Online Predictions, RL and Water Treatment: A GVF Story

by

Muhammad Kamran Janjua

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

We study the use of reinforcement-learning based prediction approaches for a real drinking-water treatment plant. Developing such a prediction system is a critical step on the path to optimizing and automating water treatment. Before that, there are many questions to answer about predictability of the data, suitable neural network architectures, how to overcome partially observability, and more. We describe this dataset, and highlight challenges with seasonality, nonstationarity, partial observability and heterogeneity across sensors and operation modes of the plant. We then describe General Value Function (GVF) predictions—discounted cumulative sums of observations–and highlight why they might be preferable to classical n-step predictions common in time series prediction. We discuss how to use offline data to appropriately pre-train our temporal difference learning (TD) agents that learn these GVF predictions, including how to select hyperparameters for online fine-tuning in deployment. We find that the TD prediction agent obtains an overall lower normalized mean-squared error than the n-step prediction agent. Finally, we show the importance of learning in deployment, by contrasting to a TD agent trained purely offline with no online updating. This final result is one of the first to motivate the importance of adapting predictions in real-time, for non-stationary high-volume systems in the real-world. Before we can hope to control a complex industrial facility, we must first ensure that learning of any kind is feasible. This work represents such a feasibility study.

Preface

The work in this dissertation is under-submission to the Machine Learning journal. Almost all of the content of this thesis is taken from the journal submission, and has been edited to fit the dissertation requirements. Some chapters, such as Chapter 2, and 3, have been added to provide more context to the work under discussion.

To Mom, Dad, and Begham

Who support me through my journey, tolerate my eccentric decisions and my adventures, and yet they continue to patiently wait for me to return home.

بچۂ شاہیں سے کہتا تھا عقابِ سالخورد
اے ترے شہپر پہ آساں رفعتِ چرخِ بریں

An eagle full of years to a young hawk said—
Easy your royal wings through high heaven spread

ہے شباب اپنے لہو کی آگ میں جلنے کا نام
سخت کوشی سے ہے تلخِ زندگانی انگبیں

To burn in the fire of our own veins is youth!
Strive, and in strife make honey of life's gall;

– Sir Muhammad Iqbal
Bal-e-Jibril, 140

This can be considered a non-scholarly attempt at translation of the Urdu poem. Therefore, this might not convey the true meaning of what the poet has to say. Reference: Link to Complete Translation

Acknowledgements

The work presented in this dissertation is the direct result of collective efforts, and brainstorming sessions of the entire Water Treatment team. First and foremost, I would like to thank my advisor Dr. Martha White for her continued support throughout my tenure as a graduate student at RLAI. She has been an immense support not only as my research advisor, but also as a career counsellor as she has helped me chart out my academic career. I am very grateful for how she has patiently listened to me, even when I did not make any sense, and gave me sound advice and direction. I was fortunate enough to be her teaching assistant, and her advice and guidance on that front has also been invaluable. Most importantly, Martha has made me understand the importance discussing scientific ideas more freely.

I thoroughly enjoyed discussing research with Dr. Adam White during our weekly meetings. He has immensely helped me in understanding how to clearly present scientific ideas as he carefully reviewed my work (both during the paper writing phase and during our conversations). He has instilled in me the importance of sound empirical work, and what experimental rigour means to science. His enthusiasm and vision for this work is nothing short of inspiring. I am very thankful to Dr. Adam White for his continued support.

Third, I would like to acknowledge the support of Dr. Marlos Machado throughout this work. Among many other things, Marlos has also worked with me to make plots (the pretty ones throughout this dissertation, and the paper). I have yet to meet a much warmer crowd of brilliant people as these three.

I would also like to thank Randy Goebel for his thoughtful review of this dissertation and his insightful suggestions.

Furthermore, this work would not have been made possible without the wonderful collaborators working on this project: Puer Liu, James Bell, James Short, Jordan Coblin, Lingwei Zhu, Erfan Miahi, Han Wang, Abhishek Naik, and Haseeb Shah.

Lastly, I would also like to gratefully acknowledge the support of Khurram Javed, and Zaheer Abbas throughout my research endeavors. They have helped me, and provided me with meaningful guidance for as long as I can remember (even before I came to the University of Alberta).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis, we study the use of reinforcement learning based prediction approaches for a real drinking-water treatment plant. In this chapter, there is first an overview of adaptive systems, motivating the importance of deploying reinforcement systems that learn in deployment. Then, the problem setting and contributions in this thesis are outlined.

1.1 An Overview of Adaptive Systems

In his seminal work on human understanding, John Locke argued that human mind resembles a blank slate (*tabula rasa*) at birth, and that all knowledge is formed through one's own experiences acquired through environmental interactions (Locke 1847). The idea of mind being a blank slate at birth was first introduced in Ibn Tufail's allegoric novel *Hayy Ibn Yaqzan* (The Improvement of Human Reason: Exhibited in the Life of Hai Ebn Yokdhan) (Ockley et al. 1708). The novel follows the life of a young boy raised in the wild as he learns and acquires knowledge solely through environmental interactions and observations. Contrary to this, Leibniz did work on a rebuttal of Locke's work (G. W. Leibniz and G. W. F. v. Leibniz 1996) and criticized his ideas while arguing in favor of prevalence of innate behaviors at birth. If human mind at birth is tabula rasa or not is up for debate in philosophical circles, however it is unanimously agreed upon that learning does not stop as humans continue to function (and live). This is inline with how living organisms continue to learn, adapt, and modify their behaviors since the world around them

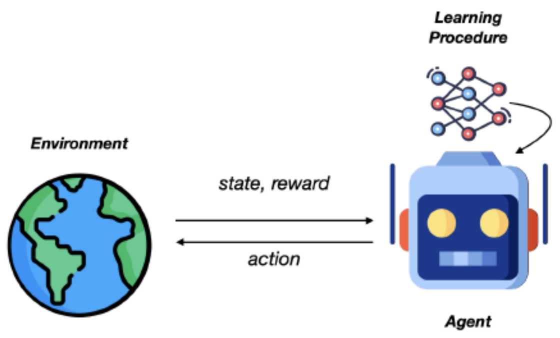

Figure 1.1: The online learning setting: An agent, equipped with some learning procedure, interacts with the environment by acting on it and gets some information and feedback signal that it can then use to improve.

favors such a tendency.

Animal, human and non-human, behavior is characterized by a certain response to some stimuli. This response is, generally, tailored by some prediction that the respondent makes pertaining to the stimuli. Voyaging through time during the span of their lives, all animals develop and perfect the tendency to make predictions (Gilbert and Wilson 2007). This tendency is deeply rooted in the survival instinct of humans and animals alike since predictions furnish the whole gamut of their existence. To this end, consider how in self-defense the prey predicts the perpetrator (predictions) and acts (control) to avoid fatal consequences before the latter attacks. For example, on comprehension of some external stimuli (hearing a sound in a nearby bush), the prey can predict the possible existence of some predator. Such predictions could be used for the most naive control approaches such as Pavlovian control (Pavlov 2010), or prey triggering its self-defense mechanism in the example. However, as predators evolve to develop techniques that aid their foraging behavior, so does the prey in order to evade the perpetrator. While many factors are at play in predator-prey dynamics (Abrams 2000), learning to adapt is crucial to both in order to survive.

In its simplest form, learning to adapt (termed as continual/online learning) can be defined as *the process of constant improvement, towards no single, final end other than improvement itself* (Ring et al. 1994). Given how organisms continually learn, it is but natural to think of learning in artificial agents as continual by design. Before we discuss design of such agents, it is important to understand what forms continual learning and adaptation. In his book on adaptive systems (Holland 1992), Holland describes three central components in the problems of adaptation: an environment that undergoes change (E), learning procedure undergoing modifications in response to the environment (τ), and some measure of the performance (μ). This environment, E, is ever-changing and supplies the actor (an agent, either artificial or natural) with complexities and regularities. The agent, equipped with some learning procedure, adjusts and adapts as it interacts with this environment; using the performance measure (feedback) as a signal of improvement.

1.1.1 Types of Learning Procedures

Generally in the reinforcement learning community, learning procedures can be segregated into two types: offline and online (or deployment, used interchangeably throughout this dissertation). In an offline learning scenario, also referred to as batch learning, the dataset is collected and made available to the agent; the agent does not interact with the environment to gather samples therefore there is no adaptation or continuous improvement. This is different from online learning wherein the agent gets to interact with the environment, collect sample and learn by acting on the environment, as illustrated in the Figure 1.1. We emphasize the importance of learning online (or learning in deployment, used interchangeably henceforth) and argue that allowing the agent to learn in deployment can be of much benefit and aligned with real-world applicability goals.

1.2 Learning Predictions Online for Water Treatment

Learning in deployment is critical for partially observable decision making tasks (Richard S. Sutton, Koop, and Silver 2007). If the evolution of state transitions is driven by both the agent's actions and state variables that the agent cannot observe, then the process will appear non-stationary to the agent.

For example, an agent controlling chemical dosing in a water treatment plant (WTP) may correctly learn the relationship between increasing chemicals to reduce turbidity in the water. However, inclement weather events can also impact water turbidity causing the agent's prediction of future turbidity—and thus choices of chemical dosing—to be suboptimal. One approach to mitigating this problem is to allow the agent to continually update its predictions and decision making policies online in deployment.

Effective multi-step prediction forms the basis for effective decision making in almost any reinforcement learning system. Classical value-based methods, such as Q-learning (Watkins and Dayan 1992), construct a prediction of future discounted reward in order to decide on what actions to take. Policy gradient methods such as PPO (Schulman et al. 2017) and SAC (Haarnoja et al. 2018) typically define the agent's policy through an estimate of the value function. In applications, often the first step is to build a prediction learning system that can predict future reward and sensor values far into the future. This is an important step to assess feasibility of adaptive control, but is also a useful first step because the tasks of feature engineering, network architecture design, optimization, and tuning of various hyperparameters will be shared and beneficial to both a prediction learning system and a full reinforcement learning system.

There has been growing interest in moving RL techniques out of video games and into the real-world. In many applications, such as chip design (Mirhoseini et al. 2021), matrix multiplication (Fawzi et al. 2022), and even video compression (Mandhane et al. 2022), the problem setting of interest is simulation. Another approach is to design and train an agent in simulation and then deploy a fixed controller, sometimes even in the real-world. This approach has been used for example in navigating stratospheric balloons (Bellemare et al. 2020), controlling plasma configurations inside a fusion reactor (Degrave et al. 2022), and robotic curling (Won, Müller, and Lee 2020).

In this dissertation we study and discuss the application of machine learning techniques, specifically prediction methods from reinforcement learning, on a real drinking water treatment plant. In our setting, we do not have ac-

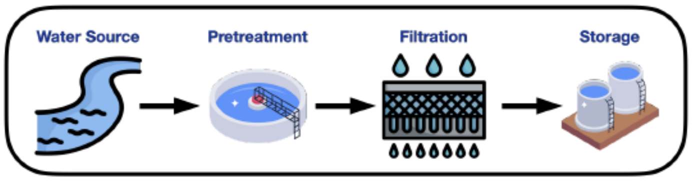

Figure 1.2: An illustration of the drinking water treatment plant. The entire plant is divided into two main stages: pretreatment and filtration. The pretreatment stage is concerned with adding chemicals to the raw water, followed by the filtration stage where the water is pumped through filters for further purification.

cess to a high-fidelity simulator of the plant, nor the resources to commission one. This work explores the feasibility of adaptive learning systems in the real-world, without access to a computer simulation for iterating design choices or pre-training the agent.

Closer to our work, recent work on automating HVAC control used an approach where the agent is first tuned on off-line data and then a learning controller is deployed that is updated once a day (Luo et al. 2022). In this work the authors explicitly avoided offline training on operator data, citing the well-known issues of insufficient action coverage. Nevertheless, *batch* or *offline learning* methods (Ernst, Geurts, and Wehenkel 2005; Riedmiller 2005; Lange, Gabel, and Riedmiller 2012; Levine, Kumar, et al. 2020) have been successfully used in settings where a fixed policy or value function is extracted from a dataset, with several practical applications (Pietquin et al. 2011; Shortreed et al. 2011; Swaminathan et al. 2017; Levine, Pastor, et al. 2018).

Drinking water treatment is basically a two stage process, as depicted in Figure 1.2. First, water is pumped into a large mixing tank where chemicals are added to cause dissolved solids to clump together. The next step is to pull the pretreated water through a filter membrane where only clean water molecules can pass through the filter membrane and the solids and other continents remain. Periodically, the primary filter is cleaned by simply running the process backwards blasting the filter membrane clean: a process called backwashing. In Canada, the operation of a water-treatment plant can represent up to 30% of a town's municipal budget (Copeland and Carter 2017).

