Deep Reinforcement Learning for the Real Time Control of Stormwater Systems

Abhiram Mullapudi[a], Matthew J. Lewis[c], Cyndee L. Gruden[b], Branko Kerkez[a,*]

[a] *Department of Civil and Environmental Engineering, University of Michigan, Ann Arbor, MI, USA*

[b] *Department of Civil and Environmental Engineering, University of Toledo, Toledo, OH, USA*

[c] *Michigan Aerospace, Ann Arbor, MI, USA*

Abstract

A new generation of smart stormwater systems promises to reduce the need for new construction by enhancing the performance of the existing infrastructure through real time control. Smart stormwater systems dynamically adapt their response to individual storms by controlling distributed assets, such as valves, gates and pumps. This paper introduces a real-time control approach based on Reinforcement Learning (RL), which has emerged as a state-of-the-art methodology for autonomous control in the artificial intelligence community. Using a deep neural network, an RL based controller learns a control strategy by interacting with the system it controls effectively trying various control strategies until converging on those that achieve a desired objective. This paper formulates and implements an RL algorithm for the real-time control of urban stormwater systems. This algorithm trains an RL agent to control valves in a distributed stormwater system across thousands of simulated storm scenarios, seeking to

*Corresponding author

achieve water level and flow setpoints in the system. The algorithm is first evaluated for the control of an individual stormwater basin, after which it is adapted to the control of multiple basins in a larger watershed ($4km^2$). The results indicate that RL can very effectively control individual sites. Performance is highly sensitive to the reward formulation of the RL agent. Generally, more explicit guidance - encoded as a more complex reward formulation – leads to better control performance, and more rapid and stable convergence of learning process. While the control of multiple distributed sites also shows promise in reducing flooding and reducing peak flows, the complexity of controlling larger systems comes with a number of caveats. The RL controller's performance is very sensitive to the formulation of the deep neural network and requires a significant amount of computational resource to achieve a reasonable performance enhancement. Overall, the controlled system significantly outperforms the uncontrolled system, especially across storms of high intensity and duration. A frank discussion is provided, which should allow the benefits and drawbacks of RL to be considered when implementing it for the real-time control of stormwater systems. An open source implementation of the full simulation environment and control algorithms is also provided.
Keywords: real-time control, reinforcement learning, smart stormwater systems

---

Highlights

- A Deep Reinforcement Learning algorithm for the real-time control urban watersheds
- The control algorithm uses deep neural network and extensive simulations to learn a control strategy

- Real-time control of stormwater systems reduces flooding and stream erosion
- Performance of the controller is highly sensitive to the reward function and neural network formulation
- An open source implementation of the entire toolchain is provided

1. Introduction

Urban stormwater and sewer systems are being stressed beyond their intended design. The resulting symptoms manifest themselves in frequent flash floods [1] and poor receiving water quality [2]. Presently, the primary solution to these challenges is the construction of new infrastructure, such as bigger pipes, basins, wetlands, and other distributed storage assets. Redesigning and rebuilding the existing stormwater infrastructure to keep in pace with the evolving inputs is cost prohibitive for most communities [3]. Furthermore, infrastructure is often upgraded on a site-by-site basis and rarely optimized for system-scale performance. Present approaches rely heavily on the assumption that that these individual upgrades will add up to cumulative benefits, while the contrary has actually been illustrated by studies evaluating system-level outcomes [4]. The changing and highly variable nature of weather and urban environments demands stormwater solutions that can more rapidly adapt to changing community needs.

Instead of relying on new construction, a new generation of smart stormwater systems promises to dynamically re-purpose existing stormwater systems. These systems will use streaming sensor data to infer real-time state of a watershed and respond via real-time control of distributed control assets, such as valves, gates and pumps [3]. By achieving system-level coordination between many distributed control points, the size of infrastructure needed to reduce flooding and improve water quality will become smaller. This presents a non-trivial control challenge, however, as any automated decisions must be carried with regard to public safety and must account for the physical complexity inherent to urban watersheds [5, 6].

In this paper, we investigate *Deep Reinforcement Learning* for the real-time control of storm-water systems. This approach builds on very recent advances in the artificial intelligence community, which have primarily focused on the control of complex autonomous systems, such as robots and autonomous vehicles [7, 8]. In this novel formulation, our algorithm will *learn* the best real-time control strategy for a distributed stormwater system by efficiently quantifying the space of all possible control actions. In simple terms, the algorithm attempts various control actions until discovering those that have the desired outcomes. While such an approach has shown promise across many other domains, it is presently unclear how it will perform and scale when used for the real time control of water systems, specifically urban drainage networks.

The fundamental contribution of this paper is a formulation of a control algorithm for urban drainage systems based on Reinforcement Learning. Given risk to property and public safety, it is imprudent to hand over the control of a real-world watershed to a computer that learns by mistake. As such, a secondary contribution is the evaluation of the Reinforcement Learning algorithm across a series of simulations, which span various drainage system complexities and storms. The results will illustrate the benefits, limitations, and requirement of Reinforcement Learning when applied to urban stormwater systems. To our knowledge, this is the first formulation of Deep Reinforcement Learning for the control of stormwater systems. The results of this study stand to support a foundation for future studies on the role of Artificial Intelligence in the control of urban water systems.

1.1. ***Real-time control of urban drainage systems***

Since the European Unions Directive on water policy [9], there has been a significant push towards the adoption of real time control for improving wastewater and sewer systems [6, 10]. During the past decade, Model Predictive Control (MPC) has emerged as a state-of-the-art methodology for controlling urban drainage and sewer networks. MPC has been used to regulate dissolved oxygen in the flows to aquatic bodies [11], control inflows to wastewater treatment plants [12], maintain water level set points in sewer networks [13], and enhance the system-level performance and coordination of sewer network assets [10, 14]. These and many other [15] applications have illustrated the benefits of control, the biggest of which is the ability to cost-effectively re-purpose existing assets in real-time without the need to build more passive infrastructure.

The performance of MPC depends on the extent to which the underlying process can be approximated using a linear model [16]. A benefit of this linearity assumption is the ability to analytically evaluate the stability, robustness and convergence properties of the controller [17], which is valuable when providing safety and performance guarantees. Network dynamics of storm and sewer systems and transformations of the pollutants in runoff are known to be heavily non-linear, however. This demands a number of approximations and a high level of expertise when applying MPC. Furthermore, real-world urban watersheds are prone to experiencing pipes blockages, sensor breakdowns, valve failures, or other adverse conditions. Adapting and re-formulating linear control models to such non-linear conditions is difficult, but is being addressed by promising research [18]. The constraints of linear approximations and the need for adaptive control algorithms open the door to exploring other control methodologies, such as the one presented in this paper.

75 2. Reinforcement Learning

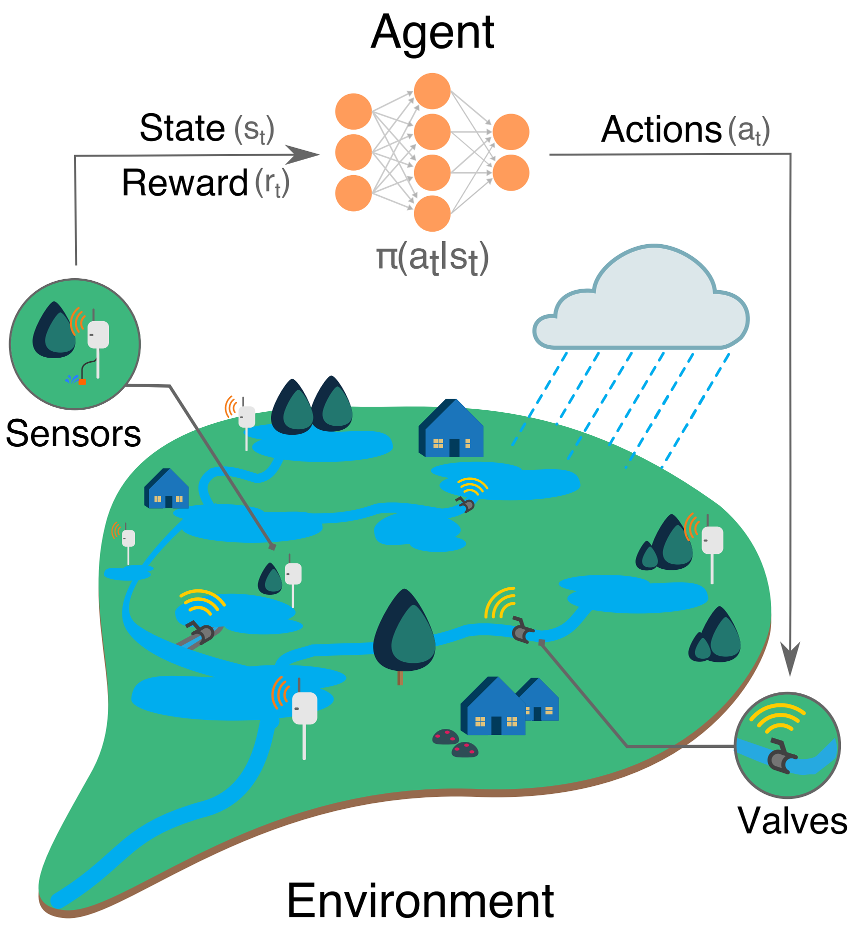

Figure 1: During a storm event, a reinforcement learning controller observes the state of the stormwater network and coordinates the responses of the distributed control assets in real time to achieve watershed-scale benefits.

