Policy gradient methods for free-electron laser and terahertz source optimization and stabilization at the FERMI free-electron laser at Elettra

F. H. O'Shea,[1,*,†] N. Bruchon,[2] and G. Gaio[1]
[1]*Elettra Sincrotrone Trieste, 34149 Trieste, Italy*
[2]*University of Trieste, 34127 Trieste, Italy*

(Received 25 July 2020; accepted 3 December 2020; published 21 December 2020)

In this article we report on the application of a model-free reinforcement learning method to the optimization of accelerator systems. We simplify a policy gradient algorithm to accelerator control from sophisticated algorithms that have recently been demonstrated to solve complex dynamic problems. After outlining a theoretical basis for the functioning of the algorithm, we explore the small hyperparameter space to develop intuition about said parameters using a simple number-guess environment. Finally, we demonstrate the algorithm optimizing both a free-electron laser and an accelerator-based terahertz source in-situ. The algorithm is applied to different accelerator control systems and optimizes the desired signals in a few hundred steps without any domain knowledge using up to five control parameters. In addition, the algorithm shows modest tolerance to accelerator fault conditions without any special preparation for such conditions.

DOI: 10.1103/PhysRevAccelBeams.23.122802

I. INTRODUCTION

In this work we demonstrate a simple model-free reinforcement learning algorithm tuning and maintaining accelerator systems. Herein, we simplify a policy gradient algorithm related to those that have recently been demonstrated solving the complex, dynamic problem of playing human players in video games [1,2]. The algorithm is noise tolerant, requires little training beforehand, has few hyper parameters, and produces both a point estimate and an uncertainty estimate for all of the controlled systems. In addition, the algorithm natively adjusts the precision of the systems settings; and has demonstrated a tolerance for short system shut downs in some cases. In this article, we describe the algorithm and how we arrived at it, show a simple simulation method for selecting the hyper parameter, and show it operating several different accelerator subsystems.

The success of accelerators is, in part, judged by the science output and a key factor in output is the time spent serving the users. Because of this, any tuning done online places a premium on fast learning, whether by human operators or computer systems. In this context, we demonstrate a simplified policy gradient method [3], a type of reinforcement learning algorithm, tuning and maintaining a free-electron laser and a terahertz source. Algorithms within this family have recently shown the ability to perform complex tasks with extensive training [1,2]. Our goal is to decrease the number of steps to convergence in the somewhat simpler realm of accelerator control.

As computing power has proliferated in recent decades, a variety of computer-based tools have been developed to improve a machine setting or maintain performance [4–9]. Among the more recent of these tools are methods based on machine learning (ML). In addition to machine tuning and control, ML models can also be used to create new diagnostics [10]. The majority of the applications of ML to accelerators are based on supervised learning, where a dataset with labeled outputs is used to fit a model to produce outputs using novel data. A drawback to these methods is the required labeled dataset. Without a dataset, the model cannot be trained. Or, even so, the dataset may not cover the desired operational range or be too noisy and the model may be poorly fit in regions of operational interest.

An approach to mitigating this problem is to use the information available to set the accelerator in approximately the correct fashion and then tune it from there. Indeed, this is standard practice at all accelerator facilities, even when humans tune the accelerator. Presumably the initial setting was not random and a superior setting might be found in the local parameter-space by either manual tuning [11], automatic feedbacks [4], random search [6], or some other optimization technique [5–8]. Similar to these types of algorithms, reinforcement learning might be

*finn.h.oshea@gmail.com
†Present address: Nusano Inc., 28575 Livingston Ave, Valencia, California 91355, USA.

used to explore the parameter space and find a better solution [12].

Reinforcement learning (RL) is a machine learning paradigm in which an *agent* is allowed to take *action* in an *environment* in which it receives *rewards* for performing desired behavior while it follows a *policy*. The goal of reinforcement learning is to find an optimal policy to follow and a distinguishing feature of reinforcement learning as compared to supervised learning is that it requires the agent to interact with the environment, rather than learn from a set of labeled data. The optimal policy is typically defined as the policy that produces the largest reward. In the context of accelerator operations, the agent is a piece of control software and the environment is the accelerator itself. The definitions of the other elements depends on what exactly the agent is doing. For example, the actions might be to adjust the settings of steering magnets to improve the energy output of a free-electron laser which is used to compute a reward.

In Sec. II we describe why we decided to work with reinforcement learning agents at FERMI. In Sec. III we briefly review policy gradient methods; we also describe the specific algorithm we use in the present work. The software used in this work was written entirely by the authors in python 3 [13], using available scientific libraries [14]. In Sec. IV we describe the types of policies we consider for deployment at the accelerator. In Sec. V we use a fast numerical simulation to guide the selection of the policy type and hyperparameter settings. In Sec. VI we demonstrate the algorithm performing various tasks on an FEL and THz source. We conclude with some remarks on the performance of the algorithm.

II. WHY USE REINFORCEMENT LEARNING AT FERMI?

The principal reason we have decided to use reinforcement learning, instead of supervised learning, is that our work with supervised learning showed that the learning would have to be continuous.

The operational conditions at FERMI are such that the FEL is typically reconfigured for a new user twice per week. The changes include everything from the undulator settings to the beam energy which can involve activating or deactivating klystrons. For every new run the accelerator physics team retunes the machine, frequently starting at the electron gun and working all the way to the beam dump. It is not unusual for the retuning process to alter the setting of dozens of features: power supply settings, klystron phases, and so on. Some configurations are used for less than a week and are not reused for months or years.

This situation creates a sparsity of data for training supervised learning models. Every time the accelerator is retuned, the machine learning model is likely to need retraining, with all previous information rendered potentially useless. Every time the accelerator is tuned to a configuration that the machine learning model has never been exposed to, the model must certainly be retrained. Taken a whole, these aspects of operations at FERMI mean that the model would probably have to be retrained several times a week, at least.

However, even this is a somewhat optimistic assessment of the longevity of the value of asynchronous training at FERMI. We would regularly find that the ability of a trained model to predict other features of the accelerator would decay in less than an hour. We show an example of the reduction in the model performance as a function of time in Fig. 1. To make the prediction model we use library functions from scikit-learn [15] to build a neural network with 2 hidden layers of size 100 and tanh activation function. The task is for the agent to predict the FEL intensity as measured by the intensity monitor [16]. This data was taken while FEL1 was in HGHG operation [17].

The 162 features used in the model are taken from the entire accelerator from the photocathode laser to the beam dump. No features from the photon transport line were used

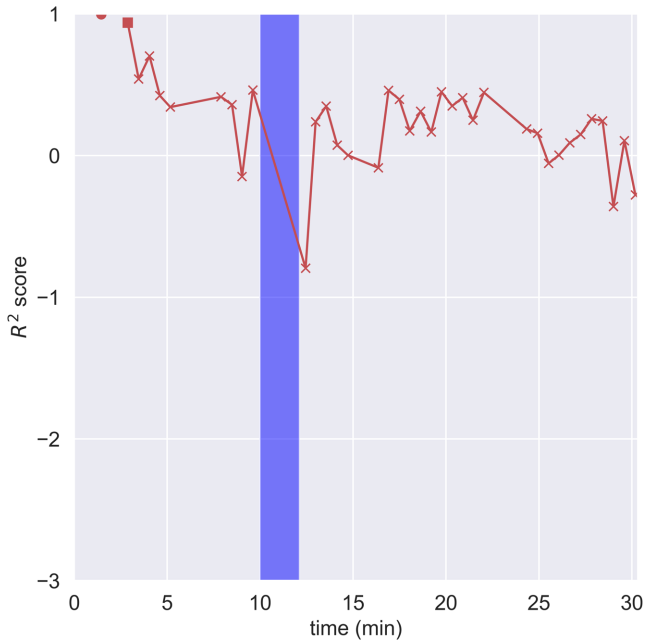

FIG. 1. Prediction performance of a neural network model at FERMI. The model uses several hundred features along the linac and FEL to predict the output intensity of the FERMI FEL. The R^2 score on the training set is shown as a circle, the score is 0.995. The score on the cross validation set is shown as a square, the score is 0.914. The x-marks show the R^2 score for the model as a function of time after the first training sample was taken. The samples are accumulated in batches of approximately 1000 and the score is computed for each batch. Before each batch is scored, any sample with any feature value greater than 5 standard deviations from the mean for that batch is removed. The shaded region beginning at approximately 10 minutes shows a period of time when Klystron #3 went in to fault and the accelerator was not running.

as they are very strongly correlated with the FEL intensity and it seemed to us to defeat the purpose of the task if we were to use photon beam measurements to predict photon beam properties. The features include: BPM readings along the whole accelerator, corrector magnet settings inside FEL1, various properties of the photocathode, laser heater and seed lasers (position, intensity, delay, etc.), the bunch charge measured at the beam dump, the time of arrival of the bunch at the bunch linearizer, and the pyro reading of the bunch length.

The accelerator and FEL were left to run with the feedbacks on for approximately 30 minutes, wherein we took approximately 27000 data points. The data was taken using a data collection system at FERMI that is asynchronous and has to be operated by the user manually. Because of these constraints, the data points are not spread out evenly over the 30 minute interval. However, the data covers the time period sufficiently well for our purposes here.

The training and model evaluation were performed offline. The neural network is trained on the first 5000 data points (approximately 10 minutes worth of data), and the next 1000 points are used as a cross-validation set for hyperparameter scans and selection. After this time period, the remaining points are grouped into sets of approximately 1000 and scored. The scoring system used is the so-called coefficient of determination (R^2) which is computed as

$$R^2 = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \bar{y})^2}. \tag{1}$$

Where the y_i are the measured values of the FEL intensity, f_i are the values of the FEL intensity predicted by the neural network, and $\bar{y}$ is the average intensity of the FEL. The maximum possible value for R^2 is 1, when the measured and predicted values are identical, and the minimum is $-\infty$. The principal reason for using this scoring system is that it is included in the scikit-learn package. We show data from two of the data points from Fig. 1 in Fig. 2 to illustrate the difference between a high coefficient of determination ($\sim$1) and a coefficient of determination near to zero.

It is clear from Fig. 1 that a few minutes of data is not enough to train a model to predict the FEL output intensity. Rather than retraining the model every few days as we might hope from the FEL tuning schedule we previously discussed, we would have to continuously train the model. It is possible that some more elaborate or sophisticated combination of models and machine learning techniques could successfully maintain the model's prediction ability over hours or even days, however once the accelerator is retuned, we would have to restart the training process anyhow.

Since continuous training appears necessary given the foregoing performance of supervised learning and the value of data collection to produce training sets seems to be very small after even half an hour, we decided that the task of tuning or retuning FERMI should be approached differently. Hence, we decided to pursue reinforcement learning, an online learning method that requires very little initial data. Indeed, the method learns from interacting with the system it is used to control, which suggests we do not need to manually scan parameters to produce a training dataset.

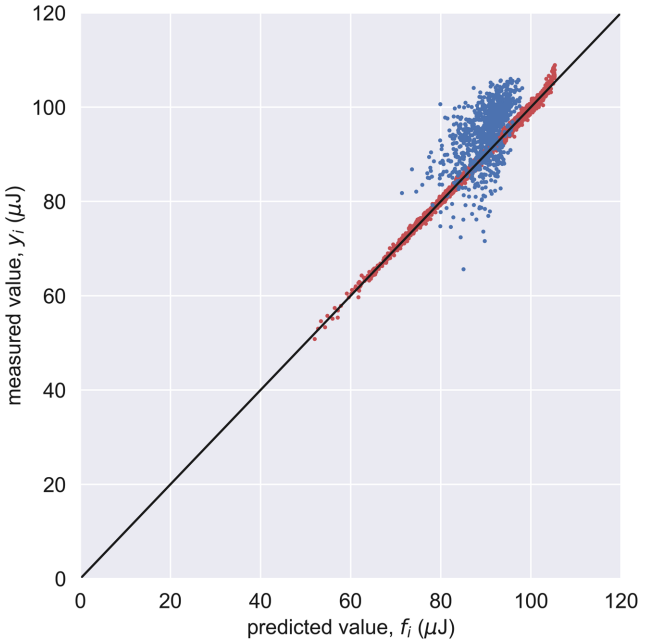

FIG. 2. Comparison of two of the data points shown in Fig. 1. The red dots compare the predicted and measured values for the training set ($R^2 = 0.995$) while the blue dots show the same for the point at approximately 16 minutes ($R^2 = -0.054$). The black line shows a model with perfect prediction capacity.