Drinking water treatment is uniquely challenging compared to other applications due to two key characteristics. The data produced by a water-treatment plant, like many real-world systems, is high-dimensional, noisy, partially observable, and often incomplete, making online, continual predictions extremely challenging. In water treatment, the plant can operate in different modes, such as production and backwashing. The mode has a profound impact on data produced by the system and even changes the range of valid sensor readings. Second, the different components of the plant operate at different timescales and decisions have delayed consequences. For example, the chemical dosing rate is typically not changed more often than once a day, backwashing happens multiple times a day, and pretreatment tank mixing-rate can be adjusted continuously. Each one of these choices can result in changes in sensor readings over minutes—chemical dosing changes the water pressure on the filter within 30 minutes—to months—too much chemical dosing can degrade filter efficiency over the long run.

We seek to understand one simple question: *Given a nonstationary, and partially observable real-world environment, can we learn an adaptive system that makes predictions many steps in future and continues to improve online?*. To answer the question, we investigate multi-variate, multi-step prediction in deployment on a real system. We provide a detailed case study on water treatment (a real-world plant), first demonstrating the inherent nonstationarity of the problem and the benefits of learning continuously in deployment. We show that, using a simple trace-based memory to overcome partially observability, we can learn accurate multi-step predictions, called general value functions (GVFs) (Richard S Sutton, Modayil, et al. 2011; Modayil, A. White, and Richard S Sutton 2014), using temporal difference (TD) learning. Because GVFs can be learned with standard reinforcement learning algorithms like TD, they can easily be update online, on every step. We show that updating online can significantly improve performance over only training from an offline log of data. The online prediction agent also benefits from this offline data, to pre-train the predictions and to set the hyperparameters for updating online in deployment. Our approach allows us to have a fully specified online prediction agent—with hyperparameters automatically selected using a simple modification on the standard validation procedure—that continues to adapt and improve in deployment.

Finally, we also contrast these GVF multi-step predictions to the more classical predictions considered in time series prediction: n-step predictions. The primary goal of this comparison is to provide intuition: n-step predictions are a more common and widely understood multi-step prediction, as compared to GVFs. Our goal is to introduce GVF predictions to a wider audience, and hopefully motivate this additional modeling tool. Beyond this, we highlight that GVFs can have benefits over n-step predictions. The target for a GVF is typically smoother, because it is an exponential weighting of future observations, rather than an observation at exactly n steps in the future. Consequently, we also expect this target to be lower variance and potentially simpler to learn. We do find that GVF predictions have high accuracy than the n-step predictions on our data, controlling for the same state encoding and network size, in terms of the normalized mean-squared error. Taken together, our work provides several practical insights on designing neural-network learning systems capable of learning in deployment, supported by real data generated by a real water-treatment plant.

The rest of this dissertation is organized as follows. In Chapter 2, we discuss the preliminaries going over some of the background, and looking at some existing literature on the topic. In Chapter 3, we discuss the construction of the water treatment plant, and go through the data generated by the plant and understand what makes it interesting. Following this, in Chapter 4 and Chapter 5, we discuss theoretical construction of both solution methods (GVFs, and n-step) and discuss algorithms that we design for the purpose of this problem. Finally, in Chapter 6, we look at how to design the experiment to answer the questions and discuss the results at length. We conclude the dissertation with a discussion of future perspectives on the topic in Chapter 7.

Chapter 2

Preliminaries

In this chapter, we briefly present overview of the context of this dissertation. This will help the reader understand what different terminologies mean and what learning methods and frameworks surround the work presented in this book. Mainly, we follow the reinforcement learning (RL) problem setting to develop our continual learning algorithms, therefore we describe the RL framework. Furthermore, we also look at some of the related work pertaining to our proposed problem and summarize previously published works in the literature on the topic.

2.1 Brief History

The idea of designing automated systems capable of learning from streams of data by working out underlying patterns goes back several decades. Efforts in the direction of building such learnable systems fall under the umbrella of machine learning, which became an active research area when the perceptron was introduced (Rosenblatt 1958). This perceptron algorithm, inspired by neural networks in human brain, was composed of a single layer and functioned to classify visual data. However, with its failure to capture non-linearity in data –specifically the XOR problem (Marvin and Seymour 1969)– a sudden decline in machine learning community was observed; this decline is referred to as *AI Winter*. With the invention of backpropagation algorithm (Rumelhart, G. E. Hinton, and Williams 1986), and proposition of utilizing multiple layers, it became feasible to extend perceptron to multiple layers and training large deep neural networks became a possibility. Earlier machine learning approaches relied on carefully engineered and handcrafted feature extractors capable of extracting necessary information from the input data. However, such approaches were limited by manual labor requirement and did not generalize to unseen data samples well. With increasing availability of compute power and large datasets (Deng et al. 2009), it became a standard in the community to stack multiple non-linear layers together to allow the neural network to learn representations from raw data (Krizhevsky, Sutskever, and G. E. Hinton 2017). This ushered a new era in deep learning, a term that was coined to signify the usage of deep neural networks, since it paved the way for construction of strong function approximators, neural networks, that could learn from vast amounts of data without manual feature extraction labor (LeCun, Bengio, and G. Hinton 2015).

2.2 Reinforcement Learning

Reinforcement learning (RL) is a problem setting concerned with an agent that learns from some scalar reward signal. We, formally, describe RL problem as a Markov Decision Process (MDP) defined by 4-tuple $(\mathcal{S}, \mathcal{A}, P, r)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions available to the agent, P is the transition probability function and r is the reward function. At any timestep, t, the agent receives some observation from the environment, S_t, takes some action, A_t, and receives some scalar reward, R_{t+1}, and next state observation, S_{t+1}. The agent learns a policy π to maximize the sum of future rewards, also referred to as the return.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{i=0}^{\infty} \gamma^i R_{t+1+i} \tag{2.1}$$

The term $\gamma \in [0, 1)$ refers to the timescale and determines what part of the reward should the agent focus on; if $\gamma = 0$ then agent is only concerned with the immediate reward. In RL, value functions are used to estimate the expected value of return in a given state.

$$v(s) = \mathbb{E}\left[G_t | S_t = s\right] \tag{2.2}$$

Generally, the value function is governed by some policy and discount, but for brevity we can drop it. We are concerned with online learning, continually learning as new experience is made available, therefore we want a solution method that can learn the value estimates online. Temporal-difference (TD) learning algorithms allow online learning due to bootstrapping, a technique that allows estimating the value on the basis of other value estimates (Richard S Sutton and Barto 2018). In the most straight-forward case, one-step TD update is given by computation of the TD error δ. We can write the one-step TD update for value estimates as follows.

$$V_{t+1}(S_t) \leftarrow V_t(S_t) + \alpha\delta_t \tag{2.3}$$

The TD error can be computed as $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$, and α refers to the stepsize. In principle, the value estimates are updated towards the samples of return. Notice how return refers to sum of future scalar rewards.

2.2.1 General Value Functions (GVFs)

In RL, we distinguish between prediction and control problems. A prediction task refers to predicting the expected total reward from any given state, given the policy π (known as policy evaluation). In other words, the value functions estimates are updated to be more accurate. In most cases, the prediction target (one that is predicted) is the reward. However, general value functions (GVFs) allow relaxation to the value function formulation by generalizing the prediction targets to be any feature of the environment available to the agent (Richard S Sutton, Modayil, et al. 2011). More generally, the prediction target is referred to as cumulant, represented by C.

$$G_t = C_{t+1} + \gamma C_{t+2} + \gamma^2 C_{t+3} + ... = \sum_{i=0}^{\infty} \gamma^i C_{t+1+i} \tag{2.4}$$

The term γ refers to discount, or timescale. GVFs are defined with respect to a timescale and since they form predictive questions, γ determines the timescale of the prediction. For example, consider an agent adding chemical dose to the raw water on a water treatment plant:

- *If I add 10mg/l of chemical dose to raw water, what will the water turbidity be in 100 timesteps?*
  Policy: 10mg/l Dose Raw Water
  Cumulant: Water Turbidity
  Timescale: 100 timesteps ($\gamma = 0.99$)

We can also determine the timesteps from the timescale, which allows for more intuitive understanding. This can also be thought of as the horizon for a GVF prediction (Richard S Sutton, Modayil, et al. 2011).

$$\tau = \frac{1}{1-\gamma} \tag{2.5}$$

The term τ has a unit of timesteps (the definition of what each timesteps refers to can vary depending on the domain). Note that GVFs are a generalization of value functions, and therefore we can employ standard TD learning algorithms as usual. We discuss how we leverage the bootstrapping updates of TD to construct our online learning algorithms for GVF predictions at length in Chapter 4.

2.3 Related Work

In real-world RL, several works focus on learning in high-fidelity simulators before deployment in real-world (which refers to halting of the learning procedure); some examples of such works include network defense (Wolk et al. 2022), congestion control (Fuhrer et al. 2022), navigation of stratospheric balloons (Bellemare et al. 2020), plasma control (Degrave et al. 2022), etc. The availability of such dense simulators is a major setback and requires years worth of effort to produce meaningful simulators that can translate well to the real-world. It is no surprise that capturing the real-world dynamics, even in such high-fidelity simulators, is an exceedingly difficult task, hence on real-world deployment such efforts face degradation in performance (Wolk et al. 2022) or are solely deployed in the simulator (Agarwal et al. 2022). As an alternative, learning from offline data is another possibility for real-world RL. However, it is well-known that the learned policy is highly dependent on the quality of offline data and has no means to improve on encountering out-of-distribution states during deployment, unable to leverage new data as it is made available. To further explain the limitations to such approaches, we look at two different works that tackle the problem of RL in the wild.

The work done in (Tessler et al. 2022) looks at the problem of congestion control in a datacenter. The authors develop a deterministic on-policy algorithm, ADPG, to learn a policy for congestion control in a simulated network. A follow up work (Fuhrer et al. 2022) simplifies the learned complex policy by transforming the underlying neural network to a decision tree, and achieve better latency on deploying it on a real-world network. Although these works deal with partial observability, they, however, solely focus on learning in a simulated network (before deployment), and have a simplified state-space consisting of present and past transmission rate and Round-Trip-Time (RTT) measurement. In addition to the lack of continuous updates to the learning agent in real-time, since on deployment the learning procedure is halted due to distillation of the neural network to a boosting tree, there is also little to no problem caused by existence of multiple time-scales and high frequency of data availability during policy iteration (pre-deployment).

In a recent work (Luo et al. 2022), the authors look at the problem of controlling commercial cooling systems using RL. The work focuses on HVAC control, leveraging the fact that HVAC is a decision-making problem and design an RL algorithm to learn an energy-efficient operating strategy by providing set-point recommendations to the chiller plant (which is an integral component of the HVAC). The algorithm digests real-world data of different sensor readings, and equipment status (on/off), however the state-space only comprises of 50 different real-valued measurements which are recorded at a five minute interval. The chiller plant has two different modes of operation, but the work does not explicitly cater to these different modes and resorts to mode-specific action masking which refers to agent returning superset of all actions, and the selection procedure masks the actions which are irrelevant to the mode plant is operating in. A crucial difference between the chiller plant and our work manifests itself in continuously digesting and learning the available data. With respect to the chiller plant, the authors only focus on re-training at the end of each day. This is in contrast to the tenet of online learning; we consider true online learning to learn at each time-step as the data is made available.

More recently, industrial water treatment problem has been posed as a collection of reinforcement learning tasks (Liu 2022). The authors detail the construction of a water treatment plant, along with suitable sub-tasks that can be framed as RL problems. The work also presents a case study on chemical dose control on the plant, and details how a suitable reward function can be designed.

2.3.1 Water Treatment Plant Automation

In several limited contexts, the literature has explored automating water treatment plants either through controlling certain components of the plant, or by learning to predict a few sensors based on some water characteristics. Backwashing is an important mode of operation in a water treatment plant, where the filters are cleaned by running the water reverse through them. Generally, plants follow a fixed backwashing schedule based on certain heuristics as determined by the plant operators. This might not be a very cost effective scenario given how backwashing is dependent on filter health which in turn is dependent on influent water characteristics. Consider how in winter the water is generally clean, and therefore would require minimum backwashing. More frequent backwashing can reduce the water produced (or productivity of the plant), whereas if done less frequently the filters can be significantly damaged. This specific problem of learning a backwash scheduler is considered (B. Zhang et al. 2020), where dynamic programming is used to learn a backwashing policy from offline logs of data which is deployed fixed on the treatment plant.

In addition to backwash schedulers, pre-treatment is a major component of the water treatment plant, in addition to filtering. Pre-treatment is responsible for treating the water before it is subjected to filtering. The work (Q. J. Zhang et al. 2007) proposes to predict turbidity in the effluent water utilizing the chemical dose and various water characteristics, as part of the pre-treatment stage. They employ a naive control mechanism that selects the chemical dose range that results in lowest turbidity. Note that these techniques are limited since they do not allow on-the-fly adaptation as the underlying data changes over time.

Chapter 3

Water Treatment Plant

In this chapter, we provide an overview of the water treatment plant (WTP) and the data that the plant generates. We discuss how the plant is constructed, and how different components combine to perform plant operations. We then introduce different properties of the data that the plant generates, and visualize different patterns that arise in the data due to various factors. This chapter, at length, introduces the water treatment plant as an RL problem and motivates why learning in deployment is the only choice in such a real-world data rich application.

