12-2021

Reinforcement Learning Policy Gradient Methods for Reservoir Operation Management and Control

Sadegh Sadeghi Tabas
sadeghs@clemson.edu

Reinforcement Learning Policy Gradient Methods for Reservoir Operation Management and Control

Sadegh Sadeghi Tabas

REINFORCEMENT LEARNING POLICY GRADIENT METHODS FOR RESERVOIR OPERATION MANAGEMENT AND CONTROL

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Science

by
Sadegh Sadeghi Tabas
December 2021

Abstract

Changes in demand, various hydrological inputs, and environmental stressors are among issues that water managers and policymakers face on a regular basis. These concerns have sparked interest in applying different techniques to determine reservoir operation policy and improve reservoir release decisions. As the resolution of the analysis rises, it becomes more difficult to effectively represent a real-world system using traditional approaches for determining the best reservoir operation policy. One of the challenges is the "curse of dimensionality," which occurs when the discretization of the state and action spaces becomes finer or when more state or action variables are taken into account. Because of the dimensionality curse, the number of state-action variables is limited, rendering Dynamic Programming (DP) and Stochastic Dynamic Programming (SDP) ineffective in handling complex reservoir optimization issues. Deep Reinforcement Learning (DRL) is an intelligent approach to overcome the aforementioned curses of stochastic optimization of reservoir system planning. This study examined various novel DRL continuous-action policy gradient methods (PGMs), including Deep Deterministic Policy Gradients (DDPG), Twin Delayed DDPG (TD3), and two different versions of Soft Actor-Critic (SAC18 and SAC19) to identify optimal reservoir operation policy for the Folsom Reservoir located in California, US. The Folsom Reservoir supplies agricultural and municipal water, hydropower, environmental flows, and flood protection to the City of Sacramento. We concluded DRL methods release decisions with respect to these demands as well as by comparing the results to standard operating policy (SOP) and base conditions using different performance criteria and sustainability indices. TD3 and SAC methods have shown promising performance in providing optimal operation policy. Experiments on continuous-action spaces of reservoir operation policy decisions demonstrated that the DRL techniques could efficiently learn strategic policies in space with the curse of dimensionality and modeling.

Keywords: Deep Reinforcement Learning, Continuous Action Spaces, Multi-Purpose Reservoir System, Sustainable Reservoir Operation

Dedication

To my parents, for their endless support.

Acknowledgments

I would like to extend many thanks to my advisors, Drs. Vidya Samadi and Nina Hubig, for their guidance throughout this project and for inspiring my interest in the application of reinforcement learning in reservoir operation problem. I am very fortunate that they were willing to accept me as an adviser and have benefited in many ways from their support. The other member of my dissertation committee has been most gracious with his time and expertise. Thank you to Dr. Feng Lou for providing me with a foundation to explore machine learning. This research was supported by the US Geological Survey (grant number # 5001-20-207-0312-216-2024917). Clemson University is acknowledged for generous allotment of computing time on the Palmetto cluster. And finally, thank you to my parents. Without them, this would not have been possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Problem Specification

Reservoir systems often exhibit high degrees of short- and long-term hydrologic variabilities, along with the complexity of multipurpose operating policies that evolved over years of litigation, court orders, and institutional regulations [41]. Stochasticity, nonconvexity, nonlinearity and dimensionality are the major determinants of complexities in solving reservoir system operational problems. In uncertain environments with complicated and unknown relations between numerous system characteristics, there are many strategies that could bypass these complexities via some type of approximation [36]. For instance, dynamic programming (DP) and stochastic dynamic programming (SDP), two well-known techniques employed for reservoir operation management and control, are plagued by the so-called dual curse of dimensionality and modeling, which prevents them from being implemented in relatively complicated reservoir operating systems. An exponential rise in computational complexities characterizes the dimensionality curse as the state–decision space and disturbance dimensions expand [4]. On the other hand, the curse of modelling requires an explicit model of each component of the reservior system in order to calculate the impact of each system transition [6].

The curse of dimensionality restricts the number of state-action variables, rendering DP and SDP ineffective in handling complex reservoir optimization problems [15]. The information included in the SDP can be either a state variable described by a dynamic model or a stochastic disturbance (time-independent) with the associated Probability Density Function (PDF). Exogenous information (such as precipitation, temperature and snowpack depth which can effectively improve operation performance) cannot be explicitly considered in making the release decision unless a dynamic model is identified for additional information that adds to the curse of dimensionality (additional state variables) [22, 51]. Furthermore, disturbances are very likely to be spatially and temporally correlated in large reservoir networks. However, including spatial variability in identifying the disturbance's PDF can be time-consuming, while temporal correlation can be properly accounted for using a dynamic stochastic model. However, this can again intensify the curse of dimensionality [11].

There have been various attempts to overcome the curse of dimensionality in the literature, e.g., Successive Approximations DP [5], Incremental DP [28], Differential DP [24], and problem-specific heuristics [35, 59]. However, these methods have been designed primarily for deterministic problems and less applicable for the optimal reservoir operation system, where the uncertainty associated with the underlying hydro-meteorological processes cannot be neglected. Depending on the procedure they adopt to alleviate the dimensionality problem, alternative approaches can be classified into two main classes: (i) methods based on the simplification of the water system model and (ii) methods based on the restriction of the degrees of freedom of the policy design problem [13].

A possible method for overcoming the mentioned curses of stochastic optimization of water resources systems is reinforcement learning (RL) [49]. RL is a prominent machine learning (ML) paradigm concerned with how intelligent agents take sequential actions through interacting with environments (deterministic or stochastic) and learn from the feedback (instantaneous or delayed) to cope with many simulation-optimization problems [49, 56]. These interactions may result in an immediate reward (or penalty) accumulated throughout the training process and is referred to as action-value functions. These values are the basis for the agent to take proper actions in different situations (states). During these interactions, the agent encounters new states, gain experience, and applies them in future decision-making. Obviously, at the beginning of the learning, the agent observes new situations that have never been encountered before. In this situation, the action taken is not based on prior knowledge (so-called exploration). However, after sufficient interactions with the environment, the agent begins to understand the behavior of the system, and thereafter it attempts to utilize this information for more accurate decision-making (so-called exploitation). Furthermore, the agent frequently seeks out new knowledge about the environment by performing a random action. Thus, in RL, the search space over the range of possible releases reduces, allowing for faster execution. As a result, RL employs a search approach that mitigates to some extent the curse of dimensionality problem that has plagued SDP applications for a long time. Furthermore, rather than requiring a priori transition probability matrices, RL incorporates a learning process that accumulates knowledge of the underlying stochastic nature of river basin hydrologic fluxes.

The first application of RL in water-related domains is proposed by Wilson (1996) for real-time optimal control of hydraulic networks [58]. Soon after, Bouchart and Chkam (1998) used RL to operate a multi-reservoir system in Scotland and offered a method to circumvent the dimensionality curse in RL [7]. An excellent application of RL Q-learning method is proposed by Castelletti et al. (2001) for optimizing the daily operation of a single reservoir system in Italy, and it was demonstrated to outperform implicit SDP [10]. In a sequence, Castelletti et al. (2002) proposed a variant of Q-learning — so called Qlp (Q-learning planning) — to conquer the limitations of SDP and standard Q-learning by incorporating the off-line approach that is typical for SDP and Q-learning model-free characteristics [12]. Lee and Labadie (2007) then compared the Q-learning technique to implicit SDP and sampling SDP for the long-term operation of a multipurpose two-reservoir system in South Korea [29]. Their results demonstrated that Q-learning outperformed implicit SDP and sampling SDP methods. Other examples of Q-learning employed for water resources systems can be found in Bhattacharya et al. (2003), Ernst et al. (2005), Mariano-Romero et al.

(2007), Mahootchi et al. (2007), Mahootchi et al. (2010), and Rieker and Labadie (2012). Castelletti et al. (2010) applied the fitted Q-iteration coupled with a tree-based regression to form an appropriate function approximator for the daily operation of a single reservoir system in Italy. Likewise, a multi-objective version of RL was proposed by Castelletti et al. (2013), which is capable of obtaining the Pareto frontier and was applied to operate a single-reservoir system in Vietnam. Recently, Delipetrev et al. (2017) implemented two novel multipurpose reservoir optimization algorithms named nested stochastic dynamic programming (nSDP) and nested reinforcement learning (nRL). Their result showed the nRL is more powerful but significantly more complex than nSDP. More recently, Mullapudi et al. (2020) implemented an RL algorithm for the real-time control of urban stormwater systems. Their procedure trains an RL agent to control valves in a distributed stormwater system with the goal of achieving water level and flow set-points in the system. Their results indicated that RL could effectively control individual sites, although its performance can be susceptible to the reward function formulation. Concurrently, Bertoni et al. (2020) developed a novel RL-based approach to integrate dam sizing and operation design. The parametric policy is computed through a novel batch-mode RL algorithm, called Planning Fitted Q-Iteration (pFQI). Their findings revealed that the proposed RL approach is capable of identifying more efficient system configurations with respect to traditional sizing approaches that could potentially neglect the optimal operation design phase.

1.2 Need for the Study

However, traditional RL tackle problems with high-dimensional observation spaces; it can only handle discrete and low-dimensional action spaces. To conquer this issue, some techniques such as simply discretizing, interpolation, regression, neural networks, and fuzzy logic are proposed to approximate value function in RL (Castelletti et al., 2010; Gosavi, 2003; Suttan and Barto, 1998). However, these strategies introduce other issues (Gosavi, 2003), and such large action spaces are challenging to explore efficiently. Dimensionality problems arise again as a result of the need for discretizing the state-action space and random variates, which leads to a massive increase in computational and memory demands. Also, deterministic discretization would result in a significant bias in the policy evaluation and improvement. Furthermore, naive discretization of action spaces may throw away valuable information on the structure of the action domain, which may be necessary for good performance. Therefore, successfully training RL networks in this context appears to be problematic.

Leveraging deep neural networks (DNNs) for function approximation, Deep RL (DRL) is recently developed to solve large-scale complex decision problems. DRL has the potential to capture hard-to-model dynamics systems due to its model-free nature and its ability to make sequential decisions in an uncertain environment by maximizing the cumulative reward (e.g., Li, 2017) [30]. Mathematically, the decision-making problem is modeled as Markov Decision Processes (MDPs), which are defined by state space, action space, the transition probability function that maps a state-action pair to a distribution on the state space, and the reward function. The agent chooses action and receives reward that depends on the current state-action pair (policy), after which the next state is randomly generated from the transition probability. The DRL-based decision-making mechanisms such as policy gradient approaches are envisioned to compensate for the limitations of existing model-based approaches and can naturally handle discrete, continuous, or even hybrid action spaces, and thus are promising to address the emerging challenges described in reservoir optimization problems [37, 44, 57].

1.3 Contribution of the Research

In this study, we developed various novel DRL algorithms including Deep Deterministic Policy Gradients (DDPG), Twin Delayed DDPG (TD3), and two different versions of SAC (SAC18 and SAC19 hereafter) to solve the DP problem for operating a single-reservoir system and effectively tackle dimensionality issues without requiring any model simplifications or sacrificing any DP advantages. The policy gradient methods (PGMs) used in this study follow an iterative learning process that takes into account delayed rewards without requiring an explicit probabilistic model of the physical (hydrologic) processes associated with the reservoir system. The proposed methodology also allows learning the best actions that maximize total expected reward through interaction with the environment. These methods were executed in a model-free stochastic environment wherein it retains the system's underlying stochastic behavior to suggest the optimal feedback operating policies. The use of accurate function approximators based on DNN for the state-value and policy functions take into consideration high-dimensional continuous action spaces. The DRL methods were applied to find optimal reservoir operational strategies for the Folsom Reservoir on the American River Basin located in California, US, in the presence of multiple non-commensurate objectives such as hydropower, flood control, agricultural and municipal water supply, and environmental flow requirements.

1.4 Thesis Organization

This dissertation is organized into five chapters, beginning with the introductory chapter in which the problem is described, recent related literature are reviewed and the contribution of this study is presented. Chapter two presents a detailed description of the dynamic of reservoir operation problem, formulations and structures of the methods used in this research including background information in RL and employed policy gradient methods followed by the standard operating policy and various performance and sustainability indices to evaluate different RL methods. Chapter three introduces the Folsom Reservoir characteristics which is used as a case study for application of the methods developed in this research. Chapter four presents the results of this research and discusses about the performance of different RL methods in identifying the optimal operating policy of the single reservoir system and the results of the continuous action DRL method used to calculate state-value functions for multiple objectives are shown. Chapter five contains the conclusions of this research and recommendations for future research.

Chapter 2

Methodology

2.1 Overview

This chapter will describe the methods used in the current research. As mentioned above, Policy Gradients Reinforcement learning techniques are used to estimate the state-value function, and determine release decisions given a single-reservoir environment and considering multiple objectives. A case study is performed to apply these decision making methods for the Folsom Reservoir, the primary water supply and flood control reservoir on the American River basin located in Northern California. The decision policies include four criteria: flood management, water supply, environmental flows and hydropower. The characteristics of the area of study is explained in-detail in Chap. 4. To evaluate the optimal policy actions provided by each PGMs, results are compared with the Standard Operating Policy (SOP) and base conditions based on different performance and sustainability indices. This chapter contains four subsections, including (i) problem statement (ii) reinforcement learning (iii) policy gradient methods, (iv) standard operating policy under flood control condition and (v) performance criteria and sustainability.

2.2 Problem Statement

As stated earlier, the Folsom Reservoir's primary functions include flood control, power generation, water supply, and environmental protection. These functions are identified as the objectives and constraints of the Folsom Reservoir optimization problem that are incorporated into the model as mathematical expressions. Therefore, the RL methods are able to estimate the release from the reservoir and satisfy the constraints and objectives in each timestep. A single-reservoir mass balance model with a monthly timestep is considered to represent the system, following Eq. 2.1.

$$S_{t+1} = S_t + Q_t - E_t - R_t - Spill_t \tag{2.1}$$

where S_t and Q_t denote reservoir storage and inflow to reservoir respectively, at time step t (available water). Also, E_t , R_t and $Spill_t$ denote evaporation losses, release and spill from the reservoir respectively, at time step t.

The Folsom Reservoir's elevation-storage relationship is calculated by interpolating each time step using the reservoir's bathymetry (Fig. 2.1-a). The Folsom Reservoir operates for flood control based on a rule curve provided by the United States Army Corps of Engineers (USACE) and the Sacramento Flood Control Agency (SAFCA). Fig. 2.1-b presents a simplified version of the maximum allowable reservoir levels for operation and it shows a maximum flood control space of about 575 thousand acre-ft (TAF) during the wet season. The evaporation values at each time step were calculated based on the monthly evaporation rates (Table 2.1) while the Folsom Lake surface area was calculated by interpolation using the reservoir's bathymetry (Fig. 2.1-a). The RL release decisions are constrained by the physical characteristics

Table 2.1: Monthly evaporation rates (in) for Folsom Reservoir.

