our gradient wherever our discharge is equal to 0. This means that an action set to 0 will remain unchanged during this phase. If we step above the legal range during an update, we clip the action trajectory to stay within legal bounds of $[d_{\min}, d_{\max}]$. This also means that a nonzero action can not be set to zero during this phase, which is the main reason for relying on other methods in the earlier stages of HG. This restriction means that our method is vulnerable to local maxima.

The second case of discontinuous points is when our reservoir filling is equal to the threshold, $x_{t,s} = x_{\min}$ or $x_{t,s} = x_{\max}$. We handle this by making it illegal to break any of the constraints. We do this by scaling our update so that the change in reservoir level does not break the constraint. However, if our initial trajectory already breaks any of these constraints, then our update equation is undefined, and the model will not improve our solution. We should also keep in mind that with $|Q_f|$ inflow scenarios, we get $|Q_f|$ different reservoir levels for each time step. If any one of them breaks the constraint we get a discontinuity.

We chose to deal with this discontinuity by scaling the update in order to avoid breaking the reservoir constraints. To do this, we compute the change in reservoir level from an update. If our current update added to the minimum reservoir level breaks the constraint, we scale the positive gradients. We do the opposite for negative gradients, where we check if the reservoir update added to the maximum reservoir level will put us above the maximum threshold. This might seem conservative in the case of the maximum threshold, as the punishment for breaking this constraint is not great. The reason we are so careful about getting too close to this constraint is that the discontinuities are very dense, as we are likely close to breaking the constraint in another scenario or time step. Gradient ascent would not work well in this discontinuous landscape. This can theoretically lead to some optimal solutions being unreachable for gradient ascent. In the case of scaling the positive gradients, we have:

$$\text{Scaling factor} = \frac{(b - \min(x_{t,s})}{\Delta_{\text{reservoir}}},$$

where $\Delta_{\text{reservoir}}$ is the change in reservoir level from doing the original update, $\min(x_{t,s})$ is the minimum reservoir level across all time steps and scenarios, and

b is a buffer to make sure we have do not get too close to breaking the constraint. Using trial and error, we found that around $b = 1000m^3$ is a reasonable buffer size. This scaling factor would only be applied to the positive gradients. When scaling the negative gradients, we use this formula to compute the scaling factor:

$$\text{Scaling factor} = \frac{(x_{\text{max}} - b - \max(x_{t,s})}{\Delta_{\text{reservoir}}},$$

This approach has some problems, as updating actions later than the maximum or minimum reservoir point will not affect the reservoir point at t. This can lead to missed potential. We still used the method described here for its simplicity. Another method we use in order to more easily control the learning rate is to only update the actions with the maximum and most negative derivative when we are close to breaking a constraint. This creates slower updates.

Technically, the production curves used to map discharge to power production and to map reservoir level to reservoir height are also created from linear interpolation of some points. Each point is therefore discontinuous, but because these curves are always increasing but diminishing, we set the derivative as the slope of the previous line. This means that for around 12 values the magnitude of the derivative will be wrong, but because it is directionally correct, this is unlikely to cause any problems. Figure 4.2 shows an example of the derivative with regard to the profit in the bottom diagram. This is shown for different actions and prices. We set the inflow equal to the discharge to get an unchanged reservoir filling. This means that the only difference for the gradient is the prices and discharge level. The first 120 steps show how the gradient changes for different actions (d_t). We see that because the production curve is not entirely continuous, the derivatives are jagged. This can lead to a misleading derivative, however, the update will in this case be very small and likely not have drastic effects on the convergence. The last 150 steps show the derivatives for the same power production at different price levels. Because $\frac{\partial p_t w_t}{\partial w_t} = p_t$, this is a straight line.

In summary, the main weakness of the gradient phase is that there are many discontinuities which means that the global maxima might not be reachable from the starting trajectory. This method can not set a nonzero action to zero or the opposite. It can only modify nonzero actions. In cases where breaking the reservoir constraints are impossible, then gradient ascent will not work.

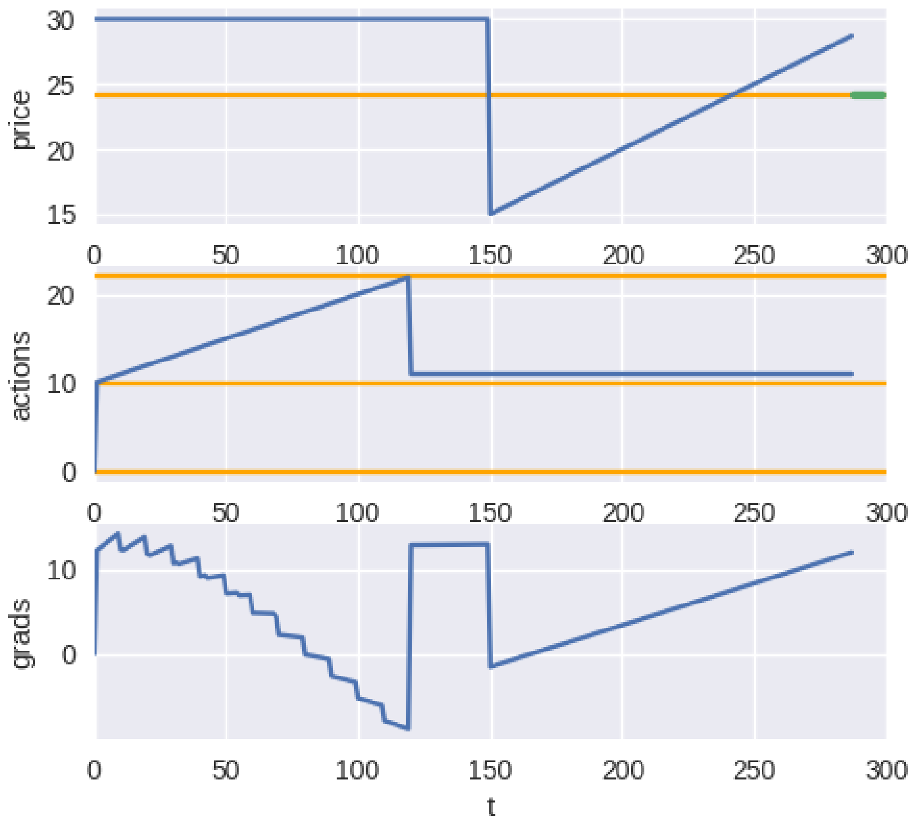

Figure 4.2: Illustration of the gradient for different prices and actions. The inflow is set to match discharges, so the reservoir filling is the same for all time steps.

Chapter 5

Experiments and Results

In this chapter, we start by showing our experimental plan to answer our third research question. In Section 5.2 the hyperparameters and some implementation details are discussed. Lastly, we present the results of the experiments in 5.3.

5.1 Experimental Plan

In our third research question, we are concerned with answering if the HG method can perform comparably to state-of-the-art methods. Our main experiment is to evaluate how well the different agents perform on 50 episodes of unseen data. We are interested in the average profit and also the distribution of the profits. HG is our main data analysis pipeline, which we described in Section 4.3. For our comparison with the State of the Art, we use a Soft Actor-Critic as described in Section 4.2, and also a stand-alone cross-entropy method. To set the profit numbers in context, we also use simple benchmark agents to understand if our agents have learned anything non-trivial. Comparing the results with benchmark agents can give an indication of whether our other agents have actually learned anything useful. We use four different benchmark agents that rely on simple rules that are easy to understand. We also use an optimization algorithm that has access to the true price and inflow data and also uses the best solutions from other methods as a starting point for optimization. The intention is to give an estimate of the optimal solution, however, the method can get stuck in local maxima. This is therefore not a perfect comparison. What follows is a description of each agent that will be compared.

Scipy solver This is a Scipy minimizer that uses the true price and inflow data. Because this method often ends with sub-optimal solutions, we use multiple starts, which include the solutions found by HG and CEM. The algorithm tested is L-BFGS-B, which is a quasi-Newtonian method. This method is intended to give an upper bound on the obtainable profit but is susceptible to local maxima, which means the solutions are not guaranteed to be optimal.

Avoid constraint agent This agent always produce at $17m^3/s$ if the reservoir filling is greater than 40%. When the reservoir is below this level, the agent always saves the water. This agent gives an indication of the profit level when only caring about the reservoir constraints and ignoring price signals.

Zero agent Zero always chooses to produce at 0 discharge. This means that the agent gets the rest value for the remaining water, but potentially loses water to overflow.

High price agent Always produces if the current price forecast is higher than rest price. When producing, it always chooses $17m^3/s$, which is the most efficient in terms of power per unit of water.

Random agent Chooses a random action uniformly from the action space.

The following more complex agents that we have discussed earlier in this thesis:

HG Heuristic search with Gradient Fine-tuning (HG) is the main system as described in Section 4.3. This method includes all four phases.

HG (w/o CEM) This is the same method as above, except that the phase using the cross-entropy method has been removed. We want to determine how much lower the running time and solution quality are compared to the above.

CEM This agent uses only the cross-entropy method but with more iterations and samples than the phase in HG. This agent should not be confused with HG, as it is an agent that only uses CEM in one phase. The hyperparameters for this method are discussed in Section 5.2.4 and an explanation of the method is given in Section 3.2.3.

SAC This is the model-free reinforcement learning algorithm described in Section 4.2.

As part of our goal, we are interested in the tradeoff between solution quality and running time. Consequently, we measure the average running time to complete an episode as a basis for discussing the tradeoff between running time and solution quality.

We also want to get an understanding of how the HG system works. We try to describe which situations the HG system performs better or worse in, what phases of HG contribute the most to the final solution and look at some of the inner workings of the methods. To do this, we visualize some episodes, show the gradient at the end of gradient ascent, and show the percentage improvement in the objective function of each phase. This can give a better understanding of the system and allow us to consider the limitations of the approach or suggest future changes to address weaknesses.

The ideal experiment would be a comparison to the stochastic programming approach that is currently used by Trønder Energi. However, this proved difficult. There are two reasons a comparison was not deemed feasible. Firstly, the program is time-consuming to set up and would be time-consuming for employees at Trønder Energi to explain. Secondly, the running time is long, and evaluating many episodes would need large amounts of computation. This is regrettable because we cannot determine precisely what the trade-off between solution quality and computation time is for our approach compared to traditional methods.

To at least test how close to optimal the results were, we tried to solve the version without uncertainty with linear programming using production systems. However, this also proved problematic. There were some slight changes in the modeling of the environment in the production system, which meant the proposed solutions broke constraints in our implementation of the environment. This meant that when running the proposed solutions in our system, the average profit was negative because of lower constraint violations. Changing either our model or the production model would be very laborious, so this step was skipped. We instead chose to compare our result with the Scipy solver which we discussed earlier in this section. This technique can be vulnerable to local maxima, and so is not entirely reliable.

5.2 Experimental Setup

In this section, we explain our experimental setup, which involves hyperparameters, neural network architectures, and implementation details. In Section 5.2.1 we give details on how we generated the environment data and the usage of noise to create uncertain but related forecasts. The interface and communication between the agents and the environment are explained in Section 5.2.2. Finally, we present the hyperparameters and training procedure for the Soft Actor-Critic in Section 5.2.3, the cross-entropy method in 5.2.4 and the HG system in 5.2.5.

5.2.1 Environment setup

Trønder Energi has used 30 price scenarios and 51 inflow scenarios in their planning system (Trønder Energi, 2022), and we use the same in this thesis. The inflow scenarios differ in length, which can create problems for simulation and machine learning, so since all scenarios have at least 12 days, all inflow and price scenarios are clipped to 12 days. In the normal case with an hourly resolution, this means a planning horizon of 288 time steps, each time step consisting of 30 possible prices and 51 possible inflow values. In the case of a 15-minute resolution, the horizon length will be 1152.

To generate synthetic data, we use historical data to learn a transition matrix. We use power price data from 2018 to September 2022, and some years of inflow data from the Mørre power station. This data is then binned into 50 discrete states. To learn the transition matrix P, we first count the number of transitions from state i to j, which are included as element $P_{i,j}$. The rows are then normalized to sum to 1, and $P_{i,j}$ now represent the probability of state j occurring next, starting from state i. We can then sample a random state, and then transition to new states with the transition matrix, creating an arbitrarily long data series. To prevent fixed values we also add some noise to get continuous data.

