Universiteit Leiden

Master Computer Science

Reinforcement Learning based optimization/control of production for a single reservoir system

Name: Sudarshna Arya Patel
Student ID: s3259927

Date: June 12, 2023

Specialisation: Master's in Computer Science: Data Science

1st supervisor: Prof. Dr. Aske Plaat
2nd reader: Dr. Mike Preuss

External Superv.: Martijn Deenen & Ruud Kassing

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

In this age, machine learning applications are everywhere [15]. This research work consists of exploratory studies of reinforcement learning in the drinking water domain. In our research, we study how reinforcement learning can be used to control the production of a reservoir in a single reservoir system while leveraging the demand forecast. The work presented here uses two classic reinforcement learning algorithms, Q-learning and N-step. We show how a single reservoir model should be simulated as an RL environment. It is essential for the environment to represent the problem as closely as possible since the quality of the learning of the agent is dependent highly on it. We find that the agents of both algorithms are able to control the production intake while satisfying the environmental constraints. Our main result is that this work shows that reinforcement learning can in principle be used to control water production and be useful for control systems. RL agents are able to maintain the water level within the boundaries. Although they are able to reduce switching between intake set-points to some extent (in comparison to conventional control), improvements can still be made to reach the level of typically advanced control solutions). Hence, further research is necessary to develop a system that is robust enough to be used in practice. Ideas to pursue are for example to create a more advanced environment set up to provide a better learning approach for the agent.

Acknowledgement

Oh, the wonders of academia! As I approach the end of this master's thesis, I feel humbled, appreciative, and a little perplexed by the route that got me here. Please allow me to take a minute to extend my sincere gratitude to the remarkable cast of individuals who have contributed to the unique and instructive nature of this academic trip. So, let's dive into this whirlwind of acknowledgments, with a sprinkle of creativity, a dash of engagement, and just a pinch of entertainment!

First and foremost, I would like to express my deepest gratitude to my ever-patient and infinitely wise LIACS supervisor, Prof. Dr. Aske Plaat along with my supervisors at Royal HaskoningDHV (RHDHV) Martijn Deenen and Ruud Kassing. They have served as my Dumbledore, Gandalf, and Obi-Wan Kenobi rather than merely serving as guiding stars in this universe of knowledge. Their experience, persistent support, and Jedi-like ability to offer helpful criticism were crucial in forming this thesis. I appreciate you using your expert wizardry to illuminate the road to academic greatness. Additionally, I would like to thank Dr. Mike Preuss at LIACS for providing constructive criticism and feedback on my thesis.

Furthermore, I send my gratitude to Daan van Es, Martijn Bakker, and Murşit Sezen at RHDHV who provided me with this wonderful opportunity to work on this project. Working at RHDHV has been nothing short of amazing and has been a delight. I have learned a great deal while working alongside my esteemed colleagues at the organization.

I would also like to extend my appreciation to the brilliant minds at LIACS. Their passion for teaching and research has pushed me to challenge my intellect and nurtured me throughout my journey during this master's program.

Moreover, I'm forever grateful and eternally indebted to my beloved family especially my brother who has been my rock ever since I can remember, and friends who have witnessed the entire journey of my master and especially me transforming from a mere person to a thesis-writing cyborg. Your unconditional love and unwavering support have been my superpower throughout this passage of my life. You have put up with my babbling, my late-night writing sessions, and the much-needed caffeine kickstarts to my thought processes. This thesis is equally yours and mine.

Lastly, a nod of appreciation to the custodians of libraries, archives, and research organizations, who are the stewards of knowledge along with all the staff at LIACS.

I feel a bittersweet mix of eagerness and nostalgia as I say goodbye to this chapter of my scholastic story. The connections I formed, the knowledge I learned, and the memories I created will always retain a special place in my heart. Thank you to everyone who took part in this huge academic show for making it not only educational but also fun, interesting, and utterly unforgettable.

Sudarshna Arya Patel
Leiden, June 2023

Contents

Abbreviations

DQN Deep Q-Networks

HP Hyperparameters

ML Machine Learning

MPFC Model Predictive Flow Control

OPIR Optimised Production by Intelligent ContRol

RL Reinforcement Learning

TD Temporal Difference

List of Figures

List of Tables

1 Introduction

With the rise of machine learning algorithms these days, especially Reinforcement Learning (RL), one starts to wonder if RL can be used in day-to-day life. Specifically, we are curious whether it can be beneficial for water supply infrastructure to improve the overall quality of life and help optimize the hydraulic networks across cities. As a first step because hydraulic networks can be vast, this research is focusing on the use case of optimizing the treatment process at the treatment plant.

Water utilities have been automated around the world for some time now. Traditional automated water systems often use feedback loops to regulate and maintain certain parameters of the system within a desired range such as water flow. These loops typically consist of sensors that measure a parameter, a controller that compares the measured value to a set-point, and an actuator that adjusts the system to maintain the set-point [26]. For example, a water treatment plant may use a control loop to regulate the dosage of a chemical based on the turbidity of the water, or a distribution network may use a control loop to maintain constant pressure in the system. Control loops are a widely used and effective method for automated control of water systems, but they require careful tuning and can be limited by their reliance on fixed set-points and rule-based decision-making [2].

A water supply network consists of the production of water and distribution of that water to the consumers. Usually, the aim is to make sure that the water is of good quality and the distribution is efficient. Traditionally, the water supply systems were controlled manually. Later since 1970s, they have been automated using control loops [8]. Now jumping to 2003, Bakker, Schagen, and Timmer published research that proposed a robust automation of drinking water plants. In the Netherlands, as mentioned in their research the production rate was directly proportional to the water level in the reservoir instead of consumption which results in frequent production rate changes irrespective of water demand, it was a simple control system. In their research, they worked on a control algorithm to get a more stable production rate. They developed a control algorithm *Optimised Production by Intelligent contRol* (OPIR) to obtain a constant production level. To do so, they generated production patterns based on a forecast of consumption for the next 48 hours. They also used extra criteria such as the maximum and minimum water level in the reservoir

The article by Bakker et al. discusses the benefits of implementing model predictive flow control (MPFC) in water supply systems. MPFC is a control strategy that uses predictive models to optimize the flow of water in a system in order to improve water quality and energy efficiency. MPFC is desirable because it provides a more efficient way to fill up the reservoirs during peak demand times. An experimental study [6] showed that predictive control provides better water quality and a more efficient water supply, compared to level-based control systems. This is because MPFC helps optimize the flow of water, and improves water quality and energy efficiency, which can lead to cost savings and other benefits for water utilities and their customers.

Machine learning can help optimize control systems. Therefore, this research aims to find the effects of machine learning on automated drinking water systems. Automated water systems based on machine learning can improve upon traditional control loops by leveraging data to make more informed and dynamic decisions. Machine learning algorithms can be trained on historical and real-time data to identify patterns and make predictions about future system behavior. These predictions can then be used to optimize system performance by

adjusting control parameters in real-time, rather than relying on fixed setpoints. For example, machine learning can be used to optimize pump schedules, reduce leaks and bursts in distribution networks, and predict water quality issues before they occur [1]. However, the success of machine learning in water systems requires access to high-quality data, sophisticated models, and careful integration with existing control systems [30]. Additionally, ensuring that the models are transparent and explainable is crucial for maintaining operator trust and confidence in the automated system [10].

This research focuses solely on the impact of reinforcement learning on a drinking water system in the Netherlands. It inspires by a study by Bakker, Schagen, and Timmer and researches how reinforcement learning can be used to generate intake setpoints predictions based on the consumption pattern forecast. To achieve our goal, we worked on determining flow intake based on current consumption for which historical consumption patterns were used for learning. Later, a learned model can be used to determine a flow intake for a consumption forecast. Reinforcement learning is a type of machine learning in which an agent learns to interact with its environment in order to maximize the reward. In reinforcement learning, an agent learns through trial and error, receiving positive or negative feedback in the form of rewards or punishments as it takes actions in an environment as shown in Figure 1. Hence, reinforcement learning can be applied to optimize a drinking water system by controlling the intake pump settings to maintain a consistent flow by training an agent to take actions that maximize a reward signal based on the system's performance. The agent receives feedback through the reward on its actions and adjusts its behavior to achieve better performance over time. This approach can be used to optimize various aspects of the drinking water system, such as water quality, energy consumption, and distribution network efficiency. For example, the agent can learn to adjust the flow rates of pumps and valves to optimize the distribution network, or to adjust treatment processes to ensure water quality meets regulatory standards. However, reinforcement learning requires a large amount of training and significant computational resources to train the agent, and the complexity of the system may make it difficult to define a suitable reward signal.

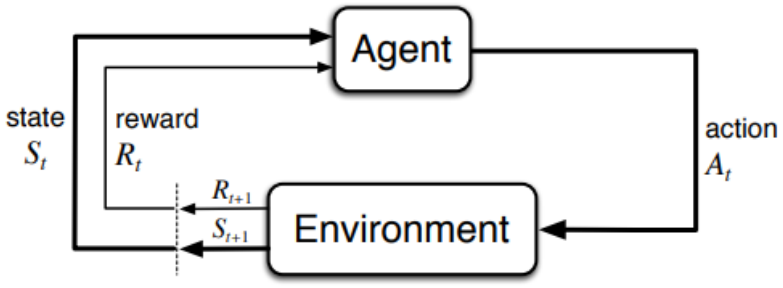


Figure 1: An agent and environment in RL adopted from [36]

1.1 Problem Statement

A drinking water system consists of two main components, water production and water supply. This thesis targets the water production sub-system of the drinking water system. The

water production systems consist of water pump intake settings, water tanks and information regarding consumption from the water tanks.
The problem statement of this thesis is: *Can we control intake flow for a single reservoir system using reinforcement learning?*

We try to answer the above-provided problem statement by researching RL agents where the aim is to maintain the water level in the reservoir for a given consumption forecast while satisfying reservoir boundary conditions and preventing frequent changes in intake pump settings. This later leads to the optimization of the drinking water control system. The boundary conditions here imply that the water level of the reservoir should not go beyond a certain minimum or maximum water level leading to underflow or overflow respectively.

This research tries to answer the following research questions by using the two main RL algorithms such as Q-learning and N-step have been used which will be discussed in the later chapter 4

1. Will the RL agent be able to learn the optimal policy using Q-learning with a desired number of production pump intakes?
2. Will a Q-learning agent be able to foresee and predict consumption values and plan/schedule accordingly?
3. If Q-learning is not able to achieve the desired results, will the N-step algorithm be beneficial in such cases?

To answer the research questions given above two types of data have been used, a noisy sine wave data artificially created and data collected from a treatment plant in Leersum. First, we measure the performance of various models on artificial data to better understand and analyze the findings of the models. Later, we use the best model found using artificial data for a case study on real data to compare the performance of the RL model with the existing control systems.

1.2 Scientific Method

This thesis answers the research questions mentioned above by performing various experiments on a drinking water system. To perform the experiments, a reinforcement learning environment for a simple reservoir system with some constraints and a demand forecast was created and used, details of which will be discussed in chapter 3. Additionally, various classical reinforcement learning algorithms have been used to run these experiments. Chapter 4 provides detailed explanations of experiments and chapter 5 presents the analysis and insights of this research.

1.3 Reading Guide

First, chapter 1 provides the introduction of the thesis. Then chapter 2 contains the background work related to this problem and the literature relevant to the topic. Subsequently, chapter 3 consists of the methodology in detail for the problem and how this real-life problem

has been simulated in a computer to run the experiments. Furthermore, chapter 4 provides details regarding experiments performed and their setups. Moreover, chapter 5 discusses the results of the experiments performed and provides in-depth insights and analysis of results and findings. And Finally, chapter 6 summarises this research work and provides the conclusion of this research work and the possible future research.

2 Related Work and Literature Review

The simulations of the movement and behavior of water in natural or man-made systems are done with the help of computer programs, these programs are called water simulation models. These models provide a way to understand and predict the behavior of water in several scenarios, such as floods, droughts, water supply, and pollution control. As a result, these models provide insights to help prevent, control and optimize various real-life situations regarding water systems. There are several water simulation models used around the world, some of the most common ones are Soil and Water Assessment Tool (SWAT [37]), Environmental Protection Agency Network (EPANET [27]), Storm Water Management Model (SWMM [11]), Public Domain Storm Water Management Model (PCSWMM [31]). These models are constantly evolving and improving, and are used by researchers, engineers, and policymakers around the world to better understand and manage water resources.

Automated hydraulic networks are computerized systems that are used to control and manage water supply and distribution networks. These systems consist of sensors, controllers and algorithms to optimize water flow, pressure, and quality in real-time. These controllers consist of various control loops using control strategies such as model predictive control (MPC). Nowadays automated hydraulic networks are everywhere and some of them are Tokyo Waterworks - Japan [39] where the Bureau of Waterworks, Tokyo Metropolitan Government (BWT) currently has the capacity to treat approximately 5.5 million m3 per day using the advanced water treatment process, Thames Water - UK [7] where they have built a weekly water quality monitoring data for the River Thames.

The Netherlands is a country that is highly advanced in terms of its water management and distribution systems [22, 24, 34]. As a result, there are several automated hydraulic networks operating throughout the country. Some of the most notable systems in The Netherlands are provided by companies such as Vitens, Waternet, Dunlea, and PWN. Most of these companies have systems that use sensors, telemetry, control systems, and data analytics to optimize water supply, pressure, and quality in real-time or reduce water loss or detect leaks in real time.

The popularity of automated hydraulic networks is increasing rapidly nowadays which is due to the fact that they help control and optimize water supply and distribution more efficiently in comparison to traditional or manual ways [12]. These automated systems can help reduce water loss, improve water quality, and ensure a reliable supply of water to customers [16].

2.1 Drinking Water Networks

Drinking water networks are the interconnected systems of pipes, valves, pumps, and other infrastructure that transport drinking water from the source of supply to the consumer as shown in Figure 2. The design, operation, and maintenance of these networks are critical for ensuring safe and reliable access to clean drinking water.

The Netherlands has a well-developed drinking water network that delivers high-quality water to its citizens. The network consists of a complex system of pipes, storage facilities, treatment plants, and distribution centers that are operated by various water supply companies.

Automated drinking water networks using machine learning are an emerging technology

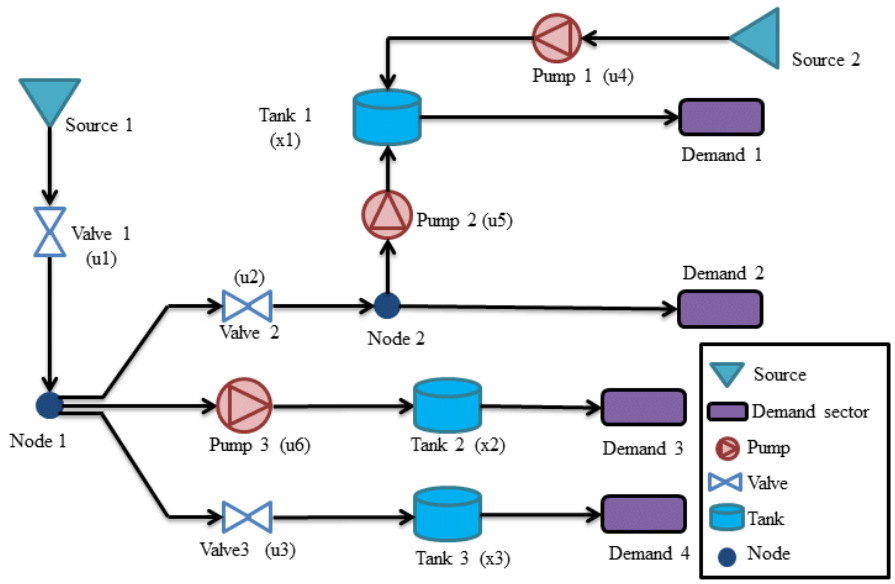


Figure 2: Drinking water network diagram [17]

that uses sensors, data analytics, and machine learning algorithms to optimize the operation and maintenance of drinking water networks. The Netherlands is one of the countries at the forefront of implementing automated drinking water networks. OPIR is a product developed by Royal HaskoningDHV, a Dutch engineering consultancy company. The program aims to optimize the performance, innovation, and reliability of water treatment plants, wastewater treatment plants, and drinking water distribution networks. In 2013, Bakker et al. published that the program can predict water demand in real-time. The fully adaptive forecasting model is able to predict with the help of regression methods, it analyzes historical water demand data and other relevant variables/disturbances, such as weather, seasonality, and day of the week, to make predictions about future demand. They also mention that the model is capable enough to adapt to changes in demand patterns and can update its parameters automatically as new data becomes available. This model output can help optimize water production and make the water production and distribution system more efficient overall. This was later shown in 2014 in a case study [4], where it was verified that by using advanced predictive models and real-time data from sensors, the water systems can optimize water supply operations, reduce water loss and energy consumption, and improve overall system reliability and performance.

Once the system has the information regarding future water consumption, it can produce water more efficiently instead of producing water without consumption knowledge and solely depending on level-based information which helps optimize the entire water operations. Water production can be automated and optimized using various techniques such as machine learning including reinforcement learning. Drinking water control is a classic problem in reinforcement learning where an agent has to learn to control the level of water in a reservoir. The goal is to maintain the water level within a specific range by taking actions such as pumping in water, releasing water, or maintaining the previous action while satisfying several other boundary conditions.

2.2 Reinforcement Learning

Reinforcement learning is a subdomain of machine learning. It consists of algorithms that help an agent learn to achieve a specific goal by interacting with an environment. An agent learns by taking action and receiving feedback from the environment, which represents how well the agent performs. It is a trial-and-error approach.

The aim is to learn the optimal policy, which represents a set of rules that help decides which action in a given state provides the maximum cumulative reward over time. Since it is a trial-and-error approach, balancing between exploration and exploitation is required to achieve the optimal policy. Exploration represents when an agent selects new actions and exploitation represents when an agent uses previously learned information to select an action in a given state.

RL can be applied to many fields and domains including robotics [19], game playing [32] and control systems [13, 18, 20]. One of the examples from the gaming domain would be AlphaGo [33], a computer program that won while playing a game of Go with the world champion, this example shows the extent of capabilities of a well-trained RL model and its many possibilities in real life.

2.2.1 Reinforcement learning methods

As mentioned in [38], there are two types of methods used in reinforcement learning. First, where the agent has the access to transition and reward model of the problem, these methods are called model-based. Second, where the agent acquires the information regarding state and reward through experience, called model-free methods. Q-learning, SARSA, and Deep Q-Networks (DQNs) are commonly used model-free algorithms whereas Dyna-Q and Monte Carlo Tree Search are model-based algorithms.

There are various algorithms in RL and each algorithm is suitable for specific types of problems and their environment. An overview of some of the most commonly used RL algorithms is provided as follows.

1. Temporal difference (TD): TD uses sequences of observations and rewards to learn about the long-term value of actions. In TD learning, the agent estimates the expected return for each action by bootstrapping, or using the current estimate of the value of the next state to update the value of the current state [35].

2. Q-learning: This is a type of TD learning algorithm that uses a Q-table to store and update estimates of the expected return for each action at each state. The Q-table is updated using the Bellman equation, which takes into account the expected future rewards for each action [38].

