Review Article

Sample-efficient reinforcement learning for CERN accelerator control

Verena Kain,* Simon Hirlander, Brennan Goddard, Francesco Maria Velotti, and Giovanni Zevi Della Porta
CERN, 1211 Geneva 23, Switzerland

Niky Bruchon
University of Trieste, Piazzale Europa, 1, 34127 Trieste TS, Italy

Gianluca Valentino
University of Malta, Msida, MSD 2080, Malta

(Received 7 July 2020; accepted 9 November 2020; published 1 December 2020)

Numerical optimization algorithms are already established tools to increase and stabilize the performance of particle accelerators. These algorithms have many advantages, are available out of the box, and can be adapted to a wide range of optimization problems in accelerator operation. The next boost in efficiency is expected to come from reinforcement learning algorithms that learn the optimal policy for a certain control problem and hence, once trained, can do without the time-consuming exploration phase needed for numerical optimizers. To investigate this approach, continuous model-free reinforcement learning with up to 16 degrees of freedom was developed and successfully tested at various facilities at CERN. The approach and algorithms used are discussed and the results obtained for trajectory steering at the AWAKE electron line and LINAC4 are presented. The necessary next steps, such as uncertainty aware model-based approaches, and the potential for future applications at particle accelerators are addressed.

DOI: 10.1103/PhysRevAccelBeams.23.124801

I. INTRODUCTION AND MOTIVATION

The CERN accelerator complex consists of various normal conducting as well as super-conducting linear and circular accelerators using conventional as well as advanced acceleration techniques, see Fig. 1. Depending on their size, each accelerator can have hundreds or thousands of tuneable parameters. To deal with the resulting complexity and allow for efficient operability, a modular control system is generally in place, where low level hardware parameters are combined into higher level accelerator physics parameters. The physics-to-hardware translation is stored in databases and software rules, such that simulation results can easily be transferred to the accelerator control room [1]. For correction and tuning, low-level feedback systems are available, together with high-level physics algorithms to correct beam parameters based on observables from instrumentation. With this hierarchical approach large facilities like the LHC, which comprises about 1700 magnet power supplies alone, can be exploited efficiently.

There are still many processes at CERN's accelerators, however, that require additional control functionality. In the lower energy accelerators, models are often not available online or cannot be inverted to be used in algorithms. Newly commissioned accelerators also require specific tools for tuning and performance optimization. Some systems have intrinsic drift which requires frequent retuning. Sometimes instrumentation that could be used as input for model-based correction is simply lacking. Examples include trajectory or matching optimization in space-charge dominated LINACs, optimization of electron cooling without electron beam diagnostics, setting up of multiturn injection in 6 phase-space dimensions without turn-by-turn beam position measurement, optimization of phase-space folding with octupoles for higher efficiency slow extraction and optimization of longitudinal emittance blow-up with intensity effects. In recent years numerical optimizers, sometimes combined with machine learning techniques, have led to many improvements and successful implementations in some of these areas, from automated alignment of various devices with beam to optimising different parameters in FELs, see for example [2–7].

For a certain class of optimization problems, the methods of reinforcement learning (RL) can further boost efficiency. With RL the exploration time that numerical optimizers

*verena.kain@cern.ch

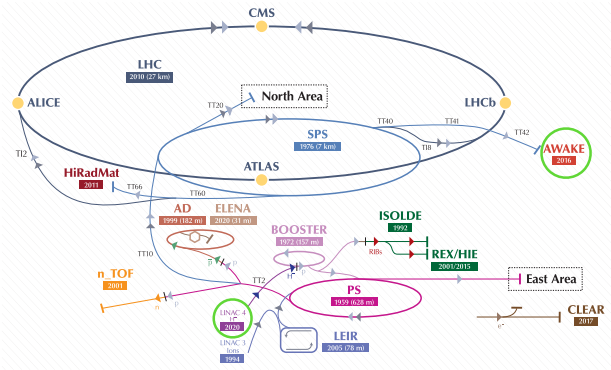

FIG. 1. The CERN accelerator complex. Reinforcement learning agents were used on the accelerators highlighted with green circles.

inevitably need at every deployment is reduced to a minimum—to one iteration in the best case. RL algorithms learn the model underlying the optimization problem, either explicitly or implicitly, while solving the control problem. Unlike optimizers, however, an additional input is required in the form of an observation defining the so-called system *state*. Numerical optimizers only need the objective function to be accessible and the algorithm finds the action a that minimizes (or maximizes) the objective function $f(a)$.

Adequate instrumentation to acquire meaningful state information often limits the applicability of RL methods. Another limitation arises from what is referred to as *sample efficiency*—the number of iterations (i.e., interactions with the accelerator) required until the problem is learned. For these reasons, not all RL algorithms are suitable for the control room, and not all problems are suitable for its use.

Nevertheless, given the performance advantages compared to numerical optimizers, investigation of RL for certain accelerator control problems is important and many laboratories have studied RL algorithms already for various control problems, albeit mostly in simulation only [8–11]. In 2019 most of the CERN accelerators were in shutdown to be upgraded as part of the LHC Injector Upgrade project [12]. The new linear accelerator LINAC4 and the proton-driven plasma wakefield test facility AWAKE [13] were, however, operated for part of the year. Both accelerators were used for various tests of advanced algorithms such as Scheinker's ES algorithm, see for example [14]. Taking advantage of recent rapid developments in deep machine learning and RL algorithms, the authors successfully implemented sample-efficient RL for CERN accelerator parameter control and demonstrated its use online for trajectory correction both at the AWAKE facility and at LINAC4.

This paper is organized as follows. A brief introduction is given to reinforcement learning, in the domain of accelerator control. The main concepts and challenges are described, with the necessary details of the specific algorithms used. The deployment framework is explained, including the interface standards chosen. In the experimental section, the problem statements and results of the tests on trajectory correction are given, both for the AWAKE 18 MeV electron beamline and the 160 MeV LINAC4. In the discussion, the results obtained are examined in the context of the main problems encountered and the experience obtained. The next steps and potential for wider application are covered in the outlook.

II. REINFORCEMENT LEARNING FOR ACCELERATOR CONTROL

The optimization and control of particle accelerators is a sequential decision making problem, which can be solved using RL if meaningful state information is available. In the RL paradigm, Fig. 2, a software agent interacts with an environment, acquiring the state and deciding actions to move from one state to another, in order to maximize a cumulative reward [15]. The agent decides which action to take given the current state by following a policy. The environment can be formally described by a Markov decision process, and the states exhibit the Markov property. This means that the state transition probability to each future state s' only depends on the current state s and the applied action a and not on any preceding states. (For orbit correction—where the orbit reading is the state s, the orbit s' is defined by the current orbit s and the delta dipole corrector settings a, also if it is corrected iteratively.)

