PAPER • OPEN ACCESS

Gaming the beamlines—employing reinforcement learning to maximize scientific outcomes at large-scale user facilities

To cite this article: Phillip M Maffettone *et al* 2021 *Mach. Learn.: Sci. Technol.* **2** 025025

View the article online for updates and enhancements.

This content was downloaded from IP address 49.207.216.61 on 15/04/2021 at 14:45

MACHINE
LEARNING
Science and Technology

CrossMark

OPEN ACCESS

RECEIVED
11 September 2020

REVISED
22 October 2020

ACCEPTED FOR PUBLICATION
12 November 2020

PUBLISHED
25 March 2021

Original Content from this work may be used under the terms of the Creative Commons Attribution 4.0 licence.

Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

PAPER

Gaming the beamlines—employing reinforcement learning to maximize scientific outcomes at large-scale user facilities

Phillip M Maffettone, Joshua K Lynch, Thomas A Caswell, Clara E Cook, Stuart I Campbell and Daniel Olds

National Synchrotron Light Source II, Brookhaven National Laboratory, Upton, NY 11973, United States of America

E-mail: dolds@bnl.gov

Keywords: reinforcement learning, artificial intelligence, machine learning, Bluesky, beamline science

Abstract

Beamline experiments at central facilities are increasingly demanding of remote, high-throughput, and adaptive operation conditions. To accommodate such needs, new approaches must be developed that enable on-the-fly decision making for data intensive challenges. Reinforcement learning (RL) is a domain of AI that holds the potential to enable autonomous operations in a feedback loop between beamline experiments and trained agents. Here, we outline the advanced data acquisition and control software of the Bluesky suite, and demonstrate its functionality with a canonical RL problem: cartpole. We then extend these methods to efficient use of beamline resources by using RL to develop an optimal measurement strategy for samples with different scattering characteristics. The RL agents converge on the empirically optimal policy when under-constrained with time. When resource limited, the agents outperform a naive or sequential measurement strategy, often by a factor of 100%. We interface these methods directly with the data storage and provenance technologies at the National Synchrotron Light Source II, thus demonstrating the potential for RL to increase the scientific output of beamlines, and layout the framework for how to achieve this impact.

1. Introduction

Reinforcement learning (RL) methods have received many accolades in recent years for demonstrating advances in artificial intelligence (AI) applications on systems previously considered too complex or dynamic for AI methods to be employed. Many of the recent advances in RL have been showcased on their ability to play games at levels matching or exceeding those of a human opponent, such as in chess, go [1], Atari games [2], and the video game StarCraft II [3]. Unlike supervised learning methods, where a very large, labeled set of data is employed to train, RL is trained via a dynamic interaction loop whereby the AI (or agent) learns 'on-the-fly' as it is interacting with an environment, similar to a Markov decision process [4]. The primary driver of this learning is the concept of a 'reward', which the environment awards the agent as a function of successfully completing defined goals. The goal of the process is that through training, the agent will eventually adopt those behaviors or policies which maximize the received reward. Beyond the realm of games, RL has been employed to train robots to walk [5], optimize telecom network strategies [6], and tune chemical reactions [7].

One appeal of RL methods to the greater scientific ecosystem is the ability to train via the so-called 'gamification' of a desired behavior. If it were possible to recast scientific challenges as repeatable, point bearing games, RL could be used to build autonomous scientific systems. To use an example in robotics, it is challenging to correctly program the precise motor behavior of a bipedal robot, such that it can effectively walk without falling over. However, it can be very simple to prescribe a reward-concept to this problem, such as distance traveled in an amount of time. RL agents driving these robots can then be left to explore this complex dynamical phase-space and, as has been shown, will teach themselves to walk [8]. Herein, we demonstrate the gamification and use of RL for automating scientific operations at a beamline. We first describe the software infrastructure, *Bluesky*, that enables these developments and demonstrate its

© 2021 US government

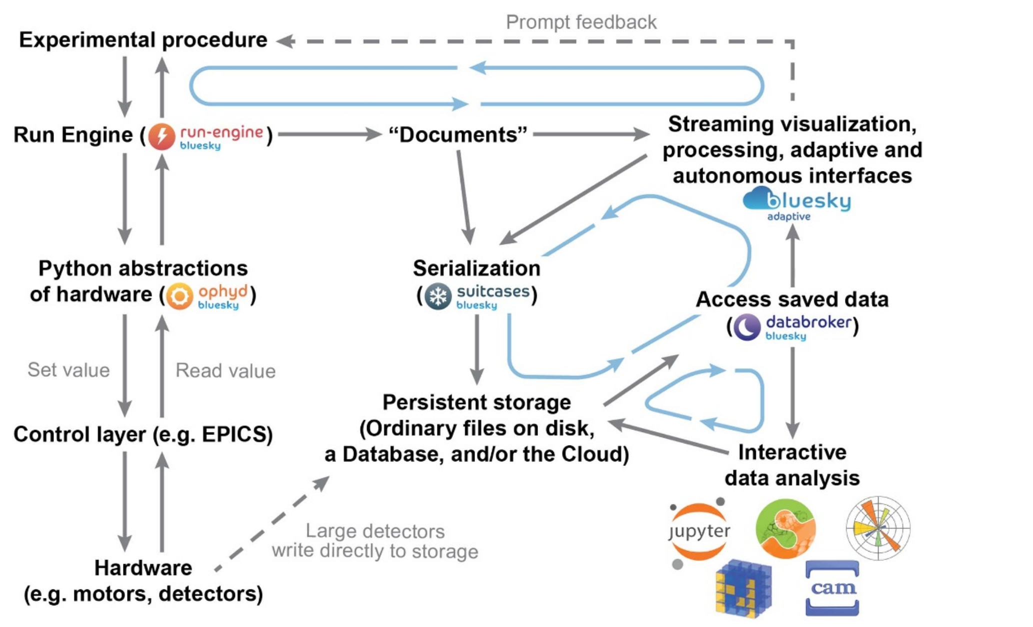
Prompt feedback
Experimental procedure
Run Engine (run-engine bluesky)
"Documents"
Streaming visualization, processing, adaptive and autonomous interfaces
bluesky adaptive
Python abstractions of hardware (ophyd bluesky)
Serialization (suitcases bluesky)
Access saved data (databroker bluesky)
Set value
Read value
Control layer (e.g. EPICS)
Persistent storage (Ordinary files on disk, a Database, and/or the Cloud)
Interactive data analysis
jupyter
cam
Large detectors write directly to storage
Hardware (e.g. motors, detectors)