To create the data needed for an episode, we first generate an initial inflow and price series for the planning horizon. Based on the initial inflow and price, we add noise and shifts to get a new series. We use this technique to get both the uncertain scenarios for price and inflow and to generate the actual inflow and price. The actual price and inflow are the true values and are used to compute the profit, while the inflow and price scenarios are uncertain but available to agents for planning.

We shift the data a maximum of 4 hours, which means that the generated data at time t is set to the initial data at time $t + a$, where a is a uniformly random number between 0 and 4. After we have shifted the data, we add noise. We use a beta noise distribution with $\alpha = 3, \beta = 5$. Sampling from this distribution gives us random values between 0 and 1, which we multiply by our maximum noise, equal to $0.25 \times$ median, where the median is the median of the price or inflow series that we want to add noise to. We then give each noise value a random sign and add this noise to the initial value. The fact that we set the maximum noise as a fraction of the median means that data with a small median do not get high variability. This is typically the case for inflow which have many small values and a small median, but some periods where the inflow can increase by an order of magnitude. The noise added to the spike will then be quite small.

The rest price is sampled from the same distribution as the price, except that only the middle 50% of states are considered. This is to prevent extremely low or high rest prices, which is unrealistic. The initial filling of the reservoir is sampled uniformly from between 20% to 80% of maximum capacity. This is to reduce the amount of starting positions where avoiding constraints is impossible or the only viable strategy is releasing no discharge.

5.2.2 Agent-environment interface

When agents interact with the environment, they can only use features created from the price forecasts, inflow forecasts, initial filling, and rest price. To compute the environment's rewards and profit for an episode, we use the actual inflow, actual price, rest price and initial filling. Figure 5.1 shows the communication between the environment and the agent. In the case of model-based reinforcement learning, the raw data of the price forecasts, inflow forecasts, initial filling and rest price are given to the agent. The price forecasts have 30 scenarios and a time horizon of 288 steps, and the inflow forecasts have 51 scenarios and 288 times steps. New forecasts are generated after 24 hours to improve accuracy because forecasts get less accurate the more distant they are. When the agent receives this information, it can use the forecasts to simulate an environment to decide which actions to take or compute features from the forecasts. However, while the agent could replan every 24th hours, as the forecasts will be more accurate, we do not do this for our model-based agents. This is because when we replan, the forecasts will include data from after the endpoint of the episode. This can influence the agent into selecting sub-optimal actions to optimize for these prices after the end of the episode. Another approach could be to cut off forecasts after the endpoint, which would mean that each replanning of the agent would have a short planning horizon. Because this

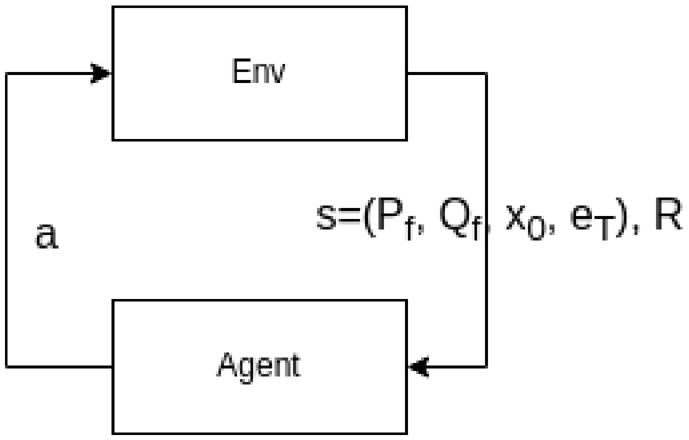

Figure 5.1: Interaction between the agent and the environment

would also be wrong compared to the production system and in addition use more time, we instead chose to only plan once at the start of an episode.

The current model-based reinforcement learning approach does not actually use rewards for anything, unlike the model-free reinforcement learning approach.

5.2.3 Soft Actor-Critic setup

For our model-free reinforcement learning approach, we used an implementation from Stable Baselines3 (Raffin et al., 2021).

Layer	Input	Output	Activation
Input	10	4	ReLU
Mu prediction	4	1	Linear
log std	4	1	Linear

Table 5.1: The neural network architecture of the actor

Layer name	Input	Output	Activation
Input	10	4	ReLU
Action-value	4	1	Linear

Table 5.2: The neural network architecture of the critic

The neural network of the actor is summarized in table 5.1. The actor starts with an input layer that maps the 10 features to 4 neurons. The values in these 4 neurons are used by 2 separate layers, which are used to predict the mean and standard deviation of the probability distribution used to sample actions. For the standard deviation, we predict the logarithm of the standard deviation, as an advantage of this is that after transforming it we can guarantee that the value is positive.

We use two critic models which sample independently from the replay buffer to create a more robust estimate of the state-value. Similar to the actor network, the critic network starts with a linear layer with ReLU and then takes a linear combination of this as the action-value estimate.

In addition to the two critics described above, there is also a separate critic target network used as a target for training the critic. It is identical in structure to the critics and plays a crucial role in stabilizing the learning process. The weights of the critic target network are not updated during the normal training procedure. Instead, they are periodically updated by copying the weights from one of the critic networks. The purpose of using a separate target network is to provide a more stable and consistent target for the critic during training.

Initial experiments with deeper networks and less feature engineering struggled to converge, as the model trained for a long time without learning to avoid breaking the constraints. This is likely because the state space is so large that adequately exploring it takes a very long time. To avoid this, we used a shallower network architecture and more feature engineering of the state representation.

The agent was trained for approximately 1.5 million episodes over two days. The hardware used to train the model is listed in the last table in Appendix A. During training, we used a replay buffer that can hold 10^6 pairs of inputs and rewards. In every tenth episode, the agent samples 3000 elements to train the neural networks. We used the learning rate 10^{-3}.

After the agent was trained, we tested the model with deterministic mode, meaning that the policy predicted a mean and used this as the action, rather than using the mean as a parameter for a probability distribution.

5.2.4 Cross-entropy method parameters

While the cross-entropy method is a phase in the HG system, we also tested an agent using just the cross-entropy method to plan. It starts by trying four different trajectories in order to get a warm start, before running the cross-entropy method as described in Section 3.2.3.

Table 5.3 shows the parameters used. For each iteration, 3000 trajectories are sampled. The 30 best-performing trajectories are then selected as the elite set and used in the update of the sampling mean and sampling dispersion in the equations 3.3 and 3.4. The learning rate is 0.1 and determines how fast the sampling parameters are updated. This is repeated for 60 episodes and the best trajectory encountered is returned. These parameters are based on grid search.

Because of the high running time and the problems due to a random rest price, which we discussed in 4.1, we sampled one trajectory for the whole episode of 12 days. This means that at the first step of the episode, the agent created a plan for the next 12 days and did not re-plan before the next episode. Originally, we planned to only submit the first 24 hours of the plan, then replan the next day. However, some initial experiments showed that the penalty related to the wrong rest price

Parameter	Value
Population Size	3000
Number of Iterations	60
Elite Ratio	0.01
Learning rate (α)	0.1

Table 5.3: Cross-entropy method parameters

created worse schedules than relying on less accurate initial forecasts.

We experimented with different constants to scale the exploration bonus and found that a relatively small constant of 0.1 was good. Too much exploration and the performance plummeted. Lastly, the model bonus is given by predicting the probability of choosing that action with the same models used during the rollout phase. We scale the number by 0.2, which means the model bonus is in $[0, 0.2]$

5.2.5 HG setup

Table 5.4 shows the parameters used in the different phases. For the tree policy in Monte Carlo tree search, which we discussed in Section 4.3.4, the exploration bonus constant is set to 0.1. The model bonus is 0.2 which means it will take a value in $[0, 0.2]$. The number of iterations is quite low due to time constraints.

For the CEM phase, we use fewer iterations and samples than in the standalone CEM discussed in Section 5.2.4. This is to reduce running time. Because the initial solution is expected to be quite close to optimum, we also use a higher learning rate to make the sampling distribution change its center and dispersion more quickly. This increases the risk of local maxima.

For the last phase, gradient ascent, we use a maximum of 1500 iterations. Based on observations, the agent typically gets an oscillating profit or very small changes in the profit at this point. The learning rate is set to 0.012, which is based on a grid search. We also tried a decaying learning rate, but this did not improve the performance of the decay values we tested.

Lastly, we start from 4 different trajectories. This is because the computer we ran our experiments on has 4 cores. A computer with more cores should increase the number of starts so that finding the global maximum is more likely.

To train the classifiers used in Monte Carlo tree search, we used 19248 samples of features mapping to a single action. Earlier samples might be worse because the Monte Carlo tree search was untrained at that point, as it was trained on actions from the seeding phase. Table 5.5 shows the parameters of the LightGBM classifier. The other classifiers do not rely on any parameters. The gradient booster is shallow with few estimators to prevent over-fitting and slow inference times.

Phase	Parameter	Value
MCTS	Exploration bonus	0.1
	Model bonus	0.2
	Iterations	100
	Early stopping chance	30%
CEM	Population size	2000
	Iterations	40
	Elite ratio	0.01
	Learning rate (α)	0.2
Gradient ascent	Iterations	1500
	α	0.012
	N starts	4

Table 5.4: The parameters for all phases of the HG agent.

Parameter name	max_depth	n_estimators	learning_rate
Value	2	300	0.5

Table 5.5: Parameters of LightGBM classifier.

Objective function implementation

The objective function was implemented using only fast, vectorized numpy operations. There are also changes to how we deal with overflow in order to speed up the code. This has to do with specifics of the numpy implementation and a wish to avoid loops. In the true environment, water above the reservoir threshold is discarded without punishment. However, to use a fast vectorized operation we instead add the changed water level for all time steps and then clip overflowed time steps to the maximum value. Because the cumulative sum operation means water above the threshold increases water at later time steps, we give a monetary punishment to ensure that the model tries to avoid overflow. This technique makes the evaluation function diverge from the evaluation of the environment, but the idea is that the profit in these cases is suboptimal either way and so the agent should quickly get negative feedback and move on to better trajectories.

5.3 Experimental Results

In this section, we will show various experimental results. The main result is the average profit of various agents across 50 episodes, which we explore in Section 5.3.1. This gives an overview of how well the different agents performed. We also show the distribution of profits across the episodes. Next, in Section 5.3.2 we analyze the total running time of the agents and discuss the trade-off between running time and solution quality. For the last two sections, we take a more detailed look at how the HG system performed. Section 5.3.3 shows how much each phase of the HG system improved the objective function, in addition to the running time of each phase. Lastly, in Section 5.3.4 we visualize some episodes and discuss which conditions HG performs well in, and what its weaknesses are.

5.3.1 Comparison of average episode profits

We now compare how our methods perform in terms of profit over 50 episodes, each episode lasting for 12 days or 288 time steps. Table 5.6 shows the average profits across 50 episodes for different agents. This is our estimation of solution quality.

The Scipy solver has an average profit of 145,127 EUR. We reiterate that this method has access to the true price and inflow data, and uses solutions from HG and CEM as a starting point. This means that the solutions of the Scipy solver will always be at least as good. Among the methods that use uncertain data, HG performs best with an average profit of 143,947 EUR, which is 1180 EUR or 0.81% less than our Scipy solver. Our version of HG without CEM gets an almost identical profit, with a difference of less than 2 EUR. Our third best agent is the stand-alone cross-entropy method which got an average profit of 141,150 EUR. This is around 2% worse than HG, equivalent to around 2800 EUR less in profit. This is a substantial difference from HG, but compared to the other benchmark methods discussed in this section, this is reasonable.

Surprisingly, the Soft Actor-Critic was outperformed by the constraint agent. The former got 116,502 EUR compared to 120,571 EUR, which is a large difference. The fact that the Soft Actor-Critic is outperformed by an agent based on a simple, hard-coded rule means that the Soft Actor-Critic has not learned anything non-trivial. One reason might be that the avoid-constraint agent always produces at the most efficient discharge level. Another might have to do with the fact that Soft Actor-Critic has an entropy term in its policy. This means that choosing a non-zero production can be very punishing, and the fact that the agent will explore might make every state seem more at risk of violating the lower boundary of the reservoir. Some inspection of the policy network shows that the policy is very sensitive to a low reservoir filling, and chooses to not use the water even at quite high reservoir levels. This might be fixed with more training. Either way, the Soft Actor-Critic did not perform well.