III. MINIMUM POLICY GRADIENT LEARNING

All of the RL methods discussed in this article are *Markov decision processes* with discrete time steps in which the problem consists of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function $r(s,a):\mathcal{S}\times\mathcal{A}\to\mathbb{R}$ and a conditional probability of transition between the states $P(s_{t+1}|s_t,a_t)$ that obeys the Markov property. The *policy* is the collection of actions that can be taken from a given state $\pi_\theta(a|s)$. Future rewards are discounted by a value $\gamma\in[0,1]$ [18].

In this article we focus on reinforcement learning in the case of continuing tasks, i.e., tasks that have no defined end states and may run, in principle, forever. This viewpoint is in contrast to recent work wherein each step is treated as a single-step episode [20]. In that case, the algorithm we present here was shown to be quite similar to the REINFORCE algorithm [21]. Herein, we start from the view that the algorithm is a policy gradient algorithm.

Policy gradient methods are an alternative to value-function based reinforcement learning methods [19].

Briefly, value-function based methods estimate the value of the states partially by using the value of the subsequent states, the desired behavior from the agent is then to visit the high value states as often as possible, and, from the algorithm designer's perspective, achieve convergence as quickly as possible. Perhaps the most well-known example of a value-function based-RL method is deep Q-learning with neural networks [22]. This RL algorithm uses function approximation to adapt an off-policy tabular learning method to problems where the number of state-action pairs is large. The actions at each step are chosen from a finite list of options that may be state-dependent via the epsilon-greedy method [23]. A shortcoming of deep Q-learning is that the policy improvement theorem, that Q-learning relies on to guarantee convergence, no longer applies when function approximation is used [19].

Alternatively, policy gradient methods parametrizes a continuous policy for the actions. This has a number of advantages over value-function based methods: (1) the policy can be fundamentally stochastic and we do not need to include off-policy techniques or variably greedy action choices (such as epsilon-greedy) to allow or encourage exploration of the action space; (2) the policy can, in principle, become deterministic, if that is what the environment rewards; (3) policies are usually what we are interested in, i.e., when the accelerator is in a certain state we want the agent to take certain actions, and by defining the policy we have the ability to incorporate accelerator domain knowledge in the definition of the policy; (4) convergence to (at least) a local optimum is guaranteed [19].

Some examples of how accelerator domain knowledge might be used to choose or constrain a policy for the agent are as follows. If the designer knows that turning a certain dipole down too low will cause unacceptable beam losses, they can prohibit the policy that the agent controls that dipole with from becoming too low. This way, the agent does not have to learn from the negative feedback of beam loss. Another example is if a particular beam line uses a quadrupole doublet. It might be useful for the doublet condition (equal and opposite gradients in the two quadrupoles) to be relaxed, but not entirely. In this case the designer could ensure the policies of the two magnets are correlated. The strength of the correlation could itself be a learned parameter.

Policy gradient methods rely on optimizing the average expected discounted reward the agent receives over a horizon, h, which is the number of steps into the "future" that are used in estimating the expected discounted reward [24]

$$J_t \doteq \mathbb{E}\left[\frac{1}{h}\sum_{u=1}^{h}\gamma^{u-1}r_{t+u}\right] = \frac{1}{h}\sum_{u=1}^{h}\gamma^{u-1}\mathbb{E}_{\substack{a\sim\pi_{t+u}\\ s\sim p_{t+u}}}[r_{t+u}]$$
$$= \frac{1}{h}\sum_{u=1}^{h}\gamma^{u-1}\int_{\mathcal{S}}\int_{\mathcal{A}} r(\boldsymbol{s},\boldsymbol{a})p_{t+u}(\boldsymbol{s},\boldsymbol{a})d\boldsymbol{a}d\boldsymbol{s}, \quad (2)$$

where t is an arbitrary step number and the expression for the average expected discounted reward at the next step is

$$J_{t+1} = \frac{1}{h}\sum_{u=1}^{h}\gamma^{u-1}\int_{\mathcal{S}}\int_{\mathcal{A}} r(\boldsymbol{s},\boldsymbol{a})p_{t+u+1}(\boldsymbol{s},\boldsymbol{a})d\boldsymbol{a}d\boldsymbol{s}. \quad (3)$$

The interpretation of $p_n(\boldsymbol{s},\boldsymbol{a})$ is the joint probability that the agent will visit the state $\boldsymbol{s}$ and take action $\boldsymbol{a}$ on step n and we see that it can change at every step, as is indicated by the subscript to p. Practically, the value for h is arbitrary and the sum can be truncated when the $h+1$ discounted future reward is small compared to the sum of the previous rewards. Formally, h can be infinite so long as we do not also have $\gamma = 1$. If $\gamma = 1$, then the system must reach a terminal state for the reward to be finite, in which case h is the number of steps until a terminal state. This would make the solution a Monte Carlo policy gradient method [19,21]. However, we are here focusing on continuing tasks, and do not pursue this further.

The policy is updated, in principle, using gradient ascent such that

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\nabla_\theta J_t(\boldsymbol{\theta}), \quad (4)$$

where $\boldsymbol{\theta}$ is the set of policy parameters that the agent controls, and α is the learning rate. In order to compute this, we need to obtain a relationship between J_t and the agent's policy.

If we change the agent's behavior at step $t+1$, we expect that J might change: $J_{t+1} = J_t + \Delta J$. Combining Eqs. (2) and (3) we find a way to compute the quantity we need as

$$\Delta J = \frac{1}{h}\int_{\mathcal{S}} d\boldsymbol{s}\int_{\mathcal{A}} d\boldsymbol{a}r(\boldsymbol{s},\boldsymbol{a})\Bigg[(1-\gamma)\sum_{u=1}^{h-1}\gamma^{u-1}p_{t+u+1}(\boldsymbol{s},\boldsymbol{a})$$
$$+\gamma^{h-1}p_{t+h+1}(\boldsymbol{s},\boldsymbol{a}) - p_t(\boldsymbol{s},\boldsymbol{a})\Bigg]. \quad (5)$$

If we know how p changes as a function of step number, we can compute the change in J, but we do not usually know p. For the purposes of the present work we make the assumption that agent changes its policy at step $t+1$ and then follows it until it has taken at least h more steps in the MDP. With this assumption, we find that

$$\Delta J = \frac{1}{h}\int_{\mathcal{S}} d\boldsymbol{s}\int_{\mathcal{A}} d\boldsymbol{a}r(\boldsymbol{s},\boldsymbol{a})$$
$$\times[\pi_{t+1}(\boldsymbol{a}|\boldsymbol{s})u_{t+1}(\boldsymbol{s}) - \pi_t(\boldsymbol{a}|\boldsymbol{s})u_t(\boldsymbol{s})]. \quad (6)$$

Where we have used the identity $p_t(\boldsymbol{s},\boldsymbol{a}) = \pi_t(\boldsymbol{a}|\boldsymbol{s})u_t(\boldsymbol{s})$, where $\pi_t(\boldsymbol{a}|\boldsymbol{s})$ is the policy at step t (the probability of taking action $\boldsymbol{a}$ from state $\boldsymbol{s}$) and $u_t(\boldsymbol{s})$ is the probability distribution of states, $\boldsymbol{s}$, at step t. We note that the discount

factor has disappeared and that h is now an arbitrary constant [25]. This equation is still difficult to use unless we know the probability of visiting the states. To further simplify the computation we might assume that $u_{t+1}(\boldsymbol{s}) = u_t(\boldsymbol{s})$, as is assumed for policy gradient methods [19]. In that case, since the policy is parametrized by $\boldsymbol{\theta}$ we can write the update rule for the parameters as

$$\boldsymbol{\theta}_{t+1} \approx \boldsymbol{\theta}_t + \alpha \mathbb{E}_{\substack{a\sim\pi_\theta \\ s\sim u}}[r(\boldsymbol{s},\boldsymbol{a})\nabla_\theta \ln(\pi_\theta(\boldsymbol{a}|\boldsymbol{s}))], \tag{7}$$

where h has been absorbed in to the learning rate. This is a restatement of the policy gradient theorem [3].

The validity of the last assumption is dubious because the goal of learning is to find a policy that takes the agent to higher-value states and thusly produces higher average return. On the other hand, the equality is approximately true if the change in policy is small. This is generally useful but might limit how quickly the agent can learn and update its policy.

An unrelated difficulty with this method is that it must calculate an expectation value in Eq. (7). We can reduce the computational complexity by approximating Eq. (7) using *stochastic* gradient ascent

$$\boldsymbol{\theta}_{t+1} \approx \boldsymbol{\theta}_t + \alpha r(\boldsymbol{s},\boldsymbol{a})\nabla_\theta \ln(\pi_\theta(\boldsymbol{a}|\boldsymbol{s})). \tag{8}$$

The convergence properties of Eq. (8) can be improved if the gradient is performed with respect to the Fisher metric rather than directly in parameter space, in which case we take $\nabla_\theta \to \mathcal{I}(\boldsymbol{\theta})^{-1}\nabla_\theta$ in the preceding equations, where $\mathcal{I}(\boldsymbol{\theta})^{-1}$ is the inverse of the Fisher matrix [26]. This is the update formula we use in this work.

Previously we had assumed that the changes to the distribution of states with policy changes was negligible and used that to develop an update rule for the policy parameters. But there is one case in which the distribution of states is always steady, regardless of the size of policy changes: if there is only one state. Not only does the assumption of a single state allow us this improvement to the theoretical basis of policy improvement, but it has other benefits as well. It also obviates the need for a definition of states of the accelerator and increases the density of rewards because they all accrue to that single state. Both of these features should decrease the number of steps required to learn the optimal policy. The cost is that the agent does not store any information about actions that are not available in the current policy. If the agent is sufficiently fast at learning, this drawback may be worth the cost. Because we are working with a single state, we note that the transition probability mentioned earlier, $P(s_{t+1}|s_t, a_t) = 1$, is deterministic.

The algorithm we used is based on the continuing actor-critic method with eligibility traces of Sutton and Barto [19]. We chose a continuing algorithm because there was no clear way to define an end state for the accelerator environment. In addition, as there is only one state, we did not use the critic. The motivation for making this change is straightforward, if naïve: we did not want to spend valuable training time learning the value of the state. This change would not be possible with state-based methods, but policy gradient methods require only a parametrized policy and do not require states. We call the algorithm minimum policy gradient learning and it is shown in Algorithm 1.

Algorithm 1. Minimum Policy Gradient Learning.

```
1  Input: Policy distribution, Fisher matrix and gradient
2  Algorithm parameters: α, α_R, λ_θ
3  Initialize the model parameters, θ ← 0
4  Initialize the eligibility trace vector, z ← 0
5  Initialize the base line, R̄ = 0
6  While True:
7     A ~ π(·|θ)
8     Take action A, observe reward R
9     δ ← R − R̄
10    R̄ ← R̄ + α_R δ
11    z ← λ_θ z + I(θ)^{-1} ∇ ln π(A|θ)
12    θ ← θ + α δ z
```

The algorithm proceeds as follows. The user chooses a policy distribution to use and then computes the associated gradient and Fisher Information Matrix (examples of these elements are given in Sec. IV). The users also selects values for the learning rate for the parameters (α), the learning rate for the base line (α_R), and the strength of the eligibility trace (λ_θ). All other parameters are initialized to arbitrary values [27]. On line 7 of Algorithm 1, the policy is sampled to return a specific action. Next, that action is taken by the agent who receives a reward (details on rewards we used can be found in Secs. V and VI) and that reward is adjusted for the baseline (line 9). On line 10 the baseline is updated using gradient ascent. Next, the eligibility trace vector is updated on line 11. Finally, the parameters are updated using gradient ascent.

In fact, because there is only one state, the base line, $\bar{R}$ in Algorithm 1, can serve as a kind of critic. However, we found that the baseline slowed learning because it rewarded the agent too generously for simply taking actions with better-than-baseline reward. Because of this, we have used $\alpha_R = 0$ in all of the simulations and accelerator control tests presented in this work.

