No More Pesky Hyperparameters: Offline Hyperparameter Tuning for RL

Anonymous Author(s)
Affiliation
Address
email

Abstract

The performance of reinforcement learning (RL) agents is sensitive to the choice of hyperparameters. In real-world settings like robotics or industrial control systems, however, testing different hyperparameter configurations directly on the environment can be financially prohibitive, dangerous, or time consuming. We propose a new approach to tune hyperparameters from offline logs of data, to fully specify the hyperparameters for an RL agent that learns online in the real world. The approach is conceptually simple: we first learn a model of the environment from the offline data, which we call a calibration model, and then simulate learning in the calibration model to identify promising hyperparameters. We identify several criteria to make this strategy effective, and develop an approach that satisfies these criteria. We empirically investigate the method in a variety of settings to identify when it is effective and when it fails.

1 Introduction

Reinforcement learning (RL) agents are extremely sensitive to the choice of hyperparameters that regulate speed of learning, exploration, degree of bootstrapping, amount of replay and so on. The vast majority of work RL is focused on new algorithmic ideas and improving performance—in both cases assuming near optimal hyperparameters. Indeed the vast majority of empirical comparisons involve well tuned implementations or reporting the best performance after a hyperparameter sweep. Although reasonable progress has been made towards eliminating the need for tuning with adaptive methods [White & White, 2016, Xu et al., 2018, Mann et al., 2016, Zahavy et al., 2020, Jacobsen et al., 2019, Kingma & Ba, 2014, Papini et al., 2019], widely used agents employ dozens of hyperparameters and tuning is critical to their success in practice [Henderson et al., 2018].

The reason domain specialization and hyperparameter sweeps are possible—and perhaps why our algorithms are so dependent on them—is because most empirical work in RL is conducted in simulation. Simulators are critical for research because they facilitate rapid prototyping of ideas and extensive analysis. On the other hand, simulators allow us to rely on features of simulation not possible in the real world, such as exhaustively sweeping different hyperparameters. Often, it is not acceptable to test poor hyperparameters on a real system that could cause serious failures. In many cases interaction with the real system is limited, or in more extreme cases only data collected from a human operator is available. Recent experiments confirm significant sensitivity to hyperparameters is exhibited on real robots as well [Mahmood et al., 2018]. It is not surprising that one of the major roadblocks to applied RL in the wild is extreme hyperparameter sensitivity.

Fortunately, there is an alternative avenue to evaluating algorithms without running on the real system: using previously logged data under an existing controller (human or otherwise). This offline data provides some information about the system which could be used to evaluate and select

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

hyperparameters without interacting with the real world. Hyperparameters are general, and can even include a policy initialization that is adjusted online. We call this problem Data2Online.

This problem setting differs from the standard offline or batch RL setting because the goal is to select *hyperparameters* offline for the agent use to *learn online in deployment*, as opposed to learning a *policy* offline. Typically in offline RL a policy is learned on the batch of data, using a method like Fitted Q Iteration (FQI) [Ernst et al., 2005, Riedmiller, 2005, Farahmand et al., 2009], and the resulting fixed policy is deployed. Our setting is less stringent, as the policy is learned and continually adapts during deployment. Intuitively, selecting just the hyperparameters for further online learning should not suffer from the same hardness problems as offline RL (see Wang et al. [2020] for hardness results), because the agent has the opportunity to gather more data online and adjust its policy. Even in the offline batch RL setting the hyperparameters of the learner must be set and most approaches either unrealistically use the real environment to do so [Wu et al., 2019] or use the action values learned from the batch to choose amongst settings [Paine et al., 2020] with mixed success.

We propose a novel strategy—to the best of our knowledge the first—to use offline data for selecting hyperparameters. The idea is simple: we use the data to learn a *calibration model*, and evaluate hyperparameters in the calibration model. Learning online in the calibration model mimics learning in the environment, and so should identify hyperparameters that are effective for online learning performance. The calibration model need not be a perfect simulator to be useful for identifying reasonable hyperparameters, whereas learning a transferable policy typically requires accurate models.

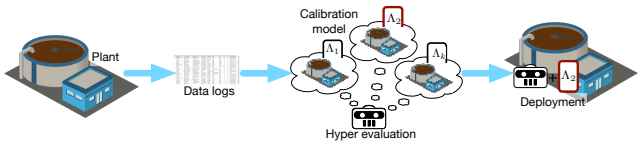

Figure 1: Data2Online: each hyperparameter setting is denoted by Λ.

Consider designing a learning system for controlling a water treatment plant, given only a set of data logs visualized in Figure 1. We want an agent to control pump speeds, mixing tanks and chemical treatments to clean the water with minimal energy usage—but how do we set the learning rate and other hyperparameters of this agent? We can learn a calibration model offline from logs of data previously collected while human operators controlled the plant. From there the calibration model can be treated like any simulator to develop a learning system, including setting the hyperparameters for learning in deployment.

In this paper, we first introduce our offline hyperparameter strategy, and outline conditions on the calibration model and learning agents for this strategy to be effective. We then investigate the approach in three problems with different types of learning dynamics, two different learning agents, under different offline data collection policies, and with ablations on the key components of our proposed calibration model. We then develop a more efficient method to search for hyperparameters in the calibration model, based on the cross-entropy method, and show that this addition can significantly improve over a hyperparameter grid search, as well as reducing the burden on the designer to design sets of hyperparameters to sweep.

