Automated Decision Optimization with Reinforcement Learning

Radu Marinescu, Tejaswini Pedapati, Long Vu, Paulito Palmes, Todd Mummert, Peter Kirchner, Dharmashankar Subramanian, Parikshit Ram, Djallel Bouneffouf

IBM Research

Abstract

Sequential decision making under uncertainty has many practical real-world applications, ranging from supply-chain management to autonomous driving. Reinforcement learning has recently emerged as a promising technique for solving these kinds of problems. However, reinforcement learning algorithms are known to be highly sensitive to their internal hyperparameters which require significant expert manual effort for tuning their values. This in turn limits the widespread applicability of reinforcement learning based solutions in practice. In this paper, we introduce a new automated decision optimization system called AutoDO for end-to-end solving of sequential decision making problems. More specifically, the system not only automatically selects the best reinforcement learning algorithm but it also finds the best configuration of its hyperparameters using a search strategy based on limited discrepancy search coupled with Bayesian optimization. Furthermore, our system supports several different flavours of reinforcement learning including online and offline as well as model-free and model-based algorithms. We experiment with classical control benchmarks as well as a more realistic inventory management problem. Our results show that the proposed system is competitive and often outperforms existing state-of-the-art in terms of the quality of the solutions found.

1 Introduction

Practical applications of sequential decision making under uncertainty are numerous, such as daily resource allocation under uncertain demand in inventory networks (Hubbs et al. 2020), autonomous driving under uncertain dynamics and unknown disturbances (Kiran et al. 2021), and intelligent scheduling of heating, ventilation and air conditioning in building control (Wei, Wang, and Zhu 2017), to name a few. Reinforcement Learning (RL) is a promising solution technique that focuses on balancing the exploration-exploitation trade-off by casting these stochastic dynamic problems in the formalism of Markov Decision Processes (Sutton and Barto 2018). It has seen many recent successes when combined with deep learning techniques for approximating the value functions and policy mappings using deep neural networks (Mnih et al. 2013). RL is based on a state-action-reward paradigm of modeling the transitions across time steps in the underlying dynamic system.

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

However, the time and skills required to create production level models with reinforcement learning are major inhibitors to its widespread use and value. This is further exacerbated by the known sensitivity of deep reinforcement learning algorithms to the internal algorithm hyperparameters and neural architecture (Franke et al. 2021). Currently, several months of experimental trial-and-error work are typically required by skilled optimization experts to create and tune such models for effective performance. Overcoming this expensive barrier to the widespread usage of deep reinforcement learning in practice is our research motivation. Analogous to how automated machine learning has empowered data scientists to focus on framing the prediction question instead of hit-or-miss search for the best pipeline, we seek to extend the reach of data scientists to rapidly experiment with state of the art RL algorithms for their respective decision making problems.

Specifically, we present an end-to-end system for automating reinforcement learning with a bilevel nested search space for optimizing the RL pipelines. At the first level, it admits pipelines with different flavors of RL, namely online and offline methods combined with model-free and model-based approaches. In an inner level, it further admits a mixed discrete and continuous search space to tune RL agent hyperparameters as well as neural architecture choices. We present a search strategy based on limited discrepancy search and couple it with Bayesian optimization to traverse the above complex search space. We also conduct experiments with classical control benchmarks and a more realistic inventory management problem to demonstrate the effectiveness of the proposed system.

In summary, our main contributions include: a) An end-to-end system architecture, AutoDO, for automating the creation of deployable RL agents that accepts both gym-compatible (Brockman et al. 2016) environments and (state-, action-, reward-, next-state-) annotated historical data sets, b) a nested search space covering Pipeline search and Hyperparameter search, c) a search algorithm, AutoDO joint optimizer, that couples limited discrepancy search and Bayesian optimization, and d) experimental demonstration of effectiveness of the AutoDO system using classical benchmarks and realistic examples. Section 2 provides the necessary background in RL, the connection to AutoML and AutoRL, and addresses where we go beyond the state of the art with

the proposed system. Section 3 covers our main ideas around the proposed end-to-end system, AutoDO, viz. its architecture, its nested search space for automated pipeline optimization, and its search strategy along with the AutoDO joint optimizer algorithm. Section 4 presents experimental results on AutoDO's competitive end-to-end performance focusing on online and offline model-free RL using classical control benchmarks and realistic examples, and Section 5 concludes the paper.