3.1 Overview of the Plant

Water treatment plants are large-scale industrial plants that perform the function of cleaning raw water, taken from the an external source, to make it fit for human consumption. These plants differ in construction based on their functions, and locality; sewage water treatment plants are different from river water treatment plants. Similarly, in localities where rivers run in abundance the treatment plant would differ from the one that is responsible for cleaning ground water. The plant in this dissertation refers to a river water treatment plant which cleans the raw water to make it fit for drinking purposes.

Most of the industrial scale WTPs divide the process of cleaning raw water into several stages chained together sequentially, since water from one stage flows into the other as it gets treated. At the very least, each plant consists of a pretreatment and a filtration stage since they combine to form the most essential components of a WTP.

3.1.1 Pretreatment

Pretreatment is often the first stage of treatment. It is combines several sub-stages, all of which work to remove solids and sludge from the raw water. Initially in pretreatment, chemicals are added to coagulate the water particles and dirt as they combine with raw water, a process known as coagulation. The next step involves mixing the coagulated water to form larger clumps of solids. This mixing is done in a large flocculation tank and is termed as the flocculation process. In sedimentation phase, these larger clumps settle to the bottom due to being heavier than the water and are eventually removed from the water. This leaves the raw water without dirt and other suspended solids, see Figure 3.1 for a simplified illlustration of the pretreatment stage.

3.1.2 Filtration

In pretreatment, the added chemicals are positively charged and they neutralize the negative char on dirt and other particles. However, smaller particles or bacteria could still remain in the water. Therefore, the filtration stage follows the pretreatment stage. Several filters are combined through which the pretreated water is pushed to clean the remaining particles. These filters can get dirty over time since the particles stick to these filters as the water is made to pass through them, and they require frequent cleaning. A process known as backwash (BW) is employed to perform the task of cleaning filters. During backwash, some of the water is pushed in reverse through the filters to remove the particles that are stuck to the filters. Once the water is filtered, it is sent to a storage reservoir which can then either provide the water for consumption.

3.1.3 Operational Cycle of Plant

During its cycle, the plant goes through several different modes of operation. The plant mode is also a type of setpoint, which can be set at specified times as the plant enters and exits these modes, forming an operational cycle; we

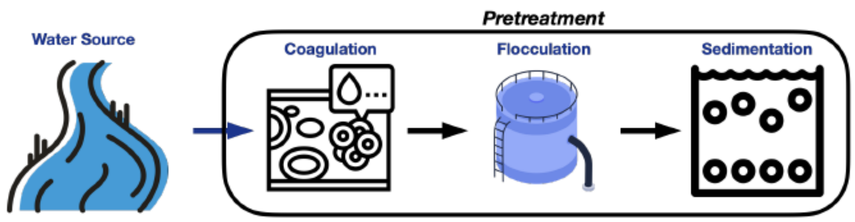

Figure 3.1: An illustration of the first stage of drinking water treatment plant: pretreatment. The river serves as the water source, and the raw water passes through the coagulation, flocculation and sedimentation processes. All of these combine to form the pretreatment stage.

list the important modes here.

1. Production (PROD) - The plant produces clean water in this mode.
2. Standby (STBY) - This is a standby mode the plant cycles through during operation.
3. Backwash (BW) - The backwash mode refers to pumping water backwards to clean the filters.
4. Drain (DRAIN) - As the name suggests, this mode is responsible for drainage of the waste water.
5. Membrane Integration Test (MIT) - The plant in this mode tracks the changes in the membrane damage, as a part of preventive maintenance strategy.

This operational cycle of the plant is a loop through these modes: STBY $\rightarrow$ PROD $\rightarrow$ BW $\rightarrow$ DRAIN $\rightarrow$ PROD $\rightarrow$ STBY, we refer to this loop as the normal cycle of the plant. Additionally as part of the preventive maintenance, MIT mode is scheduled at a specified time every day interrupting the normal cycle, which then continues after the MIT mode has ended. The production mode is the longest mode, taking up-to 20 minutes in a single pass of the normal cycle, and therefore is a dominant mode.

3.2 Water Treatment Data

Like any industrial control process, a water treatment plant has the potential to generate an immense amount of data. Our system is instrumented with a large number of sensors reporting both (1) water chemistry throughout the treatment pipeline, and (2) properties of the mechanical components of the plant. Taken together these sensor readings form a long and wide time series with several interesting properties. In this section we highlight these properties with examples from a real plant, explaining how each makes long-term prediction challenging.

3.2.1 Wide, Long, and Fast Data

Our system reports 480 distinct sensor values at a rate of one reading per second producing a large time series. One year of data consists of over 31 million observations of the plant and over 15 million individual sensor readings. In contrast, the recent M5 time-series forecasting competition used a dataset with 42,840 dimensional observations and 1969 time-steps; over 84 million samples (Makridakis, Spiliotis, and Assimakopoulos 2022). Using multiple years of water treatment data puts us in the same scale as state-of-the-art forecasting grande challenge problems. We summarize some of the sensors in Table 3.1, and provide more detail in 3.2.

Sensor Type	**Measures**
Pressure	Pressure on the membrane.
Flowmeter	Flow rate of the fluid.
pH	Acidity and alkalinity of the solution.
Temperature	Temperature of the water.
Turbidity	Turbidity of the water.
Total Organic Carbon (TOC)	Organic carbon in the water.
Conductivity	Ability to pass an electric current.

Table 3.1: A brief summary of different measurements each of the sensor type is responsible for working out.

Our data exhibits coherent structure over the year, month, day and minute. In Figure 3.2 we plot incoming water temperature at three temporal resolu-

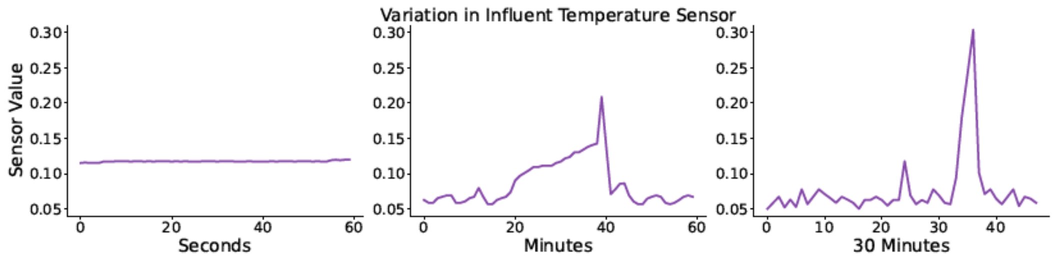

Figure 3.2: The many timescales of water treatment. Each subplot shows the incoming water temperature from the river at different temporal resolutions. Viewing right to left, if we look at temperature over the entire day (subsampled) we see a single outlier and an otherwise fluctuating baseline. In the middle subplot, looking at a singe hour of data we see the spike has more structure. Finally, the left most subplot shows one minute of data sampled at the fastest possible timescale of the system (no subsampling), which shows how in a short timescale measurements can even appear constant.

tions. Mechanical systems like ours often support sampling at rates of 1 Hz or greater, whereas data sets commonly used in time-series forecasting are wide and short; typically sampled once a day. In water treatment, high-temporal resolutions are relevant because the data can be noisy (as highlighted in Figure 3.3) and averaging is lossy. In addition, if one were to change process set-points (the ultimate end-goal of prediction), this may require rapid adjustment (for example, adjusting PID control parameters during a backwashing operation).

3.2.2 Sudden, Unpredictable Events

Our data exhibit substantial distribution shifts, largely due to unpredictable events. For example, Figure 3.3 shows the impact of cleaning different sensors. Most of these sensors get physically dirty over time due to a variety of factors. Sometimes water gets accumulated in the sensor enclosure, or moisture develops on the physical sensors, causing the readings to become noisy and unreliable. The plant operators manually clean the sensors to make sure they are as noise-free as possible and are reliably operating. Often times the sensor patterns indicate that they have recently undergone cleaning. This change in pattern manifests itself as sensor signal stabilizes over time post cleaning.

A water treatment plant operates in different modes which dramatically

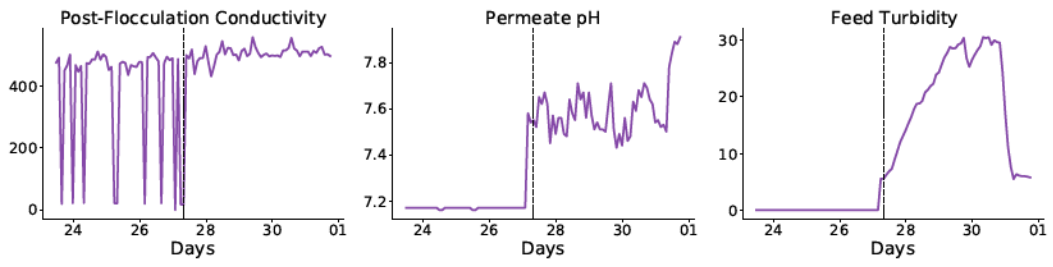

Figure 3.3: Raw values of some of the sensors before and after the cleaning. The black dotted line indicates when the sensors were manually cleaned by the plant operators. Note that the data is sub-sampled to avoid congestion in the plot.

impacts the data generated. The main modes of operation are production and backwash. In production the water is drawn through the filter to remove contaminants and it is moved to storage. In backwashing—the process of cleaning the filters—water moves backward through the system from storage, through the filters and eventually into the waste (reject) drain. In Figure 3.4 we can see the impact of these two modes across several sensors.

In our plant, mode change is driven either by a fixed schedule or human intervention. Maintenance, for example, occurs every day at 4:30am, triggering the Membrane Integration Test (MIT) mode, whereas backwashing occurs on a strict schedule. Sensor changes due to these mode changes should be predictable from the time series itself, however, more ad hoc operator interventions are better represented as unpredictable external events; for example, when the plant is shut down. In addition, unscheduled maintenance occurs periodically—it is conceivable that such maintenance could be predicted based on the state of the plant, but there are other constraints like staffing constraints that can drive mode change.

3.2.3 Sensor Drift and Seasonal Change

Water treatment is predominately driven by the conditions of incoming river water which changes throughout the year. These changes are driven by seasonal weather patterns. In the dead of Winter the river is frozen and cool, clean, low turbidity water flows under the ice into the intake valves. During

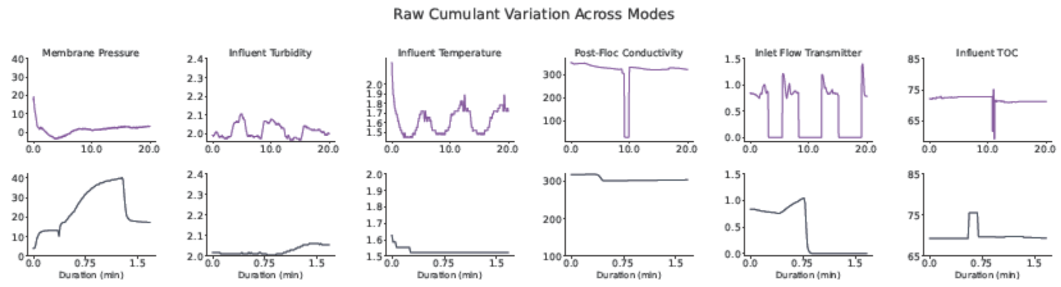

Figure 3.4: Variation across modes of different sensors. For brevity, we only produce two important modes, namely production (PROD), and backwashing (BW). The top row corresponds to the production mode, while the bottom row corresponds to the backwashing mode.

the Spring thaw—called the freshet—snow and ice all along the watershed of the river melt, increasing volume, flow, turbidity, and organic compounds in the river. Early Summer is dominated by a mixture of melted snow and ice higher up in the mountains and heavy rains that cause second and third freshets. Over the Summer, precipitation reduces, causing the late Summer and Fall to exhibit similar patters as the Winter. All of these patterns are clearly visible in Figure 3.5.

Change also happens within a single day. In Figure 3.6 we see how two different sensors evolve over a single day, on different days. As we can see in the plot of Feed Turbidity, some days are similar, but others, such as May 31, 2022, exhibit dramatically different dynamics. In some applications like HVAC control (Luo et al. 2022), it is sufficient to perform learning on a batch of data once a day. In water treatment, the sensor dynamics provide opportunity to observe sensor changes throughout the day.