Whether the trade-off between density of reward and the more detailed information about the accelerator stored by a larger number of states is worthwhile is beyond the scope of the present work. However, we make a few observations here. First, if the accelerator parameter space is large, and it usually is, and the accelerator is unlikely to revisit a state often, then the agent is unlikely to make use of the information anyway. Second, large parameter spaces require careful generalization techniques to ensure that the information is not "spread so far" that the agent is

treating different states of the accelerator as the same state. Third, it is not clear that the list of features currently measured and recorded at, for example, FERMI, is sufficient to distinguish between different accelerator states. For example, the best diagnostic for the strength of the microbunching instability [28] at FERMI is the FEL itself, and so a fundamental feature to the performance of the FEL is only known to the agent through the rewards it receives. That information is stored in the policy that the agent learns and not in the state, and it is not clear how to define a state such that it encodes that information. Presumably that information might be emergent; the agent might learn to control the microbunching instability through some complex combination of control parameters that are as-yet too contrived for a human operator.

Finally, we note a potential cost to the single-state assumption of our model. With only a single state, it seems likely that model will not respond well to sufficiently fast and large changes to the accelerator. There is some evidence that this is the case in our present work as demonstrated in the mixed results with respect to recovery from faults or user perturbation of the system being controlled, e.g., in Figs. 9 and 11. But it is also not yet clear what changes to the model would improve the agent's performance and how many resources those changes would require to be successful.

As a heuristic for selecting which systems this algorithm might successfully control, we asked ourselves a question: will changes in the accelerator state, as modeled by the input to any arbitrary control algorithm, require that the algorithm take very different actions faster than it can learn? If yes, then a reinforcement algorithm will likely need to use state representations as a way to differentiate between the different control situations it encounters. If no, then a single-state might suffice. Examples of subsystems that might be controlled well by our single state algorithm are beam trajectory control, temperature stabilization (cryogenic or otherwise), maintaining rf system parameters against drift, fine tuning of undulator gaps in a chain of undulators, or optimizing the laser spot profile on a photocathode. During normal operation of these systems, the agent should make modest changes to the parameters being controlled, i.e., the parameters being controlled should be adjusted by a small amount to explore for better potential settings nearby in parameter space, but should not be taking more drastic actions, such as shutting down a magnet in a single step. We can view tune-up of an accelerator as a subset of this problem, with the principal difference being the mode and variance of the policy (larger variance leading to more exploration), which have to be supplied by the user, even if arbitrary. Exploration of how to improve the behavior of the algorithm to more sophisticated control situations is the subject of future work.

In contrast to the focus of some recent machine learning applications both outside [1,2] and inside the field of accelerator physics [29], we do not use a neural network, even though neural networks are compatible with policy gradient methods. The reason is once again that we want to minimize the number of steps the agent takes to learn. One of the reasons that deep neural networks have become such a powerful machine learning tool is that they can learn a representation of the data, but that must clearly take more data than simply adapting an assumed representation to the data. As such, we focus on the simplest representation of a control setting that we can conceive of that is sufficiently complex to capture the required features for setting a control system parameter. In the case of this work, that is a unimodal distribution of finite support. In addition to the amount of data needed to train, it is often difficult to explain why a neural network acts as it does. With parametrized policies, it is very easy to observe the evolution of the agent's policies for running the machine.

The agents used in this work consist of independent univariate policy distributions for each system controlled, i.e., we assume no correlation between the systems being controlled. In addition, as all accelerator control systems have a finite range of allowed input that is often set by physical limits, e.g., the current limits on a power supply, we focused on three policy distributions with finite support: the von Mises distribution, the beta distribution and an alternative parametrization of the beta distribution using mode and concentration. The range of support for the various distributions is easily scaled to fit the range of the system being controlled.

A recent comparison between the normal policy distribution and the beta policy distribution in the context of policy gradient methods found that, while the beta-distribution learned faster on some benchmark tasks with controls of finite range, the agents based on the normal distribution policy successfully completed the tasks [30]. In contrast, we found that the normal distribution was quite unstable in the context of the fast learning that we desire here and we did not pursue it further. As the other authors note, this is likely because of the bias caused by the mismatch between the regions of support of the normal distribution and the control system.

A. Conceptual comparison of policy gradient methods with other machine tuning techniques

We focused on policy gradient methods for a number of technical reasons which were just covered, but also because they show some appealing similarities and synergies with previously studied techniques. A recent publication has produced an overview of automatic accelerator tuning mechanisms in the context of a model-free optimizer [8]. In this section we build on that overview by including policy gradient methods and compare them to a few other online tuning algorithms.

Fundamentally, all automatic tuning algorithms are an attempt to balance exploration and exploitation. Exploitation

is the process of the agent moving toward an extrema directly, i.e., it is exploiting its knowledge of the local parameter space to quickly find a local extrema. Exploration is the process of evaluating new regions of the parameter space. Typically, it is straightforward to define an exploitative change in parameters, but they lead to only a local extrema. To find even larger extrema, the agent must explore to some extent, but exploration is costly in terms of time and potentially other resources. Most online tuning algorithms, therefore, differ in how they explore the parameter space.

An example of a similarity with other tuning techniques is the conceptual similarity between random walk optimization and policy gradient methods. In random walk optimization the available range to search is set by the random number generator (with perhaps some user enforced limits). The algorithm samples the random number generator, applies the results to the machine, and keeps the new setting if the target variable is better. Once the exploration has found a better setting for the machine, the random sampling continues as is clear from Fig. 2 in [6].

In policy gradient methods the policy itself provides the variance allowed at each step, rather than being set by the user beforehand. As the policy is updated during operation the variance will grow for settings that are not critical to the target variable or are badly miss-set, while the variance will shrink for settings that are well set and whose setting strongly affects the target variable. Once the policies are converged and presumably the variances on the important settings are small, the variation in the settings can become quite small, reducing the noise in the target variable. As the machine drifts and the current policies perform more poorly, the variance in the policies will grow and adjust to the new state of the machine.

As an example of a synergy with other ML techniques, Bayesian methods for predicting the setting of a machine will result in a point estimate of the setting and an uncertainty in that setting, or more generally, a probability distribution of available settings. Since the policies in policy gradient methods are nondeterministic, one can in principle start the policy-gradient-method-based tuning of the machine with all the information available from the Bayesian model by using the aforementioned probability distribution as the initial policy. Or conversely, policy gradient methods can be used to generate data for Bayesian methods which might benefit from information for developing the prior, rather than starting with an arbitrary prior.

Finally, we compare policy gradient methods to the previously mentioned model-free optimizer (MFO) [8]. In that algorithm, the parameter space is explored by including noise in the update of the parameter settings. This helps prevent the agent from becoming trapped in a local minimum, and it also allows the agent to be more tolerant to noise in the target variable(s) or, equivalently, soft precision in the parameter settings.

Policy gradient methods accomplish this feature by using a stochastic policy; the finite variance of the policy allows the agent to explore regions of parameter-space that do not directly lead to higher reward. On the other hand, the policy gradient method also takes care of the details about how to set the gradient parameters because it is the policy itself that is being updated, reducing the number of parameters that have to be set by some other means. The MFO has three parameters for each systems being controlled, and they have to be recalculated after drifts in the machine. The policy gradient method has, in our case, one parameter that is used for each of the systems being controlled. In this work we end up using one parameter for all the systems being controlled, even when the systems are totally different.

Finally, we note that for minimum policy gradient learning the policies themselves are simple to interpret and can be very informative to users. For example, as the agent converges on a solution, the variance of the policy will decrease if the system being controlled has an effect on the target variable. It is a clear indication that a control system does not strongly affect the target variable if a policy for that system shows high variance while the other system's polices are much lower in variance. This can clearly be seen by comparing the policies for the horizontal corrector (psch_mbd_fel02.02) and vertical corrector (pscv_mbd_fel02.02) in Fig. 8.

IV. POLICY TYPES STUDIED IN THIS WORK

As we previously mentioned, we did not find that an agent with policies that are normally distributed were very successful. Such agents frequently produced polices in which the mean value was outside the prescribed range of support. We studied three other policy types: beta distributions with α and β as parameters, beta distributions with mode and concentration as parameters, and the von Mises distribution. In the following each agent uses only a single policy type and, as such, we name the agents after the policy type it uses.

A. The beta agent

The beta distribution has a region of support of [0,1] and the policy distribution is given by

$$\pi(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \tag{9}$$

where $\Gamma(x)$ is the gamma function. The mode of this distribution is given by

$$m = \frac{\alpha-1}{\alpha+\beta-2}, \tag{10}$$

and the variance is given by

$$\sigma^2 = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}. \tag{11}$$

To ensure the distribution is unimodal, we do not allow either α or β to be less than 1 by parametrizing the policy parameters using model parameters. For the beta distribution the two types of parameters are given by $\alpha = 1 + \ln(1 + e^{\theta_\alpha})$ and a similar equation for β. The model parameters are $\{\theta_\alpha, \theta_\beta\}$. The Fisher matrix for this distribution is

$$\mathcal{I}(\alpha,\beta) = \begin{pmatrix} \psi^{(1)}(\alpha) - \psi^{(1)}(\alpha+\beta) & -\psi^{(1)}(\alpha+\beta) \\ -\psi^{(1)}(\alpha+\beta) & \psi^{(1)}(\beta) - \psi^{(1)}(\alpha+\beta) \end{pmatrix}, \tag{12}$$

where $\psi^{(m)}$ is the polygamma function of order m.

For the beta policy distribution, as the agent adjusts the values for the policy parameters using the update rule in Eq. (8), it is not independently adjusting the mode (the most likely value for the optimal setting) and the variance (the confidence in that prediction). As we will see later, this kind of agent will learn much slower than the von Mises agent, in which the mode and variance can be independently updated. Because of this difference, we attribute this slower learning to the fact that the mode and the variance cannot be updated independently.

B. The mode and concentration agent

To test the importance of separate control of mode and variance, we used an alternative parametrization of the distribution, which we call "mode and concentration" wherein we define a concentration $\kappa = \alpha + \beta$ and the mode is defined in Eq. (10). Solving these two equations we find $\alpha = m(\kappa-2)+1$ and $\beta = (1-m)(\kappa-2)+1$ and the policy distribution is given by

$$\pi(x) = \frac{\Gamma(\kappa)x^{m(\kappa-2)}(1-x)^{(1-m)(\kappa-2)}}{\Gamma(m(\kappa-2)+1)\Gamma[(1-m)(\kappa-2)+1]}. \tag{13}$$

The policy parameters are defined in terms of the model parameters in the following way: $m = 1/(1+e^{-\theta_m})$ (the sigmoid function) and $\kappa = 2 + e^{\theta_\kappa}$. This does not allow the concentration to go below 2 to retain a unimodal policy. The Fisher matrix for this distribution is

$$\mathcal{I}(m,\kappa) = \begin{pmatrix} \mathcal{I}_{11} & \mathcal{I}_{12} \\ \mathcal{I}_{12} & \mathcal{I}_{22} \end{pmatrix}, \tag{14}$$

with

$$\mathcal{I}_{11} = (\kappa-2)^2[\psi^{(1)}(\alpha) - \psi^{(1)}(\beta)],$$
$$\mathcal{I}_{12} = (\alpha-1)\psi^{(1)}(\alpha) - (\beta-1)\psi^{(1)}(\beta), \quad \text{and}$$
$$\mathcal{I}_{22} = -\psi^{(1)}(\kappa) + m^2\psi^{(1)}(\alpha) + (1-m)^2\psi^{(1)}(\beta),$$

where we have used α and β in most places for simplicity of the functional representation.