The action value function or simply Q-function, denoted $Q^\pi(s, a|\theta^Q)$, in deep RL is a measure of the overall expected reward assuming the agent in state s performs action a and then continues until the end of the episode following policy π. It is defined as:

$$Q^\pi(s, a|\theta^Q) = \mathbb{E}_\pi \sum_{k=0}^{N} \gamma^k r_{i+k+1}|s_i = s, \qquad a_i = a \quad (1)$$

where N is the number of states from state $s = s_i$ till the terminal state, γ is a discount factor and r is the reward the

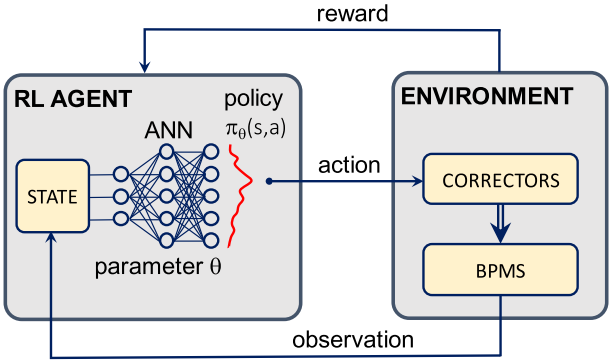

FIG. 2. The RL paradigm as applied to particle accelerator control, showing the example of trajectory correction.

agent receives after performing action a in state s during iteration i. θ^Q are the network parameters.

The value or V-function, denoted $V^\pi(s|\theta^V)$, is related to the Q-function. It measures the overall expected reward associated with state s, assuming the agent continues until the end of the episode following policy π. It is defined as:

$$V^\pi(s|\theta^V) = \mathbb{E}_\pi \sum_{k=0}^{N} \gamma^k r_{i+k+1} | s_i = s \tag{2}$$

A. Classes of reinforcement learning algorithms

Reinforcement learning algorithms can be divided into two main classes: model-free and model-based. Model-free control assumes no *a priori* model of the environment and can be further subdivided. Policy optimization methods consist of policy gradient or actor-critic methods which are more suitable for continuous action spaces. These methods attempt to learn the policy directly, typically using gradient descent. Q-learning methods seek to find the best action to take given the current state, i.e., the action that maximizes the Q-function, and are specifically suitable for discrete action spaces. For these, an epsilon-greedy policy may be used during training, which greedily takes the action leading to the highest reward most of the time (exploitation) and a random action the rest of the time (exploration).

In model-based methods, the control method uses a predictive model of the environment dynamics to guide the choice of next action. Establishing a reliable enough model is clearly critical to success. Several subdivisions of this class of RL algorithms exist, including analytic gradient computation methods such as LQR [16], sampling-based planning [17], and model-based data generation methods such as Sutton's original Dyna [18] and related algorithms using model ensembles to capture the uncertainty of the model and avoid model bias. An overview of the various algorithms is provided in [19]. Model-based algorithms tend to be more complex due to the additional stage of model learning before and while training agents. Despite the advantage of better sample-efficiency for model-based RL, the authors only used model-free algorithms during this first study with the goal to demonstrate that RL is suitable for accelerator control problems.

For most real-world applications, where the state-space is large, the Q-function or the policy need to be approximated using, e.g., neural networks. Advances in the use of deep learning to train such models in the RL paradigm have led to a range of algorithms such as deep Q-network (DQN) [20], deep deterministic policy gradient (DDPG) [21], and normalized advantage function (NAF) [22].

B. Sample-efficient RL agent using Q-learning

One of the key issues in all RL approaches is sample efficiency—the number of iterations required to learn either the optimum policy, Q- or value function. Within the class of model-free algorithms, policy gradient algorithms are generally less sample-efficient than Q-learning ones and cannot take advantage of experience replay [15].

For accelerator applications a continuous action space is usually required. This can be a serious limitation for Q-learning, due to the need to perform the nontrivial maximization of the Q-function with respect to continuous a. It can however be overcome by assuming a specific algebraic form of the Q-function, such that $Q(a, s)$ is straightforward to optimize with respect to the action. This approach is applied in the normalized advantage function (NAF) algorithm. It assumes a quadratic dependence of Q on the action a (θ are the network parameters to be fitted):

$$Q(s, a|\theta^Q) = A(s, a|\theta^A) + V(s|\theta^V), \tag{3}$$

$$A(s, a|\theta^A) = -\frac{1}{2}(a - \mu(s|\theta^\mu))^T P(s, \theta^P)(a - \mu(s|\theta^\mu)) \tag{4}$$

$A(s, a|\theta^A)$ is the advantage function, $V(s|\theta^V)$ the value function and $P(s|\theta^P)$ a state-dependent, positive-definite square matrix; $a^* = \mu(s|\theta^\mu)$ corresponds to the action that maximizes the Q function.

Assuming this specific representation of Q of course limits the representational power of the algorithm, but many accelerator optimization problems are convex and for these cases a quadratic Q function can be a reasonable assumption. Another sample-efficient algorithm without the limited representational power that was tested recently at CERN's accelerators is the DDPG variant TD3 [23]. Its performance is comparable to NAF for the same problems. A comparison of NAF in terms of sample efficiency to other algorithms such as DDPG can be found in [24]. Comparative plots of the reward evolution during training for the AWAKE trajectory correction problem in simulation for the NAF algorithm, the on-policy algorithm PPO as well as TD3 are given in the Appendix A.

The results in this paper were all obtained with the very sample-efficient NAF algorithm.

C. NAF network architecture

In practice the NAF agent is a neural network with an architecture as sketched in Fig. 3. The input layer receives s, the state, followed by two fully-connected dense hidden layers, typically of 32 nodes for our cases. The outputs of

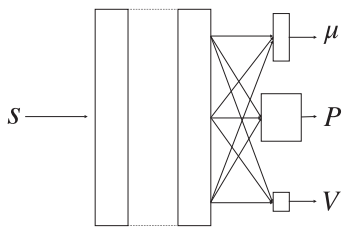

FIG. 3. The NAF network architecture.

the network are $\mu(s,a|\theta^{\mu})$, $P(s,a|\theta^{P})$, $V(s|\theta^{V})$. More details are given in Appendix B 1.

The loss function used in the network training is the temporal difference error L between the predicted Q value and target y_i when the agent takes an action moving to state s_i:

$$L = (y_i - Q(s_i, a_i|\theta^{Q}))^2 \tag{5}$$

where $y_i = r_i + \gamma V'(s_i|\theta^{Q'})$ with γ the discount factor and V' the value function belonging to the target network Q'. (The target network is a copy of the main network, but it is only infrequently updated to slowly track the main network and avoid value function divergence.)

To improve training stability, prioritized experience sampling [25] was implemented in addition. The implementation of the algorithm as used in the control room is available as PER-NAF [26]. Benchmark results for prioritized experience replay and other details are provided in Appendix B.

D. Operational deployment

The CERN accelerator control system offers a PYTHON package (PYJAPC [27]) to communicate directly with the hardware systems or with the high-level control system for parameters like deflection angle or tune.