3. SARSA: This is another temporal difference learning algorithm that uses a similar approach to Q-learning, but it updates the Q-table using the expected return for the next action that the agent will take, rather than the maximum expected return over all possible actions [28].

4. Deep Q-Network (DQN): This is a variant of Q-learning that uses a neural network to approximate the Q-function, rather than a Q-table. This allows the agent to handle more

complex environments and learn more efficiently. DQN is capable of handling discrete as well as continuous observation spaces. Additionally, it is also good at problems that have a high-dimensional observation space [23].

For drinking water production based on future consumption prediction problems using reinforcement learning, various RL methods can be used. This is a control system problem where the system is required to look ahead to the future to determine future actions. One could use model-free algorithms, such as Q-learning or deep Q-networks, to learn a mapping from states to actions directly. In this case, there is no need to learn a dynamics model explicitly, but the agent would need to explore the environment to learn an optimal policy. However, prior to RL algorithms, one would have to define the observation space, action space, reward function, and environment dynamics to implement this real-life problem into computer language.

2.3 Water systems using Reinforcement Learning

As stated previously, RL can be applied to optimize the water systems. These water systems are of many types such as sewage treatment plants, irrigation systems, water conservation systems, and water distribution networks. There has been some research done in the field to verify the usefulness of RL. In 2021, [9] explored the possibilities to optimize control for sustainable wastewater treatment plants where they used a Multi-Agent Deep Reinforcement Learning (MADRL) technique to simultaneously optimize dissolved oxygen and chemical dosage in a wastewater treatment plants (WWTP) to achieve sustainable optimization. Lambregts provided exploratory research on the potential for a reinforcement learning controller for an optimization problem of IJmuiden pumping station which drains water from the Noordzeekanaal-Amsterdam-Rijnkanaal system into The North Sea where the author used deep Q-learning to solve this optimization problem. Hu et al. published research regarding real-time scheduling of pumps for a water distribution system using exploration-enhanced deep RL to maximize energy efficiency. Saikai, Peake, and Chenu proposed a deep reinforcement learning (RL) framework for irrigation optimization. The authors demonstrated the effectiveness of the framework by performing a case study where the framework used deep RL to learn the irrigation schedule.

There are many other existing and ongoing research using reinforcement learning in the water domain. These researches do provide insight into how RL can be beneficial for water systems with sufficient proof. However, due to limited data and many constraints related to how a real-life problem is simulated in the computer, these systems are not very practical and may not be used in real life to the extent one would hope to achieve because of technical limitations.

3 Methodology

This section provides details of the methodology and experimental setup to solve drinking water optimization. As mentioned earlier, the drinking water system contains two main subsystems and those are water production and water distribution. For this optimization problem, we are focusing on water production optimization based on predicted future demand. We are working with a one-reservoir system with various intake pump settings and predicted future demand, a sample diagram is shown in fig 3.

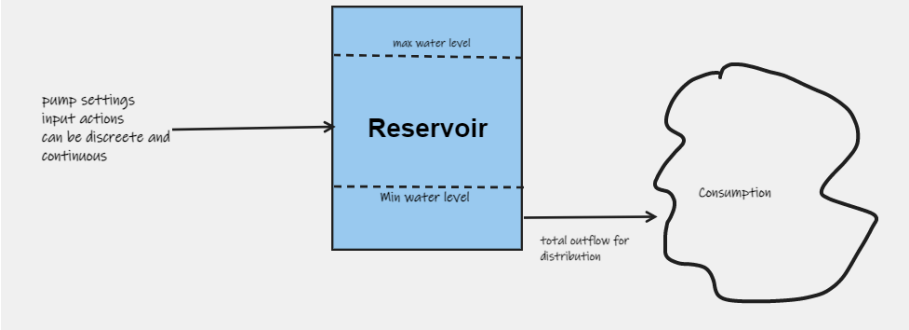


Figure 3: A single reservoir drinking water network diagram for RL

3.1 Method

There are certain steps required prior to performing the experiments using various RL methods. We have to build a reinforcement learning environment for the drinking water system. This process consists of a few substeps which are mentioned as follows:

1. Define the problem: This step requires clearly defining the problem. In this case, the problem to solve is maintaining the water level in the reservoir while the water level satisfies certain boundary conditions.

2. Implement the environment: An environment should be designed in such a way that it provides feedback similar to the real world to the agent based on state and action spaces.

   (a) Design the state space: The state space contains the relevant information regarding the system which is required to make decisions about the problem. Here, for the drinking water system, it would be the water level, the hour of the day, the day of the week etc.

   (b) Design the action space: Action space consists of actions that can be taken to modify the water network. This could include water production pump settings to maintain the flow in the reservoir.

   (c) Design the reward function: The reward function is used to redirect the agent to take actions that lead to the desired outcome. Here, for this problem, positive feedback to stay within certain water boundaries and negative feedback on frequent switching between water pump settings.

(d) Design the transition function: The transition function is required to update changes in the state when an action is taken by the agent.

3. Test and Evaluate: The last step is to train and test for various RL algorithms to find the optimal policy for a water system RL environment.

3.2 Environmental Design

As mentioned previously, we are working with one reservoir system. An overview of the reservoir environment and its major components including state space, action space, and reward structure is provided in the following subsections.

3.2.1 State Space

The state space represents the reservoir configuration at a given time and provides all the necessary information to the agent to determine which action to take. For our system, the state consists of the following features.

- Water level in the reservoir
- Hour of the day
- Day of the week

Water Level
The water level represents the current volume of water in the water tank in m^3. We used discretization to represent the current water level because it provides dimensionality reduction and helps with the tabular methods to store Q-values. Two types of discretization were used, integer and bucketing. In the case of the integer, the round-off value of the current water level was considered and for bucketing, various bucket sizes were used to represent certain water level ranges. For the training, the water level was initialized randomly within the bounds to provide more exploration of the state space helping the agent learn optimal policy.

Hour
The hour represents the time of the day. Since the consumption of water is dependent on the time of the day, the time has been taken into account for the state space. For the hour, 24-hour-numeric representation is used where $i \in [0, 23]$.

Day
The day represents the day of the week. Similar to the hour of the day, consumption is also dependent on the day of the week. To represent the day in the state space, a numeric system was used where Monday to Sunday was represented by $[0, 6]$.

3.2.2 Action Space

The agent should be able to take actions related to pumping the water into the reservoir to let the water in. Since we are using a discrete action space, the action will be consisting of switching on/off the pump with a few intake settings in m^3/h.

3.2.3 Reward Structure

The reward is the feedback provided by the environment to the agent upon taking an action in a given state, it essentially directs the agent to move in the desired direction and helps the agent learn. The reward is one of the key aspects of RL. The reward may consist of various sub-components depending on the problem. For the given problem, the goal is to maintain the water level in the reservoir within certain bounds with an optimal switching of intake pump settings. This implies that there will be two types of sub-rewards which make the total reward for an action taken. The first one is based on the water level and the second one will be dependent on the action switch.

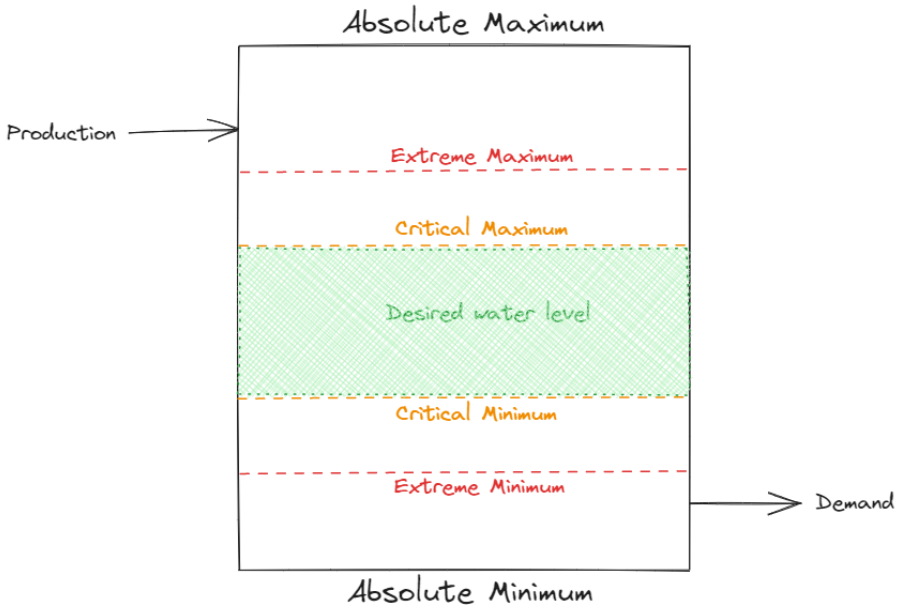


Figure 4: A representation of a reservoir water boundaries

Water level reward
There are six boundary conditions for the water level in the reservoir as shown in Figure 4 and their meaning is mentioned below.

- Absolute maximum - represents the completely full reservoir i.e. the total volume of the reservoir in m^3. Reservoirs should never be filled to this extent, it is an overflow scenario.
- Extreme maximum - represents the hard boundary for the water level in the reservoir, this is the second boundary limit
- Critical maximum - represents the soft boundary for the water level in the reservoir meaning the water level should be less than this, this limit acts as a first warning
- Critical minimum - represents the soft boundary for the water level in the reservoir meaning the water level should be higher than this, this limit acts as a first warning
- Extreme minimum - represents the hard boundary for the water level in the reservoir, this is the second boundary limit

- Absolute minimum - represents a completely empty reservoir i.e. 0 m^3. Reservoirs should never be completely empty, it is an underflow scenario.

The severity of the reward depends on the water level boundaries given critical, extreme and absolute represent the ascending severity order and the rewards are -10, -100 and -1000 respectively. A reward of $+1$ is given if the water level is within the desired range.

Action switch reward
Since the aim is to reduce frequent switches between intake pump settings, there is a negative reward of -10 when the picked action differs from the previous action to prevent frequent switching. However, in certain situations, it is necessary to switch to maintain the water level boundary conditions.

Scaling
Scaling is essential when there are multiple components of rewards as each component represents a desired output and we would have to decide which output/sub-problem should be prioritized to solve the overall problem and find the most optimal solution. For the given problem, we have decided to find the optimal policy which predicts the sequence of actions for a given consumption pattern where the water level satisfies the boundary conditions with an adequate number of intake pump switches. Hence, we are prioritizing water level over the number of switches in this scenario which is why the reward for water levels is severe in comparison to the action switch. There are many ways to scale the rewards and for this problem, we decided to use the logarithmic scale where the reward values are $+10^0, -10^1, -10^2, -10^3$.

3.2.4 Consumption

Unlike RL agents, the environment has access to the deterministic consumption pattern of the next three days which is being used for learning, Figure 5 displays a sample consumption pattern. Hence, the environment contains the knowledge of consumption at any given time and this consumption value is used in environment state transition. As mentioned in chapter 1, this consumption pattern is taken from historical data during the learning process of RL agents. Here, the agent doesn't have access to consumption directly like the environment but it learns the changes in consumption implicitly during the learning phase. And because of this, these learned agents can be used to determine the intake flow for real-time consumption as well as for a consumption forecast where they will provide a series of actions that could be taken for a consumption forecast based on a simulation.

3.2.5 State Transition

The transition function determines the next state of the environment when the agent takes an action in a given state. Here, the next state is calculated using the current state, the action taken i.e. the water intake and water consumption at that moment of time.

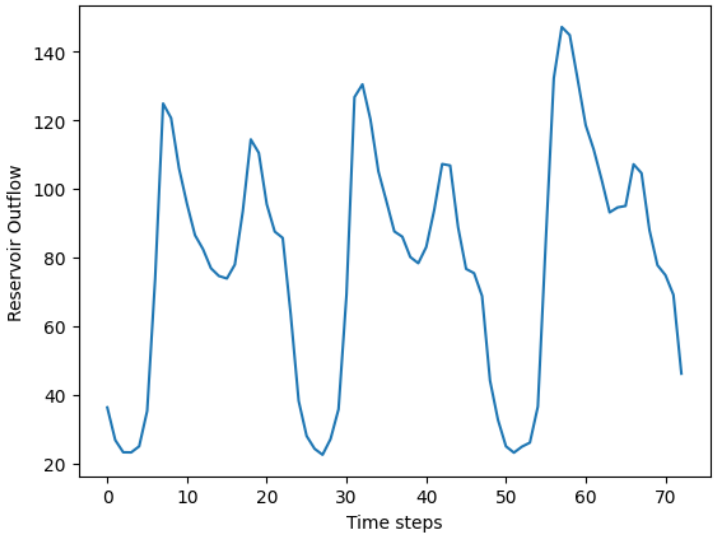


Figure 5: A sample consumption(outflow) pattern for a reservoir for a period of 72 hours (three days). Y-axis represents outflow in m^3/h and X-axis shows time steps where 1 timestep = 1 hour implying 72 timesteps

3.3 Reinforcement Learning Algorithms

For experiments, we used Q-learning and N-step to answer the research questions provided in chapter 1 by analyzing and understanding the results of the experiments, details of which are given in this section.

For our experimental setup, we experimented with various numbers of episodes during training and each episode consists of 72 timesteps per epoch (representing the next 3 days) where each time step is an hour.

3.3.1 Q-Learning

The Q-learning algorithm uses Q-table where Q-table is a finite array with some default values in the beginning that represents the mapping of the state space vs actions and is filled with Q-values. These Q-values are updated using the weighted average of the current value and the new information. Since Q-table is finite, the state space and action space are required to be discrete.

For this algorithm, various types of observation space were experimented with several state-space parameters, details of which are provided in section 4.5. In summary, the state space consisted of either or in combination of parameters such as water level, hour of day and day of the week. The action space contains the pump intake settings as mentioned previously.

Additionally, Q-learning uses a few hyperparameters which can be set to optimize the learning process of the algorithm and those are the learning rate, discount factor and exploration rate.

- The speed of updates of Q-values is dependent on learning rate α as it determines how

fast or slow the agent should be updating these values. Hence, α determines the rate of agent learning.

- The discount factor γ decides how much emphasis should be given future rewards. A lower value of γ means the agent puts an emphasis on immediate rewards while a higher γ means the agent prioritizes future rewards.
- The rate at which the agent will explore or exploit the previously learned knowledge is called exploration rate ϵ. A lower rate means the agent will use already learned knowledge more which helps the agent go deeper in an area to find the best solution while a higher ϵ represents the agent will explore more instead of exploiting which provides more state-space coverage.

For our experiments, we used various values of these hyperparameters to find the values that yield the most optimized solution.

3.3.2 N-Step Temporal Difference

N-step algorithm was used to consider future rewards into account and to overcome some of the Q-learning disadvantages. We used a similar algorithmic setup as Q-learning with similar state space, action space and the same values of hyperparameters. Additionally, various values of step-size n were used to find the most optimal value of n as n represents how many steps should be considered to perform bootstrapping meaning looking ahead in the future. Since a time step represents an hour, the value of n represents how many hours to foresee in the future.

4 Experiments

The methods described previously were implemented and applied to the artificial data and the data collected over the years for a real system. The experimental setup provided in this chapter were designed to answer the research questions mentioned in the beginning in section 1.1. The results are given in chapter 5.

4.1 Dataset

Two types of datasets were used to run and compare the results.

Artificial Data
The first dataset used to train the agent for all the algorithms was artificially created. We simulated a dataset containing future consumption values of a year. This dataset consists of noisy sine wave data. This dataset was used because it has fewer disturbances in the pattern and will be easier for an agent to learn. This would provide a better understanding of the agent's learning on consistent data vs data with significant disturbances.

Leersum dataset
Leersum is a town located in the Netherlands. This dataset contains the consumption history of a reservoir that provides water to the town of Leersum. This dataset contains data from 2022 onward implying data from a year. It consists of the water level in the reservoir, actual consumption and inflow in the reservoir.

Boundaries	**Water volume value (m^3)**	**Filling %**
Full capacity	1350	100%
Extreme maximum	1080	80%
Critical maximum	1053	78%
Critical minimum	567	42%
Extreme minimum	540	40%
Minimum capacity	0	0%

Table 1: Water level boundaries for the experiment setup

Pump intake	Value (m^3/h)
Off	0
Intake setting 1	70
Intake setting 2	140
Intake setting 3	210

Table 2: Water production pump intake set-points

4.2 Reservoir Parameters

All the experiments were run on a reservoir setup with water boundaries defined in Table 1 with pump intake settings given in Table 2 unless otherwise stated in experiments and results. These values of the reservoir volume, water boundaries and intake capacities are taken from the real-world reservoir located near Leersum.

4.3 Training and Testing

Since RL is inherently an online learning method, there is no necessity to train and test separately. The comparison of cumulative rewards curves would suffice to compare the performances of various algorithms or models. However, if we desire to achieve more accurate results comparison for testing and have a more precise comparison between approaches, it is better to have a separate testing setup with learning disabled to evaluate learned policies. Hence, an end-to-end evaluation for a system would be to just compare the cumulative rewards but for engineering/experimental purposes, a separate evaluation would be more accurate.

We used 70% of the total data for training and the remaining for testing. Each pattern in training represents three day period consisting of 72 timesteps representing an episode. During training, each pattern was used 10 times to increase the possible learning combinations wrt the initial state of the system since we did not have enough data to train with a single pattern being used only once.

The remaining 30% was used for testing. For testing, two types of testing modes were used while learning was disabled.

- Testing with exploration enabled - This was done to make the system more realistic as in the real-world, there is usually some randomness involved.
- Testing with exploration disabled - This was done to provide more supervised testing and to provide the exact comparison between learned policies.

4.4 Exploration Strategy and Hyperparameter Optimization

For all the experiments, the epsilon-greedy strategy has been used to select the action for a state.

For all the algorithms used, hyperparameter optimization was used to find the combination of hyperparameters such as learning rate α, exploration rate ϵ and discount factor γ that provides the most optimized policy which was later used for testing. The values that were used are $\alpha = [0.001., 0.01, 0.1]$, $\epsilon = [0.1, 0.2, 0.3]$ and $\gamma = [0.95, 0.85, 0.8, 0.6]$. These hyperparameter values were inspired from the book by Plaat.

4.5 Experimental Setups: Q-Learning

To answer the research questions related to Q-learning which are research question 1 and 2 given in section 1.1, there were two types of environment state spaces that were used to

experiment as we moved from the simplest environment to slightly more complex environments to represent this real-world problem and those are given as follows.

- Approach 1: States were represented using only one parameter i.e. water level in the reservoir.
- Approach 2: States were represented using three parameters water level, hour and day.

4.5.1 Approach 1

Since states are represented using only one parameter i.e. water level in the reservoir in format *(waterlevel)*, the observation space of the environment is represented as an integer value which is a discretized representation of the water level in the reservoir as an integer in the range is [0, maximum capacity of reservoir). The action space contains the pump intake settings as mentioned earlier.

Action / State	0	150
0	-100	0
1	0	1
. . .	10	0
299	0	0

Figure 6: A sample qtable

For approach 1, two implementations of Q-table were used.