C. The von Mises agent

The final distribution we use is the von Mises distribution given by

$$\pi(x) = \frac{e^{\kappa\cos(x-\mu)}}{2\pi I_0(\kappa)}, \tag{15}$$

where $I_h(x)$ is the modified Bessel function of the first kind of order h. The distribution has region of support $[-\pi, \pi]$. The policy parameters are defined as $\mu = \theta_\mu$ and $\kappa = e^{\theta_\kappa}$. We chose to work with this distribution because it is symmetric, its mode and concentration can be adjusted independently, and because it can be generalized to any number of dimensions in the form of the von Mises-Fisher distribution. The Fisher matrix for this distribution is

$$\mathcal{I}(m,\kappa) = \begin{pmatrix} \frac{\kappa I_1(\kappa)}{I_0(\kappa)} & 0 \\ 0 & 1 - \frac{I_1(\kappa)}{\kappa I_0(\kappa)} - \left(\frac{I_1(\kappa)}{I_0(\kappa)}\right)^2 \end{pmatrix}. \tag{16}$$

Contrary to the previously mentioned features that we view as recommending the von Mises distribution, the region of support is periodic and, in particular to our use here, when the policy distribution comes close to one of the support boundaries the policy can "wrap around" to the other side of the interval. This feature will bias the policy distribution as the system being controlled is unlikely to have a similar characteristic, a notable exception might be a phase shifter in an FEL. In practice, we found that this was rarely a problem.

We previously stated that the normal distribution did not make a successful policy distribution in the present work due to bias caused by the mismatch between the regions of support of the distribution and the system being controlled [30], whereas we will see that the von Mises distribution performs well relative to the other distributions mentioned here. We attribute this difference to the finite region of support. Using a large learning rate to speed up training means that the mean of the normal distribution can quite easily leave the desired region of support. In fact, because the region of support of the normal distribution is infinite, there are many more values that the mean can take on that are not valid for the system being controlled. In contrast, the bias in the von Mises distribution is caused by the periodic region of support. Thus, whatever value the mean takes on, the policy is always contained within the region of

support. This does not rule out the use of a normal distribution in the context of policy gradient methods, but it does require more decisions about how to reconcile the normal distribution to the finite interval. For this work, and based on our early experience with the poor performance of the normal distributions, we opted to stick to policies with finite support, which seems a better fit for a control system.

V. SIMULATIONS

Before deploying the agent on the accelerator we performed a number of tasks in a simulated environment to reduce the amount of time spent interacting with the accelerator. This is useful because, in addition to the slow reaction of the accelerator as compared to a computer simulation, access to accelerator run time is itself a valuable resource that is typically given to users. In addition, simulations allow us to separate the characteristics of the agents we use from the characteristics of the accelerator at FERMI and the particular tasks we had the agent perform.

The goal of these simulations is to guide the tuning of hyperparameters and differentiate between a number of policy types. As the agents we deploy are model-free, they can be tested just as well using what we call a number guess environment as any accelerator simulation. This simple environment is fast and flexible, allowing us to test a number of the agent characteristics and tune the model hyperparameters.

In this environment the task is for the agent to guess a real number within a specified range in one dimension (in this case the range is $[-5, 5]$). The reward structure is given by a normal distribution as

$$r(x) = \exp\left(-\frac{(x-t)^2}{2\sigma_r^2}\right) - 1, \tag{17}$$

where t is the target number to guess and σ_r is the standard deviation of the reward. The reward is negative, so the agent will tend to move away from the areas of large negative reward and toward areas of (relatively) higher reward. This negative reward system is advantageous over a positive reward system that always has $r(x) \geq 0$, because that system could have large areas of the region of support, depending on the standard deviation of the reward function, for which the reward is near zero and there is no update to the policy [see Eq. (7)].

The first comparison of the three agents is shown in Fig. 3, the task is to guess the number $t = -3.5$. For this task the policy starts at $m_0 = -2.5$, within 10% of the full range of the number it should guess. The standard deviation of the reward is $\sigma_r = 0.1$, 1% of the full scale range, and the initial standard deviation of the policy is set to $\sigma_0 = 1.0$, 10% of the full scale range. This simulation is meant to represent a case in which the agent starts with some low-confidence knowledge of the correct policy by the relatively close proximity of m_0 to the target and the large policy variance.

The plots show the reward received as a function of step number. We expect to see the rewards increase for a successful agent whose policy is converging to a higher reward region of the number guess environment. Agents that fail to learn will show rewards near -1 for the entire trial.

From Fig. 3 we see that, regardless of the parameter settings, the beta agent does not learn a useful policy in the allowed number of steps. The mode and concentration agent appears to have lower variance than the von Mises agent, i.e., each of the five trials for each pair of parameters is more similar for the mode and concentration agent. On the other hand, the von Mises agent finds a sufficiently narrow policy such that the reward averages very close to zero at the end of the trial, while the mode and concentration agent converges to a value just below zero. This is an artifact of a numerical library used to compute the gamma function in the beta distribution, where the computation fails if the argument becomes too large. To sidestep this problem, the concentration is not allowed to become larger than about 9900, which limits how deterministic the policy can become. There is no such limit on the von Mises agent, so its policy becomes more deterministic and, thus, more successful at this task.

When the learning rate is large ($\alpha = 0.01$) the more successful agents show very fast convergence to a successful policy, although it can take time to search the range, as evidenced by the up to five thousand steps of lethargy that some of the trials exhibit. In addition, it is clear that the middle of the range for the learning rate performs best ($\alpha = 0.005$ to 0.01). It is also clear that the smaller value ($\lambda_\theta = 0.1$) for the eligibility trace memory result in lower variance between trials.

Because the goal of this work is to evaluate fast learning agents, we discard the beta agent at this point.

Comparison of Figs. 3 and 4 illustrates that, in general, narrower standard deviation of the reward and larger difference between the target and the initial mode of the policy leads to slower convergence (the latter will be described shortly). On the other hand, wider regions of improved reward, i.e., larger σ_r, improved the speed of learning. This is an intuitive result as narrow rewards are more difficult to find and larger differences between the initial mode and target require more exploration.

For the next simulation, shown in Fig. 4, we move the target to $t = -0.5$ (20% full range), increase the standard deviation of the reward to $\sigma_r = 1.0$ (10% of full range), and reduce the allowed number of steps per trial to 1000. In addition we now allow λ_θ to take on the values of 0 and 0.1 and the values of α are now ten times larger than the previous simulations. We see that, once again, the mid-range for the learning rate, α, is a compromise between faster learning speed and lower variance. Further, the

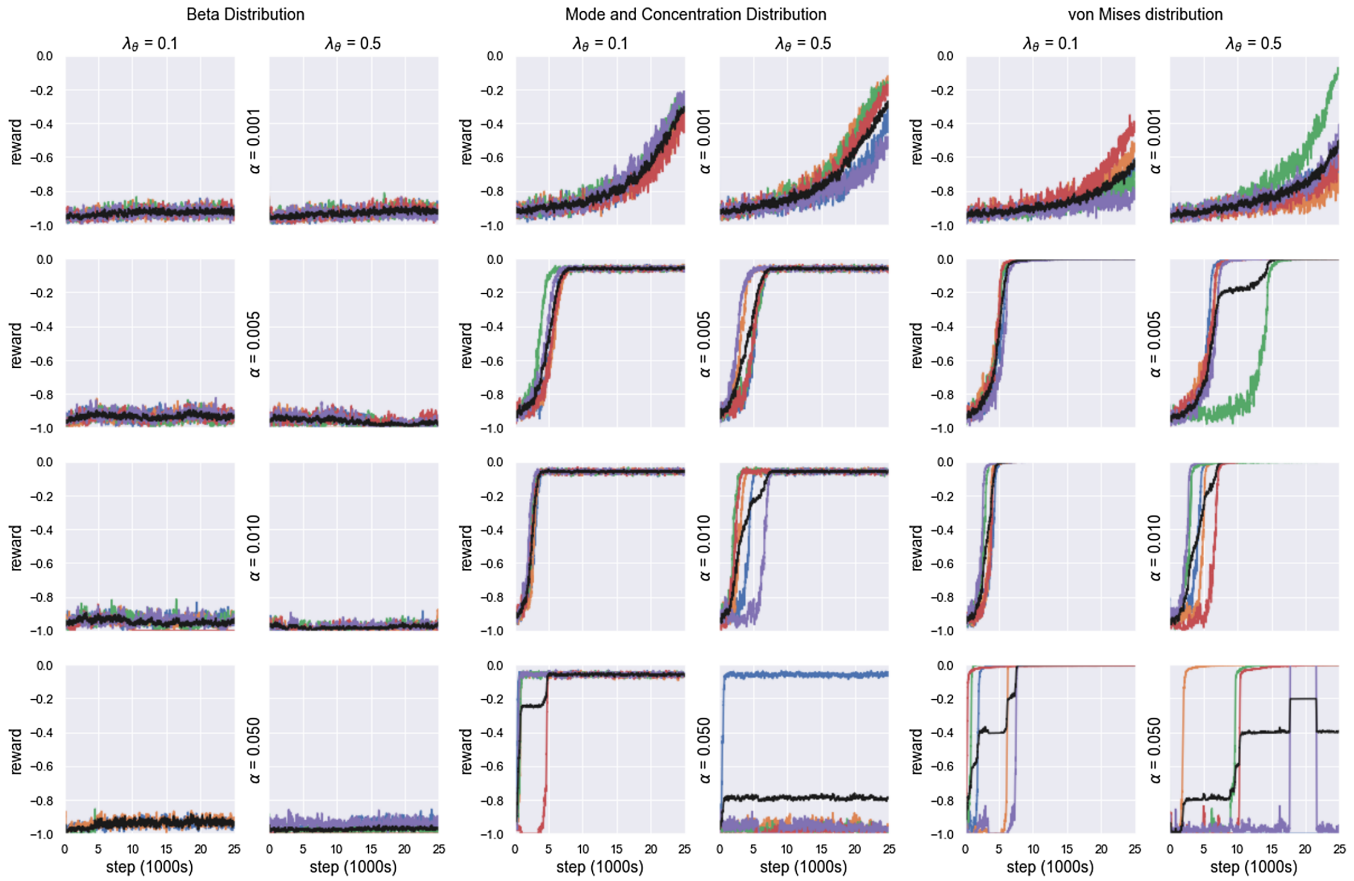

FIG. 3. Plot of the reward received by the agent versus step number. For this task the agent must "guess" the number −3.5. The standard deviation of the reward is 0.1. The agent begins the task with mode −2.5 and standard deviation 1.0. For each of the three agent types, we varied the eligibility trace memory parameter, λ_θ, and the learning rate, α, as shown in the figure. For each setting 5 trials (represented by the colored lines) were performed and the task was ended after 25,000 steps. A 100-step rolling average was performed to smooth the plots for ease of comparison. The black line is the average of all 5 trials in each plot.

eligibility trace memory parameter is best set to zero ("memoryless"). Finally, the plots with largest α and λ_θ showed no convergence, and would prematurely end the simulation via numerical errors and were labeled unstable.

Interestingly, we see that for this reward profile, both agents converge to a similarly well performing policy. In this task the reward region is large enough that the limited concentration allowed of the mode and concentration agent does not hinder policy convergence as much as the previous task.

For the next simulation, we add a noise term to the target variable by substituting $t \to t_0 + \Delta t$ in Eq. (17). Where t_0 is the target set at the beginning of the simulation and Δt is sampled from a uniform distribution, $\Delta t \sim U(-0.2, 0.2)$ and the target is moved to $t = 0.5$. In other words, noise allows the target to vary by 4% of the full range. Otherwise, the parameters for this simulation are the same as those for the previous simulation. The results for these simulations are shown in Fig. 5.

For this simulation, both agents perform better for the lower values of α. However, the mode and concentration agent is more sensitive to the eligibility trace memory parameter than is the von Mises agent. For all but the highest value of α, the convergence has been slowed down, taking approximately 50% more steps. For the highest α, both agents fail to learn a good policy often enough that they are unlikely to be useful.