For the problem description (environment), the generic OpenAI Gym framework [28] was chosen, to enforce standardization and allow easy switching between different control problems or agents. The OpenAI Gym environment python class provides an interface between RL agent and optimization problem and is used by many available RL algorithm implementations. Gym environments contain all the control problem specific code—the interaction with the machine or simulation for setting the action, reading or calculating observation data as well as calculating or measuring reward. PER-NAF was implemented to be compatible with OpenAI Gym.

As the various functions (e.g., advantage function, value function) are approximated through neural networks, the observations, actions and the reward need to be normalized to match the practical range for the accelerator settings, avoid issues with vanishing gradients and allow the network training to converge. Wrappers extending the OpenAI Gym environment class help with this task.

The training of an RL agent is done in episodes. The maximum length of the episode is a parameter of the environment and corresponds to the maximum number of interactions (steps) the agent may take to iteratively solve the problem, and therefore reach the maximum cumulative reward $\sum r$ or *return*. It is a key hyperparameter for efficient training. For our cases, 50 iterations was a good maximum episode length.

For each iteration during an episode, the agent expects a state transition from state s_i to state s_{i+1}. For the accelerator context, this means the change of setting corresponding to a_i needs to be added to the existing setting that defined s_i. The implementation of this logic needs to be provided in the method *step*(a_i) of the environment as illustrated in the pseudo-code below (k_i refers to the scaled action):

Algorithm 1. add a_i to current setting in method *step*(a_i) for iteration i

```
rescale a_i to Δk_i;
get current setting k_i;
set new setting k_{i+1} = k_i + Δk_i;
collect reward r_{i+1} and observe s_{i+1};
```

Instead of using epsilon-greedy exploration, exploration with PER-NAF is managed by adding Gaussian noise to the proposed action at each iteration and reducing the noise level by 1/episode number in the course of the training. For implementation details of the action noise see Appendix B 3.

The training is significantly accelerated by defining a target cumulative reward. If reached, the problem is considered to be sufficiently solved and the episodes are *finalized*. After finalising an episode the *reset*() method of the environment is called, which randomly configures another starting situation for the agent to solve and a new episode starts. This target needs to be set high enough for useful training to occur, but low enough for the agent to be able to reach it with sufficient exploration. Careful choice of reward target proved to be one of the critical factors in effective performance. (Increasing the target throughout the training was tried as well, but did not improve the results for our environments significantly).

Other reasons for the episodes to be finalized in our case were violating hardware constraints, causing the beam to be lost or going beyond machine protection watchdog limits.

III. PROOF-OF-PRINCIPLE APPLICATION OF RL AGENT TO AWAKE TRAJECTORY CORRECTION

The AWAKE electron source and line are particularly interesting for proof-of-concept tests for various algorithms due to the high repetition rate, various types of beam instrumentation and insignificant damage potential in case of losing the beam at accelerator components. The first RL agents were trained for trajectory correction on the AWAKE electron line with the goal that the trained agents correct the line with a similar efficiency as the response matrix based SVD algorithm that is usually used in the control room, i.e., correction to a similar RMS as SVD within ideally 1 iteration. Figure 4 shows an example of a correction using the SVD implementation as available in the control room.

The AWAKE electrons are generated in a 5 MV RF gun, accelerated to 18 MeV and then transported through a beam line of 12 m to the AWAKE plasma cell. A vertical step of 1 m and a 60° bend bring the electron beam parallel to the proton beam shortly before the plasma cell. The trajectory

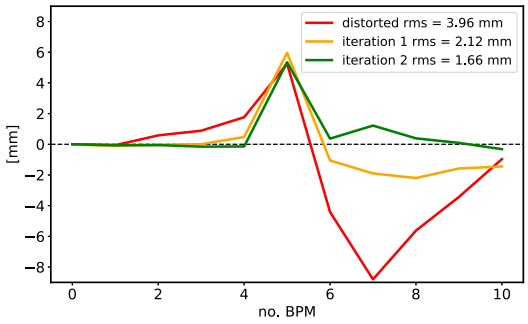

FIG. 4. Results of an SVD correction in the horizontal plane for the AWAKE electron line. The initial distorted trajectory was obtained with the *reset()* method of the OpenAI Gym environment. It took 2 iterations to correct below 2 mm RMS, which is the target cumulative reward of the Gym environment. It cannot correct for BPM 5, as it is at a location of peak dispersion. Its reading depends on the energy distribution coming out of the source, which can vary from day to day. Also, the RL algorithms cannot correct for it if it has only dipole correctors as actions. The target cumulative reward defined for the AWAKE OpenAI Gym environment needs to take this into account.

is controlled with 11 horizontal and 11 vertical steering dipoles according to the measurements of 11 beam position monitors (BPMs). The BPM electronic read out is at 10 Hz and acquisition through the CERN middleware at 1 Hz.

For reference, numerical optimizers had been tested as well. For example the algorithm ES took about 30 iterations to converge for the AWAKE trajectory correction problem [14]. Figure 5 shows the example results with the numerical optimization algorithm COBYLA [29] on simulation. It took more than 35 iterations with the used setup, but it would have reached the threshold (red line in Fig. 5) as defined in the OpenAI Gym environment after about 20 iterations.

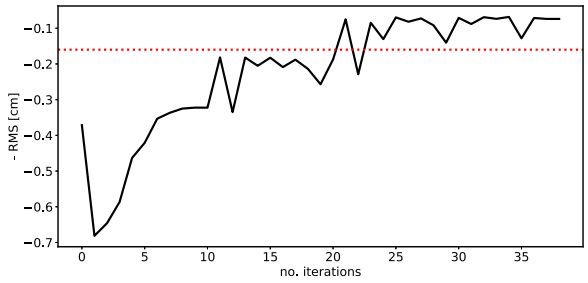

FIG. 5. Results of numerical optimization with the COBYLA algorithm on the simulated OpenAI Gym environment for AWAKE steering. With the hyperparameters used for launching the algorithm, it took 37 iterations to converge at an RMS of 0.5 mm. It would however have reached the OpenAI Gym environment threshold (indicated in red) already after 21 iterations. The threshold in the OpenAI Gym environment was set to 1.6 mm RMS. The results on the y-axis are shown in the same units as for the plots on the RL results below.

A. Simulated and online OpenAI Gym AWAKE environments for trajectory steering

The electron transfer line with all of its equipment is modeled in MAD-X [30]. The transfer functions from field to current for the various magnets are available in the control system and the line is operated in normalized strength (i.e., angle for the steering dipoles). Typical trajectory corrections are in the order of 1 mrad per corrector. The MAD-X model allows to calculate the response matrix as expected change in BPM readings for a given change in corrector settings. The calculated response matrix was used to prepare a simulated OpenAI Gym environment with the purpose to test various RL algorithms offline and define the hyperparameters for the chosen algorithm for optimum sample efficiency.

To most profit from the simulated environment, observations (the BPM readings in difference from a reference trajectory) for the RL agent and actions from the agent (the corrector delta settings) need to have the same units and normalization as in the OpenAI Gym environment connected to the real equipment. Also, the actions in the environment connected to the real hardware were applied as angle through the high level settings database instead of current directly on the power supplies to ensure compatibility with the simulated environment.