The zero-agent, which always selects zero discharge, gives an indication of the profit one could acquire by not producing any water. This means that the only source of revenue is the rest value, given by $I(x_T, e_T)$. This model will often lose water due to overflow but is safe from breaking the lower constraint. Finally, the random agent and the high-price agent use too much water and often break the lower constraint. The result is a negative average profit. This shows that avoiding the lower constraint violation has a large influence on the optimal choice of actions.

We will now look at the distribution of profits over episodes. Because of differences in initial reservoir filling and prices and rest price, some episodes will have much higher profits than other episodes, regardless of strategy. We can see the distribution of the profits in Figure 5.2. The Scipy solver achieves a slightly higher profit than HG

Agent	Average episode profit
Scipy solver	145,125.2
HG	143,947.1
HG (w/o cem)	143,945.4
CEM	141,150.2
Avoid constraint agent	120,570.6
SAC	116,502.4
Zero agent	81,978.6
High price agent	-5,167,617.3
Random agent	-41,549,111.9

Table 5.6: Average profit across 50 episodes for all agents, measured in EUR.

and HG without CEM, both of which are generally outperforming CEM. However, we see that the maximum value of CEM is higher than that of either of the HG versions. This happened in episode 13 which has the highest profit for all our top-performing agents. In this episode, the inflow is very large, and avoiding overflow appears impossible in the uncertain case. The Soft Actor-Critic chose the same strategy as both HG versions, with almost full production on each time step.

Figure 5.3 shows the relative profit of HG and HG without CEM compared to the Scipy solver. The profit of each agent is subtracted from the profit of the Scipy solver. The diagram shows that the solutions of the versions of HG are not always identical, as HG with CEM performs slightly better in the median episode and the first and third quartile than the version without CEM. While we remember that the average episode profit only had a difference of 2 EUR, we can see from the box plot that the median of HG is higher relative to the version without CEM. However, HG has an additional outlier that performs much worse compared to the Scipy solver. It should also be noted that during the training phase, HG with CEM got around 200 EUR more profit on average than the version without CEM. This indicates that the version with CEM often is slightly better. Still, our main result is that the difference in performance between the two methods is small. The fact that the two methods sometimes perform differently in the same episode also shows that the methods do not always find the global optimum. In 17 out of 50 episodes, HG outperforms HG without CEM by more than 50 euros, and the opposite is true in 5 episodes, where HG without CEM outperforms HG.

We see the same diagram but with CEM included in Figure 5.4. This diagram shows that CEM gets many outlier episodes where profits lag drastically behind the two HG versions. HG with and without CEM has a much tighter distribution, though in some cases still gets 10,000 EUR less than the solution of the Scipy solver. If we compare SAC to the constraint agent, we see that SAC has a greater dispersion in profits.

We should note that while it might appear that HG and HG without CEM have almost the same solutions, there is some variance in episode profit. One outlier occurred where HG got a much worse solution than HG without CEM. This also illustrates that these methods can be brittle and end up in local optima.

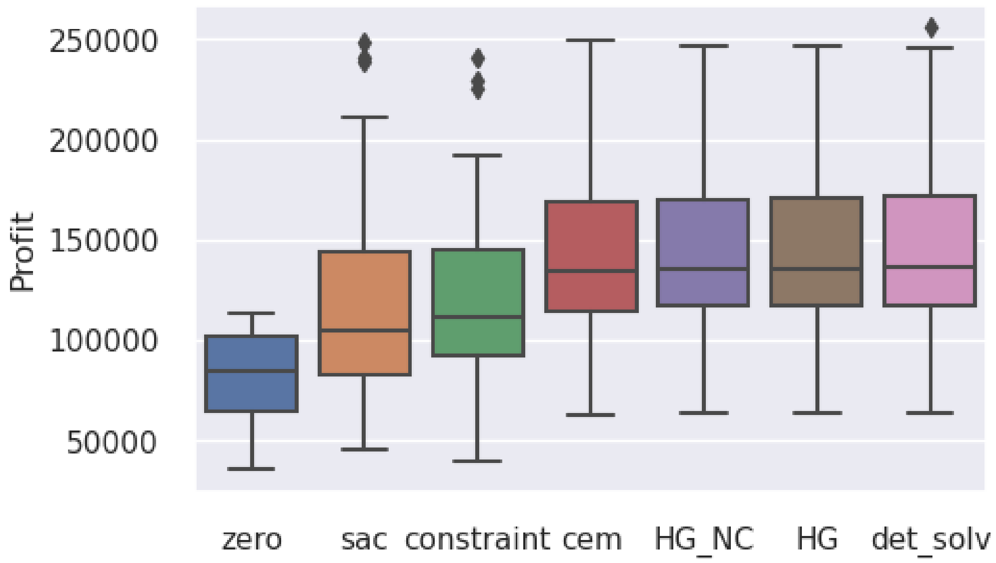

Figure 5.2: Profit distribution of different agents. The random agent and the high-price agent who broke constraints are omitted. HG(NC) refers to the version of HG without CEM. det_solv is the Scipy solver

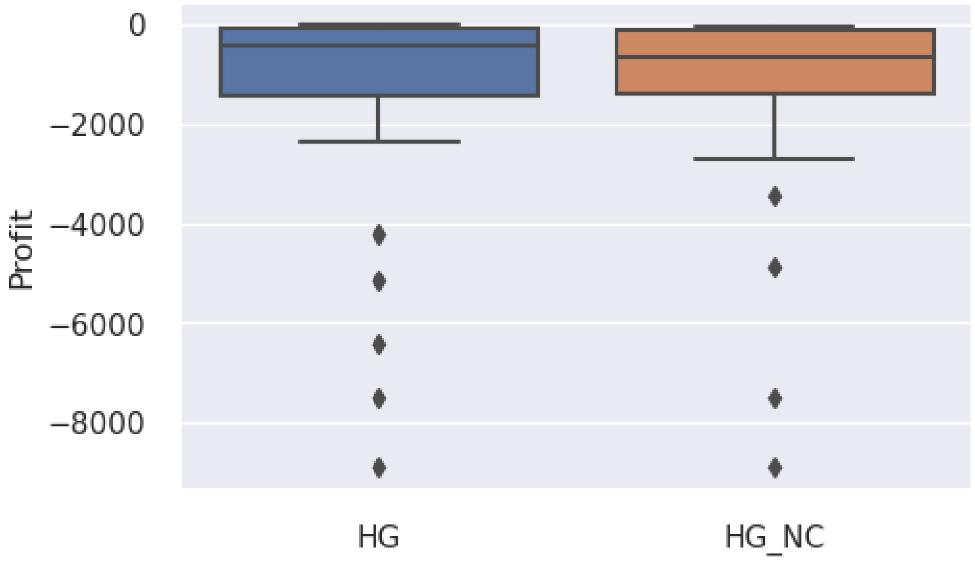

Figure 5.3: Profit relative to the Scipy solver. -10000 means that the agent got a profit that was 10000 EUR lower than that of the Scipy Scipy solver for a particular episode. The random agent and the high-price agent who broke constraints are omitted. HG_NC refers to the version of HG without CEM.

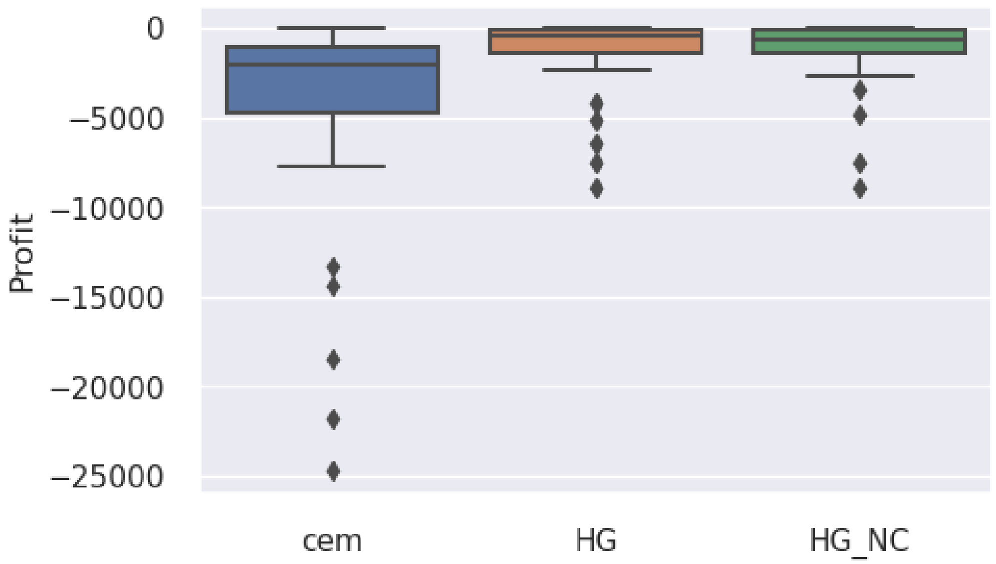

Figure 5.4: Profit relative to the Scipy solver. -10000 means that the agent got a profit that was 10000 EUR lower than that of the Scipy solver for a particular episode.

5.3.2 Total running time

We will now discuss the running time of the various agents. Table 5.7 shows the total running time for our main agents. We observe that the cross-entropy method is the slowest method at almost 2 minutes. While this can be reduced by using fewer iterations and samples, the solution quality is still worse than HG. Consequently, CEM might struggle to scale to a 15-minute frequency. The reason it is slower than HG, which also makes use of the cross-entropy method is that CEM does more iterations and more samples in each iteration, which gives us a higher running time.

HG uses 1 minute and 15 seconds, where most of the computation comes from CEM. We see that the version of HG without CEM only uses 28 seconds. This means that HG, especially without CEM, could likely scale to a 15-minute frequency with four times as long a planning horizon.

The Soft Actor-Critic and all the benchmark agents have an inference time below one second.

This benchmark was completed with a quite slow CPU with 4 cores and 2.8 GHz, as shown in Appendix A. A lower running time might be achieved with better hardware.

Agent	CEM	HG	HG w/o CEM	SAC
Running time	1min 55s	1min 15s	28s	<1s

Table 5.7: Total running time for the cross-entropy method and the HG system with and without the CEM-phase.

5.3.3 Detailed results from the HG system

Our attention will now turn to examining the performance of each phase in the HG system. Table 5.8 shows the percentage improvement for each stage in the HG system. Each improvement is relative to the best trajectory found in a previous phase. The improvements are based on profits from the objective function and not the profit with the true prices and inflows. Consequently, this is not an ablation study and does not give a perfect indication of the performance of the system without that phase included. For instance, a method that shows a certain improvement might have made improvements that would otherwise have been found in a later phase. In that case, excluding the method would give a 0% improvement, but will give a positive improvement in our table. The results should therefore be interpreted as a metric for how much each phase improves its initial solution and not its contribution to the final profit. When interpreting the improvement results, we should also keep in mind that the profit is generally very large for most production plans as long as the lower constraint is not broken. This means that the percentage improvements look quite modest, even though they can make a large difference in profits if we consider longer time horizons and that Trønder Energi optimizes many hydropower plants.

Monte Carlo tree search gives the least improvement, with 0.12% on average, which is equivalent to 192 EUR. Only in 7 out of 50 episodes did Monte Carlo tree search manage to improve the objective function. When we inspected the trajectories of the Monte Carlo tree search we often see that they are very close but slightly worse than the best trajectory from the seeding phase. However, in rare cases, Monte Carlo tree search can find a substantially better trajectory. The maximum improvement was almost 5%. In addition, the Monte Carlo tree search also provides many alternative starting positions for gradient ascent, which is another benefit.

CEM gives the best improvement, with an average improvement of 0.8%, equivalent to 1000 EUR. CEM manages to find a better trajectory than the input from Monte Carlo tree search in 42 out of 50 episodes. The gradient phase also gives a noteworthy average improvement of 794 EUR. The improvement results for HG without CEM are shown in Table 5.9. Here we see that the gradient ascent phase gets a greater improvement than in HG with CEM. This indicates that the improvement from CEM sometimes did not get out of local maxima, but instead got closer to the local peak. Without CEM, the gradient phase sometimes finds the same solution. There is also a small difference in the Monte Carlo tree search improvement, which is due to randomness.