Across the artificial intelligence and behavioral research communities, Reinforcement Learning (RL) has emerged as a state-of-the-art methodology for autonomous control and planning systems. Unlike in classical feedback control, where the controller carries out a pre-tuned and analytical control action, an RL controller (i.e. an RL agent) learns a control strategy by interacting with the system effectively trying various control strategies until learning those that work well. Rather than just learning one particular control strategy, an RL agent continuously attempts to improve its control strategy by assimilating new information and evaluating new control strategies [19]. RL can be used in a model free context since the systems dynamics are implicitly learned by evaluating various control actions. Leveraging the recent advancements in Deep Neural Networks and the computational power afforded by the high performance clusters (HPC), RL agents have been able plan complex tasks, such as observing pixels to play video games at a human level [7], defeating world champions in the game of GO [20], achieving superhuman performance in chess [21], controlling high speed robots [22], and navigating autonomous vehicles [23]. Despite the wide adoption of Deep RL in various disciplines of engineering, its adoption in civil engineering disciplines has been limited [24, 25, 26]. Deep RL control has yet to be applied to the real time control of urban drainage systems.

Deep RL agents approximate underlying system dynamics implicitly, hence not requiring a simplified or linearized control model [19]. A Deep RL agent instantaneously identifies a control action by observing the network dynamic, thus reducing delay in the decision process [7, 21]. The explorative nature of the Deep RL agents also enables the methodology to adapt its control strategy to changing conditions of the system [19]. Hence, Reinforcement Learning shows promise as a potential alternative or supplement to existing control methods for water systems. To that end, the goal of this paper is to formulate and evaluate of Reinforcement Learning for the real-time control of urban drainage systems. The specific contributions of the paper are:

1. The formulation and implementation of a reinforcement learning algorithm for the control of urban stormwater systems.
2. An evaluation of the control algorithm under a range of storm inputs and network complexities (single stormwater basins and an entire network), as well as an equivalence analysis that compares the approach to passive infrastructure solutions.
3. A fully open-sourced implementation of the control algorithm to promote transparency and permit for the direct application of the methods to other systems, shared on open-storm.org.

Symbol	**Definition**
s_t	state observed by agent at time t
a_t	action taken by agent at time t
r_t	reward received by the agent at time t
π	policy of the agent
$v(s_t)$	value estimate for a given state s_t
$q(s_t, a_t)$	action value estimate for s_t, a_t
q	action value estimator
q^*	target estimator
ϵ	rate of exploration
α	step size
γ	discount factor
h_t	basin's water level at time t
f_t	channel's flow at time t
H	desired water height in basin
H_{max}	height threshold for flooding
F	flow threshold for erosion

Table 1: Summary of notation used in paper.

3. Methods

3.1. Reinforcement learning for stormwater systems

When formulated as a Reinforcement Learning problem, the control of stormwater systems can be fully described by an agent and environment (Figure 1). The environment represents an urban stormwater system and the agent represents the entity controlling the system. At any given time t, the agent takes a control action a_t (e.g. opening a valve or turning on a pump) by observing any number of states s_t (e.g. water levels or flows) in the environment. Based on the outcomes of its action, the agent receives a reward r_t from the environment. The reward is formulated to reflect the specific control objectives. For example, an agent could receive positive reward for a preventing flooding or a negative reward for causing flooding. By quantifying these rewards in response to various actions over time, the agent learns the control strategy that will achieve its desired objective [19]. The agents control actions are governed by its policy π, which governs the action that agent will take in any given state. Formally, the policy is a mapping from a given state to the agents actions:

$$\pi : s_t(\mathbb{R}^n) \rightarrow a_t(\mathbb{R}) \tag{1}$$

The primary objective of the RL control problem is to learn a policy that maximizes the total reward earned by the agent.

While the reward r_t at the end of each control action teaches the agent the immediate desirability of taking a particular action for a given state, it does not necessarily covey any information about the long-term desirability of that action. For many water systems, maximizing short-term rewards will not necessarily lead to the best long-term outcomes. An agent controlling a watershed or stormwater system should have the ability to take individual actions in the context of the entire storm duration. For example, holding water in a detention basin may initially provide high rewards since it reduces downstream flooding, but may lead to upstream flooding if a storm becomes too large. Instead of choosing an action that maximizes the reward r_t at time t, the agent seeks to maximize the expected long-term reward described by state-value v or action-value q.

$$v(s_t) = \mathbb{E}\Big[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \Big| s_t\Big] = \mathbb{E}\Big[\sum_{k=0}^{\infty}\big[\gamma^k r_{t+k+1}\big| s_t\big]\Big] \tag{2}$$

$$q(s_t, a_t) = \mathbb{E}\Big[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \Big| s_t, a_t\Big] = \mathbb{E}\Big[\sum_{k=0}^{\infty}\big[\gamma^k r_{t+k+1}\big| s_t, a_t\big]\Big] \tag{3}$$

The state-value provides an estimate for an instantaneous action, as well as potential future rewards that may arise after state s_t, discounted with a factor $\gamma (0 \leq \gamma \leq 1)$. The action-value provides a similar estimate conditioned, also however, on taking an action a_t in state s_t. The discount factor γ governs the temporal context of the reward. For example, a γ of 0 forces the agent to maximize the instantaneous reward, while a γ of 1 forces it to equally weigh all the rewards it might receive for present and future outcomes. γ is specific to the system being controlled and can vary based on the control objective[19].

An RL agent can learn to control a system either by learning the policy directly[27]. Alternatively, the agent can learn the state-value or action-value estimates and follow a policy that guides it towards the states with high estimates [19]. Several methods based on dynamic programming [28, 29] and Monte Carlo sampling [19] have been developed to learn the functions that estimate the policy and value functions. While these algorithms were computationally efficient and provided guarantees on the convergence, their application was limited to simple systems whose state action space can be approximated using lookup tables and linear functions [19, 30].

Given the scale and the complexity of urban watersheds and stormwater networks, a simple lookup table or a linear function cannot effectively approximate the policy or value functions for each state the agent may encounter while controlling the system. As a simple example, considering just ten valves in a stormwater system and assuming that each valve has ten possible control actions (closed, 10% open, 20% open, ...) this gives 10^{10} (10 billion) possible actions that can be taken at any given state, making it computationally impossible to build an explicit lookup table for all possible states. This, however, is where very recent advances in Deep Learning, become important. It has been shown that for systems with a large state-action spaces, such as stormwater systems, these functions can be approximated by a deep neural network [19, 7].

Deep neural networks are a class of feed forward artificial neural networks with large layers of inter connected neurons. This deeply layered structure permits the network to approximate highly complex functions [31], such as those needed for RL-based control. Each layer in the network generates its output by processing the weighted outputs from the previous layer. This means that each layer's output is more complex and abstract than its previous layer. Given the emergence of cheap and powerful computational hardware over the past decade – in particular graphical processing units (GPUs) and high performance clusters (HPCs) – deep neural networks and their variants have emerged as the state of the art in the approximation of complex functions in large state spaces [32]. This makes them a good candidate for approximating the complex dynamics across stormwater systems. For purposes of this paper, a brief mathematical summary of deep neural networks in provided in appendix A.

3.2. ***Deep Q Learning***

Deep reinforcement learning agents (deep RL) use deep neural networks as approximators for value or policy functions to control complex environments. In their relatively recent and seminal Deep Q Network(DQN) paper Mnih et al. (2015) [7] demonstrated the first such algorithm, which used deep neural networks to train an deep RL agent to play *Atari* video games at a human level. This algorithm identifies the optimal control strategy for achieving an objective by learning a function that estimates the action values or q-values. This function (i.e. q-function), maps a given state-action pair (s_t, a_t) pair to the action value estimate.

At the beginning of the control problem, the agent does not know its environment. This is reflected by assigning random q-values for all state-action pairs. Over time, as the agent takes actions, new information obtained from the environment is used to update these initial random estimates. After each action the reward obtained from the environment is used to incorporate the new knowledge:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha \Big[r_{t+1} + \gamma \max_a q(s_{t+1}, a) - q(s_t, a_t) \Big] \tag{4}$$

The more actions an agent takes at any given state, the closer it gets to converging to the true action value function [19]. The α (step-size) parameter governs how much weight is placed on the new knowledge [19].

An agent will choose an action that maximizes its long-term reward. This process is known as exploitation since it greedily seeks to maximize a known long-term reward. This may not always be the best choice, however, since taking another action may lead the agent to discover a potentially better action, which it has not yet tried. As such, the agent also needs to explore its environment. This is accomplished by taking a random action periodically, just in case this action leads to better outcomes. In such a formulation, the *exploration vs. exploitation* is addressed via a ϵ-greedy policy, where the agent explores for ϵ percent of time and chooses an action associated with the highest action value for the rest. This gives the final policy for the RL agent:

$$\pi(s_t) = \begin{cases} \text{random } a, & \epsilon \\ \arg\max_a q(s_t, a), & else \end{cases} \tag{5}$$

ϵ is often set at a high value (e.g. 50%) at the start of the learning process and gradually reduced to lower value (e.g. 1%) as the agent identifies a viable control strategy.