The online environment connected to the AWAKE equipment also had to deal with measurement artefacts. In case the beam is lost in the line, the subsequent BPMs return zero as reading, which is however within the possible observation space. Exact zero readings were therefore overwritten by a trajectory reading corresponding to the aperture of the line at the particular BPM location (i.e., 17 mm). Due to shot-by-shot momentum variations from the electron source, the observations were averaged over 5 acquisitions. The reward was defined as the negative RMS value of the difference trajectory between measured and reference one. The reward was normalized to be typically between -1 and 0, but certainly to not go beyond 0. This is to ensure that the agent learns to use the fewest iterations to solve the problem.

B. Experiment results from AWAKE RL tests

The first successful online training of a NAF agent on trajectory steering in the horizontal plane was obtained on November 22, 2019. The training for 11 degrees of freedom (DOF) took roughly 30 minutes, corresponding to about 350 iterations. At each start of an episode the correctors were reset to the initial setting before the training. A random Δ setting was then sampled from a Gaussian distribution with $\sigma = 300$ μrad for each corrector and added to the initial setting, leading to maximum 7 mm RMS (a factor 2-3 above the normal trajectory distortions caused by drifts and different initial conditions). The maximum step a corrector could do per iteration was set ± 300 μrad. The main training parameters are summarized in Table I.

TABLE I. AWAKE horizontal steering NAF agent training parameters.

DOF	11
Reward target [cm]	−0.2
Max episode length	50
Max Δ_{corr} [μrad]	300
Min allowed reward [cm]	−1.2

The objective of the training was twofold: to maximize the reward from each initial condition, and to maximize the reward in the shortest possible time. Figure 6 shows the evolution of the 200 episode online training. The upper plot gives the length of the episodes in number of iterations as training evolves, while the lower plot shows the initial reward (i.e., negative RMS) at the beginning of the episode (green line) as well as the final reward achieved (blue line) at the end of each episode. For a successful termination of the episode, the final reward had to be above the target (dashed red line).

At the beginning of the training, the agent could not correct the line to an RMS below 2 mm, despite many iterations. It even further deteriorated the trajectory. After about 15 episodes it had learned to successfully correct the trajectory within 1-2 iterations to mostly even below 1 mm RMS starting from any initial condition. Figure 7 shows the evolution of the value function $V(s)$ and the loss function for the network training as a function of iterations. The value function started to stabilize after about 90 iterations (equivalent to the 15 episodes when successful correction was observed), continued to improve for another 100 iterations and finally converged to −0.05 corresponding to 0.5 mm RMS after correction in case of only one iteration required. After the online training where exploration noise is still present (albeit decaying very rapidly with increasing episode number), the agent was tested in an operational configuration. No noise is added to the actions

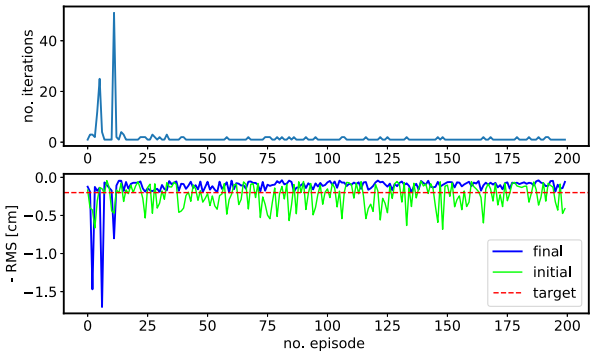

FIG. 6. Online training of NAF agent of AWAKE electron line trajectory steering in the horizontal plane. In the upper plot the number of iterations per episode is given. The lower plot shows the initial and final negative RMS value for each episode. The target negative RMS value is indicated in red.

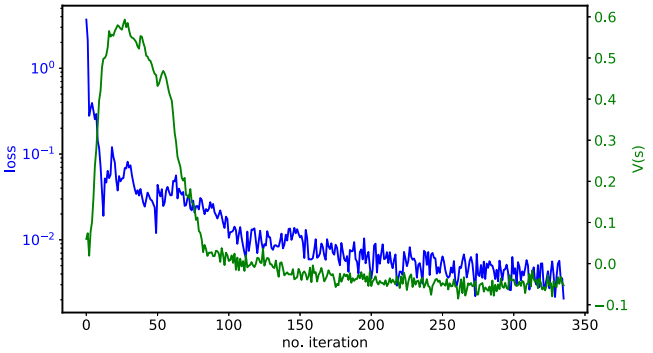

FIG. 7. Evolution of loss and value function during training on November 22, 2019.

predicted by the trained agent in this case. The agent was presented with randomly sampled observations (by invoking the *reset()* method of the environment) and it had to correct the line accordingly. Figure 8 shows the validation run with 24 episodes. The plot is arranged as for the training results above, with the upper plot showing the number of iterations per episodes and the lower one, the initial and final negative RMS per episode. The trained agent required 1 or 2 iterations to correct the trajectory to better than the target (it requires more than 1 iteration from time to time as its maximum step per iteration is limited to 300 μrad). A longer validation run was carried out beginning of June 2020 with an agent that had only been trained for 35 episodes. The results of this validation can be found in Fig. 9.

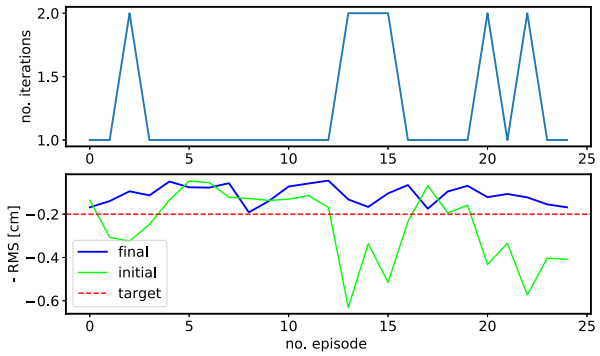

FIG. 8. Short validation run after training with 200 episodes on November 22, 2019. The agent corrects the trajectory to better than 2 mm RMS target within 1-2 iterations. (The initial trajectories were established by randomly applying corrector settings, which can lead to initial trajectories already above the target. In the test setup no check on initial RMS was used before calling the agent and it would therefore correct trajectories already above target generalizing from the training earlier, with few trajectories in this region as episodes would be finalized at that stage. All results are above target, but trajectories with initially very good RMS were sometimes slightly deteriorated because of this test setup artefact. The test setup was improved for the next validations.)

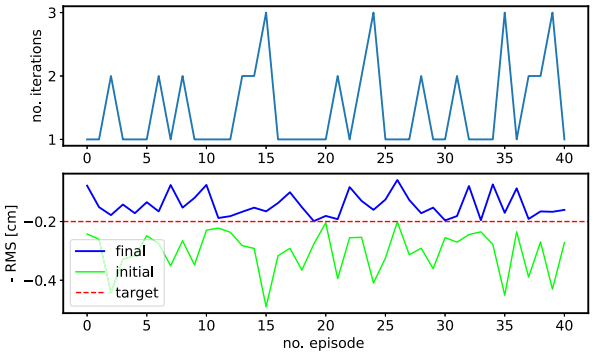

FIG. 9. Validation right after short training of 35 episodes on June 8, 2020. Despite the short training, the agent corrects the trajectory to better than the 2 mm RMS target within 1-3 iterations.