- A Q-table that represent all possible combination of observation space vs action space i.e. a 2-dimensional array with Q-values. The sample Q-table for a system that has a maximum water capacity of 300 m^3 with two pump intake settings of 0 or 150 $m^3/hour$ is shown in fig 6.
- A 3-dimensional Q-table that represents the Q-values for a combination of observation space vs action space vs previous action space. This was done to provide additional learning incentives for the agent along with a reward function in relevance to switching between actions and the direction of which an agent should move. The details with results are provided in section 5.1.1

Since approach 1 considers water level only, approach 2 was developed to verify if the learning of agents can be improved with additional information in the observation space. Hence, approach 2 was created in order to provide an opportunity for the agent to learn predicted consumption value based on time implicitly while training.

4.5.2 Approach 2

For the second approach, a bucket system was used to represent the water level in the reservoir where the bucket number will be calculated based on a bucket size to make the state space a bit more discrete, the bucket size used was $27m^3$ for all the experiments which would create 50 buckets since the reservoir's maximum capacity is $1350m^3$. This discretization was considered to reduce the combinatorial values when hour and day were added to the state space. Each state is represented by the tuple *(bucket, hour, day)* and observation is an integer that encodes the corresponding state. The action space used remains same as the approach 1 where the action space contains the pump intake settings. Lastly, the Q-table of 2-dimensions representing observation vs action space possible combinations were used to learn Q-values.

4.6 Experimental Setups: N-Step

To answer our research question 3 regarding N-step given in section 1.1, the same environment used for Q-learning approach 1 was used with a similar 3-d Q-table as Q-learning approach 1 first configuration. Additionally, various value of n were experimented with $n \in [1, 5, 10, 24, 48, 72]$.

5 Results and Discussions

This chapter provides an in-depth analysis and insights into the experiments performed for this research.

5.1 Noisy Sinewave Data

5.1.1 Q-Learning

In this subsection, we discuss our findings on Q-learning RL agents.

5.1.1.1 Approach1

Figure 7 contains the training results for Q-learning using a 2-dimensional Q-table. It is seen that as the training progresses the average cumulative reward per episode increases and starts to plateau after certain episodes, representing that the RL agent is able to learn the optimal policy based on the conditions implemented. Additionally, the number of overflow and underflow are also reducing over time. It is to be noticed that, the number of action/pump-intake switches also reduces over time and it lies in range (42, 70). However, unlike overflow/underflow count, the reduction in action switch is not drastic.

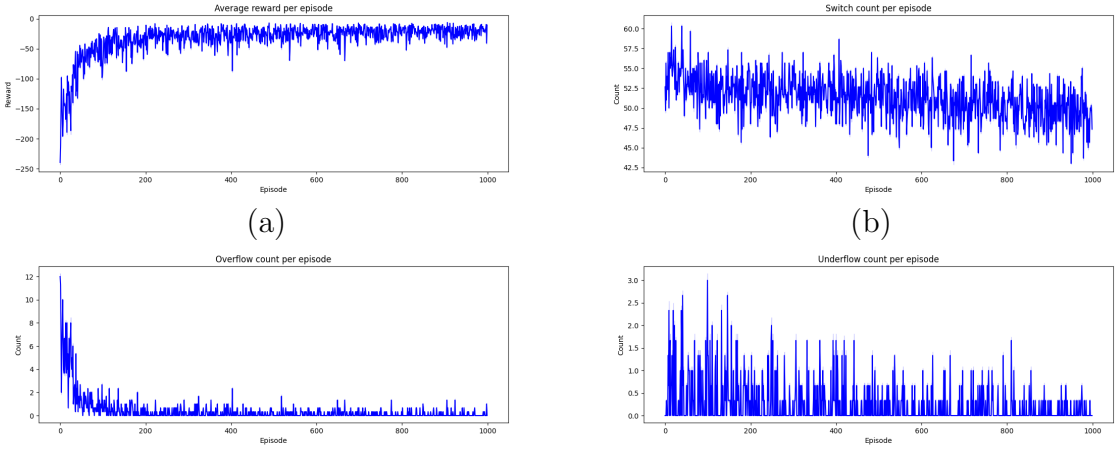


Figure 7: Approach1 - Q-learning with 2-dimensional Q-table ($\alpha = 0.1, \epsilon = 0.2, \gamma = 0.6$): Training results **(a)** Average reward per episode during training, **(b)** Action change count per episode during training **(c)** Overflow count per episode during training **(d)** Underflow count per episode during training

Note: For this chapter, the term "overflow count" represents all the instances with water level above the critical maximum boundary, and the term "underflow count" represents all the instances with water level below critical minimum boundary

Additionally, we ran experiments to find the best value for α, ϵ and γ while keeping other two hyperparameters constant. Figure 8 depicts the effect of learning rate on the performance of Approach1 where the $\alpha \in [0.001, 0.01, 0.1]$ with an exploration rate ϵ of 0.1 and discount factor γ of 0.85. The learning rate determines how quickly the model is able to adapt. The smaller learning rate will require higher episodes to learn and will train much slower, while a higher learning rate will require fewer episodes to learn but might have a risk to jump over minimas and might not able to achieve optimal policy. Hence, it is important to find the right value of the learning rate for the problem as it depends on the actual problem that one is trying to solve. In this scenario, the agent is able to learn faster and reaches optimal policy quickly when $\alpha = 0.1$ compared to $\alpha = 0.001$ as we can see in the plot given at the top. The disturbances in reward are more frequent for $\alpha = [0.01, 0.001]$ while they are significantly less for $\alpha = 0.1$. The same trend can be seen in the other three plots given in the same figure where the number of switches, the number of overflow counts and the number of underflow counts per episode were able to reach their respective optimal results quickly in case of $\alpha = 0.1$.

Figure 9 presents performance changes for various values of exploration rate where $\epsilon \in [0.1, 0.2, 0.3]$ with a learning rate of 0.1 and discount factor of 0.85. The exploration rate represents the probability at which the agent will explore the environment rather than use the information learned or exploit. The exploration is necessary to improve the agent's knowledge regarding each action for the long-term benefit while the exploitation represents the agents using current estimated values and using those to get the most reward. Hence, the higher exploration rate implies that agent will explore more and often times it can be a good strategy, but it is essential to find the balance between exploitation-vs-exploration to achieve the most optimal policy as it helps the agent to navigate in the desired direction. The plots show that the $\epsilon = 0.1$ provides best results among all three values of exploration rate. The plot for average reward per episode has minimum fluctuations implying that the agent is able to achieve the desired balance between exploration and exploitation.

Figure 10 contains the performance plots of training for various value of discount factor $\gamma \in [0.6, 0.8, 0.85, 0.95]$ with a learning rate of 0.1 and an exploration rate of 0.1. The discount factor determines if the agent should care about future rewards relative to the immediate reward. if $\gamma = 0$ the the agent only considers immediate reward while if γ is close to 1 then the agent considers most of its future rewards. Hence, usually for continuous problems the discount factor is close to 1. For this problem, it could be continuous or episodic depends how one plans to set up the details. Hence, we tested with four different values of γ as mentioned previously. It is seen that $\gamma = 0.6$ provides the best results for the Approach1 setup with $\gamma = 0.8$ performing almost at the same level. In average episode reward plot, they look almost identical but the significant difference can be seen in switch count plot where $\gamma = 0.6$ takes the lead. However as the training time increases, the performance of $\gamma = [0.6, 0.8]$ becomes almost identical as shown in all the pots of the figure mentioned.

We saw that for various learning rates $\alpha = 0.1$, for various exploration rates $\epsilon = 0.1$ and for discount factors $\gamma \in [0.6, 0.8]$ provide the best results when measuring the performance for each hyper-parameter individually. However, it might happen that combining them together might not provide the best performance as sometimes, the combination of individual hyper-

parameter's best values may not lead to the most optimal solution. Hence, we performed a grid search hyperparameter optimization where $\alpha \in [0.001., 0.01, 0.1]$, $\epsilon \in [0.1, 0.2, 0.3]$ and $\gamma \in [0.95, 0.85, 0.8, 0.6]$. Table 3 contains the five hyperparameter combinations that provide the best performance. To measure the performance of each combination, we measured the average cumulative reward per episode per last 100 episodes along with the other quantifiable variables such as switch count, overflow count, and underflow count, the configuration given in this table are the ones that provides the highest reward.

Rank	α	ϵ	γ	**Reward**	**Switch count**	**Overflow count**	**Underflow count**
1	0.1	0.1	0.60	-12.351250	37.660000	0.010000	0.033333
2	0.1	0.1	0.80	-13.184167	38.103333	0.036667	0.030000
3	0.1	0.2	0.60	-15.712222	40.693333	0.073333	0.060000
4	0.1	0.1	0.85	-16.728333	44.500000	0.093333	0.063333
5	0.1	0.2	0.80	-17.700833	41.593333	0.090000	0.073333

Table 3: Approach1 - Q-learning with 2-dimensional Q-table: Training results of configurations with highest average cumulative reward per last 100 episodes

Figure 11 visualizes the results of specific test cases run using and agent which is trained using Q-learning with 2-dimensional Q-table shown in Figure 6. Each of these figures shows the water level in the reservoir, the distribution value and the action taken at each timestep during the test run for a specific test scenario. As mentioned in section 4.3, we tested with learning disabled with and without exploration strategy showing in the left and right part of the Figure 11 respectively. It seen from most of these plots given in Figure 11 is that the reservoir water level stays in water boundaries for a given distribution value at a given timestep. It is seen that the reservoir water level does overflow and underflow to some extent in left-side figures while this doesn't happen in right-side figures, this is because the exploration strategy is enabled in left-side test mode and disabled in right-side test mode, implying that this is due to randomness present in the left-side system. However, there is not a significant difference in action switch count for both scenarios. It is seen that the agent with an exploration-enabled strategy crosses water boundaries and this is because the enabled exploration strategy introduces randomness in the environment while the agent with only exploitation is able to stay within the water bounds.

From the reinforcement learning point of view, it is an acceptable solution because it is seen that the agent is able to learn the optimum policy based on the environmental constraints provided nevertheless it might not be the most practical solution for our real-world problem. The frequent switch between intake-pumps will have practical limitations such as the reduction in pump's durability which could lead to a deteriorating infrastructure and maintaining the infrastructure is essential as it is expensive to install/fix something in the hydraulic networks. Hence, the reduction in switch count is not up to the desired level.

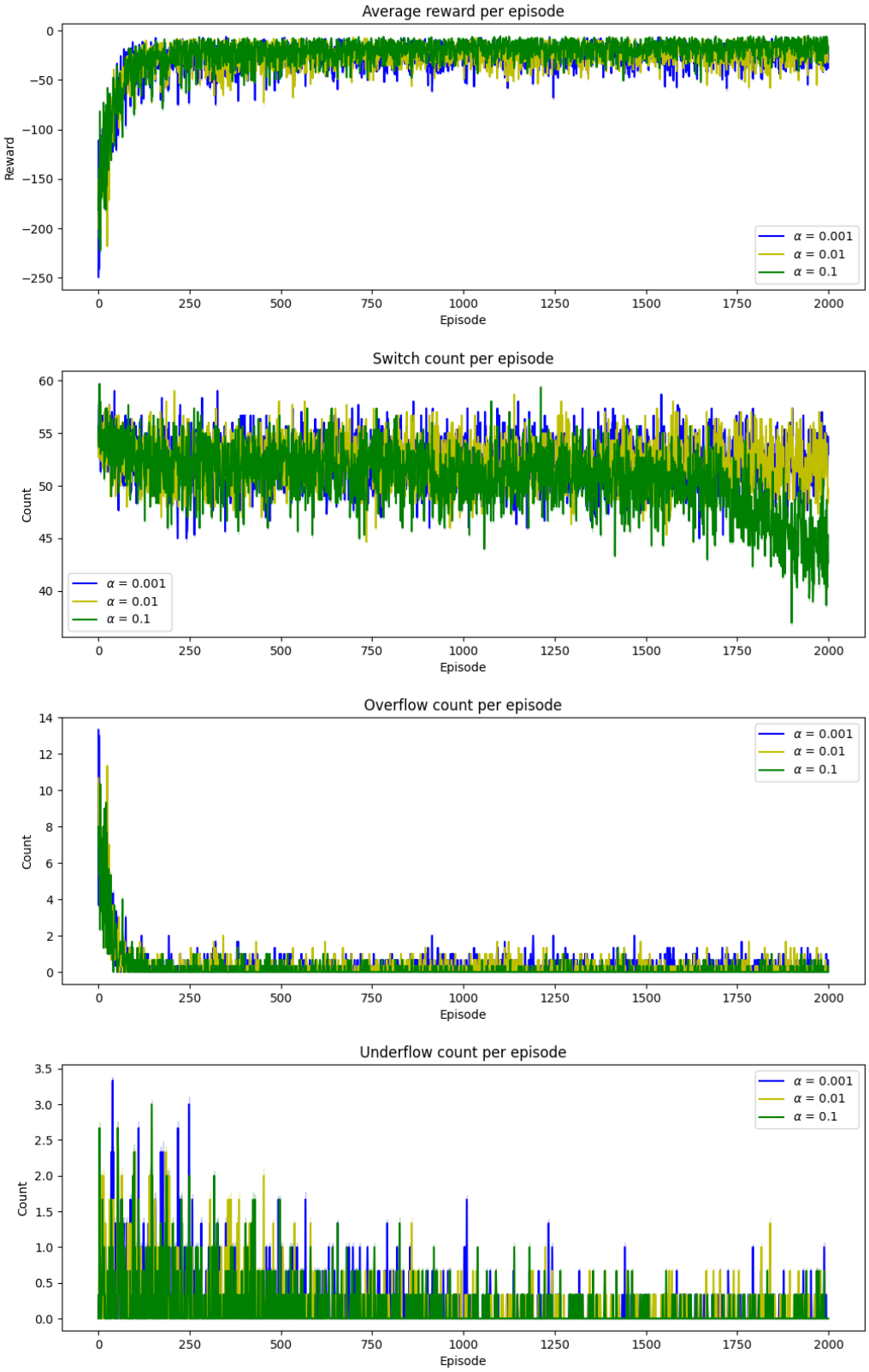


Figure 8: Approach1 - Q-learning with 2-dimensional Q-table: Training performance with different values of learning rate $\alpha = 0.001, 0.01, 0.1$ with exploration rate $\epsilon = 0.1$ and discount factor $\gamma = 0.85$

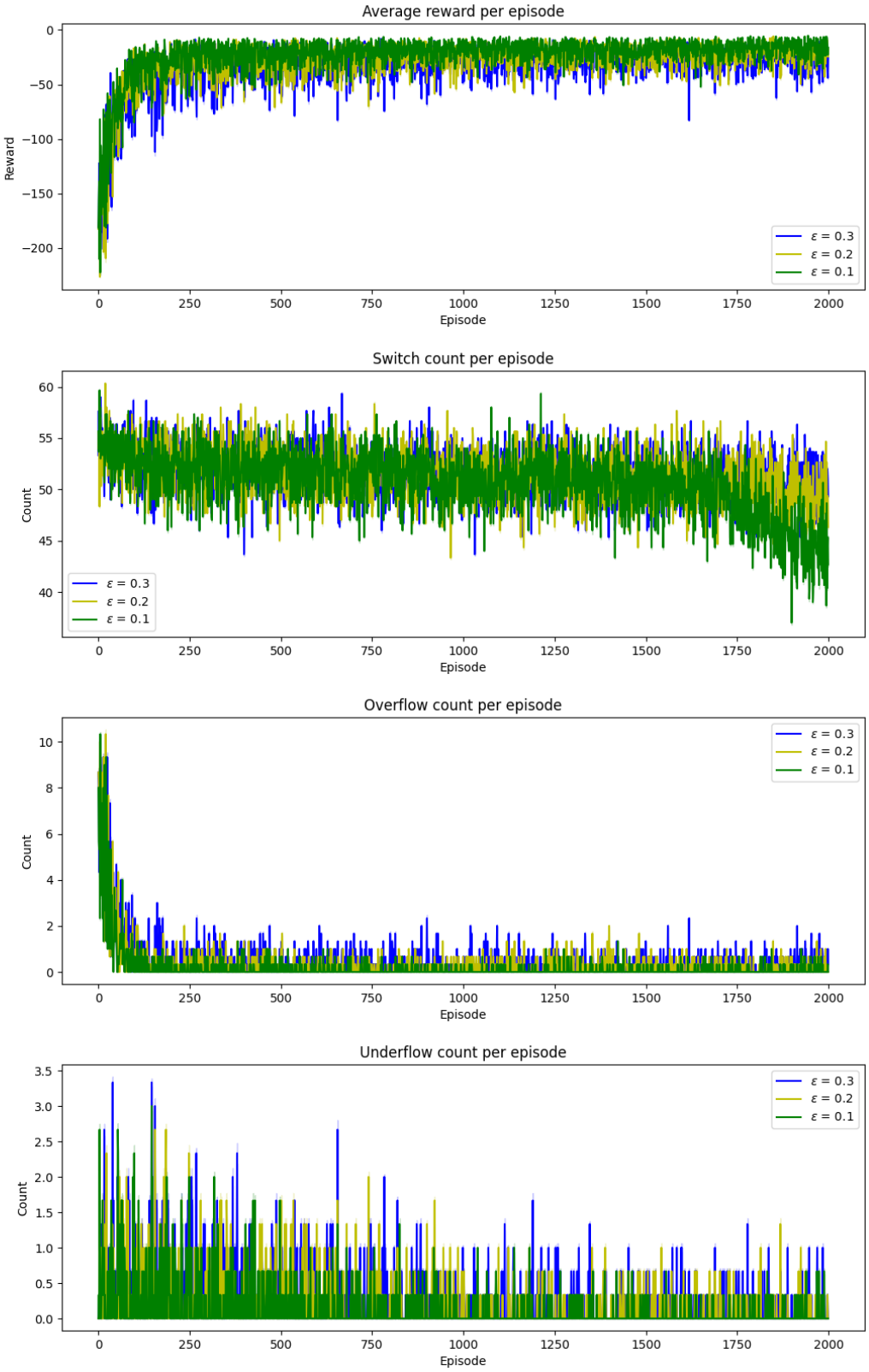


Figure 9: Approach1 - Q-learning with 2-dimensional Q-table: Training performance with different values of exploration rate $\epsilon = 0.1, 0.2, 0.3$ with learning rate $\alpha = 0.1$ and discount factor $\gamma = 0.85$