3.2.4 The State of a Water-treatment Plant?

What information would we need to predict water treatment data many steps into the future, with high accuracy? The plots above paint a clear picture of a partially observable complex dynamical system. Consider the Spring freshet. The volume and flow of the river will be driven by weather patterns and by the snow accumulation all along the watershed throughout the Winter. Digging deeper, the turbidity and other metrics are also driven by erosion

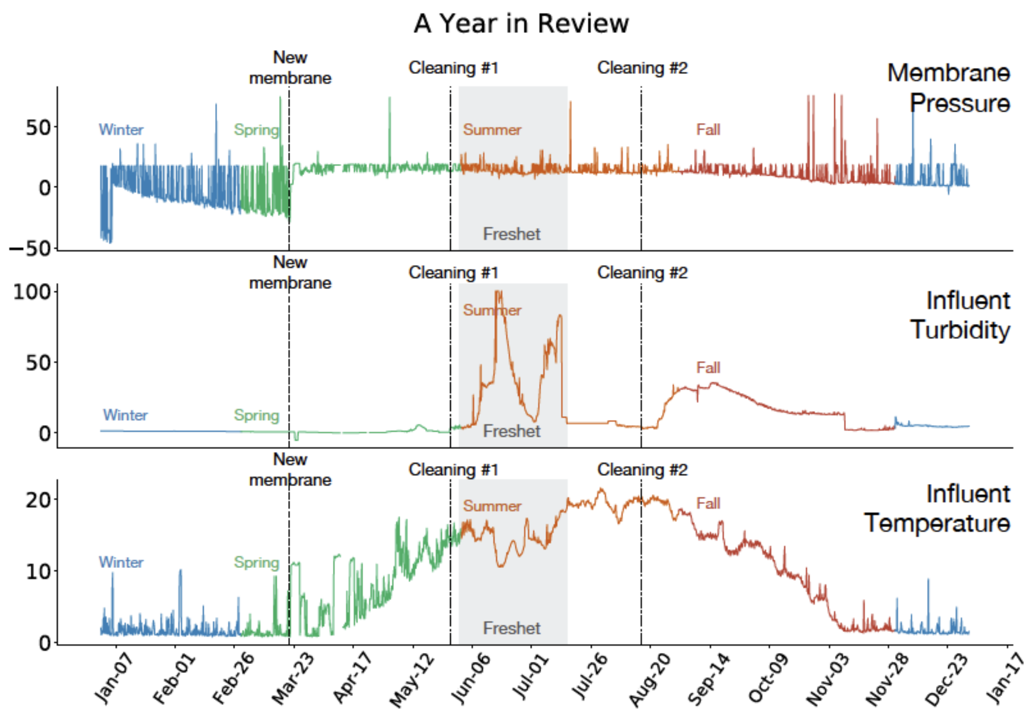

Figure 3.5: A year's worth of data for three different sensors, namely Membrane Pressure, Influent Turbidity, and Influent Temperature. These three sensors are representative of the impacts that seasonal variations, or change in physical state of plant's component have on the underlying telemetric stream of data. Note that the data is sub-sampled to avoid congestion in the plot.

and composition of the riverbed, which changes all the time. The chemical makeup of the water could spike if there is a change in farming practices in the area—water runoff from fields along the river. Even everyday things like a fire in the town can add huge pressure demands on the plant—many plants have dedicated pumps just for fires.

In all the examples above, it would be impractical to sensorize these events so they could be detected in the plant. In fact, we would need to predict these events in advance of their occurrence (including the weather) in order to accurately predict our data in advance. Perhaps, we could simply make predictions based on the entire history of the time-series—approximately two years of data. The history would still only approximate the state, because we do not know the starting conditions: data from five and ten years ago. In addition, such an approach is not scalable if the end goal is to build a continual

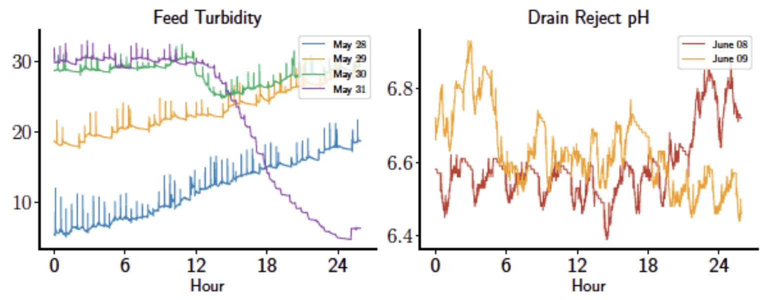

Figure 3.6: Feed Turbidity and Drain Reject pH sensors, respectively. An example of data drift in sensor values over hours, both within a day, and over multiple consecutive days.

learning system that runs for years generating tens of millions of samples a year.

In the end, capturing the true underlying state is likely impossible and we must be content using learning methods that continue to learn in deployment in order to achieve accurate prediction. Such methods track the changing underlying state of the plant. The idea is to use computation and extra processing of the recent data to overcome the limitations of the agent's state representation (Richard S. Sutton, Koop, and Silver 2007; Tao, A. White, and Machado 2022), similar to how an approximate model of the world can be used to deal with non-stationary tasks in reinforcement learning.

3.2.5 Details on Construction of State

As discussed at length before, learning directly on the raw data from a water treatment plant is very challenging due to the noisy, stochastic and partially-observable nature of the data. In addition to this, different sensors operate at different timescales and frequencies; we summarize some of the sensors in Table 3.2. In order to minimize the effect of these issues on the predictions, we take a series of preprocessing steps on the raw data.

Note that we do not have significant missing data issues. Our system rarely misses sensor readings. However, in the rare case where we do have a missing value, we simply use zero-imputation and fill in the missing values with zeros.

Categorical Observations

Some of the observations are recorded in the form of discrete categorical variables as opposed to continuous real numbers. For such observations, we encode them in a one-hot vector format. Consider an observation which can only take on values from one of k categories: we convert it into a binary vector of size k in which only the corresponding index of the category is set to 1.

Data Normalization

Since different sensors have different ranges, we normalize their values into the [0,1] range. For each sensor $\mathcal{X} = [x_0, x_1, x_2, \dots]$, we compute the minimum and maximum values from the logs over the duration of a year. Afterwards, we compute the normalized sensor value x'_t as:

$$x'_t = \frac{(x_t - \mathsf{min}(\mathcal{X}))}{(\mathsf{max}(\mathcal{X}) - \mathsf{min}(\mathcal{X}))} \tag{3.1}$$

Encoding Time of Day

The observations contain the information regarding the current time of the day in seconds. This is important since there are certain events that happen at a particular time of the day. Additionally, there are some events are repeated at regular intervals. Let $\mathcal{S} = [s_0, s_1, s_2, \dots]$ denote the time-stamp in seconds for that day, then we encode it using sine and cosine transforms as:

$$s_t^{(sin)} = \sin\left(\frac{2\pi s_t}{86400}\right) \tag{3.2}$$

$$s_t^{(cos)} = \cos\left(\frac{2\pi s_t}{86400}\right) \tag{3.3}$$

where $86,400$ is the total number of seconds present within a day. It is the maximum value that s_t can take.

Encoding Plant Mode Length

Understanding which mode the plant is in, and when the mode change will happen is crucial for the agent. This information is only available as a binary indicator, as mode value 1 against a certain mode indicates that the plant is currently in this mode, while it is 0 otherwise. This limitation to binary indication adds to the partial observability inherent in the state-space. We find that cyclically encoding the mode furnishes extra information that alleviates this associated partial observability. Since the agent has access to when the mode starts, and ends (albeit only as binary indicators), we utilize it to construct a cyclical thermometer encoding of the mode. Let o_t^i an ith observation at time t, and m_t^i be the mode indicator as part of this observation. For the sake of simplicity, assume that m_t^i is a production mode indicator, though the process is similar for all the modes. We define two thermometers w_s, and w_c as vectors of length 7 initialized to all zeros at the start, and the total mode length as m_l^i, which is a scalar value, and let s be the timestamp in seconds. Since the mode is characterized with respect to an observation that is furnished at a certain timestep, we avoid explicitly denoting mode length with timestep for clarity. The thermometers then get filled up at each timestep (which is in seconds), each index j of both thermometers gets filled by computing sine (for thermometer w_s) and cosine (for thermometer w_c) transforms by increasing periodicity.

$$w_{s_j} = \sin\left(2^j\pi\left(\frac{s}{m_l^i}\right)\right),\ w_{c_j} = \cos\left(2^j\pi\left(\frac{s}{m_l^i}\right)\right) \tag{3.4}$$

These thermometers, when plotted, have sine and cosine waves between the start and end of each mode, and their rotations about the period increase by a factor of 2^j. This equips the agent with the ability to understand when the mode is going to end, or when a mode shift is expected. Unlike binary indicators, this allows gradual increase towards the end of the mode with each new observation.

State Approximation and Summarizing History

In order to make use of the historical information during predictions, we compute memory traces of the observations. The state is constructed by appending the original observations and these memory traces, in addition to the mode length and time of day described in the above two sections. Given a normalized observation x'_t at time-step t, we compute its memory trace z_t using:

$$z_t = \beta z_{t-1} + (1 - \beta) x'_t \tag{3.5}$$

where β is the trace decay rate hyper-parameter. All the memory traces are initialized with zeros and are updated in an incremental manner when iterating over the dataset.

Sensor Name	Measures
Feed Flow PID	PID control for feed flow
Pump Flow PID	PID control for feed/drain pump flow
Permeate Pump Flow PID	PID control for permeate pump flow
Feed Water Sample	Condition of feed water sampling valve, indicating if it is open or not
Post Flocculation Sample	Condition of post flocculation sample isolation valve, indicating if it is open or not
Process/Permeate Pump Control Speed Output	Speed control for process/permeate pump
Sulphuric Acid Pump Dose Speed	Speed of sulphuric acid pump dosing
Hypochlorite Pump	Hypochlorite pump dosing
Sodium Hydroxide Pump Dose Speed	Sodium hydroxide pump dosing
Citric Acid Pump	Citric acid pump dosing
Feed Inlet Valve	Condition of feed inlet valve, indicating if it is open or not
Feed/Waste Pump Inlet	Condition of feed/waste pump inlet valve, indicating if it is open or not
Feed/Waste Pump Outlet	Condition of feed/waste pump outlet valve, indicating if it is open or not
Membrane Tank Outlet Valve	Condition of membrane tank outlet valve, indicating if it is open or not
Membrane Tank Recirculation Valve	Condition of membrane tank recirculation valve, indicating if it is open or not
Permeate Pump Recirculation Valve	Condition of permeate pump recirculation valve, indicating if it is open or not
Permeate Outlet Value	Condition of permeate outlet valve, indicating if it is open or not
BP/CIP Tank Inlet Valve	Condition of cleaning tank inlet valve, indicating if it is open or not
BP/CIP Tank Recirculation Valve	Condition of cleaning tank recirculation valve, indicating if it is open or not
Blower Inlet Valve	Condition of inlet blower's valve (A/B/C), indicating if it is open or not
Membrane Aeration Blower Control Speed Output	Control speed output of membrane aeration blower
Aeration Controller	Mode of the aeration (cyclic, constant, etc.)
Plant Mode	Mode of the plant (production, backwashing, etc.)

Table 3.2: Summary of a few sensors measuring pump speeds, setpoints valves, blowers, and PID control. All of these combine to form the agent-state space.

Chapter 4

Multi-step Predictions

We are interested in scalar predictions of multi-dimensional time-series, many steps into the future. On each discrete time-step, $t = 1, 2, ...$, the learning algorithm observes a new observation vector, $\mathbf{o}_t \in \mathbb{R}^d$, which form a sequence of vectors from the beginning of time.

$$\mathbf{o}_{0:t} \doteq \mathbf{o}_0, \mathbf{o}_1, \mathbf{o}_2, ..., \mathbf{o}_t$$

We do not assume knowledge of the underlying process that generates the series. That is, the next generation of observation vector may depend not just on $\mathbf{o}_{0:t}$, but other quantities not observable to the learning system. For example, the future turbidity the river water is impacted by future weather which is not observable and generally not predictable.

The goal is to estimate some scalar function of the future values of the time-series on time-step t, given $\mathbf{o}_{0:t}$. In this paper we focus on classical n-step predictions from time-series forecasting and exponentially weighted infinite horizon predictions commonly used in reinforcement learning, which we discuss in the following sections.

4.1 Classical Time-Series Forecasting

The first prediction problem we consider is simply predicting a component of the time-series on the next time-step, $o^{[i]}_{t+1}$. This scalar one-step prediction can be approximated as a function of a finite history of the time-series:

$$\hat{v}_t \doteq f_{\text{TS}}(o^{[i]}_{t-\tau:t}, \theta_t) \approx o^{[i]}_{t+1} \tag{4.1}$$

where $\theta_t \in \mathbb{R}^k$ is the learned weights. For a classical autoregressive model, f_{TS} is a linear function of this history $o^{[i]}_{t-\tau:t}$. More generally, f_{TS} can be any nonlinear function, such as one learned by a neural network.

In order to predict more than one step into the future we can iterate a one-step prediction model. The naive approach is to simply feed the model's prediction of the next observation into itself as input to predict the next step, now 2 steps into the future, and so on. For example a three step prediction:

$$\hat{v}_{t+2} \doteq f_{\text{TS}}([o^{[i]}_{t-\tau:t-1}, \hat{v}_{t+1}, \hat{v}_t]\theta_t) \approx o^{[i]}_{t+3} \tag{4.2}$$

Notice how two components of the history of the time series have been replaced by estimates. As we iterate the model beyond τ steps into the future all the inputs to f_{TS} will become model estimates.