2 Related Problem Settings

Offline RL involves learning from a dataset, but their hyperparameter selection approaches are quite different due to the fact that they deploy fixed policies. Yang et al. [2020] introduce the Offline Policy Selection problem and use an offline RL algorithm to learn and evaluate several different policies corresponding to different hyperparameter settings. The key difference is those hyperparameters will never be deployed in an online learning system, as we do in this paper. Some offline RL work examines learning from data, and then adapting the policy online [Ajay et al., 2020], including work that alternates between data collection and high confidence policy evaluation [Chandak et al., 2020a,b]. Our problem is complementary to these, as a strategy is needed to select their hyperparameters.

In Sim2Real the objective is to construct a high fidelity simulator of the deployment setting, and then transfer the policy and, in some cases, continue to fine tune in deployment. We focus on learning the calibration model from collected data, whereas in Sim2Real the main activity is designing and iterating on the simulator itself [Peng et al., 2018]. Again, however, approaches for Data2Online can be seen as complementary, and can highlight how to use the simulator developed in Sim2Real to pick hyperparameters for fine tuning. It could suggest design decisions in the developed simulator

for which hyperparameter selection for online learning is more effective. Further, if the simulator is expensive to sample, then our approach could be used to select hyperparameters.

Domain adaptation in RL involves learning on a set of source tasks, to transfer to a target task. The most common goal has been to enable zero-shot transfer, where the learned policy is fixed and deployed in the target task [Higgins et al., 2017, Xing et al., 2021]. Our problem has some similarity to domain adaptation, in that we can think of the calibration model as the source task and the real environment as the target task. Domain adaptation, however, is importantly different than our Data2Online problem: (a) in our setting we train in a *learned* calibration model not a real environment, and need a mechanism to learn that model (b) the relationship between our source and target is different than the typical relationship in domain adaptation and (c) our goal is to select and transfer hyperparameters, not learn and transfer policies.

Learning from demonstration (LfD) and imitation learning involve attempting to mimic or extract a policy at least as good as a demonstrator. If the agent is learning to imitate online, then it is unrealistic to assume the demonstrator would generate enough training data required to facilitate hyperparameter sweeps. If the learner's objective is to imitate from a dataset, then this is exactly the problem study of this paper. Unfortunately, hyperparameter tuning in LfD is usually not addressed; instead it is common to use explicit sweeps [Merel et al., 2017, Barde et al., 2020, Ghasemipour et al., 2020, Behbahani et al., 2019] or manual, task-specific tuning [Finn et al., 2017]. According to Ravichandar et al. [2020] hyperparameter tuning is a major obstacle to the deployment of LfD methods.

3 Problem Formulation

In RL, an *agent* learns to make decisions through interaction with an *environment*. We formulate the problem as a Markov Decision Process (MDP), described by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$. $\mathcal{S}$ is the state space and $\mathcal{A}$ the action space. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward, a scalar returned by the environment. The transition probability $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ describes the probability of transitioning to a state, for a given state and action. On each discrete timestep t the agent selects an action A_t in state S_t, the environment transitions to a new state S_{t+1} and emits a scalar reward R_{t+1}.

The agent's objective is to find a policy that maximizes future reward. A policy $\pi : S \times A \to [0, 1]$ defines how the agent chooses actions in each state. The objective is to maximize future discounted reward or the *return*, $G_t \doteq R_{t+1} + \gamma_{t+1} G_{t+1}$ for a discount $\gamma_{t+1} \in [0, 1]$ that depends on the transition (S_t, A_t, S_{t+1}) [White, 2017]. For continuing problems, the discount may simply be a constant less than 1. For episodic problems the discount might be 1 during the episode, and become zero when S_t, A_t leads to termination. Common approaches to learn such a policy are Q-learning and Expected Sarsa, which approximate the action-values—the expected return from a given state and action—and Actor-Critic methods that learn a parameterized policy (see Sutton & Barto [2018]).

We assume that the agent has access to an offline log of data that it can use to initialize hyperparameters before learning online. This log consists of n_{data} tuples of experience $\mathcal{D} = \{(S_t, A_t, R_{t+1}, S_{t+1}, \gamma_{t+1})\}_{i=1}^{n_{\text{data}}}$, generated by interaction in the environment by a previous controller or controllers. For example, an agent that will use Expected Sarsa might want to use this data to decide on a suitable stepsize α, the number of layers l in the neural network (NN) architecture for the action-values and an initialization θ_0 for the NN parameters. There are several options for each hyperparameter combination, $\lambda = (\alpha, l, \theta_0)$, resulting in a set of possible hyperparameters Λ to consider. This set can be discrete or continuous, depending on the underlying ranges. For example, the agent might want to consider any $\alpha \in [0, 1]$ and a θ_0 only from a set of three possible choices. The central question for this Data2Online problem is: how can the agent use this log data to select hyperparameters before learning in deployment?