Table 5.10 shows the running time for different phases of the HG system. The running time is on average per episode. The seeding phase is very fast at under a second and gives a decent solution quality for a small amount of computation. Monte Carlo tree search is also quite fast, considering that some of its running time includes unnecessary things like saving training data, models, and tracking results. The cross-entropy method is the slowest of the methods with a 46-second running time. Gradient ascent uses almost 25 seconds. For completeness, we also show the equivalent Table 5.11 for HG without the cross-entropy method. Due to randomness, the results are slightly different for Monte Carlo tree search and gradient ascent.

Improvement	mcts	cem	grad
mean (%)	0.12%	0.80%	0.52%
25%	0.00%	0.15%	0.00%
50%	0.00%	0.43%	0.15%
75%	0.00%	0.94%	0.46%
max (%)	4.96%	4.72%	7.68%
mean (EUR)	191.7	1000.1	794.2
median (EUR)	0.0	634.3	222.6

Table 5.8: The % improvement for each of the phases for HG. The last two rows show the mean and median improvement in euros, not percentages. This version of the HG has all stages included.

Improvement	mcts	grad
mean (%)	0.2%	1.15%
25%	0.00%	0.11%
50%	0.00%	0.54%
75%	0.00%	1.10%
max (%)	8.52%	7.89%
mean (EUR)	395.5	1589.3
median (EUR)	0.0	670.5

Table 5.9: The % improvement for each of the phases for HG without CEM. The last two rows show the mean and median improvement in euros, not percentages.

We will now examine whether the classifiers used as part of the Monte Carlo tree search have reasonable performance. In Table 5.12 we show the objective value of each of the classifiers relative to the objective value of the seeding phase. That is, we show $f_{\text{classifier}}/f_{\text{seeding}}$. In the median case, naive Bayes gets 93.2% of the objective value of the seeding phase. Linear discriminant analysis and Lightgbm are better with 95.8% and 95.6%. We also see that in the maximum case, the models give a marginal improvement over the seeding phase. However, the typical case is that the classifiers do slightly worse than the seeding phase. Due to some cases of constraint violations, the mean ratio is negative, because a few cases of constraint violation give

Phase	**Running time (s)**
Total	74.9
Seeding	0.6
MCTS	3.6
CEM	45.9
Gradient Ascent	24.7

Table 5.10: Average running times in seconds for the HG. The result is for the 50 episodes in the test set.

Method	**Running time (s)**
Total	28.0
Seeding	0.6
MCTS	4.0
Gradient Ascent	23.3

Table 5.11: Average running times in seconds for HG (w/o CEM). The result is for the 50 episodes in the test set.

negative objective values with large magnitude. One could argue that the classifiers could do a better job avoiding the reservoir constraint violations by keeping track of the reservoir filling and using it as input. The question is whether or not the improvement would be enough to justify more running time for the same number of iterations. The models do appear to have found a reasonable mapping in terms of the current price relative to both future prices and the rest price. Another observation is that the classifiers get different results, which means that our approach of randomly choosing a model allows us to introduce some randomness that aids exploration. Overall, these results show that the classifiers have learned some useful mappings and likely contribute positively to the Monte Carlo tree search.

	nb	lda	lgbm
25%	0.837	0.864	-0.007
50%	0.932	0.958	0.956
75%	0.974	0.991	0.995
max	1.008	1.003	1.012
mean	-91.002	0.626	-99.018

Table 5.12: Visualizes the performance of the classifiers from the Monte Carlo tree search. The score is given by the objective function if that classifier had been used to decide all actions in a trajectory divided by the objective value of the seeding phase.

5.3.4 Visualizing some episodes

We will now take a more detailed look at some episodes, and how the HG agent deals with them. In general, the episodes can be classified into four types.

In the easiest case, breaking the reservoir filling constraint is not a concern, and the only concern is to produce the optimal amount whenever the revenue from producing out-weights the loss in rest value. Figure 5.5 visualizes the most important variables from episode 0. The figure shows the time steps on the x-axis, and the upper diagram shows the true price in blue, and the price forecasts are in red. The rest price is visualized as a straight, orange line. The second diagram shows the discharge decisions of the HG agent, and the orange lines indicate the legal action boundaries. Thirdly, we have the reservoir filling, normalized so that the maximum reservoir is equal to 1. The bottom diagram shows the true inflow in blue and the inflow forecasts in red.

In episode 0, we can see that the reservoir filling is far from the constraint boundaries, which makes this episode relatively easy. We can see that HG chooses to produce when the price is above the rest price. The discharge chosen is also greater when the price is greater, as this can compensate for reduced power efficiency. In this episode, HG with and without CEM arrive at the same solution, and the profit difference compared to the Scipy solver is quite small. This indicates that our model does reasonably well in these easy scenarios. In total, we have 29 episodes that are in this category.

Another example is episode 1, which is visualized in Figure 5.6. The model produces at the highest prices. At the peaks, the production is slightly larger because the high prices compensate for the lower efficiency at higher discharge. Compared to episode 0, the difference between the price and the rest price in this episode is relatively small so the optimal production is generally close to the most efficient level at $17m^3/s$. Figure 5.7 shows the gradient at the end of the optimization procedure. The derivatives are close to zero and oscillate between positive and negative. A lower learning rate could likely end the episode with a gradient with a lower magnitude.

The second type of episode is when many prices are higher than the rest price, or the initial filling is low. In these cases, avoiding breaking the lower constraints is of primary concern. The optimal strategy will use as much water as possible and likely be at the edge of breaking the lower constraint. Among the test episodes, there are 8 episodes in this category. One example is episode 19, which is visualized in Figure 5.9. The actions are all quite close to $17m^3/s$ in this episode, even at the time steps where the prices are highest. One reason might be that the reservoir's lower boundary is so close to being broken, that the allowed gradient updates are very small. This close to the reservoir boundary, only the maximum and minimum gradients are used in the update, and we also scale the gradient as described in Section 4.3.6. When we inspect the gradient at the end of this episode, we see that all actions have positive derivatives. Figure 5.8 shows the gradient at the end of episode 19. They are all positive, but increasing the discharge when the reservoir filling is so low is impossible. Therefore, the discharges are almost unchanged by the gradient ascent phase. Overall, the objective function only increased by 50 EUR in this episode.

Episode 26 is very similar and is visualized in Figure 5.10. At the end of this episode, the average nonzero derivative was 14.5. At the last steps of this episode, there is enough reservoir filling to increase the discharge, and the derivatives for these last time steps are positive. However, due to our naive update rule, this is not allowed, and we likely miss some profit by not increasing the discharge at the end.

Thirdly, we have the opposite situation where avoiding an overflow of the reservoir is of primary concern. In these episodes, most prices are quite low relative to the rest price, and the initial filling is high. If the inflow is also high, then an optimal strategy will often barely avoid overflow and get most of the profit from the rest price. 8 out of 50 episodes are in this category, and one example is episode 11, which is visualized in Figure 5.11. In this episode, HG needs to use nonzero discharge even when the price is much lower than the rest price. It does select 0 discharge at a few of the lowest prices. The reservoir in the worst-case inflow scenario ended at $1000m^3$ from the maximum threshold, which is the buffer we used in our scaling of the update. At the end of the episode, the average nonzero derivative is equal to -12. Another example episode is given in Figure 5.13, where episode 48 is visualized.

In the fourth case, the inflow is so high that avoiding an overflow of water is impossible. In these cases, the gradient optimizer does not work due to the number and density of discontinuities. Often, the chosen solution ends up being close to full production in all time steps. We investigate whether or not HG performs worse in these episodes compared to the 29 normal episodes. We find that in the normal episodes, the Scipy solver led to an 0.13% improvement in the median case, and in the episodes where overflow is impossible to avoid, the Scipy solver improved the solution by 0.97% in the median case. When the Scipy solver manages to find a much better solution, this indicates that HG is worse on overflow episodes. This is not only true for the median case, but also for the mean and first and third quartile of the improvements. This indicates that the solutions of HG are more often suboptimal when overflow is impossible to prevent.

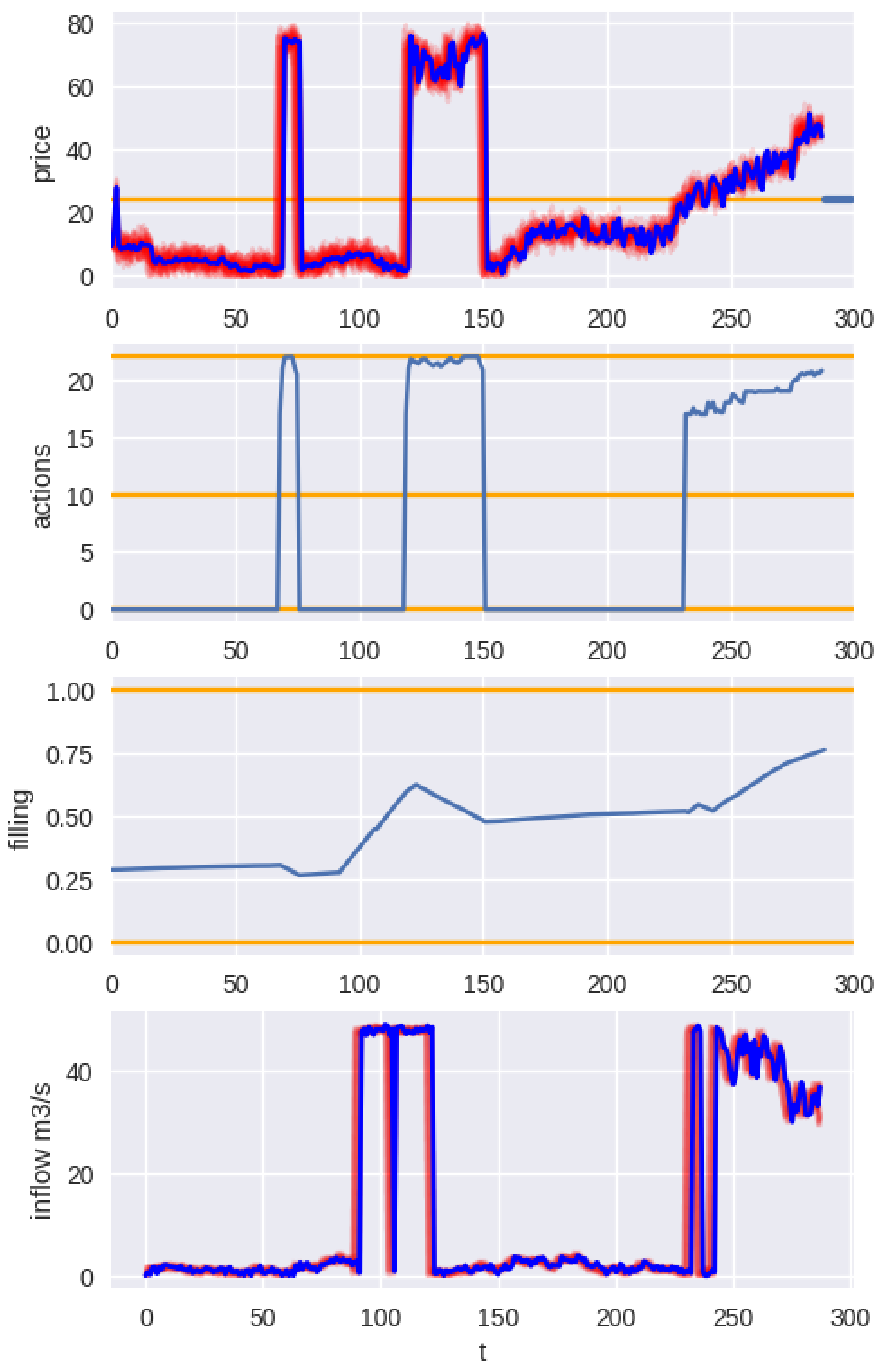

Figure 5.5: Visualization of the key variables from episode 0. The x-axis shows the time steps. The upper diagram shows the true price in blue and the price forecasts in red. The orange line is the rest price. The second diagram shows the actions, the lines are constraints. The third diagram shows the reservoir filling normalized so that the minimum and maximum are 0 and 1. Lastly, we have the true inflow in blue and the inflow forecasts in red.