Another approach is to directly learn a k-step prediction and avoid iterating altogether. One-step models are convenient because they can be updated on every timestep. Unfortunately, if the one-step model is inaccurate the model produces worse and worse predictions as you iterate it further. A *direct method* estimates a k prediction as a function of the history of the series:

$$\hat{v}_t \doteq f_{\text{DE}}(o^{[i]}_{t-\tau:t}, \theta_t) \approx o^{[i]}_{t+1+k} \tag{4.3}$$

In many applications we are interested in multi-dimensional data and in predicting many steps in the future. We can go beyond auto-regressive approaches by simply consider these time series prediction problems as supervised learning problems. For example, we can learn a neural network f_{DE} that inputs the last k multi-dimensional observation vectors $\mathbf{o}_{t-k:t}$ and predicts $o^{[i]}_{t+1+k}$, trained by constructing a dataset of pairs $(\mathbf{o}_{t-k:t}, o^{[i]}_{t+1+k})$. We can also go beyond finite k-length histories, and use recurrent neural networks, which is becoming a more common practice in time series prediction. When we move to using this supervised learning framing, we lose some of the classical strategies for dealing with correlation in the data, but in general, evidence is mounting that we can obtain improved performance (Crone, Hibon, and Nikolopoulos 2011; Hewamalage, Bergmeir, and Bandara 2021).

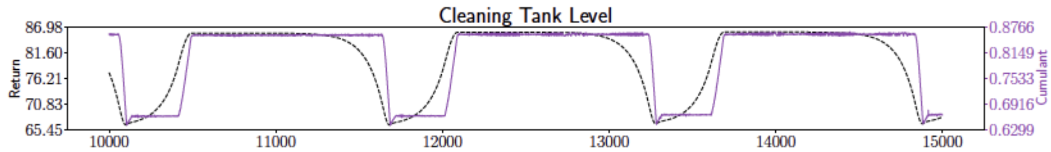

Figure 4.1: A sample time-series of tank level from a real water-treatment plant and an idealized prediction (labeled return). The prediction is ideal in the sense that we can simply compute the exponentially weighted sum in Equation 4.4 given a dataset—the idealized prediction is not the output of some estimation procedure. Later we will show learned predictions and how the match the ideal. Notice how the idealized prediction increases well before the time-series reaches its maximum value, and falls well before the time-series does. In this way, the idealized prediction at any point in time provides an anticipatory measure of the rise or fall of the data in the future.

4.2 GVFs and Temporal Difference Learning

In reinforcement learning, multi-step predictions are formalized as value functions. Here the objective is to estimate the discounted sum of all the future values of some observable signal, with discount $\gamma \in [0, 1)$

$$G_t \doteq \sum_{j=0}^{\infty} \gamma^j o^{[i]}_{t+1+i} \tag{4.4}$$

Technically G_t summarizes the infinite future of the time-series, but values of $o^{[i]}$ closer to time t contribute most to the sum. These exponentially weighted summaries of the future automatically smooth the underlying data $o^{[i]}$—potentially making estimation easier—and provide a continuous notion of anticipation of the future as discussed in Figure 4.1. For this reason they have been called "Nexting" predictions (Modayil and Richard S Sutton 2014), but more generally were introduced as general value functions (GVFs) (Richard S Sutton, Modayil, et al. 2011), where they generalize the notion of a value by allowing any cumulant to be predicted beyond a reward.

GVF predictions can be learned using temporal difference learning. As before, the prediction is approximated with a parameterized function, $f_{\text{TD}}(\mathbf{s}_t, \theta) \approx G_t$, where $\mathbf{s}_t$ is a summary of the entire series, $\mathbf{o}_{0:t}$, up to time t. The prediction on time-step t is updated using the temporal-difference error:

$$\theta_t \leftarrow \theta_t + \alpha(c_t + \gamma f_{\text{TD}}(\mathbf{s}_t, \theta) - f_{\text{TD}}(\mathbf{s}_{t-1}, \theta))\nabla f_{\text{TD}}(\mathbf{s}_{t-1}, \theta) \tag{4.5}$$

where $\alpha \in (0,1]$ and $c_t \doteq o_t^{[i]}$.

Chapter 5

Algorithms

We investigate methods that can be pre-trained from offline logged data and perform fine-tuning in deployment. The algorithms we investigate can be used offline, online, or a combination of the two. Offline algorithms can randomly subsample and update from the offline data as much as needed (i.e., until the training loss converges). Online data, generated in the *deployment phase* can only be resampled from a replay buffer once it has been observed. Using the online data is restricted: the algorithms cannot look ahead into the future of the time-series, they must wait for each data point to become available step-by-step. After a sample is observed it can be resampled over and over via a replay buffer. In this chapter, we outline the algorithms, and how they can combine offline and online learning. We present the algorithmic details, including discussion of methods for both GVF and n-step predictions. Furthermore, we also describe our hyperparameter selection procedure.

5.1 Constructing State

Although many of these algorithms can be used in real-time, making and updating predictions live as the plant is operating, we only simulate that setting here using a static dataset. Online data, the one we simulate, is processed as a stream, one sample at a time, as if it were generated live. The offline batch of data is $\mathcal{D}_{\text{offline}} = \{(o_t, c_{t+1}, o_{t+1})\}_{t=1}^{N}$, where N is the total number of transitions, $o_t \in \mathbb{R}^d$ is the observation vector, $c_{t+1} \in \mathbb{R}$ is the signal to predict or *cumulant*, o_{t+1} is the next observation vector.

The data, however, is partially observable and the agent should construct an approximate state. A typical approach using in machine learning is to use recurrent neural networks, to summarize history (Hochreiter and Schmidhuber 1997; Cho et al. 2014; Hausknecht and Stone 2015; Vinyals et al. 2019). However, we found for our sensor-rich problem setting, that a simpler memory-based approach was just as effective and much easier to train. The general idea an exponentially weighted moving average of the observations; such an exponential memory trace has previously been shown to be effective (c.f. Tao, A. White, and Machado 2022; Mozer 1989; Rafiee et al. 2023). We include more explicit details on how we created our approximate state observation vector in Chapter 3.

Once we have constructed this approximate state vector, which we denote $\hat{s}_t \in \mathbb{R}^{d+k}$, we then apply the algorithms directly on this $\hat{s}_t$ without further considering history or state estimation. These augmentations are sometimes referred to as auxiliary inputs and, in toy problems, it has been shown they alleviate the problem of partial observability in reinforcement learning (Tao, A. White, and Machado 2022). In other words, we construct an augmented dataset $\mathcal{D}_{\text{augmented}} = \{(\hat{s}_t, c_{t+1}, \hat{s}_{t+1})_{t=1}^{N}\}$ and apply our algorithms as if we have access to the environment state—namely as if we are in the fully observable setting. All the algorithms we consider use a neural network f to compute the prediction $f_{w_t}(\hat{s}_t)$.

Once we have constructed this approximate state vector, which we denote $\hat{s}_t \in \mathbb{R}^{d+k}$, we then apply the algorithms directly on this $\hat{s}_t$ without further considering history or state estimation. These augmentations are sometimes referred to as auxiliary inputs and, in toy problems, it has been shown they alleviate the problem of partial observability in reinforcement learning (Tao, A. White, and Machado 2022). In other words, we construct an augmented dataset $\mathcal{D}_{\text{augmented}} = \{(\hat{s}_t, c_{t+1}, \hat{s}_{t+1})_{t=1}^{N}\}$ and apply our algorithms as if we have access to the environment state—namely as if we are in the fully observable setting. All the algorithms we consider use a neural network f to compute the prediction $f_{w_t}(\hat{s}_t)$, where $\boldsymbol{w}$ are the parameters of the neural network. The predictions may either be GVF predictions or n-step time series predictions, with the algorithms described in the next two sections.

5.2 Algorithms for GVFs

The goal for GVF predictions it to estimate the expected discounted sum of future cumulants, as described in Section 4.2. The simplest approach is to simply use textbook 1-step temporal difference (TD) learning Richard S Sutton and Barto 2018. Data is processed as a stream, one sample at a time. The approach is summarized in Algorithm 1.

Algorithm 1 OnlineTD

```
1: Hyperparameters: stepsize $\eta > 0$
2: Initialize $\boldsymbol{w}_0$: the weights of the network (e.g., uniform)
3: Obtain initialize observation $o_t$ for $t = 0$, set $\hat{s}_0 = o_0$
4: while in deployment do
5:     Observe next observation $o_{t+1}$ and cumulant $c_{t+1}$
6:     $\hat{s}_{t+1} \leftarrow U(o_{t+1}, \hat{s}_t)$                ▷ compute augmented observation vector
7:     $v_{t+1} \leftarrow f_{w_t}(\hat{s}_{t+1})$                        ▷ compute prediction
8:     $\delta_t \leftarrow c_{t+1} + \gamma v_{t+1} - f_{w_t}(\hat{s}_t)$               ▷ compute the TD error
9:     $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \eta \delta_t \nabla f_{w_t}(\hat{s}_t)$      ▷ or Adam using $-\delta_t \nabla f_{w_t}$ as a gradient
10:    $t \leftarrow t + 1$
11: end while
```

We can also adapt this update to an offline dataset. We can use TD offline making multiple passes over the data set, updating the network weights via mini-batches. Here we follow the standard approach used in offline RL, for the fully observable setting. In other words, we can treat each tuple $(\hat{s}_t, c_{t+1}, \hat{s}_{t+1})$ separately, without having to keep the data in order—In contrast, if we were using a recurrent neural network, we would need to more carefully maintain dataset order. In each epoch, we shuffle the dataset $\mathcal{D}_{\text{augmented}}$ and update the neural network using a mini-batch TD update. Algorithm 2 summarizes the approach.

We use the Adam optimizer (Kingma and Ba 2015) to update with the mini-batch TD updates. We set all but the stepsize η to the typical default values: momentum parameter to 0.9, exponential average parameter to 0.99, and the small constant in the normalization to 10^{-4}. The algorithm returns

Algorithm 2 OfflineTD

```
1: Hyperparameters: stepsize $\eta > 0$, batchsize $k$, number of epochs $n_{\text{epochs}}$,
2: Input $\mathcal{D}_{\text{augmented}} = \{(\hat{s}_t, c_{t+1}, \hat{s}_{t+1})\}$
3: Initialize $\boldsymbol{w}$: the weights of the network (e.g., uniform)
4: Initialize $s_{\text{opt}}$: the state of the optimizer (e.g., zero momentum, zero exponential average)
5: for epoch in $n_{\text{epochs}}$ do
6:     for batch in $\mathcal{D}_{\text{offline}}$ do
7:         $\Delta \leftarrow -\frac{1}{k}\sum_{i \in \text{batch}} \left(c_{i+1} + \gamma f_{\boldsymbol{w}}(\hat{s}_{i+1}) - f_{\boldsymbol{w}}(\hat{s}_i)\right) \nabla f_{\boldsymbol{w}}(\hat{s}_i)$
8:         $\boldsymbol{w}, s_{\text{opt}} \leftarrow \texttt{opt}(\boldsymbol{w}, \Delta, \eta, s_{\text{opt}})$
9:     end for
10: end for
11: Return $\boldsymbol{w}, s_{\text{opt}}$
```

Algorithm 3 OnlineTD using Offline Pretraining

```
1: Hyperparameters: offline stepsize $\eta > 0$, batchsize $k$, number of epochs $n_{\text{epochs}}$, online stepsize $\alpha > 0$, number of replay steps $n_{\text{replay}}$
2: Input $\mathcal{D}_{\text{augmented}} = \{(\hat{s}_t, c_{t+1}, \hat{s}_{t+1})\}$
3: $\boldsymbol{w_0}, s_{\text{opt}} = \text{OfflineTD}(\mathcal{D}_{\text{augmented}}, \eta, k, n_{\text{epochs}})$
4: Initialize the replay buffer $\mathcal{B}$ with last $n_{\text{replay}}$ samples in $\mathcal{D}_{\text{augmented}}$
5: Obtain initial observation $o_t$ for $t = 0$, set $\hat{s}_0 = o_0$
6: while in deployment do
7:     Observe next observation $o_{t+1}$ and cumulant $c_{t+1}$
8:     $\hat{s}_{t+1} \leftarrow U(o_{t+1}, \hat{s}_t)$          ▷ compute augmented observation vector
9:     $v_{t+1} \leftarrow f_{\boldsymbol{w}_t}(\hat{s}_{t+1})$          ▷ compute prediction
10:    $\delta_t \leftarrow c_{t+1} + \gamma v_{t+1} - f_{\boldsymbol{w}_t}(\hat{s}_t)$          ▷ compute the TD error
11:    $\boldsymbol{w}_{t+1}, s_{\text{opt}} \leftarrow \texttt{opt}(\boldsymbol{w}_t, -\delta_t \nabla f_{\boldsymbol{w}_t}(\hat{s}_t), \alpha, s_{\text{opt}})$
12:    $t \leftarrow t + 1$
13: end while
```

the state of the optimizer—such as the exponential averages of squared gradients and momentum—because our online variants continue optimizing online starting with this optimizer state.

We expect purely offline methods to perform poorly in our nonstationary (partially observable) setting compared with those that update in deployment. The offline data may not perfectly reflect what the agent will see in deployment, and, in general, tracking—namely updating with the most recent data—can also help under partially observability.