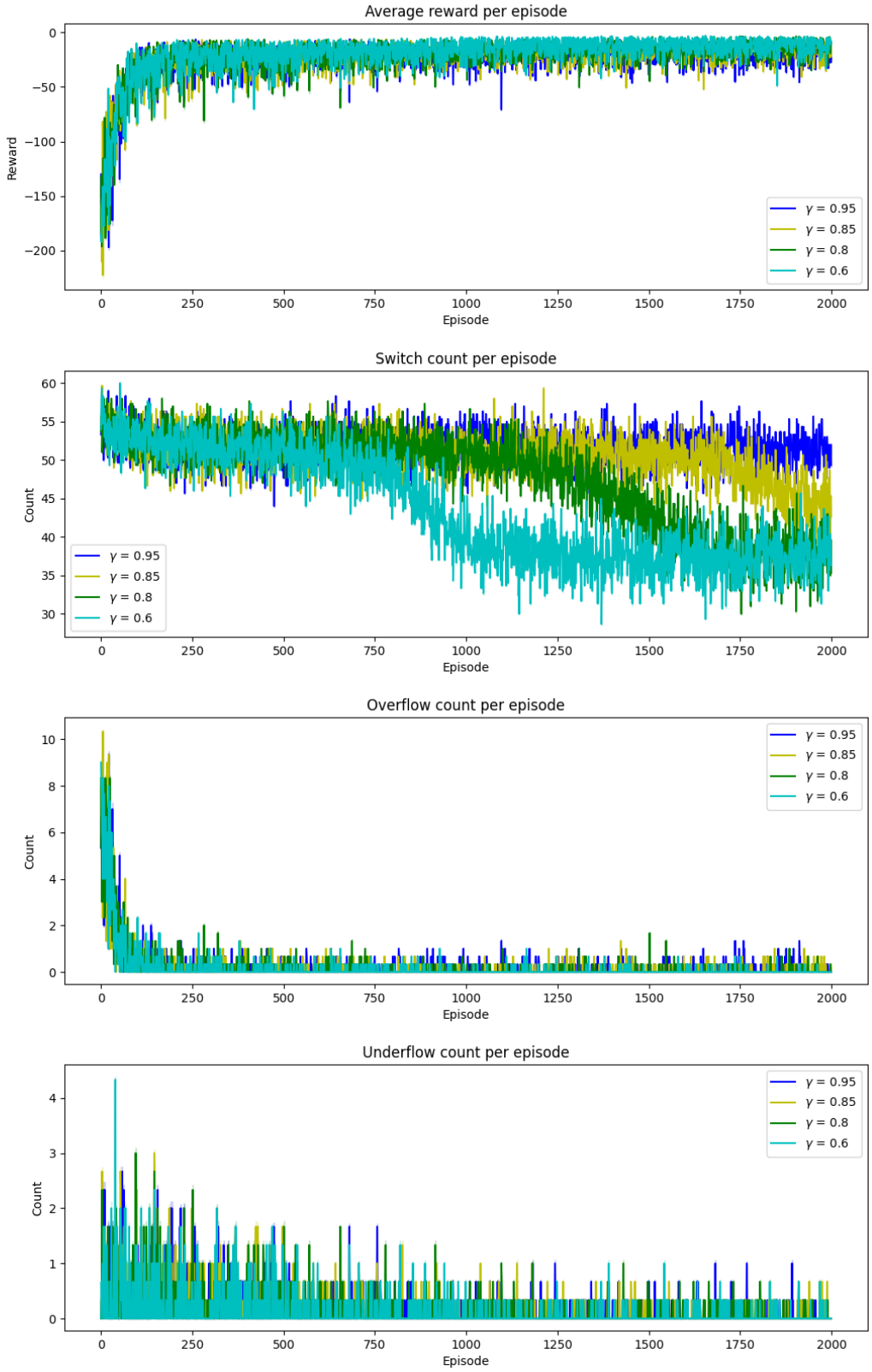


Figure 10: Approach1 - Q-learning with 2-dimensional Q-table: Training performance with different values of discount factor $\gamma = 0.95, 0.85, 0.8, 0.6$ with exploration rate $\epsilon = 0.1$ and learning rate $\alpha = 0.1$

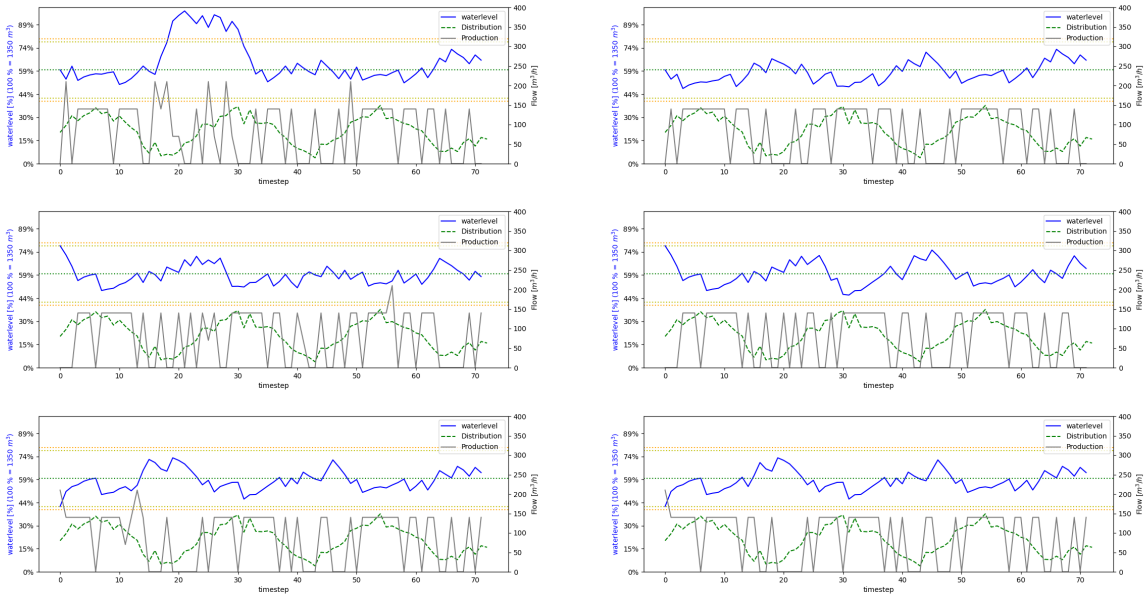


Figure 11: Approach1 - Q-learning with 2-dimensional Q-table ($\alpha = 0.1, \epsilon = 0.1, \gamma = 0.6$): Results of three different test scenarios where initial reservoir water level is average, critical maximum and critical minimum (top, middle and bottom respectively). **Left:** exploration strategy is enabled for testing, **Right:** exploration strategy is disabled for testing, only exploitation possible. Each plot has two Y-axis representing waterlevel in percentage on left-axis and Flow (production intake, distribution) on right-axis against timestep on X-axis

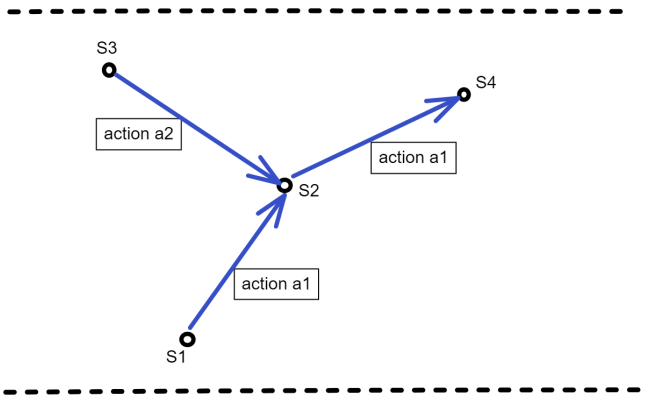


Figure 12: An example mentioning why 2d Q-table didn't work

The frequent action switch is happening in desired water level range because the agent does not have the historical data with it mentioning that in a given state which direction it should be moving forward. This implies that a state-action pair in the desired range can have multiple reward values based on if the previous action was the same or different since we have a negative reward for the action switch, an example of how a state-action pair can have two reward values is shown in Figure 12. Here for (s2, a1) the reward can be either 1 (if previous action was a1) or 1-10=-9 (if previous action was a2) based on the action taken in the last timestep. This reward value will affect the q-value for a state-action pair. Since after each step, q-value is updated meaning the q-value will keep switching between two values as well and the agent will determine the current action based on the latest updates of q-value for the pair, this leads to frequent action switches. Hence, this issue could be resolved if we were to provide the agent with additional information such as previous action along with current action in q-table making it a triplet *(state, previous action, action)* instead of a *(state, action)* pair.

This leads us to the second configuration for Approach1 mentioned in section 4.5.1. Similar to first configuration for Approach1, we experimented with several values of learning rate as shown in Figure 13, 14 and 15. It was seen that the learning rate $\alpha = 0.1$ performed best among $\alpha \in [0.001, 0.01, 0.1]$ where the average reward per episode plot has the least fluctuations and highest rewards throughout the training period. For exploration rate $\epsilon = 0.1$ took the lead among $\epsilon \in [0.1, 0.2, 0.3]$. And finally, for discount factor $\gamma \in [0.6, 0.8]$ both perform almost at the same level returning the highest average reward during training with minimum fluctuations in the plots.

Furthermore, we also performed a grid search hyperparameter optimization for this configuration of Q-learning and the results are given in Table 4. To measure the performance of each combination, similar to last configuration we measured the average cumulative reward per episode per last 100 episodes along with the other quantifiable variables mentioned previously, the combinations given in this table are the ones those provide the highest reward. The table contains the top five hyperparameter combinations with the best reward of -17.4 and switch count of 22.4. We see in Table 3 that the highest reward was -12.3 with a switch count of 37.6. If we only compare the average training reward of the two configurations for Approach1

then first setup clearly outperforms from reinforcement learning point of view. However, one of the requirement of this problem is to have an acceptable number of action switches implying that the action switches should not be frequent. This was the reason lead us to the second setup. The second setup does return lesser average reward but it has drastically reduce the number of action switch as well where the difference in reward is not that significant.

Rank	α	ϵ	γ	**Reward**	**Switch count**	**Overflow count**	**Underflow count**
1	0.1	0.1	0.60	-17.410324	22.413333	0.120000	0.120000
2	0.1	0.1	0.80	-17.480417	23.583333	0.113333	0.130000
3	0.1	0.1	0.85	-17.594769	24.723333	0.146667	0.096667
4	0.1	0.1	0.95	-17.875556	25.556667	0.110000	0.140000
5	0.001	0.1	0.85	-20.079028	20.850000	0.210000	0.120000

Table 4: Approach1 - Q-learning with 3-dimensional Q-table: Training results of configurations with highest average cumulative reward per last 100 episodes

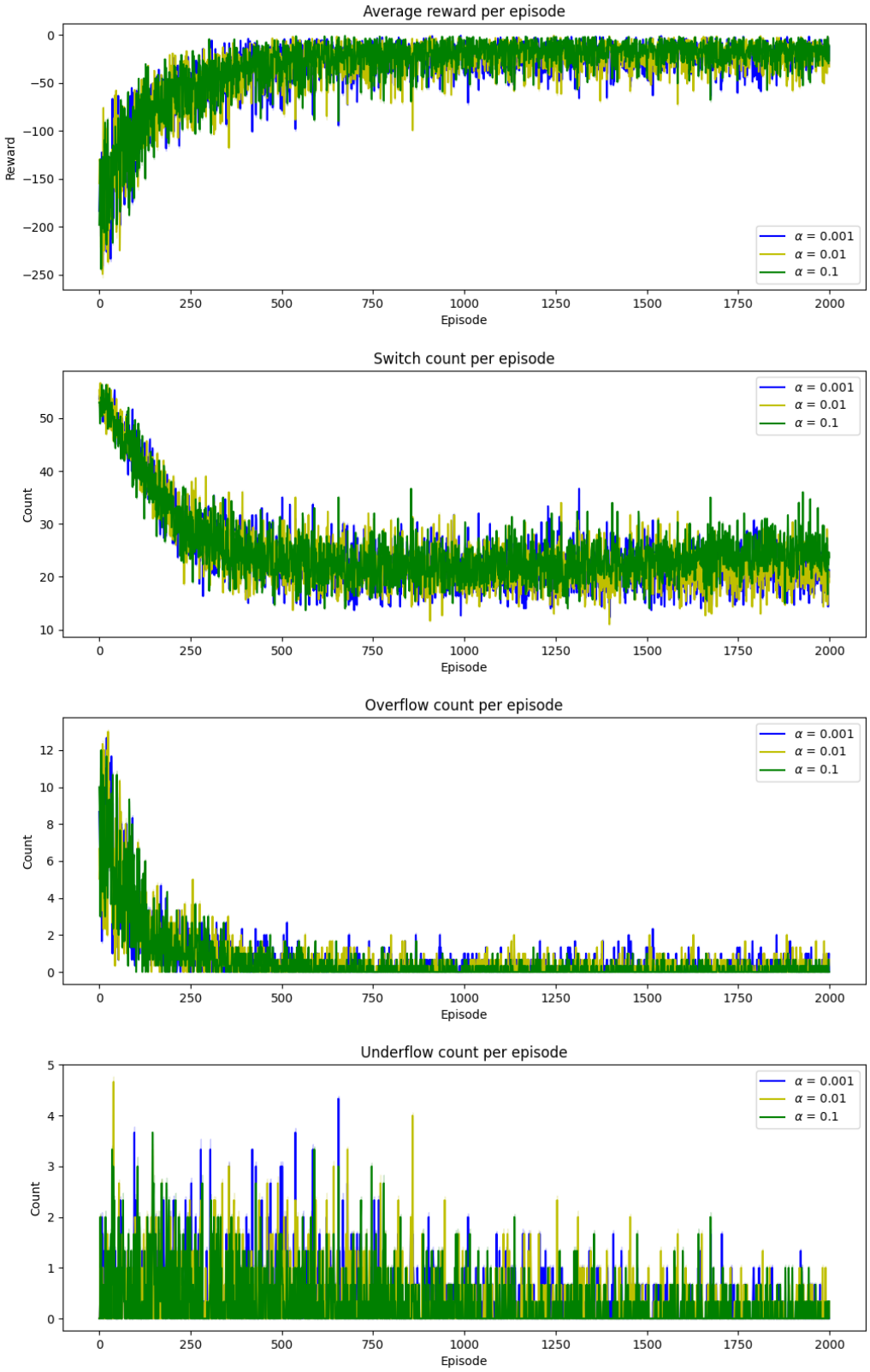


Figure 13: Approach1 - Q-learning with 3-dimensional Q-table: Training performance with different values of learning rate $\alpha = 0.001, 0.01, 0.1$ with exploration rate $\epsilon = 0.1$ and discount factor $\gamma = 0.85$

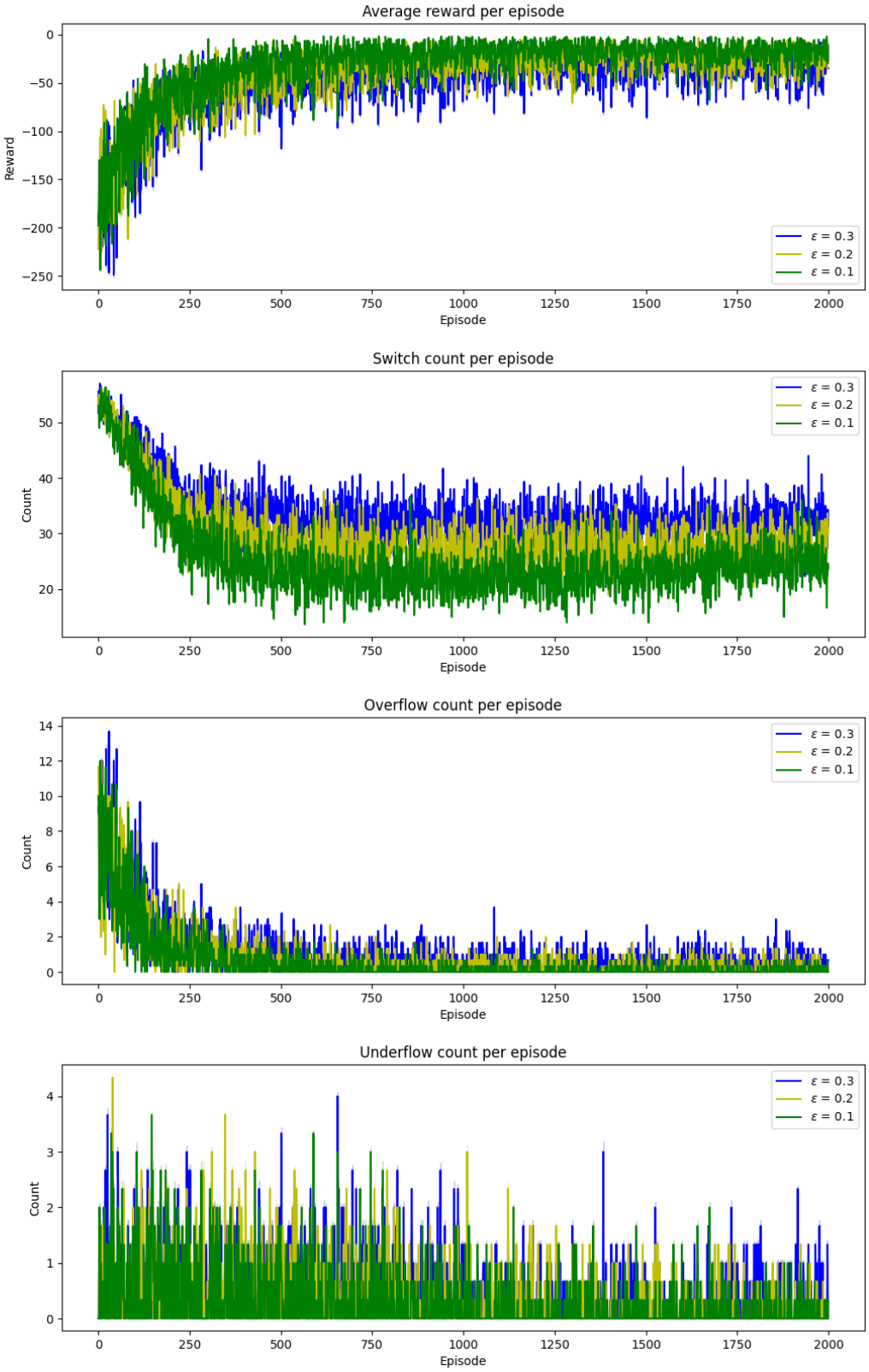


Figure 14: Approach1 - Q-learning with 3-dimensional Q-table: Training performance with different values of exploration rate $\epsilon = 0.1, 0.2, 0.3$ with learning rate $\alpha = 0.1$ and discount factor $\gamma = 0.85$