4 Data2Online using Calibration Models

In this section, we introduce the idea of calibration models and how they can be used for hyperparameter selection. We first discuss how to use the calibration model to select hyperparameters, before providing a way to learn the calibration model. We then discuss certain criteria on the calibration model and agent algorithm that make this strategy more appropriate. We then propose a non-parametric strategy to obtain a calibration model from a batch of offline data.

4.1 Using Calibration Models to Select Hyperparameters Offline

A calibration model is a learned simulator of the environment. The calibration model is learned from an offline batch of data, and used to specify (or calibrate) hyperparameters in the agent. With the calibration model, the agent can test different hyperparameter settings. It evaluates the online performance of different hyperparameter choices in the calibration model and selects the best one. It can then be deployed into the true environment without any remaining hyperparameters to tune.

Algorithm 1 Hyperparameter Selection with Calibration Models

Input: $\mathbf{\Lambda}$: hyperparameter set for learner $Agent$
$\mathcal{D}$: the offline log data
n_{steps}: number of interactions or steps
n_{runs}: number of runs

```
Train calibration model C with D
for λ in Λ do
    Perf[λ] = AgentPerfInEnv(C,
                  Agent(λ), n_steps, n_runs)
```

Return: $\arg\max_{\lambda \in \mathbf{\Lambda}} \text{Perf}[\lambda]$

The basic strategy is simple: we train a calibration model, then do grid search in the calibration model and pick the top hyperparameter, as summarized in Algorithm 1. For each hyperparameter, we obtain a measure of the online performance of the agent across n_{runs} in the calibration model, assuming it gets to learn for n_{steps} of interaction. The pseudocode for AgentPerfInEnv is in Algorithm 3 in the appendix. Note this evaluation scheme guarantees at least 30 episodes are experienced by the agent during training in the model by cutting off episodes that run longer than $n_{\text{steps}}/30$ steps.

Many components in this approach are modular, and can be swapped with other choices. For example, instead of expected return during learning (online performance), it might be more desirable to optimize the hyperparameters to find the best policy after a budget of steps. This would make sense if in deployment cumulative reward during learning was not important. We might also want a more robust agent, and instead of expected return, we may want median return. Finally, the grid search can be replaced with a more efficient hyperparameter selection method; we demonstrate this in the experiments.

We can also make this hyperparameter search more robust to error in the calibration model by obtaining performance across an ensemble of calibration models. This involves using $n_{\text{ensembles}}$ random subsets of the log data, say by dropping at random 10% of samples, and training $n_{\text{ensembles}}$ calibration models. The hyperparameter performance can either be averaged across these models, or a more risk-averse criterion could be used like worst-case performance. Using an average across models is like using a set of source environments to select hyperparameters—rather than a single source—and so could improve transfer to the real environment. In this work, we examine only the simplest form of this Data2Online strategy, and use only one calibration model, and leave these natural additions to future work.

4.2 Criteria for Designing the Calibration Model and Picking Hyperparameter Sets

In this section, we highlight three key criteria for designing the calibration model and selecting agents for which Data2Online should be effective. This includes 1) stability under model iteration, 2) handling actions with low data coverage and 3) selecting agent algorithms that only have initialization hyperparameters, namely those that affect early learning but diminish in importance over time.

Producing reasonable transitions under many steps of model iteration is key for the calibration model. The calibration model is iterated for many steps, because the agent interacts with the calibration model as if it were the true environment—for an entire learning trajectory. It is key, therefore, that the calibration model be *stable* and *self-correcting*. A stable model is one where, starting from any initial state in a region, the model remains in that region. A self-correcting model is one that, even if it produces a few non-real states, it comes back to the space of real states. Otherwise, model iteration can produce increasingly non-sensical states, as has been repeatedly shown in model-based RL [Talvitie, 2017, Jafferjee et al., 2020, Abbas et al., 2020, Chelu et al., 2020].

The model also needs to handle actions with no coverage, or low coverage. For unvisited or unknown states, the model simply does not include such states. The actions, however, can be queried from each state. If an action has not been taken in a state, or not taken in a similar state, the model cannot produce a reasonable transition. A simple option is to produce a transition for such unknown states, back to a default state—like the start state—that will not cause the agent to get stuck.

The third criterion is a condition on the agent, rather than the model. Practically, we can only test each hyperparameter setting for a limited number of steps in the calibration model. So, the calibration

model is only simulating early learning. This suggests that this approach will be most effective if we tune *initialization hyperparameters*: those that provide an initial value for a constant but wash away over time. Examples include an initial learning rate which is then adapted; magnitude of optimism for optimistic initialization; and an initial architecture that is grown and pruned over time.

4.3 Using KNNs to Obtain Stable Calibration Models

We develop a non-parametric k-nearest neighbor (KNN) calibration model that (a) ensures the agent always produces real states and (b) remains in the space of states observed in the data, and so is stable under many iterations. The idea is simple: the entire offline data log constitutes the model, with trajectories obtained by chaining together transitions. From a given transition (s, a, s', r), with the agent selecting action a' from s', the next transition is chosen based on selecting amongst k nearest $(\tilde{s}, \tilde{a}, \tilde{s}', \tilde{r})$ based on similarity between (s', a') and $(\tilde{s}, \tilde{a})$. Though simple, this approach perfectly satisfies the criteria from the previous section,[1] whereas learning stable parametric non-linear models can be quite complex [Manek & Kolter, 2020]. We provide pseudocode and summarize the approach in the appendix, and highlight the key details here.