While there have been prior attempts to approximate the action value function using deep neural networks, they were met with minimal success since the learning is highly unstable [7]. Mnih et al. (2015) [7] addressed this by introducing a replay buffer and an additional target neural network. The replay buffer acts as the RL agents memory, which records only its most recent experience (e.g. the past 10^3 states transitions and rewards). During the training the RL agent randomly samples data from the replay buffer, computes the neural networks loss and updates its weights using stochastic gradient descent:

$$Loss = ||\big(r_t + \gamma \max_{a'} q^*(s_{t+1}, a')\big) - q(s_t, a_t)||^2 \tag{6}$$

This random sampling enables the training data to be uncorrelated and has been found to improve the training process. The target neural network q^* has the same network architecture as the main network q, but acts as a moving target to help stabilize the training process by reducing variance [7]. Unlike the neural network approximating q, whose weights are constantly updated using gradient decent, q^* weights are updated sporadically (e.g. every 10^4 timesteps). For more background information, Mnih et al. (2015)[7] and Lillicrap et al. (2016)[8] provide an in-depth discussion on the importance of replay memory and target neural networks in training deep RL agents.

3.3. ***Evaluation***

Here, we investigate the real-time control of urban stormwater infrastructure using Deep RL. To begin, we formulate and evaluate reward functions for the control of an individual stormwater basin. We then extend these lessons to the control of a larger, interconnected stormwater network. Given the relatively nascent nature of Deep RL, the need to account for public safety, and the desire to evaluate multiple control scenarios, a real-world evaluation is outside of the scope of this paper. As such, our analysis will be carried out in simulation as a stepping-stone toward real-world deployment in the future. To promote transparency and broader adoption, the entire source code, examples, and implementation details of our implementation are shared freely as an open source package[1].

[1]https://github.com/kLabUM/rl-storm-control

3.4. *Study Area*

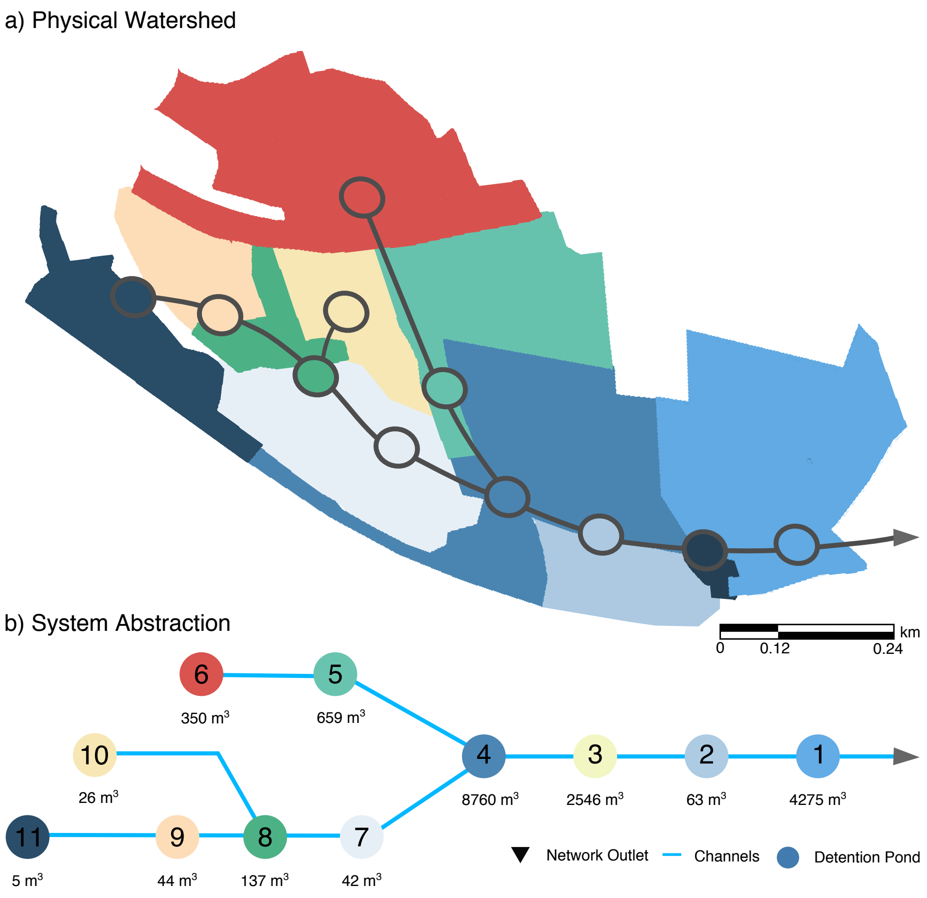

Figure 2: Stormwater system being controlled in this paper. The urban watershed includes a number of sub-catchments which drain to 11 stormwater basins. The first control scenario applies RL to the control of a single basin, while the second scenario evaluates control of multiple basins. Average volumes experienced by the ponds

Motivated by a real-world system, we apply RL control to a stormwater system inspired by an urban watershed Ann Arbor, Michigan, USA (2). Our choice to use this watershed is motivated by the fact that it has been retrofitted 250 by our group with wireless sensors and control valves already [33] and will in the future serve as a real-world testbed for the ideas proposed in this paper. This headwater catchment features 11 interconnected stormwater basins that handle the runoff generated across $4km^2$ of predominantly urbanized and impervious sub-catchment areas. A Stormwater Management Model (SWMM) of the watershed has been developed and calibrated in prior, peer-reviewed studies [15]. It is assumed that each controlled basin in the system is equipped with a $1m^2$ square gate valve. The valves can be partially opened or closed during the simulation, which represents the action taken by an RL agent. The states of the control problem are given by the water levels and outflows at each controlled location. Given the small size of the study area, as well as the need to constrain this initial study, uniform rainfall across the study area is assumed. Groundwater baseflow is assumed to be negligible, which has also been confirmed in prior studies [15].

3.5. ***Analysis***

Prior Deep RL studies have revealed that performance is dependent on the formulation of reward function, quality of neural networks approximating action value function, as well as the size of state space [19, 34]. This creates a number of "knobs", whose sensitivity must be evaluated before any conclusion can be reached regarding the ability to apply Deep RL to control real stormwater systems. As such, in this paper we formulate a series of experiments across two scenarios to characterize Deep RLs ability to control stormwater systems. In the first scenario, we control a single valve at the outlet of the watershed, comparing in particular performance under various reward function formulations. Given that Deep RL has not been used to control water systems, this will constrain the size of the state space to establish a baseline assessment of the methodology. In the second scenario, we scale these findings to simultaneously control multiple valves across the broader watershed and to analyze sensitivity to function approximation (neural networks). Finally, the system-scale scenario is subjected to storm inputs of varying intensities and durations to provide broader comparison of the benefits of the controlled system in relation to the uncontrolled system.

*3.6. **Scenario 1: Control of a single basin***

In this scenario, we train a deep RL agent to control the most downstream detention basin in the network (basin 1 in Figure 2). This basin was chosen because it experiences the total runoff generated in the watershed, and because its actions have direct impact on downstream water bodies. At any given point in time, the RL agent is permitted to set the basins valve to a position between fully closed or open, in 1% increments (i.e. 0%, 1%, 2%, ..., 100% open) based on the water height in the basin. All other upstream basins remain uncontrolled.

The overall control objective is to keep the water height (state h_t) in the basin below a flooding threshold H_{max} and the outflows from the basin (state f_t) below a desired downstream flooding or stream erosion threshold F:

$$h_t \leq H_{max} \tag{7}$$

$$f_t \leq F \tag{8}$$

Three reward functions are formulated to reach this objective, varying in complexity from most simple to complex.

In the first reward function the RL agent receives a positive reward for maintaining the basins outflow below the specified threshold, a negative reward for exceeding the threshold, as well as a larger but less likely negative reward if the basin overflows:

$$r_1(s_t) = \begin{cases} +1, & f_t \leq F \\ -1, & f_t > F \\ -10, & h_t > H_{max} \end{cases} \tag{9}$$

The reward function is represented visually in the first row of Figure 3. This reward function formulation is inspired from the classic inverted pendulum problem [28] where the agent receives +1 for success and -1 for failure.

The second reward function is formulated to exhibit a more complex and gradual reward structure. In lieu of a jagged or discontinuous “plus minus” reward structure, the agent is rewarded for reaching flows that are close to the desired flow threshold. It has been shown that more smooth and continuous rewards such as this may help the agent converge onto a solution faster [19, 35]. Visually, the reward function looks like a parabola (Figure 3), where the maximum reward is achieved when the flow threshold is met exactly:

$$r_2(s_t) = c_1(f_t - c_2)(f_t - c_3) \tag{10}$$

c_1,c_2,and,c_3 are constants representing the scaling and inflection points of the parabola. Here we choose c_1=400 e, c_2=0.05, and c_3=0.15 to maintain the general scale of the first reward function. Note that this formulation does not explicitly include the local constraint on the basins water level since the agent gets implicitly penalized by receiving a negative reward for low outflows.