An important question is also how long a training remains valid and how often an RL agent would have to be re-trained. Obviously this depends on the specific problem and how much state information is hidden. In our case the expectation was that if the agent is trained once, it will not need any re-training unless the lattice of the line is changed. To verify this expectation, a NAF agent trained on June 10, 2020, was successfully re-validated on September 22, 2020, without any additional training. The validation results are shown in Fig. 10.

Another important application of RL will result from training RL agents on simulation and then exploit the agent with or without additional short training on the accelerator. The obvious advantage of this approach, if possible, is that in this case the algorithm does not have to be restricted to be a very sample-efficient one, as accelerator time for training is either zero or limited. To test this principle of offline training, another NAF agent was trained—this time on the simulated AWAKE trajectory correction environment. This agent was then used on the accelerator in operational configuration as trajectory correction algorithm.

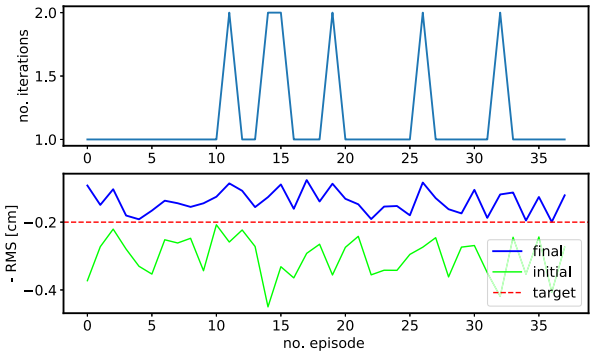

FIG. 10. Validation on accelerator on September 22, 2020, of agent that was trained more than three months earlier (June 10, 2020). The agent corrects the trajectory to better than 2 mm RMS within 1-2 iterations. No retraining was required.

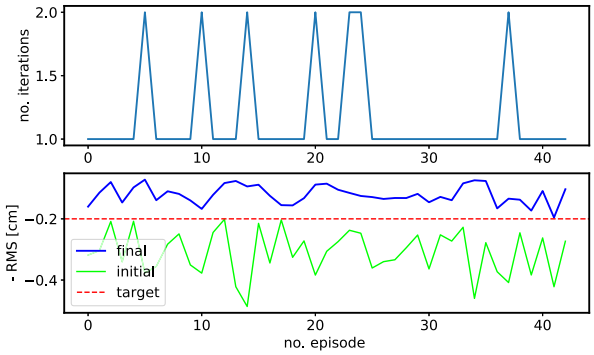

FIG. 11. Validation on accelerator of agent that was trained on simulation. The agent corrects the trajectory to better than 2 mm RMS within 1-2 iterations.

As expected, the results—maximum 2 iterations for correction to well below 2 mm RMS for each episode—were as good as with the online trained agent. Figure 11 shows the results.

IV. RL AGENT FOR LINAC4 TRAJECTORY CORRECTION

Training the AWAKE RL agent for trajectory correction was a test case for algorithm development, since classical optics model-based steering algorithms are available for the AWAKE 18 MeV beamline. The CERN LINACs, on the other hand, do not have online models, since the approach for defining models in the control system developed for transfer lines and the synchrotrons does not fit LINACSs without adaptation. Beam parameters are usually tuned manually, guided by offline calculations and experience. RL and numerical optimization could be obvious and inexpensive solutions to many typical LINAC tuning problems. The CERN accelerator complex comprises two LINACs. LINAC3 is used for a variety of ions and the 160 MeV LINAC4 will provide H^- to the upgraded CERN proton chain through charge exchange injection into the PS Booster [31]. LINAC4 had its final commissioning run at the end of 2019, where some time was also allocated to test various advanced algorithms for different control problems. Figure 12 shows an example of numerical opimization with the algorithm COBYLA for trajectory correction in LINAC4. Also, an RL agent using the NAF algorithm was trained for trajectory steering in the LINAC exploiting the experience with AWAKE.

LINAC4 accelerates H^- from 3 MeV after source and RFQ to 160 MeV. The medium energy beam transport (MEBT) after the RFQ is followed by a conventional drift tube linac (DTL) of about 20 m that accelerates the ions to 50 MeV, then to 100 MeV in 23 m by a cell-coupled drift tube LINAC (CCDTL) and finally to 160 MeV by a π-mode structure (PIMS). The total length of the LINAC up to the start of the transfer line to the PSB is roughly 75 m. The pulse repetition rate is 0.83 Hz. The trajectory in

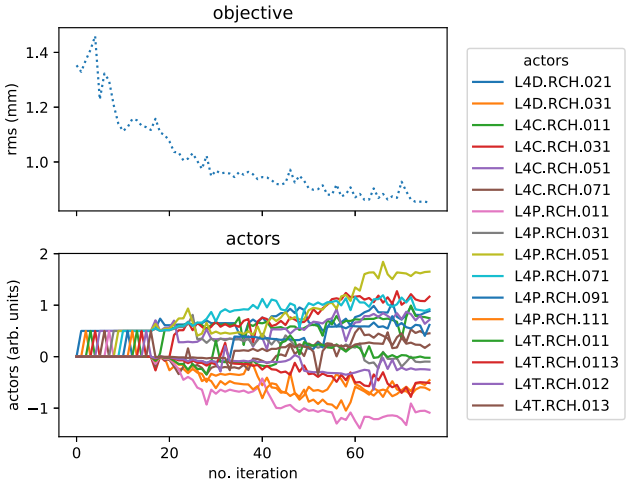

FIG. 12. Optimization with COBYLA of the LINAC4 trajectory correction problem in the horizontal plane. The algorithm converged after around 70 iterations.

the MEBT is fine tuned for optimizing chopping efficiency and should not be modified during general trajectory optimization. In addition there are no BPMs available in the MEBT as observable for an RL agent.

A. Online OpenAI Gym LINAC4 environment

The LINAC4 Gym environment comprised state information from 17 BPMs and actions possible on 16 correctors, through DTL, CCDTL, PIMS and start of the transfer line in the horizontal plane (the allocated accelerator development time was not sufficient to also train for the vertical plane). No simulated environment was available for this case and the tuning of the hyperparameters had to be carried out online. The hyperparameters obtained earlier with AWAKE were not directly re-usable as the allowed trajectory excursion range was much reduced due to machine protection reasons (e.g., the maximum allowed RMS had to be set to 3 mm, which affected the normalization of the returned reward). The LINAC4 trajectory steering OpenAI Gym environment had to respect the machine protection constraints and finalize episodes in case of violation, reset to safe settings as well as to deal with various hardware limitations (e.g., the power supplies the steering dipoles cannot regulate for $|I| < 0.1$ A).