The KNN relies heavily on the distance metric. The default choice of Euclidean distance in the input space may not be appropriate. For example, if inputs correspond to (x, y), and the environment is a maze, two nearby points in Euclidean distance may actually be far apart in terms of transition dynamics. Images are another example where Euclidean distance is not appropriate.

Instead, we learn and use a distance based on the Laplace representation Wu et al. [2018]. The Laplacian representation is trained by pushing the representations of two random states to be far away from each other, while encouraging the representations of *close* states to be similar. Two states are close if it only takes a few steps for the agent to get to one state from the other. Euclidean distance in this new space is reflective of similarity in terms of transition dynamics. The Laplace representation $\psi(s)$ is trained using a two-layer NN on a batch of data, using the approach given by Wu et al. [2018]. The distance between (s, a) and $(\tilde{s}, \tilde{a})$ is given by $d((s, a); (\tilde{s}, \tilde{a}))=\mathbf{1}(a=\tilde{a})\|\psi(s) - \psi(\tilde{s})\|_2$ for discrete actions and $\|a - \tilde{a}\|\|\psi(s) - \psi(\tilde{s})\|_2$ for continuous actions.

The stochasticity in transitions can be increased by increasing the number of neighbors k. Amongst the k closest neighbors, we pick randomly proportionally to $1 - \frac{d((s_j,a_j);(s_i,a_i))}{\max_{z\in[k]} d((s_z,a_z);(s_i,a_i))}, \forall j \in [k]$. Otherwise, the agent would see the same trajectories, deterministically, when interacting with the calibration model. A trajectory is therefore generated as follows. A start state s_0 is randomly selected from the dataset of start states. The agent selects a_0, and the model finds a transition (s, a, s', r) using the k nearest neighbors to (s_0, a_0) based on (s, a), say transition $(\tilde{s}, \tilde{a}, \tilde{s}', \tilde{r})$. The agent then observes $s_1=\tilde{s}'$ and $r_1=\tilde{r}$. The agent then selects action a_1, and so on. If the agent picks an action where the distance between (s, a) and its nearest neighbor is above some threshold, the calibration model transitions the agent to a new random start state—but without indicating termination—with a default negative reward.

5 Experiments

We conducted a battery of experiments to provide a rounded assessment of when an approach can or cannot be expected to reliably select hyperparameters for online learning. We investigate varying the data collection policy and size of the data logs to mimic a variety of deployment scenarios ranging from a near optimal operator to random data. We explore selecting hyperparameters of different types for several different agents, and investigate a non-stationary setting where the environment changes from data collection to deployment. We begin with the simplest first question: how does our approach compare to simple baselines and with different choices of calibration model type. Experiments were conducted on a cluster and a powerful workstation using 8252 cpu hours and no gpus. Full lists of all the hyperparameters can be found in the appendix.

Experiment 1: Comparing models
In this experiment we investigate the benefits of our approach with different choices of model in two

[1] A natural question is why we do not use a kernel density estimator (KDE) model, which is also non-parametric. The reason is that the KDE model can produce non-existent states. Consider a gridworld with a wall. A KDE model with Euclidean distance in the (x, y) state space might produce an outcome state within the wall. The KDE model still results in significant generalization.

classic control domains. We compare our KNN calibration model with learned Laplace similarity metric to an NN model trained to predict the next state and reward given input state and action observed in the calibration data. In addition we also test an NN calibration model that takes the *Laplacian encoding* (see Section 4.3) of the current state as input and predicts the next state and reward to provide the network with a better transition-aware input representation. We use Fitted-Q to learn a policy from the calibration data and then deploy the learned policy fixed in the deployment environment representing a classical batch RL fixed-deployment baseline. We used two continuous state, discrete action, episodic deployment environments, Acrobot and Puddle World, as described in the appendix and in introductory texts [Sutton & Barto, 2018].

In this first experiment we select the hyperparameters for a linear Softmax-policy Expected Sarsa agent (from here on linear Sarsa) from data generated by a simple policy with good coverage. The agent uses tile coding to map the continuous state variables to binary feature vectors (see for a detailed discussion of tile coding Sutton & Barto [2018]). This on-policy, Sarsa agent learns quickly but is sensitive to several important hyperparameters. We investigate several dimensions of hyperparameters including the step-size and momentum parameters of the Adam optimizer, the temperature parameter of the policy, and the value function weight initialization. We choose these hyperparameters because their impact on performance is somewhat transient and can be overcome by continued learning; this reflects our desire for the agent to continually learn and adapt in deployment. We used a near-optimal policy for each domain to collect data for building the calibration models. The near-optimal data collection policy for Acrobot can solve the task in 100 steps, and the near-optimal data collection policy in Puddle World achieves an average return of -25. In both cases the policy will provide the system with many examples of successful trajectories to the goal states in the 5000 transition data log.