A summary of the simulations performed in the number guess environment is given in Table I to allow comparison of the parameters used in the simulations that are not shown in the figures. What we see from these simulations is that the eligibility trace parameter is harmful and should be left at zero. This is because the trace allows the agent to receive rewards from previously "better" settings (i.e., settings for which the reward is nearer to zero) while it is currently making "worse" decisions (i.e., settings for which the reward is nearer −1). We also see that the learning rate can be set as high as 0.5 in some cases, but the best compromise between performance and learning speed appears to occur with a value of 0.1. In addition, we have seen that an agent acting in a one-dimensional control space can find a successful policy in

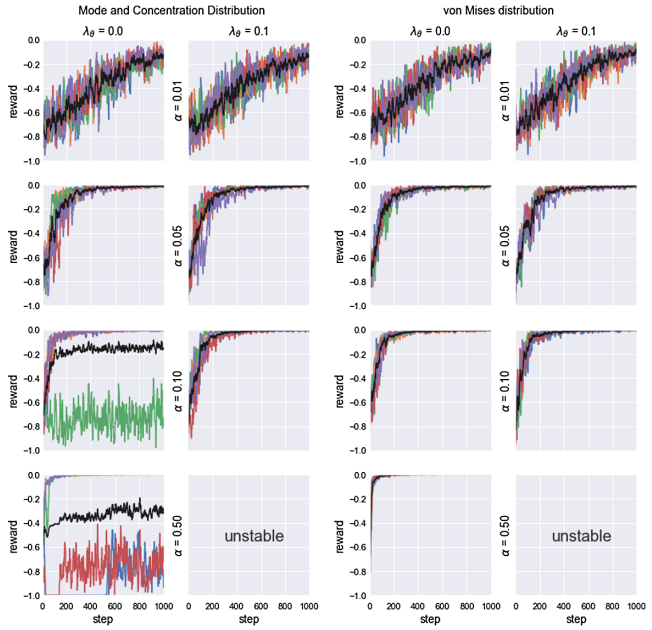

FIG. 4. Plot of the reward received by the agent versus step number. For this task the agent must guess the number −0.5. The standard deviation of the reward is 1.0. The agent begins the task with mode −2.5 and standard deviation 1.0. For each of the three agent types, we varied the eligibility trace memory parameter, λ_θ, and the learning rate, α, as shown in the figure. For each setting 5 trials (represented by the colored lines) were performed and the task was ended after 1000 steps. A 10-step rolling average was performed to smooth the plots for ease of comparison. The black line is the average of all 5 trials in each plot.

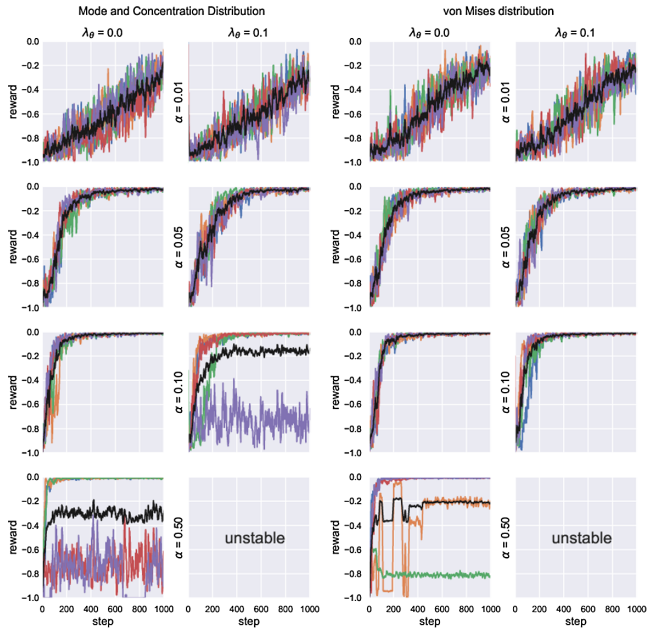

FIG. 5. This task has almost the same parameters as those given in Fig. 4, the only change is that the target is now 0.5 instead of −0.5. In addition to those parameters, the value of the target at each step is changed to include uniformly sampled noise that is updated at every step.

TABLE I. A summary table of all the simulations presented from the number guess environment. All simulations were done within a range of $[-5, 5]$.

Simulation	t	σ_r	m_0	σ_0	$\lvert\Delta t\rvert$	Figure
1	−3.5	0.1	−2.5	1.0	0.0	3
2	−0.5	1.0	−2.5	1.0	0.0	4
3	0.5	1.0	−2.5	1.0	0.2	5

a few hundred steps under the conditions that produced Figs. 4 and 5.

In the next section, we will use agents based on the von Mises distribution. It will be useful to see the evolution of the mean and standard deviation of this agent's policy during the number guess simulation, this is shown in Fig. 6 for the values of α that we will use most often. The behavior of the policy for the $\alpha = 0.1$ case shows the policy starting off at the pre-defined value and, relative to higher values of α, making slow and steady progress toward the target value, with relatively small overshoots, i.e., the policy does not go very far past the target value. In contrast, when $\alpha = 0.5$ the agent's policy can be seen moving to 1 standard deviation to the other side of the target value in the cases that eventually converge to the target value. Perhaps more interesting are the two cases where the policy becomes unstable. What is clear is that the concentration becomes very large (i.e., the variance becomes very small), this leads to large changes in the mode, which is what we refer to as "unstable."

This unstable behavior is due to the update rule for stochastic gradient ascent as given in Eq. (8). For this distribution the derivatives in that equation are given by

$$\frac{1}{\pi}\frac{\partial \pi}{\partial m} = \kappa \sin(a - m), \quad \text{and}$$
$$\frac{1}{\pi}\frac{\partial \pi}{\partial \kappa} = \cos(a - m) - \frac{I_1(\kappa)}{I_0(\kappa)}.$$

Here we use the variable a to mean the action taken for that particular update. What is clear from these equations is that if κ becomes very large the mode will change very quickly, but the concentration will still change with the right hand side of the update rule being of order unity. The sudden changes in the mode of two of the examples shown in Fig. 6 are because the agent has learned a poor solution and become to "sure" of it. This is the principal trade-off in determining where to set α. Too small and the agent learns too slowly or, at the very least, could learn faster. Too large and the agent might learn too much from its early choices. Because there is no method for selecting α *a priori*, the user must resort to scans and experience.

VI. EXAMPLES OF ACCELERATOR OPTIMIZATION

In this section we present demonstrations of the algorithm controlling the accelerator at FERMI@Elettra.

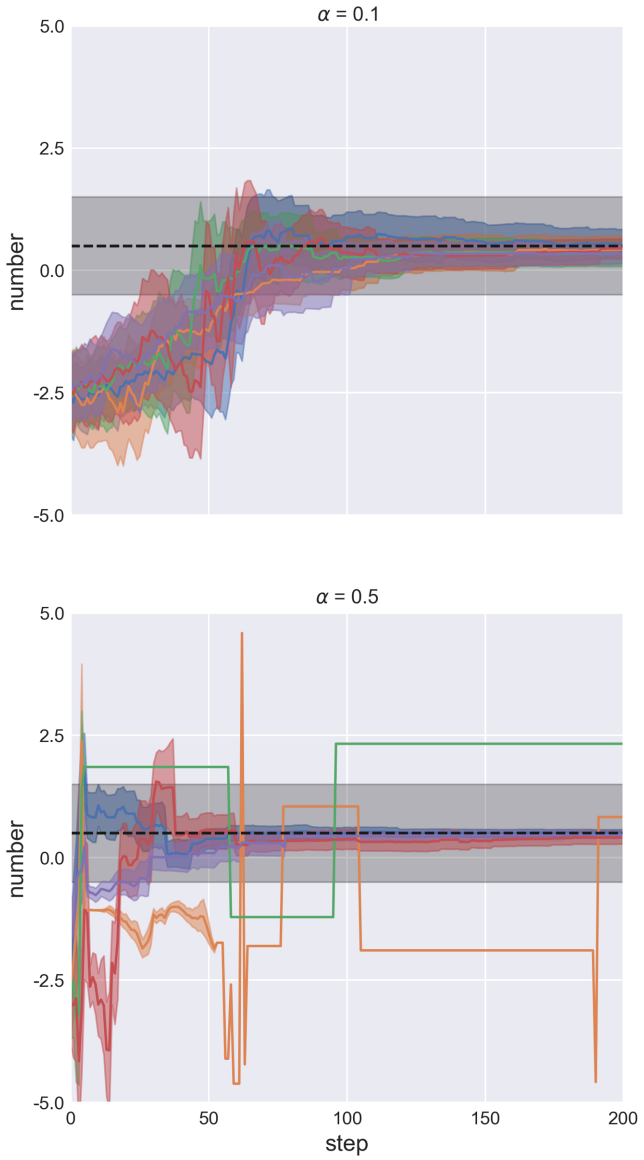

FIG. 6. The mean (solid colored lines) and standard deviation (shaded regions) for the von Mises agent during the number guess trials with $\alpha = 0.1$ and 0.5, and $\lambda_\theta = 0$ shown in Fig. 5. The colors in this figure correspond to those shown in Fig. 5. The black dashed line shows the target value, t, and the associated shading shows the standard deviation of the reward, σ_r.

In these demonstrations, the environment is very different and the agent is almost unchanged from the previous section. For the agent, we introduce a new parameter and force the learning rate to decay exponentially as a function of step number using the formula

$$\frac{\Delta\alpha}{\Delta t} = -\frac{\alpha}{\tau}, \tag{18}$$

where τ is the decay rate parameter. This small modification was done to keep the agent from reacting strongly to a fault after convergence, so that the operator had time to shut down the agent.

The environment is the accelerator and we no longer have the ability to set the reward schedule. To prevent the agent from becoming "satisfied" with only a mediocre improvement over the initial performance, we structure the reward in one of two methods. In both cases, when the agent is started it reads the target variable (the one it will optimize) for dozens of readings and takes the mean value of the readings, and stores this value as the target value, $I^\star$. At each step thereafter the reward is calculated as $I/I^\star - 1$, where I is the mean value of a new set of dozens of readings. The difference in the two methods is in how $I^\star$ is updated.

In the first method, which we call greedy, the target variable is simply set to the highest value encountered by the agent thus far during the run. We call the second method measured. In this method, after the agent is updated, if the new reward is greater than the target value, the target value is moved toward the new value using the formula

$$I^\star \leftarrow I^\star + 0.1(I - I^\star). \tag{19}$$

This slow approach to the new, higher value prevents the target value from being increased to an anomalously high value due to a chance fluctuation in the value of the target. A further benefit to this slowing of reward changes is that it stabilizes the reward for taking the same action on consecutive steps, as might happen when the agent is confident (low variance) in the control setting, but has not yet discovered the optimal setting. Both reward systems allow positive reward, in contrast to the number guess simulations in Sec. V where the reward is never positive.

A. Optimization of TeraFERMI

As a first demonstration of the optimizer, we used it to optimize the signal at the TeraFERMI beam line. The function of the TeraFERMI beamline is described elsewhere [31]. Briefly, the beam dump transport line after the FEL contains a screen that produces THz radiation as the electron beam passes through it on its way to the dump. A layout of the beam line is given in Fig. 7. While the agent is optimizing the signal, all other feedbacks in the TeraFERMI region of the beam line are disabled. The typical time for an operator to tune TeraFERMI is a few hours as the operator tries various combinations of magnet settings. Part of the reason for tuning taking this long is that the signal at TeraFERMI is quite sensitive to the details of FEL operation and the operator cannot adjust those systems or even hold them constant during operation. Even when the FEL users are not making intentional adjustments to the machine, the feedback systems there are working to keep the FEL pulse properties that the user requested stable, which can lead to changes in the TeraFERMI signal.

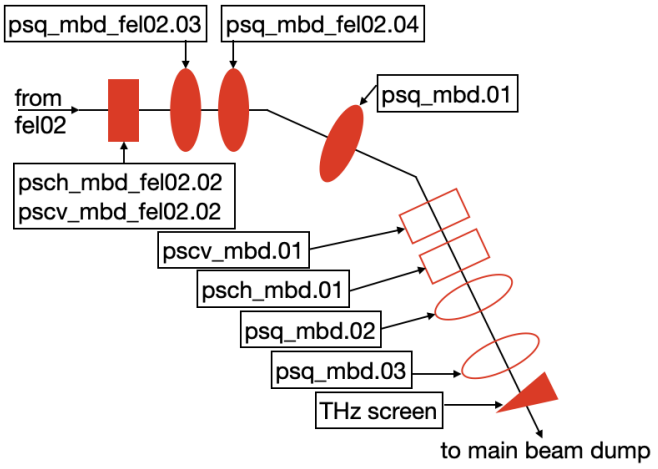

FIG. 7. Synoptic layout of the beam line after FEL02 showing the magnetic elements used for the optimization (filled shapes) and the magnetic elements left at their nominal values (empty shapes). The two bends are shown as vertices in the line.