Month	Evaporation (in)	Month	Evaporation (in)	Month	Evaporation (in)
January	0.91	**May**	8.07	**September**	7.64
February	1.61	**June**	10.08	**October**	5.00
March	3.50	**July**	11.50	**November**	2.05
April	3.50	**August**	10.20	**December**	0.91

of reservoir outlets, downstream channels, bathymetry of the reservoir (Fig. 2.1-a), hydropower intakes and turbine capacity, and rule-based operational objectives (Eqs. 2.2-2.5). The model does not account for several smaller upstream reservoirs that

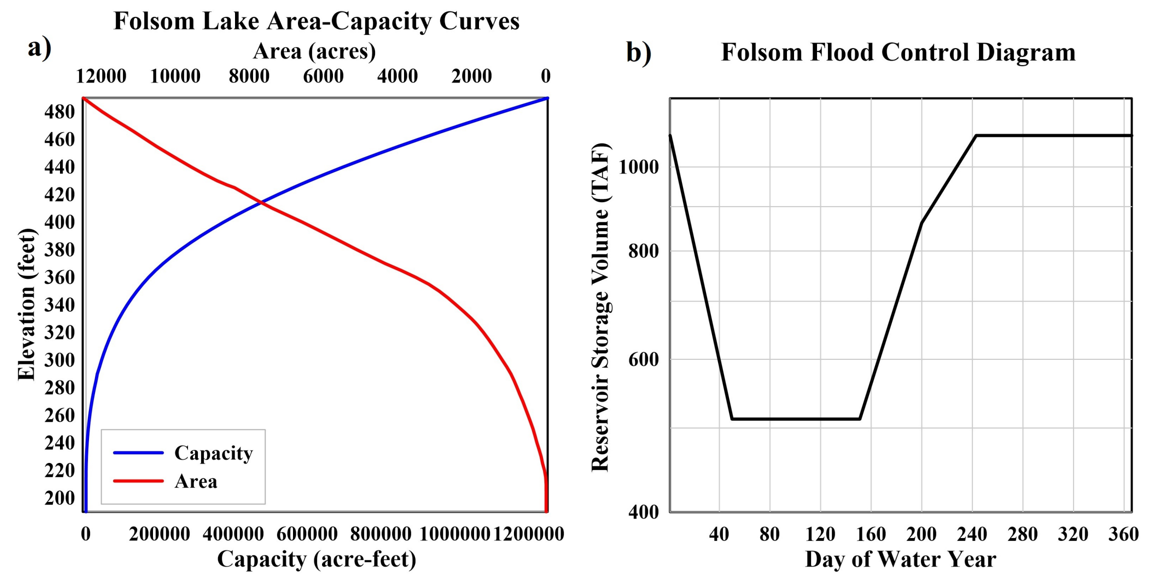

Figure 2.1: a) Folsom Reservoir's elevation-storage-area relationship. b) A simplified version of the maximum allowable reservoir levels for operation of the Folsom Reservoir.

provide additional flood control space during large storms.

$$S^{\min} \leq S_t \leq S_t^{\max} \tag{2.2}$$

$$R^{\min} \leq R_t \leq R^{\max} \tag{2.3}$$

$$HP_t = \eta g \gamma_w \overline{h_t} R_t^{Turb} \times 10^{-6} \tag{2.4}$$

$$R_t^{Turb} \leq R_{\max}^{Turb} \tag{2.5}$$

where S_{min} and S_t^{max} denote minimum reservoir volume and maximum allowable reservoir storage at time step t (Fig. 2.1-b), respectively; R^{min} and R^{max} are the minimum and maximum total water release from the reservoir; both depend on the minimum and maximum flow constraints at the downstream of reservoir (maximum safe downstream release is 130,000 cfs); $\eta = 0.85$ is the turbine efficiency, $g = 9.81(m/s^2)$ the gravitational acceleration, $\gamma_w = 1000(kg/m^3)$ the water density, $\overline{h_t}(m)$ the net hydraulic head (i.e., reservoir level minus tailwater level), $R_t^{Turb}(cms)$ the turbined flow, $HP_t(MWh)$ is the hourly energy production and is the turbine maximum release capacity.

2.3 Reinforcement Learning

The nature of reservoir operation problem led to the use of RL as a means of estimating the state-value function. RL refers to a class of ML methods in which an intelligent agent can learn advantageous behaviours from interactions with its environment. The intelligent agent is given an optimization goal to conduct actions that, along with possible random input, result in changes to the environment. A reward signal from the environment alerting the agent of the value of the action taken, as well as a state signal returning the environment's consequent state. This section is mostly elaborated based on Peacock (2020) study [40].

2.3.1 Markov Decision Processes

The Markov Decision Process (MDP) serves as the foundation for the design of an RL problem. A finite MDP problem consists of a discretized state, $S_t \in S$, a set of actions that can be taken in a given state, $A_t \in A(s)$, and the rewards that can be achieved, $R_t \in R$. Through the system dynamics equation, the received reward and the system's resulting state are associated to the initial state and chosen action.

After starting in state s and performing action a, Eq. 2.6 gives the probability of finding the system in state s' and getting a reward r.

$$p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{2.6}$$

The agent maximizes its learning objective or expected cumulative reward over a sequence of time steps (Eq. 2.7)

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_t \tag{2.7}$$

The concept of discounted returns is introduced to avoid infinite returns and to reflect the diminished value given to future rewards (Eq. 2.8).

$$G_t = \sum_{\tau=0}^{T} \gamma^{\tau} R_{t+\tau+1} \tag{2.8}$$

where $\gamma \in [0, 1]$ and T is the number of time steps. As a result, the goal is to determine the set of actions that maximize the expected value of the return (Eq. 2.9).

$$J = \mathrm{E}\left[G_t|\, S_t = s\right] \tag{2.9}$$

2.3.2 Value Functions and Optimal Policies

DP and other methods for solving MDPs often consider two value functions. The state-value function, $\upsilon_{\pi}(s)$, reflects the return's expected value based on a known state and actions taken according to a policy (Eq. 2.10). The policy is allowed to be stochastic or deterministic. In the stochastic case, $\pi(a|s)$ is a mapping from a state to the probability of taking action a. In the deterministic case, the policy maps from a state to an action and the notation is adjusted to $\mu(s)$ to indicate the difference. The action-value function (q_π) describes the expected reward for a given state conditioned by taking a particular action and following the policy π at all subsequent stages (Eq. 2.11). The Bellman equations, a recursive characteristic of value functions, are used to solve this problem (Bellman, 1957). Eq. 2.12 presents the state-value function based on Bellman equation.

$$\upsilon_\pi(s) \doteq E_\pi \left[G_t | \, S_t = s \right] \tag{2.10}$$

$$q_\pi \doteq E_\pi \left[G_t | \, S_t = s, A_t = a \right] \tag{2.11}$$

$$\upsilon_\pi(s) = E_\pi \left[R_{t+1} + \gamma \upsilon_\pi(s') | \, S_t = s \right] \tag{2.12}$$

Therefore, the agent can find the optimal policy π^* based on state value function (Eq. 2.13). For the optimal policy and all possible states, Eq. 2.14 presents the action-value function. Thus, the state-value and action-value functions are related by Eq. 2.15.

$$\upsilon^*(s) = \max_\pi \upsilon_\pi(s) \tag{2.13}$$

$$q^*(s,a) = \max_\pi q_\pi(s,a) \tag{2.14}$$

$$\upsilon(s) = \max_a q(s,a) \tag{2.15}$$

Combining optimality notation with the recursive characteristic of the value functions, the agent can treat the problem as a greedy optimization of the return over the next stage plus the value of the resulting state, which is represented by the Bellman optimality equations:

$$\upsilon^*(s) = \max_a E\left[R_{t+1} + \gamma \upsilon^*(S_{t+1}) \,|S_t = s, A_t = a\right] \tag{2.16}$$

$$q^*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q^*(S_{t+1}, a') \,|S_t = s, A_t = a\right] = E\left[R_{t+1} + \gamma \upsilon^*(s')\right] \tag{2.17}$$

The optimal value functions (Eqs. 2.16 and 2.17) can be computed using DP by assuming an initial state value, iteratively solving backward the Bellman optimality equations and updating the current value at each iteration. This process is repeated until a satisfactory level of convergence is achieved.

The agent's current view of the state provides all information required to characterize the environment, which is implicitly identified in the problem formulation. So that, the current state should have all of the information required to provide reliable probabilities to the state dynamics equation. A partially observable state occurs when there is insufficient information about the state of the system, and the problem becomes a Partially Observable MDP (POMDP). Many research studies have been conducted to solve POMDPs [34].

As many elements contribute to the transition from one state to the next, such as historical precipitation and soil moisture, which are not always available, a realistic assessment of a reservoir system problem may require considering the state as partially observable. Besides the partially observable states issue, there's the issue of relying on probability distributions to express the state dynamics equation. The transition probabilities for a reservoir system can be imprecise and unable to capture the temporal and spatial correlations of the random variables when there are more than a few states or random variables, or time steps are short. Furthermore, when a system is evaluated at fine temporal resolution, routing effects may lead to rewards to be realized several time steps after the decision was made, a concept known as delayed rewards. This is an issue that a probabilistic model cannot address.

2.3.3 Temporal-Difference Learning

RL and DP have a lot in common, however utilizing trial and error approaches such as Monte Carlo (MC) is one of their dissimilarities [50]. The process of learning in RL relies on interactions between environment and reinforcing information (identified as rewards) and it does not require any knowledge of a probabilistic model of the environment. Methods that don't make use of probabilistic models are so-called model-free procedures. Temporal-difference learning, which is utilized in the highly successful Q-learning approach, is another feature of some RL methods [56]. In this method, the agent stores an estimate, $Q(s, a)$, of the $q(s, a)$ function and updates the estimate by iteratively interacting with the environment and getting a reward at each time step. The agent takes actions based on a policy that insures exploration such as the ϵ-greedy method, in which for a value of $\epsilon \in [0, 1]$ the action $A_t = argmax_a Q(S_t, a)$ is selected with probability $1 - \epsilon$, and a random action is selected with probability ϵ. The estimate updates as follows:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a \{Q(S', a)\} - Q(S, A)\right] \tag{2.18}$$

The update enhances the current estimate, $Q(S, A)$, in the direction of a new estimate by a step size α. The temporal difference error (TD-error), the difference between the previous estimate, $Q(S, A)$, and the new estimate made up of the single step reward and the best possible value of Q at the next state, $R_t + \gamma max_a Q(S', a)$, is defined as follows:

$$\delta_t = R + \gamma \max_a Q(S', a) - Q(S, A) \tag{2.19}$$

The method is referred to an off-policy method since the new estimate is based on the possibility of taking the action that maximizes Q rather than the action described by a fixed policy π.

The Q function becomes a look up table for discrete action and state spaces, and thus is referred to as a tabular technique. In the tabular case, Q-learning has been proved to be converged to the optimal state-value function [56]. The convergence proof requires visiting all states an infinite number of times, which is problematic to implement, hence methods like ϵ-greedy action selection are used to ensure state-space exploration.

2.3.4 N-step Temporal Difference Learning

As mentioned above, in a simple form of TD learning, TD-error (Eq. 2.20) is computed using a single step reward where $V(S_t)$ presents the estimation of the value function at the current state, and $R_{t+1} + \gamma V(S_{t+1})$ is the new estimation of value function at state S_{t+1}.

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{2.20}$$

However, in order to increase speed of the learning process, the agent needs to collect successive rewards for computing new estimation of the value function [50]. Eq. 2.21 presents the new form of the TD-error called N-step TD error that can be applied to any form of TD learning approach such as Q-learning (i.e. N-step Q-learning).

$$\delta_t = (R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(s_{t+n})) - V(s_t) \tag{2.21}$$

2.3.5 DNN as a Value Function Approximator

It is beneficial to use DNNs as an approximator of the value functions. There are various possible configurations to employ DNNs for approximating value functions, a brief overview of DNNs is discussed as follows. In this study, only fully connected feed-forward neural networks (FFNN) was employed. The state vector identified to the network's first layer as an input whereas the final layer presents the network's output and the prior layers are referred to as hidden layers. Depending on the application, there are various forms of activation functions to map the transformed input to a single value output. In this study, leaky rectified linear unit (LReLU) (also known as the parameterized rectified linear unit) was selected as it is easy to differentiate, does not suffer from vanishing gradients problem, and the leaky characteristic of the LReLU solve the problem of dying nodes (the dying ReLU refers to the problem when ReLU neurons become inactive and only output 0 for any input) [21]. The LReLU activation function is used for all hidden layers and a linear function is selected for the output layer as an activation function that simply passes along the matrix transformation and allows the output to take on any real number. In addition, in the case of action policy network output function, the sigmoid and hyperbolic tangent functions were used to ensure the bounds of the max/min release constraints.

2.3.6 RL Methods with Continuous Action Spaces

As previously stated, the tabular RL approaches suffer from the curse of dimensionality as the number of state and action variables grow and the state and action spaces become more finely discretized. To overcome this issue, the Q function should be estimated using a function (such as artificial neural networks) that can serve as an approximator. DNNs are considered universal function approximators, however recent studies demonstrated that employing non-linear functions as approximators shows instability [52]. Mnih et al., (2015) proposed two strategies to overcome this instability: a replay buffer and incremental updates, both of which reduce the impact of correlation between samples [38]. However, this method again can only be applied to discretized actions and suffers from the same dimensionality problem as the action space grows. There is also an embedded optimization step with the Q-learning update, which can be time consuming. Lillicrap et al., (2015) developed the deep deterministic policy gradient (DDPG) technique, which is able to use a continuous action space, by coupling the deterministic policy gradient (DPG) method with an advanced version of deep Q network (DQN) [31].

2.3.7 Experience Replay Buffer

Mnih et al., (2015) proposed experience reply to stabilize the RL process in the case of using DNNs as value functions approximator [38]. In this strategy, the agent's experiences will be memorized in a buffer, $\mathcal{D}$, with a size of $\mathcal{D}_{max}$. A single experience vector includes the current state, the selected action, the subsequent reward and the state transitions that occur as a result of the performed action (tuple of $[S, A, R, S']$). During the training the DNN, mini-batches of memories are randomly taken from the buffer and utilize to update decision action. The oldest memories diminish and new experiences store in the buffer once the buffer maximum capacity is reached.

2.3.8 Prioritized Replay

As mentioned, during the training, batches of prior experience are drawn from the replay buffer memory. The samples in these training batches are drawn randomly and uniformly from memory. However, regardless of their significance, this approach just replays transitions at the same frequency that they were originally experienced. Prioritized experience replay is an optimized form of replay buffer. The rationale behind prioritized replay is that not all experiences are equal when it comes to productive and efficient DRL. This strategy prioritizes experiences in order to replay important transitions more frequently and hence learn more efficiently.

2.3.9 Incremental Updates

The other technique proposed by Mnih et al., (2015) to stabilize the learning process is an incremental update in which a target DNN is employed to compute the temporal difference used for training the primary DNN [38]. The incremental update strategy reduces the correlation between the temporal difference and the policy that selects the action resulted in a particular reward. This strategy can be applied after a given number of updates to the primary DNN.

2.3.10 States, Decision Actions, and Reward Function

AIn reservoir operation problem the state variable specifies the system's attributes under consideration. Our proposed DRL approaches were implemented for a single reservoir system, and the first property of the state vector was the reservoir storage vector from the first month to the last month of a water year. The storage values were then normalized, with the limits of $[0, 1]$ corresponding to the storage levels which were designated as boundary states. This study also took into account the calendar date as a state variable similar to Castelletti et al. (2010) but with a variation to make the temporal dimension periodic [11]. This was done by representing the calendar date as a Cartesian transformation of the calendar date represented in polar coordinates on a circle with radius one and angle described by an indicator variable. These states were subsequently normalized so that all values fall into the range of $[0, 1]$. Each calendar month is assigned an indicator $i \in [0, 11]$, with January having a value $i = 0$. The state variable then represents the month as

$$[d_{1i}, d_{2i}] = \left[\frac{\cos(\frac{2\pi i}{12}) + 1}{2}, \frac{\sin(\frac{2\pi i}{12}) + 1}{2} \right] \tag{2.22}$$

Thus, the state variable is a three-dimensional vector $s_t = [d_{1t}, d_{2t}, c_t]$.