Our evaluation pipeline involves several steps. First we evaluate the *true performance* (steps per episode for Acrobot and return per episode in Puddle World) of each hyperparameter combination in the deployment environment running for 15,000 steps in Acrobot and 30,000 steps in Puddle World, averaging over 30 runs. We used the data collection policy to generate the calibration data log and learn each model. We record the *true performance* of the selected hyperparameters to summarize the performance. This whole process—running the data collection policy to generate a data log, learning the calibration model, and evaluating the hyperparameters—is repeated 30 times (giving 30 datasets with 30 corresponding hyperparameter selections). The statistic of interest is the median and distribution of the *true performance* for the hyper-parameters selected across runs. In the ideal case, if there is one truly best set of hyperparameters, the system will choose those every time and the variance in *true performance* would be zero. For the FQI baseline we simply plot the distribution of performance of each of the 30 extracted policies on the deployment environment. We tested FQI with a tile coded representation and a NN representation; the tile coded version performed the best and we report that result. Figure 2 summarizes the results. The NN calibration model using raw inputs (no Laplacian encoding) was not as effective (see appendix).

In both environments the KNN calibration model performed well, selecting the same hyperparameters as would a sweep directly in the deployment environment. The NN calibration models perform poorly overall. Their performance can be unstable, choosing hyperparameters with good performance in some runs, but often choosing poor hyperparameters. However, in Acrobot the neural network calibration model using Laplacian encoding as inputs outperforms random hyper selection, whereas the FQI baseline performs much worse. This suggests the calibration data log is too limited to extract a good policy and deploy it without additional learning, but the data appears useful for selecting hyperparameters with our approach. We also used our approach to tune both step-size parameters of an linear Actor-critic agent with tile coding. Our approach was able to select top performing hyperparameters for Actor-critic in both Acrobot and Puddle World—though the agent performed worse than linear Sarsa (results can be found in the appendix).

Experiment 2: Varying data collection

The objective of this experiment is to evaluate the robustness of our approach to changing both the amount of offline data available and the quality of the policy used to collect the data. We experimented with 3 different policies corresponding to *near-optimal*, *medium*, and *naive* performance to collect 5000 transitions for training our KNN Laplace calibration model. The near-optimal policy was identical to the one used in the previous experiment. The medium policy was designed to achieve roughly half the visits to goal states after 5000 steps (approximately 90 for Puddle World & 25 for Acrobot) compared to the near optimal policy. The naive policy was designed such that it

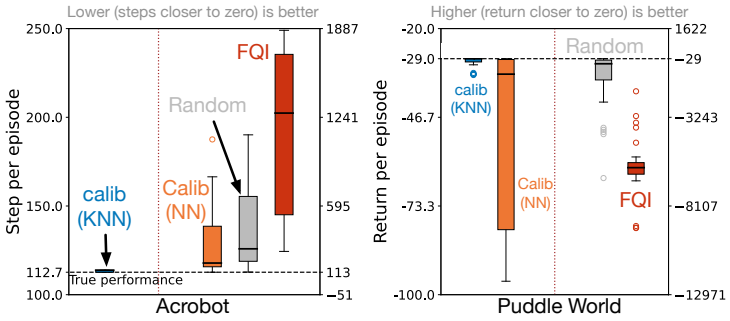

Figure 2: **Hyperparameter transfer with calibration model.** Each subplot shows the performance of two calibration models compared against FQI and a random baseline. The dotted horizontal line indicates the performance of the best hyperparameter setting in the sweep in the deployment environment. Each box shows the distribution summarizing the true performance in deployment of the best hyperparameters selected in each run of the experiment. In Acrobot (lhs) **lower is better**, and in Puddle World (rhs) **higher is better** If the centre of mass is close to the dotted horizontal line, then the system is choosing hyperparameters well. Low variance indicates that the system reliably chooses hyperparameters that perform similarity across runs. We include the performance of randomly selecting hyperparameters on each run as a baseline. In each box the bold line represents the median, the boxes represent the 25th and 75th percentiles, and outliers are circles. The lhs of each box uses the lhs y-axis and the rhs (separated by the dotted vertical line) uses the rhs y-aixs.

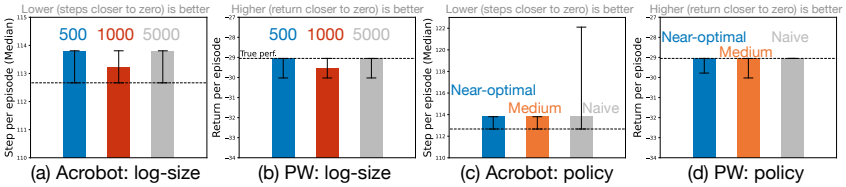

Figure 3: **The role of data logs.** Plots (a) and (b) show the median calibration model performance (with 25% and 75% quartiles) when changing the data log size. Plots (c) and (d) show the median performance when the data collecting policy changes. In Acrobot (lhs) **lower is better**, and in Puddle World (rhs) **higher is better**. Note, these are barplots: the median is shown by the top of the bar, and the quartiles are shown by the whiskers (extending lines). Overall our approach is largely insensitive to the data log size and policy in these two domains.

achieved significantly fewer visits (approximately 35 for Puddle World & 12 for Acrobot). Figure 3 summarizes the results. We also tried different data log sizes of 500, 1000, and 5000 samples using the medium policy, as shown in Figure 3.