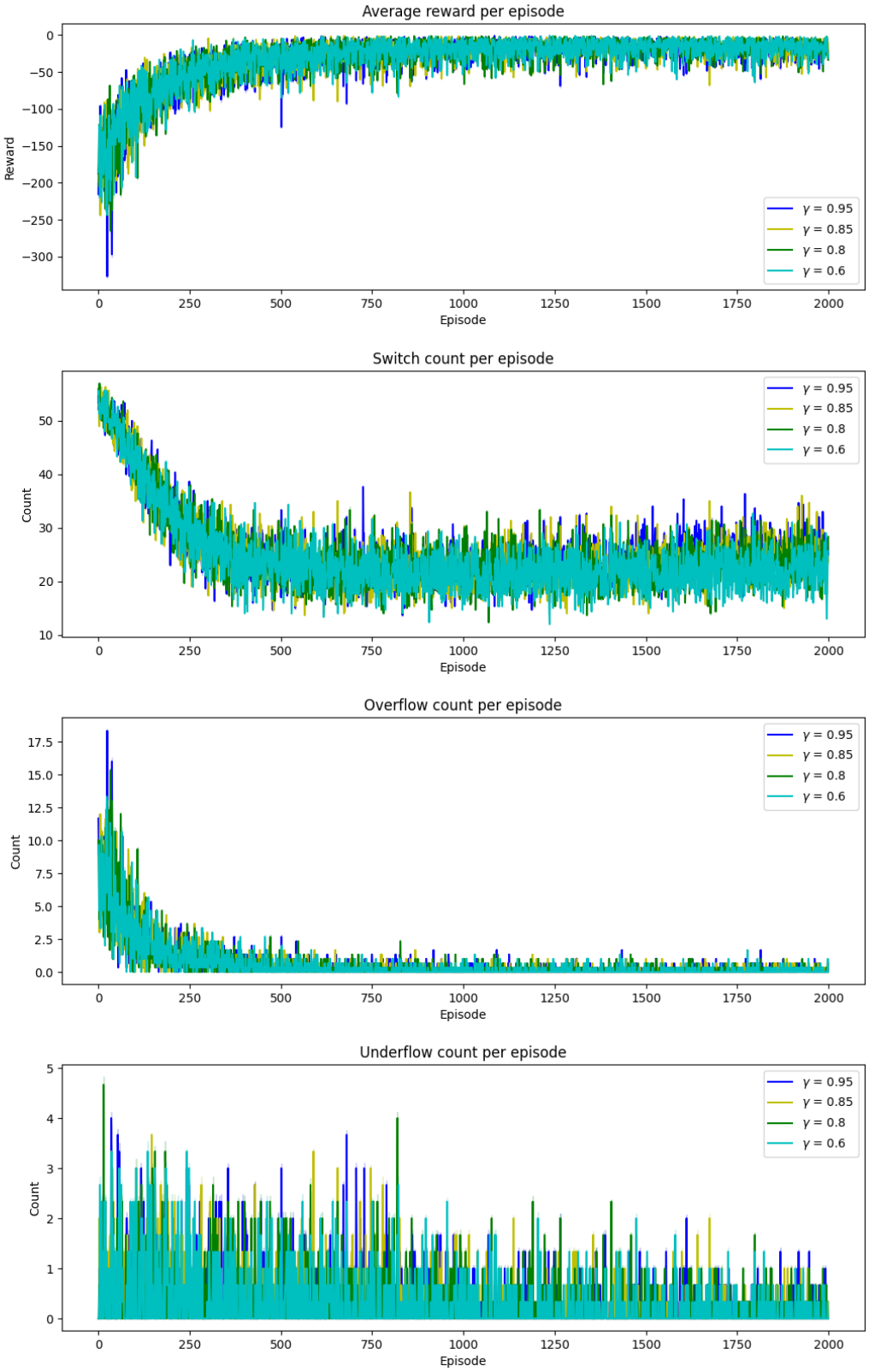


Figure 15: Approach1 - Q-learning with 3-dimensional Q-table: Training performance with different values of discount factor $\gamma = 0.95, 0.85, 0.8, 0.6$ with exploration rate $\epsilon = 0.1$ and learning rate $\alpha = 0.1$

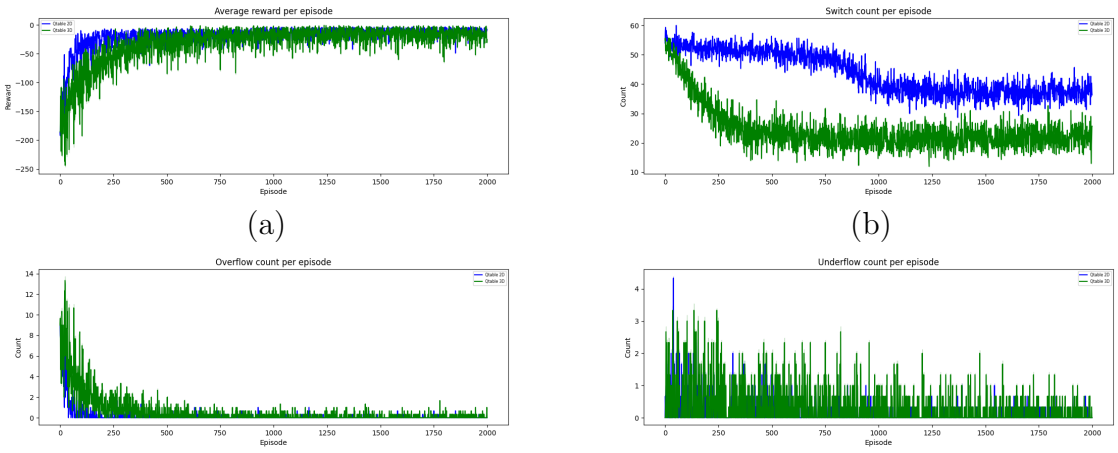


Figure 16: Approach1 - Training results for Q-learning with 2-dimensional Q-table with ($\alpha = 0.1, \epsilon = 0.1, \gamma 0.6$) vs 3-dimensional Q-table with ($\alpha = 0.1, \epsilon = 0.1, \gamma = 0.6$) **(a)** Average reward per episode during training, **(b)** Action change count per episode during training **(c)** Overflow count per episode during training **(d)** Underflow count per episode during training

Figure 16 describes the training results for both configurations with their best hyperparameter combinations for Approach1. The average reward per episode during training reaches almost at the same level when training is complete, the plots for both configurations looks almost similar except in the initial stages of training where 2d q-table configuration takes the lead. This is because there are lesser possible combination to explore compared to 3d q-table and system requires more time to explore them and learn. Similarly, the plots for overflow count and underflow count for both configurations also look similar and follow same learning pattern where the number of overflow and underflow reduces over time. The main difference between these two configurations is seen in plot for the switch count, where it is clearly seen that Approach2 (3d q-table) outperforms Approach1 (2d q-table) and switch count reduces drastically as the time passes and dropping switch count from around 60 to 20 from the beginning of training to the end. Additionally, it was also noticed that the number of overflow and underflow were increased slightly for this setup in comparison to previous setup, it is because there is a trade-off between number of switch vs water level in the reservoir.

Similar results were also found in testing as well as showcased in Figure 17. The results for testing resemble the results for 2d q-table results shown in Figure 11 except the fact that the number of action switch in an episode is far less for 3d q-table configuration displaying that a pump intake setting stays put for multiple timesteps meaning a significant amount of time, making the production water intake pattern more realistic and practical.

The results discussed above provide evidence to prove that a Q-learning agent is able to learn the optimum policy where water level in the reservoir satisfies the boundary conditions with an acceptable number of production pump intake changes. However, it can be seen in test results provided that the current agent is not able to foresee and schedule/plan ahead

based on the upcoming distribution/consumption in near future. To tackle this drawback of the current system, Approach2 using Q-learning was implemented.

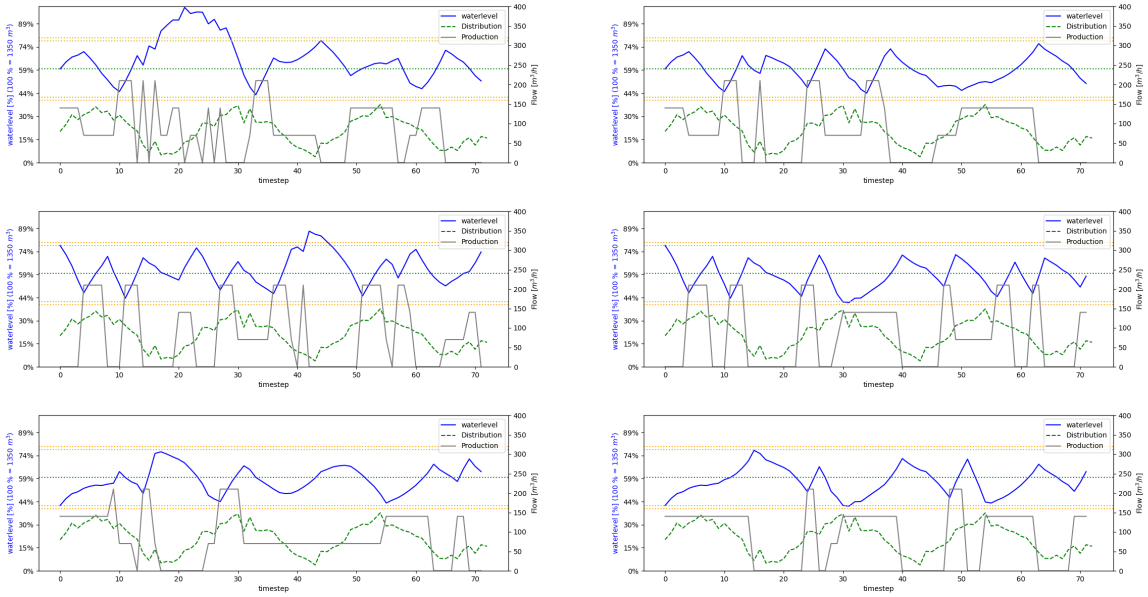


Figure 17: Approach1 - Q-learning with 3-dimensional Q-table: Results of three different test scenarios where initial reservoir water level is average, critical maximum and critical minimum (top, middle and bottom respectively). **Left:** exploration strategy is enabled for testing, **Right:** exploration strategy is disabled for testing, only exploitation possible. Each plot has two Y-axis representing waterlevel in percentage on left-axis and Flow (production intake, distribution) on right-axis against timestep on X-axis

5.1.1.2 Approach2

Similar to Approach1, we experimented with various values of learning rate, exploration rate and discount factor to see how each hyperparameter affects the performance and results of which are shown in Figure 18, 19 and 20. It was seen that the $\alpha = 0.1$, $\epsilon = 0.1$ and $\gamma = 0.6$ performs best among their respective other values. The behavior of Approach2 is quite similar to the behavior of Approach1 and this is because they both use the same algorithm with a change in the environment definition.

Furthermore, we also performed a grid search hyperparameter optimization where $\alpha \in [0.001., 0.01, 0.1]$, $\epsilon \in [0.1, 0.2, 0.3]$ and $\gamma \in [0.95, 0.85, 0.8, 0.6]$. Table 5 contains the five hyperparameter combinations that provide the best performance. To measure the performance of each combination, we measured the average cumulative reward per episode per last 100 episodes along with other quantifiable variables mentioned previously, the configuration given in this table are the ones that provides the highest reward. The is seen that the highest average reward achieved for this approach is -19.17 which is lower than both the implementations present for Approach1. This implies that the agent has not been able to learn the optimal policy properly. The same can also be seen in Figure 21 where it is shown in plots that system

does perform worst among all three configurations and this comparison is done using most optimised hyperparameter configuration of all three models.

Rank	α	ϵ	γ	**Reward**	**Switch count**	**Overflow count**	**Underflow count**
1	0.1	0.1	0.60	-19.176157	43.233333	0.120000	0.080000
2	0.1	0.1	0.85	-20.089213	46.850000	0.150000	0.086667
3	0.1	0.1	0.80	-21.120648	46.463333	0.136667	0.133333
4	0.1	0.1	0.95	-22.735278	48.420000	0.176667	0.160000
5	0.01	0.1	0.85	-24.859491	48.626667	0.246667	0.090000

Table 5: Approach2 (three-parameter in environment state space) - Q-learning with 2-dimensional Q-table: Training results of configurations with highest average cumulative reward per last 100 episodes

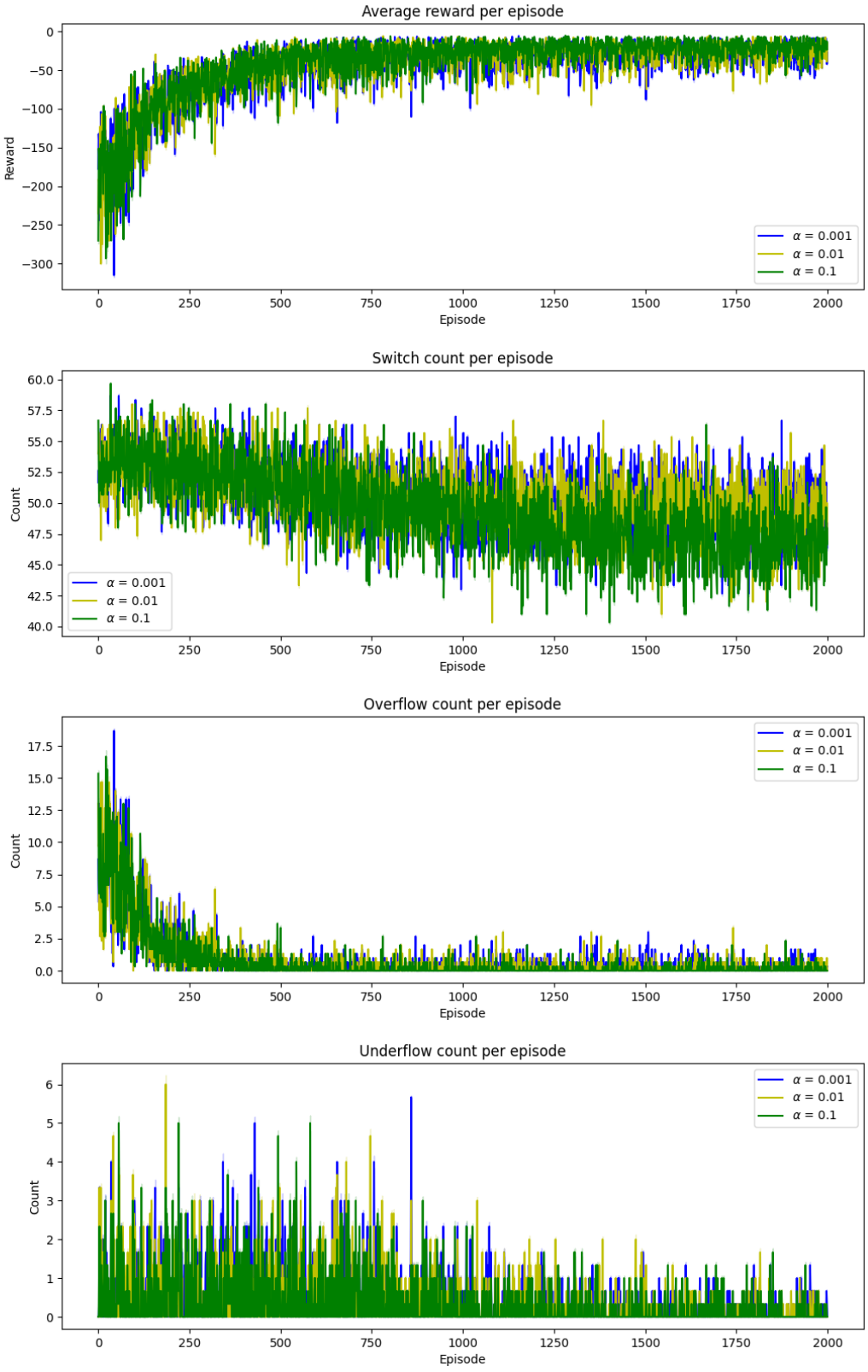


Figure 18: Approach2 - Q-learning with state space contains (waterlevelbucket, hour, day): Training performance with different values of learning rate $\alpha = 0.001, 0.01, 0.1$ with exploration rate $\epsilon = 0.1$ and discount factor $\gamma = 0.85$

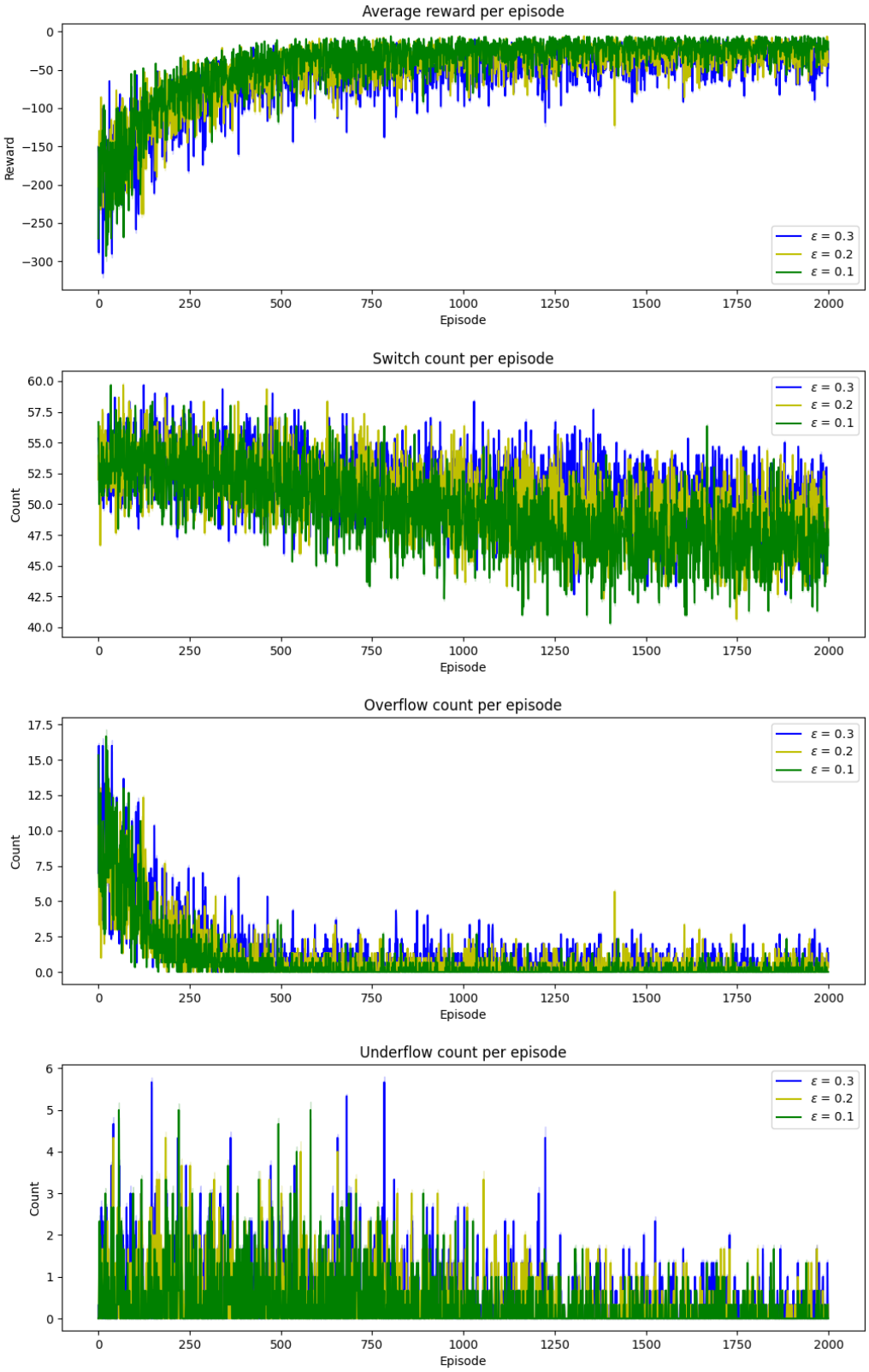


Figure 19: Approach2 - Q-learning with state space contains (waterlevelbucket, hour, day): Training performance with different values of exploration rate $\epsilon = 0.1, 0.2, 0.3$ with learning rate $\alpha = 0.1$ and discount factor $\gamma = 0.85$