2 Background

2.1 Reinforcement Learning

The goal in reinforcement learning (RL) is to learn a policy $\pi(a|s)$ that maximizes the expected cumulative discounted reward in a Markov decision process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ represent (discrete/continuous) state and action spaces, $T(s'|s,a)$ and $r(s,a)$ represent the dynamics and reward function, and $\gamma \in (0,1)$ represents the discount factor (Sutton and Barto 2018). In *online* RL, the agent learns the policy by interacting with an environment that encodes the dynamics of the underlying MDP. Alternatively, in *offline* RL (Levine et al. 2020), the policy is learnt from a dataset that represents a collection of state transitions with the corresponding actions and rewards. Furthermore, *model-free* RL agents rely solely on the interaction with the environment (or the offline dataset), while the *model-based* ones first build a model of the MDP from sampled experience (which is obtained by either interacting with the environment for a limited time or by sub-sampling the offline dataset) and subsequently use the model to guide the policy learning process.

2.2 AutoML and AutoRL

Automated machine learning (or AutoML) seeks to automatically select, compose and parameterize machine learning algorithms into a workflow or pipeline of operations that aims at maximizing performance on a given dataset. It has received increasing attention over the past decades but it became more acute in light of the recent explosion in machine learning applications thus spurring AutoML systems (e.g., Kotthoff, Thornton, Hoos, Hutter, and Leyton-Brown (2017); Olson, Bartley, Urbanowicz, and Moore (2016); Feurer, Klein, Eggensperger, Springenberg, Blum, and Hutter (2015); Mohr, Wever, and Hüllermeier (2018) to name a few).

Motivated by the recent success of AutoML, automated reinforcement learning (or AutoRL) has become increasingly popular. Existing AutoRL systems focus on hyperparameter optimization of a given RL algorithm. Specifically, (Zahavy et al. 2020) were the first to propose a meta-gradient optimization scheme where the differentiable parameters of the algorithm are optimized as a part of the loss function. More recently, (Franke et al. 2021) developed a sample efficient search method for finding the best neural architecture and hyperparameters of a given RL agent using a variant of Population Based Training (Jaderberg et al. 2017). In contrast to the existing approaches, our proposed AutoRL system searches for the best agent (and model for model-based RL) and its best hyperparameters and neural architectures.

3 Automated Decision Optimization

In this section, we present the main contribution of the paper which is an end-to-end automated system, AutoDO, for computing optimized decision policies using reinforcement learning. We start by describing the architecture of the system together with its main components. Subsequently, we present a novel approach for hyperparameter optimization based on limited discrepancy search (LDS) (Harvey and Ginsberg 1995) which we use to tune the reinforcement learning agents. We note that our system supports model-free as well as model-based RL agents in online and offline settings.

3.1 System Architecture

The architecture of our proposed AutoDO system is shown in Figure 1. The input to the system consists of a collection of RL agents and either a gym-compatible environment or a dataset containing tuples (s, a, r, s'), where s and s' represent the current and respectively the next state of the underlying MDP, a is the current action, and r is the reward. The output of the system consists of the set of top k pipelines produced by the joint pipeline and hyperparameter optimizer. We note that, depending on the types of inputs, the system facilitates online and offline reinforcement learning using model-free and model-based RL algorithms.

The system can optimize four types of reinforcement learning pipelines (or RL pipelines for short) which are depicted in Figure 2. For model-free RL (e.g., (Mnih et al. 2013)), the pipeline is a linear structure with two modules: the environment (resp. the dataset) module and the agent module (see Figures 2(a) and 2(c), respectively). Alternatively, for model-based RL (e.g., (Janner et al. 2019)), the pipeline is also a linear structure with an additional module which corresponds to the dynamics model (see Figures 2(b) and 2(d), respectively).

Clearly, each of the RL pipelines above defines a specific search space. Furthermore, each agent in the pipeline has a specific set of hyperparameters which in turn define another search space. The AutoDO system jointly traverses these two search spaces in order to produce an optimized RL pipeline which corresponds to an optimized policy.

3.2 Search Spaces

We now describe in more details the RL pipeline and hyperparameter search spaces.