Figure 1. Schematic overview of the Bluesky system. The components in the left column are responsible for orchestration of the beamline for data acquisition. As the experiment progresses, the data is published to the consumers which may serialize it to disk for later access via DataBroker, display to the screen, or preform some prompt analysis. In this paper, we are primarily interested in the loop at the top where we feed back from prompt analysis to the currently running plan via Bluesky-adaptive.

application with the cannonical RL problem of cartpole: keeping a pole upright on a mobile cart. We then use these tools to automate the measurement strategy in a typical use case where an unknown subset of samples produce a weak signal. This complements recent advancements in measurement strategies, such as compressive sensing [9], adaptive learning [10], and RL-based instrument controls [11], and has immediate pertinence across the many experiments at large-scale user facilities.

Large-scale user facilities, such as the National Synchrotron Light Source II (NSLS-II) at Brookhaven National Laboratory (BNL), are a great strategic resource to thousands of researchers in a diverse set of fields including physics, chemistry, engineering, biology, and material science. At synchrotron lights sources such as the NSLS-II, users take advantage of the intense, highly focused light (often x-rays) to study the structure and dynamics of materials down to the atomic scale through various specialized techniques offered at different beamlines such as diffraction, spectroscopy, imaging, and tomography. On modern beamlines at these facilities, the configuration, optimization, and the data collection procedures employed present a complex and multi-variable phase-space. Optical components (monochromators, mirrors, choppers, beam-defining slits, filters) can require careful alignment to produce the desired beam characteristic at the sample. Detectors themselves present challenges in both the positioning and operating configuration for optimising a desired measurement.

The use of these beamlines is typically allocated to researchers under a competitive general user program, in which those successful proposals are awarded an allotment of beamtime based on the anticipated needs to complete the defined work. This work can range widely from straightforward *ex situ* characterization of static samples under ambient conditions to complex *in situ* studies of advanced sample environments and everything in between. It is common for a single beamtime to involve multiple phases, which include a number of different samples and experiments all related to an overarching theme of study.

The general users of these facilities are visiting researchers there to carry out, with the aid of the beamline staff, the measurement described in their proposal. To this end, while these users are adept in their own scientific domain, they often rely on the expertise of the facility staff for training and guidance on beamline operations.

For a visiting user group, a single allocation of beamtime can often last from hours to days, during which experimental plans are often adjusted and altered on-the-fly based on the as-measured results. For example, a sample of particular interest may prove less straightforward to measure than planned, requiring the dedication of additional measurement time to reach the required statistics. Such a decision may require the group to drop lower-priority parts of their planned experiment. Thus, even well planned experiments by experienced user groups can involve dynamic decision-tree like processes based on real-time feedback and data interrogation at the beamline.

Given the complex phase-space that optimization of beamline configuration and data collection strategies can lie within, automation through AI agents trained with RL offers a potential paradigm shift in the approach to beamline operations. Through the use of reward-based objectives and realistic system simulations, the programming of complex tasks can be accomplished using only a set of researcher-prescribed desired behaviors. Although much development is still required to realize this vision, the accessible and easy-to-use RL toolsets available today serve as an excellent starting point to foster and grow the community towards realizing this future.

In this paper, we demonstrate the use of RL methods for optimizing beamline operations. We first introduce the Python-based Bluesky suite of data acquisition software, and explain how it enables RL applications at a beamline. We demonstrate this functionality by solving a classical RL problem, cartpole, in the Bluesky environment. We then demonstrate the use of RL methods to address a scenario often encountered on high-throughput beamlines: maximizing data quality across multiple samples of different scattering strength within a limited window of time. An approach for interfacing with and exploiting the data storage and provenance technologies at the NSLS-II is discussed in relation to AI efforts. Finally, we layout the challenges and overall strategy to realizing wider-spread development of RL methods at large-scale user facilities.

2. Enabling advanced data acquisition and control with the Bluesky suite

2.1. The 'Why' of Bluesky

The Bluesky Suite is a set of tools, originally developed at NSLS-II, for data acquisition and management [12, 13] that is now being used at both large-scale user facilities and small research labs [14]. The tools were designed from the ground up to support streaming data analysis and intelligent, compute-in-the-loop, data acquisition as first-class concepts. Bluesky is implemented in Python which has an extensive and well used eco-system of high-quality scientific libraries that can be leveraged and is highly accessible to scientists. Taken together, these make Bluesky uniquely suited for training and integrating AI agents into experimental workflows.

A design goal of Bluesky is provide both streaming access to the primary scientific data and the associated metadata for prompt on-the-fly analysis and efficient bulk-access to historical data for post-hoc analysis. All data and metedata generated during an experiment using Bluesky is in real-time emitted to other processes (consumers) in the form of 'documents', that is Python dictionaries with well defined schema. This document-model approach to data distribution can occur either locally or distributed over a network. This approach readily allows for data processing pipelines where, for example, an analysis node can listen for raw-data emitted during a measurement, perform some prescribed data reduction, and then broadcast that reduced data as additional documents. Access to historical data is through an API called DataBroker, which presents a consistent interface for all data to the user. At NSLS-II the data is stored in a combination of a Mongo database, for searching and rich structured meta-data, and a variety of beamline and hardware specific formats for image data. However, independent of how the data is stored on disk, the API for data access is uniform. Thus, Bluesky's document-based data model supports complex, asynchronous data collection and enables sophisticated live, prompt, streaming, and post-facto data analysis.

All beamline hardware is accessed via a library called ophyd, which like Bluesky and DataBroker has a consistent Python API. The orchestration and scheduling of these processes during a single experiment (or 'Run') is handled by the RunEngine. A specific experiment is then implemented in Bluesky as a Python function, referred to as a 'plan', addressing specific hardware components (motors, detector) to carry out the measurement of interest. Smaller plans can be used in larger plans, allowing for complex experiments to be constructed out of simpler building blocks.