The results in Figure 3 show that our approach is largely insensitive to data log size and policy in these classic domains. Even 500 transitions contains enough coverage of the state-space and successful terminations to produce a useful calibration model. This is in stark contrast to the FQI results in Experiment 1 (Fig. 2), where a policy trained offline from the same size data log failed to solve either task. Exploration in both these domains is not challenging; therefore, the success of the calibration model is not surprising. In Experiment 4, we investigate a failure case in Cartpole.

Experiment 3: When the environment changes

Learning online is critical when we expect the environment to change. This can happen due to wear and tear on physical hardware, un-modelled seasonal changes, or the environment may appear non-stationary to the agent because the agent's state representation does not model all aspects of the true state of the MDP. In our problem setting, the deployment environment could change significantly between calibration data collection and the online learning deployment phase. Intuitively we would expect batch approaches that simply deploy a fixed policy learned from data to do poorly in such settings. The following experiment simulates such a scenario with the Acrobot environment.

The experiment involves two variants of the environment. As before, we collected 5000 transitions using the near-optimal policy in Acrobot, apply our approach to find the hyperparameters for the linear Sarsa agent, and train a policy with tile coding FQI. Unlike before, we eval-

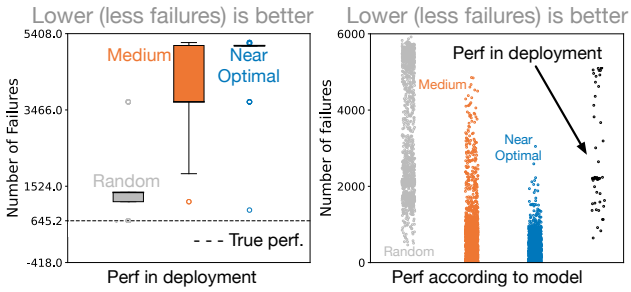

Figure 5: **Success and failure in Cartpole.** This plot shows performance of three different calibration models constructed from random, near-optimal and medium policies. **Left**: performance of the hypers in deployment as picked by different calibration models—lower is better. **Right**: each model's evaluation of all hypers across all 30 runs. Ideally the distribution of performance would match that of the hyperparameter performance in the deployment environment—black dots fair right.

uate the hyperparameters selected and the FQI policy on a second, *changed* Acrobot environment. In the second Acrobot environment we double the length and mass of the first link length. This changes the dynamics but does not prevent learning reasonably good policies. As a sanity check, we also trained a policy using linear Sarsa on the first environment and evaluated its performance on the second environment. Finally we attempted to transfer the best performing policy learned by Sarsa in our calibration model to the second environment. This whole process was repeated 30 times (generating 30 datasets with a corresponding 30 calibration models) to aggregate the results presented in Figure 4.

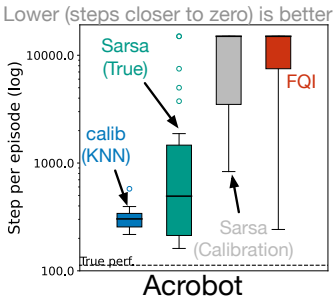

Figure 4: **When Acrobot changes.** Our approach is compared to FQI, a baseline transferring the policy learned in the original environment, and a baseline transferring the policy learned in the calibration model. This plot can be interpreted exactly as Figure 2: the median close to the vertical dotted line represents good performance and small spread indicates that the system reliably chooses hypers that perform similarity across runs.

The results in Figure 4 highlight that transferring static policies can be problematic when the environment changes. Our approach performs best and appears very robust under the abrupt non-stationarity tested here. Clearly the difference between the two environments is significant enough such that transferring a policy directly learned on the first environment performs worse than using our approach and learning a new policy from scratch. The poor transfer performance of the policy learned in the calibration model suggests that our approach is somewhat robust by design to changes between data collection and deployment: our approach uses the same model but transfers hyperparameters, not the policy.

Experiment 4: A failure case

Our approach is not robust to all environments and data collection schemes. In this section we investigate when it can fail. One obvious way our approach can fail is if the agent's performance in the calibration model is always the same: no matter what hyperparameter we try, the system thinks they all perform similarly. To illustrate this phenomenon we use the Cartpole environment. In Cartpole the agent must balance a pole in an unstable equlibrum as long as it can. If the cart reaches the end of the track or the pole reaches a critical angle, failure is inevitable. In Cartpole, near-optimal policies can balance the pole for hundreds of steps rarely experiencing failures or much of the state-space. A data log collected from the near-optimal policy would likely produce a calibration model where failures are impossible and all hyperparameters appear excellent. Figure 5 plots the performance of the best hyperparameters selected according to the calibration model from three different policies, averaged over 30 runs. We used a random policy, a near-optimal policy with random initial pole angles, a medium policy that was half as good as near-optimal.

Figure 5 indeed shows that the dynamics of Cartpole combined with particular data collection policies can render the calibration model ineffective. Even with random starting states the calibration model for near-optimal policy *never simulated dropping the pole*. The random policy produced the best calibration model but it still could not identify the best hyperparameters. Unsurprisingly, the

random policy drops the pole every few steps and thus the log contained many failures but high state coverage—explaining why the performance was better than near-optimal. One could certainly argue that many applications might not exhibit this behavior—especially since it is largely caused by a task with two modes of operation (failing or balancing). Additionally, using a random policy to achieve coverage in unrealistic in applications like water treatment.