The action that the agent may take is the daily volume of release from the reservoir. This again is normalized to the range of $[0, 1]$ corresponding to the minimum and maximum releases based on the physical or operational constraints of the system. This is a single reservoir system, and thus the action variable is one-dimensional, $a_t = [a_t]$. The input to the policy function, $\pi(s)$, is the three-dimensional state variable. The input to the action-value function, $Q(s, a)$, is a 4-dimensional concatenation of the state and action variables:

$$x_t = [s_t, a_t] = [d_{1t}, d_{2t}, c_t, a_t] \tag{2.23}$$

One of the benefits of PGMs is that they are not dependent on the form of reward functions, and other functions could easily be substituted if found to align more closely with the stakeholders' objectives. There is a benefit to keeping the reward functions simple as it allows for a better conceptual understanding of the value functions resulting from a selected reward function. The reservoir system operational objectives can be grouped into four main categories: flood control, water supply, environmental flow, and hydropower. The value functions are based on the reward signal that the agent receives as a result of an action taken and the subsequent state transition. The reward signal must then provide information to the agent that correlates with the outcomes the stakeholders either seek or wish to avoid. Rewards were developed to consider the water supply and hydropower, and it is conditioned to meet flood control and minimum environmental flow. Also, a value of penalty value was chosen to ensure that spill exceeding the downstream flow capacity will be penalized far more heavily than supply deficit.

$$r_t = GP_t - {D_t}^2 + P_t \tag{2.24}$$

Where GP_t and D_t are respectively, generated power (GWh) and deficit (TAF), at time step t. P_t denotes penalty value (negative) for deviation from system requirements. A squared water supply deficit (to be minimized) is used in the reward function, reflecting the fact that large deficits are disproportionately more costly than small ones [53].

2.4 Policy Gradient Methods

Deep RL (DRL) is a new type of RL that has emerged as a result of advances in deep learning (DL) techniques. DRL can deal with high-dimensional inputs such as photos, recognize complicated patterns, and extract their features [2]. Policy gradient methods (PGMs) are a class of DRL approaches that use gradient descent (GD) to optimize parametrized policies in terms of expected return (long-term cumulative rewards). They overcome many problems plaguing traditional RL approaches, such as the lack of value function guarantees, the intractability problem deriving from uncertain state information, and the complexity arising from continuous states and actions.

All PGMs employed in this study are based on actor-critic networks which is a temporal difference (TD) version of policy gradient. Although the actor and critic are separate NNs, they are connected and collaborate constantly. An actor performs the action, while a critic evaluates the actor's performance. Based on the critic's gradient, the actor conducts actions in the environment. The critic collects information about the environment and assigns a reward value to the actor's proposed action. Generally, the actor and critic interactions determines how agent learns, which is based on a reward function that reinforces the learning [47]. The most relevant PGMs employed in the current study are discussed below.

2.4.1 Deep Deterministic Policy Gradient

DDPG is a model-free off-policy actor-critic algorithm resulted from coupling Deterministic Policy Gradient (DPG) and Deep Q-Network (DQN) [31]. DQN leverages experience replay and the frozen target network to stabilize Q-function learning. The original DQN operates in discrete space, while DDPG utilizes the actor-critic architecture to expand it to continuous space while learning a deterministic policy. DPG is an actor-critic technique very popular for continuous control problems, as it uses a separate actor network and exploit policy gradient to directly search policies in the continuous action space. This method presents a more efficient approach by reducing computations compared to traditional stochastic gradient methods [48]. Stochastic strategies integrating over both the action and state spaces are usually required to explore the entire action-state space. DPG reduces computations by integrating only over the state space, leads a deterministic policy. Exploration is ensured by making the algorithm off-policy and executing actions according to a stochastic Gaussian behavior policy during learning. In terms of sample efficiency, this technique effectively outperforms both on and off-policy stochastic gradient methods [48]. DDPG takes the actor-critic structure of DPG while implementing DQN learning method. In addition, DDPG introduced modifications to DPG to use DNNs as non-linear function approximators in both actor and critic structures [32]. Here, the math behind the two parts of DDPG (DPG and DQN) is presented as follows (mostly adapted from [1].

Learning an approximator begins with the Bellman equation. Eq. 2.25 shows the optimal action-value function described through Bellman equation is given by:

$$Q^*(s,a) = \underset{s' \sim P}{\mathrm{E}} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right] \tag{2.25}$$

where $s' \sim P$ denotes the new state, s', is sampled from a distribution $P(\cdot|s,a)$ by the environment. Supposing the approximator $Q_\phi(s,a)$ as a NN with parameters ϕ and a set of transitions $\mathcal{D} = (s,a,r,s',d)$, Eq. 2.26 presents the mean-squared error (MSE) of the Bellman function.

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d) \sim \mathcal{D}}{\mathrm{E}} \left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d) \max_{a'} Q_\phi(s',a') \right) \right)^2 \right] \tag{2.26}$$

where d is a binary variable indicating that whether the state s' is terminal.

In DQN part of DDPG, aim is to minimize Eq. 2.26 which is a loss function shows how close Q_ϕ comes to satisfy the Bellman equation. The loss function shows the error between the Q-function and the term target (Eq. 2.27).

$$r + \gamma(1-d)\max_{a'} Q_\phi(s', a') \tag{2.27}$$

DQN and DDPG employed two tricks in the process of minimizing the mentioned loss function including: (i) replay buffer and (ii) target networks. As it previously mentioned, replay buffer is a large enough space containing a wide range of experience that helps DDPG to have a stable behavior.

The process of minimizing loss would be unstable as the target term is relied on the same parameters we are going to train. The solution to this problem is to employ a second network called target network, ϕ_{targ}, which lags the first. Thus, we can use a set of parameters of the target network which come close to ϕ, but with a time delay. In DDPG algorithm, the target network updates using polyak averaging method (PAM) once per main network update (Eq. 2.28).

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi \tag{2.28}$$

where ρ is polyak hyperparameter ranging between 0 and 1 that needs to be optimized.

In the continuous action-spaces, computing the maximum over actions in the target is challenging. To deal with this, DDPG uses a target policy network (identified similar to target Q-function) to compute an action which approximately minimizes $Q_{\phi_{\text{targ}}}$. The resulting Q-learning with the stochastic gradient decent method is performed by minimizing the following loss function.

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \big(r + \gamma(1-d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))\big)\right)^2\right] \tag{2.29}$$

where $\mu_{\theta_{\text{targ}}}$ is the target policy.

In policy learning process of the DDPG, a deterministic policy, $\mu_\theta(s)$, should be learned to take actions that maximizes $Q_\phi(s,a)$. In a continuous action-spaces only a GDM can be used to solve Eq. 2.30 assuming the Q-function is differentiable with respect to action.

$$\max_{\theta} \mathop{\mathrm{E}}_{s\sim\mathcal{D}} \left[Q_\phi(s,\mu_\theta(s))\right] \tag{2.30}$$

To improve DDPG policy exploration, recent studies suggested adding a time-uncorrelated Gaussian noise to the actions during the training. However, to determine how well the policy exploits the learned information, it doesn't need to be added during the test period. A detailed explanation of the DDPG algorithm process presented in the following pseudocode.

2.4.2 Twin-Delayed DDPG

One of the DDPG's drawbacks is its tendency to overestimate the value function. This overestimation can spread through the training iterations and negatively impact the policy. In 2018, an extension to the DDPG called Twin-Delayed DDPG was introduced, aiming to shrink this effect by applying a couple of techniques [17]. The first one is clipped double Q-learning. In this technique, TD3 learns two twin Q-functions and chooses a pessimistic bound over the two during updating policy. The second technique is the delayed update of target and policy networks: It was found that in order to further reduce the variance in the presence of target networks, the policy required to be updated at a lower frequency than the Q-function; so that the Q-function error first reduces before utilizing it to update the policy. The authors suggested updating the policy at half the rate of the Q-function. The third technique is target policy smoothing. To deter the agent from selecting overestimated Q-function

Algorithm 1 DDPG, adapted from [1]

```
1: Input: initial policy parameters $\theta$, Q-function parameters $\phi$, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
3: repeat
4:    Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
5:    Execute $a$ in the environment
6:    Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
7:    Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
8:    If $s'$ is terminal, reset environment state.
9:    if it's time to update then
10:       for however many updates do
11:          Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:          Compute targets
                 $$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$
13:          Update Q-function by one step of gradient descent using
                 $$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} (Q_\phi(s, a) - y(r, s', d))^2$$
14:          Update policy by one step of gradient decent using
                 $$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_\phi(s, \mu_\theta(s))$$
15:          Update target networks with
                 $$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
                 $$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$
16:       end for
17:    end if
18: until convergence
```

values, a small amount of noise, clipped around the taken action, is added to the target action. This approach follows the idea of the SARSA update and enforces that similar actions should have similar values [42]. Here, the equations of TD3 method is elaborated (mostly adapted from [1].

Similar to DDPG in learning its single Q-function, TD3 concurrently learns two Q-functions Q_{ϕ_1} and Q_{ϕ_2} to minimize mean square Bellman error (MSBE) defined as loss function. A clipped noise should be added on each dimension of the actions to form Q-learning target relied on the target policy $\mu_{\theta_{\text{targ}}}$. Then the target action is bounded to the valid action range (Eq. 2.31).

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma) \tag{2.31}$$

This process also called target policy smoothing serves as a regularizer of the algorithm and resolve a particular failure mode happening in DDPG. In other words, TD3 smooths out the Q-function over similar actions which is what target policy smoothing is intended to do. Both Q-functions employ a single target, which is calculated by utilizing the Q-function that produces a smaller target value (Eq. 2.32). Then, both Q-functions are learned by regressing to the selected target which helps fend off overestimation in the Q-function (Eqs. 2.33 and 2.34). Thus, overestimation in the Q-function can be avoided by using the smaller Q-value for the target and regressing towards it.

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s')) \tag{2.32}$$

$$L(\phi_1, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}} \left[\left(Q_{\phi_1}(s, a) - y(r, s', d)\right)^2\right] \tag{2.33}$$

$$L(\phi_2, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}} \left[\left(Q_{\phi_2}(s, a) - y(r, s', d)\right)^2\right] \tag{2.34}$$

Finally, similar to DDPG, by maximizing Q_{ϕ_1} the policy would be learned (Eq. 2.35). However, in contrast to Q-functions, the policy updates much less frequently in TD3. This helps to mitigate the volatility that frequently occurs in DDPG as a result of how a policy update alters the target.

$$\max_{\theta} \mathop{\mathrm{E}}_{s \sim \mathcal{D}} [Q_{\phi_1}(s, \mu_\theta(s))] \tag{2.35}$$

Similar to DDPG, an uncorrelated mean-zero Gaussian noise should be added to the actions during the training to improve TD3 policy exploration. However, to determine how well the policy exploits the learned information, it doesn't need to be added during the test period. A detailed explanation of the TD3 algorithm process presented in the following pseudocode (Alg. 2).

2.4.3 Soft Actor Critic

SAC, another DDPG enhancement, was proposed by Haarnoja et al., (2018a; SAC18) as an off-policy algorithm based on maximum entropy [18]. Unlike DDPG and TD3, SAC uses a stochastic policy that is intrinsically more stable during learning while retaining the off-policy updating of DDPG for increased sample efficiency. SAC outperformed DDPG and proximal policy optimization (PPO) on many control tasks, demonstrating more stable learning with enhanced average return and sampling efficiency [18]. In this method, mean of the policy distribution was selected at evaluation time to make actions deterministic and ensure consistent performance.

Due to the stochastic nature of its policy, exploration is an inherent mechanism to SAC. An entropy term was added to the objective function (standard optimal policy expression) to control this mechanism, which is the measure of the randomness in its probability distribution. In the context of SAC, it measures the randomness in the

Algorithm 2 Twin Delayed DDPG, adapted from [1]

```
 1: Input: initial policy parameters $\theta$, Q-function parameters $\phi_1$, $\phi_2$, empty replay buffer $\mathcal{D}$
 2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
 3: repeat
 4:   Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
 5:   Execute $a$ in the environment
 6:   Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
 7:   Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
 8:   If $s'$ is terminal, reset environment state.
 9:   if it's time to update then
10:     for $j$ in range(however many updates) do
11:       Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:       Compute target actions
            $$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$
13:       Compute targets
            $$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$
14:       Update Q-functions by one step of gradient descent using
            $$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$
15:       if $j \mod \texttt{policy\_delay} = 0$ then
16:         Update policy by one step of gradient ascent using
              $$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$$
17:         Update target networks with
              $$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$
              $$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$
18:       end if
19:     end for
20:   end if
21: until convergence
```

policy at a given state. Incorporating it into the Q-function encourages exploration of high-entropy regions and avoids early convergence to sub-optimal solutions.

It is traded-off against the expected rewards with the temperature hyperparameter in the updated Bellman equation. In other words, this objective favors the most random policy that still achieves a high return. This creates an inherent exploration mechanism that also prevents premature convergence to local optima. Multiple control strategies that achieve a near-optimal reward are captured, allowing for more robustness to disturbances. To reduce DDPG's overestimation bias of the Q-function, SAC makes use of the double Q-function trick by learning two approximators and using a pessimistic bound over the two. The Q-function critic is modeled as a DNN, and the standard Bellman equation is modified with the entropy expression to obtain a recursive expression of the soft Q-function. The policy, or actor, is modeled as an m-dimensional multivariate Gaussian distribution with a diagonal covariance matrix. Its actions are passed to a squashing function to ensure they are defined on a finite bound. The mean vector and the covariance matrix are estimated for each state by a DNN. Unlike DDPG's deterministic policy, no target policy is needed as the policy's stochasticity has a smoothing effect. The stochasticity also means that the policy objective depends on the expectation over actions and is therefore non-differentiable. The authors also proposed a reparameterization trick using the known mean and standard deviation of the stochastic policy along with an independent noise vector and applying the squashing function.

It was found that the SAC algorithm is unstable with respect to the temperature hyperparameter. Optimal entropy is not only dependent on the learning task but also on the learning state of the policy. An updated version of SAC proposes automatically adjusting it based on another optimization problem [19].

SAC is recognized as one of the state-of-the-art DRL algorithms with a proven record of successful real-world applications on continuous tasks. It has consistently outperformed previous state-of-the-art DRL algorithms such as DDPG, PPO, and TD3 in terms of sample efficiency, partly thanks to its stochastic policy that encourages exploration and avoids early convergence to suboptimal policies. Here, in-detail explanation and equations of SAC method is elaborated as follows (mostly adapted from [1].

To begin explaining SAC, we should first discuss the entropy-regularized RL environment. The formulae for value functions in entropy-regularized RL are slightly different.