2.2. The Bluesky-adaptive harness

Many use cases of an AI agent at a beamline will operate in an interaction loop with the instrumentation and data. Depending on the scale of response-time required for the task at hand, there are different places that this beamline-to-agent interaction must be implemented and consequentially different amounts of information that can be ferried to the agent. For instance, in the case of very fast processes that require sub-millisecond decisions, such as a hardware control loop dynamically maintaining alignment of an optical component, it may become necessary to place the agent in direct access to the hardware-control layer (below ophyd). Contrasting, if the required response time of an agent is relatively slow, on the scale of seconds-to-minutes or longer, it can interact downstream of the Bluesky control layer such that it is consuming documents directly via a message stream (e.g. using Kafka) [15] or after-the-fact through inspection of DataBroker. In this way, the agent gains access to the full breadth of information available both historically and in the document data stream. There likely exists special use-cases for intermediary levels of interaction loops between the very-short and long described here. However, when possible it is advisable that all AI interactions be pushed into the latter modal—downstream of the Bluesky control regime. This is because standardized interface protocols for Bluesky, such as the methods used herein, can be broadly applied at this level eliminating the need for bespoke control solutions unique to each AI agent. Additionally, whether the agents are running as processes locally inspecting data in real-time, remotely being streamed data, or years later training on the results of past experiments, the fundamental interface can be unified.

Here, we present a general implementation to facilitate the use of AI agents at this downstream layer called Bluesky-adaptive [16]. This is an implementation of a generic plan allowing AI agents to select the next decision-point in the experimental phase space to measure (e.g. motor position, temperature set point, sample number, etc). The separation of components also makes it straightforward to distribute the computation, such that the agent need not be run locally.

In Python pseudo code the adaptive plan can be described as:

```
def adaptive_plan(first_point, agent, ...):
  yield from subscribe_to_stream(agent)
  next_point = first_point
  while next is not Done:
    yield from move_to(next_point)
    yield from take_data()
    next_point = get_next(agent)
```

and the agent is:

```
def agent_callback(data):
  next_point = do_compute(data)
  5publish_to_plan(next_point)
```

By defining a protocol between the experimental plan and the agents, we may readily integrate any compliant agent, regardless of the underlying code and dependencies. This effectively separates the decision-making logic from the execution of the plan and acquisition the data at that step. This maintains the possibility to incorporate future advanced recommendation logic without having to dramatically change the core beamline acquisition code.

3. Solving cartpole with Bluesky

We demonstrate RL in a beamline-like context by training an AI agent to play cartpole using the Bluesky framework. Cartpole is a standard benchmark RL environment often used to compare learning by different RL agents [17]. It is a simple game based on the inverted pendulum with two elements: a cart that is allowed to move left and right and a pole balanced vertically on the cart. The game is played in turns, with each turn representing a finite amount of time passing. At the beginning of each turn the agent observes the current state of the cartpole system (position and velocity of both the cart and pole) and chooses one of two actions: (1) nudge the cart to the left or (2) nudge the cart to the right. The turn is completed by calculating forward over a fixed time step the dynamics of the cart and pole using the initial state and the selected move.

The goal is for an agent to learn how to move the cart so that pole does not fall over for as long as possible while keeping the cart within defined bounds. Each episode of the game is initialised with a different random initial configuration of the cart and pole, such that no trivial state-independent policy (e.g. move left, move right, repeat) will be universally successfully. In essence, the agent must learn to make decisions based on observations of the environment and act accordingly, even if that action space is small. To develop an optimal policy, an agent must balance exploitation—making decisions it expects to receive a reward from—and exploration—making random decisions or decisions with an uncertain reward—while it learns.

Key to RL is the concept of a reward (or points) given to the agent for successfully completing certain tasks. In the case of cartpole, the agent is rewarded with a point for each turn the game continues, and the game ends when the pole has fallen past a terminal angle. If the cart is moved incorrectly or randomly for several turns, the pole will quickly fall over. An agent is evaluated in terms of how well it learns to play the game, specifically how many points received before the game ends. The problem is generally considered solved if an agent is able to score an average of 195 points over 100 consecutive episodes. The schematic demonstrating the interaction loop of an RL agent with cartpole is shown in figure 2.

Framed in a beamline context, the cart is analogous to a motor (these are the actuators that change state of the environment), the pole is analogous to a detector (these are the sensors or parts of interest in the environment), and the reward is given for optimal decision making (keeping the pole upright, or making a good measurement). We employed the OpenAI cartpole environment [18] and both the Tensorforce