The third reward function seeks to provide the most explicit guidance to the

RL agent by embedding the most relative amount of information (third column, Figure 3). In this heuristic formulation, the agent receives the highest reward for keeping the basin empty (water levels and flows equal to zero). Intuitively, this reward formulation seeks to drain all of the water from the basin as fast as possible without exceeding the flow and height thresholds. If water level in the pond rises, the agent gets penalized, thus forcing it to release water. If flows remain below the flow threshold F, the agent is penalized linearly proportional to the water level in the basin, with a more severe factor applied if the height of the basin exceeds the height threshold H. If the outflow exceeds the flow threshold F an even more severe penalty is incurred:

$$r_3(s_t) = \begin{cases} c_1 - c_2 h_t, & h_t < H f_t \leq F \\ c_1 - c_3 h_t, & h_t \geq H f_t \leq F \\ -c_4 f_t - c_2 h_t + c_5, & h_t < H f_t > F \\ -c_4 f_t - c_3 h_t + c_5, & h_t \geq H f_t > F \end{cases} \tag{11}$$

The penalty rates are governed by a set of five parameters c=$\{c_1, c_2, c_3, c_4, c_5\}$, which were paramterized $\{2.0, 0.25, 1.5, 10, 3\}$ to match the scales of the other two reward functions.

*3.7. **Scenario 2: Controlling multiple basins***

This scenario evaluates the ability of an agent to control multiple distributed stormwater basins. Specifically, basins 1, 3, and 4 (Figure 2) are selected for control because they experience the largest average volume during a storm event. It is assumed that at any time step the agent has knowledge of the water levels and valve positions for each of these basins, as well as the basin between them (basin 2 in Figure 2), thus quadrupling the number of observed states compared to the control of a single basin. The action space must also be reduced to make the problem computationally tractable. For the control of the single basin there are 101 possible actions at any given time step (valve opening with 1% granularity). For 3 controlled basins this increases to 101^3 possible control actions at any given time step. This is not only intractable given our own computational resources, but is well beyond the size of any action space covered in other RL literature. Here, to reduce the action space the agent is allowed to only throttle the valves. Specifically, at any time step the agent can only open or close the valve in 5% increments or leave its position unchanged. This results in only three possible actions for each site and thus 27 (or 3^3) possible actions for the entire network.

The agent receives an individual reward for controlling each basin. These rewards are weighted equally and added together to provide a total reward for controlling the larger system. The reward for controlling each basin is given by:

$$r_4(s_t) = \begin{cases} -c_1 h_t + c_4, & h_t \leq H, f_t \leq F \\ -c_2 h_t^2 + c_3 + c_4, & h_t > H, f_t \leq F \\ -c_1 h_t + (F - f_t)c_5, & h_t \leq H, f_t > F \\ -c_2 h_t^2 + c_3 + (F - f_t)c_5, & h_t > H, f_t > F \end{cases} \tag{12}$$

where reward parameters c=$\{c_1, c_2, c_3, c_4, c_5\}$ are chosen as $\{0.5, 1, 3, 1, 10\}$ to retain the relative scale of the single-basin reward formulations. This reward seeks to accomplish practically identical objectives of the third reward function used in the single-basin control scenario. The difference is the quadratic penalty term that is applied to the height constraint. This modification is made to provide the agent with a more explicit guidance in response to the relatively larger state space compared to the single-basin control scenario. In the rare instance that flooding should occur at one of the basins, agent also receives an additional penalty of -10.

3.8. ***Simulation, Implementation, and Evaluation***

Beyond the formulation of the reward function, the use of RL for the control of stormwater systems faces a number of non-trivial implementational challenges. The first relates to the hydrologic and hydraulic simulation framework, which needs to support the integration of a simulation engine that is compatible with modern RL toolchains. The second challenge relates to the implementation of the actual RL toolchain, which must include the deep neural network training algorithms.

Most popular stormwater modeling packages, such as the Stormwater Management Model (SWMM) [36] and MIKE Urban [37] are designed for event based or long-term simulation. Namely, the model is initialized, inputs are selected, and the model run continues until the rainfall terminates or simulation times out. While these packages support some rudimentary controls, the control logic is pre-configured and limited to simple site-scale action, such as opening a valve when level exceed a certain value. The ability to support system-level control logic is limited, let alone the ability to interface with external control algorithms, such as the one proposed in this paper. To that end, we implement a step-wise co-simulation approach that was described in one of our prior studies [5].

Our co-simulation framework separates the hydraulic solver from the control logic by halting the hydraulic model at every time step. The states from the model (water levels, flows, etc.) are then transferred to the external control algorithm, which makes recommendation on which actions to take (valves settings, pump speeds, etc.). Here, we adopt a python based SWMM package for simulating the stormwater network [38]. This allows the entire toolchain to be implemented using a high-level programming environment, without requiring any major modifications to hydraulic solvers that are often implemented in low-level programming languages and difficult to fuse with modern libraries and open source packages. While other or more complex stormwater or hydrologic models could be substituted, model choice is not necessarily the main contribution of this paper. Rather, we content that SWMM adequately captures runoff and flow dynamics for the purposes of this paper. SWMM models the flow of water in the network using an implicit dynamic wave equation solver[36]. This allows it to effectively model the nuanced conditions (e.g. back channel flows, flooding) that might develop in the network though real time control. Furthermore, the authors have access to calibrated version of the model for this particular study area, which has been document in a prior study [39, 15].

One major task is the implementation of the deep neural network that is used to approximate the RL agents action value function. Deep neural networks are computationally expensive to train [40]. Efficient implementation address this by leveraging a computers graphical processing unit (GPU) to carry out this training, which is a non-trivial task. To that end, a number of open source and community libraries have emerged, the most popular of which is *TensorFlow* [41]. This state-of-the-art library has been used in some of the most well-cited RL papers and benchmark problems, which is the reason we choose to adopt it for this study. *TensorFlow* is a python library and can be seamlessly interfaced with our python-based stormwater model implementation.

Four agents are trained and evaluated across the two scenarios: three for the control of individual basins, and one agent for the multi-basin control. A deep neural network is designed and implemented to learn the value function of each agent. The network contains 2 layers with 50 neurons per layer. This network is set up with a *ReLu* activation function [42] in the internal layers and a linear activation function in the final layer. The full parameters used in the study, including those for gradient descent and the DQN, are provided in appendix A of this paper. A Root Mean Square Propagation (RMSprop) [42] form of stochastic gradient descent is used for updating the neural network as this variant of gradient descent has been observed to improve convergence.

One storm event is used to train these agents. The SWMM model is forced with a 25-year storm event of 6 *hour* duration and 63.5 *mm* intensity (Figure 3). This event generates a total runoff of $3670.639m^3$ with a peak flow of $0.35m^3/s$ at the outlet of watershed. The agents are provided with an operational water level goal H of $2m$, flooding level H_{max} of 3.5m and outflow exceedance threshold of F of $0.10m^3s^{-1}$. These setpoints are chosen to reflect realistic flooding and stream erosion goals in the study watershed. Agents are trained on a Tesla K20 GPUs on University of Michigans advanced research computing's high performance cluster.

A fifth agent is also trained, focusing specifically on the multi-basin control scenario. The agent uses the same neural network architecture of the other multi-basin RL control agent, trained this time, however, using *batch normalization*[43]. Batch normalization is the process of normalizing the signals between the internal layers of the neural network to minimize the internal covariance shift and has been observed to improve the performance of the deep RL agents [8]. Ioffe et al. (2015) [43] provides a detailed discussion on batch normalization.

The performance of each agent is evaluated by comparing the RL-controlled hydrographs and water levels to those that are specified in the reward functions. For the agents controlling the individual basins this is used to determine the importance of the reward formulation on performance, reward convergence, and training period duration. For the multi-basin control scenario, the same approach is used to quantify overall performance, comparing this time the agent that uses the batch normalized neural network to the agent that uses the non-normalized network.

To evaluate the ability of an RL-agent to control storms that it is not trained on, a final analysis is carried out. Since the agent controlling multiple basins presents the most complex of the scenarios, it is first trained on one of storms and evaluated on a spectrum of storm events with varying return periods (1 to 100 years) and durations (5 min to 24 hours). These storm events are generated based on the SCS type II curve and historical rainfall intensities for the study region [44]. The performance of the agent across these 70 storms is compared to the uncontrolled system to evaluate the boarder benefits of real-time control. To allow for a comparison between the controlled and uncontrolled system, a non-negative performance metric is introduced to capture the magnitude and time that the system deviates from desired water level and flows thresholds. Specifically, across a duration T the final performance P adds together the deviation of all N controlled sites from their desired water level (P_h) and flow thresholds (P_f), where:

$$P_h(h) = \begin{cases} h - H, & h > H \\ h - H + 100h, & h > H_{max} \\ 0, & otherwise \end{cases} \tag{13}$$

$$P_f(f) = \begin{cases} 10(f - F), & f > F \\ 0, & otherwise \end{cases} \tag{14}$$

$$P = \sum_{n=1}^{N} \sum_{i=0}^{T} P_h(h_i^n) + P_f(f_i^n) \tag{15}$$

A relatively lower performance value is more desirable, since it implies that the system is not flooding, nor exceeding desired flow thresholds.