Experiment 5: Automatic hyperparameter tuning with CEM

The calibration model is an offline artifact that we can use as we like without impacting the deployment environment. We can use the model in smarter ways to discover and evaluate good hyperparameters for deployment. In this section we discuss and evaluate how to use a simply black-box optimization strategy based on cross-entropy method (CEM) to search the model for high performing hyperparameters. Instead of evaluating a discrete set we can search a continuous space, exploiting smoothness of the performance surface, to improve performance in deployment.

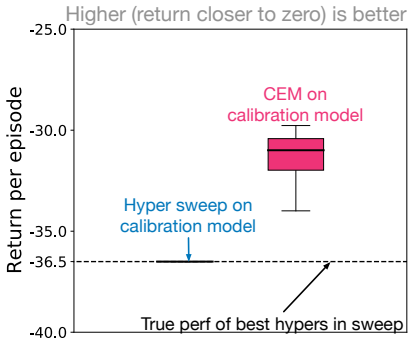

Figure 6: **Combining calibration model with CEM in Puddle World.** The performance of hyperparameter chosen by CEM in calibration model compared with hyperparameter sweeping in the calibration model. The y-axis is same as the one in Figure 2.

CEM is a family of algorithms, where the general principle is to (1) maintain a distribution over plausibly maximal inputs and (2) sample from that distribution and progressively concentrate around the maximal input. We need a blackbox optimizer, because our function—the performance of a hyperparameter h in the calibration model—is not differentiable. Our setting has two nuances compared to the typical setting where CEM is used: our function is expensive to query and we only get a stochastic sample. We provide a modified CEM algorithm, that still reflects the general framework for CEM, but using an incremental update—similar to stochastic gradient descent—to account for the stochasticity in our function query. We describe this algorithm in the appendix.

In this experiment, we use the same settings as above, but now optimize the temperature τ and stepsize α as continuous values in the ranges [0.0001, 10.0] and [0.0, 1.0] respectively for Puddle World. CEM can search this continuous space efficiently, because it uses a truncated multivariate normal that allows for generalization about the utility of hyperparameter values. The performance improvements are stark. Even when tuning only on the calibration model, the agent can significantly outperform the best hyperparameters found by a grid search on the true environment—this is why CEM is outperforming the best hyperparameters line in Figure 6. The result in Acrobot is similar and is included in the appendix.

6 Conclusion

In this work, we introduced the Data2Online problem: selecting hyperparameters from log of data, for deployment in a real environment. The basic idea is to learn a calibration model from the data log, and then allow the agent to interact in the calibration model to identify good hyperparameters. Essentially, the calibration model is treated just like the real environment. We provide a simple approach, using k-nearest neighbors, to obtain a calibration model that is stable under many iterations and only produces real states. We then conduct a battery of tests, under different data regimes.

Naturally, as the first work explicitly tackling this problem, we have only scratched the surface of options. There is much more to understand about when this strategy will be effective, and when it might fail. As we highlight throughout, this problem should be more feasible than offline RL, which requires the entire policy to be identified from a log rather than just suitable hyperparameters for learning. Our own experiments highlight that offline methods that attempted to learn and deploy a fixed policy performed poorly, whereas identifying reasonable hyperparameters was a much easier problem with consistently good performance across many policies and even small datasets. Nonetheless, we did identify one failure case, where the data resulted in a model that made the environment appear too easy and so most hyperparameters looked similar. Much more work can be done, theoretically and empirically, to understand the Data2Online problem.

References

Abbas, Z., Sokota, S., Talvitie, E., and White, M. Selective dyna-style planning under limited model capacity. In *International Conference on Machine Learning*, pp. 1–10, 2020.

Ajay, A., Kumar, A., Agrawal, P., Levine, S., and Nachum, O. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *arXiv preprint arXiv:2010.13611*, 2020.

Barde, P., Roy, J., Jeon, W., Pineau, J., Pal, C., and Nowrouzezahrai, D. Adversarial soft advantage fitting: Imitation learning without policy optimization. *arXiv preprint arXiv:2006.13258*, 2020.

Barto, A. G., Sutton, R. S., and Anderson, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, (5):834–846, 1983.

Behbahani, F., Shiarlis, K., Chen, X., Kurin, V., Kasewa, S., Stirbu, C., Gomes, J., Paul, S., Oliehoek, F. A., Messias, J., et al. Learning from demonstration in the wild. In *2019 International Conference on Robotics and Automation*, pp. 775–781. IEEE, 2019.

Bentley, J. L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.

Chandak, Y., Jordan, S. M., Theocharous, G., White, M., and Thomas, P. S. Towards safe policy improvement for non-stationary mdps. *Advances in Neural Information Processing Systems*, 2020a.

Chandak, Y., Theocharous, G., Shanka, S., White, M., Mahadevan, S., and Thomas, P. S. Optimizing for the future in non-stationary mdps. *International Conference on Machine Learning*, 2020b.