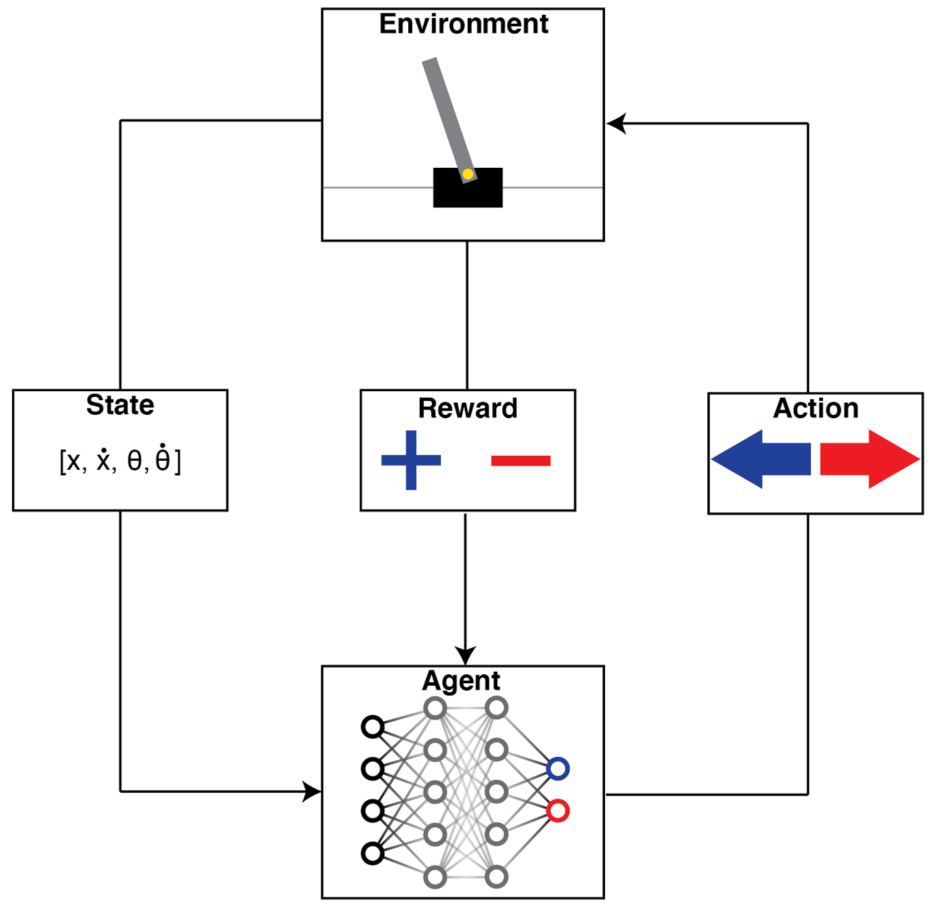

Figure 2. The cartpole environment consist of a rigid rod connected to a cart by a single joint. The RL agent is aware of the position and velocity of the cart, the angle of the rod, and the angular velocity. The action space consists of left or right movements with rewards given for keeping the pole above an angular threshold.

implementation of proximal policy optimization (PPO) and Advantage Actor Critic (A2C) methods to generate and compare agents [19]. We developed ophyd interfaces to the OpenAI environment and employed the Bluesky-adaptive framework described in section 2.2. We trained the agent by reading and acting on the environments strictly through the ophyd layer, as though the environment were a motor (controlling cart motion) and a detector (reports position and velocity of the pole). While training, the results were streamed and saved in DataBroker so that both plots could be readily accessed and agents stored with all associated training metadata.

For the PPO training, batch sizes of 10 were used with variable noise 0.1 and a maximum episode length of 500 turns. The results of training over 1000 episodes are shown in figure 3. Note that both the reward during training and a periodically performed 'evaluation mode', in which the agent does not attempt any exploration but only selects based on it is current optimal policy, are shown. We see that the PPO agent successfully solves cartpole around episode 200, and then continues to improve until the agent is able to often achieve a maximum score (500) around episode 800.

For the A2C training, batch sizes of 11 were used with variable noise 0.1, L2-regularization of 0.05, and a horizon of 10. The results of training over 1000 episodes are shown in figure 4. We see that the agent solves cartpole in less approximately 170 episodes—faster than the PPO agent. However, shortly thereafter the agent begins to underperform, with the agent averaging less than 150 points even after training continues for 1000 episodes as seen in figure 4 (right). This is likely a consequence of fixed memory: as the agent begins to succeed regularly, it begins to forget states that led to failure, and thus forgets how to behave in those circumstances. This may be avoided if the agent was run again with a different learning rate scheduler or the hyper-parameters were tuned (such as memory buffer size), and is discussed further in section 5.

For both agents used here, the results of training with the Bluesky plan were functionally equivalent to training by the usual Tensorforce implementations. The agent's learning algorithm was not influenced by the plan, nor was the reward scheme. Thus, we have shown how AI agents may be both trained and employed via Bluesky without the need for further specialized RL libraries.

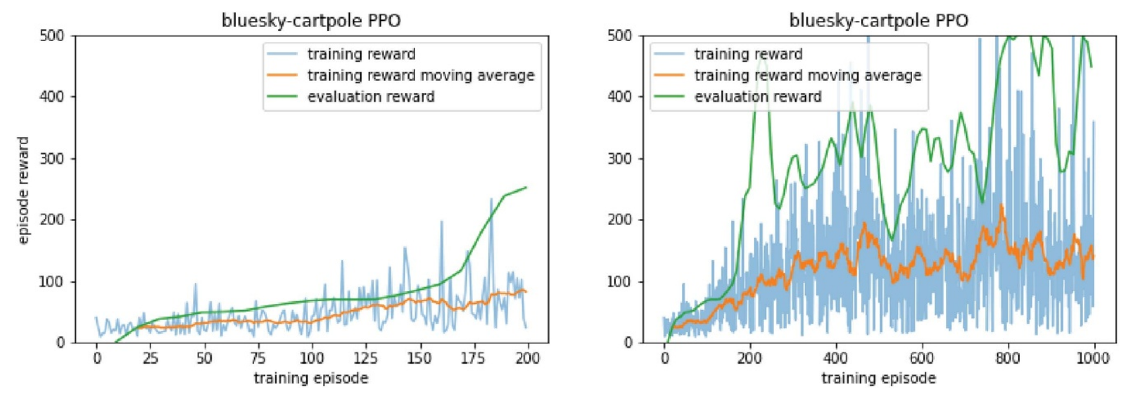

Figure 3. Plots showing the evolution of the rewards from Bluesky cartpole using the PPO algorithm from Tensorforce. The evaluation reward is the average reward of 100 episodes and is done every 10 training episodes. A select early region of training (left) of the whole 1000 episodes (right) demonstrates the slower, but more stable learning of PPO.

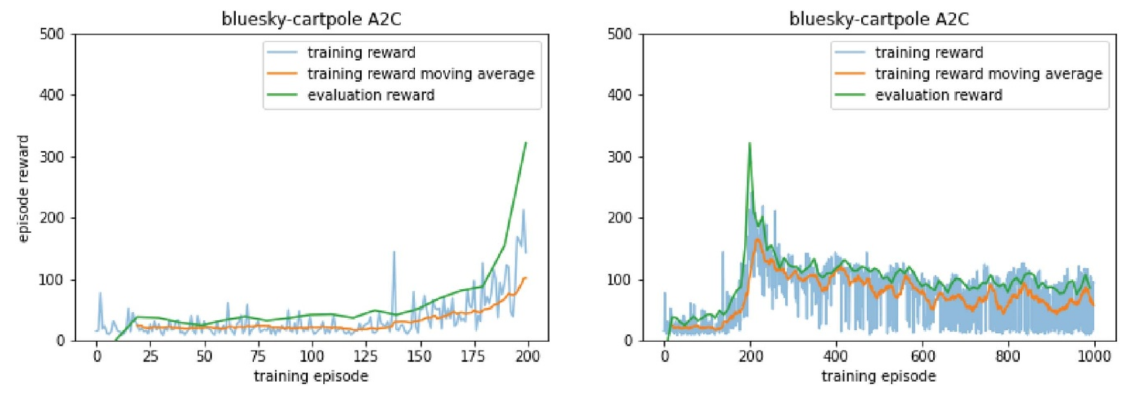

Figure 4. Plots showing the evolution of the rewards from Bluesky cartpole using the A2C algorithm from Tensorforce. The evaluation reward is the average reward of 100 episodes and is done every 10 training episodes. A select early region of training (left) of the whole 1000 episodes (right) highlights the challenge of catastrophic forgetting in some RL algorithms.