Pipeline Search Space This is a discrete search space defined by a set of discrete variables each with a finite domain of values. Therefore, a pipeline configuration corresponds to value assignment to its variables.

More specifically, referring again to Figure 2(a), the online model-free RL pipeline has 2 variables called Environment and Agent, respectively. The values for Environment is a list of gym environment instances each one initialized with a different random seed, while the values for Agent is a list of RL algorithms that are compatible with the respective environment. For example, given an environment env, 3 agents

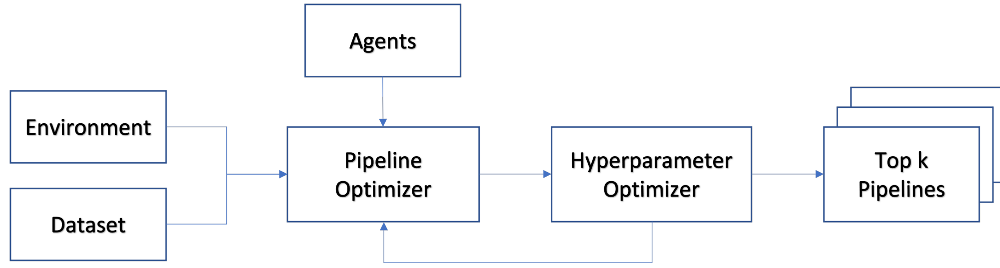

Figure 1: System architecture.

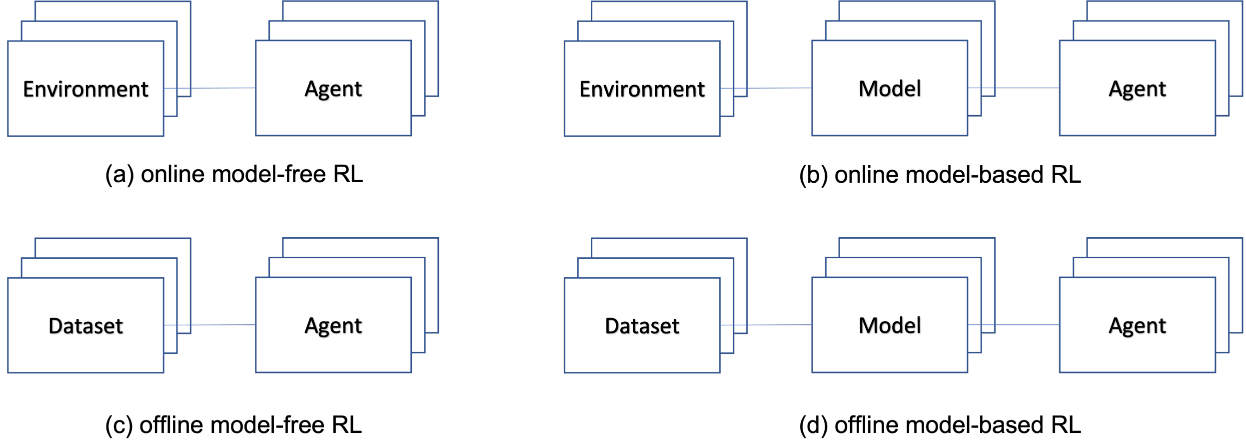

Figure 2: Types of pipelines supported by the system.

$\{a_1, a_2, a_3\}$, and 3 random seeds $\{x, y, z\}$ then the discrete domains of the pipeline variables are $Environment \in \{env_x, env_y, env_z\}$ (where env_x is an instance of env initialized with seed x) and $Agent \in \{a_1, a_2, a_3\}$ which in turn define a search space with 9 RL pipelines each one corresponding to a particular instantiation of the variables.

In case of online model-based RL, the pipeline has an additional variable called Model which corresponds to the learnt dynamics model of the environment (see Figure 2(b)). We assume that the data required for learning this model is available (e.g., it can be easily obtained by sampling the environment using an arbitrary policy). In principle, the dynamics model contains a reward function model (e.g., regression model) and a state transition model (e.g., density estimator). In principle, we can use many different machine learning algorithms to learn these models (e.g., a random forest regressor for rewards and a Gaussian Process based density estimator for state transitions). Therefore, the values of the Model variable correspond to the different combinations of ML algorithms used to learn the dynamics models.