Entropy is a measure of how unpredictable a random variable is. If a coin is weighted such that it usually always comes up heads, it has a low entropy; if it is uniformly weighted, it has a 50% chance of either outcome. Considering x, a random variable with a probability mass function or density function P. The entropy H of x is calculated from its distribution P using the following equation.

$$H(P) = \underset{x \sim P}{\mathrm{E}} [-\log P(x)] \tag{2.36}$$

In entropy-regularized RL, the agent receives a bonus reward at each time step proportionate to the entropy of the policy at that time step. As a result, the RL problem becomes:

$$\pi^* = \arg\max_{\pi} \underset{\tau \sim \pi}{\mathrm{E}} \left[\sum_{t=0}^{\infty} \gamma^t \bigg(R(s_t, a_t, s_{t+1}) + \alpha H \left(\pi(\cdot|s_t) \right) \bigg) \right] \tag{2.37}$$

where $\alpha > 0$ denotes the trade-off coefficient. In this new setting, the value functions are defined slightly different and the entropy rewards from each time step are included in V^π as follows:

$$V^\pi(s) = \underset{\tau\sim\pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\Big)\middle| s_0 = s\right] \tag{2.38}$$

In addition, Q^π is updated to consider entropy rewards from each time step except the first:

$$Q^\pi(s,a) = \underset{\tau\sim\pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^t H\left(\pi(\cdot|s_t)\right)\middle| s_0 = s, a_0 = a\right] \tag{2.39}$$

Considering the above definitions, V^π and Q^π are associated by Eq. 2.40 and the Bellman equation for Q^π is presented by Eq. 2.41:

$$V^\pi(s) = \underset{a\sim\pi}{\mathrm{E}}\left[Q^\pi(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right) \tag{2.40}$$

$$\begin{aligned} Q^\pi(s,a) &= \underset{\substack{s'\sim P\\ a'\sim\pi}}{\mathrm{E}}\left[R(s,a,s') + \gamma\left(Q^\pi(s',a') + \alpha H\left(\pi(\cdot|s')\right)\right)\right] \\ &= \underset{s'\sim P}{\mathrm{E}}\left[R(s,a,s') + \gamma V^\pi(s')\right] \end{aligned} \tag{2.41}$$

SAC simultaneously learns a policy π_θ,as well as two Q-functions, Q_{ϕ_1} and Q_{ϕ_2}. SAC is now available in two versions: one with a fixed entropy regularization coefficient, α, and another with an entropy constraint enforced by varying α over the course of training. In this study both versions are employed to examine their performance on operating reservoir problem. In SAC, Q-functions are learned in a similar manner to TD3, with a few exceptions including: (i) a term from entropy regularization is also included in the target (ii) Instead of a target policy, the target uses next-state actions from the current policy (iii) As SAC adopts a stochastic policy, there is no explicit smoothing of target policy and the noise generated by that stochasticity is enough to provide the same result (TD3 trains a deterministic policy, therefore it smooths the next-state actions by injecting random noise).

In order to fully understand where the Q-loss comes from, let's first see how entropy regularization contribute to Q-loss. Considering recursive Bellman equation for the entropy-regularized Q^π:

$$\begin{aligned} Q^\pi(s,a) &= \underset{\substack{s'\sim P \\ a'\sim\pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma\left(Q^\pi(s',a') + \alpha H\left(\pi(\cdot|s')\right)\right)\right] \\ &= \underset{\substack{s'\sim P \\ a'\sim\pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma\left(Q^\pi(s',a') - \alpha \log \pi(a'|s')\right)\right] \end{aligned} \tag{2.42}$$

The right hand side is an expectation over next states which comes from replay buffer and next actions (comes from the current policy). Thus, it can be approximated by sampling as it is an expectation:

$$Q^\pi(s,a) \approx r + \gamma\left(Q^\pi(s',\tilde{a}') - \alpha\log\pi(\tilde{a}'|s')\right), \qquad \tilde{a}' \sim \pi(\cdot|s') \tag{2.43}$$

For each Q-function, SAC calculates the MSBE loss by approximating the target with the samples. Similar to TD3, SAC also employs the clipped double-Q trick and takes the smaller Q-value between the two Q approximators. To summarize, considering the target (Eq. 2.44), the loss function for Q-functions of the SAC is suggested as follows (Eq. 2.45.

$$y(r,s',d) = r+\gamma(1-d)\left(\min_{j=1,2} Q_{\phi_{\text{targ},j}}(s',\tilde{a}') - \alpha\log\pi_\theta(\tilde{a}'|s')\right), \qquad \tilde{a}' \sim \pi_\theta(\cdot|s') \tag{2.44}$$

$$L(\phi_i,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2\right] \tag{2.45}$$

Getting to know the rules of the game. The Policy in each state should aim to maximize expected future rewards and entropy. In other words, it should maximize the $V^{\pi}(s)$ presented below:

$$\begin{aligned} V^{\pi}(s) &= \underset{a\sim\pi}{\mathrm{E}}\left[Q^{\pi}(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right) \\ &= \underset{a\sim\pi}{\mathrm{E}}\left[Q^{\pi}(s,a) - \alpha \log \pi(a|s)\right] \end{aligned} \tag{2.46}$$

The reparameterization approach is applied to optimize the policy, which computes a deterministic function of state, policy parameters, and independent noise to draw a sample from $\pi_{\theta}(\cdot|s)$. To do so, a squashed Gaussian policy (e.g. tanh) suggested by author to draw samples:

$$\begin{aligned} V^{\pi}(s) &= \underset{a\sim\pi}{\mathrm{E}}\left[Q^{\pi}(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right) \\ &= \underset{a\sim\pi}{\mathrm{E}}\left[Q^{\pi}(s,a) - \alpha \log \pi(a|s)\right] \end{aligned} \tag{2.47}$$

In contrast to vanilla policy gradient (VPG), trust region policy optimization (TRPO) and PPO, tanh function employed in SAC policy squashes all actions to a specific range. The SAC reparameterization trick make it possible to rewrite the expectation over actions (the distribution relies on the policy parameters which is a pain point) into expectation over noise (the distribution is no longer depends on parameters).

$$\underset{a\sim\pi_{\theta}}{\mathrm{E}}\left[Q^{\pi_{\theta}}(s,a) - \alpha \log \pi_{\theta}(a|s)\right] = \underset{\xi\sim\mathcal{N}}{\mathrm{E}}\left[Q^{\pi_{\theta}}(s,\tilde{a}_{\theta}(s,\xi)) - \alpha \log \pi_{\theta}(\tilde{a}_{\theta}(s,\xi)|s)\right] \tag{2.48}$$

The next step is to substitute $Q^{\pi_{\theta}}$ with one of the function approximators (minimum of the two Q approximators) in order to obtain the policy loss. Therefore, the policy would be optimized based on Eq. 2.49 which is similar to DDPG and

TD3 policy optimization (except for the min-double-Q trick, the stochasticity, and the entropy term).

$$\max_{\theta} \underset{\substack{s\sim\mathcal{D} \\ \xi\sim\mathcal{N}}}{\mathrm{E}} \left[\min_{j=1,2} Q_{\phi_j}(s, \tilde{a}_{\theta}(s,\xi)) - \alpha \log \pi_{\theta}(\tilde{a}_{\theta}(s,\xi)|s) \right] \tag{2.49}$$

As mentioned, SAC is an on-policy algorithm which trains a stochastic policy leveraging entropy regularization. The entropy regularization coefficient, α, controls the trade-off between exploration and exploitation (higher α value indicates more exploration and vice versa). During the test period, to determine how well the policy exploits the learned information, the stochasticity should be removed and the actions mean should be used instead of sampling from the distribution. A detailed explanation of the TD3 algorithm process presented in the following pseudocode (Alg. 3).

2.5 Standard Operating Policy (SOP) under Flood Control Condition

The most straightforward operating policy for a single reservoir delivering water supplies downstream is to meet as much of the target demand as possible. If there isn't enough water to meet the demand, the reservoir is drained to release as close to the demand as possible. Excess water is held if it is available beyond the target release [45, 46]. Water spills downstream when the release target and available storage capacity (as shown in the flood control diagram, Fig. 2.1-b) are exceeded. Fig. 2.2 illustrates the release curve for the standard operating policy (SOP) under flood control.

SOP is rarely employed in actual reservoir operations, but it is frequently used

Algorithm 3 Soft Actor-Critic, adapted from [1]

```
1: Input: initial policy parameters $\theta$, Q-function parameters $\phi_1$, $\phi_2$, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
3: repeat
4:    Observe state $s$ and select action $a \sim \pi_\theta(\cdot|s)$
5:    Execute $a$ in the environment
6:    Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
7:    Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
8:    If $s'$ is terminal, reset environment state.
9:    if it's time to update then
10:       for $j$ in range(however many updates) do
11:          Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:          Compute targets for the Q functions:
                 $y(r, s', d) = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$
13:          Update Q-functions by one step of gradient descent using
                 $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$
14:          Update policy by one step of gradient ascent using
                 $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|\, s\right)\right),$
             where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt $\theta$ via the reparametrization trick.
15:          Update target networks with
                 $\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1-\rho)\phi_i \qquad \text{for } i = 1, 2$
16:       end for
17:    end if
18: until convergence
```

for operational studies and planning, especially firm yield studies. Although SOP provides a straightforward manner of determining release decisions, strict obedience

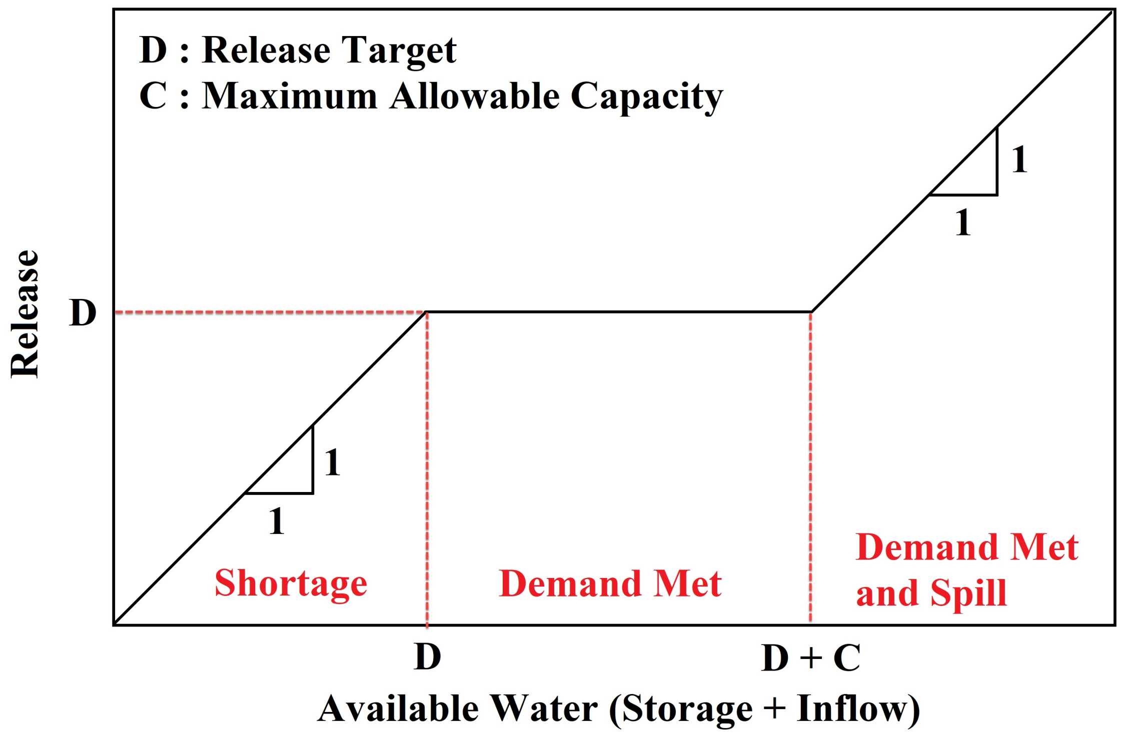

Figure 2.2: Standard operating policy under flood control condition (maximum allowable capacity, C, comes from flood control diagram, Fig. 2.1-b).

to this rule is practically rare due to a desire to keep at least some water to avoid extremely severe shortages. Operators would rarely empty a reservoir if the amount of available water was less than the target demand. Water conservation is often used in most reservoir systems to reduce demand before a reservoir runs empty.

2.6 Performance Criteria and Sustainability

Four different performance criteria were used to evaluate the model results and compare alternative management policies proposed by PGMs and SOP as well as the baseline conditions. The metrics include volumetric reliability, resilience, vulnerability, and maximum annual deficit. These performance criteria quantify the sustainability index of the water resources system for different alternatives. All performance criteria are based on water supplied deficiency, D_t, which is the difference between water demand and water supplied for each time period (Eq. 2.50):

$$D_t = \begin{cases} X_t^T - X_t^S, & \text{if} X_t^T > X_t^S \\ 0, & \text{if} X_t^T \leq X_t^S \end{cases} \tag{2.50}$$

Where X_t^T, X_t^S and D_t denote water demand, supplied water and deficit, respectively, at time step t. The deficit is positive when the water demand is more than water provided, and it will be zero when the water supplied is equal to or greater than water demand [33].

Volumetric reliability (Rel) is the total volume of water supplied divided by the total water demand [20, 54] (Eq. 2.51).

$$Rel = \frac{\sum\limits_{t=1}^{t=n} X_t^S}{\sum\limits_{t=1}^{t=n} X_t^T} \tag{2.51}$$

Resilience (Res) is a measure of the system capacity to adapt to changing conditions, defined as the probability that the system will remain in a non-failure state [27, 39, 43, 54] (Eq. 2.52):

$$Res = \frac{No.of times D_t = 0 follows D_t > 0}{No.of times D_t > \ 0 \ \ ocurred} \tag{2.52}$$

Vulnerability (Vul) demonstrates the average severity of a deficit during the total number of months simulated or, in others words, the likely damage from a failure event [3, 26, 43] (Eq. 2.53):

$$Vul = \frac{\left(\frac{\sum_{t=1}^{t=n} D_t}{No.of times D_t > 0 ocurred}\right)}{\sum_{t=1}^{t=n} X_t^T} \tag{2.53}$$

The maximum annual deficit ($Max.Deficit$), is the worst-case annual deficit for the entire period of simulation [39]. A dimensionless maximum deficit was calculated by dividing the maximum annual deficit by the annual water demand [43] (Eq. 2.54):

$$Max.Deficit = \frac{\max(D^i_{annual})}{X^T_{annual}} \tag{2.54}$$

The sustainability index (SI) is an index that measures the sustainability of water resources systems and can be used to estimate and compare the sustainability among proposed water policies [43]. Sandoval-Solis et al. (2011) proposed a variation of Loucks' SI where the index is defined as a geometric average of M performance criteria (C_m) (Eq. 2.55):

$$SI = \left[\prod_{m=1}^{M} C_m\right]^{\frac{1}{M}} \tag{2.55}$$

For this research, the sustainability index (SI) proposed for the Folsom Reservoir is formulated in Eq. 2.56.

$$SI = [Rel \times Res \times (1 - Vul) \times (1 - Max.Deficit)]^{1/4} \tag{2.56}$$

Chapter 3

Study Area

3.1 Folsom Reservoir Overview

The case study selected to investigate the application of reinforcement learning policy gradient methods in water infrastructure operating and management is Folsom Reservoir located in California, US.

Folsom Reservoir is a concrete gravity reservoir on the American River of Northern California in the United States, about 25 mi (40 km) northeast of Sacramento (Fig. 3.1). The dam is 340 ft (100 m) high and 1,400 ft (430 m) long, flanked by earthen wing dams. It was completed in 1955, officially opening the following year.