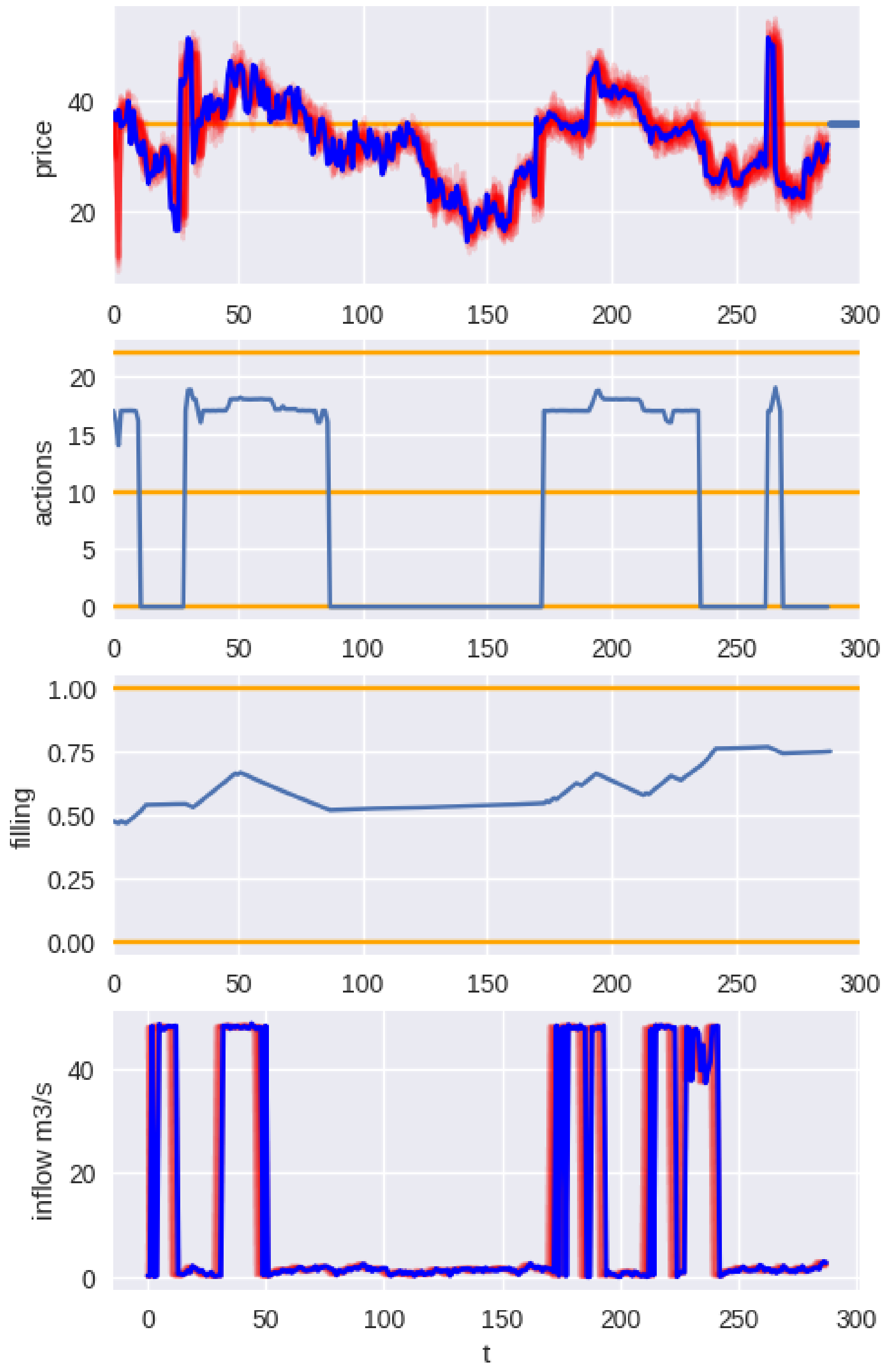

Figure 5.6: Shows the key variables from episode 1. The x-axis shows the time steps.

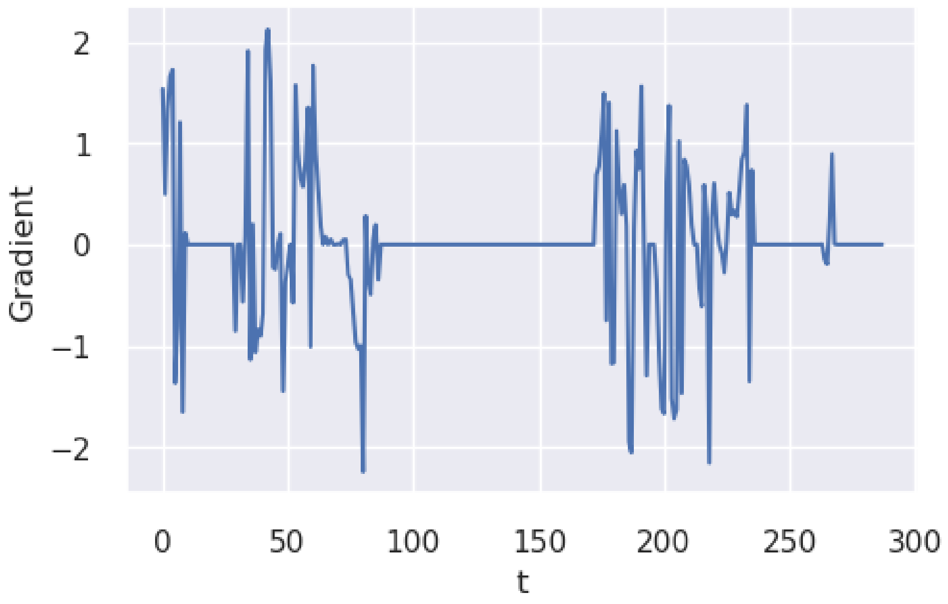

Figure 5.7: Shows the gradient at the end of episode 1.

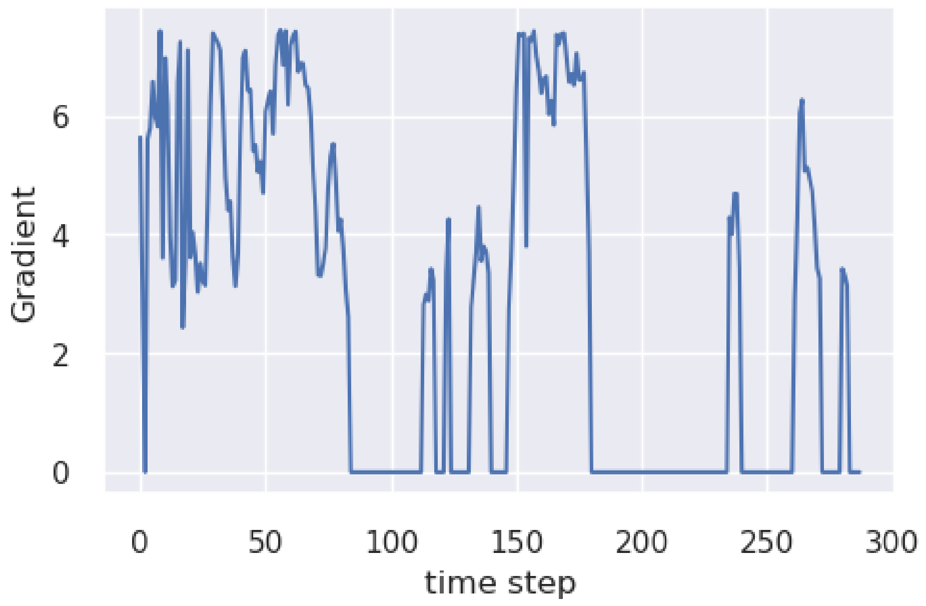

Figure 5.8: Shows the gradient at the end of episode 19.

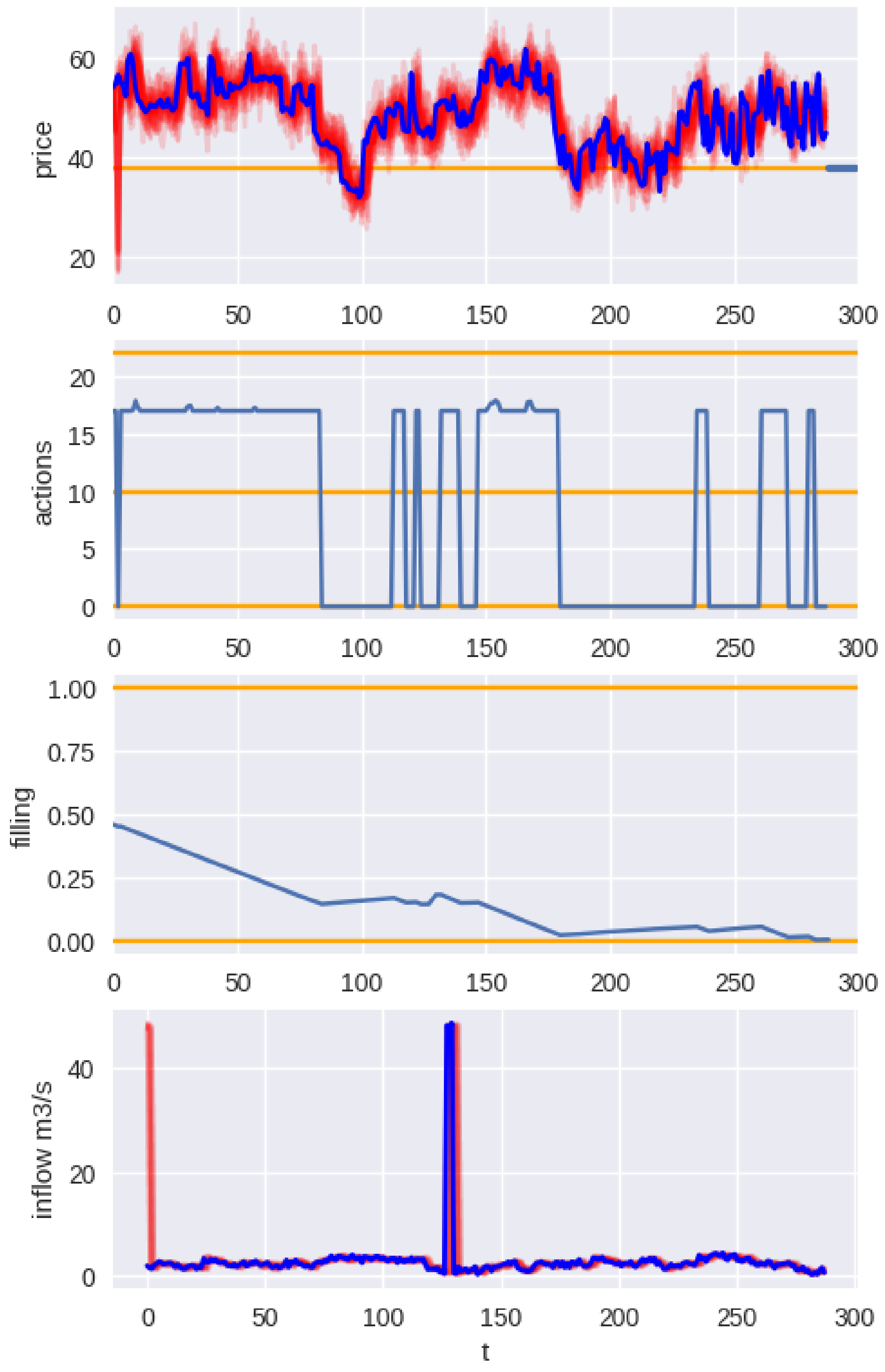

Figure 5.9: Visualization of episode 19, an example where the price to rest price ratio is high, and the optimal strategy is to almost empty the reservoir. The x-axis shows the time steps. The upper diagram shows the true price in blue and the price forecasts in red. The orange line is the rest price. The second diagram shows the actions, the lines are constraints. Third, we have the reservoir filling between 0 and 1. The bottom diagram shows the inflow in blue and inflow forecasts in red.