455 ## 4. Results

4.1. *Scenario 1: Control of single basin*

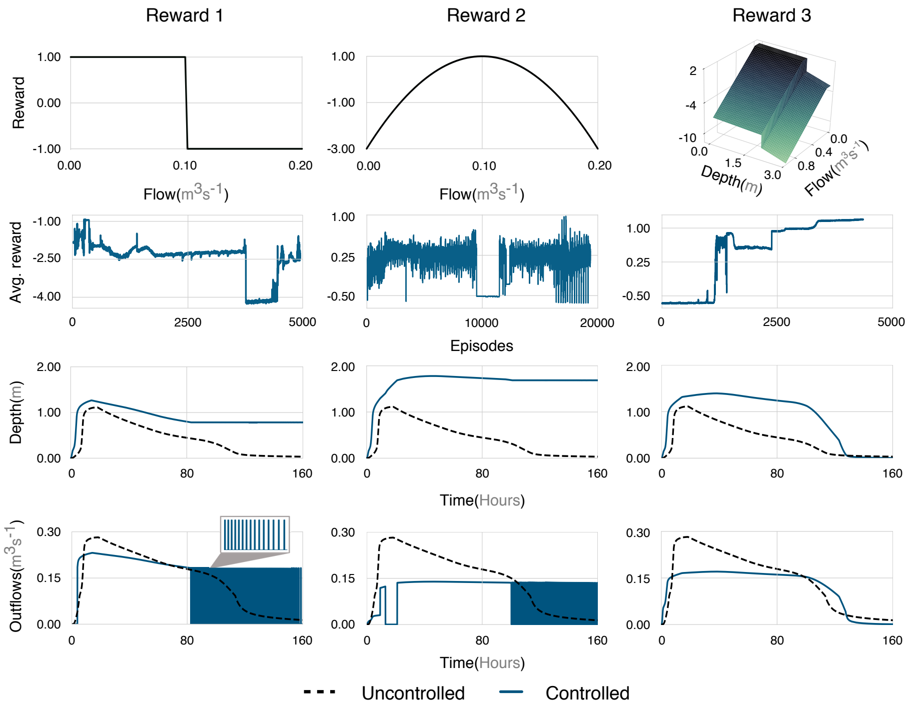

Figure 3: RL control of a single basin, trained using three reward formations (grouped by column). The first row plots each reward function used during training. The second row plots the average reward received during training. The third and fourth rows compare the uncontrolled flows and water levels, for the episode that resulted in the highest reward. Generally, more complex reward formulations lead to relatively better control performance and improved convergence during raining. Agents trained using relatively simple reward also exhibit a rapidly-changing and unstable control behavior, shown as a close up in the bottom left plot.

The ability of an RL agent to control a stormwater basin is highly sensitive to the reward function formulation. Generally, a more complex reward function – one that embeds more information and explicit guidance – performs better, as illustrated in Figure 3. Each column of the figure corresponds with an individual RL agent, each of which is trained using a different reward function (r_1, r_2, r_3). 460

The reward functions are plotted in the first row, while the reward received during training is plotted in the second row. The training period is quantified in terms of episodes, each of which corresponds to one full SWMM simulation across an entire storm. The third and fourth rows in the figure compare the uncontrolled flows and water levels, respectively, for the episode that resulted in the highest reward.

The RL agent that uses the simplest reward function has the relatively worst performance (Figure 3, first column). Even after 5000 training episodes (a week of real-world simulation time), the mean reward does not converge to a stable value. Playing back the episode that resulted in the highest reward (Figure 3, rows 3-4, column 1), reveals that the RL agent does retain more water than would have been held in the uncontrolled basin. While this lowers the peak flows relative to the uncontrolled basin, the RL agent is generally not able to keep flows below the desired threshold. More importantly, the RL agent's control actions begin oscillating and become unstable toward the middle of the simulation. In this episode, the agent keeps the water level in the basin relatively constant by opening the valve very briefly to release just a small amount of water. This chattering behavior (shown as a close up in the figure) results in an unstable outflow pattern that oscillates in a step-wise fashion between $0m^3/s$and $0.18m^3/s$. For various practical reasons, such rapid control actions are not desirable. Since the RL agent never once receives a positive reward, it may have converged onto an undesirable local minimum during the training. Providing more time training time does not appear to resolve this issue, which may also suggest that a stable solution cannot be derived using this particular reward formulation.

Increasing the complexity of the reward formulation improves the control performance of the RL agent (Figure 3, second column). When the second and more continuous reward function is used by the agent, the highest reward episode reveals that the RL agent is relatively more effective at maintaining flows at a constant value. Unlike the RL agent using the simple step-wise reward function, the RL agent using the parabolic reward function has more opportunities to receive smaller, more gradual rewards. During most of the episode, this increased flexibility allows the second RL agent to receive positive rewards and keep outflow below a flow threshold of $0.14m^3/s$. While relatively improved, the RL agent using the parabolic reward also does not converge to a stable reward value, however. Toward the end of the episode, this RL agent also carries out irregular and sudden control actions by opening and closing the valve in short bursts. In this case, the RL agent is oscillating between a maximum (valve open) and minimum (valve closed) reward rather than taking advantage of variable rewards in other configurations. This suggests that the agent has either not yet learned a better strategy or, again, that a stable solution cannot be converged upon using this particular reward formulation.

The RL agent using the third and most complex reward function exhibits the relatively best control performance. This agent regulates flow and water levels in a relatively gradual and smooth manner. Unlike in the case of the other two RL agents, after 3500 training episodes the third agent does converge to a steady reward. Evaluating the episode resulting in the highest reward (Figure 3, rows 3-4, column 3), the desired flat outflow hydrograph is achieved. No unstable or oscillatory control actions are evident, as in the case of the other two reward functions. The agent is able to maintain flows below a constant threshold of $0.15m^3/s$. While this is not the exact threshold that was specified ($0.1m^3/s$), it is close considering that achievable threshold is dependent on water levels the ability to only open the valve in 1% increments. As such, matching the exact threshold may not be physically realizable in any given situation. Interestingly, this agent does not change its valve configuration at all. Rather, it keeps its valve 54% open the entire time of the simulation, which allows it to meet a mostly constant outflow given the specific inflows. Overall, the general shape of the outflows is improved compared to the uncontrolled scenario. Furthermore, an added benefit of real-time control is that the overall volume of water leaving the basin is also reduced by 50% due to infiltration.

This scenario, which focuses on the control of a single site, emphasizes the importance of the reward function formulation in RL control of stormwater systems. The complexity of the reward formulation plays an important role in allowing the RL agent to learn a control policy to meet the desired hydrologic outcomes. The importance of reward formulations has been acknowledged in prior studies[19, 45]. Generally, more complex reward function lead to a more rapid convergence of a control policy, while avoiding unintended control actions, such as the chattering behavior seen in figure 3. In fact, prior studies have attributed such erratic control actions to the use of oversimplified reward functions [45], but have offered little specificity or concrete design recommendations that could be used to avoid such actions. As such, our approach heuristically evaluates reward formulations of increasing complexity until arriving at one that mostly meets desired outcomes. This introduces an element of design into the use of RL for the real-time control of stormwater, as the one cannot simply rely on the implicit black box nature of neural networks to solve a control problem under complex system dynamics. The reward function needs to embed enough information to help guide the RL agent to a stable solution. This introduces only a limited amount of overhead, as reward functions can be intuitively formulated by someone with knowledge of basic hydrology.

For control individual basins, the cost function presented here should be directly transferable. If more complex outcomes are desired, modifications to the cost function may need to be carried out. Objectively, the convergence of the reward will serve as one quality measure of control performance. The ultimate performance of RL for the control of individual sites will, however, need to be assessed on a case-by-case basis by a designer familiar with the application. Taking the baseline lessons learned during the control of a single basin, the second scenario can now evaluate the simultaneous control of multiple basins.

*4.2. **Scenario 2: Control of multiple basins***

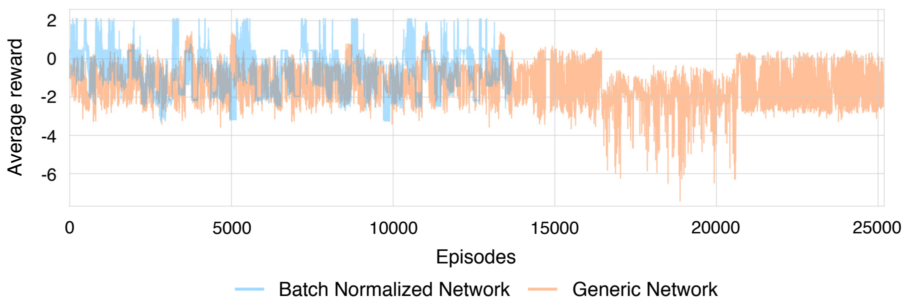

Figure 4: Average reward earned by the RL agent when learning to control multiple basins. The use of neural network batch normalization (blue) leads to consistently higher rewards when compared to the use of a generic neural network (orange). The batch normalized network also leads to higher rewards earlier in the training process.

When trained using the generic feed forward neural network configuration that was used for the control of a single basin, the RL agent controlling multiple assets was unable to converge to a stable reward, even after 25000 episodes of training (Figure 4). This totaled to 52 days of computation time on our GPU cluster, after which the training procedure was halted due to lack of improved reward. Overall, learning performance was low in this configuration. Not only did the learning procedure not converge to stable reward, but the vast majority of rewards were negative. Given this observation, this ineffective neural network was then replaced with one that was batch normalized. The agent using the batch normalized neural network achieved a higher average reward than the agent with a generic feed forward neural network (Figure 4). Furthermore, the agent using the batch normalized neural network achieved a relatively high rewards early on in the training process, thus making it more computationally favorable.