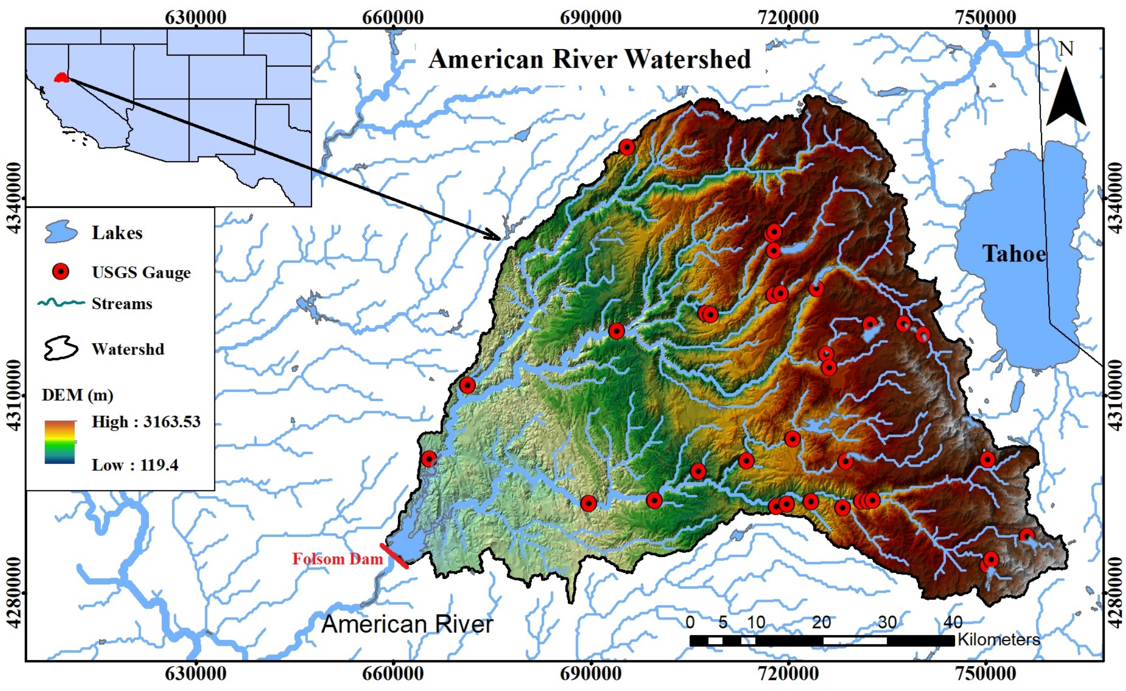

Figure 3.1: Folsom Lake and the American River Basin location in the northern California.

Located at the juncture of the north and south forks of the American River, the dam was built by the United States Army Corps of Engineers (USACE), and was transferred to the United States Bureau of Reclamation upon its completion.

The dam and its reservoir, Folsom Lake, are part of the Central Valley Project, a multipurpose project that provides flood control, hydroelectricity, and irrigation and municipal water supply. In order to increase Sacramento's flood protection to 200 year flood protection (meaning that the area is protected from a flood that has a 0.5 percent chance of occurring in any given year), the USACE recently constructed an auxiliary spillway, which was completed in October 2017 and enables Folsom Dam operators to increase outflows to prevent lake level from reaching or exceeding the height of the main dam gates [55].

3.2 Folsom Reservoir Specifications

Folsom Dam is located just north of the city of Folsom and consists of a 340 ft (100 m) high, 1,400 ft (430 m) long hollow core concrete gravity dam containing 1,170,000 cu yd (890,000 m3) of material. The dam is flanked by two earthen wing dikes, and the reservoir is held in place by an additional nine saddle dams on the west and southeast sides. The wing dams total a length of 8,800 ft (2,700 m), and the saddle dams measure 16,530 ft (5,040 m) long combined. The dam and appurtenant dikes total a length of 26,730 ft (8,150 m), more than 5 mi (8.0 km). Floodwaters are released by a spillway located on the main channel dam, controlled by eight radial gates with a capacity of 567,000 cfs (16,100 cms), as well as a set of outlet works with a capacity of 115,000 cfs (3,300 cms).

The impounded water behind the dam forms Folsom Lake, with a normal maximum pool of 977,000 acre-ft (1.205 cu-km) and a surcharge capacity of 110,000 acre ft (0.14 cu-km), for a total capacity of 1,087,000 acre ft (1.341 km3). The original capacity was 1,010,000 acre feet (1.25 km3), but it has been reduced somewhat due to sedimentation. At its maximum elevation of 480 ft (150 m), the reservoir covers 11,930 acres (4,830 ha), with 75 mi (121 km) of shoreline. The dam and reservoir control runoff from an area of 1,875 sq-mi (4,860 sq-km), or 87.6 percent of the 2,140 sq-mi (5,500 sq-km) American River watershed with most precipitation historically falling as snow at elevations above 5000 ft [9, 23]. The average amount of runoff entering the reservoir is 2,700,000 acre-feet (3.3 cu-km), forcing the release of 1,700,000 acre-feet (2.1 cu-km) for flood control.

Fig.3.2-a shows the historical variability of monthly inflows vs. demands over the period of 1955-2020. The inflow values were retrieved from two USGS gages located on the American River at Fair Oaks (USGS11446500, available since 1904) and North Fork American River (USGS11427000, available since 1941). Inflow data for 1995-2020 is provided by the U.S. Bureau of Reclamation and daily storage and release data through the California Data Exchange Center (CDEC; http://cdec.water.ca.gov). Fig. 3.2-b exhibits the average exceedance daily inflow to Folsom over 1955-2020 period. Fig. 3.2-c shows average daily releases over the water year, averaged during 1995-2016, which serve as a proxy for daily water demand (D_t) based on Herman and Giuliani (2018) [23]. The averages exclude flood control releases, which are defined as releases exceeding 12 TAF/day. A 25-day centered moving average is applied to smooth daily variability. In reality, Folsom water demand is far more complicated, as the reservoir operates in coordination with other CVP reservoirs to meet statewide urban, agricultural, and environmental needs. Folsom water demand also varies from year to year based on climate and economic conditions. However, the historical average releases demonstrated in Fig. 3.2-c reveals apparent seasonality changes, with a peak during the irrigation season, a reasonable simplification for this illustrative case study [23].

Folsom Power Plant is located on the north side of the river, at the base of the dam. It has three Francis turbines with a combined capacity of 198.72 megawatts

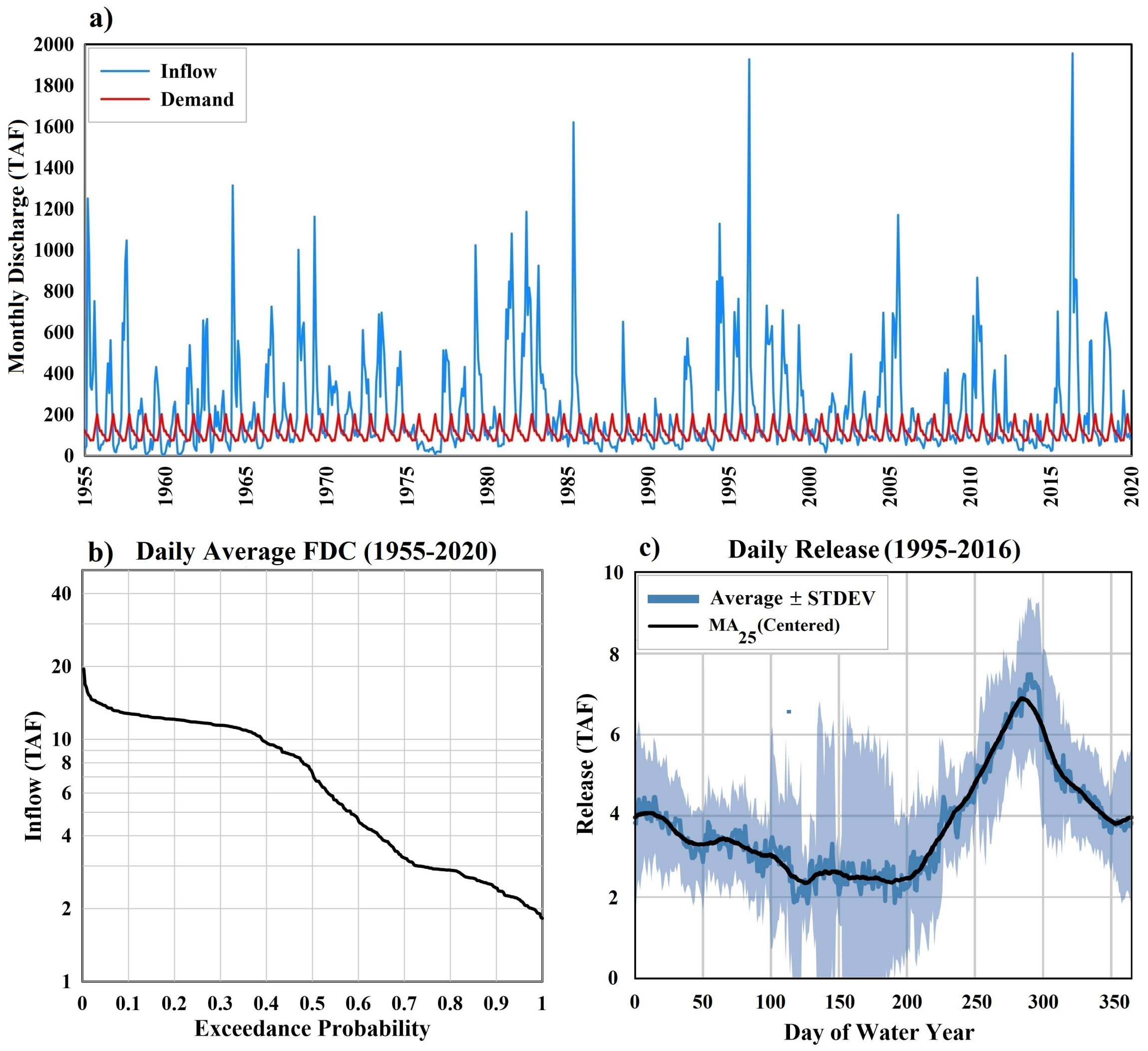

Figure 3.2: a) Historical variability of inflow to Folsom reservoir vs. demand during 1955-2020 period. b) Mean daily historical reservoir inflow exceedance curves. c) Mean daily releases from 1995 to 2016.

(MW), uprated from its original capacity of 162 MW in 1972. The power plant's electricity production is intermediate, between peaking and base load. It generally operates during the day, when the demand and price for electricity is the highest. The plant produces an average of 691,358 megawatt hours (MWh) each year.

3.3 Folsom Reservoir History

Folsom Dam was proposed as early as the 1930s under California's State Water Plan, in response to chronic flooding in low-lying Sacramento. The flood risk to the state capital had been exacerbated since the 1850s by hydraulic mining debris and the construction of levees to protect farms and towns, which reduced the channel capacity of the Sacramento and American Rivers. The current dam was originally authorized by Congress in 1944 as a 355,000 acre ft (0.438 cu km) flood control unit, and was reauthorized in 1949 as a 1,000,000 acre ft (1.2 cu km) multiple-purpose facility.

The current Folsom Dam replaced an earlier, smaller dam that had been completed in 1893 by Horatio Gates Livermore. The earlier dam had fed the Folsom Powerhouse, generating electricity that was transmitted to Sacramento over a 22 mi (35 km)-long distribution line, the longest electrical distribution system in the world at the time. The remains of the earlier dam can be seen downstream from the Folsom Lake Crossing.

Construction of the dam began in 1951 with preliminary excavations for the Folsom Power Plant. The primary contract was awarded to Savin Construction Corp. of East Hartford, Connecticut, and Merritt-Chapman & Scott of New York for $29.5 million, with oversight by the USACE. On October 29, 1952, the first concrete was poured for the foundation. Flooding washed out the temporary cofferdam three times in 1953, delaying work and causing damage to Nimbus Dam which was also under construction at the time. Water storage in Folsom Lake began in February 1955, and the final concrete in the main dam was poured on May 17, 1955. The first hydroelectric power was generated in September of that year. In order to acquire the necessary land for development of future Folsom Lake bed, the government had to relocate families on 142 properties, including the settlements of Mormon Island and

Salmon Falls.

Even before the dam was complete, it demonstrated its effectiveness as a flood control facility during the record storms of December 1955, which completely filled Folsom Lake in a matter of weeks, and preventing $20 million of property damage. The dam was officially dedicated on May 5, 1956, and operation was transferred to the Bureau of Reclamation on May 14.

3.3.1 Spillway Gate Failure

On the morning of July 17, 1995, the Folsom Dam power plant was shut down and spillway gate three was opened to maintain flows in the American River. As the gate was operated, a diagonal brace between the lowest and second lowest struts failed. The failure resulted in the uncontrolled release of nearly 40 percent of Folsom Lake and a flood of 40,000 cfs (1,100 cms) moving down the American River. The freshwater reaching San Francisco Bay was atypical for the summer season and confused Pacific salmon and striped bass, whose instincts told them that fall rains had arrived; they began their annual fall migrations months ahead of schedule.

The hydraulic load on this type of spillway gate (Tainter gate) is transmitted from the cylindrical skin plate, which is in contact with the reservoir, through a number of struts to a convergence at the trunnion hub. The hub collects the load from the struts and transfers it across an interface to the trunnion pin, which is stationary and is connected to the dam. When the gate is operated, the hub rotates around the pin. The struts are primarily compression members, but friction at the pin-hub interface induces a bending stress during gate operation. Typically, and in this case, the struts are oriented such that the trunnion friction stress is applied to the weak axis of the struts. In order to better handle these loads, the struts are connected with diagonal braces that take the stress as axial loads. At Folsom Dam, increasing corrosion at the pin-hub interface had raised the coefficient of friction and, therefore, the bending stress in the strut and the axial force in the brace. The capacity of the brace connection was exceeded and it failed. This caused the load to redistribute and the failure progressed, eventually buckling the struts.

After a year-long investigation, the Bureau of Reclamation attributed the failure to a design flaw: the Corps of Engineers, which designed the dam, did not consider trunnion friction (at the pin-hub interface) in the gate analyses. While this is true, this was one of five identical service gates that operated under the same circumstances for nearly 40 years without problems being observed. This suggests that the failure resulted from a condition that changed over time; specifically, there was a gradual increase in the coefficient of friction at the pin-hub interface. While one would expect maintenance frequency to increase as a gate ages, Reclamation decreased the frequency of regular maintenance and lubrication over time due to budget constraints. In addition, the lubricant used by Reclamation did not conform to the Corps' original design specifications; it was a new, environmentally-friendly lubricant that was not sufficiently waterproof, allowing water to enter the pin-hub interface and cause the corrosion that resulted in increased friction.

This failure caused no fatalities and it had a significant positive impact on the dam industry. A renewed focus was placed on maintenance and monitoring of radial gates, many of which were retrofitted to strengthen struts and bracing and ensure sufficient lubrication.

3.4 Folsom Reservoir Safety

3.4.1 Security

After the 2001 terrorist attacks, the Bureau of Reclamation analyzed potential targets for vulnerability and measures that could be taken to eliminate or reduce possible threats. With 500,000 residents in the vicinity of the Folsom Dam, the possibility of an attack on the dam was great enough concern for Bureau officials to close Folsom Dam Road. The road over the dam had been a major artery for the city of Folsom. With its closure, traffic became severely congested during rush hour. The impact was so great that residents and city officials petitioned the federal government to reconsider the road closure, which the government initially considered. Continued security concerns prevented them from re-opening the road and a new bridge, named Folsom Lake Crossing, was constructed and opened on March 28, 2009.

3.4.2 Flood Risk

During a severe storm in December 1964, the inflow into Folsom Lake reached a record high of 280,000 cfs (7,900 cms), with a river release of 115,000 cfs (3,300 cms).

In February 1986, nearly 500,000 people faced the possibility of flooding when engineers at Folsom Dam were forced to open the spillway gates after heavy rains. The flooding was made worse by the failure of the Auburn Dam cofferdam upstream which released an extra 100,000 acre feet (120,000,000 m^3) into the American River. A peak flow of 250,000 cfs (7,100 cms) entered Folsom Lake, forcing operators at Folsom Dam to open all the spillway gates, releasing 130,000 cfs (3,700 cms) into the American River. This was 15,000 cfs (420 cms) above the safe capacity of downstream levees.