4. Reinforcement learning on a beamline

4.1. Gamifying the beamline

In the previous section we demonstrated the classic example of how a simple objective, control a cart to balance a pole, can be 'gamified' through a well described reward to successfully train an RL agent. This general approach of defining desired behavior, assigning goals, and allowing an agent to discover optimal policy, is particularly well suited to the challenges facing the operation of a beamline. For example aligning optical components to produce maximal flux onto a sample could be achieved by delivering a reward proportional to the flux measured.

To demonstrate this approach here, we will employ RL methods to address an often encountered challenge on high-throughput powder diffraction beamlines: optimizing data collection strategies across a set of samples with different scattering strengths. In this scenario, a user comes to a beamline with many (on the order of 100) different samples. All of the samples are loaded into a multi-sample holder which will sequentially place each one into the beam, allowing them to be measured for the desired exposure time. Once the sequence is complete, the multi-sample holder will return to the first sample, and the procedure can be repeated. This naive policy treats all samples as equivalent; however, it is possible that the samples are of varying overall scattering power due to differences in material composition, density, and crystallinity. These physical differences will mean that the strongly scattering 'good' samples will require significantly less overall measurement time to properly characterize than the 'bad' or weakly scattering samples. While this knowledge is not available to the user *a priori*, it is essential to the effective use of limited resources.

Without being initially aware of the sample quality, the default strategy that may be adopted is to measure each sample sequentially. Within the limits of the allocated total beamtime, the user may set this procedure to repeat, thus measuring each sample many times with the end-goal of building sufficient total statistics on each sample to be able to confidently utilize the data from each sample. However, this is not an ideal behavior.

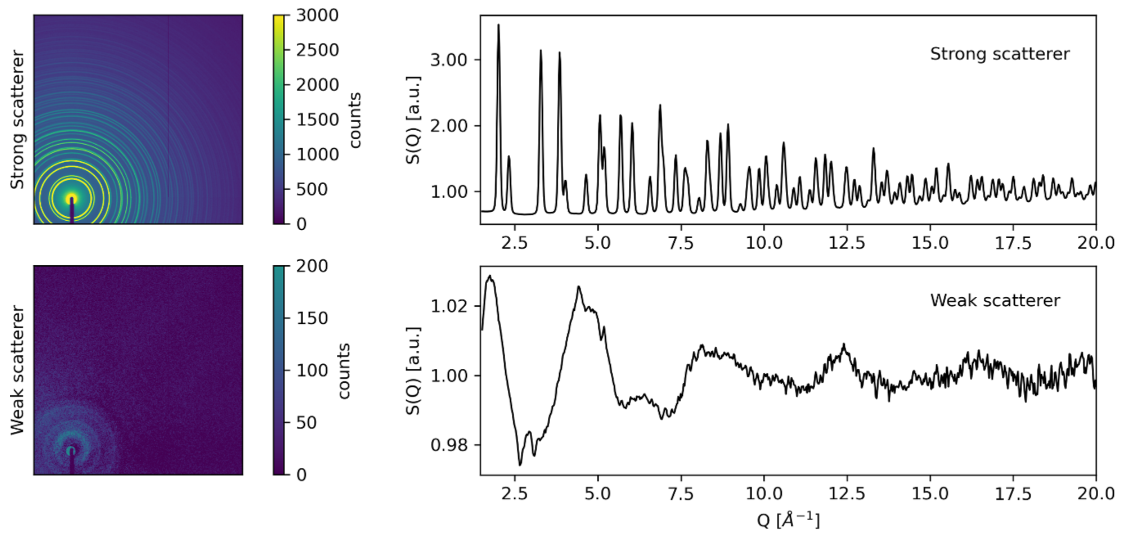

Figure 5. Example diffraction data measured on the PDF beamline at NSLS-II of ceria, a strongly scattering crystalline material (top) compared to a weakly scattering Al-based amorphous alloy sample (bottom). For each case, the measured 2D-image data is presented on the left with the as-reduced 1-D structure function, $S(Q)$, shown on the right. Note that differences between measured raw counts, as well as associated signal-to-noise, which is much more clearly defined for the strongly scattering sample. Both samples were measured under similar conditions at the beamline.

Some samples, the so-called 'bad seeds', can require significantly more measurement time to achieve the desired statistics. An omniscient collection policy would plan to measure 'bad seeds' consecutively, building up 10–20 exposures, before moving on to other samples. Well scattering 'good egg' samples would only require a single exposure to reach the required statistics. By contrast, the sequential measurement strategy will thus over-measure some samples, and depending on the total amount of beamtime available, potentially under-measure the bad seeds. What is desired is an agent which would dynamically adjust the measurement strategy based on the as-collected data, maximizing the total scientific productivity of the experiment.

4.2. The bad seed environment

We created a Tensorforce environment, referred to as 'BadSeed', to simulate the described behavior. Here, we have simplified the data-quality evaluation by assigning an abstracted scalar value to each sample which represents current uncertainty in cumulative measured data quality, such that low numbers indicate good quality, and high numbers indicate poor. In real-world application, this value could be derived from variance between successive measurements or indicators of signal-to-noise. For reference, experimental data from both good and poorly scattering materials measured under similar conditions at the PDF beamline at NSLS-II is shown in figure 5. In practice, from the initial signal-to-noise ratio, an expert would identify weak scatterers, and use a heuristic to suggest a multiplier for the number of scans (in this case 10 times), where a scan is fixed measurement time. We assume here the knowledge of these heuristics, and a mapping of a simulation time-step to a single scan.