For offline model-free and model-based RL, the Environment variable in the pipeline structure is replaced by the Dataset variable whose values correspond to different train-test splits of the input dataset. As in the online case, the train-test splits can be obtained using different random seeds.

Hyperparameters Search Space This is a mixed discrete and continuous search space defined by a set of variables that correspond to the hyperparameters of a given RL agent (e.g., learning rate, discount factor γ, policy/value network architectures, activation function, etc.). We note that each RL agent defines its own hyperparameter search space and a hyperparameter configuration is an instantiation of the corresponding variables. Therefore, in the remainder of the paper, an *instantiated pipeline* is a pipeline configuration together with a configuration of its RL agent's hyperparameters.

Algorithm 1: LDS: Limited Discrepancy Search

```
1:  procedure LDS
2:    for all k = 0 ... n do
3:      if PROBE(root, k) then
4:        return true
5:  function PROBE(node, k)
6:    if isLeaf(node) then
7:      return isGoal(node)
8:    if k = 0 then
9:      return PROBE(left(node), 0)
10:   else
11:     return PROBE(right(node), k-1) or
12:            PROBE(left(node),k)
```

3.3 Limited Discrepancy Search

We present now a search strategy based on limited discrepancy search that can be employed to traverse efficiently the pipeline and hyperparameter search spaces.

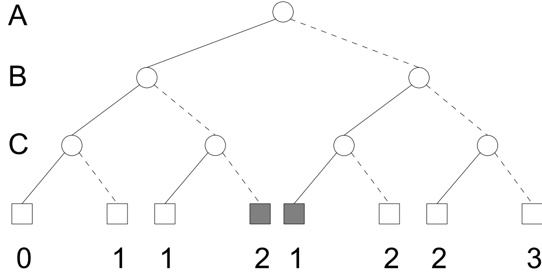

Figure 3: Search space explored by LDS (height is 3). The number of discrepancies is indicated below the leaf nodes.

For our purpose, we consider a search space that is a *complete binary tree* with bounded height h. Leaf nodes correspond to *goals* or *failures* and the task of interest is to find a goal leaf node. Each internal node represents a decision that has to be made to reach a goal. Furthermore, the left child of each internal node represents following the recommendation of a value-ordering heuristic and the right child represents going against that recommendation. Disregarding the heuristic recommendation is called a *discrepancy* (Harvey and Ginsberg 1995). The number of discrepancies of a leaf node is the number of right turns in the path from the root to that leaf node.

Limited Discrepancy Search (LDS) (Harvey and Ginsberg 1995; Korf 1996) is a depth-first search algorithm that searches for a goal node while increasing the number of discrepancies in an iterative manner. The pseudo-code is given in Algorithm 1. The k-th iteration of the main loop will visit all the leaves having k or fewer discrepancies. Function PROBE is a standard recursive implementation of depth-first search such that: (i) it keeps track (parameter k) of the number of discrepancies still available, (ii) if a discrepancy is consumed, k is decreased before the recursive call and (iii) if no further discrepancies are available, the algorithm does not disregard the heuristic. Since the last iteration visits all the leaves, the algorithm is complete. In practice, LDS is used in a anytime manner until a solution (or goal node) is found or a time limit is reached.

Example 1 *Figure 3 shows a search tree with height 3. The gray leaves correspond to goals (solutions). LDS stops during iteration $k = 1$ where it finds the solution with 1 discrepancy.*

In our case, we use the notion of discrepancy without any value-ordering heuristic. Instead, the discrepancy represents deviations from an initial solution (i.e., assignment to variables), while the leaves are all goal nodes and correspond to pipeline and hyperparameter configurations, respectively. Furthermore, we consider an optimization problem where the goal nodes are associated with a score (i.e., the value of a black-box objective function) and the task is to find the goal or solution node with the highest score.