The magnets used are psch_mbd_fel02.02, a horizontal corrector, pscv_mbd_fel02.02, a vertical corrector, and psq_mbd_fel02.03, psq_mbd_fel02.04 and psq_mbd.01, which are quadrupoles.

In this task, an operator tunes the magnets in the TeraFERMI beam line until a satisfactory signal is read on a pyro detector. For this experiment the operator reached 140k with approximately 13% standard deviation as shown in Fig. 8. The magnets to be controlled are then detuned by −0.3 A. This typically resulted in the signal on the pyro dropping by approximately 50%. The agent is then turned on and allowed to optimize the signal on the pyro within a range of ±3 A of the detuned setting. This latter constraint was used to reduce the likelihood of prolonged beam loss as the agent searched the available control settings, which would have interrupted simultaneous operations. Thusly, the agent begins its optimization within 10% of the allowed controls range of a known satisfactory solution. The target variable is updated with the measured method.

In these demonstrations, each step takes approximately 3.5 seconds, with that time dominated by the time the magnets take to settle at their new settings, which is signaled by the control system (not our algorithm). All of the runs for this test use the von Mises agent. For the sake of brevity, we will discuss three runs of the optimizer. The data for these runs is shown in Fig. 8.

In the first run we set $\alpha = 0.1$ and $\tau = 10000$. The optimizer recovers the target signal in about 45 minutes, while controlling the 5 magnets. The optimizer's confidence in the setting of all the magnets suggests that the signal is more sensitive to the quadrupole settings than the steering magnet settings. This result is consistent with both the operator's reported experience and the scaling of coherent transition radiation power with the fourth power of the beam spot size at the screen [32]. The agent finds

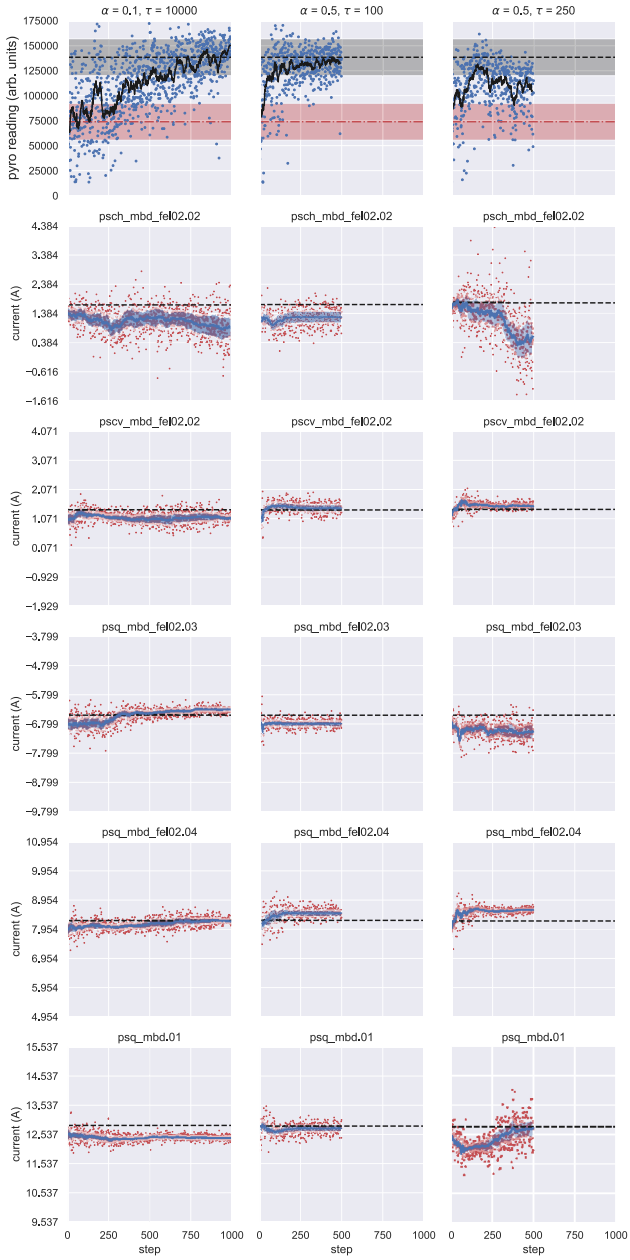

FIG. 8. Optimization runs for TeraFERMI. Each column represents a different run with the parameters given at the top of the column. The top row of plots shows the signal of the TeraFERMI pyro, while the remaining rows show the setting of the various magnets used for optimization. In the top row, the blue dots are the readings at each step; the black solid line is a rolling 20 point average of the readings; the black dashed line is the mean signal strength found by an operator with the surrounding shaded region showing the standard deviation of that measurement; and the red dash-dot line is the mean signal strength of the initial setting of the magnets after detuning with the surrounding shaded region showing the standard deviation of that measurement. In the remaining plots, the red dots are the magnet setting for each step; the blue line is the mean of the policy distribution; the shaded blue region is the standard deviation of the policy; and the black dashed line is the magnet setting found by the operator. 1000 steps takes approximately 1 hour.

slightly different settings for the magnets than the operator did. It is not clear if this difference is caused by machine drift while the optimizer was running or simply because it found a different nearby local maximum, but the output signal of TeraFERMI is similar in both cases.

In the next run, we set $\alpha = 0.5$ and $\tau = 100$, shown in the second column of Fig. 8. For these settings we expect that the initial changes to the policy will be quite large but in about 1000 steps the policy changes will be relatively modest. We see that the optimizer has almost converged after about 15 minutes (roughly 250 steps) and in the following 15 minutes the policy has found an approximately equivalent solution to that of the operator.

In contrast to the previous two successful runs, we show what an unsuccessful run looks like in the third column in Fig. 8. Although the agent failed to replicate the operator's performance in 30 minutes, it is still outperforming the detuned settings on average. It is possible that the agent became stuck in a local maximum, but the large standard deviation in the setting of quadrupole psq_mbd_fel02.02, which was quite narrow in the other runs suggests that there has been some underlying shift in the accelerator between the second run and the third run. We speculate that this is the case, because the agent has become less sure about the setting (i.e., the variance increases after a change in the mode that occurs at roughly step 300). This is in contrast to the unstable runs shown in Fig. 6, where the variance of the unstable agents becomes very small. The strength of the TeraFERMI signal depends on factors beyond the agent's control in these tests, which we could not investigate further due to user operations. We are reassured that, whatever underlying shift in the accelerator environment, the agent did not compound the problem in this case by reducing the target signal below the starting point.

B. Optimization of seed laser and electron beam overlap

In the previous section, we found that policy gradient methods can be used to optimize a particular signal when the initial conditions are near a maximum in the target variable. In this section, we demonstrate a related task, that is we show that the target signal can be maintained after the accelerator is perturbed. There is a wide variety of perturbations that the accelerator may experience from relatively slow changes in the environment, e.g., temperature changes in the accelerator hall or drift in a critical power supply, to fast changes, such as sudden failure of instrumentation. For experimental expedience, we focus on short time-scale perturbations here.

The task for the agent is to maintain (or recover) the output energy of a seeded HGHG free-electron laser when the overlap between the seed laser and the electron beam is perturbed [33]. This experiment was performed in the first HGHG stage of FEL2 at FERMI, a cascade HGHG free-electron laser [17]; the second HGHG stage remained off for this experiment. The FEL was operating at 36 nm with a 900 MeV electron beam and produced 400 μJ when tuned by an operator.

Typical operation of the first stage of the cascade FEL is to fix the position of the seed laser on two alignment screens, one upstream and one downstream, of the undulator where the seed laser and electron beam interact. Next, the electron beam is steered to overlap with the laser spot on the same two screens. After a little bit of optimization by an operator, feedbacks are enabled to prevent the seed laser and electron beam from wandering too far from each other during user operations [4].

The seed laser alignment system is two consecutive mirrors along the laser transport that has two control levels: coarse and fine. Because the fine level has limited range, the coarse alignment on the screens is performed by means of stepper motors. The fine level is performed by the piezo motors included in a feedback system.

For this demonstration we disable the feedback on the overlap, allow the agent to control the fine motors and perturb the FEL using the coarse motors. All other feedbacks operating on the electron beam and seed laser remain on. The target variable is the FEL energy as measured by an ionization monitor [16]. Note that the alignment screens cannot be inserted during FEL operation, so the agent and the operators have no knowledge of the overlap between the two beams on these screens. Unlike the example in the previous section, where the measured update was used, here the target variable is updated greedily. We made this decision because we want the agent to learn as much as possible from each step, because greedy updates are faster (i.e., take fewer steps) to punish suboptimal machine settings.

In Fig. 9, we show the performance of the agents during various operational difficulties, either human induced as previously described, or due to a fault in the accelerator system. For all of these runs we have taken $\tau = 10^8$ so that α is effectively constant throughout each run. To limit the control range, the fine adjustment level is allowed to operate within a range of ± 5000 steps from the initial value when the agent is started. Perturbations to the FEL performance are made phenomenologically by adjusting either the vertical or horizontal coarse motor on the downstream mirror while monitoring the FEL energy. In addition to some operator adjustment, the feedbacks were enabled in between runs to approximately begin at the same initial conditions in terms of FEL energy for each run.

The first run (labeled by (1) in Fig. 9) demonstrates a problem with using a greedy target variable update. We can see in Fig. 10 that it is the only run to begin with unusually large (relative to the other runs) positive rewards in the first two steps. Since the agent updated its target expectations greedily the subsequent steps were relatively strongly punished and the agent went searching for a different maximum. The maximum it found was about 13% (~ 50 μJ) lower than where it started, but because it just happened to get strongly rewarded at the very beginning of