For each episode of BadSeed, N samples are created with a random subset assigned as weak scatterers at the start of each game. This assignment comes from a first pass measurement of signal-to-noise or another measure of scattering strength, and maps to a proxy metric for data-quality by heuristics. In this case, weak scatters should be measured 10 times more than strong scatterers; however, more complex heuristics could be readily prepared. The episode begins with the beamline measuring the first-sample, which returns a data-quality metric (a scalar between 0–10) that the agent observers. The agent may decide between two actions: remain on the current sample to or move-on to the next (looping back to the first sample when it reaches the end). Successive measurements on a single sample will act to linearly decrease the data-quality metric, as attributed to the effect of averaging multiple exposures to produce better average data. The general interaction loop for BadSeed is illustrated in figure 6.

The RL agent is rewarded according to equation (1) for measuring poorly scattering samples, up to a certain limit at which point it is expected that the changes brought about by further measurements would make little statistical change to the overall data quality.

$$R = \begin{cases} 1 & \text{if countdown} > 0, \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

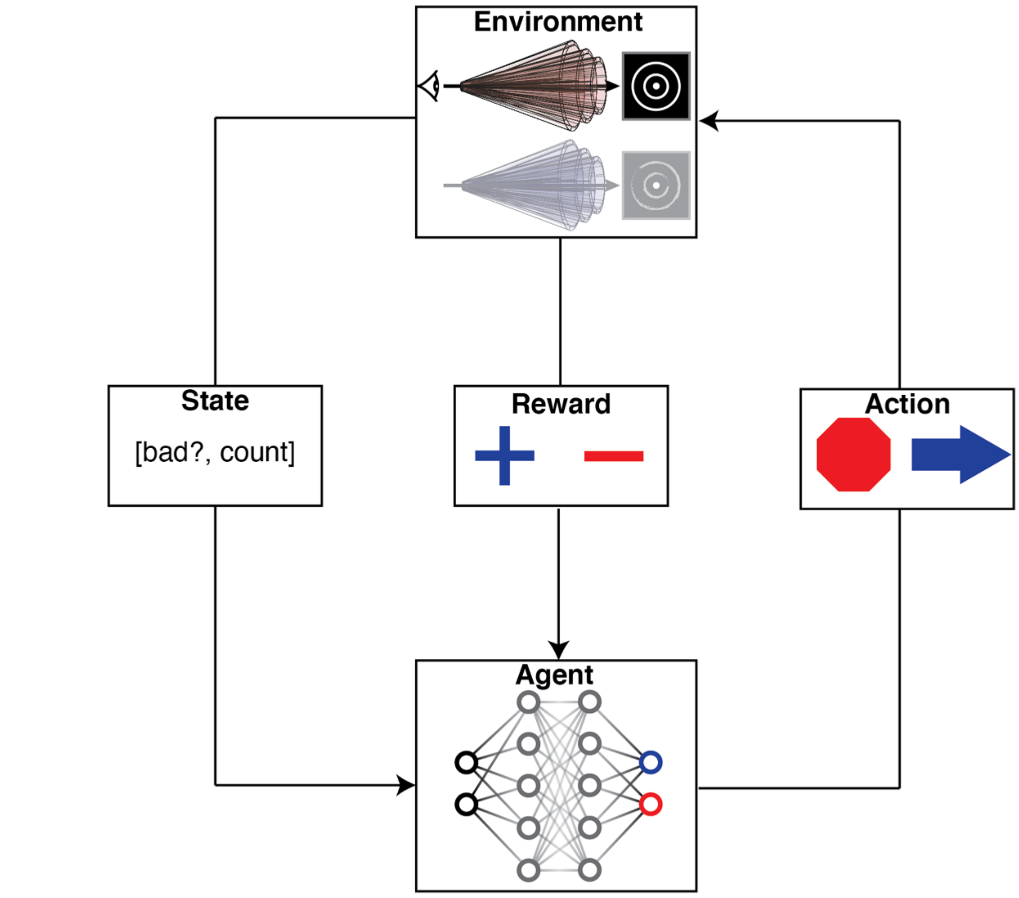

Figure 6. Schematic of the BadSeed environment, in a state where the optimal move is to move on to the next sample. This environment consists of all of the available samples and the number of times they have been measured. The state space contains a single boolean for the scattering strength, and a countdown for the number of efficacious measurement remaining on this sample. This countdown is an effective proxy for the measured data quality. The agent must decide whether to continue measuring this sample or move onto the next in the sequence. The agent is rewarded for each measurement of a weak scatterer, up to a maximum count, after which it receives no reward.

Each measurement of a weak scatterer, up to the maximum count, is awarded with one point, while the measurement of a strong scatterer or measurement of a weak scatterer beyond the maximum count is awarded no points. This abstraction allows an RL agent to develop a policy for effective measurement over many diverse pseudo-experiments, while knowing only the information about its most recent sample, and restricting its movement to the next sample in the bracket.

4.3. Solving bad seed with reinforcement learning

We solved this challenge using an A2C agent deployed using Tensorforce, with a policy network composed of two dense hidden layers with 64 nodes each, optimized using the Adam optimizer with a learning rate of 0.0003, Tensorforce defaults for other hyperparameters and a variable batch size. To test the RL agent's learning ability we trained it over 3000 episodes, where the length of each episode was the minimum time needed to earn every point. Thus, episodes with fewer poor scatterers were shorter than those with many. In the case of all samples being strong scatterers, the episode was long enough to measure each sample once, and by extension zero points could be earned. The environment was set up with 10 samples, and a maximum effective count of 10. Figure 7 shows the normalized training results as an exponential average over episodes.

The results are normalized between 0 and 100, according to the minimum and maximum possible points for each episode. The agent consistently learns a policy that produces the ideal score, and substantially exceeds the expectation value for a sequential policy given the same amount of experimental time. Note that here the average score for the ideal policy is 90, since in 10% of the episodes, no points are available (all samples are strong scatterers).