Although the dam and the Sacramento levee system held without major damage, the requisite winter flood control space was increased 50%, from 400,000 to 600,000 acre feet, to protect against future floods. In addition, about 33,000 acre feet (41,000,000 m^3) of sediment carried down from the mountains was deposited in Folsom Lake, considerably reducing its capacity. The consequence was a reduced capacity to store winter rainfall for summer use. Folsom Dam may have prevented as much as $4.7 billion in damages in 1986 alone.

The New Year's Day storm of 1997 was the most severe in recent history, with a total inflow of 1 million acre feet (equal to the entire capacity of Folsom Lake) over a 5-day period. However, this time the Bureau of Reclamation was able to limit releases to less than 110,000 cfs (3,100 cms). The 1997 storm was a classic example of a "rain on snow" event, during which a warm tropical storm melted existing snowpack at lower and middle elevations, effectively doubling the volume of runoff. Prior to the New Year's storm, the winter of December 1996 had also been one of the wettest ever recorded, saturating the ground and depositing a considerable amount of snow.

The Bureau of Reclamation's Safety of Dams Program determined the risk of flooding in the Sacramento area made it one of the most at-risk communities in the United States.

Two projects to increase flood protection are currently underway. The first will raise the surrounding dikes by 7 feet (2.1 m) to increase flood protection. The second, a new spillway, is designed to handle the runoff from large storms and snowmelt floods that might cause damage in the region. The new spillway is built with gates 50 ft (15 m) lower than the existing spillway, allowing for more efficient evacuation of reservoir storage before flooding events.

Chapter 4

Results and Discussion

4.1 Overview

This study developed various PGMs to solve DP problem for the Folsom Reservoir system in California, US. PGMs were developed to obtain the monthly optimal rule curve over the period of 65 years (1955-2020). Their performance was tested using various performance metrics and results were compared with base condition and SOP under flood control. This chapter contains four subsections, including (i) network structure, (ii) hyperparameter tuning, (iii) relation between state variables and decision actions, (iv) overall comparison of RL agents and finally (v) performance assessment and sustainability.

4.2 Networks Structure

The PGMs' value networks (both main and target) and the soft Q-networks are fully connected neural networks, each with 256 hidden layers, leaky rectified linear unit (LReLU) activation function, and an output layer with a linear activation function. The number of neurons in all hidden layers is a hyperparameter to be set by the user. The policy network has two outputs, the mean and the logarithm of the standard deviation (clamped to be in a sane region). These are used for a reparameterization to ensure that the sampling from the policy is differential and the errors can be appropriately backpropagated, leading to faster convergence. The action taken from a given state is obtained from the policy function by sampling noise from a standard normal distribution, multiplying it with the standard deviation, adding it to the mean, and then transform it using a sigmoid activation function to ensure the action between 0 and 1.

For all PGMs developed in this study, the networks updated policy decision by performing three general steps (i) initialization of the networks, (ii) initialization of the environment, and (iii) training loop. At the beginning of each episode for the 1955-2020 period, the environment was reset, and initial reservoir storage was assigned. For every monthly time step, the RL model chose an action from the policy (or randomly in the early exploration phase) and computed the reward and subsequent state variables. In this research, Adaptive Moment Estimation (Adam) [25] was selected as an optimization algorithm to train all models. Other details such as networks architecture were held fixed across conditions. Next, the model saves the obtained vector containing state variables, action, and associated reward to the memory buffer. Then, networks weights will be updated using memory buffer. The networks' updating process has the following steps: (i) prediction of Q-functions, value and policy networks (for all states in the batch and their suggested actions), (ii) evaluate the policy network to get the next states, (iii) prediction of the target value network, (iv) calculate Q-functions and value network losses and do one back-propagation (BP; update weights), (v) adjust the target value function using the next step Q-value, (vi) compute the target value function loss and do one BP, (vii) calculate the policy network loss and one BP, and (viii) update target value network. Once the learning episodes are completed, the model weights will be saved.

4.3 Hyperparameters Tuning

In the DRL modeling, the choice of the network architecture and optimizer along with the hyperparameters can have a dramatic effect on the final performance. The hyperparameters control how much the weights of DRL models can be adjusted with respect the loss gradient. An agent was identified based on different methods and network structures hardcoded and not considered hyperparameters. For any method, the agent takes action according to the action policy $\mu(s)$. At the beginning of a learning session, the experience agents are uniformly distributed in the state spaces (both in the storage and time dimensions). To set the learning rate, we first recorded the learning rate at each iteration and plotted the learning rate (log) against loss. As the learning rate increased, there was a point where the loss stopped decreasing and instead started to increase. The learning process was conducted for 30,000 episodes (iterations) of 780 time steps (monthly). After each time step, the experience $[s, a, r, s']$ of each agent is stored in the memory buffer. The stochastic gradient updates were then applied to the parameters of both actor and critic networks. To track the learning process and whether the process was converging, it was decided to observe the mean value of the value function. Table 4.1 presents the optimal values for hyperparameters of each method. The total training time, including the function evaluations for each technique, was approximately 25 hours on a Linux High-performance Computing (HPC) machine with 16-Core (2.4 GHz) processors, 256 GB RAM, and 2-Core Nvidia P100 GPU processors. Generally, the models converged, though with some level of noise always present. The results array was stored after each evaluation, and the best array among the last ten iterations was selected to get the output of the learning process.

Table 4.1: The optimal values of PGMs hyperparameters identified based on a trial-and-error process

Model	**Gamma**	**Tau**	**Buffer Size**	**Critic Learning Rate**	**Actor Learning Rate**	**Batch Size**	**Q Learning Rate**	**Alpha**
DDPG	0.99	0.015	1e6	1e-4	1e-3	64	-	-
TD3	0.99	0.015	1e6	1e-4	1e-3	64	-	-
SAC18	0.99	0.015	1e6	1e-3	3e-3	64	3e-3	0.2
SAC19	0.99	0.015	1e6	1e-3	3e-3	64	3e-3	0.2

As mentioned, Adam optimizer was selected as an optimization algorithm to train all models. Adam combines the advantages of two SGD extensions (RMSProp; Root Mean Square Propagation, and AdaGrad; Adaptive Gradient Algorithm) and computes individual adaptive learning rates for different parameters. In order to fairly compare the results across methods, all architectural details that are not specific to the method under consideration were held fixed across conditions. We noted a lower learning rate for the policy network than the value functions to collect experience at a faster rate than the policy adapts to the experience.

Interactions between reservoir system model and PGMs are processed through OpenAI Gym [8] environment, a standardized framework for RL and environment interaction.

4.4 State vs Decision Action Variables

All the operating policies designed by the DDPG, TD3, SAC18 and SAC19 were simulated under historical conditions (measured reservoir inflows, evapotranspiration rates, and the Folsom Reservoir properties) over the time horizon 1955-2020. Fig. 4.1 synthetically illustrated the state variables along with the resulting actions of the four PGMs employed in this study. In all cases, the agents were trained for 30,000 episodes corresponding to 65 years of monthly training data. The agents successfully learned a meaningful policy for reservoir release decision, given the limited state information. When the state variables associated with storage (which is dependent on the inflow value) and the time are dominated by fluctuations, the resulting actions partially mimic the states and shows a range of fluctuations in all methods. Considering the policy actions provided in Fig. 4.1-a, TD3 and SAC18 mostly presented the higher and lower extreme policy actions, respectively, while the SAC19 and DDPG provided moderate decision actions.

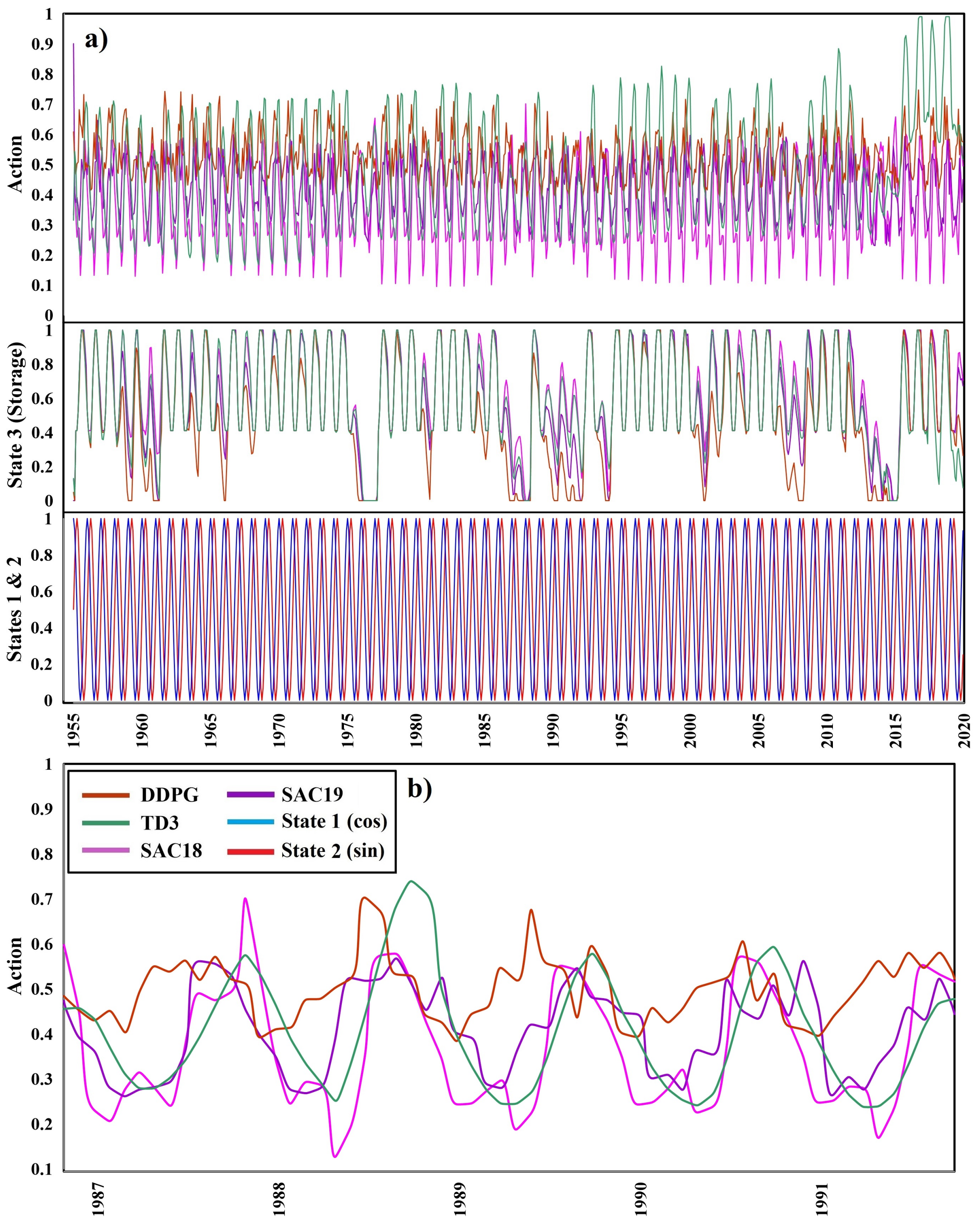

Figure 4.1: a) The relationship between the state variables and optimal policy actions taken by the PGMs. b) Optimal policy actions during the dry period (1987-1992) for the Folsom Reservoir.

Concerning the optimal policy actions provided by each method, as expected, the DDPG method overestimated the value function and revealed a lower range of variability than its variants. On the other hand, TD3 illustrated a smoother sequence of decision policy action with a higher variability range, demonstrating the model's ability to react in different conditions quickly. It is interesting to note that there are more fluctuations in the actions taken by both SAC18 and SAC19 methods than DDPG and TD3. This behavior is due to entropy optimization, where the SAC agent automatically optimized the entropy and increase the random exploration for regions of the parameter space with large fluctuations. This behavior is part of the reason why SAC is ideal for stochastic dominated environments where the actor learning performs the same action and fail miserably when conditions change. Fig. 4.1-b presents the policy actions for a 5-year dry period (1987-1992) to illustrate the different model's optimal policy when inflow to the Folsom Reservoir significantly decreased over time. The 5-year dry period was identified based on the 5-year monthly moving average of reservoir inflow over 65 years of simulation.

4.5 DRL Results

Fig. 4.2 presents the rewards associated with the optimal policy actions provided by each method. As mentioned before, the reward (or penalty) in each time step originated from the supply water deficiency and generated power amounts based on Eq. 2.24. The agent's goal is to maximize the cumulative rewards and learn by adjusting its policy (the agent's strategy) based on the obtained rewards. The results for the entire simulation period show both SAC18 and SAC19 with the cumulative rewards equal to -426409 and -426077, respectively (see Table 4.2) outperform TD3 and DDPG. This also can be realized by looking into reward values and its components

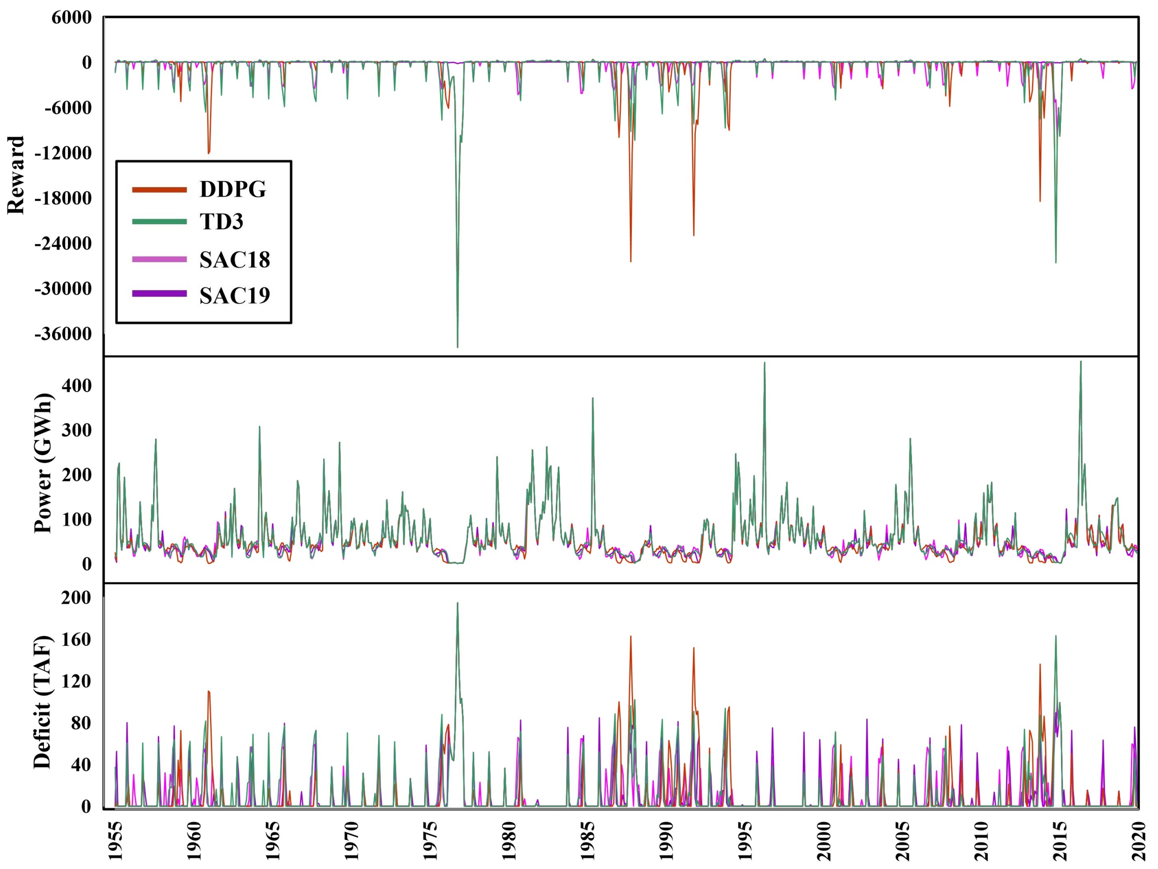

Figure 4.2: Rewards and its components as a result of policy actions taken by each method.

presented in Fig 4.2.