B. Experimental results from LINAC4 RL tests

LINAC4 had 8 weeks of final commissioning run in 2019. On November 27, half a day was allocated to training and testing the NAF agent. A big fraction of this time was used in fine tuning hyperparameters such that the agent would not immediately run into rather tight machine protection limits during the exploration phase. A successful training could be achieved, with the agent training parameters given in Table II. The training is shown in Fig. 13

TABLE II. LINAC4 horizontal steering NAF agent training parameters.

DOF	16
Reward target [mm]	−1
Max episode length	15
Max Δ_{corr} [A]	0.5
Min allowed reward [mm]	−3

and the convergence of the value function $V(s)$ and loss function in Fig. 14. The total number of episodes was set to 90 (taking in total about 300 iterations).

After about 25 episodes (or the equivalent of about 125 iterations), the agent had learned to correct the trajectory to below 1 mm RMS within a maximum of 3 iterations each time. The value function converged to -0.85 corresponding to 0.85 mm RMS in case of correction in one iteration. No other tests could be performed at that stage due to lack of time. It would obviously be of interest to deploy the

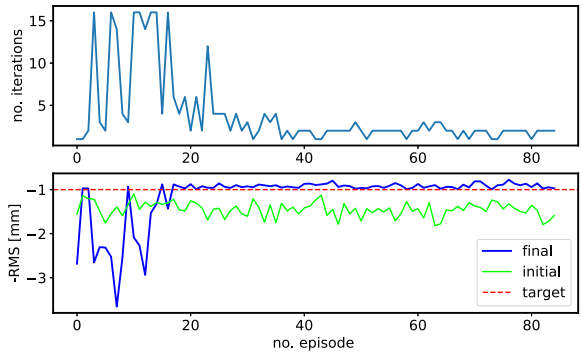

FIG. 13. Online training of NAF agent on LINAC4 trajectory steering in horizontal plane. The maximum allowable RMS was limited to 3 mm due to machine protection reasons. The target for the training was set to reach 1 mm RMS.

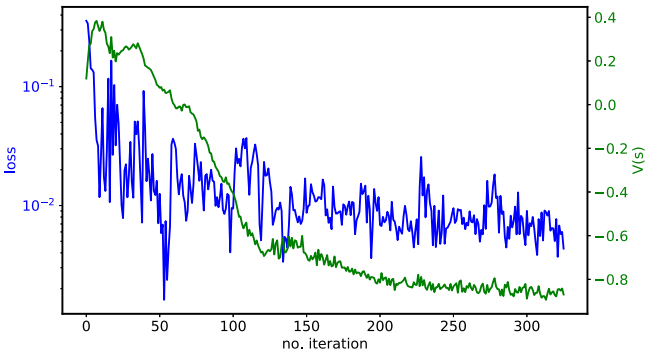

FIG. 14. Online training of NAF agent on LINAC4 trajectory steering in horizontal plane: convergence of loss function and value function V. The latter converges to −0.85 corresponding to an RMS better than 1 mm in case of correction in one iteration.

trained agent as part of the standard LINAC4 control room tools for the 2020 run and to test long term stability.

V. DISCUSSION AND OUTLOOK

The experience with the AWAKE and LINAC4 RL agent deployment has proved that the question of sample efficiency for the model-free RL approach can be addressed for real accelerator control problems. The training with algorithms such as NAF (and TD3) is sample-efficient enough to allow for deployment in the control room. It requires more iterations than a numerical optimization algorithm, but after training it outperforms numerical optimizers. The resulting product is a control algorithm like SVD. Whereas the linear control problem of trajectory correction can be solved with SVD or other standard methods, single-step correction for nonlinear problems are not available with these standard algorithms. RL does not require a linear response and can thus provide controllers for also these cases.

The standardization of the environment description using OpenAI Gym proved a big advantage, allowing rapid switching between simulated and online training, and between agents. Correct normalization and un-normalization of all actions, state observations and rewards was of course crucial. For the training, care needed to be taken in constructing the reward function and episode termination criteria. The key hyperparameters were found to be the maximum number of iterations per episode, and the reward target for early episode termination. Note that these hyperparameters belong to the environment description and not to algorithms themselves. The same environment hyperparameters can be used with different RL algorithms e.g., NAF and TD3. Tuning of these hyperparameters took some time, and valuable experience was gained from having a simulated AWAKE environment, although such a simulated environment cannot be counted on for many application domains and indeed was not available for the LINAC4 tests. Also, the algorithms come with hyperparameters, but the default parameters gave mostly already very good results.

Model-free RL algorithms have the advantage of limited complexity and insignificant computer power requirements. The most sample-efficient algorithms (i.e., NAF, TD3) are straightforward to tune and can solve accelerator control problems, as shown in this paper. For accelerators with a lower repetition rate (e.g., repetition period of >1 minute), the number of iterations needs to be even further reduced to allow for stable training conditions or even allow for the training at all, given that accelerator time is expensive and normally overbooked. Model-based RL is a promising alternative to overcome the sample efficiency limitation, depending on the algorithm, however, at the expense of requiring significant computing resources. Another advantage of these algorithms is that an explicit model of the control problem response is a byproduct of the training of the agent.

The beam time reserved for advanced algorithms in 2020 at AWAKE will be used to deploy various model-based RL algorithms as well as model predictive control such as the iLQR [16] algorithm on a model obtained through supervised learning.

Another general challenge for RL agents next to the question of sample efficiency, addressed in this paper, is the availability of meaningful state observation. The RL agent for the automatching [32] of AWAKE source initial conditions to the transfer line lattice uses computer vision machine learning algorithms for the interpretation of OTR screen measurements, to implicitly encode the state.

In addition to studying new algorithms, infrastructure and frameworks will have to be deployed in the control system to easily make use of advanced algorithms and machine learning. This is also part of the goal for the 2020 AWAKE tests, where we aim to provide a generic optimization framework for the control room, including centrally stored neural networks as well as registering environments and algorithms for common use.

VI. CONCLUSIONS

Modern particle accelerators are complex, with often very dense user schedules, and need to be exploited efficiently. Deterministic operation and automation are the cornerstones for a new operations paradigm. Numerical optimizers are already used on a regular basis in the control room, for applications inaccessible to classical correction algorithms. With the recent progress in the field of deep reinforcement learning, parameter tuning in the control room can be learned by algorithms to further boost efficiency and extend the application domain. Not only are these algorithms readily available, but they are also straightforward to tune.

In this paper the sample-efficient RL algorithm NAF was successfully trained on real-world accelerator tuning problems with the trajectory steering agents at the CERN AWAKE electron line and the H^- accelerator LINAC4. We have also shown the potential of transfer learning for an RL agent, by training it on simulation and applying it on the real accelerator successfully with no retraining needed.

The main challenge is the implementation of the domain specific optimization problem with the adequate definition of reward in the chosen OpenAI Gym environments. Several guidelines are compiled in this paper to help with that task.