Even with batch normalization, the RL agent did not consistently return to the same reward or improve its performance when perturbed. The exploration in its policy caused the RL agent to oscillate between local reward maxima. Similar outcomes have been observed in a number of RL benchmark problems[34, 7], which exhibited a high degree of sensitivity to their exploration policy. Prior studies have noted that the exploration-exploitation balance is difficult to parameterize because neural networks tend to latch onto a local optimum [46]. As such, it is likely that the lack of convergence observed in this scenario was caused by the use of a neural network as a function approximator. Forcing neural networks to escape local minima is still an ongoing problem of research [47]. Nonetheless, even without a consistent optimum, the maximum reward obtained during this scenario can still be used as part of an effective control approach.

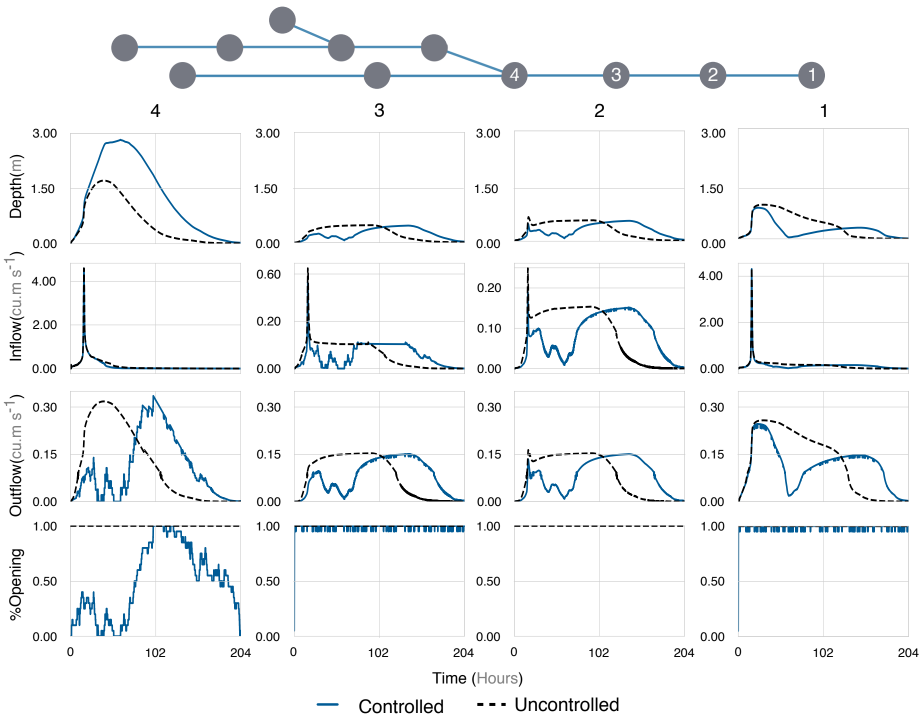

Figure 5: RL agent controlling multiple stormwater basins during a 6-hour, 25-year storm event. Control actions at each of the controlled basins are shown as valve settings in the fourth row of the plot. In this scenario, the agent achieves a high reward by just controlling the most upstream control asset (4) and shifting the peak of the hydrograph.

Selecting the episode with the highest reward revealed the actions taken by the RL agent during the training storm (Figure 5). The figure compares the controlled and uncontrolled states of the four basins during a 25-year 6-hour storm event, showing the depth in each basin, inflows, outflows, and control actions taken by the RL agent. No flooding occurred during this simulation, which means that the reward received by the RL agent was entirely obtained by meeting outflow objectives. The valves on basins 1 and 3 throttled between 100% and 95% open, which for all practical considerations could be considered uncontrolled. As such, the RL agent in this scenario earned its reward by only controlling the most upstream basin in this network.

While the outcome of control was somewhat favorable compared to the uncontrolled systems, the playback of the highest reward in figure 5 does not show drastically different outcomes. Control of the 4^{th} basin shifted the timing of the outflows from the basin but did not reduce its outflows. This resulted in improvements at the 1^{st}, 2^{nd} and 3^{rd} basins. By delaying flows from the 4^{th} basin, the RL agent allowed the downstream basins to drain first and to spend less time exceeding the flow threshold. Interestingly, the RL agent did not control basin 1, even while the single-basin control scenario makes it is clear that a more favorable outcome can be achieved with control (Figure 3). As such, a better control solution may exist, but converging to such a solution using a neural network approximator is difficult. This likely has to do with the larger state action space. While the site-scale RL agent was only observing water level at one basin, the system level RL agent had to track levels and flows across more basins, which increases the complexity of the learning problem. The rewards received by the RL agent in the scenario are cumulative, which means that improvement at just a few sites can lead to better rewards, without the need to control all of them. Increasing the opportunity to obtain rewards thus increases the occurrence of local minima during the learning phase.

In the single basin control scenario the RL agent can immediately observe the impact of its control actions. In the system scale scenario more time is needed to observe water flows through the broader system, which means that the impact of a control action may not be observed until later timesteps. This introduces a challenge, as the RL agent has to learn the temporal dynamics of the system. This challenge has been observed in other RL studies, which have shown better performance for reactive RL problems, as opposed to those that are based on the need to plan for future outcomes [35]. The need to include planning is still an active area of RL research. Potential emerging solutions include adversarial play[20, 21], model-based RL[48], and policy-based learning

615 [49]. The benefits of these approaches have recently been demonstrated for other application domains and should be considered in the future for the control of water systems.

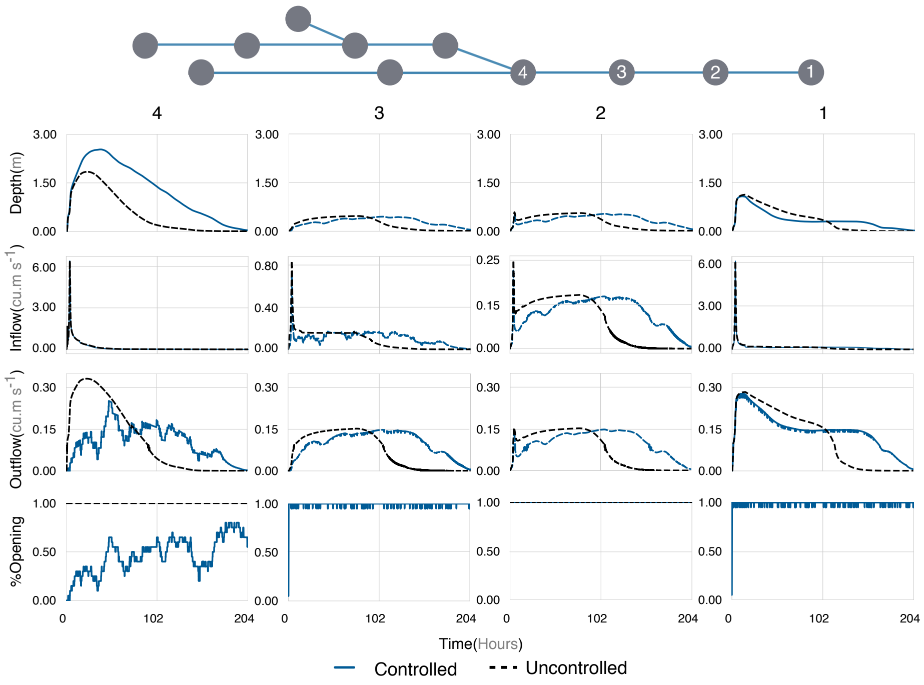

Figure 6: RL agent controlling multiple stormwater basins during a 24-hour, 10-year storm event. Control actions at each of the controlled basins are shown as valve settings in the fourth row of the plot. In this scenario, the agent achieves a high rewar, by maximizing the storage utilization in the most upstream control asset (4) and regulating the outflow from it to meet the downstream objectives.

It is important to note that figure 5 represents an evaluation of the RL agent for one storm only namely, the training storm. Realistically, the control system

620 will need to respond to storms of varying durations and magnitudes. As an example, the RL agents response to a 24-hour, 10-year storm is shown in figure 6. Here, the RL agent outperformed the uncontrolled system much more notably compared to the training storm. The controlled outflows were much closer to

the desired threshold, even when only one basin was controlled. This broader performance is captured in figure 7, which quantifies performance (eq 15) across a spectrum of storm inputs. Figure 7 compares the uncontrolled system to the RL-controlled system. Both the controlled and uncontrolled systems perform equally well during small-magnitude and short events (e.g. the training storm in Figure 5). The benefits of control become more pronounced for larger events, starting at 10-year storms and those that last over 2 hours. This visualization holistically captures the benefits of real-time control by highlighting new regions of performance and showing how control can push existing infrastructure to perform beyond its original design.