To evaluate optimal policy actions provided by each method, Fig. 4.3 illustrated the monthly release values optimized by each method as well as the SOP and historical releases (base conditions). The monthly release values during the 5-year dry period revealed that the DRLs' agents perform more reliable decision than the base conditions with respect to meeting the demands.

As noted, during wet years, the PGMs' agents increased release amounts in anticipation of high inflow that led to the lowest water level and also reduced the

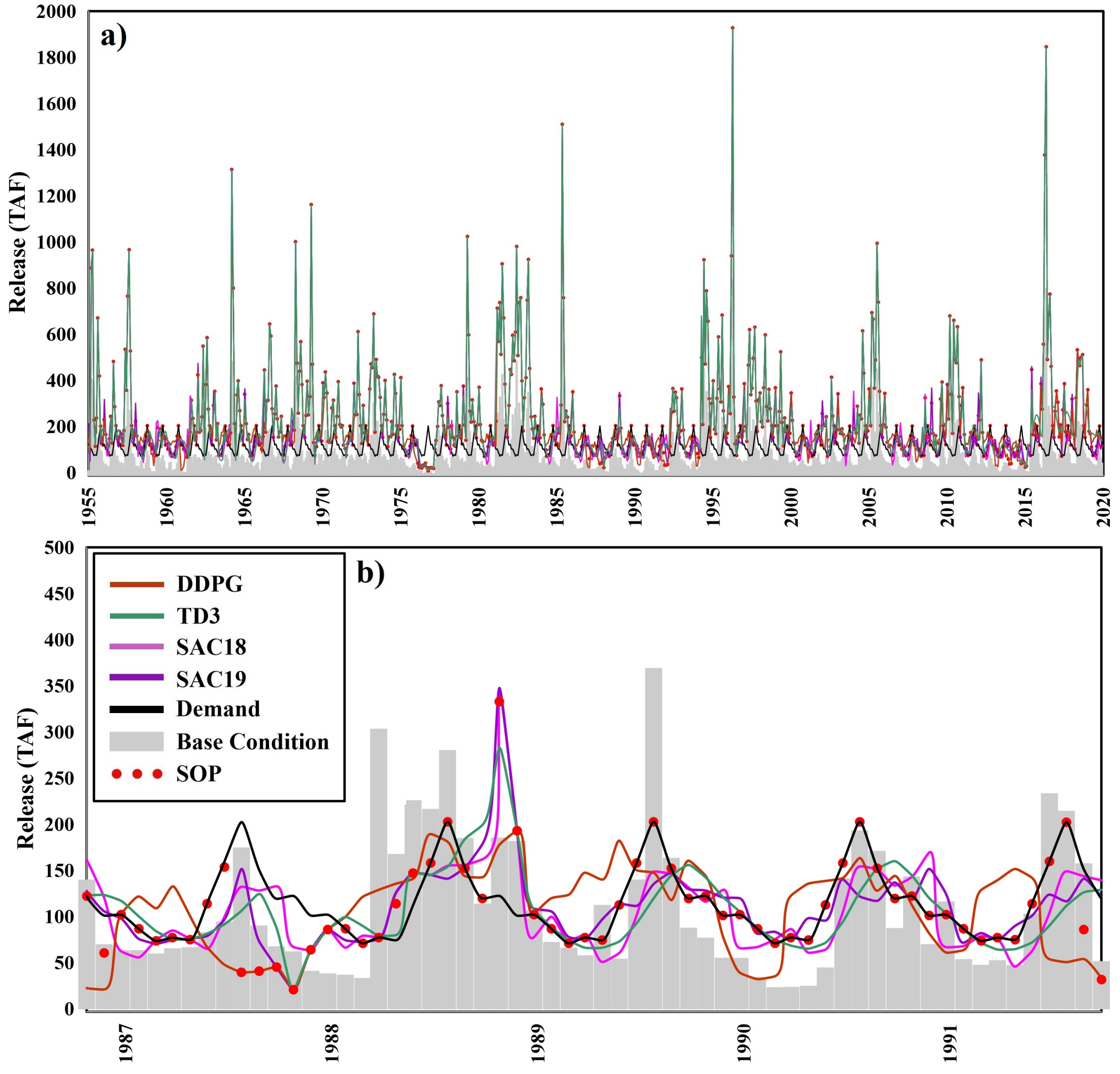

Figure 4.3: a) Monthly release values obtained from optimal policy actions provided by each PGMs. b) Monthly release values during the 5-year dry period (1987-1992).

volume of spilled water over the year (see Fig. 4.3). Contradictory during dry years, the PGMs' agents released less water to maintain high water levels in anticipation of low flow conditions in the upcoming time steps.

Fig. 4.4 illustrated the Folsom monthly storage levels during the simulation period as well as during the 5-year dry period to evaluate the state of the Folsom Reservoir based on different policies identified by PGMs, SOP, and base conditions. Results indicated that TD3 and both SAC18 and SAC19 predicted a higher reservoir storage level during the dry period than base conditions and SOP. As stated before, SOP releases water as close to the delivery target as possible, saving only surplus water for future delivery. As discussed, SOP is practical during periods of operation when inflow is plentiful. However, it neglects to consider potential shortage vulnerability during later periods, and strict obedience to this rule is practically rare due to a desire to maintain reservoir average level to avoid extremely severe shortages.

To evaluate the reliability of different methods in the case of supplying demands, Fig. 4.5 presents the annual deficit (%) calculated based on the optimal decision actions provided by each method during the simulation period. As illustrated in the Fig. 4.5, all PGMs excluding the DDPG successfully minimized the deficiency and performed relatively well compared to the base condition. As expected, the maximum deficiency in supplying water demands occurred during dry period (1987-1992).

Fig. 4.6 illustrated different methods performances in generating power that is totally depend on the reservoir state variable (head of water over turbine). As shown, the PGMs' agents learned to release the majority of water when there is a high level of water over the turbine to maximize hydropower generation. This implies that DRL is flexible with learning the objectives and provided varied releases to maximize the total hydropower benefit in response to dynamic inflow conditions. In contrast, SOP generated releases with less variation and could not adjust outflows due to its routine

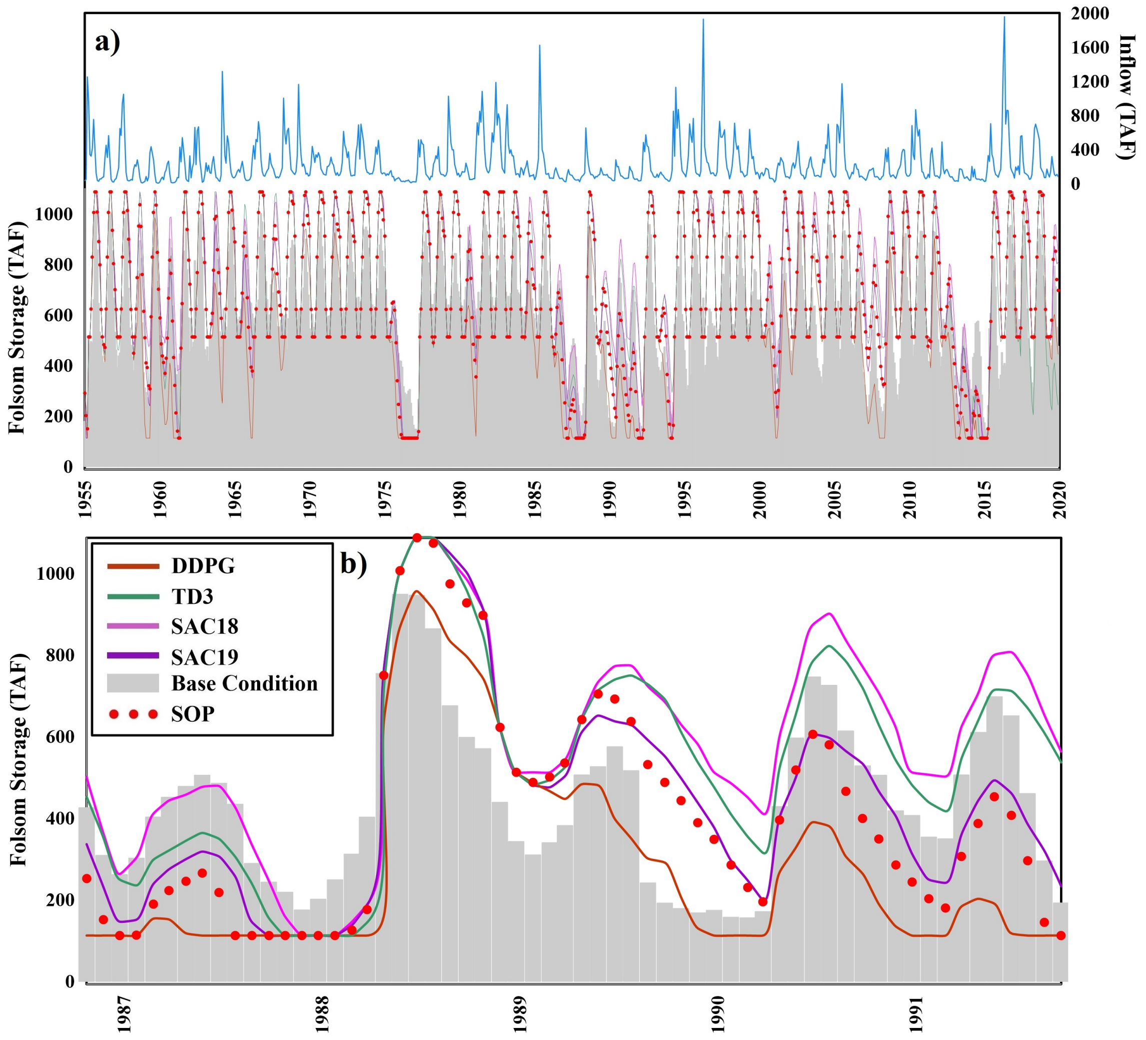

Figure 4.4: a) Folsom monthly storage amounts identified by different methods as well as base conditions during the simulation period. b) Reservoir storage levels during the 5-year dry period.

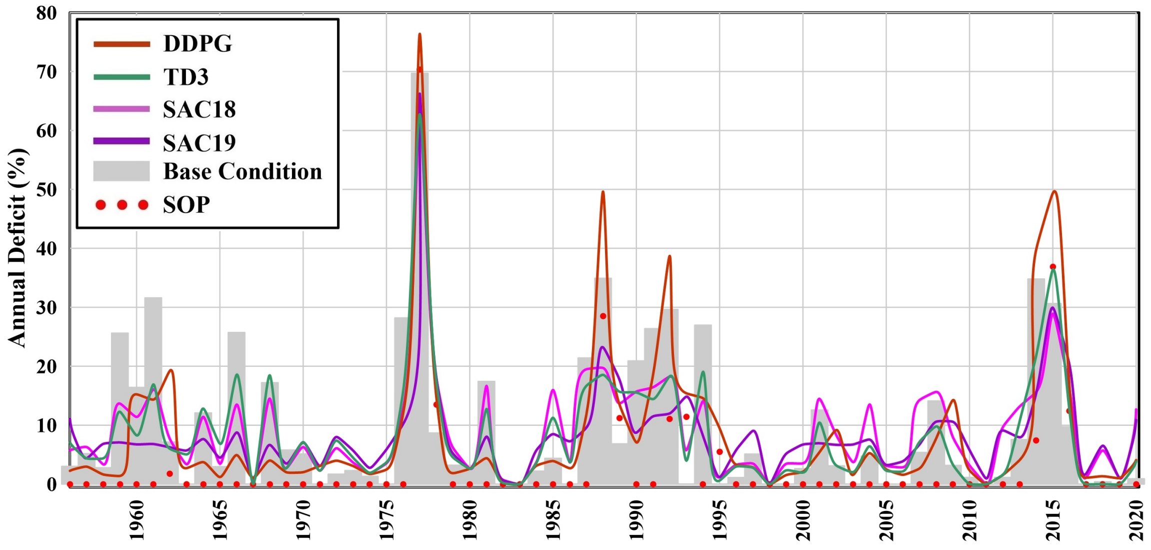

Figure 4.5: Annual deficit (%) calculated based on the optimal decision actions provided by each method vs SOP and base conditions.

operations procedure.

4.6 Performance Assessment and Sustainability

Table 4.2 presents the performance criteria such as reliability, resilience, vulnerability (RRV), and maximum deficit, as well as annual average generated power for DDPG, TD3, SAC18, SAC19, SOP, and base conditions. The cumulative rewards over the simulation period is also provided for the RL methods in Table 4.2.

In terms of different sustainability factors SOP, due to its policy's nature, delivered water as close as to the target demand to show more reliable (volumetric) results in supplying water demand than the other methods and base conditions. Overall, all the PGMs showed similar results to the base conditions in the case of volumetric reliability of supplying demands. However, PGMs presented promising results in the case of resilience and robustness, which shows the ability of the system

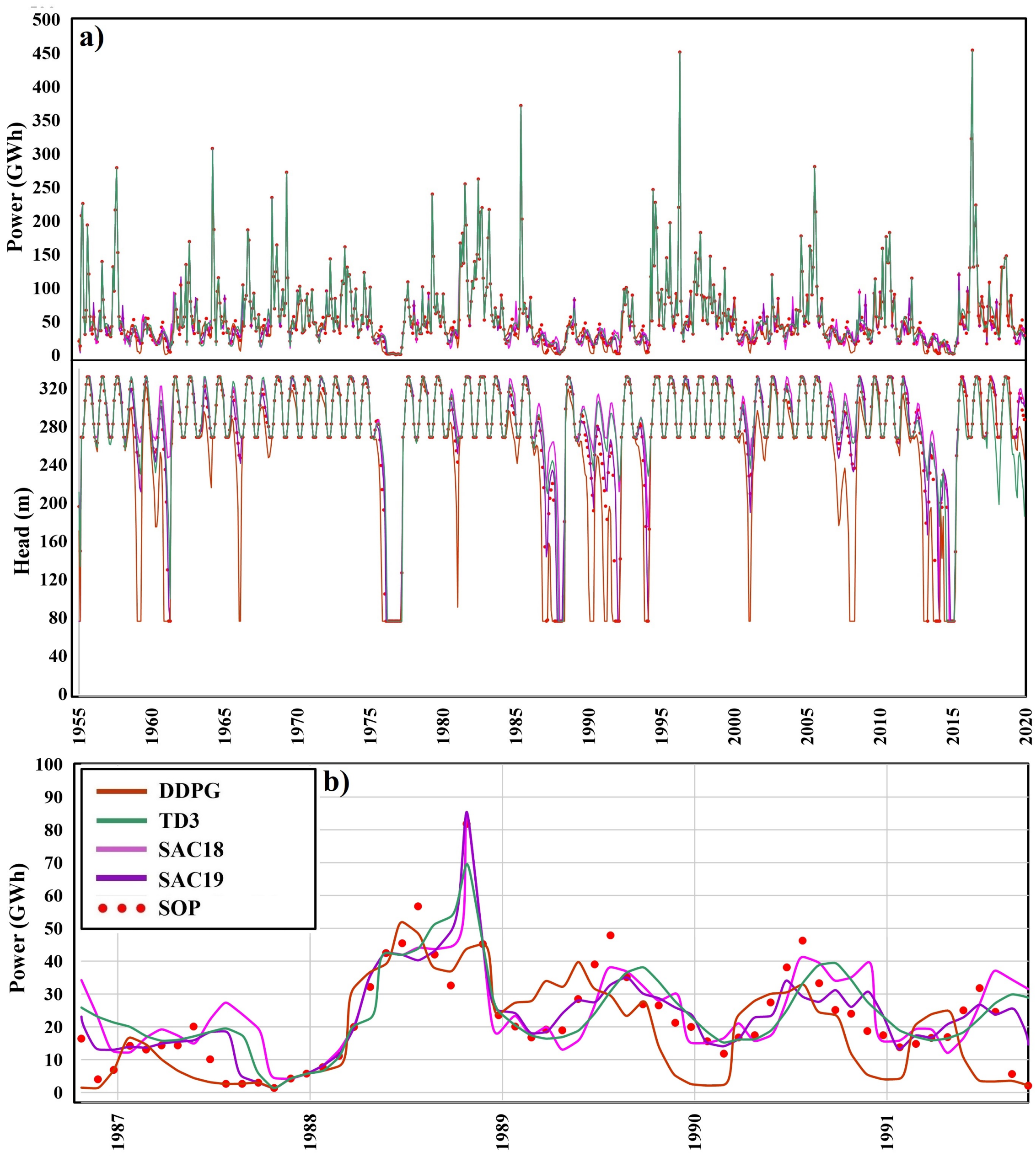

Figure 4.6: a) Generated power based on the suggested operating policy by each method as well as related reservoir state variable; head of water over the turbine. b) Generated power over the dry period (1987-1992).