We can combine offline and online, by pre-training offline and then allowing the agent to continue learning online. The primary nuance here is that we can either continue to use a replay update online, or switch to the simplest online variant of TD that simply updates once per sample. We found that the simpler update was typically just as good as the variant using replay, and so we use this simpler variant in this work. We summarize this procedure in Algorithm 3, and for completeness include the replay variant and results comparing to it (this is presented later in Chapter 6).

It is worth mentioning that we could further improve these algorithms with the variety of advances combining TD and neural networks. TD can diverge when used with neural networks (Tsitsiklis and Van Roy 1997), and several new algorithms have proposed gradient-based versions of TD that resolve the issue (Dai, He, et al. 2017; Dai, Shaw, et al. 2018; Patterson, A. White, and M. White 2022). In control, a common addition is the use of target networks, which fix the bootstrap targets for several steps (Mnih et al. 2015). We found for our setting that the simple TD algorithm was effective, so we used this simpler approach.

5.3 Algorithms for N-Step Predictions

We can similarly consider the offline and online variants of n-step predictions. The offline dataset[1] consists instead of $\mathcal{D}_{\text{n-step}} = \{(\hat{s}_t, c_{t+n})\}_{t=0}^{N-n}$ where we predict the cumulant n steps into the future from t, given the approximate state $\hat{s}_t$. The targets for GVF predictions were returns G_t—discounted sums of cumulants into the future—whereas the targets for n-step predictions is the cumulant exactly n steps in the future. Learning f_w offline corresponds to a regression problem on this dataset, which can be solved using any standard techniques. Similarly to OfflineTD, we use stochastic mini-batch gradient descent and the Adam optimizer.

As a supervised learning problem, it is straightforward to update in deployment, online. However, there is one interesting nuance here, that the targets are not observed until n steps into the future. The online algorithm, therefore,

[1]The underlying data is the same as in the TD setting, but the targets are different, and so we explicitly construct a supervised learning dataset from this underlying data.

Algorithm 4 OnlineNStep using Offline Pretraining

```
1: Hyperparameters: offline stepsize $\eta > 0$, batchsize $k$, number of epochs $n_{\text{epochs}}$, online stepsize $\alpha > 0$
2: Input $\mathcal{D}_{\text{n-step}} = \{(\hat{s}_t, c_{t+n})\}$
3: $\boldsymbol{w}_\mathbf{0}, s_{\text{opt}} =$ OfflineNStep($\mathcal{D}_{\text{n-step}}$, $\eta$, $k$, $n_{\text{epochs}}$)
4: Create size $n$ circular array PastStates set index ind $\leftarrow 0$
5: Obtain initial observation $o_t$ for $t = 0$, set $\hat{s}_0 \leftarrow o_0$
6: PastStates[ind] $\leftarrow \hat{s}_0$, and ind $\leftarrow 1$
7: for $n - 1$ steps do                                  ▷ store first $n$ inputs
8:     Observe next observation $o_{t+1}$ and cumulant $c_{t+1}$
9:     $\hat{s}_{t+1} \leftarrow U(o_{t+1}, \hat{s}_t)$
10:    PastStates[ind] $\leftarrow \hat{s}_{t+1}$
11:    $t \leftarrow t + 1$ and ind $\leftarrow$ ind $+ 1$
12: end for
13: ind $\leftarrow 0$
14: while in deployment do
15:    Observe next observation $o_{t+1}$ and cumulant $c_{t+1}$
16:    $(s, c) \leftarrow (\text{PastStates[ind]}, c_{t+1})$
17:    $\Delta \leftarrow \big(f_{\boldsymbol{w}_t}(s) - c\big) \nabla f_{\boldsymbol{w}_t}(s)$
18:    $\boldsymbol{w}_{t+1}, s_{\text{opt}} \leftarrow \texttt{opt}(\boldsymbol{w}_t, \Delta, \alpha, s_{\text{opt}})$
19:    $\hat{s}_{t+1} \leftarrow U(o_{t+1}, \hat{s}_t)$
20:    PastStates[ind] $\leftarrow \hat{s}_{t+1}$
21:    $t \leftarrow t + 1$ and ind $\leftarrow \text{mod}(\text{ind}, n)$
22: end while
```

has to *wait* to update the prediction $f_w(\hat{s}_t)$ until it sees the outcome c_{t+n} at time step $t + n$. This involves maintaining a short buffer of size n, until we can obtain the pair $(\hat{s}_t, c_{t+n})$. This procedure is summarized in Algorithm 4.

Though seemingly a minor issue, it is less ideal that the OnlineNStep algorithm has to wait n steps to update the prediction for input $\hat{s}_t$. The TD algorithm for GVF predictions, on the other hand, does not have to wait to update, because it bootstraps off of its own estimates. Instead of using bootstrapping, we could have used a Monte Carlo algorithm, that regresses $\hat{s}_t$ towards computed returns, turning this into a supervised learning problem like for the n-step time series problem. However, it has been shown that being able to update immediately can result in faster tracking (Richard S. Sutton, Koop, and Silver 2007; Richard S Sutton and Barto 2018), and typically TD algorithms are preferred to Monte Carlo algorithms. The issue is worse for

Monte Carlo than for n-step targets, because the returns extend further than n steps into the future, but nonetheless there is some suggestive evidence that algorithms that need to wait could be disadvantageous.

5.4 Selecting Hyperparameters for Deployment

The above algorithms have many hyperparameters. Fortunately, we can use a simple validation strategy to select them, including the *online* stepsize parameters. The key idea is to treat the validation just like a deployment scenario, where the agent updates in temporal order on the dataset. For example, consider selecting the offline stepsize η and online stepsize α, assuming all other hyperparameters are specified (number of offline epochs is fixed, etc.). Then we can evaluate each hyperparameter pair (η, α) by following simple steps.

1. Splitting the dataset into a training and validation set,
2. Pre-training with offline stepsize η on the training set,
3. Updating online with the stepsize α on the validation set (in one pass) as if it is streaming, recording the prediction accuracy as the agent updates.

The online prediction accuracy is computed as follows. For the current weights $\boldsymbol{w}_t$, the agent gets $\hat{s}_t$ and makes a prediction $\hat{v}_t = f_{\boldsymbol{w}_t}(\hat{s}_t)$. Because we (the experimenter) can peek ahead in the validation set, we can compute the error $\mathrm{err}_t = (\hat{v}_t - c_{t+n})^2$. The agent, of course, cannot peek ahead, since it would not be able to do so in deployment. After going through the validation set once, we have our set of errors. Note that we only evaluate $\boldsymbol{w}_t$ on the pair $(\hat{s}_t, c_{t+n})$. Right after this step, we update the weights to get $\boldsymbol{w}_{t+1}$ and then evaluate the prediction under these new weights for the next step: $\hat{v}_{t+1} = f_{\boldsymbol{w}_{t+1}}(\hat{s}_{t+1})$.

This validation procedure helps us pick a suitable pair of (η, α) precisely because validation mimics deployment. We want η to be chosen to produce a good initialization and we want α to be chosen to facilitate tracking when updating online. For example, if α is too big for tracking (or fine-tuning), then the validation error will be poor because the weights will move away from a good solution while updating on the validation set and the errors will start to get larger, resulting in a poor final average validation error. As another example, if η is too small and does not converge on the training set within the given number of epochs, then the initialization will not be as good and the validation errors will start higher than they otherwise could, until the online updating starts to reduce them.

Though this hyperparameter selection approach is described specifically for n-step predictions with the offline and online stepsizes, it can be used for TD as well as for other hyperparameters. The key point is that, even though the offline hyperparameters are only used on the training set and the online hyperparameters only when updating on the validation, they are both jointly evaluated based on validation error. The primary difference for TD is simply that the target is different. Again, because we the experimenter can look ahead in the data, we can simply compute the return on the future data, and compute the errors $\text{err}_t = (f_{w_t}(\hat{s}_t) - G_t)^2$.

Chapter 6

Experiments & Design

In this chapter, we discuss the experimental details and results. We first detail our dataset splits, and the function approximator, neural network, we design for the experiments, including the learning procedure and hyperparameters. We then detail the posed questions and discuss the results backed by empirical evidence.

6.1 Experimental Details

We consider five consecutive days of data from the middle of November, 2022 for first set of experiments. The first four days are used as the offline training logs while the fifth day is used for the deployment phase. We use the final 2% of the offline training logs as the validation data, which is used for selecting hyper-parameters. For the final experiment, to better test the impacts of nonstationarity, we use data from the duration of an entire month: the data from first 24 days of November are used as the offline training logs and the last day of the month (30th November) is used as the deployment data.

All the methods share similar settings. We train a 2-layer feed-forward neural network with 512 units in each layer with ReLU activation functions. This network is optimized for 2000 epochs using the Adam optimizer with an L2 weight decay rate of $\lambda = 0.3$ and a batch size of 512, in the offline phase. After the offline training phase ends, we save the optimizer state variables and use them to initialize the optimizer during the deployment phase. In deployment, the algorithms update on one sample at a time, and use a

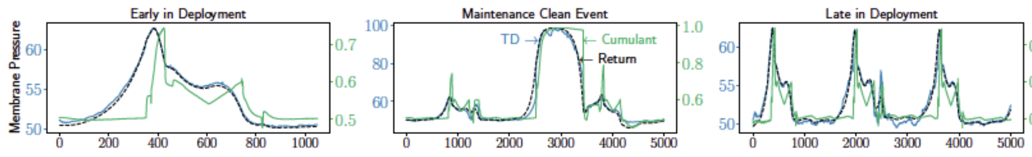

Figure 6.1: Predictions of the filter membrane pressure roughly 100 seconds into the future. The plot shows the pressure sensor in green labelled cumulant (whose magnitude corresponds to the right y-axis). We show three snippets of the deployment data. The first in subplot a thousand seconds at the beginning of deployment. The middle subplot shows data during a maintenance clean, and the last subplot features data near the end of the deployment phase (24 hours later). Each subplot highlights a different characteristic pattern in pressure change. The blue curve shows the TD prediction, first trained offline, then updated in deployment. The return represents the ideal prediction and is plotted in black. Note both the TD prediction and the return use the left blue axis. The TD predictions tightly match the target's pattern in all three scenarios.

different online stepsize. For all the methods we use the validation procedure described in Chapter 5 to select the learning rates. We swept over offline learning rates $\eta \in \{1\text{x}10^{-3}, 1\text{x}10^{-4}, 1\text{x}10^{-5}, 1\text{x}10^{-6}, 1\text{x}10^{-7}\}$ and online learning rates $\alpha \in \{1\text{x}10^{-4}, 1\text{x}10^{-5}, 1\text{x}10^{-6}, 1\text{x}10^{-7}, 1\text{x}10^{-8}\}$. The validation procedure is done separately for each algorithm and sensor.

6.2 Posed Questions & Answers

A natural first question is can we predict the time series well in deployment, given the size, complexity, and partially observable characteristics of our data. From there we contrast the GVF predictions to n-step predictions, to better understand the GVF results relative to a well-understood multistep prediction. Finally, we investigate one of the key claims in this work: does learning in deployment help or is offline learning all we need?

6.2.1 GVF Predictions are Accurate in Deployment

The object of our first set of results is to gain some intuition about GVF predictions. Although widely used in RL to model the utility or value of a policy, exponentially weighted predictions are uncommon. In Figure 6.1 we visualize predictions from the OnlineTD approach[1] on one sensor at three different periods of time in deployment. Here we plot the cumulant (sensor value to be predicted into the future), the prediction, and the return—our stand in for an *idealized prediction*. The time series of the return changes before the cumulant, because the return summarizes the future values of the cumulant. A good prediction should closely match the return as we see in the figure.

In the middle subplot of Figure 6.1 we see a large perturbation in the cumulant corresponding to a difficult to predict event. This event, a maintenance clean, happens in the early morning. This causes a large increase in pressure on the filter, and unlike the vast majority of the training data, this increase is sustained for a long period of time. We can see the prediction correctly anticipates this event but does not get the precise shape of the prediction correct.

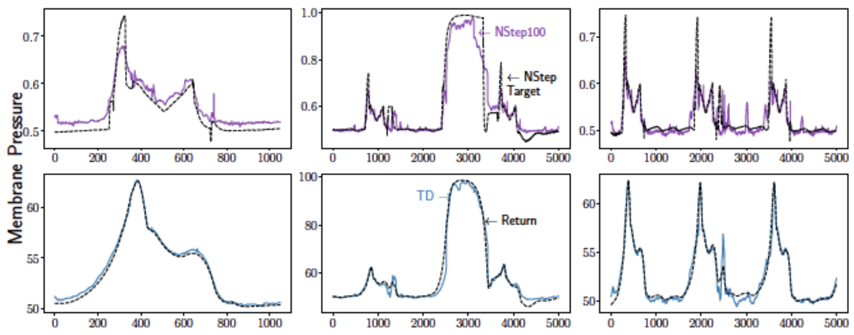

Figure 6.2: Comparing GVF Predictions (blue) and n-step predictions (purple) of filter membrane pressure. The top row shows the n-step predictions on the same three segments of deployment data used in Figure 6.1. Here we only plot the prediction (labelled TD and NStep100), and the ideal prediction (labelled return and NStep target). Generally both types of predictions are well aligned with their respective targets, however, sometimes the n-step prediction is off. Figure 6.3 includes the results for several other sensors.