We then explored the common use case where the researcher is under time constraints, that is, they have a fixed amount of time regardless of how well each sample scatters. In this test we varied the number of weak scatterers as before; however, the amount of time allowed for the measurement was fixed. Therefore, there may not be enough time to award all available points, even with an optimal policy. Figure 8 shows the

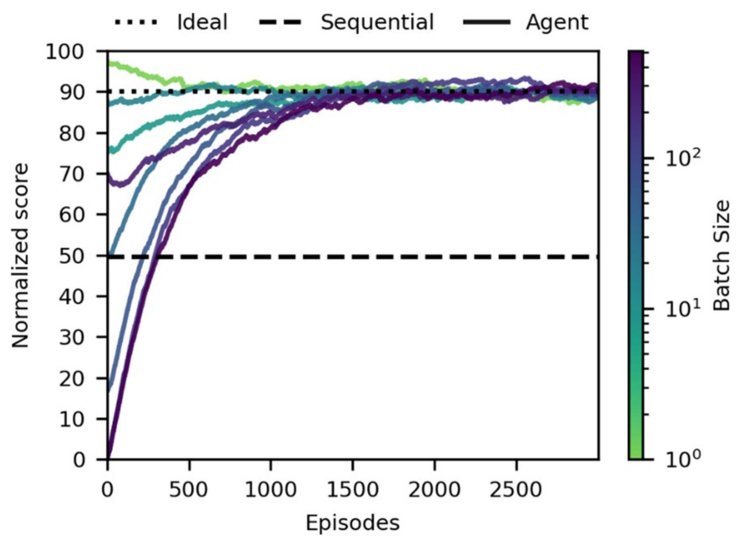

Figure 7. Learning curves of RL agent with varying batch size. The rate at which the RL agent learns an effective policy is dependent on the training batch size, i.e. the number of events it views during a training step. In all batch sizes tested, the effectiveness of the resultant policy approaches the ideal policy (dotted line). This exceeds the average score of a sequential policy (dashed line) by 80%.

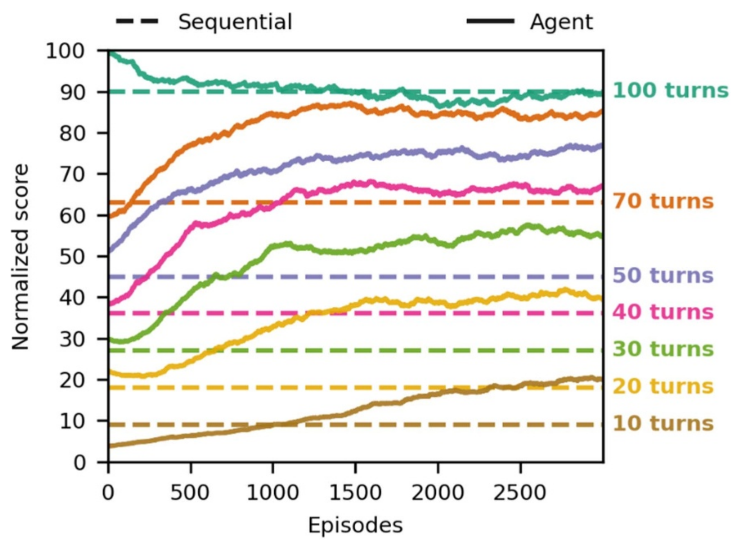

Figure 8. Learning curves of time constrained RL agent. The time on trained policy learned by the RL agent (solid line) out-performs a sequential policy (dashed line) in all cases except for when there is enough time to measure all samples in excess. The agent's guidance for effective use of resources during beamtime is a key advantage over conventional measurement strategies.

performance of the RL agent under different time constraints, with 10 samples and maximum effective count of 10 for each weak scatterer.

In the case that is not resource limited (100 time-steps for a maximum of 100 points), the sequential measurement policy performs equivalently with the RL agent, with minor variation in the RL agent due to the random initialization of weak scatterers. This is to be expected as it is the limit under which there is sufficient time to measure all samples adequately. This facile problem is also learned easily by the RL agent: as shown in the early stage of this learning curve, the agent has an excess of time to measure when there are fewer than 10 weak scatterers, and begins with time averaged score greater than the expectation value.

Under all other scenarios, the RL agent quickly learns to outperform the sequential policy, often doubling the normalized score of the default behavior. This demonstrates the RL agent is successful in focusing its efforts on the measurements that provide the scientist with the most utility given limited time. The benefit of the agent guided measurement is even shown with only 10 available measurements, offering a 100% increase on utility of the sequential policy.

5. Leveraging databroker for AI/ML

It is advantageous to store the history of these AI agents in much the same way as we want to store the collection and processing provenance [20–22] of experimental data. As stated above, we already have a solution that is in use for the experimental data at NSLS-II. In an analogous way to how we store experimental data by recording a reference to the actual detector images, we can store a pointer to each training step for a given agent. By tagging epochs, we are able to directly access any step within the training history in the same way as we would access a particular data collection measurement within a larger scan series. Any amount of information detail can be stored in this way—from simple metadata and reward performance to the full neural network weights required to reconstruct the agent at any point along the training or use history.

It is reasonable to ask why do we not just use check pointing in files? This question parallels the concerns initially raised about replacing files as a primary storage with a database. The benefit of a tool like DataBroker is that it provides a standard interface to the data without necessitating file reading and writing code embedded within the scientific application code. This means that for any particular data format the code to read and write it only has to be implemented once, rather than re-written for each application that needs to interact with the file. Additionally, as new features are adding in order to support experimental and processed data, such as integration with data analytic and management platforms with integrated Digital Object Identifier (DOI) creation and provenance capture, then these will just be automatically also available to the agent training histories we have stored. It also enables robust information schema enforcement and high-performance database capabilities like search and filtering on arbitrary criteria.

An excellent use-case was inadvertently demonstrated in the training of the A2C agent for cartpole shown in section 3. We saw in figure 4 a clear example of catastrophic forgetting, but only upon inspection of the training curve. If only the final version of the A2C agent had been stored, or this history not inspected, the 'in use' operation of this agent would have been unacceptable. However, through the use of DataBroker, not only is this issue quickly identified, but a better-performing version of the agent from episode 180 can be readily retrieved.