Only model-free RL algorithms were tested as part of this initial study. Model-free algorithms have insignificant requirements on computing infrastructure, but model-based ones have additional interesting advantages, particularly further increased sample efficiency. They will be part of the next test series.

Last but not least, as machine learning in all its forms will inevitably be part of operating particle accelerators, this work has shown that controls infrastructure will have to provide for storing and retrieving neural networks centrally

along with algorithms and the equivalent to OpenAI Gym environments.

APPENDIX A: COMPARISON: Q LEARNING VERSUS ON-POLICY REINFORCEMENT LEARNING

The NAF algorithm was compared with the state-of-the-art on policy algorithm PPO [33], see Fig. 15. As expected from literature, Q learning is more sample-efficient than on-policy learning—by at least one order of magnitude for this specific problem. A comparison between the TD3 algorithm and the implementation of NAF with prioritized experience replay is given in Fig. 16.

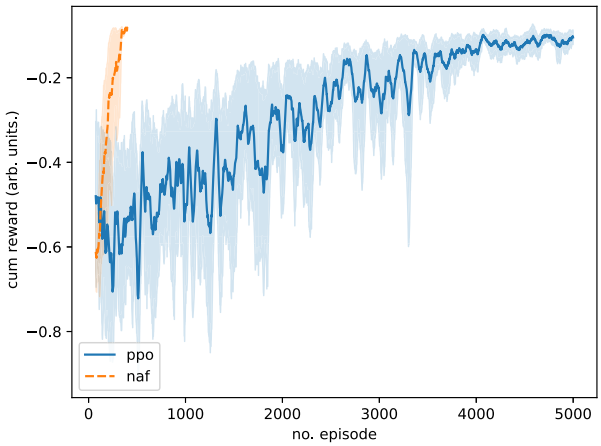

FIG. 15. A comparison between the on-policy state-of-the-art algorithm PPO and the NAF algorithm for the AWAKE trajectory correction problem. The cumulative reward was averaged over five different random seeds.

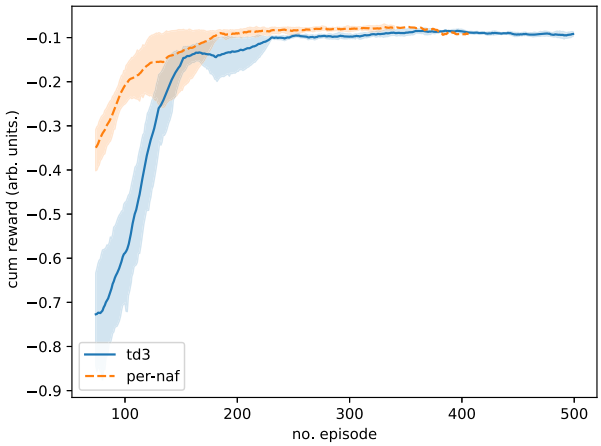

FIG. 16. A comparison between the state-of-the-art TD3 algorithm and PER-NAF on the AWAKE simulated environment. The overall performance is comparable.

APPENDIX B: PER-NAF DETAILS

1. Network and training parameter details

The NAF network architecture as described in this paper was used with activation function *tanh*(). The weights were initialized to be within $(-0.05, 0.05)$ at the beginning of the training and the learning rate α was set to 1×10^{-3} with a batch size of 50. For the Q-learning, a discount factor of $\gamma = 0.999$ was applied and τ for the soft target update was set to $\tau = 0.001$.

2. The prioritization of the samples

The data distribution for off-policy algorithms like Q-learning is crucial and current research proposes several strategies [34,35]. New data is gathered online following the current policy, while off-line data from the replay buffer is used to improve the policy. In our implementation, the data selected to minimize the loss function, or temporal difference error, during the training is weighted according to their previous temporal difference error. Data with a larger error is used more often for the weight update than other data [25]. Two parameters, α and β are important to control the priority and the weighting of the samples. The probability P_i to select a sample i from the replay buffer is

$$P_i = \frac{l_i^\alpha}{\sum_k l_i^\alpha}, \tag{B1}$$

where l_i is the temporal difference error of sample i. The sample is used with the weight w_i

$$w_i = \left(\frac{1}{N}\frac{1}{P_i}\right)^\beta \tag{B2}$$

For the NAF agents in this paper α and β were chosen to be 0.5.

Figure 17 shows a comparison of the obtained cumulative reward for different seeds during a training with the simulated Gym environment of AWAKE. The blue and red curves were obtained with prioritization (blue $\beta = 0.5$ and red $\beta = 0.9$), the green curve without. The variance of the results is smaller in case of using prioritization and the training faster such that higher cumulative rewards are reached earlier.

3. The exploration policy

For exploration during the training in the experiments discussed in this paper, Gaussian random noise was added to the proposed actions with a standard deviation $\sigma = 1$ decaying with $1/(0.1 \times n + 1)$, where n is the episode number. In Fig. 18 this exploration strategy, which is labeled as *inverse strategy* in the plot, is compared to a strategy with a standard deviation reduced linearly over 100 episodes, labeled as *linear strategy*. The upper plot shows the reduction of the standard deviation for the Gaussian

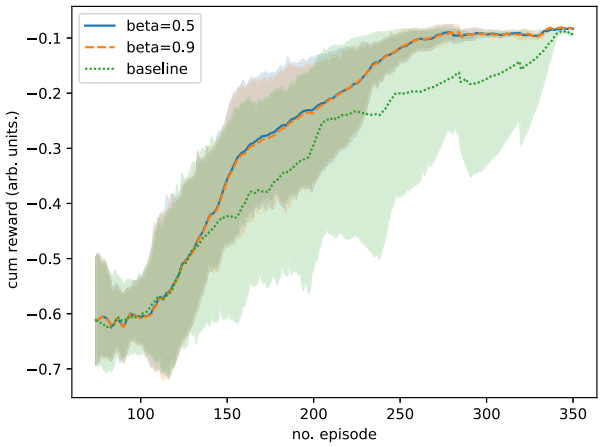

FIG. 17. The training with prioritization of the samples for two different values of β compared to the baseline without prioritization.

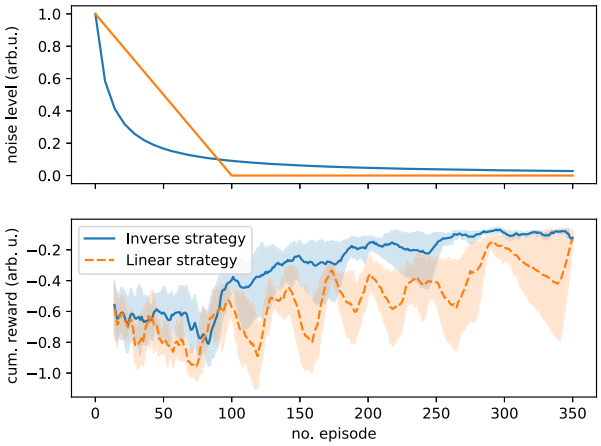

FIG. 18. Two exploration strategies: the *inverse strategy* results in a more stable training.

random noise as function of episode and the lower plot the cumulative reward per episode obtained for training with five different seeds. The *inverse strategy* results in a more stable training.