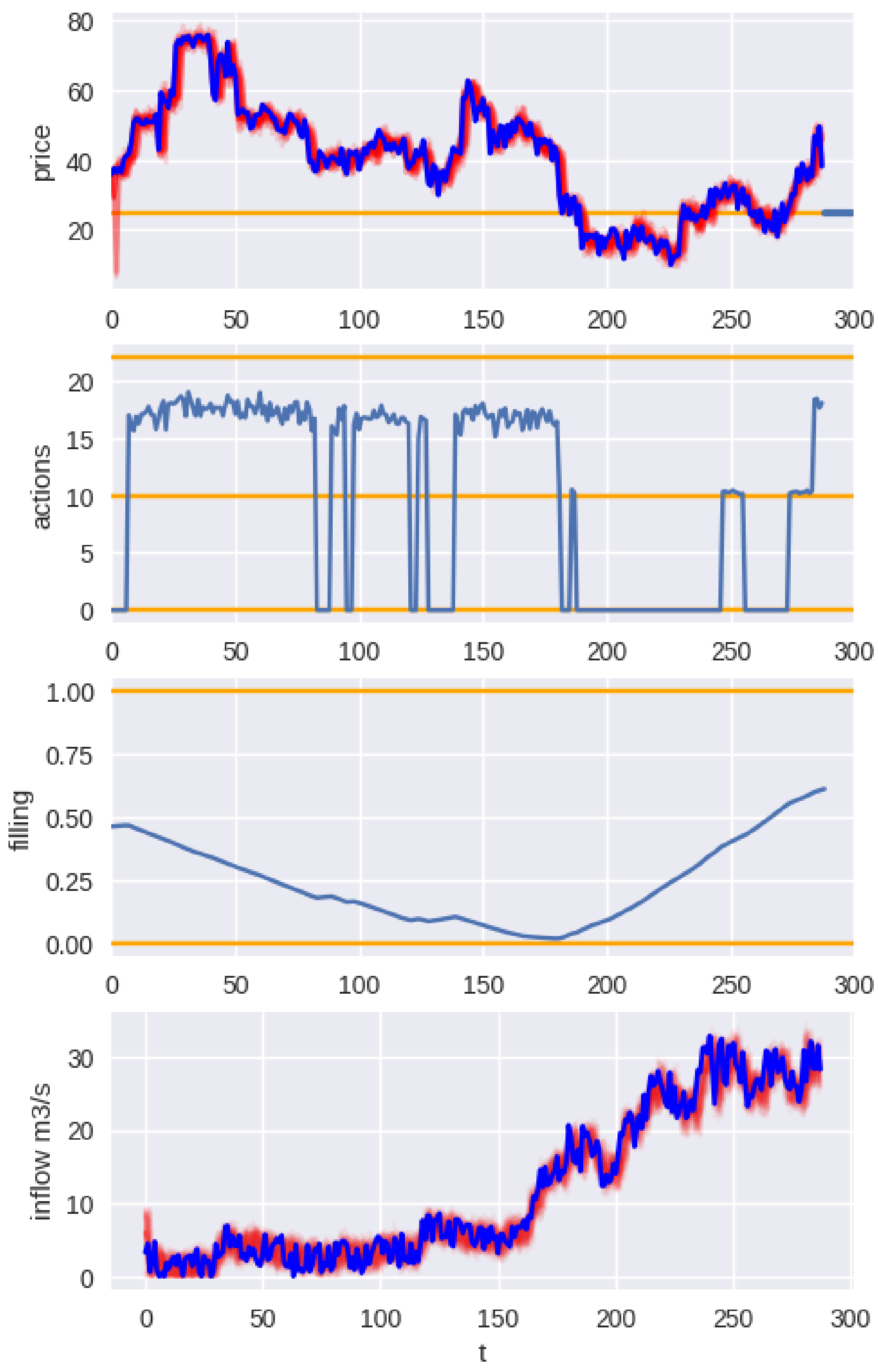

Figure 5.10: Visualization of episode 26, an example where the optimal strategy is to almost empty the reservoir. The x-axis shows the time steps. The upper diagram shows the true price in blue and the price forecasts in red. The orange line is the rest price. The second diagram shows the actions, the lines are constraints. Third we have the reservoir filling between 0 and 1. The bottom diagram shows the inflow in blue and inflow forecasts in red.

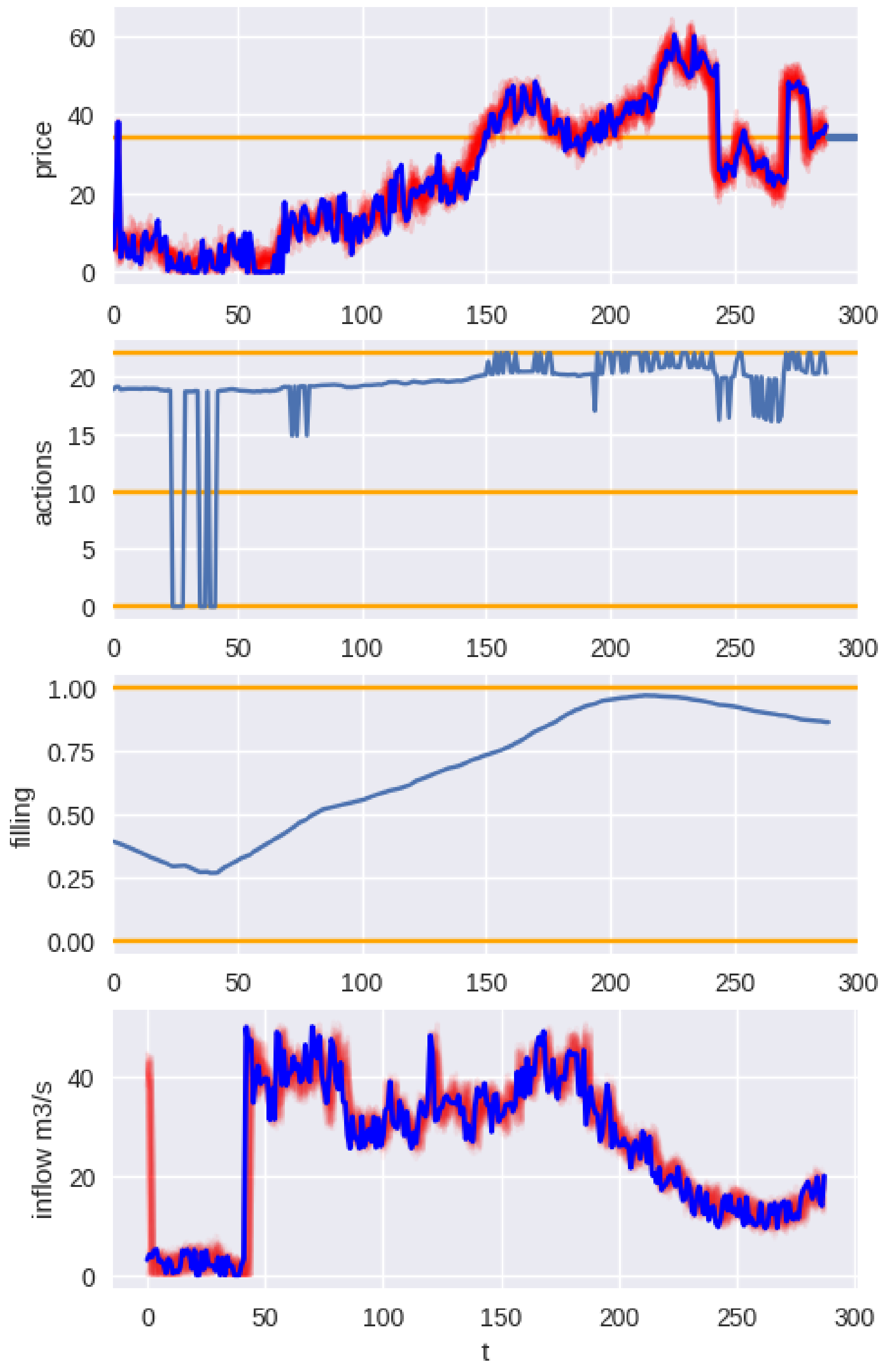

Figure 5.11: Visualization of episode 11, an example where the price to rest price ratio is relatively low, but the agent still uses water to avoid overflow. The x-axis shows the time steps. The upper diagram shows the true price in blue and the price forecasts in red. The orange line is the rest price. The second diagram shows the actions, the lines are constraints. Third we have the reservoir filling between 0 and 1. The bottom diagram shows the inflow in blue and inflow forecasts in red.

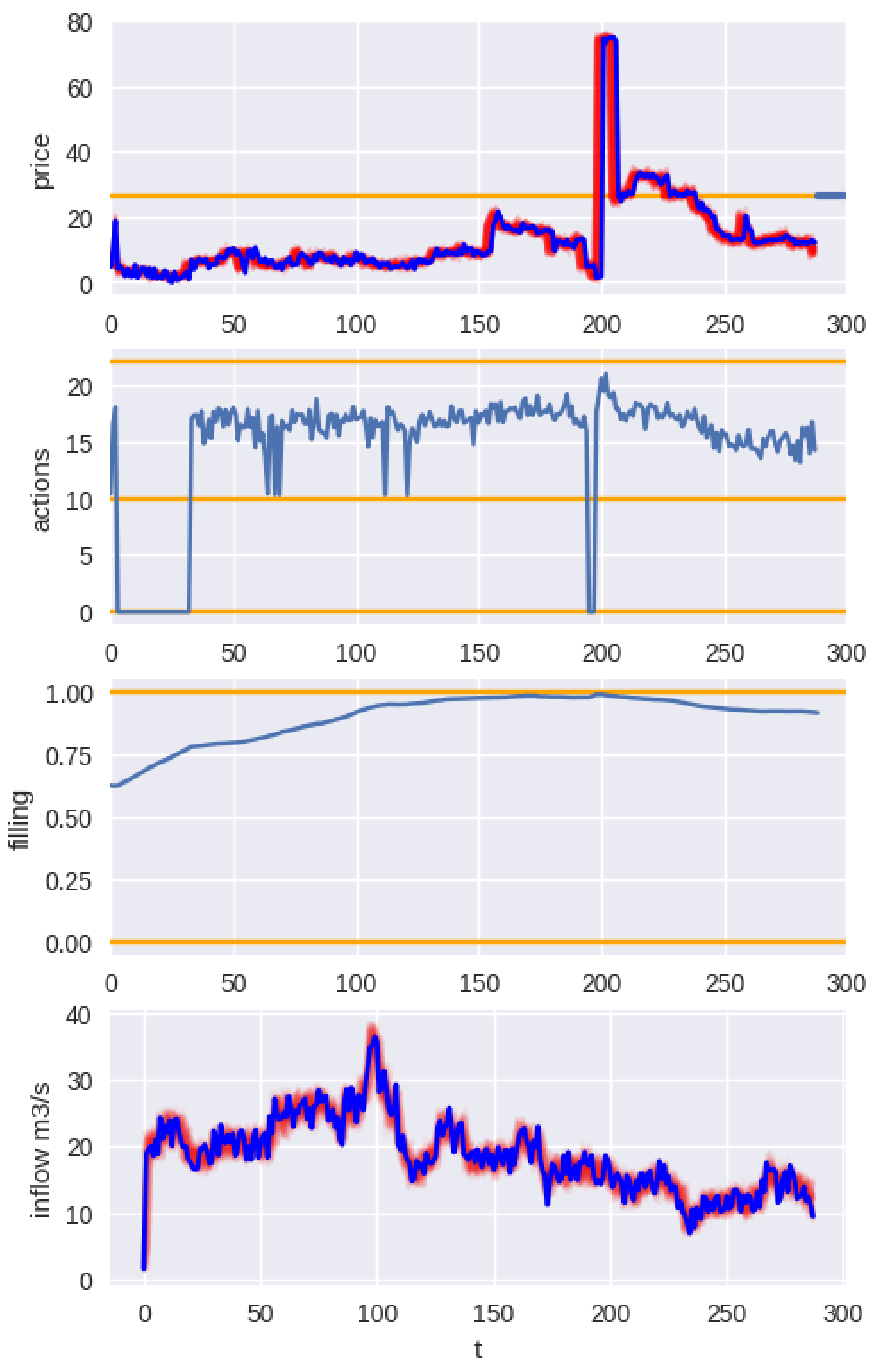

Figure 5.12: Shows the key variables from episode 43. The x-axis shows the time steps. The upper diagram shows the true price in blue and the price forecasts in red. The orange line is the rest price. The second diagram shows the actions, the lines are constraints. Third, we have the reservoir filling between 0 and 1. The bottom diagram shows the inflow in blue and inflow forecasts in red.