Chelu, V., Precup, D., and van Hasselt, H. Forethought and hindsight in credit assignment. *arXiv preprint arXiv:2010.13685*, 2020.

de Boer, P., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. A tutorial on the cross-entropy method. *Ann. Oper. Res.*, 134(1):19–67, 2005.

Ernst, D., Geurts, P., and Wehenkel, L. Tree-Based Batch Mode Reinforcement Learning. *The Journal of Machine Learning Research*, 6:503–556, 2005. ISSN 1532-4435.

Falkner, S., Klein, A., and Hutter, F. BOHB: robust and efficient hyperparameter optimization at scale. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1436–1445, 2018.

Farahmand, A. M., Ghavamzadeh, M., Szepesvári, C., and Mannor, S. Regularized fitted q-iteration for planning in continuous-space markovian decision problems. In *Proceedings of the 2009 conference on American Control Conference*, pp. 725–730, 2009.

Finn, C., Yu, T., Zhang, T., Abbeel, P., and Levine, S. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning*, pp. 357–368, 2017.

Ghasemipour, S. K. S., Zemel, R., and Gu, S. A divergence minimization perspective on imitation learning methods. In *Conference on Robot Learning*, pp. 1259–1277, 2020.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. DARLA: Improving zero-shot transfer in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1480–1490, 06–11 Aug 2017.

Jacobsen, A., Schlegel, M., Linke, C., Degris, T., White, A., and White, M. Meta-Descent for Online, Continual Prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33: 3943–3950, July 2019.

Jafferjee, T., Imani, E., Talvitie, E., White, M., and Bowling, M. Hallucinating value: A pitfall of dyna-style planning with imperfect environment models. *arXiv preprint arXiv:2006.04363*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52, 2017.

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. Benchmarking Reinforcement Learning Algorithms on Real-World Robots. In *CoRL*, 2018.

Manek, G. and Kolter, J. Z. Learning stable deep dynamics models. *arXiv preprint arXiv:2001.06116*, 2020.

Mann, T. A., Penedones, H., Mannor, S., and Hester, T. Adaptive Lambda Least-Squares Temporal Difference Learning. *ArXiv*, 2016.

Merel, J., Tassa, Y., TB, D., Srinivasan, S., Lemmon, J., Wang, Z., Wayne, G., and Heess, N. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201*, 2017.

Paine, T. L., Paduraru, C., Michi, A., Gulcehre, C., Zolna, K., Novikov, A., Wang, Z., and de Freitas, N. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.

Papini, M., Pirotta, M., and Restelli, M. *Smoothing Policies and Safe Policy Gradients*. May 2019.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. In *IEEE International Conference on Robotics and Automation*, pp. 3803–3810, 2018.

Ravichandar, H., Polydoros, A. S., Chernova, S., and Billard, A. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:297–330, 2020.

Riedmiller, M. Neural Fitted Q Iteration First Experiences with a Data Efficient Neural Reinforcement Learning Method. In *Machine Learning: ECML 2005*, Lecture Notes in Computer Science, pp. 317–328, Berlin, Heidelberg, 2005. Springer.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. Taking the human out of the loop: A review of bayesian optimization. *Proc. IEEE*, 104(1):148–175, 2016.

Sutton, R. S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, pp. 1038–1044, 1996.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Talvitie, E. Self-correcting models for model-based reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Wang, R., Foster, D. P., and Kakade, S. M. What are the statistical limits of offline rl with linear function approximation? *arXiv preprint arXiv:2010.11895*, 2020.

White, M. Unifying task specification in reinforcement learning. In *International Conference on Machine Learning*, 2017.

White, M. and White, A. A Greedy Approach to Adapting the Trace Parameter for Temporal Difference Learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pp. 557–565, May 2016.

Wu, Y., Tucker, G., and Nachum, O. The laplacian in rl: Learning representations with efficient approximations. *arXiv preprint arXiv:1810.04586*, 2018.

Wu, Y., Tucker, G., and Nachum, O. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Xing, J., Nagata, T., Chen, K., Zou, X., Neftci, E., and Krichmar, J. L. Domain adaptation in reinforcement learning via latent unified state representation. *arXiv preprint arXiv:2102.05714*, 2021.

Xu, Z., van Hasselt, H., and Silver, D. Meta-gradient reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 2402–2413, 2018.

Yang, M., Dai, B., Nachum, O., Tucker, G., and Schuurmans, D. Offline policy selection under uncertainty. *arXiv preprint arXiv:2012.06919*, 2020.

Zahavy, T., Xu, Z., Veeriah, V., Hessel, M., Oh, J., van Hasselt, H. P., Silver, D., and Singh, S. A self-tuning actor-critic algorithm. *Advances in Neural Information Processing Systems*, 33, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See 5 Experiment 4 and 4.2 discuss limitations and present negative results.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] This is part of a larger applied project, and the application which is not yet public has a huge positive societal impact.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Submitted with Appendix
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See experiment sections starting at Section 5.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Yes of course. Always at least 30 runs and either standard error bars, boxplots with quartiles and outliers.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] All our experiments were conducted on a local cluster and a powerful workstation using 8252 cpu hrs, no gpus.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A] We use several classic control RL benchmarks which we cite. Our code is in GO so everything was implemented from scratch.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]