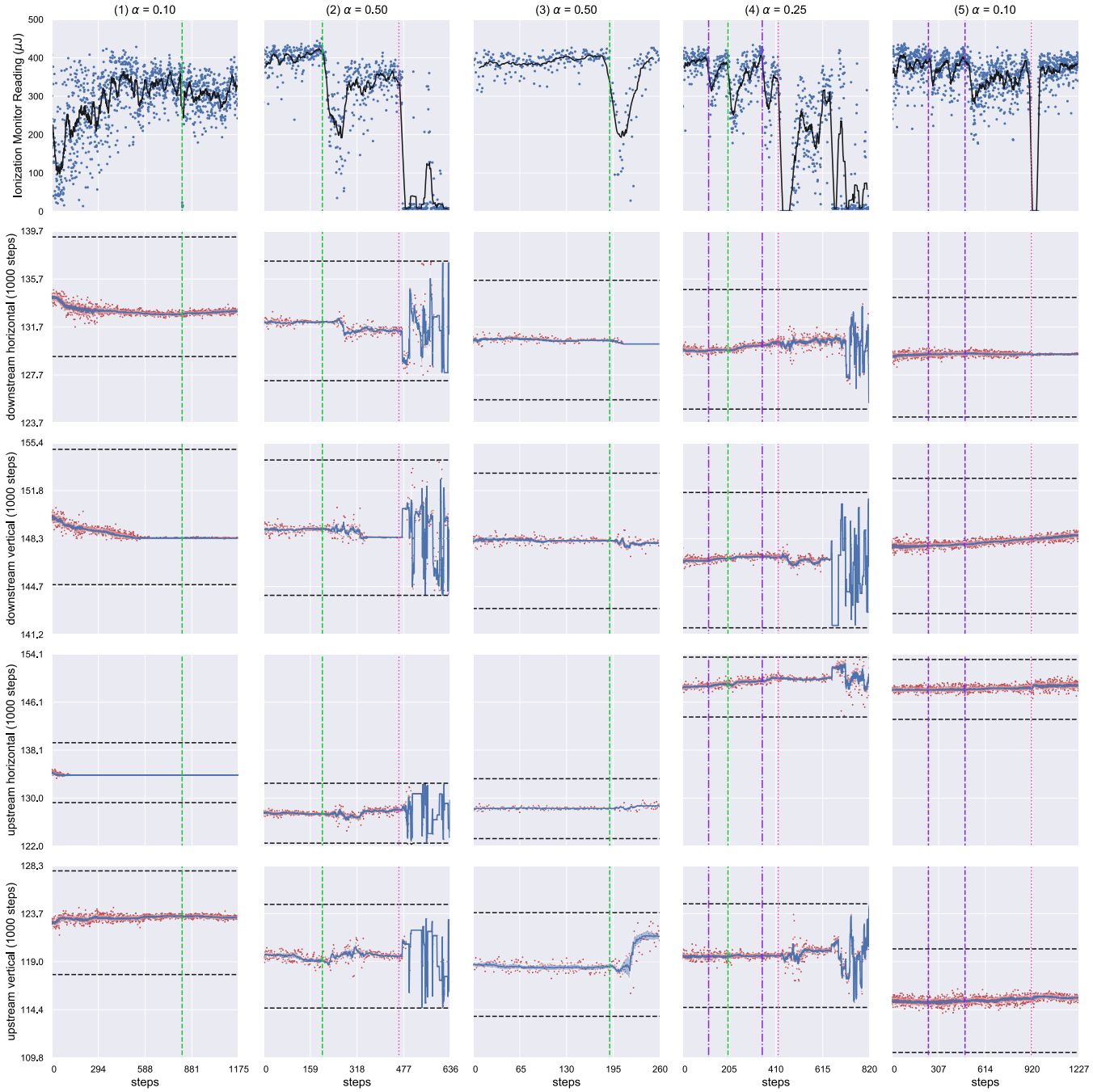

FIG. 9. Maintenance of a single stage HGHG FEL using policy gradient methods. The agent controls the pointing of the seed laser through the modulator undulator where it interacts with the electron beam. The target variable to maximize is the FEL energy, as measured by an ionization monitor. Each column represents a different run and the learning rate for each run is shown at the top of each column. The first row is the FEL pulse energy. The remaining rows are the setting (in steps) of the four piezo motors that control the seed laser pointing. The vertical lines represent changes made to the FEL during the run: green, dashed lines represent the seed laser being moved horizontally by a human operator; purple, dash-dot lines represent the seed laser being moved vertically by a human operator; and the magenta dotted lines represent a fault in the accelerator that resulted in the electron beam being shut off by turning off the cathode laser. The black dashed horizontal lines show the allowed range of operation for each device and each run.

its learning, due to statistical fluctuations in the FEL output, it was not satisfied with where it started and found itself in a different local maximum. We can clearly see that the downstream motors moved several thousand steps to find this new maximum and the agent is quite certain (low variance) this is the best performance it can find. If the policies had begun with a smaller variance, it might have prevented the agent from straying so far from the

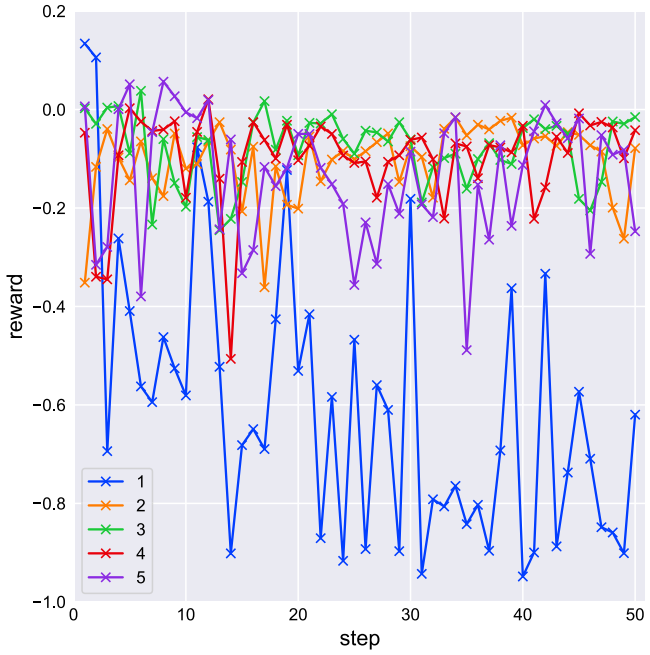

FIG. 10. The reward received for the first 50 steps for all the runs shown in Fig. 9. Of note is that run 1 is the only run to start with an unusually large positive reward and the only one where the agent immediately begins to look for a better optimum.

initial, operator-tuned optimum. After about 800 steps, the seed laser pointing is slightly perturbed (the FEL energy drop is within the statistical noise) and the agent makes small corrections to the motors and recovers the previous performance.

The second run, with much larger learning rate, does not wander from the initial optimum. When it is strongly perturbed (the FEL energy drops by about 50%) it very quickly finds the same maximum that was found during the first run. The third run is perturbed in a similar way, but it manages to find its way back to the initial, more intense optimium. We surmise, based on these similarities, that there are two slightly different optimums nearby each other and that which optimum the agent ends at depends on which actions are taken when sampling the policy distribution. The setting of the motors changes dramatically between the first two runs, which suggests that simple control systems that return to the last known "good" position for these devices is not sufficient to restore performance of the FEL. Indeed, this is a regular experience of the FERMI operations team.

Runs 4 and 5 show similar behavior. The agent continues to recover the FEL energy after a variety of perturbations of the seed laser pointing.

Perhaps the most interesting feature shown in Fig. 9 is the reaction of the agents to changes to the accelerator due to faults in the RF power system. These faults are related to the machine protection system wherein some monitored parameter exceeds a safe operational range and the electron beam is disabled at the cathode while automatic recovery systems intervene to correct the problem. Three different events occurred, one each during runs 2, 4, and 5 wherein α was 0.50, 0.25, and 0.10, respectively. The faults last for approximately the same amount of time (although that is hard to see in the case of run 2) and only the agent with $\alpha = 0.10$ was able to recover. This result is consistent with the results for the simulations with a noisy target in Fig. 5. While it is difficult to extrapolate from such a small dataset, the consistency is reassuring.

We assume that the agents with larger α learn too much while the accelerator is in the fault state. A simple way to address this problem would be to disable learning during a fault state. This can be accomplished within the framework of the policy gradient method by allowing two states, "normal" and "fault," or it can be done at the control system level where the agent is not allowed to learn during fault conditions. Either way, this is the subject of future work.

C. Optimization of FEL energy with mixed controls

In the final example of policy gradient methods the goal is to show the agent tuning a system using very different tools. In the previous two examples the agent was changing either magnets or piezo motors. In this example, the agent will be adjusting the R_{56} of the HGHG FEL (through the current in the dispersion-generating magnets) and the relative time of arrival between the seed laser and the electron beam (through a mechanical delay system). The seed laser wavelength is 251 nm and the undulators are tuned to lase at the 7th harmonic, approximately 36 nm. The electron beam is ~2 ps long, the seed laser is ~100 fs long and the jitter in the relative time of arrival is ~50 fs. For simplicity we assume that the slice energy spread and mean energy are both constant along the bunch.

As usual, the FEL is tuned by a human operator and then detuned to allow the agent to operate. For these tasks we have used greedy target updates and reset the FEL to its initial settings between each run. After the FEL was reset to its initial settings, a feedback is allowed to optimize the FEL output energy. This procedure allowed us to detune from a local optimum for each run as the control systems drifted in the intervening time. Because of the large inductance of the dispersive magnet, each step takes approximately 5 seconds. Once again we take $\tau = 10^8$.

The task for the agent is to maximize the FEL energy when the seed laser energy is reduced from 20 μJ to 10 μJ, i.e., it is reduced by half. The bunching factor in an HGHG FEL is given by [33]

$$b = J_h\left(\frac{\Delta E}{E_0} h k_r R_{56}\right) \exp\left[-\frac{1}{2}\left(\frac{\sigma_E}{E_0} h k_r R_{56}\right)^2\right], \quad (20)$$

where $J_h(x)$ is the Bessel function of the first kind of order h, ΔE is the strength of the energy modulation of the electron beam induced by the seed laser, E_0 is the energy of

the electron beam (900 MeV), R_{56} is the longitudinal dispersion applied after the energy modulation, $k_r = 2\pi/\lambda_r$ is the wave number of the energy modulation ($\lambda_r = 251$ nm), σ_E is the slice energy spread of the electron beam (∼100 keV), and h is the harmonic number. The dispersive element is a 4-pole chicane for which the dispersive strength depends quadratically on the current: $R_{56} \propto I_c^2$.

Running in the exponential gain regime, but not saturating, we expect that the FEL energy will be proportional to b^2 [34]. As such, even though the agent might be able to find the first maximum in $J_h(x)$, the larger R_{56} will reduce the output energy of the FEL by

$$f = \exp\left[-(\sigma h k_r)^2 (R_{56,2}^2 - R_{56,1}^2)\right],$$

with $R_{56,2} = \sqrt{2} R_{56,1}$, $R_{56,1} = 36.6$ μm ($I_c = 45$ A), and the FEL parameters as given above, we estimate that the maximum possible signal that the FEL can produce after reducing the seed laser energy is 60% of the original signal. The FEL energy was about 375 μJ when initially tuned by the operator (before the seed laser energy is reduced).

We show two tests of the agent's performance at this task in Fig. 11. In the previous examples of optimization, we found that $\alpha = 0.5$ did well, but in this case the optimization was unstable. Our conjecture is that this is due to the relatively large jitter in the seed delay as compared to the other control systems used here. The seed delay jitter of 50 fs covers 2.5% of the total available tuning range (the length of the electron beam), while the precision of the current setting in the magnets is well sub-mA and the piezo motors should only miss a few steps in the ten thousand step range used in the previous experiment. Thus, we expect the seed delay to act similarly to the noisy number guess simulation presented in Sec. V.

From the number guess simulations we conclude that the strategy to improve agent performance is to reduce the learning rate. For the two presented simulations we start the agent before turning down the seed laser energy. From the first example, it is clear that $\alpha = 0.05$ is too slow because the agent takes a very long time to learn, and only recovers a small fraction of the FEL energy after more than 400 steps.

In the second run with $\alpha = 0.10$, the agent learns much faster and after only 100 steps has recovered a bit less than half of the FEL energy lost. Unfortunately, at this point the accelerator went in to fault for a short period (14 steps, about a minute). Afterward, the agent recovers its original rate of improvement of the FEL energy, and improves the signal until it appears to reach a limit. After the fault, there is a 600 fs shift in the optimal setting of the seed delay. This kind of shift is to be expected because faults cause some part of the rf system to be altered as the fault is recovered

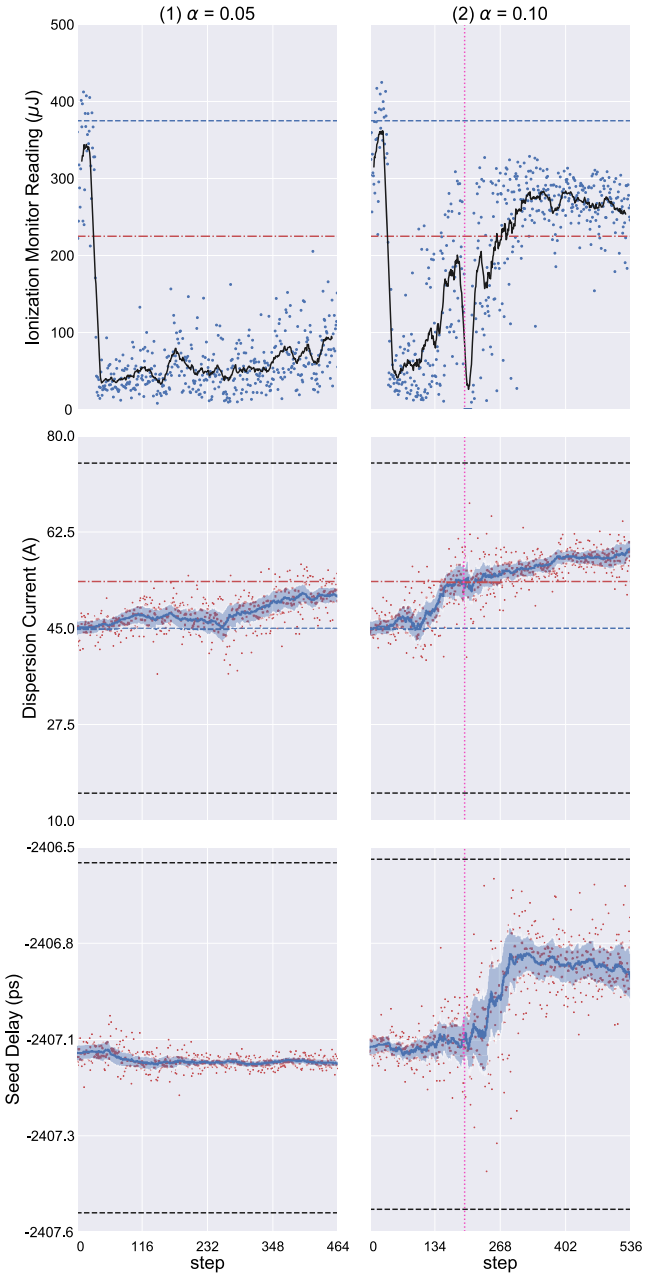

FIG. 11. Optimization of an HGHG free-electron laser using the dispersive strength and seed laser delay. The two columns show two different trials with the learning rate given at the top of each column. The top row shows the FEL energy as measured by an ionization monitor wherein the blue dots are the mean reading after each step and the black line is a rolling 20-point average. The remaining panels show the policy distribution of the different control parameters. The blue dashed lines show the settings found by a human operator, the red dash-dot lines show the settings (if any) predicted by a simplified HGHG model and the black dashed lines show the allowed range of operation of the control parameters. The dotted magenta, vertical lines show the occurrence of a fault in the accelerator system, whereafter the beam is temporarily disabled.

and when the system comes back online there are bound to be changes in the timing. These small changes are sometimes recovered by the feedback system.