[1]All of our results are with pre-training, as this performed significantly better than without using the offline data at all. This result is to be expected. Further our OnlineTD algorithm with pre-training also leverages the offline data to automatically set all hyper-parameters, providing a fully specified algorithm. The conclusion for our setting is that it simply makes the most sense to leverage offline data, rather than learning from scratch.

6.2.2 Comparing GVF and N-Step Predictions

To help calibrate our performance expectations, and provide a point of comparison we also learned and plotted the more conventional 100-step predictions of future membrane pressure in deployment in Figure 6.2. We chose a horizon of 100 steps to provide rough alignment with the horizon of a $\gamma = 0.99$ GVF prediction. The figure shows the n-step prediction and the GVF prediction on the same segments of data in deployment.

The plot of the n-step prediction and the shifted cumulant (labelled NStep Target) should align if the predictions are accurate. At least for membrane pressure, the GVF predictions better match their prediction target (the return) compared with n-step predictions.

Generally, across sensors the learned GVF predictions are smoother than their n-step counterparts as shown in Figure 6.3. This is perhaps to be expected because the γ weighting in the GVF prediction targets smooths the raw sensor data. If there are sharp, one time-step spikes, as we see in the Inlet Pressure date, the n-step target itself will be spikey—that is, the ideal prediction is not smooth. Otherwise the main objective of Figure 6.3 is to allow you the reader to better understand GVF predictions by simply visually comparing them with n-step predictions—something that is easy to interpret and you might have more natural intuitions for.

	Membrane Pressure	Influent Temperature	Inlet Pressure	Inlet Flow Transmitter	Drain Reject Pressure
TD	0.129	0.313	0.047	0.030	0.060
NStep100	0.212	0.768	0.182	0.190	0.306

Table 6.1: Normalized MSE averaged over deployment for GVF and n-step predictions on five different sensors.

One perhaps surprising conclusion from Figure 6.3 is that the GVF and n-step predictions look surprisingly similar, and thus it is reasonable to ask if there are reasons to prefer one to the other. From a performance perspective we compare the two in Table 6.1 reporting the Normalized Mean Squared Error

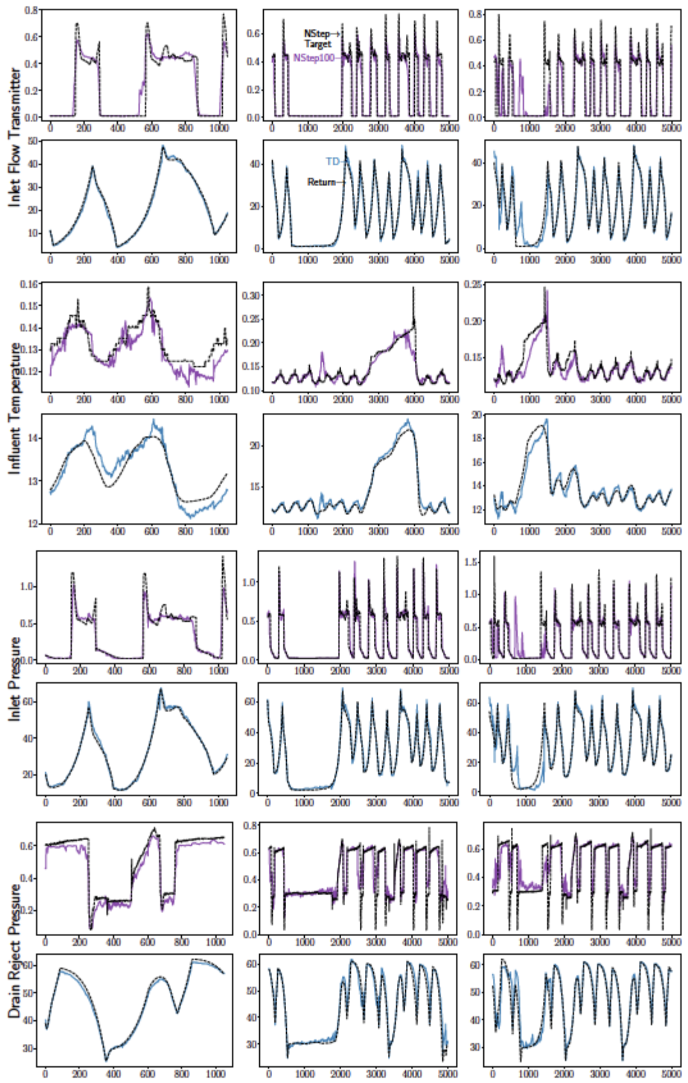

Figure 6.3: Comparing n-step predictions (Row 1) and GVF Predictions (Row 2) across several sensors. The structure of this plot mirrors Figure 6.2.

averaged over the deployment data:

$$\text{NMSE}_t \doteq \frac{\text{MSE}_t}{\sigma^2(G_t)} \qquad \text{where} \quad \text{MSE}_t \doteq \overline{(\hat{v}_t - G_t)^2}$$

and $\overline{x}$ denotes the exponentially weighted moving average of the squared GVF prediction error over the deployment data. Similarly, $\overline{\sigma^2}(G_t)$ denotes the variance of the returns up to time t computed using an exponentially weighted variant of Welford's online algorithm. The NMSE is equivalent to the variance unexplained and is a simple ratio measure of the MSE of the predictor to the MSE of the mean prediction. NMSE less than one indicates that the prediction explains more variance in the data than a mean prediction. We use the exponential moving average variants of these measures because our data is non-stationary. Finally, NMSE for n-step prediction can be computed by replacing G_t with $o^{[i]}_{t+100}$ in the above equations. Across five sensors, the NMSE is lower for GVF predictions compared with n-step predictions, as shown in Table 6.1.

Algorithmically, GVF predictions are interesting for several reasons. GVF predictions can be updated, via TD, online and incrementally from a stream of data, whereas n-step predictions involve storing the data and waiting 100 steps until the prediction targets are observed. The longer the prediction horizon, the longer the system must wait without updating the predictions in between. In contrast, TD methods by their recursive construction have memory and computational requirements independent of the prediction horizon—independent of γ. These points highlight the potential of GVF predictions for time-series prediction, as an additional choice for multistep predictions. For any given applications the ultimate choice of prediction type and learning method will be driven my many factors.

6.2.3 Mitigating Partial Observability via Online Adaption

As shown extensively in Chapter 3 the data from our plant is highly partially observable, appearing non-stationary when plotted. For the smaller dataset we used in the experiments so far, however, we find that the agent trained only on offline data predicts the deployment data well. This outcome is not surprising because the training data was collected from only four days and the deployment was the following 24 hour period. It is reasonable to expect that the data is mostly stationary during this period since there would be no major seasonal weather changes, unexpected events like fires are rare, and sensor fouling takes weeks to show up in the data stream.

To highlight the need for online learning and demonstrate how changes in the data can significantly impact non-adaptive approaches, we used a dataset of 24 days for training and a deployment day six days later. Figure 6.4 com-

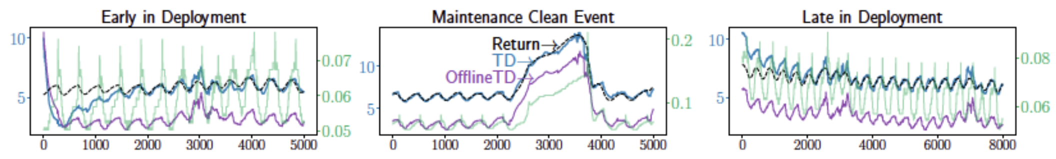

Figure 6.4: Comparing GVF with OfflineTD and Online TD on deployment data six days after training data had ended. Both agents trained offline of data from November 1st to November 24th. The deployment data was taken from a 24 hour period on November 30th. In this case we expect a significant distribution shift between training and deployment data. The result clearly shows this, both predictors start of far from the ideal target. Only methods that learn in deployment can close the gap.

pares GVF predictions of a frozen pre-trained agent with the online TD agent that was pre-trained on the sample data but continues to learn in deployment. Due to the differences in the training data and the deployment data both predictors start far from the ideal prediction (the return), but only online TD can adjust as shown in the first subplot. Throughout the remainder of the deployment data online TD predictions continued to match their targets.

This result not only highlights when online methods can be beneficial but also mimics a fairly realistic deployment scenario. Oftentimes, when working with real systems, we cannot always access the most recent data. An industrial partner might have limited data logs; or sometimes technical problems cause logs to be lost. In our specific application, water treatment, the training data might be out of date because the plant could have been out of commission. Regardless of the reason, its useful that simple online methods like TD can adapt to such situations.

Additionally, for the result in Figure 6.4, we pre-trained the predictions on 24 days of data instead of four. We made this choice to give the pre-trained agent the best chance to extract as much as possible from a wide range of operating conditions represented by nearly a month of data.

6.2.4 Does Replay Help When Learning Online?

Replay has been an important addition to RL algorithms (Mnih et al. 2015). Replay buffers help the agent 'replay' previously seen experience. In general

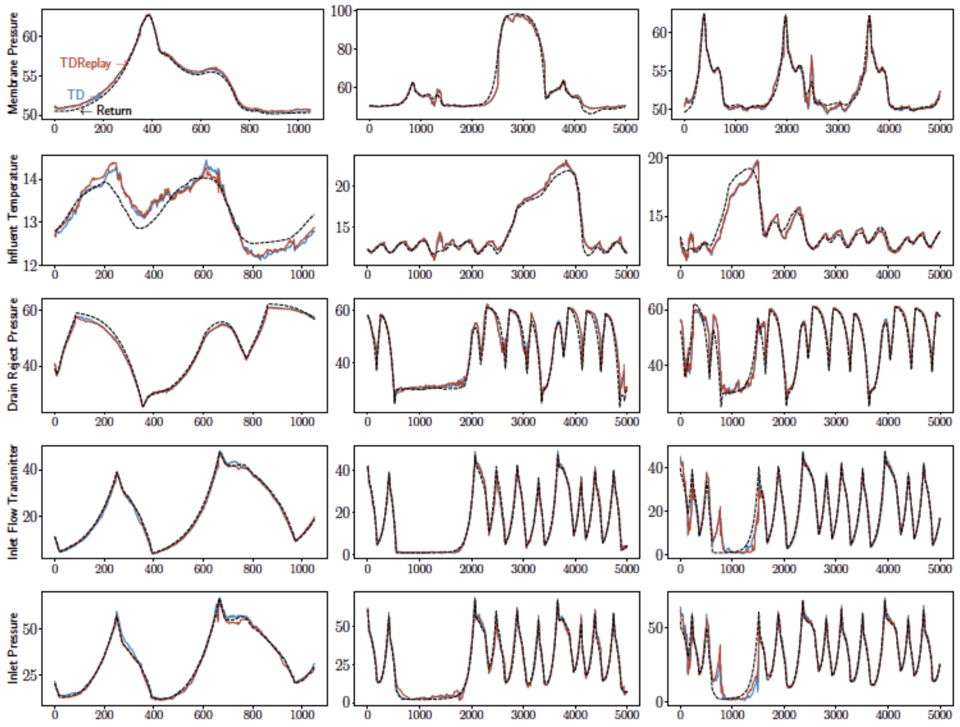

Figure 6.5: Comparing online TD and online TD with replay for GVF Predictions across several sensors. The structure of this plot mirrors Figure 6.2.

they prove to be helpful essentially because they break correlation between consecutive transition samples. Without replay buffers, it could be the be case that learning is inefficient. However, the addition of replay buffers is not universal and is highly dependent on the data, and the learning algorithm.

In this auxiliary experiment, we want to understand if online learning can benefit with replay buffers. Therefore, we considered both the simpler online TD update in deployment, as well as using TD with replay. The TD with replay algorithm is summarized in Algorithm 5. We found, though, that they performed very similarly (see Figure 6.5), so we used the simpler online TD update in the main body.