Table 4.2: Performance criteria results for employed PGMs as well as base condition and SOP.

Method	**Reliability (Volume)**	**Resilience**	**Vulnerability**	**Max Annual Deficit (%)**	**Sustainability Index**	**Ave. Annual Power Production (GWh)**	**Cum. Rewards**
DDPG	0.91	0.39	5.18E-04	0.76	0.54	683.60	-556,289
TD3	0.91	0.37	4.52E-04	**0.62**	0.60	705.86	-459,503
SAC18	0.91	**0.45**	4.35E-04	0.63	**0.62**	**708.76**	-426,409
SAC19	0.91	0.38	**3.74E-04**	0.66	0.59	701.52	**-426,077**
SOP	**0.97**	0.23	8.09E-04	0.71	0.50	700.46	-
Baseline	0.90	0.27	3.96E-04	0.70	0.56	620.00	-

to recover and bounce back from a failure compared to the SOP and base condition. If failures are long-lasting events and system recovery is slow, this may cause serious consequences. Among multiple PGMs used in this study, SAC18 shows the most promising results in the case of robustness. In addition, the vulnerability index is computed to see how severe the consequences of a failure in different policies could be. As expected, the results show the operating policy suggested by SOP is the most vulnerable among all methods as its operation is along with severe deficiency. While SAC19 with the minimum vulnerability index shows more acceptable failure consequences against harsh conditions and dry periods. Comparing different PGMs actions policy, it can be noted that DDPG with the maximum annual deficit around 76 percent has quite poor performances even after a quite large number of iterations (30,000) while TD3 maintained the lowest maximum annual deficit (around 62 percent). Combining all four aforementioned indices into sustainability index, SAC18 method provided the best policy in terms of sustainability performance assessment. Also, in terms of generating hydropower, again SAC18 provides a better policy compared to the other operating policies. However, considering the base condition in generating power (average annual production of 620 GWh based on USBR Environmental Water Account report), all the PGMs policies show promising results in providing an optimal policy for maximizing power production.

From an optimization problem point of view, SAC19 found the best solution compared to the other methods with respect to the maximum value of the objective function (Cum. Rewards). However, SAC18 results is also promising, with slightly less objective function value compared to the SAC19.

Chapter 5

Conclusion and Insights for Future Work

Stochastic optimization is one of the primary challenges in integrated basin-wide water resources system management. Although various DP and SDP techniques have been developed in recent years, they have all been plagued by major issues such as the curse of modeling and the curse of dimensionality, preventing their application to large-scale water systems. It is interesting to note that traditional RL approaches (or discrete state and action space methods) can address some of the issues associated with classic DP and SDP methods; but, in larger-scale problems, they are still hampered by the curse of dimensionality. Thanks to the DNN, recently, various DRL-based techniques with continuous state and action spaces have been proposed which are able to deal with complicated control optimization problems without any simplification and approximation.

This study presents a generalized framework for creating an intelligent agent for reservoir systems management and control leveraging DRL methodologies. While DRL has been used successfully in the computer science communities, to our knowledge, this is the first instance for which it has been explicitly adopted for the control of reservoir operating systems. Different variants of PGMs such as DDPG, TD3, SAC18, and SAC19, applied to solve DP problem and tackle dimensionality issues without requiring any model simplifications and design optimal operating policy for the Folsom Reservoir system in California, US. The results were then compared to the SOP and baseline conditions using different performance criteria and sustainability indices. The models were evaluated using 65 years of historical data from the Folsom Reservoir system. The proposed PGMs relied on the continuous action spaces to sample actions from an appropriately parameterized Gaussian distribution. The agents can interact with any simulation model of the environment, utilize any user-defined reward function, and evaluate a wide array of options and variations of DRL. We showed that among different PGMs, both versions of the SAC (SAC18 and SAC19) are competitive and is able to outperform other policy methods on high-dimensional control tasks. Indeed, SAC18 effectively produced long-term strategies for the use of the stored water, focusing on mitigating water supply deficiency, more specifically during drought periods, flood management and control during wet seasons, and increasing annual hydropower production. Overall, the use of historical datasets for agent learning indicated that DRL approaches, and the agents' design as presented could adapt to stochastic conditions and function approximation through the use of DNN was shown to be a viable alternative to the tabular approach. The results from the employed methods were consistent and reflect our intuitive knowledge of reservoir operation. The study's findings also revealed that RL agents were able to learn and find the best strategies for reservoir system management without explicit knowledge of the underlying probabilistic models governing the stochastic hydrometeorological behavior of the system. Function approximation through the use of DNN was shown to be a viable alternative to the tabular approach.

Given the recent emergence and popularity of DRL, much research still remains to be conducted to understand if it has the potential to be a viable technique/tool for system management. Our research reveals a number of advantages and challenges related to this task. The ability to simply pass the learning task and apply it to policies without any concerns about complexities, non-linearities, and formulations that sometimes hinder other control approaches, appears to be the major advantage of utilizing DRL for operating reservoir systems. However, there are a number of important caveats, such as the challenges in selecting function approximators, determining the complexity of the control problem, and dealing with real-world implementation concerns. Contrary to the framework and learning functions formulations, the implementation may be hampered by inherent complexity and objectives of control problems. We demonstrated that an DRL agent can efficiently manage a single reservoir system but that simultaneously handling multiple reservoirs is challenging. The rise in the number of states and actions that must be represented using the NNs is one of the primary challenges. While more computational time may alleviate this issue, the NN structure may also need to be modified. In a cascade reservoirs system scenario, actions taken at one reservoir may impact another reservoir later on. As a result, the agent would benefit from a planning-based approach that took into account both current and future states. Such planning-based strategies have been developed in the RL literature and should be examined to realize if they can enhance policy decision and modeling performance [14, 16].

RL has previously been attempted in reservoir operating and management; however, those studies applied Q-learning and consequently discrete state/action spaces; PGMs employed in this study allow for continuous state/action spaces and are more stable towards stochastic variation. The framework is flexible, and the RL methods can be easily applied to different time scales or extended with additional constraints, e.g., cascading reservoirs, minimum power production requirements, and non-linear production functions. In addition, the DNN structure can be updated to take temporal aspects into account (e.g., recurrent neural networks; RNNs) or structured information in the form of graphs. In the case of cascading reservoirs, further constraints must be considered in the environment part of the framework which is defined through OpenAI Gym. Then, the problem can be solved by employing multiple agents associated with different reservoirs to discover optimal policy actions. It is also important to note that, due to the increased size of the action space, it expects that the model runtime take longer, and the data must be sufficient for the training model.

The methodology and implementation presented here show promising results for using RL as an automated tool-chain to learn control rules. The intelligent agent can interact with any simulation model of the environment, utilize any user-defined reward function, and evaluate a wide array of options and variations of RL. However, the use of RL for a more complex system faces many challenges, as laid out in the discussion. To that end, this study's concepts, initial results, and formulations should help build a foundation to support RL as a viable option for reservoir operation management and control. The source code accompanying this paper should also allow others to evaluate many other possible architectures and parameterizations that could be used to improve the results presented in the article.

Bibliography

[1] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[3] Tirusew Asefa, John Clayton, Alison Adams, and Damann Anderson. Performance evaluation of a water resources system under varying climatic conditions: Reliability, resilience, vulnerability and beyond. *Journal of Hydrology*, 508:53–65, 2014.

[4] R Bellman. Dynamic programming princeton university press princeton. *New Jersey Google Scholar*, 1957.

[5] Richard E Bellman and Stuart E Dreyfus. *Applied dynamic programming*. Princeton university press, 2015.

[6] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. volume 1, pages 560–564. IEEE, 1995.

[7] F J C Bouchart and H Chkam. A reinforcement learning model for the operation of conjunctive use schemes. *WIT Transactions on Ecology and the Environment*, 26, 1998.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[9] Theresa M Carpenter and Konstantine P Georgakakos. Assessment of folsom lake response to historical and potential future climate scenarios: 1. forecasting. *Journal of Hydrology*, 249(1-4):148–175, 2001.

[10] A Castelletti, G Corani, A E Rizzoli, R Soncini-Sessa, and E Weber. A reinforcement learning approach for the operational management of a water system. pages 22–23. Elsevier Yokohama, J, 2001.

[11] A. Castelletti, S. Galelli, M. Restelli, and R. Soncini-Sessa. Tree-based reinforcement learning for optimal water reservoir operation. *Water Resources Research*, 46(9):1–19, 2010.

[12] Andrea Castelletti, Giorgio Corani, A Rizzolli, R Soncinie-Sessa, and Enrico Weber. Reinforcement learning in the operational management of a water system. pages 325–330, 2002.

[13] Andrea Castelletti, Francesca Pianosi, and Rodolfo Soncini-Sessa. Water reservoir control under economic, social and environmental constraints. *Automatica*, 44(6):1595–1607, 2008.

[14] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. pages 617–629. PMLR, 2018.

[15] Blagoj Delipetrev, Andreja Jonoski, and Dimitri P. Solomatine. A novel nested stochastic dynamic programming (nsdp) and nested reinforcement learning (nrl) algorithm for multipurpose reservoir optimization. *Journal of Hydroinformatics*, 19(1):47–61, 2017.

[16] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *arXiv preprint arXiv:1605.07127*, 2016.

[17] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. pages 1861–1870. PMLR, 2018.

[19] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, and Pieter Abbeel. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[20] Tsuyoshi Hashimoto, Jery R Stedinger, and Daniel P Loucks. Reliability, resiliency, and vulnerability criteria for water resource system performance evaluation. *Water resources research*, 18(1):14–20, 1982.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[22] Mohamad I Hejazi, Ximing Cai, and Benjamin L Ruddell. The role of hydrologic information in reservoir operation–learning from historical releases. *Advances in water resources*, 31(12):1636–1650, 2008.

[23] Jonathan D Herman and Matteo Giuliani. Policy tree optimization for threshold-based water resources management over multiple timescales. *Environmental modelling and software*, 99:39–51, 2018.

[24] David H Jacobson and David Q Mayne. Differential dynamic programming. 1970.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Thomas Rodding Kjeldsen and Dan Rosbjerg. Choice of reliability, resilience and vulnerability estimators for risk assessments of water resources systems/choix d'estimateurs de fiabilité, de résilience et de vulnérabilité pour les analyses de risque de systèmes de ressources en eau. *Hydrological sciences journal*, 49(5), 2004.

[27] Belize A Lane, Samuel Sandoval-Solis, and Erik C Porse. Environmental flows in a human-dominated system: Integrated water management strategies for the rio grande/bravo basin. *River Research and Applications*, 31(9):1053–1065, 2015.

[28] Robert Edward Larson. State increment dynamic programming. 1968.

[29] Jin Hee Lee and John W. Labadie. Stochastic optimization of multireservoir systems via reinforcement learning. *Water Resources Research*, 43(11):1–16, 2007.

[30] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[31] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[32] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[33] Daniel P Loucks. Quantifying trends in system sustainability. *Hydrological Sciences Journal*, 42(4):513–530, 1997.

[34] William S Lovejoy. A survey of algorithmic methods for partially observed markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

[35] David G Luenberger. Cyclic dynamic programming: a procedure for problems with fixed delay. *Operations Research*, 19(4):1101–1110, 1971.

[36] Masoud Mahootchi, Hamid R. Tizhoosh, and K. Ponnambalam. Reservoir operation optimization by reinforcement learning. *Journal of Water Management Modeling*, 6062(January), 2007.

[37] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[39] Wai-See Moy, Jared L Cohon, and Charles S ReVelle. A programming model for analysis of the reliability, resilience, and vulnerability of a water supply reservoir. *Water resources research*, 22(4):489–498, 1986.

[40] Matthew E Peacock. *A Value-Function Based Method for Incorporating Ensemble Forecasts in Real-Time Optimal Reservoir Operations.* PhD thesis, Colorado State University, 2020.

[41] Jeffrey D. Rieker and John W. Labadie. An intelligent agent for optimal river-reservoir system management. *Water Resources Research*, 48(9):1–16, 2012.

[42] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. Citeseer, 1994.

[43] S Sandoval-Solis, D C McKinney, and Daniel P Loucks. Sustainability index for water resources planning and management. *Journal of water resources planning and management*, 137(5):381–390, 2011.

[44] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[45] Jhih-Shyang Shih and Charles ReVelle. Water-supply operations during drought: Continuous hedging rule. *Journal of Water Resources Planning and Management*, 120(5):613–629, 1994.

[46] Jhih-Shyang Shih and Charles ReVelle. Water supply operations during drought: A discrete hedging rule. *European journal of operational research*, 82(1):163–175, 1995.

[47] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, and Thore Graepel. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[48] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

[49] R Suttan and A Barto. Reinforcement learning: An introduction, mit press. 1998.

[50] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[51] J Alberto Tejada-Guibert, Sharon A Johnson, and Jery R Stedinger. The value of hydrologic information in stochastic dynamic programming models of a multireservoir system. *Water resources research*, 31(10):2571–2579, 1995.

[52] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.

[53] S W D Turner and S Galelli. Regime-shifting streamflow processes: Implications for water supply reservoir operations. *Water Resources Research*, 52(5):3984–4002, 2016.

[54] Edson de O Vieira and Samuel Sandoval-Solis. Water resources sustainability index for a water-stressed basin in brazil. *Journal of Hydrology: Regional Studies*, 19:97–109, 2018.

[55] CY Wan and CM Nolan. Folsom dam auxiliary spillway–design innovations and construction lessons learned. 2016.

[56] Christopher J C H Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[57] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[58] G Wilson. Reinforcement learning: A new technique for the real-time optimal control of hydraulic networks. 1996.

[59] Peter J Wong and David G Luenberger. Reducing the memory requirements of dynamic programming. *Operations Research*, 16(6):1115–1125, 1968.