We note that with DataBroker, this inspection and retrieval is done in a manner that requires no specific knowledge of how the given agent has stored it is history. Both training and operational decisions can be stored and retrieved, allowing us to link the AI performance with resultant environmental data. At a beamline, this allow us to link the data from both the AI and the experiment together easily and therefore leverage the strengths of each to the benefit of the other.

6. Conclusions

As has been shown, RL shows great promise as a tool to aid in the operation and automation of beamlines at large-scale user facilities. With the added challenges originating from the COVID-19 pandemic and the associated transition from a predominantly onsite user model to one focused on remote-access for the foreseeable future, the development of AI tools such as those shown here is made even more critical. Furthermore, upon a return to normal operation modes, these tools offer continued functionality as autonomous 'best-effort' decision agents confined within a prescribed set of rules.

The novel nature of how these agents are trained, without the need for a previously labeled and well curated dataset but instead via direct interaction with an environment, is also noteworthy to beamline operations. The definitions required to create these RL agents being (a) an environment which applies the necessary rules governing the system transitions between states and (b) reward values (or points) associated with the system reaching desired states. In this way, the task can become 'gamified' in a very similar context that we as humans often seek to maximize and optimize operations and discovery.

We have illustrated this gamification with a characteristic problem of maximizing scientific output on a high-throughput beamline. We have trained a RL agent to achieve defined goals without the need to explicitly design the decision-tree logic, and demonstrated how this agent would outperform a sequential measurement policy. Although this example was only within a simulated environment, it highlights the potential of these RL methods for wider application across large-scale user facilities.

With beamline operations and data acquisition enabled through the Python-based Bluesky suite, and the growing number of widely accessible Python libraries to train RL agents, there is great potential to develop useful tools in the near future. The challenges facing development in this area will likely involve the transition from simulated environments initially used in training towards their application on real world systems. However, given the number of highly accurate forward-calculation methods that already exist (simulated diffraction and spectroscopy patterns, beam flight simulations, motor control system loops), and the inherent ability of RL agents to continuously adapt their policies to a changing environment, there is great opportunity in the future development of these capabilities.

Acknowledgments

This research used the PDF beamline of the National Synchrotron Light Source II, a U.S. Department of Energy (DOE) Office of Science User Facility operated for the DOE Office of Science by Brookhaven National Laboratory (BNL) under Contract No. DE-SC0012704 and resources of a BNL Laboratory Directed Research and Development (LDRD) projects 20-032 'Accelerating materials discovery with total scattering via machine learning'.

Code details

The source code relevant to this work is presented at https://github.com/BNL/pub-Maffettone_2021_02, with a persistent ID doi: https://doi.org/10.11578/dc.20210303.7.

ORCID iDs

Phillip M Maffettone https://orcid.org/0000-0001-7173-7972
Joshua K Lynch https://orcid.org/0000-0001-6032-841X
Thomas A Caswell https://orcid.org/0000-0003-4692-608X
Clara E Cook https://orcid.org/0000-0003-4187-273X
Stuart I Campbell https://orcid.org/0000-0001-7079-0878
Daniel Olds https://orcid.org/0000-0002-4611-4113

References

[1] Silver D *et al* 2018 *Science* **362** 1140–4
[2] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D and Riedmiller M 2013 (arXiv:1312.5602)
[3] Vinyals O *et al* 2019 Grandmaster level in StarCraft II using multi-agent reinforcement learning *Nature* **575** 350–4
[4] Howard R A 1960 Dynamic Programming and Markov Processes 1st edn (New York: Technology Press and Wiley)
[5] Kober J, Bagnell J A and Peters J 2013 *Int. J. Robot. Res.* **32** 1238–74
[6] Luong N C, Hoang D T, Gong S, Niyato D, Wang P, Liang Y C and Kim D I 2019 *IEEE Commun. Surv. Tutorials* **21** 3133–74
[7] Zhou Z, Li X and Zare R N 2017 *ACS Cent. Sci.* **3** 1337–44
[8] Benbrahim H and Franklin J A 1997 *Robot. Auton. Syst.* **22** 283–302
[9] Kourousias G, Billè F, Borghes R, Alborini A, Sala S, Alberti R and Gianoncelli A 2020 *Sci. Rep.* **10** 9990
[10] Noack M M, Yager K G, Fukuto M, Doerk G S, Li R and Sethian J A 2019 *Sci. Rep.* **9** 1–19
[11] Bruchon N, Fenu G, Gaio G, Lonza M, O'Shea F H, Pellegrino F A and Salvato E 2020 *Electronics* **9** 781
[12] Allan D, Caswell T, Campbell S and Rakitin M 2019 *Synchrot. Radiat. News* **32** 19–22
[13] Bluesky website (https://blueskyproject.io)
[14] Koerner L J, Caswell T A, Allan D B and Campbell S I 2020 *IEEE Trans. Instrum. Meas.* **69** 1698–707
[15] Thein K M M 2014 *Int. J. Sci. Eng. Technol. Res.* **3** 9478–83
[16] Bluesky adaptive source code (https://github.com/bluesky/bluesky-adaptive)
[17] Lillicrap T P, Hunt J J, Pritzel A, Heess N, Erez T, Tassa Y, Silver D and Wierstra D 2015 (arXiv:1509.02971)
[18] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J and Zaremba W 2016 (arXiv:1606.01540)
[19] Kuhnle A, Schaarschmidt M and Fricke K 2017 Tensorforce: a tensorflow library for applied reinforcement learning web page (https://github.com/tensorforce/tensorforce)
[20] Pouchard L, Juhas P, Billinge S, Wright C, Campbell S, Park G, Stavitski E and Van Dam H 2019 *Handbook on Big Data and Machine Learning in the Physical Sciences* vol 2 (Singapore: World Scientific Publishing) (https://doi.org/10.1142/11389)
[21] Park G and Pouchard L 2019 Scientific Literature Mining for Experiment Information in Materials Design *2019 New York Scientific Data Summit (NYSDS)* (New York, NY: IEEE) pp 1–4 (https://ieeexplore.ieee.org/document/8909726/)
[22] Park G, Rayz J T and Pouchard L 2020 Figure descriptive text extraction using ontological representation *Florida Artificial Intelligence Research Society Conf.* (https://aaai.org/ocs/index.php/FLAIRS/FLAIRS20/paper/view/18418)