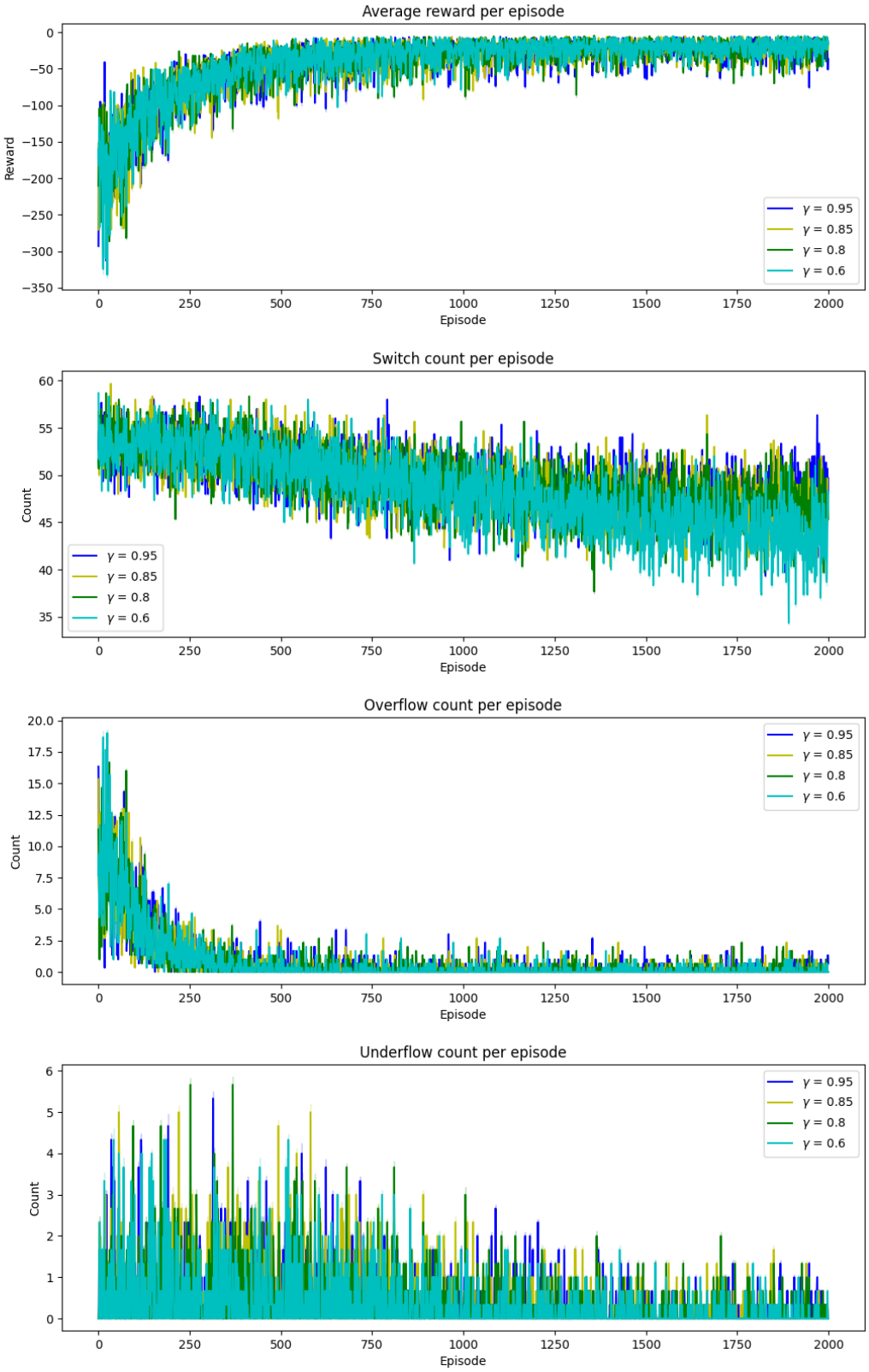


Figure 20: Approach2 - Q-learning with state space contains (waterlevelbucket, hour, day): Training performance with different values of discount factor $\gamma = 0.95, 0.85, 0.8, 0.6$ with exploration rate $\epsilon = 0.1$ and learning rate $\alpha = 0.1$

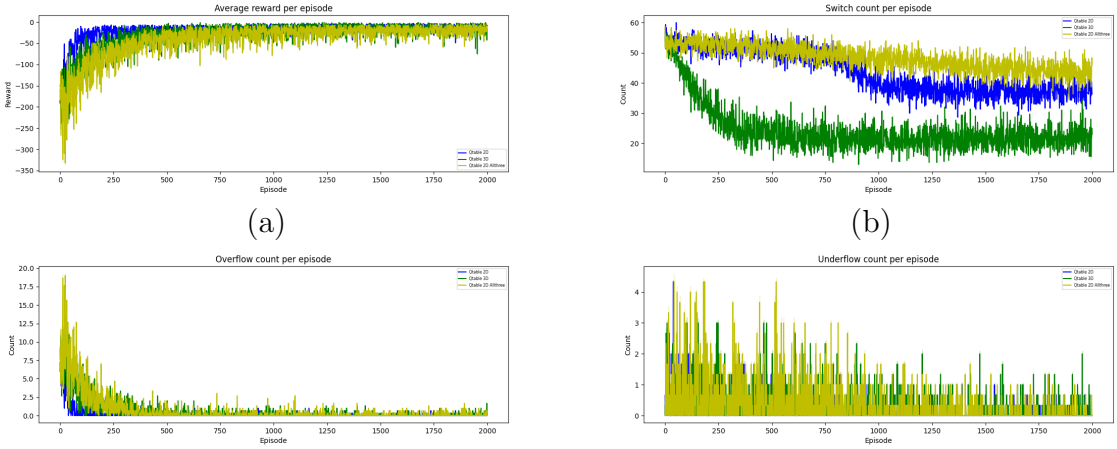


Figure 21: Approach2 - ($\alpha = 0.1, \epsilon = 0.1, \gamma = 0.6$) vs Approach1 - Training results for Q-learning with 2-dimensional Q-table with ($\alpha = 0.1, \epsilon = 0.1, \gamma 0.6$) vs 3-dimensional Q-table with ($\alpha = 0.1, \epsilon = 0.1, \gamma = 0.6$) **(a)** Average reward per episode during training, **(b)** Action change count per episode during training **(c)** Overflow count per episode during training **(d)** Underflow count per episode during training

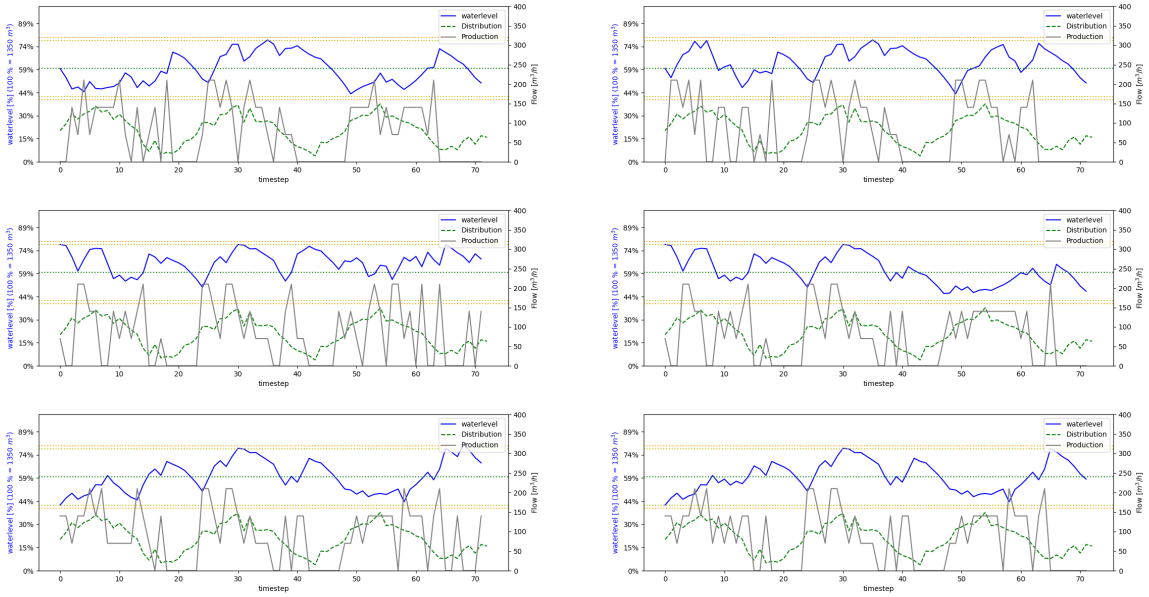


Figure 22: Approach2 - Q-learning with state space contains (waterlevelbucket, hour, day): Results of three different test scenarios where initial reservoir water level is average, critical maximum and critical minimum (top, middle and bottom respectively). **Left:** exploration strategy is enabled for testing, **Right:** exploration strategy is disabled for testing, only exploitation possible. Each plot has two Y-axis representing waterlevel in percentage on left-axis and Flow (production intake, distribution) on right-axis against timestep on X-axis

We implemented Approach2 to overcome Approach1's drawbacks such as its inability to foresee and schedule/plan ahead based on the upcoming distribution/consumption in the near future. However, we see in the test scenario results in Figure 22 that the agent is not able to schedule in advance based on the upcoming consumption.

5.1.2 N-Step

To overcome the limitations of Approach 1 and 2 which were implemented using Q-learning, we decided to implement n-step because it involves bootstrapping from an estimate of the value function of the next n steps implying next n hours since each timestep represents an hour in this problem.

Similar to all the other setups prior to this, we also observed effect of individual hyperparameter on the N-step's performance which is shown in Figures 23, 24 and 25. Additionally, we experimented with various hyperparameters to find the best possible combination of hyperparameters which are given in Table 6. The highest average reward achieved was -12.5 with a switch count of 27, overflow count of 0.05 and underflow count of 0.12 per 100 episodes. Hence, this system is able to perform better compared to all the previous setups since its reward value is close to Q-learning Approach1's first configuration and switch count is close to Approach1's second configuration implying that it is able to find the balance between switch and overflow/underflow count trade-off.

Figure 26 shows how the trained agent behaves in specific test scenarios. We see that the water level of the reservoir is able to stay within the bounds with slightly more frequent action changes in comparison to Q-learning shown in Figure 17. This is because N-step considers future rewards while calculating and updating the q-value for a state-action pair.

Additionally, various values of n for the algorithm were tried and their performance was analyzed. Figure 27 depicts the training performance across time and displays how the agent with different values of n learns. We notice that the plot for average training reward per episode looks almost the same for all the values of n with slightly slower learning as the value of n increases, this is because the higher the value of n the more future steps needs to be taken into account to calculate the q-value and that results in increasing the learning time slightly. For the switch count plot, there is a significant difference between $n = 1$ and other values of n. The final values of switch count are minimum when $n = 1$ as shown in the plot. However, the average reward per episode is lower for higher values of n. This is because that the number of overflow and underflow is smaller for higher values of n and that is logical because the agent is able to consider a higher number of steps in advance resulting in lesser overflow/underflow count, the same can be seen in bottom two plot of the figure. Furthermore, Table 7 contains the training performance as an average over the last 100 episodes. Here, $n = 48$ provided the highest average reward with minimum overflow/underflow on average. However, there isn't a significant difference in overall performance between various values of n. The same can also be seen in Figure 28, the figure showcases the actions taken by the learned agent during testing in a specific test scenario, these plots are just one instance of these trained agents and results might vary slightly for other instances. The plots are not conclusive enough to determine which value of n would be better. However, it is seen that the higher value n facilitates a smaller

jump between actions.

Rank	α	ϵ	γ	**Reward**	**Switch count**	**Overflow count**	**Underflow count**
1	0.1	0.1	0.60	-12.509491	27.226667	0.066667	0.120000
2	0.1	0.1	0.80	-16.566852	38.023333	0.110000	0.173333
3	0.1	0.2	0.60	-18.701157	33.696667	0.170000	0.160000
4	0.1	0.1	0.85	-19.213380	41.566667	0.196667	0.170000
5	0.1	0.2	0.80	-22.699028	43.146667	0.150000	0.273333

Table 6: N-step with 3-dimensional Q-table with n=5: Training results of configurations with highest average cumulative reward per last 100 episodes

Rank	n	**Reward**	**Switch count**	**Overflow count**	**Underflow count**
1	48	-13.434259	29.336667	0.060000	0.180000
2	10	-15.013426	29.550000	0.126667	0.166667
3	24	-15.823889	29.253333	0.116667	0.273333
4	72	-16.434352	29.513333	0.140000	0.253333
5	5	-16.672731	27.510000	0.166667	0.310000
6	1	-18.937778	21.983333	0.176667	0.193333

Table 7: N-step with 3-dimensional Q-table with different values of $n = 1, 5, 10, 24, 48, 72$: Training results of configurations with highest average cumulative reward per last 100 episodes

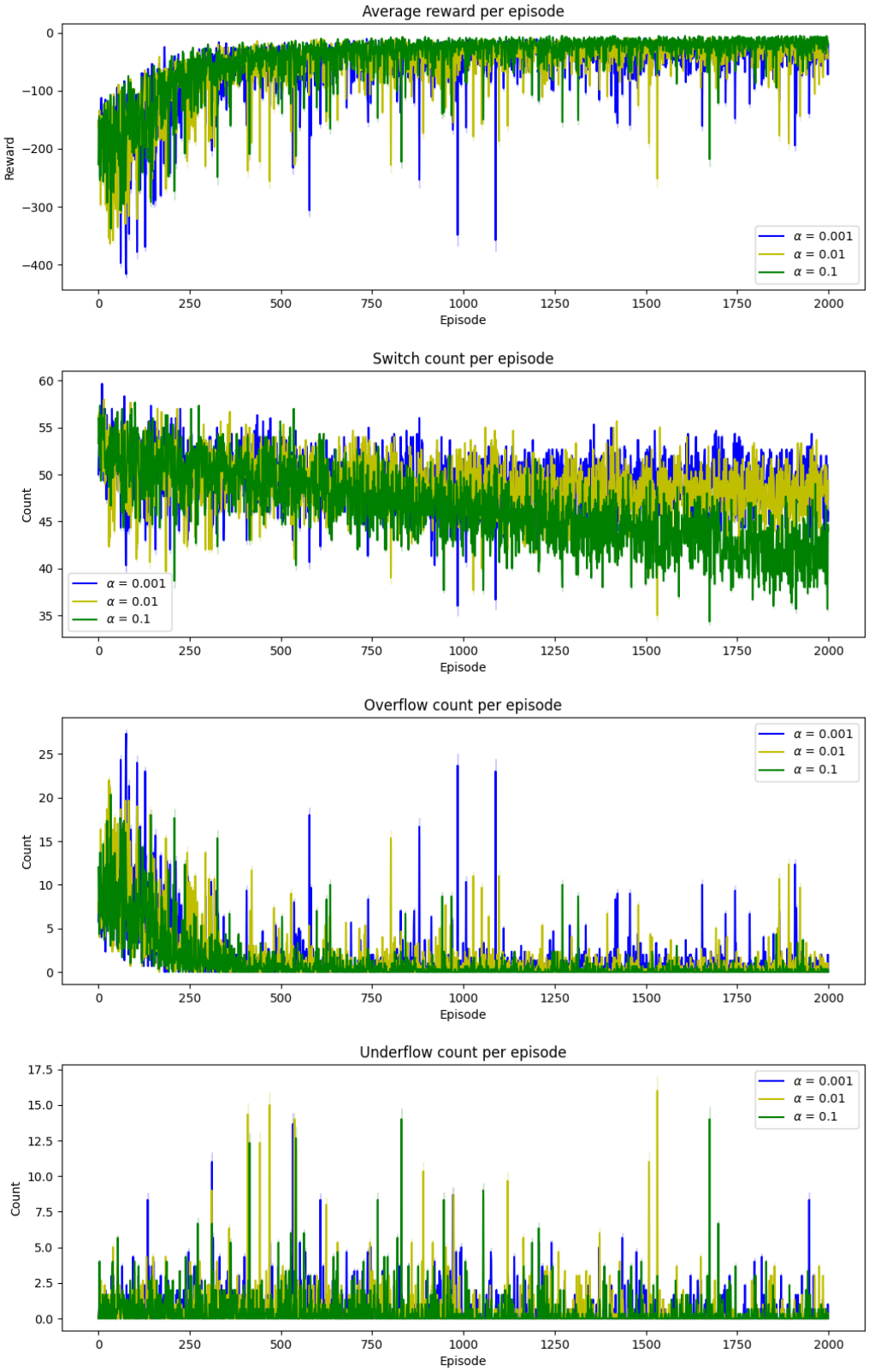


Figure 23: N-step with 3-dimensional Q-table with n=5: Training performance with different values of learning rate $\alpha = 0.001, 0.01, 0.1$ with exploration rate $\epsilon = 0.1$ and discount factor $\gamma = 0.85$

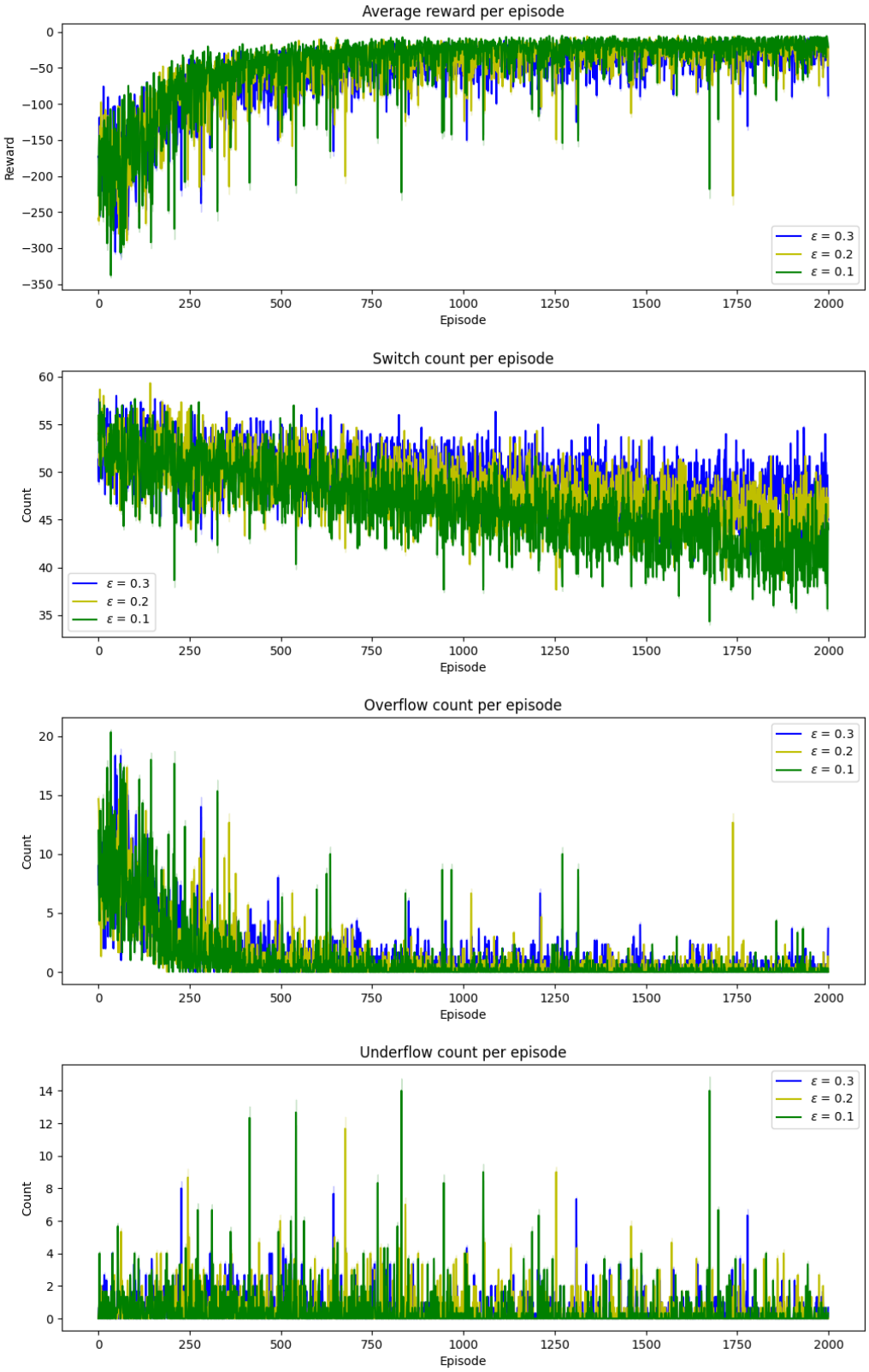


Figure 24: N-step with 3-dimensional Q-table with n=5: Training performance with different values of exploration rate $\epsilon = 0.1, 0.2, 0.3$ with learning rate $\alpha = 0.1$ and discount factor $\gamma = 0.85$