It is tempting to say that the variance in the parameter settings might have been used to predict the fault before it happened, as the variance in both parameters appears to grow before the fault occurs. However, the large change in the target variable during this time period means that the agent should not have high confidence (low variance) in its choice of seed delay setting regardless of the oncoming fault state.

VII. CONCLUSIONS

As with most machine learning systems, it is difficult to make observations about a model that generalize to another application or accelerator, even when they are similar. However, we make a few general remarks to summarize our findings about policy-gradient methods using minimum policy gradient learning.

Simulations are important to the success of the agents. Magnets react slowly, accelerator systems are noisy and hyperparameters are notoriously hard to select *a priori* in machine learning models. By investigating agent performance using simulations we were able to choose an agent that appeared most likely to learn quickly as well as choose a practically useful range for the learning rate.

In contrast to supervised learning, the reinforcement learning model we use here does not require a great deal of prior information about what constitutes a desirable setting of the parameters under its control, i.e., we did not need a labeled dataset. It is not only model-free, but also requires very little information to get started. It was able to improve or maintain the performance of the accelerator by tuning the controlled parameters up to approximately 25% of the allowed range away from the initial setting using simple target update rules, arbitrary limits to the parameters, and some prior estimate for a useful value for the learning rate. This feature of the policy-gradient method means that an agent might be deployed where information about a desirable accelerator setting is sparse. Of course, like any other machine learning model, it should be designed not to drive the accelerator into an unsafe state if such information is available, for example, magnet settings that cause unacceptably large beam losses. If not available, signals of unsafe operation can be included in the reward function. In addition, the agent can be used to explore a parameter space and its confidence in the setting of the various control systems might be used to select important control parameters and eliminate parameters which do not appear useful to the task at hand.

We demonstrate agents performing two different accelerator-relevant tasks at the FERMI FEL: machine optimization and machine stabilization. These tasks are performed using a variety of different accelerator-relevant control systems: three kinds of magnet, piezo motors for laser alignment and a mechanical delay stage for a seed laser. Two different target variables are used: the output energy of an HGHG FEL and the amount of Terahertz radiation produced at the TeraFERMI beamline. The model-free agent is not altered between these tasks, except to explore the agent's capabilities (as described herein) and allow it to communicate properly with the different systems.

Reinforcement learning techniques such as the one presented in this work complement other machine learning and optimization methods currently being pursued in the context of accelerator physics. For example, RL might be used to complete the tuning of an accelerator that has been arrived at via Gaussian processes (GP) and the data generated during the exploration of the parameter space can then be used to improve the GP model. Adroit combination of different machine learning models can lead to systems of models that build on each other's strengths [35]. Policy-gradient methods might thusly find expanding use in accelerator physics.

ACKNOWLEDGMENTS

The authors thank the FERMI team for preparing the FEL and linac for data acquisition and numerous suggestions on which features to observe and parameters to control.

[1] O. Vinyals, I. Babuschkin, W. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. Agapiou, M. Jaderberg, and D. Silver, Grandmaster level in starcraft ii using multi-agent reinforcement learning, Nature (London) **575** (2019).

[2] C. Berner *et al.*, Dota 2 with large scale deep reinforcement learning, arXiv:1912.06680.

[3] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in *Advances in Neural Information Processing Systems 12*, edited by S. A. Solla, T. K. Leen, and K. Müller (MIT Press, Cambridge, MA, 2000), pp. 1057–1063.

[4] G. Gaio, N. Bruchon, M. Lonza, and L. Saule, Advances in automatic performance optimization at FERMI, in *Proceedings, 16th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2017): Barcelona, Spain, October 8-13, 2017* (2018), p. TUMPA07, https://doi.org/10.18429/JACoW-ICALEPCS2017-TUMPA07.

[5] X. Huang, J. Corbett, J. Safranek, and J. Wu, An algorithm for online optimization of accelerators, Nucl. Instrum. Methods Phys. Res., Sect. A **726**, 77 (2013).

[6] M. Aiba, M. Böge, N. Milas, and A. Streun, Random walk optimization in accelerators: Vertical emittance tuning at SLS, Conf. Proc. **C1205201**, 1230 (2012), https://accelconf.web.cern.ch/IPAC2012/papers/tuppc033.pdf.

[7] J. Duris, D. Kennedy, A. Hanuka, J. Shtalenkova, A. Edelen, P. Baxevanis, A. Egger, T. Cope, M. McIntire, S. Ermon, and D. Ratner, Bayesian Optimization of a Free-electron Laser, Phys. Rev. Lett. **124**, 124801 (2020).

[8] A. Scheinker, D. Bohler, S. Tomin, R. Kammering, I. Zagorodnov, H. Schlarb, M. Scholz, B. Beutner, and W. Decking, Model-independent tuning for maximizing free electron laser pulse energy, Phys. Rev. Accel. Beams **22**, 082802 (2019).

[9] N. Bruchon, G. Fenu, G. Gaio, M. Lonza, F. A. Pellegrino, and L. Saule, Free-electron laser spectrum evaluation and automatic optimization, Nucl. Instrum. Methods Phys. Res., Sect. A **871**, 20 (2017).

[10] C. Emma, A. Edelen, M. J. Hogan, B. O'Shea, G. White, and V. Yakimenko, Machine learning-based longitudinal phase space prediction of particle accelerators, Phys. Rev. Accel. Beams **21**, 112802 (2018).

[11] A. Bartnik *et al.*, Cbeta: First Multipass Superconducting Linear Accelerator with Energy Recovery, Phys. Rev. Lett. **125**, 044803 (2020).

[12] 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, in *2019 23rd International Conference on Mechatronics Technology (ICMT)* (2019), pp. 1–6, https://ieeexplore.ieee.org/document/8932150.

[13] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual* (CreateSpace, Scotts Valley, CA, 2009).

[14] P. Virtanen *et al.*, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nat. Methods **17**, 261 (2020).

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Scikit-learn: Machine learning in Python, J. Machine Learning Research **12**, 2825 (2011), https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html.

[16] M. Zangrando, A. Abrami, D. Cocco, C. Fava, S. Gerusina, R. Gobessi, N. Mahne, E. Mazzucco, L. Raimondi, L. Rumiz, C. Svetina, and F. Parmigiani, The photon beam transport and diagnostics system at FERMI@Elettra, the Italian seeded FEL source: commissioning experience and most recent results, in *SPIE Optical Engineering + Applications*, edited by S. P. Moeller, M. Yabashi, and S. P. Hau-Riege (SPIE, 2012), pp. 850404–8, https://www.spiedigitallibrary.org/conference-proceedings-of-spie/8504/850404/The-photon-beam-transport-and-diagnostics-system-at-FERMIElettra-the/10.1117/12.929749.short.

[17] E. Allaria *et al.*, Two-stage seeded soft-X-ray free-electron laser, Nat. Photonics **7**, 913 (2013).

[18] This parameter controls how much the agent weights future rewards for the purposes of computing the current reward, as shown in the expected discounted reward [Eq. (2)]. If $\gamma = 0$, the expected discounted reward would be only computed at the current step, t. Alternatively, the somewhat special case of $\gamma = 1$ is covered in the text after Eq. (2), when we discuss the meaning of the horizon, h. In practice, the discount factor doesn't change the ordering of policies in the continuing tasks we consider here, see section 10.4 of [19].

[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (The MIT Press, Cambridge, MA, 2018).

[20] 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).

[21] R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. **8**, 229 (1992).

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, Playing Atari with Deep Reinforcement Learning, arXiv:1312.5602.

[23] Epsilon-greedy refers to a policy in which the agent generates a number between 0 and 1. If that number is less than epsilon, the agent takes a random action. Otherwise, it takes the action which has previously returned the highest reward.

[24] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, Deterministic policy gradient algorithms, in *Proceedings of the 31st International Conference on International Conference on Machine Learning—Volume 32*, ICML'14 (JMLR.org, 2014), pp. 387–395, http://proceedings.mlr.press/v32/silver14.html.

[25] It is clear that this happens when going from Eq. (5) to Eq. (6) because we assume that the agent will follow the new policy after the update. With the policy now fixed the order in which the states are visited does not matter because the agent has nothing to learn from them. Section 10.4 of [19] gives a similar symmetry argument.

[26] J. Peters, S. Vijayakumar, and S. Schaal, Natural actor-critic, in *Machine Learning: ECML 2005*, edited by J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, and L. Torgo (Springer Berlin Heidelberg, Berlin, Heidelberg, 2005), pp. 280–291.

[27] Adroit choice for the initial values of the parameters allows the user to incorporate information they have into the agent before it starts to learn using Eq. (8). Herein, we simply use zero.

[28] Z. Huang, J. Wu, and T. Shaftan, Microbunching instability due to bunch compression, ICFA Beam Dyn. Newslett. **38**, 37 (2005), https://icfa-usa.jlab.org/archive/newsletter/icfa_bd_nl_38.pdf.

[29] A. L. Edelen, S. G. Biedron, B. E. Chase, D. Edstrom, S. V. Milton, and P. Stabile, Neural networks for modeling and control of particle accelerators, IEEE Trans. Nucl. Sci. **63**, 878 (2016).

[30] P.-W. Chou, D. Maturana, and S. Scherer, Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution, in *Proceedings of the 34th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 70, edited by D. Precup and Y. W. Teh (PMLR, 2017), pp. 834–843, https://dl.acm.org/doi/10.5555/3305381.3305468.

[31] A. Perucchi, S. Di Mitri, G. Penco, E. Allaria, and S. Lupi, The terafermi terahertz source at the seeded fermi free-electron-laser facility, Rev. Sci. Instrum. **84**, 022702 (2013).

[32] A. Tremaine, J. Rosenzweig, S. Anderson, P. Frigola, M. Hogan, A. Murokh, C. Pellegrini, D. Nguyen, and R. Sheffield, Measured free-electron laser microbunching

using coherent transition radiation, Nucl. Instrum. Methods Phys. Res., Sect. A **429**, 209 (1999).

[33] L. H. Yu, Generation of intense uv radiation by subharmonically seeded single-pass free-electron lasers, Phys. Rev. A **44**, 5178 (1991).

[34] K.-J. Kim, Z. Huang, and R. Lindberg, *Synchrotron Radiation and Free-Electron Lasers* (Cambridge University Press, Cambridge, England, 2017).

[35] P. Domingos, A few useful things to know about machine learning, Commun. ACM **55**, 78 (2012).