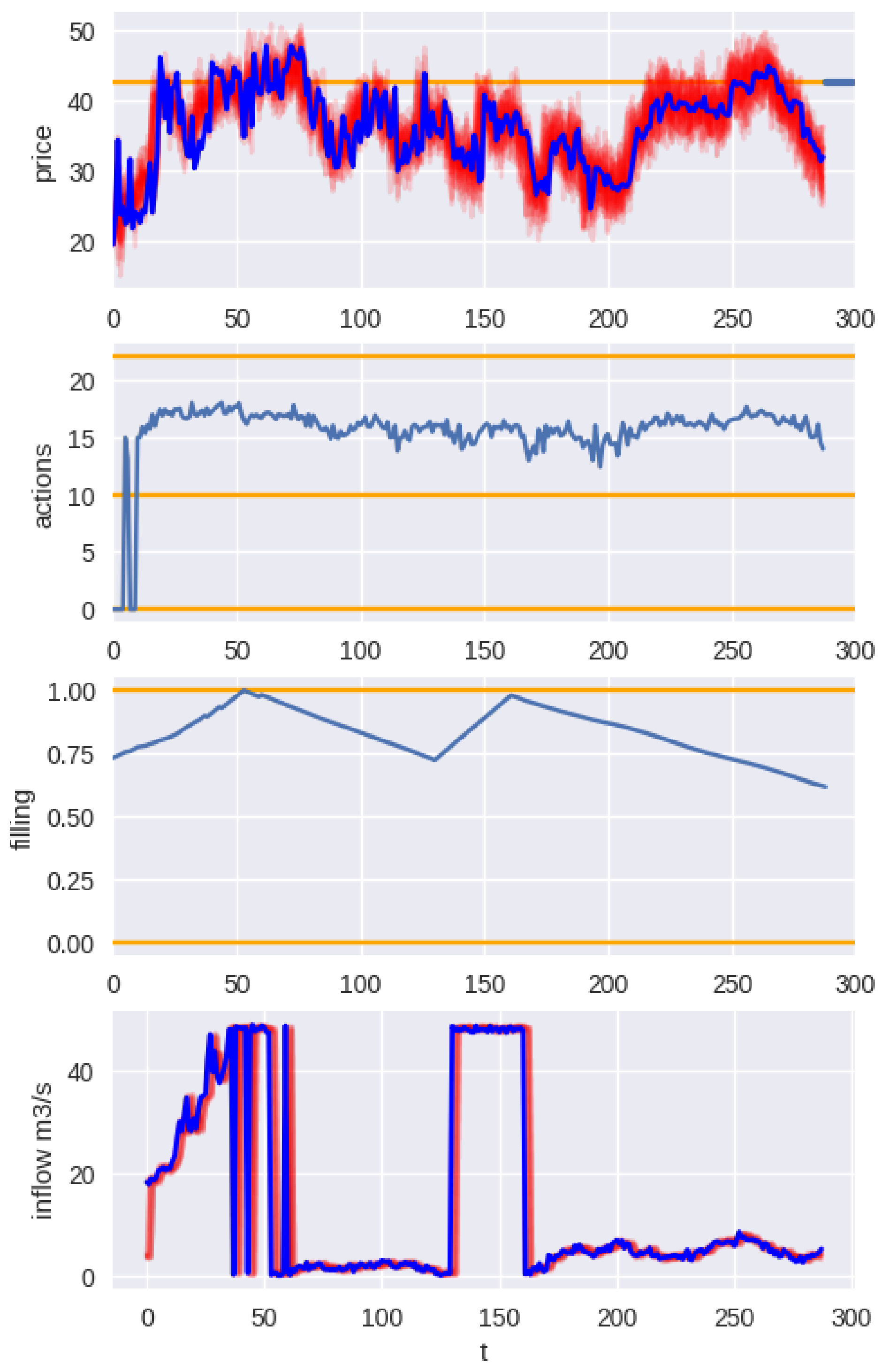

Figure 5.13: Visualization of the key variables from episode 48. The x-axis shows the time steps. The upper diagram shows the true price in blue and the price forecasts in red. The orange line is the rest price. The second diagram shows the actions, the lines are constraints. Third, we have the reservoir filling between 0 and 1. The bottom diagram shows the inflow in blue and inflow forecasts in red.

Chapter 6

Evaluation and Conclusion

In this final chapter, we interpret our results and draw some overall conclusions. In Section 6.1 we reiterate our research questions and attempt to answer them using background knowledge, surveyed literature, and experimental results. Subsequently, we focus on the limitations of both our results and the HG pipeline in Section 6.2. In Section 6.3, we briefly consider the effect on HG by transitioning to a 15-minute time resolution in the power market. Finally, we discuss some possible directions for future work in Section 6.4.

6.1 Evaluation

Our goal for this thesis was to find solutions to the short-term hydropower scheduling problem that balances computation time and solution quality. To reach this goal we reduced it to three research questions, which we defined in Section 1.2. We reiterate the research questions here:

1. What techniques have been used to solve continuous control problems with uncertainty?
2. Which techniques exhibit the best fit for addressing the hydropower scheduling problem?
3. Is it possible to create a data analysis pipeline that is competitive with State of the Art methods while subject to constraints on the running time?

To answer our first research question we surveyed some methods used to solve continuous control problems in Chapter 3. The field is large, so only a limited selection of methods was mentioned in this thesis. Model-free reinforcement learning can address continuous control under uncertainty with very few assumptions or knowledge about the domain, but a disadvantage is that a large amount of interactions with the environment is required. The Soft Actor-Critic is a common choice for continuous control and was discussed in Section 3.2.2. The Soft Actor-Critic deals with the continuous action space by using a policy that predicts the parameters of a continuous probability distribution. Some other methods we discussed were supervised learning to map a state directly to an action, and the cross-entropy method which samples trajectories from a probability distribution and progressively updates the probability distributions towards more promising areas. Monte Carlo tree search can be used for continuous control if the actions are discretized. We also discussed a type of reinforcement learning that computed the derivative of the objective function with regard to the policy parameters.

The second research question asks which of these methods exhibits the best fit for our problem. This discussion is dispersed throughout Chapter 3, 4, 5. Model-free reinforcement learning is known to not be sample-efficient and requires large amounts of training. In our experiments, the model-free reinforcement learning agents did not perform well. Despite 3 days of training and laborious adaptation to the environment, the Soft Actor-Critic did not outperform a simple rule-based agent. With more training or another model-free reinforcement learning agent, the agent could potentially perform better. In our problem formulation, actions have consequences far into the future. For instance, if we produce now, we might be unable to exploit a much higher price in 200 steps. The delayed feedback of this information is challenging for model-free reinforcement learning. Model-free reinforcement learning is known to struggle with delayed feedback, and when the power market changes frequency to 15 minutes, we will get four times as many time steps in the planning horizon. That will make the problem even harder. In addition, the amount of information required to optimally choose one action makes it hard to learn a good, one-shot mapping from state to action. All things considered, the Soft Actor-Critic does not seem like a great fit for the hydropower problem.

We also reviewed the use of supervised learning for the hydropower problem in Section 3.2.1. The results surveyed do not use an identical problem formulation or use a comparable planning horizon, which makes them inconclusive in answering how they would perform on the problem in this thesis. Much of the same critique aimed at model-free reinforcement learning is true for supervised learning. Why use a one-shot decision when using less than a second for inference is not necessary? It would require a large amount of training and data to learn the optimal discharge of the large states in our problem.

For model-based reinforcement learning, we discussed and tested one method used for planning. The cross-entropy method performed much better than the Soft Actor-Critic, and was close to HG in performance. However, it required the most amount of computation and might be unable to scale when applied to a planning horizon four times longer. However, considering that the method did not use any domain knowledge and could likely be further improved, this method fits the hydropower industry surprisingly well.

We also discussed some applications of heuristic search and Monte Carlo tree search in Section 3.2.5. We tested Monte Carlo tree search as part of the HG system. In a few cases, it did find better trajectories and also gives some robustness. The running time is also quite low at only 4 seconds. Arguably, the success of this method was mixed. Further experimentation with extensions and hyperparameters would likely be needed to make Monte Carlo tree search work well. Finally, in Section

3.2.4 we discussed an example of using a differentiable simulator. We have shown with experiments that gradient ascent is a good fit for the hydropower problem. If gradient ascent is given a good starting point, then the method seems to find a local maximum.

Our third research question was whether or not we can create a data analysis pipeline that is competitive with state-of-the-art methods. We built the HG pipeline, which outperformed two state-of-the-art methods, the Soft Actor-Critic and the cross-entropy method in terms of solution quality. While we can change the running time of both HG and the cross-entropy method by modifying parameters, HG in our experiment dominates the cross-entropy method in both running time and solution quality. This shows that HG can outperform some state-of-the-art methods, though objections regarding hyperparameter tuning and inadequate training of the Soft Actor-Critic could be raised. There is also no evidence that HG outperforms all state-of-the-art methods.

If we also include traditional methods as part of State of the Art, then our answer to this question would be inconclusive. We discussed linear programming and stochastic programming in Section 2.2.1. We did not manage to get a comparison of our method to that of the stochastic programming approach used in the production system. However, we can make some rough estimates. Optimization of the Mørre powerplant uses around 2-5 minutes, according to a private discussion with an employee at Trønder Energi (Trønder Energi, 2022). This means that the running time for HG is lower. In terms of solution quality, we do not have comparable data and can only speculate. A result that the stochastic programming got 95%-97% of the deterministic profit for a certain testing period has been mentioned (Trønder Energi, 2022). This result is not comparable to our simulation for many reasons. One reason is the differences in price and inflow patterns and more importantly that our forecasts might be more accurate than the real world. We know that HG sometimes gets stuck in local maxima in our episodes. HG has also been tested and adapted on a flawed simulation, unlike the production system which has been adapted to the real power market. This raises the question of how much the solution quality differs between HG and the production system. This is left as future work.

Finally, our overall goal was to find solutions to the short-term hydropower problem that balances computation time and solution quality. In pursuit of this goal, we designed and tested the HG system. With a running time under 2 minutes, HG adhered to our approximate threshold on running time. While we have no perfect evaluation of the solution quality, the system performs better than some simple benchmark agents and two state-of-the-art methods. We also compared HG to an imperfect solver used on the true data, and HG got on average a 0.8% lower profit. Overall, we consider the HG system to accomplish our goal of finding a method that balances computation time and solution quality.

6.2 Limitations

In this section, we will discuss some limitations caused by inadequate comparisons in our tests and some problematic modeling choices in our experiments.

The most critical limitation of the work in this thesis is that our results are not compared to the optimal solution or traditional methods. Consequently, we do not know how good the solution quality of our agents is. Comparisons to simple benchmark agents and more complex methods like the Soft Actor-Critic or the cross-entropy method indicate that HG finds decent solutions, but exactly how good compared to the optimal solution is unknown.

There are also limitations related to our modeling choices and our environment assumptions. This could lead to a different performance in a real word deployment of the system. We discussed many of the assumptions of the environment in Section 4.1. We will now discuss what consequences this has for our interpretation of the results. Assumptions 2, 3, and 4, are that prices and inflow are independent, that the first-order Markov assumption holds and that inflow and prices are stationary processes. We argued that these assumptions are likely wrong. The consequence of this is that the kind of prices and inflows seen in our test might be very different from the real world. It is possible that our agents would perform worse or better on real-world data. For instance, in real-world scenarios, it might be more or less common to encounter situations where overflow is impossible to avoid. If it occurs more often, the HG system would perform worse. Another concern might be seasonality in prices, which could create patterns in real-world data that our methods happen to do better or worse in. In general, violations of these assumptions lead to changes in the data distribution which can reduce the performance of all our methods. When we want to assess the solution quality of different methods we are interested in the data distributions typically encountered in the day-ahead market and Norwegian hydropower plants, and therefore the experiments in this thesis can be misleading.

Another potential source of error is the relation between the forecasts and the true prices and inflows. Because the noise magnitude was based on the trajectory-wide median, this noise is small when the median is small. As we can see in the episode visualizations (5.3.4), the forecasts of the inflow are often quite close to the true inflow. If the real-world inflow forecasts have a much larger dispersion, then HG might perform worse. Our choice to try to avoid breaking the reservoir constraints for all scenarios might be impossible more often if the dispersion is greater. Furthermore, our agent will more often be close to violating the constraints for a scenario, and we showed that HG performs worse in these cases.

We do not use multistage optimization, which we explained in Section 2.1.4 and 2.2.1. This means that a perfect optimization of the objective function in the HG system might be a lower bound on the optimal decision. How much this impacts solution quality is not quantified. In general, there are many modeling choices for our environment and objective function that could skew the experimental results.

We will also highlight some limitations of the HG system. Our experiments show that HG performs worse when the good solutions are very close to breaking the reservoir constraints. Because the gradient optimization does not work when overflow is impossible to prevent, HG performs poorly in these cases.