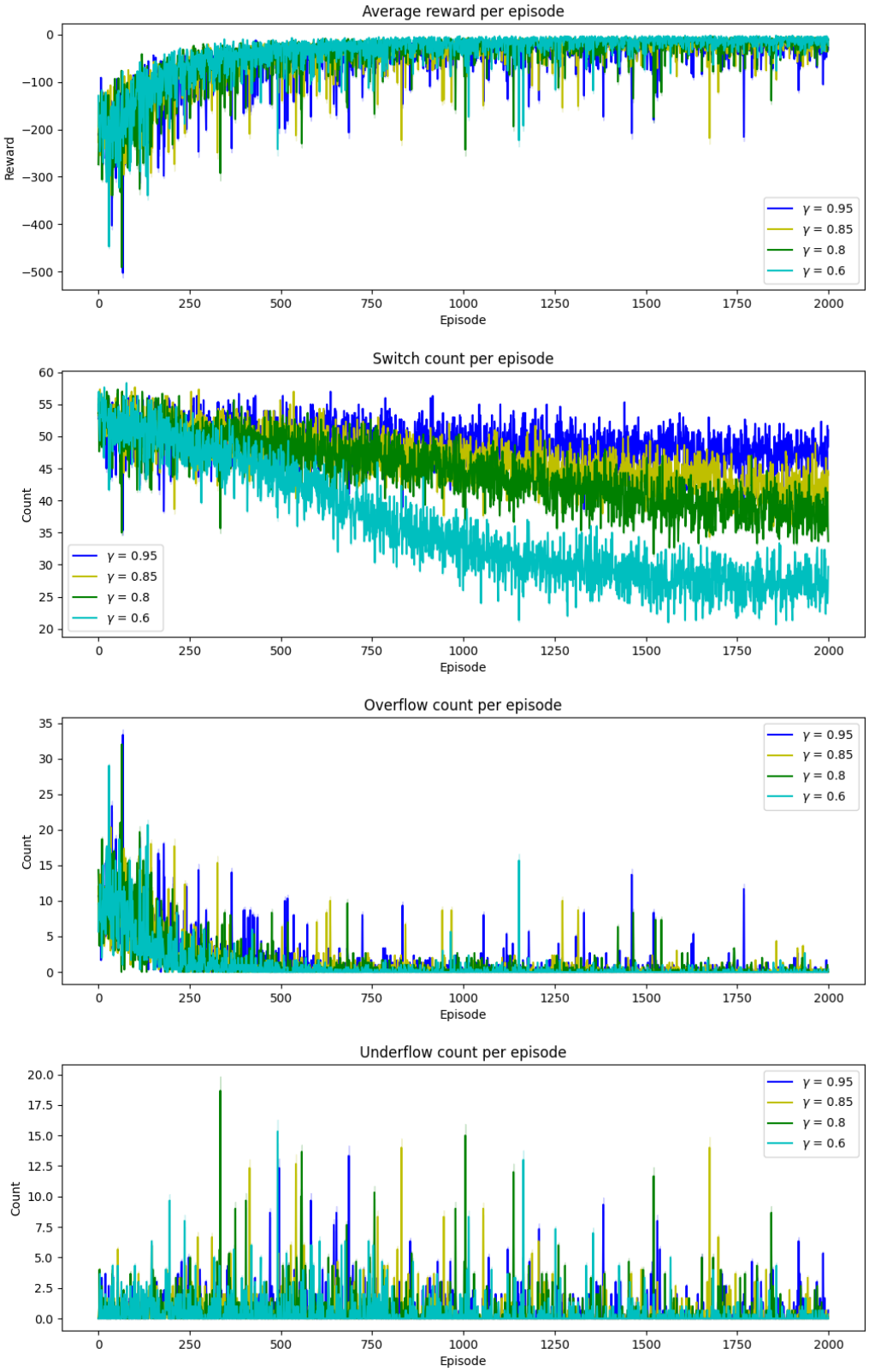


Figure 25: N-step with 3-dimensional Q-table with n=5: Training performance with different values of discount factor $\gamma = 0.95, 0.85, 0.8, 0.6$ with exploration rate $\epsilon = 0.1$ and learning rate $\alpha = 0.1$

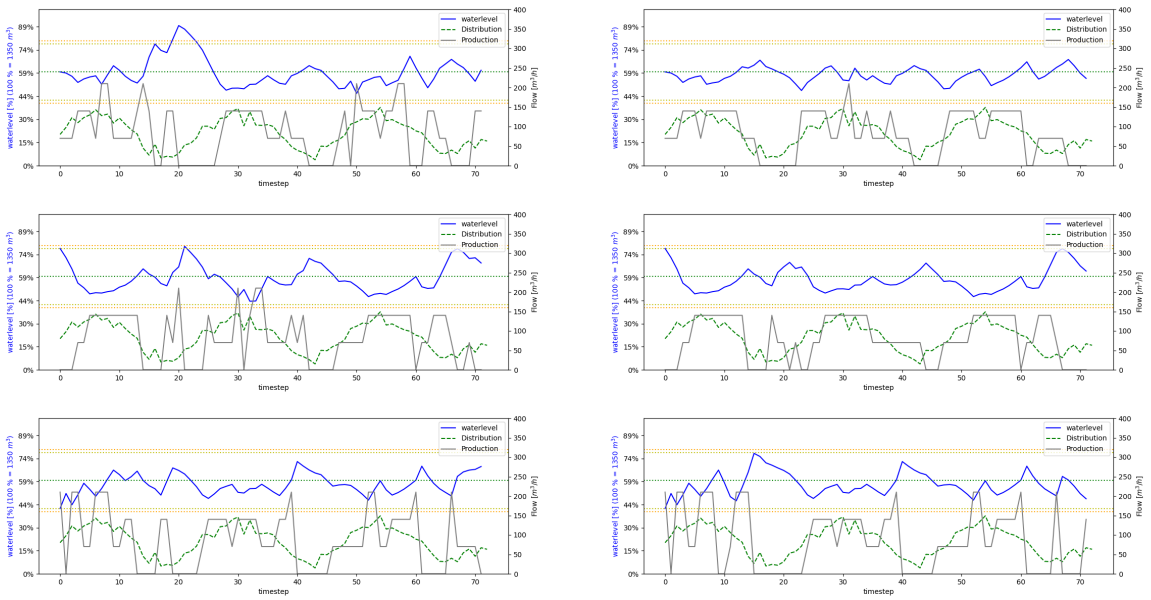


Figure 26: N-step with 3-d Q-table with n=5: Results of three different test scenarios where the initial reservoir water level is average, critical maximum and critical minimum (top, middle and bottom respectively). **Left:** exploration strategy is enabled for testing, **Right:** exploration strategy is disabled for testing, only exploitation possible. Each plot has two Y-axis representing water level in percentage on the left-axis and Flow (production intake, distribution) on the right-axis against timestep on X-axis

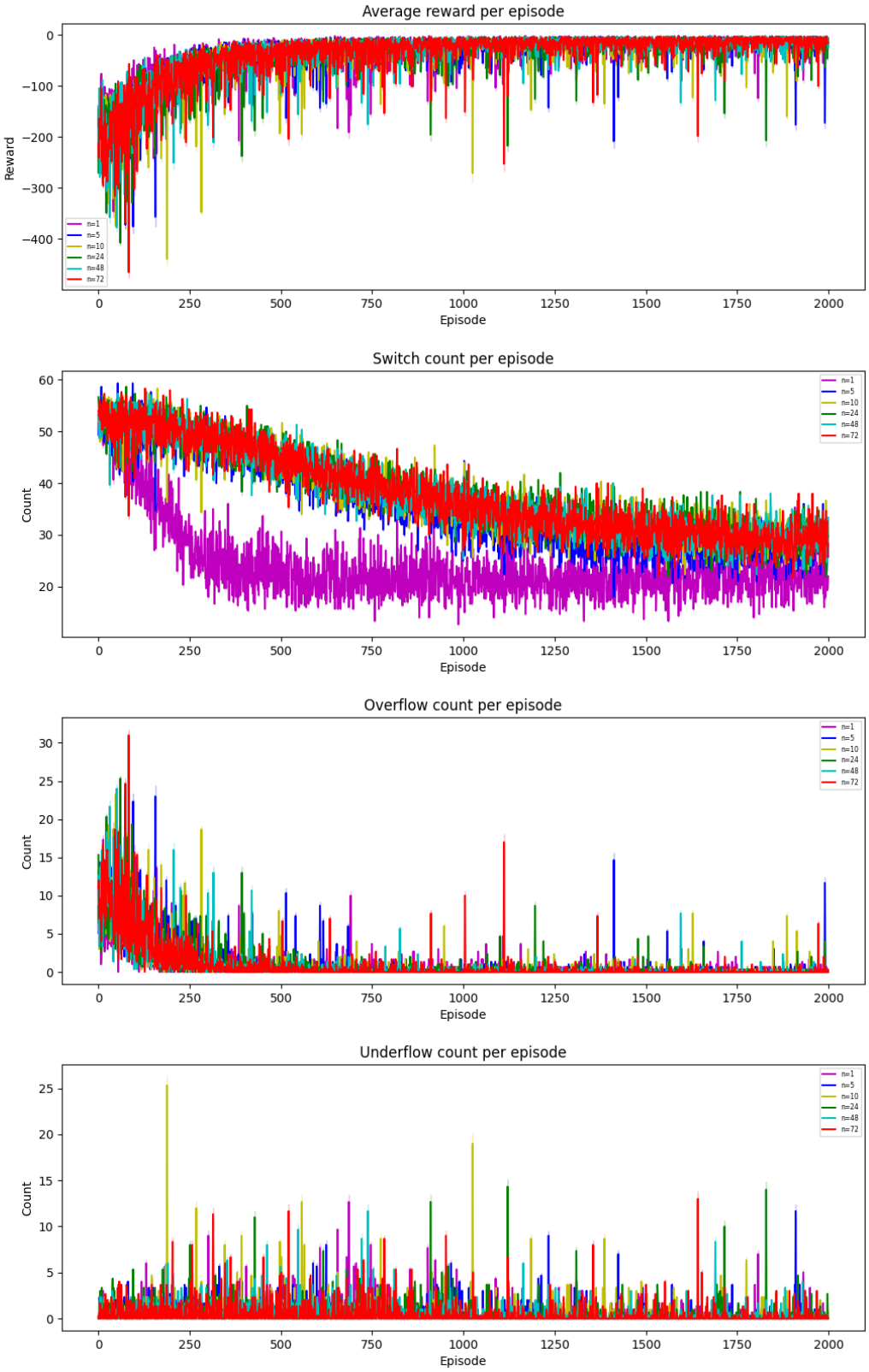


Figure 27: N-step with 3-dimensional Q-table with different values of $n = 1, 5, 10, 24, 48, 72$: Training performance with exploration rate $\epsilon = 0.1$, learning rate $\alpha = 0.1$ and discount factor $\gamma = 0.85$

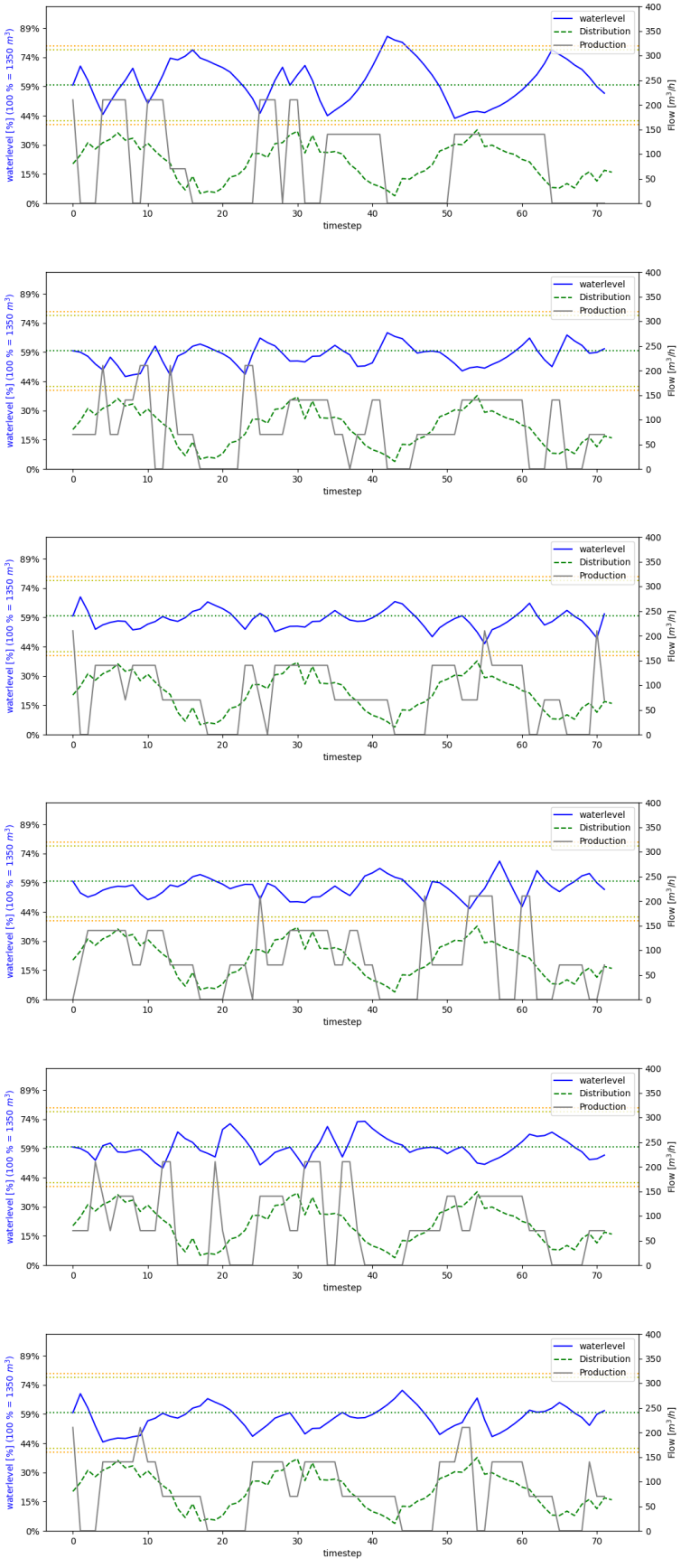


Figure 28: N-step with 3-d Q-table with different values of $n = 1, 5, 10, 24, 48, 72$, top to bottom respectively: Results of a test scenario where the initial reservoir water level is average and exploration strategy is disabled for testing, only exploitation possible. Each plot has two Y-axis representing water level in percentage on the left-axis and Flow (production intake, distribution) on the right-axis against timestep on X-axis

5.2 Real Data

Once all the experiments with artificial data were done and we determined the models providing the best results. The real data from Leersum was used to train and test for those models. As stated previously N-step algorithm and Q-learning Approach1 with 3-d Q-table were the top two configurations. Hence, we used them on real data to measure their performance. Table 8 contains the training performance for both agents. Here, the N-step algorithm performs worse than Q-learning overall as the average total reward is lower for N-step with a higher switch count, and overflow/underflow count. This is because the temporal difference is more sensitive in comparison to learning and the real data has more inconsistencies as the water demand changes over the course of a year.

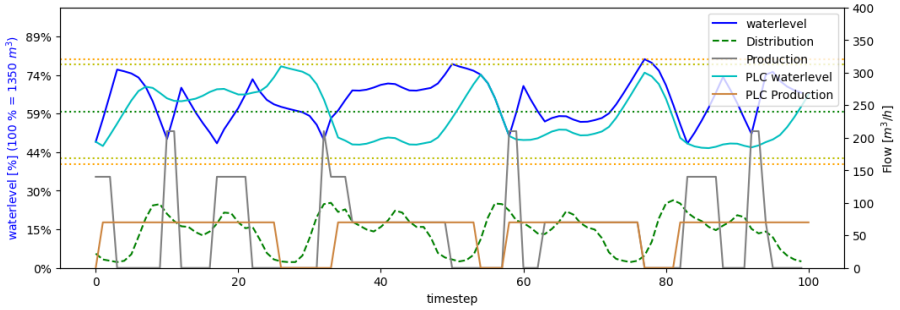
Rank	**Algorithm setup**	**Reward**	**Switch count**	**Overflow count**	**Underflow count**
1	qlearning3DQtable	-16.059	22.84	0.209	0.063
2	nstep3DQtable	-18.514	31.300	0.333	0.203

Table 8: N-step with 3-dimensional Q-table (n=5) vs Q-learning with 3-dimensional Q-table: Training results of configurations with highest average cumulative reward per last 100 episodes

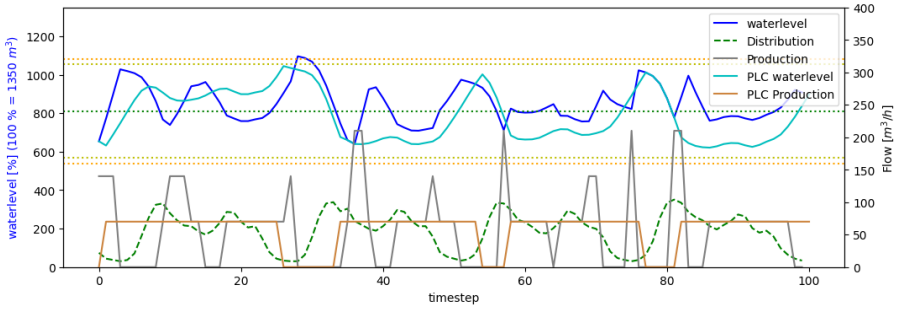
For quantification purposes in a hypothetical scenario, the agent should switch once in 10 hours and the water level should stay within the desired range. This leads to the total reward of $72 - 10 * 7 = 2$ for an episode meaning an average reward of 0.027 per episode on the scale of $[-1000, 1]$ where -1000 is when the reservoir is always overflowing or underflowing and 1 when the water level in the reservoir is in the desired range and there is no intake pump switch for the entire episode. Here, in Table 8 we see that RL agents have an average reward of -16.05 and -18.51.