Algorithm 5 TDwithReplay using Offline Pretraining

```
1: Hyperparameters: offline stepsize $\eta > 0$, batchsize $k$, number of epochs $n_{\text{epochs}}$, online stepsize $\alpha > 0$, number of replay steps $n_{\text{replay}}$
2: Input $\mathcal{D}_{\text{augmented}} = \{(\hat{s}_t, c_{t+1}, \hat{s}_{t+1})\}$
3: $\boldsymbol{w_0}, s_{\text{opt}} = \text{OfflineTD}(\mathcal{D}_{\text{augmented}}, \eta, k, n_{\text{epochs}})$
4: Initialize the replay buffer $\mathcal{B}$ with last $kn_{\text{replay}}$ samples in $\mathcal{D}_{\text{augmented}}$
5: Obtain initial observation $o_t$ for $t = 0$, set $\hat{s}_0 = o_0$
6: while in deployment do
7:     Observe next observation $o_{t+1}$ and cumulant $c_{t+1}$
8:     $\hat{s}_{t+1} \leftarrow U(o_{t+1}, \hat{s}_t)$
9:     $v_{t+1} \leftarrow f_{w_t}(\hat{s}_{t+1})$
10:    $\delta_t \leftarrow c_{t+1} + \gamma v_{t+1} - f_{w_t}(\hat{s}_t)$
11:    $\boldsymbol{w} \leftarrow \boldsymbol{w_t} + \alpha\delta_t \nabla f_{w_t}(\hat{s}_t)$
12:    Add tuple $(\hat{s}_t, c_{t+1}, \hat{s}_{t+1})$ to $\mathcal{B}$
13:    for $n_{\text{replay}}$ steps do
14:        Sample a random mini-batch batch of size $k$ from $\mathcal{B}$
15:        $\Delta \leftarrow -\frac{1}{k}\sum_{i\in\text{batch}} \left(c_{i+1} + \gamma f_{w_t}(\hat{s}_{i+1}) - f_w(\hat{s}_i)\right) \nabla f_w(\hat{s}_i)$
16:        $\boldsymbol{w}, s_{\text{opt}} \leftarrow \texttt{opt}(\boldsymbol{w}, \Delta, \alpha, s_{\text{opt}})$
17:    end for
18:    $\boldsymbol{w_{t+1}} = \boldsymbol{w}$
19:    $t = t + 1$
20: end while
```

Chapter 7

Conclusion

In this dissertation we took the first steps toward optimizing and automating water treatment on a real plant. Before we can hope to control such a complex industrial facility, we must first ensure that learning of any kind is feasible. This work represents such a feasibility study. We provided extensive visualization and analysis of our plant's data, highlighting how it generates a large, high-dimensional data stream that exhibits interesting structure at the second, minute, day and month timescales. Unlike the data commonly used in RL benchmarks, ours is subject to seasonal trends, mechanical wear and tear, making it highly non-stationary. Through a combination of feature engineering and extensive offline pre-training from operator data we were able to learn accurate multi-step predictions encoded as GVFs. Compared with classical n-step methods used in time-series predictions, the GVF predictions were more accurate and could be learned incrementally in deployment.

7.1 Future Directions

The next steps for this project involve control: automating subproblems within water treatment. There are numerous such subproblems, for example, controlling the rate at which chemicals are added in pre-treatment. Backwashing is also promising because it is, by far, the most energy intensive part of the operation. We could control the duration of backwashing or how often to backwash. At the lowest level, we can adapt the parameters of the PID controllers that control the pumps during backwashing. Classical PID controls are not sensitive to the state of the plant; they are tuned when the plant is first commissioned and can become uncalibrated over time.

Algorithmically, we plan to investigate using our learned predictions for control, directly. Traditionally, one would define a reward function and use a reinforcement learning method such as Actor Critic to directly control aspects of the plant operation. These methods are notoriously brittle and difficult to tune. A more practical approach is to use the predictions to directly build a controller. Prior work has explored using learned predictions inside basic if-then-else control rules to control mobile robots (Modayil and Richard S Sutton 2014). The advantage of this approach is that the control-rules are easy to explain to human operators, but since control is triggered by predictions that are continually updated in deployment the resultant controller adapts to changing conditions. An extension of this idea is to use GVF predictions—like the ones we learned in this work—as input to a neural-network based RL agent, similarly to how it was done for autonomous driving (Graves et al. 2020; Jin et al. 2022). This work provides the foundations for these next steps in industrial control with RL.

References

Locke, John (1847). *An essay concerning human understanding.* Kay & Troutman. 1

Ockley, Simon et al. (1708). *The Improvement of Human Reason: Exhibited in the Life of Hai Ebn Yokdhan.* E. Powell. 1

Leibniz, Gottfried Wilhelm and Gottfried Wilhelm Freiherr von Leibniz (1996). *Leibniz: New essays on human understanding.* Cambridge University Press. 1

Gilbert, Daniel T and Timothy D Wilson (2007). "Prospection: Experiencing the future." In: *Science* 317.5843, pp. 1351–1354. 2

Pavlov, P Ivan (2010). "Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex." In: *Annals of neurosciences* 17.3, p. 136. 2

Abrams, Peter A (2000). "The evolution of predator-prey interactions: theory and evidence." In: *Annual Review of Ecology and Systematics* 31.1, pp. 79–105. 2

Ring, Mark Bishop et al. (1994). "Continual learning in reinforcement environments." In: 2

Holland, John H (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press. 3

Sutton, Richard S., Anna Koop, and David Silver (2007). "On the role of tracking in stationary environments." In: *International Conference on Machine Learning.* 3, 23, 37

Watkins, Christopher J. C. H. and Peter Dayan (1992). "Technical note: Q-learning." In: *Machine Learning* 8.3-4. 4

Schulman, John et al. (2017). "Proximal policy optimization algorithms." In: *CoRR* abs/1707.06347. 4

Haarnoja, Tuomas et al. (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." In: *International Conference on Machine Learning.* 4

Mirhoseini, Azalia et al. (2021). "A graph placement methodology for fast chip design." In: *Nature* 594.7862, pp. 207–212. 4

Fawzi, Alhussein et al. (2022). "Discovering faster matrix multiplication algorithms with reinforcement learning." In: *Nature* 610.7930, pp. 47–53. 4

Mandhane, Amol et al. (2022). "MuZero with self-competition for rate control in VP9 video compression." In: *CoRR* abs/2202.06626. 4

Bellemare, Marc G et al. (2020). "Autonomous navigation of stratospheric balloons using reinforcement learning." In: *Nature* 588.7836, pp. 77–82. 4, 11

Degrave, Jonas et al. (2022). "Magnetic control of tokamak plasmas through deep reinforcement learning." In: *Nature* 602.7897, pp. 414–419. 4, 11

Won, Dong-Ok, Klaus-Robert Müller, and Seong-Whan Lee (2020). "An adaptive deep reinforcement learning framework enables curling robots with human-like performance in real-world conditions." In: *Science Robotics* 5.46, p. 9764. 4

Luo, Jerry et al. (2022). "Controlling commercial cooling systems using reinforcement learning." In: *CoRR* abs/2211.07357. 5, 12, 21

Ernst, Damien, Pierre Geurts, and Louis Wehenkel (2005). "Tree-based batch mode reinforcement learning." In: *Journal of Machine Learning Research* 6, pp. 503–556. 5

Riedmiller, Martin A. (2005). "Neural fitted Q iteration – First experiences with a data efficient neural reinforcement learning method." In: *European Conference on Machine Learning.* 5

Lange, Sascha, Thomas Gabel, and Martin A. Riedmiller (2012). "Batch reinforcement learning." In: *Reinforcement Learning.* Vol. 12. Adaptation, Learning, and Optimization. Springer, pp. 45–73. 5

Levine, Sergey, Aviral Kumar, et al. (2020). "Offline reinforcement learning: Tutorial, review, and perspectives on open problems." In: *CoRR* abs/2005.01643. 5

Pietquin, Olivier et al. (2011). "Sample-efficient batch reinforcement learning for dialogue management optimization." In: *ACM Transactions on Speech and Language Processing* 7.3, 7:1–7:21. 5

Shortreed, Susan M. et al. (2011). "Informing sequential clinical decision-making through reinforcement learning: an empirical study." In: *Machine Learning* 84.1-2, pp. 109–136. 5

Swaminathan, Adith et al. (2017). "Off-policy evaluation for slate recommendation." In: *Advances in Neural Information Processing Systems (NeurIPS).* 5

Levine, Sergey, Peter Pastor, et al. (2018). "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection." In: *International Journal of Robotics Research* 37.4-5, pp. 421–436. 5

Copeland, Claudia and Nicole T. Carter (2017). *Energy-water nexus: The water sector's energy use.* CRS Report No. R43200. Washington, DC: USA. 5

Sutton, Richard S, Joseph Modayil, et al. (2011). "Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction." In: *International Conference on Autonomous Agents and Multiagent Systems*, pp. 761–768. 6, 10, 11, 30

Modayil, Joseph, Adam White, and Richard S Sutton (2014). "Multi-timescale nexting in a reinforcement learning robot." In: *Adaptive Behavior* 22.2, pp. 146–160. 6

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386. 8

Marvin, Minsky and A Papert Seymour (1969). "Perceptrons." In: *Cambridge, MA: MIT Press* 6, pp. 318–362. 8

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors." In: *nature* 323.6088, pp. 533–536. 8

Deng, Jia et al. (2009). "Imagenet: A large-scale hierarchical image database." In: *2009 IEEE conference on computer vision and pattern recognition.* Ieee, pp. 248–255. 9

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2017). "Imagenet classification with deep convolutional neural networks." In: *Communications of the ACM* 60.6, pp. 84–90. 9

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning." In: *nature* 521.7553, pp. 436–444. 9

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction.* MIT press. 10, 34, 37

Wolk, Melody et al. (2022). "Beyond CAGE: Investigating Generalization of Learned Autonomous Network Defense Policies." In: *arXiv preprint arXiv:2211.15557.* 11

Fuhrer, Benjamin et al. (2022). "Implementing Reinforcement Learning Datacenter Congestion Control in NVIDIA NICs." In: *arXiv preprint arXiv:2207.02295.* 11, 12

Agarwal, Pranav et al. (2022). "Automatic Evaluation of Excavator Operators using Learned Reward Functions." In: *arXiv preprint arXiv:2211.07941.* 11

Tessler, Chen et al. (2022). "Reinforcement learning for datacenter congestion control." In: *ACM SIGMETRICS Performance Evaluation Review* 49.2, pp. 43–46. 12

Liu, Puer (2022). "Improving Water Treatment Using Reinforcement Learning." In: 13

Zhang, Bopeng et al. (2020). "Backwash sequence optimization of a pilot-scale ultrafiltration membrane system using data-driven modeling for parameter forecasting." In: *Journal of Membrane Science* 612, p. 118464. 13

Zhang, Qing J et al. (2007). "Artificial neural network real-time process control system for small utilities." In: *Journal-American Water Works Association* 99.6, pp. 132–144. 13

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos (2022). "The M5 competition: Background, organization, and implementation." In: *International Journal of Forecasting* 38.4, pp. 1325–1336. 18

Tao, Ruo Yu, Adam White, and Marlos C. Machado (2022). "Agent-state construction with auxiliary inputs." In: *CoRR* abs/2211.07805. 23, 33

Crone, Sven F., Michele Hibon, and Konstantinos Nikolopoulos (2011). "Advances in forecasting with neural networks? Empirical evidence from the

NN3 competition on time series prediction." In: *International Journal of Forecasting* 27.3, pp. 635–660. 29

Hewamalage, Hansika, Christoph Bergmeir, and Kasun Bandara (2021). "Recurrent Neural Networks for Time Series Forecasting: Current status and future directions." In: *International Journal of Forecasting* 37.1, pp. 388–427. 29

Modayil, Joseph and Richard S Sutton (2014). "Prediction driven behavior: Learning predictions that drive fixed responses." In: *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence.* 30, 50

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory." In: *Neural Computation* 9.8, pp. 1735–1780. 33

Cho, Kyunghyun et al. (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation." In: *Conference on Empirical Methods in Natural Language Processing.* 33

Hausknecht, Matthew J. and Peter Stone (2015). "Deep recurrent Q-learning for partially observable MDPs." In: *AAAI Fall Symposia.* 33

Vinyals, Oriol et al. (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning." In: *Nature* 575.7782, pp. 350–354. 33

Mozer, Michael C. (1989). "A focused backpropagation algorithm for temporal pattern recognition." In: *Complex Systems* 3.4. 33

Rafiee, Banafsheh et al. (2023). "From eye-blinks to state construction: Diagnostic benchmarks for online representation learning." In: *Adaptive Behaviour* 31.1, pp. 3–19. 33

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A method for stochastic optimization." In: *International Conference on Learning Representations (ICLR).* 34

Tsitsiklis, J.N. and B. Van Roy (1997). "An Analysis of Temporal-Difference Learning with Function Approximation." In: *IEEE Transactions on Automatic Control* 42.5, pp. 674–690. 36

Dai, Bo, Niao He, et al. (2017). "Learning from Conditional Distributions via Dual Embeddings." In: *International Conference on Artificial Intelligence and Statistics.* 36

Dai, Bo, Albert Shaw, et al. (2018). "SBEED: Convergent Reinforcement Learning with Nonlinear Function Approximation." In: *International Conference on Machine Learning.* 36

Patterson, Andrew, Adam White, and Martha White (2022). "A Generalized Projected Bellman Error for Off-policy Value Estimation in Reinforcement Learning." In: *Journal of Machine Learning Research.* 36

Mnih, Volodymyr et al. (2015). "Human-Level Control through Deep Reinforcement Learning." In: *Nature* 518.7540, pp. 529–533. 36, 46

Graves, Daniel et al. (2020). "Learning predictive representations in autonomous driving to improve deep reinforcement learning." In: *CoRR* abs/2006.15110. arXiv: `2006.15110`. URL: `https://arxiv.org/abs/2006.15110`. 50

Jin, Jun et al. (2022). "Offline learning of counterfactual predictions for real-world robotic reinforcement learning." In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3616–3623. 50