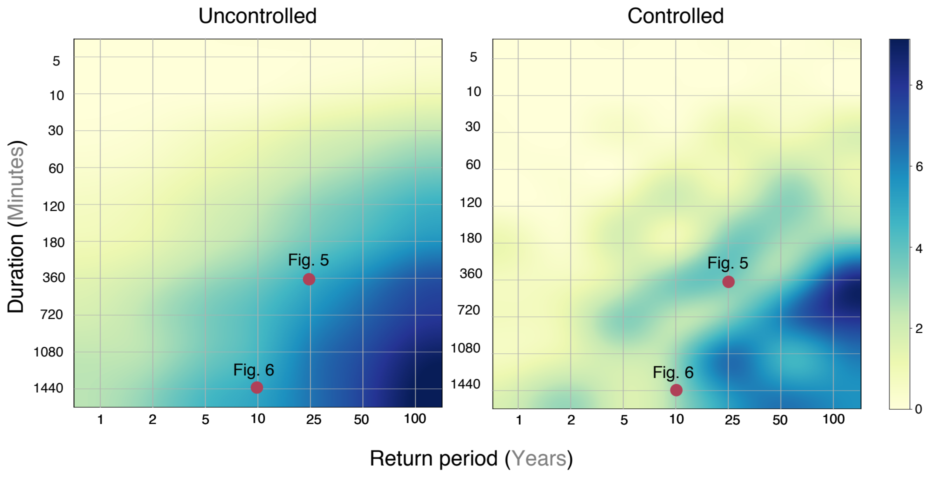

Figure 7: Performance of stormwater system (eq 15) for the uncontrolled system (left) and RL-controlled system (right). The use of RL control enhances the performance stormwater network by allowing the system to achieve desired outcomes across larger and longer storms. The lighter color corresponds with a relatively better performance.

5. Discussion

Given the recent emergence and popularity of Reinforcement Learning, much research still remains to be conducted to evaluate its potential to serve as a viable methodology for controlling water systems. Our study brings to light a number of benefits and challenges associated with this task. Arguably, it seems that the major benefit of using RL to control water systems is the ability to simply hand the learning problem to a computer without needing to worry about the many complexities, nonlinearities and formulations that often complicate other control approaches. However, as this study showed, this comes with a number of considerable caveats. These include the challenges associated with formulating rewards, choosing function approximators, deciding on the complexity of the control problem, as well as contending with practical implementation details.

Our study confirms that the performance of RL-based stormwater control is sensitive to the formulation of the reward function, which has also been observed in other application domains[45]. The formulation of the reward function requires domain expertise and an element of subjectivity, since the RL agent has to be given guidance on what constitutes appropriate actions. In the first scenario it was shown that a reward function that is too simple may lead to adverse behavior, such as the chattering or sudden actions. The reward may also not converge to a stable solution since the neural network can take advantage of the simple objective to maximize costs using sudden or unintuitive actions. The formulation of the problem, which depends heavily on neural networks, also makes it difficult to determine why one specific reward function may work better than another. Increasing the complexity of the reward function was shown here to help guide the RL agent to a more desirable outcome. Reward formulations are an ongoing research area in the RL community and some formal methods have recently been proposed to provide a more rigorous framework for reward synthesis [50]. These formulations should be investigated in the future.

Even when the choice of reward function is appropriate or justifiable, the control performance can become sensitive to the approximation function, which in our case took the form of a deep neural network. Choosing the architecture and structure of the underlying network becomes an application dependent task and can often only be derived through trial and error [19, 34]. Secondly, for challenging control problems, such as the one studied here, learning the mapping between rewards and all possible control decisions becomes a complex task. The neural network must be exposed to as many inputs and outputs as possible, which is computationally demanding. In our study we ran simulations for many real-world months on a high performance cluster, but it appears that the learning phase could have continued even longer. This, in fact, has been the approach of many successful studies in the RL community, where the number of computers and graphical processing units can be in the hundreds [51, 52]. This was not feasible given our own resources, but could be evaluated in the future.

Aside from the formulation of the learning functions and framework, the actual complexity and objectives of the control problem may pose a barrier to implementation. We showed that an RL agent can learn how to control a single stormwater basin effectively, but that controlling many sites at the same time is difficult. A major reason is the increase in the number of states and actions that must be represented using the neural network. While computational time may remedy this concern, the structure of the neural network may also need to be altered. In a system-scale stormwater scenario, actions at one location may influence another location at a later time. As such, the agent would benefit from a planning-based approach which considered not only current states, but future forecasts as well. Such planning-based approaches have been proposed in the RL literature and should be investigated to determine if they lead to an improvement in performance [48, 53]. Furthermore, model-based approaches have also recently been introduced and could allow some elements neural network to be replaced with an actual physical or numerical water model [54]. Such approaches should be evaluated in the future since they may permit more domain knowledge from water resources to be embedded in the learning problem.

Finally, the use of RL for the control of stormwater systems is underpinned by a number of practical challenges. Computational demands are very high, especially compared to competing approaches, such as dynamical systems control, model predictive control, or market-based controls. While computational resources are becoming cheaper, the resources require to carry out this study were quite significant and time demanding. Since actions taken by neural networks cannot easily be explained and explicit guarantees cannot be provided, this may limit adoption by decision makers who may consider the approach a black box. It is also unlikely that the control of real-world stormwater systems will simply be handed over to a computer that learns through mistakes. Rather, simulation-based scenarios will be required first. It has recently been shown as long as a realistic simulator is used - in our case SWMM - then the agent can be effectively trained in a virtual environment before refining its strategy in the real world[52].

6. Conclusion

This paper introduced an algorithm for the real-time control of urban drainage systems based on Reinforcement Learning (RL). While RL has been used successfully in the computer science communities, to our knowledge this is the first instance for which it has been explicitly adopted for the real-time control of urban water systems. The methodology and our implementation show promise for using RL as an automated tool-chain to learn control rules for simple storage assets, such as individual storage basin. However, the use of RL for more complex system topologies faces a number of challenges, as laid out in the discussion.

Simultaneously controlling multiple distributed stormwater assets across large urban areas is a non-trivial problem, regardless of the control methodology. To that end, the concepts, initial results and formulations provided by this paper should help build a foundation to support RL as a viable option for stormwater control. The source code accompanying this paper should also allow others to evaluate many other possible architectures and parameterizations that could be used to improve the results presented in the paper.

Acknowledgements

This research was supported by Great Lakes Protection Fund (grant number # 1035) and the US National Science Foundation (grant number # 1737432). We also would like to acknowledge the support provided by Advanced Research Computing at the University of Michigan, Ann Arbor.

Appendix A: Deep Neural Networks

Broadly, neurons are the fundamental processing elements of a neural network. They receive their inputs (x_{ij}) as the weighted (w_{ij}) outputs from the neurons in the previous layers and produce a single output signal (y_j) dependent upon a bias (b_j) and activation function $f(*)$

$$z_j = \sum_i^n w_{ij} x_{ij} + b_j \tag{A1}$$

$$y_j = f(z_j) \tag{A2}$$

More specifically, deep neural networks are a collection of such neurons organized as distinct layers. A neural network approximates a function by fine tuning its weights and biases so that its output signal closely resembles the output of the function it is approximating for the same inputs. The degree of resemblance between the signals is computed based on a loss function $L(*)$

$$Loss = L(Q_p, Q_o) \tag{A3}$$

$$w_{ij} = w_{ij} - \alpha \times \frac{dL(Q_p, Q_o)}{(dw_{ij})} \tag{A4}$$

$$b_j = b_j - \alpha \times \frac{dL(Q_p, Q_o)}{(db_j)} \tag{A5}$$

The choice of the loss function is dictated by the nature of the function being approximated. For example, a neural network approximating rainfall runoff may use mean squared error [55]

$$Loss = |Q_p - Q_o|^2 \tag{A6}$$

The closer the correspondence between the signals, lower the loss.

Neural networks minimize the loss though stochastic gradient descent, starting with a set of weights and biases (either random or values uniformly sampled from a distribution). Based on the value of loss function, the values of weights and biases are adjusted, and the neural network attempts to approximate the function with these updated values. This process of tuning the weights and biases is repeated until the neural network can approximate the function to satisfaction or loss is minimized. While deep neural networks show significant promise in approximating functions, their ability to do so is contingent upon several factors: Stability of the learning process, the size and depth of the network, the underlying data distributions[42, 56]. Fundamental descriptions of deep neural networks in established textbooks (e.g. Deep Learning [42]).

Appendix B: Hyper parameters and architecture

Neural Network

Layers	2
Number of Neurons per layer	50

Gradient Descent

Learning Rate	10^{-3}
Rho	0.9
Epsilon	10^{-8}
Decay	0.0

Batch Normalization

Momentum	0.99
Epsilon	0.001

Deep Q Network

Target Network Update	10000
Gamma	0.99
Replay Buffer	100000

References

[1] K. U. Laris Karklis, J. Muyskens, Before-and-after photos of Harvey flooding in Texas - Washington Post (2017).
URL `https://www.washingtonpost.com/graphics/2017/national/harvey-photos-before-after/?utm_term=.5976b1e7a7ed`

[2] S. B. Watson, C. Miller, G. Arhonditsis, G. L. Boyer, W. Carmichael, M. N. Charlton, R. Confesor, D. C. Depew, T. O. Höök, S. A. Ludsin, G. Matisoff, S. P. McElmurry, M. W. Murray, R. Peter Richards, Y. R. Rao, M. M. Steffen, S. W. Wilhelm, The re-eutrophication of Lake Erie: Harmful algal blooms and hypoxia, Harmful Algae 56 (2016) 44–66. doi:10.1016/J.HAL.2016.04.010.
URL https://www-sciencedirect-com.proxy.lib.umich.edu/science/article/pii/S1568988315301141

[3] B. Kerkez, C. Gruden, M. J. Lewis, L. Montestruque, M. Quigley, B. P. Wong, A. Bedig, R. Kertesz, T. Braun, O. Cadwalader, A. Poresky, C. Pak, Smarter Stormwater Systems, Environmental Science & Technology 50 (14) (2016) acs.est.5b05870. doi:10.1021/acs.est.5b05870.
URL http://pubs.acs.org/doi/abs/10.1021/acs.est.5b05870

[4] C. H. Emerson, C. Welty, R. G. Traver, Watershed-Scale Evaluation of a System of Storm Water Detention Basins, Journal of Hydrologic Engineering 10 (3) (2005) 237–242. doi:10.1061/(ASCE)1084-0699(2005)10:3(237).
URL http://ascelibrary.org/doi/10.1061/%28ASCE%291084-0699%282005%2910%3A3%28237%29

[5] A. Mullapudi, B. P. Wong, B. Kerkez, Emerging investigators series: building a theory for smart stormwater systems, Environ. Sci.: Water Res. Technol.doi:10.1039/C6EW00211K.
URL http://xlink.rsc.org/?DOI=C6EW00211K

[6] M. Schütze, A. Campisano, H. Colas, W. Schilling, P. A. Vanrolleghem, Real time control of urban wastewater systemswhere do we stand today?,

Journal of Hydrology 299 (3) (2004) 335–348. doi:10.1016/j.jhydrol.2004.08.010.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533. doi:10.1038/nature14236.
URL http://dx.doi.org/10.1038/nature14236

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971.