[1] D. Jacquet, R. Gorbonosov, and G. Kruk, LSA—The High Level Application Software of the LHC and its Performance during the first 3 years of Operation, *14th International Conference on Accelerator & Large Experimental Physics Control Systems, San Francisco, CA, USA, 6–11 Oct 2013*, pp. thppc058, https://cds.cern.ch/record/1608599.

[2] A. Edelen, N. Neveu, M. Frey, Y. Huber, C. Mayes, and A. Adelmann, Machine learning for orders of magnitude speedup in multiobjective optimization of particle accelerator systems, Phys. Rev. Accel. Beams **23**, 044601 (2020).

[3] G. Azzopardi, A. Muscat, G. Valentino, S. Redaelli, and B. Salvachua, Operational results of LHC collimator alignment using machine learning, in *Proc. IPAC'19, Melbourne, Australia* (JACoW, Geneva, 2019), pp. 1208–1211.

[4] S. Hirlaender, M. Fraser, B. Goddard, V. Kain, J. Prieto, L. Stoel, M. Szakaly, and F. Velotti, Automatisation of the SPS ElectroStatic Septa Alignment, in *10th Int. Particle Accelerator Conf.(IPAC'19)* (JACoW, Geneva, 2019), p. 4001–4004.

[5] J. Duris *et al.*, Bayesian Optimization of a Free-Electron Laser, Phys. Rev. Lett. **124**, 124801 (2020).

[6] A. Hanuka *et al.*, Online tuning and light source control using a physics-informed Gaussian process Adi, https://arxiv.org/abs/1911.01538.

[7] M. McIntire *et al.*, Sparse Gaussian processes for Bayesian optimization, https://www-cs.stanford.edu/~ermon/papers/sparse-gp-uai.pdf.

[8] N. Bruchon, G. Fenu, G. Gaio, M. Lonza, F. A. Pellegrino, and E. Salvato, Toward the Application of Reinforcement Learning to the Intensity Control of a Seeded Free-Electron Laser, *2019 23rd International Conference on Mechatronics Technology (ICMT), SALERNO, Italy, 2019*, pp. 1–6, https://doi.org/10.1109/ICMECT.2019.8932150.

[9] N. Bruchon, G. Fenu, G. Gaio, M. Lonza, F. H. O'Shea, F. A. Pellegrino, and E. Salvato, Basic reinforcement learning techniques to control the intensity of a seeded free-electron laser, Electronics **9**, 781 (2020).

[10] T. Boltz *et al.*, Feedback design for control of the microbunching instability based on reinforcement learning, in *10th Int. Particle Accelerator Conf.(IPAC'19)*, https://doi.org/10.18429/JACoW-IPAC2019-MOPGW017.

[11] A. Edelen *et al.*, Using a neural network control policy for rapid switching between beam parameters in an FEL, in *38th International Free Electron Laser Conference*, https://doi.org/10.18429/JACoW-FEL2017-WEP031.

[12] E. Shaposhnikova *et al.*, LHC injectors upgrade (LIU) project at CERN, in *7th International Particle Accelerator Conference, Busan, Korea, 8–13 May 2016*, pp. MOPOY059, https://doi.org/10.18429/JACoW-IPAC2016-MOPOY059.

[13] E. Adli, A. Ahuja, O. Apsimon, R. Apsimon, A.-M. Bachmann, D. Barrientos, F. Batsch, J. Bauche, V. B. Olsen, M. Bernardini *et al.*, Acceleration of electrons in the plasma wakefield of a proton bunch, Nature (London) **561**, 363 (2018).

[14] A. Scheinker, S. Hirlaender, F. M. Velotti, S. Gessner, G. Z. Della Porta, V. Kain, B. Goddard, and R. Ramjiawan, Online multi-objective particle accelerator optimization of the AWAKE electron beam line for simultaneous emittance and orbit control, AIP Adv. **10**, 055320 (2020).

[15] R. Sutton and A. Barto, *Introduction to Reinforcement Learning* (MIT Press, Cambridge, MA, USA, 2018).

[16] H. Kwakernaak and R. Sivan, *Linear Optimal Control Systems* (Wiley-Interscience, New York, 1972).

[17] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning, arXiv:1708.02596.

[18] R. Sutton, Dyna, an integrated architecture for learning, planning, and reacting, *AAAI Spring Symposium* 2, 151 (1991).

[19] T. Wang *et al.*, Benchmarking model-based reinforcement learning, https://arxiv.org/abs/1907.02057v1.
[20] V. Mnih *et al.*, Human-level control through deep reinforcement learning, Nature (London) **518**, 529 (2015).
[21] T. Lillicrap *et al.*, Continuous control with deep reinforcement learning, in *Proc. ICLR 2016*, https://arxiv.org/abs/1509.02971.
[22] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, Continuous deep Q-learning with model-based acceleration, in *Proc. 33rd International Conference on Machine Learning, New York, NY, USA, 2016* [arXiv:1603.00748].
[23] S. Fujimoto, H. van Hoof, and D. Meger, Addressing function approximation error in actor-critic methods, arXiv:1802.09477.
[24] Y. Gal, R. McAllister, and C. E. Rasmussen, Improving PILCO with Bayesian neural network dynamics models, in Data-Efficient Machine Learning workshop, ICML **4**, 34 (2016).
[25] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, Prioritized experience replay, arXiv:1511.05952.
[26] S. Hirlaender, PER-NAF available at https://github.com/MathPhysSim/PER-NAF/ and https://pypi.org/project/pernaf/.
[27] PYJAPC available at https://pypi.org/project/pyjapc/.
[28] http://gym.openai.com.
[29] M. J. D. Powell, A direct search optimization method that models the objective and constraint functions by linear interpolation, in *Advances in Optimization and Numerical Analysis*, edited by S. Gomez and J.-P. Hennart (Kluwer Academic, Dordrecht, 1994), pp. 51–67.
[30] MAD-X documentation and source code available at https://mad.web.cern.ch/mad/.
[31] G. Bellodi, Linac4 commissioning status and challenges to nominal operation, *61st ICFA Advanced Beam Dynamics Workshop on High-Intensity and High-Brightness Hadron Beams, Daejeon, Korea, 17–22 Jun 2018*, pp. MOA1PL03, https://doi.org/10.18429/JACoW-HB2018-MOA1PL03.
[32] F. Velotti, B. Goddard *et al.*, Automatic AWAKE electron beamline setup using unsupervised machine learning (to be published).
[33] J. Schulmann *et al.*, Proximal policy optimization algorithms, https://arxiv.org/pdf/1707.06347.pdf.
[34] A. Kumar, A. Gupta, and S. Levine, DisCor: Corrective feedback in reinforcement learning via distribution correction, arXiv:2003.07305.
[35] A. Kumar, J. Fu, G. Tucker, and S. Levine, Stabilizing off-policy Q-learning via bootstrapping error reduction, arXiv:1906.00949.