Figure 29 visualizes actions taken by the trained agent for both the models and their comparison with the actions taken by a simple logic similar to the one found in Programmable Logic Controller (PLC) systems. We see here that RL agents are able to perform in such a way that the water level is within bounds similar to the PLC model. However, the switch count for intake set-points is higher in comparison to PLC model. This is because RL agents are sensitive to the current consumption value or any disturbances in the consumption and they try to navigate in the desired direction based on the changes or disturbances.

Similarly, we wanted to do a comparison study of actions taken by our RL agents with the OPIR system. However, the real OPIR system timestep size is 15 minutes so it wouldn't have been a fair comparison because an hour as a timestep was used for RL agents to provide more discretization to the system and provide easier learning for the agents. To provide a better understanding of OPIR, Figure 30 provides a visualization of OPIR flow with a timestep size of 15 minutes.



(a: Q-learning with 3-dimensional Q-table)



(b: N-step with 3-dimensional Q-table (n=5))

Figure 29: Trained agent used for testing with exploration strategy disabled: A comparison with PLC system. Each plot has two Y-axis representing water level in percentage on the left-axis and Flow (production intake, distribution) on the right-axis against timestep on X-axis

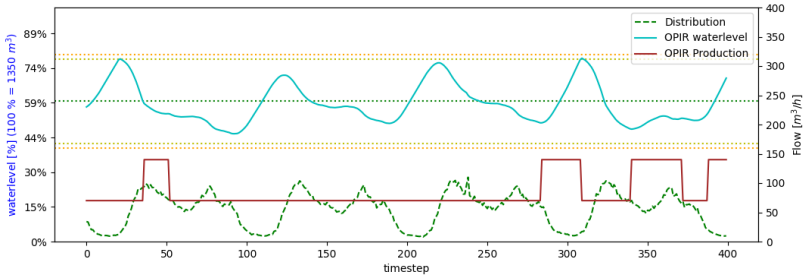


Figure 30: Real-data plot OPIR system: The water level and demand plot with 15 minutes timestep

5.3 Discussions

For each model we experimented with various hyperparameters, the most optimized models for both algorithms were with a learning rate of 0.1, an exploration rate of 0.1 and a discount factor of 0.6. From this, it can be said that for this problem and the environment setup, a higher value of α is good implying the learning agent is highly flexible and adaptable. For ϵ, the best value was 0.1, implying that being greedy is good for the agent while determining which action to take. Lastly, for γ the best value was 0.6, concluding that living in the moment is better for the agent.

We see that Q-learning is able to determine actions in a desired manner where the water level in the reservoir stays within the water bounds and the switch between actions is not too frequent. We also see that when using Q-learning agent, the system touches the critical maximum or minimum occasionally but it rarely crosses the extreme maximum or minimum water boundaries. This implies that most of the time, the Q-learning agent determines the intake pump setpoints in a desired manner where the reservoir does not overflow/underflow and the water level stays within the critical bounds with a low frequency of the pump setpoints switch. This behavior follows for both types of data for consumption pattern i.e. sine wave data which is often unbiased with a bit of noise added in it and the real data which has more disturbances and fluctuations present.

When using an agent trained using N-step, the agent takes actions in such a way that the water level in the reservoir usually always stays within the water bounds implying the reservoir does not underflow/overflow for sine wave data. The same trend follows with real data as well with occasional critical water boundary touches but then the agent navigates the system to desired water level with the next action taken. The frequency of critical water boundary touches is more for real data because real data has more disturbances present and N-step is good at picking even minor changes to improve and lead the agent's learning in the targeted direction. Furthermore, the frequency of the switch between pump intake setpoints is slightly higher with N-step agent because it tries to learn and navigate more frequently whenever there are any changes in demand values.

Both the algorithms Q-learning and N-step perform (training reward) almost the same with a slight difference in the number of action switches. However, even though the action switch count is higher for N-step, the difference/jump between the two pump setpoints is higher for Q-learning.

Hence, for noisy sine wave data, since the data is unbiased for whole years with a slight noise in it, the algorithms are able to learn properly and both algorithms perform almost the same. For real data for the Leersum reservoir, we could clearly see the differences in performance for N-step and Q-learning where overall Q-learning performs better because the real data varies as the months pass, resulting in more fluctuations and disturbances in the data. However, there is a trade-off between the number of the switch count and water boundaries. N-step is good at maintaining the water level within bounds in comparison to Q-learning while Q-learning has less frequent pump intake setpoint switches. Additionally, we observed that the Q-learning and N-step agents achieved an average episodic reward of -16.05 and -18.51

on a scale of $[-1000, 1]$ where optimal average reward would be 0.027. Overall, we learned that these agents are capable of keeping the water level within the desired water range with a reasonable number of intake pump setpoint switches. However, the frequency of intake pump switches is not optimum which could be later improved in future work. This is because as the agents learn to keep the water level within the bounds with a reasonable number of intake switches, it becomes difficult agents to explore and learn as there is a trade-off between exploration and exploitation.

6 Conclusion and Future Work

The aim of this thesis was to explore the possibilities to control and optimize water production in a reservoir based on a demand forecast for the next three days using reinforcement learning. To do so two classical reinforcement algorithms such as Q-learning and N-step were used to answer our research questions. We used these approaches as these are simple yet powerful reinforcement learning algorithms that can be used for various machine learning problems. Q-learning is simple and robust while N-step provides more efficient learning and stability. Additionally, we performed these experiments on noisy sine wave data which we created and real data to truly understand and analyze these methods and their performance.

The main research question we looked into the optimization of inflow for an outflow while satisfying the water level boundary conditions and less frequent action switches. This question consists of three sub-questions that we answered below.

1. *Will the RL agent be able to learn the optimum policy using Q-learning with an optimum number of production pump intakes?*

   We saw that the RL agent using Q-learning is able to learn the optimal policy and is capable to determine the pump intake set point in such a way that the reservoir water level boundaries are usually satisfied with a few occasional critical boundaries touches with not-so-frequent intake pump setpoint changes.

2. *Will a Q-learning agent be able to foresee and predict consumption values and plan or schedule accordingly?*

   We noticed that the Q-learning agents are not able to foresee and determine action based on future demand values in the near future as these agents were expected to learn the changes in consumption implicitly during their learning phase which was seen in results of both the approaches used for Q-learning. As a result, it is not able to schedule in advance.

3. *If Q-learning is not able to achieve the desired results, will the N-step algorithm be beneficial in such cases?*

   We looked into if N-step algorithm will be able to overcome Q-learning shortcomings regarding not being able to schedule in advance. It was seen that N-step the agent is able to manage close-by future consumption values slightly but because of this, the frequency of action switch has increased slightly. However, since N-step is able to see a few steps in advance, it is more resistant to overflow/underflow in comparison to the Q-learning agent.

Moreover, after analyzing these agent's performance, we see that there is a trade-off between the intake setpoint switch frequency and water level overflow/underflow. We saw that Q-learning agent has a smaller number of intake set point change count in comparison to N-step agent while N-step has lower occurrences for overflow/underflow.

For hyperparameter configurations, it was also seen that it is better to be greedy instead of exploring, taking immediate rewards into account instead of considering future rewards and being able to learn quickly during the hyperparameter optimization for this problem as this combination provides the best performance for these RL agents for both the algorithms.

Lastly, we did a comparison study between these agents with a PLC system. We saw that RL agents are able to satisfy the water level boundary conditions however, these agents are not up to the level of PLC or OPIR model when it comes to the frequency of intake pump setpoint switches. This is because these agents try to navigate based on current demand so even a slight change in demand can make them switch the action.

Future Work

For an agent to learn properly, the environment should be defined as close as possible to the actual problem in the real world. The observation space and the reward calculation method of the environment should be in such a way that represents the problem most accurately. In this work, we used a simple environment with a simple reward and penalty to help learn agents to satisfy water level boundary conditions and action switches. For future work, we could look into a more advanced and more realistic environment for a single reservoir system. Additionally, for this problem, there is a trade-off between water level boundaries and action switch while calculating the reward as one would have to prioritize which is more important and define the reward based on that. Furthermore, a more realistic environment may have additional parameters such as pressure, temperature etc so those can also be explored. Moreover, how the consumption forecast can be a part of state space in the environment could also be explored.

Once we move to a more complex environment which might lead to a high-dimensional or continuous observation space, then we would require an algorithm that is able to overcome shortcomings of Q-learning and N-step such as Deep Q-Networks (DQN) since Q-learning and N-step are suitable for discrete observation space with low-dimensional observation space.

Since Q-learning and N-step algorithms are model-free approaches in reinforcement learning, we could also look into model-based approaches as they can use a model to simulate a planning problem since they are able to use samples more efficiently as model-free approaches rely on trial-and-error learning.

Furthermore, one could also explore feedback-feedforward policies in reinforcement learning as these policies use a combination of both feedback and feedforward control to make decisions as they leverage the advantages of feedback along with feedforward control, which helps reach more effective decision-making. However, designing such a system would be challenging.

Lastly, this research is focused on a single reservoir system. However, in real life, we can have more complex hydraulic networks with multiple reservoirs so a more complex environment and combinations of approaches can be explored.

References

[1] Néda .M et al. "Use of Machine Learning for Leak Detection and Localization in Water Distribution Systems". In: *Smart Cities* 4 (Oct. 2021), pp. 1293–1314. DOI: 10.3390/smartcities4040069.

[2] Linda Åmand. "Ammonium Feedback Control in Wastewater Treatment Plants". PhD thesis. Apr. 2014.

[3] M Bakker et al. "Better water quality and higher energy efficiency by using model predictive flow control at water supply systems". English. In: *Journal of Water Supply: Research and Technology. Aqua* 62.1 (2013). Harvest, pp. 1–13. ISSN: 0003-7214. DOI: 10.2166/aqua.2013.063.

[4] M. Bakker et al. "A fully adaptive forecasting model for short-term drinking water demand". In: *Environmental Modelling Software* 48 (2013), pp. 141–151. ISSN: 1364-8152. DOI: https://doi.org/10.1016/j.envsoft.2013.06.012. URL: https://www.sciencedirect.com/science/article/pii/S1364815213001576.

[5] M. Bakker et al. "Advanced control of a water supply system: A case study". English. In: *Water Practice and Technology* 9.2 (June 2014), pp. 264–276. ISSN: 1751-231X. DOI: 10.2166/wpt.2014.030.

[6] Martijn Bakker, Kim van Schagen, and Jan Timmer. "Flow control by prediction of water demand". In: *Journal of Water Supply: Research and Technology-Aqua* 52.6 (Sept. 2003), pp. 417–424. ISSN: 0003-7214. DOI: 10.2166/aqua.2003.0038. eprint: https://iwaponline.com/aqua/article-pdf/52/6/417/402382/417.pdf. URL: https://doi.org/10.2166/aqua.2003.0038.

[7] M. J. Bowes et al. "Weekly water quality monitoring data for the River Thames (UK) and its major tributaries (2009–2013): the Thames Initiative research platform". In: *Earth System Science Data* 10.3 (2018), pp. 1637–1653. DOI: 10.5194/essd-10-1637-2018. URL: https://essd.copernicus.org/articles/10/1637/2018/.

[8] S. Bunn. "Closing the Loop in Water Supply Optimisation". In: May 2007, pp. 71–82. ISBN: 978/0-86341-791-7. DOI: 10.1049/ic:20070562.

[9] Kehua Chen et al. "Optimal control towards sustainable wastewater treatment plants based on multi-agent reinforcement learning". In: *Chemosphere* 279 (2021), p. 130498. ISSN: 0045-6535. DOI: https://doi.org/10.1016/j.chemosphere.2021.130498. URL: https://www.sciencedirect.com/science/article/pii/S0045653521009681.

[10] Guangtao Fu et al. "The role of deep learning in urban water management: A critical review". In: *Water Research* 223 (2022), p. 118973. ISSN: 0043-1354. DOI: https://doi.org/10.1016/j.watres.2022.118973. URL: https://www.sciencedirect.com/science/article/pii/S0043135422009204.

[11] Jorge Gironás et al. "A new applications manual for the Storm Water Management Model (SWMM)". In: *Environmental Modelling Software* 25 (June 2010), pp. 813–814. DOI: 10.1016/j.envsoft.2009.11.009.

[12] J. Gouthaman, R. Bharathwajanprabhu, and A. Srikanth. "Automated urban drinking water supply control and water theft identification system". In: *IEEE Technology Students' Symposium*. 2011, pp. 87–91. DOI: 10.1109/TECHSYM.2011.5783807.

[13] Mengjie Han et al. "A review of reinforcement learning methodologies for controlling occupant comfort in buildings". In: *Sustainable Cities and Society* 51 (2019), p. 101748. ISSN: 2210-6707. DOI: https://doi.org/10.1016/j.scs.2019.101748. URL: https://www.sciencedirect.com/science/article/pii/S2210670719307589.

[14] Shiyuan Hu et al. "Real-Time Scheduling of Pumps in Water Distribution Systems Based on Exploration-Enhanced Deep Reinforcement Learning". In: *Systems* 11.2 (2023). ISSN: 2079-8954. DOI: 10.3390/systems11020056. URL: https://www.mdpi.com/2079-8954/11/2/56.

[15] Youqin Huang, Jiayong Li, and Jiyang Fu. "Review on Application of Artificial Intelligence in Civil Engineering". In: *Computers, Materials Continua* 61 (Jan. 2019), pp. 845–875. DOI: 10.32604/cmes.2019.07653.

[16] Kiran Joseph, Ashok K. Sharma, and Rudi van Staden. "Development of an Intelligent Urban Water Network System". In: *Water* 14.9 (2022). ISSN: 2073-4441. URL: https://www.mdpi.com/2073-4441/14/9/1320.

[17] Fatemeh Karimi Pour, Vicenç Puig, and Gabriela Cembrano. "Economic Health-Aware LPV-MPC Based on System Reliability Assessment for Water Transport Network". In: *Energies* 12 (Aug. 2019), p. 3015. DOI: 10.3390/en12153015.

[18] Hussain Kazmi et al. "Gigawatt-hour scale savings on a budget of zero: Deep reinforcement learning based optimal control of hot water systems". In: *Energy* 144 (2018), pp. 159–168. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2017.12.019. URL: https://www.sciencedirect.com/science/article/pii/S0360544217320388.

[19] Jens Kober, J. Bagnell, and Jan Peters. "Reinforcement Learning in Robotics: A Survey". In: *The International Journal of Robotics Research* 32 (Sept. 2013), pp. 1238–1274. DOI: 10.1177/0278364913495721.

[20] Florian Köpf et al. "Reinforcement Learning for Speed Control with Feedforward to Track Velocity Profiles in a Real Vehicle". In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. 2020, pp. 1–8. DOI: 10.1109/ITSC45102.2020.9294541.

[21] Dorien Lambregts. "Reinforcement learning for water system control". MA thesis. Delft University of Technology, 2022.

[22] Harry Lintsen. "Two Centuries of Central Water Management in the Netherlands". In: *Technology and Culture* 43 (July 2002), pp. 549–568. DOI: 10.1353/tech.2002.0126.

[23] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: (2013). cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013. URL: http://arxiv.org/abs/1312.5602.

[24] OECD. *Water Governance in the Netherlands*. 2014, p. 296. DOI: https://doi.org/https://doi.org/10.1787/9789264102637-en. URL: https://www.oecd-ilibrary.org/content/publication/9789264102637-en.

[25] Aske Plaat. *Deep Reinforcement Learning*. Springer Nature Singapore, 2022. DOI: 10.1007/978-981-19-0638-1. URL: https://doi.org/10.1007%2F978-981-19-0638-1.

[26] R. Romero et al. "Research on automatic irrigation control: State of the art and recent results". In: *Agricultural Water Management* 114 (2012). For a better use and distribution of water, pp. 59–66. ISSN: 0378-3774. DOI: https://doi.org/10.1016/j.agwat.2012.06.026. URL: https://www.sciencedirect.com/science/article/pii/S0378377412001746.

[27] L A Rossman. "EPANET water quality model". In: (Jan. 1993). URL: https://www.osti.gov/biblio/5795398.

[28] G. A. Rummery and M. Niranjan. *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. TR 166. Cambridge, England: Cambridge University Engineering Department, 1994.

[29] Yuji Saikai, Allan Peake, and Karine Chenu. *Deep reinforcement learning for irrigation scheduling using high-dimensional sensor feedback*. 2023. arXiv: 2301.00899 [cs.LG].

[30] Iqbal Sarker. "Machine Learning: Algorithms, Real-World Applications and Research Directions". In: *SN Computer Science* 2 (Mar. 2021). DOI: 10.1007/s42979-021-00592-x.

[31] Lariyah Mohd Sidek et al. "Application of PCSWMM for the 1-D and 1-D–2-D Modeling of Urban Flooding in Damansara Catchment, Malaysia". In: *Applied Sciences* 11.19 (2021). ISSN: 2076-3417. DOI: 10.3390/app11199300. URL: https://www.mdpi.com/2076-3417/11/19/9300.

[32] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (Jan. 2016), pp. 484–489. DOI: 10.1038/nature16961.

[33] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550 (2017), pp. 354–359.

[34] Patrick Smeets, Gertjan Medema, and J. Dijk. "The Dutch secret: How to provide safe drinking water without chlorine in the Netherlands". In: *Drinking Water Engineering and Science* 2 (Mar. 2009). DOI: 10.5194/dwes-2-1-2009.

[35] Richard Sutton. "Learning to Predict by the Method of Temporal Differences". In: *Machine Learning* 3 (Aug. 1988), pp. 9–44. DOI: 10.1007/BF00115009.

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

[37] Yinping Wang et al. "Soil and Water Assessment Tool (SWAT) Model: A Systemic Review". In: *Journal of Coastal Research* 93 (Sept. 2019), p. 22. DOI: 10.2112/SI93-004.1.

[38] C. J. C. H. Watkins. “Learning from Delayed Rewards”. PhD thesis. King’s College, Oxford, 1989.

[39] Kenichi Yoshizawa. “Advanced Water Treatment System of the Bureau of Waterworks, Tokyo Metropolitan Government”. In: *Ozone: Science & Engineering* 41.4 (2019), pp. 312–321. DOI: 10.1080/01919512.2019.1599780. eprint: https://doi.org/10.1080/01919512.2019.1599780. URL: https://doi.org/10.1080/01919512.2019.1599780.