[9] The European Parliament and the council of European Union, Directive 2000/60/EC of the European Parliament and of the Council of 23 October 2000 establishing a framework for Community action in the field of water policy, Official Journal of the European Communities (2000) 01 – 73.
URL http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32000L0060

[10] A. L. Mollerup, P. S. Mikkelsen, D. Thornberg, G. Sin, Controlling sewer systems a critical review based on systems in three EU cities, http://dx.doi.org/10.1080/1573062X.2016.1148183.

[11] M. Mahmoodian, O. Delmont, G. Schutz, Pollution-based model predictive control of combined sewer networks, considering uncertainty propagation, International Journal of Sustainable Development and Planning 12 (01) (2017) 98–111. doi:10.2495/SDP-V12-N1-98-111.
URL http://www.witpress.com/journals/SDP-98-111

[12] M. Pleau, H. Colas, P. Lavallee, G. Pelletier, R. Bonin, Global optimal real-time control of the quebec urban drainage system, Environmental Modelling & Software 20 (4) (2005) 401–413.

[13] P.-J. van Overloop, S. Weijs, S. Dijkstra, Multiple Model Predictive Control on a drainage canal system, Control Engineering Practice 16 (5) (2008) 531–540. doi:10.1016/J.CONENGPRAC.2007.06.002.
URL https://www-sciencedirect-com.proxy.lib.umich.edu/science/article/pii/S0967066107001190

[14] E. Meneses, M. Gaussens, C. Jakobsen, P. Mikkelsen, M. Grum, L. Vezzaro, Coordinating Rule-Based and System-Wide Model Predictive Control Strategies to Reduce Storage Expansion of Combined Urban Drainage Systems: The Case Study of Lundtofte, Denmark, Water 10 (1) (2018) 76. doi:10.3390/w10010076.
URL http://www.mdpi.com/2073-4441/10/1/76

[15] B. Wong, B. Kerkez, Real-time control of urban headwater catchments through linear feedback: performance, analysis and site selection, Water Resources Research.

[16] P.-J. Van Overloop, Model predictive control on open water systems, IOS Press, 2006.

[17] K. Ogata, Modern Control Engineering, 5th Edition, Prentice Hall, 201.
URL https://books.google.com/books?id=Wu5GpNAelzkC&q=Modern+Control+Engineering&dq=Modern+Control+Engineering&hl=en&sa=X&ved=0ahUKEwjtzbbizbfOAhVU-GMKHZy0ChcQ6AEIHDAA

[18] L. Vezzaro, M. Grum, A generalised dynamic overflow risk assessment (dora) for real time control of urban drainage systems, Journal of Hydrology 515 (2014) 292–303.

[19] R. Sutton, A. Barto, Reinforcement learning, MIT Press, Cambridge, 1998.

[20] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al., Mastering the game of go without human knowledge, Nature 550 (7676) (2017) 354.

[21] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al., Mastering chess and shogi by self-play with a general reinforcement learning algorithm, arXiv preprint arXiv:1712.01815.

[22] J. Kober, J. A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, The International Journal of Robotics Research 32 (11) (2013) 1238–1274. `doi:10.1177/0278364913495721`.
URL `http://journals.sagepub.com/doi/10.1177/0278364913495721`

[23] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, E. Liang, Autonomous inverted helicopter flight via reinforcement learning, in: Experimental Robotics IX, Springer, 2006, pp. 363–372.

[24] B. Abdulhai, L. Kattan, Reinforcement learning: Introduction to theory and potential for transport applications, Canadian Journal of Civil Engineering 30 (6) (2003) 981–991.

[25] B. Bhattacharya, A. Lobbrecht, D. Solomatine, Neural networks and reinforcement learning in control of water systems, Journal of Water Resources Planning and Management 129 (6) (2003) 458–465.

[26] A. Castelletti, S. Galelli, M. Restelli, R. Soncini-Sessa, Tree-based reinforcement learning for optimal water reservoir operation, Water Resources Research 46 (9).

[27] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: Advances in neural information processing systems, 2000, pp. 1057–1063.

[28] C. J. C. H. Watkins, P. Dayan, Q-learning, Machine Learning 8 (3-4) (1992) 279–292. `doi:10.1007/BF00992698`.
URL `http://link.springer.com/10.1007/BF00992698`

[29] R. S. Sutton, Planning by incremental dynamic programming, in: Machine Learning Proceedings 1991, Elsevier, 1991, pp. 353–357.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing Atari with Deep Reinforcement Learning, arXiv preprint arXiv: ... (2013) 1–9`doi:10.1038/nature14236`.
URL `http://arxiv.org/abs/1312.5602`

[31] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (5) (1989) 359–366.

[32] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444. `doi:10.1038/nature14539`.
URL `http://www.nature.com/articles/nature14539`

[33] M. Bartos, B. Wong, B. Kerkez, Open storm: a complete framework for sensing and control of urban watersheds, arXiv preprint arXiv:1708.05172.

[34] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, D. Meger, Deep reinforcement learning that matters, arXiv preprint arXiv:1709.06560.

[35] Y. Aytar, T. Pfaff, D. Budden, T. L. Paine, Z. Wang, N. de Freitas, Playing hard exploration games by watching youtube, arXiv preprint arXiv:1805.11592.

[36] L. A. Rossman, Storm Water Management Model User's Manual Version 5.1, US Environmental Protection Agency, 2010.

[37] A. Elliott, S. Trowsdale, A review of models for low impact urban stormwater drainage, Environmental Modelling & Software 22 (3) (2007) 394–405. `doi:10.1016/J.ENVSOFT.2005.12.005`.
URL `https://www.sciencedirect.com/science/article/pii/S1364815206000053`

[38] G. Riaño-Briceño, J. Barreiro-Gomez, A. Ramirez-Jaime, N. Quijano, C. Ocampo-Martinez, MatSWMM An open-source toolbox for designing real-time control of urban drainage systems, Environmental Modelling & Software 83 (2016) 143–154. `doi:10.1016/J.ENVSOFT.2016.05.009`.
URL `https://www.sciencedirect.com/science/article/pii/S1364815216301451`

[39] CDMSmith, City of ann arbor stormwater model calibration and analysis project, accessed: 2018-09-25 (2015).

[40] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, nature 521 (7553) (2015) 436.

[41] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: a system for large-scale machine learning., in: OSDI, Vol. 16, 2016, pp. 265–283.

[42] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep learning, Vol. 1, MIT press Cambridge, 2016.

[43] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167.

[44] U. Scs, Urban hydrology for small watersheds, technical release no. 55 (tr-55), US Department of Agriculture, US Government Printing Office, Washington, DC.

[45] A. Y. Ng, D. Harada, S. Russell, Policy invariance under reward transformations: Theory and application to reward shaping, in: ICML, Vol. 99, 1999, pp. 278–287.

[46] H. Larochelle, Y. Bengio, J. Louradour, P. Lamblin, Exploring strategies for training deep neural networks, Journal of machine learning research 10 (Jan) (2009) 1–40.

[47] I. Osband, C. Blundell, A. Pritzel, B. Van Roy, Deep exploration via bootstrapped dqn, in: Advances in neural information processing systems, 2016, pp. 4026–4034.

[48] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, P. Abbeel, Model-based reinforcement learning via meta-policy optimization, arXiv preprint arXiv:1809.05214.

[49] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347.

[50] J. Fu, K. Luo, S. Levine, Learning robust rewards with adversarial inverse reinforcement learning, arXiv preprint arXiv:1710.11248.

[51] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al., Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, arXiv preprint arXiv:1802.01561.

[52] OpenAI, Openai five (2018).

[53] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, S. Udluft, Learning and policy search in stochastic dynamical systems with bayesian neural networks, arXiv preprint arXiv:1605.07127.

[54] S. Gu, T. Lillicrap, I. Sutskever, S. Levine, Continuous deep q-learning with model-based acceleration, in: International Conference on Machine Learning, 2016, pp. 2829–2838.

[55] A. S. Tokar, P. A. Johnson, Rainfall-runoff modeling using artificial neural networks, Journal of Hydrologic Engineering 4 (3) (1999) 232–239.

[56] F. Chollet, Deep learning with python, Manning Publications Co., 2017.