6.3 Scaling to 15-minute timesteps

When the power market transitions from hourly time steps to 15-minute time steps, a 12-day planning horizon would have four times as many time steps. Consequently, the solution space grows exponentially following the relationship x^4, where x is the solution space at an hourly resolution. The original solution space is given by a^T, where a is the number of discretized actions and T is the planning horizon, given by 288 in our case. Our new solution space becomes $a^{4T} = a^{1152}$.

To illustrate the implications of this exponential growth, let us consider some examples. We estimate that the HG system scales linearly and will use $4c$ times as long to run, where c is the additional running time from the evaluation function needing to evaluate longer trajectories. The running time of evaluating longer trajectories increases by less than four times, due to many constant overheads in the objective function. Consequently, c is a number below 4. Consider the seeding phase, where we need to test both four times as long and four times as many trajectories. Similarly, Monte Carlo tree search will need to choose four times as many actions for each iteration. Consequently, the running time can be estimated to use around four to sixteen times longer, depending on c. However, because the solution space has grown exponentially, we explore less of the solution space. Thus, our solution quality will likely get worse. Linear programming, on the other hand, faces an exponential increase in both the solution space and the running time in the worst case, though not in the average case.

6.4 Future Work

We will now suggest some directions for how this work could be continued in the future. The most important future work would be to address the limitation of inadequate comparisons of the HG system. A test of the HG system with real power prices and inflow data, and a comparison to the production system would give conclusive evidence about the solution quality. Additionally, testing the HG system with four times as long a planning horizon could show how the system would scale to a 15-minute frequency in the power market. These experiments could address our concerns with the modeling of the environment and indicate whether or not the approaches in this thesis are worth further development.

Additionally, all phases of the HG system could likely be improved. In the experiments conducted during this thesis, we observed some cases where the performance of our system was suboptimal. Experiments with real data might uncover other blind spots in our pipeline. We observed that the seeding phase is more often suboptimal when we are close to breaking constraints. Sometimes adding the next greedy action might break a constraint, but adding the action at a later point might be beneficial. Some rules that incorporate reservoir filling could perhaps improve the seeding phase in these cases. Keeping track of reservoir filling might also be worth the extra computation for Monte Carlo tree search, as this method sometimes broke the reservoir constraints. The reservoir filling could be included as a feature for the classifiers. The gradient ascent phase also has a similar problem. The scaling factor on the update in gradient ascent is naive. For instance, in some cases, the reservoir threshold might be at risk at time step t, preventing an increase in discharge before time t. This does not necessarily mean that increasing discharge after this point should be forbidden. A more sophisticated technique could likely improve the performance of gradient ascent when close to the reservoir boundaries. In cases where overflow is impossible to prevent, one possibility is to use a different method since gradient ascent does not work in this case. Another potential marginal improvement could be the usage of continuous production curves when applying gradient ascent. This would make the derivative less jagged, as we saw earlier in Figure 4.2, perhaps leading to better convergence. Better hardware with more CPU cores would allow us to use more than 4 starting points for gradient ascent, which might increase performance additionally. There are numerous possibilities for extending or changing the Monte Carlo tree search or the cross-entropy method. One option might be to entirely replace or remove them, as they did not always give substantial improvements.

One could also consider using the solution of the HG system as a starting point for stochastic programming. The concept of a warm start in linear programming solvers is to give a near-optimal starting point, which can substantially speed up the solver.

There are also some problems with similarities to the hydropower problem. In general, problems involving energy storage such as battery arbitrage or pumped hydropower share many characteristics, and adapting HG to these problems might work.

Lastly, we would also like to mention that attempting to model the problem as a multistage problem could be rewarding. All phases would need changes, and representing all the scenario-dependent action trajectories would require more memory and computation, as we mentioned in Section 2.2.1. Investigating whether the gradient optimization works well for the multistage problem and how much of a difference this problem formulation would make for solution quality could be an interesting research direction, though several challenges would have to be solved.

Bibliography

Boer, Pieter-Tjerk de et al. (Feb. 2005). 'A Tutorial on the Cross-Entropy Method'. en. In: *Annals of Operations Research* 134.1, pp. 19–67. ISSN: 0254-5330, 1572-9338. DOI: 10.1007/s10479-005-5724-z. URL: http://link.springer.com/10.1007/s10479-005-5724-z (visited on 22nd Mar. 2023).

Browne, Cameron et al. (2012). 'A Survey of Monte Carlo Tree Search Methods'. Undefined/Unknown. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1, pp. 1–43. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2012.2186810.

Chua, Kurtland et al. (2018). 'Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models'. In: *CoRR* abs/1805.12114. arXiv: 1805.12114. URL: http://arxiv.org/abs/1805.12114.

Clarke, L. et al. (2022). 'Energy Systems'. In: *IPCC, 2022: Climate Change 2022: Mitigation of Climate Change. Contribution of Working Group III to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*. Ed. by P.R. Shukla et al. Cambridge, UK and New York, NY, USA: Cambridge University Press. Chap. 6. DOI: 10.1017/9781009157926.008.

Cybenko, G. (1989). 'Approximation by superpositions of a sigmoidal function'. In: *Mathematics of Control, Signals and Systems* 2.4, pp. 303–314. ISSN: 1435-568X. DOI: 10.1007/BF02551274. URL: https://doi.org/10.1007/BF02551274.

Dariane, Alireza B. and Amir Mohammad Moradi (2016). 'Comparative analysis of evolving artificial neural network and reinforcement learning in stochastic optimization of multireservoir systems'. In: *Hydrological Sciences Journal* 61.6, pp. 1141–1156. DOI: 10.1080/02626667.2014.986485. eprint: https://doi.org/10.1080/02626667.2014.986485. URL: https://doi.org/10.1080/02626667.2014.986485.

Energy Information Administration (2022). *Hydropower*. [Online; accessed October 17., 2022]. URL: https://www.eia.gov/energyexplained/hydropower/.

Ernst, Damien et al. (July 2007). 'The cross-entropy method for power system combinatorial optimization problems'. In: *2007 IEEE Lausanne Power Tech*, pp. 1290–1295. DOI: 10.1109/PCT.2007.4538502.

Graabak, Ingeborg et al. (2017). 'Norway as a Battery for the Future European Power System—Impacts on the Hydropower System'. In: *Energies* 10.12. ISSN: 1996-1073. DOI: 10.3390/en10122054. URL: https://www.mdpi.com/1996-1073/10/12/2054.

Haarnoja, Tuomas et al. (2018). 'Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor'. In: *CoRR* abs/1801.01290. arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

Higle, Julia L. (2005). 'Stochastic Programming: Optimization When Uncertainty Matters'. In: *Emerging Theory, Methods, and Applications*. Chap. Chapter 2, pp. 30–53. DOI: 10.1287/educ.1053.0016. eprint: https://pubsonline.informs.org/doi/pdf/10.1287/educ.1053.0016. URL: https://pubsonline.informs.org/doi/abs/10.1287/educ.1053.0016.

IEA (2022). *Renewable Energy Market Update*. URL: https://www.iea.org/reports/renewable-energy-market-update-may-2022 (visited on 3rd Nov. 2022).

Ke, Guolin et al. (2017). 'LightGBM: A Highly Efficient Gradient Boosting Decision Tree'. In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html (visited on 25th Mar. 2023).

Kuznetsov, Arsenii et al. (2020). 'Controlling Overestimation Bias with Truncated Mixture of Continuous Distributional Quantile Critics'. In: *CoRR* abs/2005.04269. arXiv: 2005.04269. URL: https://arxiv.org/abs/2005.04269.

Li, Wenzhe et al. (2023). *A Survey on Transformers in Reinforcement Learning*. DOI: 10.48550/ARXIV.2301.03044. URL: https://arxiv.org/abs/2301.03044.

Luo, Jerry et al. (2022). *Controlling Commercial Cooling Systems Using Reinforcement Learning*. DOI: 10.48550/ARXIV.2211.07357. URL: https://arxiv.org/abs/2211.07357.

Madeka, Dhruv et al. (2022). *Deep Inventory Management*. DOI: 10.48550/ARXIV.2210.03137. URL: https://arxiv.org/abs/2210.03137.

Mars, Patrick de and Aidan O'Sullivan (2022). 'Reinforcement learning and A* search for the unit commitment problem'. In: *Energy and AI* 9, p. 100179. ISSN: 2666-5468. DOI: https://doi.org/10.1016/j.egyai.2022.100179. URL: https://www.sciencedirect.com/science/article/pii/S2666546822000301.

Matheussen, Bernt Viggo, Ole-Christoffer Granmo and Jivitesh Sharma (2019). 'Hydropower Optimization Using Deep Learning'. In: *Advances and Trends in Artificial Intelligence. From Theory to Practice*. Ed. by Franz Wotawa et al. Cham: Springer International Publishing, pp. 110–122. ISBN: 978-3-030-22999-3.

Matouek, Jirí and Bernd Gärtner (2006). *Understanding and Using Linear Programming (Universitext)*. Berlin, Heidelberg: Springer-Verlag. ISBN: 3540306978.

Niu, Wen-Jing et al. (2019). 'Comparison of Multiple Linear Regression, Artificial Neural Network, Extreme Learning Machine, and Support Vector Machine in Deriving Operation Rule of Hydropower Reservoir'. In: *Water* 11.1. ISSN: 2073-4441. DOI: 10.3390/w11010088. URL: https://www.mdpi.com/2073-4441/11/1/88.

NVE (18th Apr. 2023). *Kraftproduksjon*. no. URL: https://www.nve.no/energi/energisystem/kraftproduksjon/ (visited on 15th May 2023).

— (2022). *Licensing*. URL: https://www.nve.no/licensing/ (visited on 15th Oct. 2022).

Pinneri, Cristina et al. (Aug. 2020). *Sample-efficient Cross-Entropy Method for Real-time Planning*. arXiv:2008.06389 [cs, stat]. URL: http://arxiv.org/abs/2008.06389 (visited on 12th Mar. 2023).

Powell, Warren (Sept. 2014). 'Energy and Uncertainty: Models and Algorithms for Complex Energy Systems'. In: *Ai Magazine* 35, pp. 8–21. DOI: 10.1609/aimag.v35i3.2540.

Raffin, Antonin et al. (2021). 'Stable-Baselines3: Reliable Reinforcement Learning Implementations'. In: *Journal of Machine Learning Research* 22.268, pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

Riemer-Sørensen, Signe and Gjert Hovland Rosenlund (2020). 'Deep Reinforcement Learning for Long Term Hydropower Production Scheduling'. In: *CoRR* abs/2012.06312. arXiv: 2012.06312. URL: https://arxiv.org/abs/2012.06312.

Sauhats, Antans et al. (2016). 'A multi-objective stochastic approach to hydroelectric power generation scheduling'. In: *2016 Power Systems Computation Conference (PSCC)*, pp. 1–7. DOI: 10.1109/PSCC.2016.7540821.

Sintef (2022). *Sharm*. URL: https://www.sintef.no/prosjekter/2013/the-stochastic-short-term-model-sharm/ (visited on 15th Sept. 2022).

Statnett (2022). *Quarterly resolution and the energy markets*. URL: https://www.statnett.no/en/for-stakeholders-in-the-power-industry/system-operation/the-power-market/quarterly-resolution-and-the-energy-markets/ (visited on 28th Sept. 2022).

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Second. The MIT Press. URL: http://incompleteideas.net/book/the-book-2nd.html.

Trønder Energi, as (2022). *Private talks with Trønder Energi*.

Xu, Wei et al. (2020). 'Deep Reinforcement Learning for Cascaded Hydropower Reservoirs Considering Inflow Forecasts'. In: *Water Resources Management* abs/2012.06312. URL: https://link.springer.com/article/10.1007/s11269-020-02600-w.

Appendix A

Hardware specifications

The computer with specifications in Table A.1 were used in for everything except training the model-free reinforcement learning method. To train the model-free reinforcement learning method we used Table A.2.

Component	**Specification**
Operating system	Ubuntu 22.04
Processor	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
CPU cores	4
Threads per core	2
Graphics Card	NVIDIA GeForce GTX 1050 Ti
RAM	16GiB

Table A.1: System Specifications for laptop

Component	**Specification**
Operating system	Ubuntu 18.04
Virtual CPUs	4
RAM	16GiB

Table A.2: System Specifications for Azure virtual machine