3.4 The AutoDO Joint Optimizer

Algorithm 2 describes our LDS based joint pipeline and hyperparameter optimizer. Since LDS assumes a discrete

Algorithm 2: Joint Pipeline and Hyperparameter Optimizer

Require: envs, agents, hyperparams, discrepancy $disc_p$, $disc_h$
1: **procedure** OPTIMIZER(env, agents, hyperparams, $disc_p$, $disc_h$)
2: p_{init} = FINDINITIALPIPELINE(envs, agents)
3: **repeat**
4: s = HPO(p_{init}, hyperparams, $disc_h$)
5: **for all** (θ = 1; $\theta \leq disc_p$; $\theta = \theta + 1$) **do**
6: p_{new} = SEARCH1(p_{init}, s, θ, hyperparams, 1)
7: **if** ($p_{new} \neq \phi$) **then**
8: $p_{init} = p_{new}$; **break**
9: **until** no better pipeline is found
10: **return** best pipeline obtained
11: **function** SEARCH1(Pipeline p, score l, discrepancy θ, hyperparams, stage i)
12: **if** ($\theta = 0 \lor i > length(p)$) **then**
13: s' = HPO(p, hyperparams, $disc_h$)
14: **if** ($s' > s$) **then return** p
15: **else return** ϕ
16: **else**
17: **for all** agent $a \in agents$ **do**
18: **if** ($p[i] = a$) **then**
19: r = SEARCH1(p, l, θ, hyperparams, $i + 1$)
20: **else**
21: $p_{new} = p$; $p_{new}[i] = a$
22: r = SEARCH1(p_{new}, l, $\theta - 1$, hyperparams, $i + 1$)
23: **if** ($r \neq \phi$) **then return** r
24: **return** ϕ
25: **procedure** HPO(Pipeline p, hyperparams, $disc_h$)
26: c_{init} = FINDINITIALCONFIG(p, hyperparams)
27: **repeat**
28: s = TRAINANDEVAL(p.env, p.agent, c_{init})
29: **for all** (θ = 1; $\theta \leq disc_h$; $\theta = \theta + 1$) **do**
30: c_{new} = SEARCH2(p, c_{init}, l, θ, env, 1)
31: **if** ($c_{new} \neq \phi$) **then**
32: $c_{init} = c_{new}$; **break**
33: **until** no better configuration is found
34: **return** best configuration obtained
35: **function** SEARCH2(Pipeline p, Configuration c, score l, discrepancy θ, stage i)
36: **if** ($\theta = 0 \lor i > length(c)$) **then**
37: l' = TRAINANDEVAL(p.env, p.agent, c)
38: **if** ($l' > l$) **then return** c
39: **else return** ϕ
40: **else**
41: **for all** hyperparam $a \in hyperparams$ **do**
42: **if** ($c[i] = a$) **then**
43: r = SEARCH2(p, c, l, θ, $i + 1$)
44: **else**
45: $c_{new} = c$; $c_{new}[i] = a$
46: r = SEARCH2(p, c_{new}, l, $\theta - 1$, $i + 1$)
47: **if** ($r \neq \phi$) **then return** r
48: **return** ϕ

search space, we discretize a continuous hyperparameter by uniformly splitting its range into a fixed number of bins. The following notation is used. Function FINDINITIALPIPELINE generates a randomly initialized RL pipeline $p_{init} = (a_1, \ldots, a_m)$, however, the algorithm can start with any pipeline obtained with other methods. Position i in pipeline p_{init} is also also referred to as *stage* i, where $1 \leq i \leq m$. Similarly, function FINDINITIALCONFIG generates an initial configuration c_{init} of the hyperparameters for a given RL pipeline. Furthermore, function TRAINANDEVAL trains the pipeline's agent, p.agent, with hyperparameters given by c_{init} on the pipeline's environment, p.env. The trained policy is then evaluated on the same environment to determine the score of the respective instantiated pipeline. The optimizer assumes that a better pipeline (resp. configuration of hyperparameters) tends to be instantiated in a similar fashion to p_{init} (resp. c_{init}) and, therefore, it examines a limited search space where similar pipelines (resp. hyperparameter configurations) are located. The symbol ϕ is used to indicate that the algorithm could not find a pipeline (resp. hyperparameter configuration) better than p_{init} (resp. c_{init}) with current discrepancy value θ.

The algorithm starts with a discrepancy value θ of 1 and conducts an iterative search that allows to change the modules of at most θ stages in p_{init} while incrementing θ until a better pipeline p_{new} is found or θ exceeds $disc_p$ (see lines 2-10). If p_{new} is found, the algorithm uses it as a new initial pipeline and attempts to improve it further. Function SEARCH1 (lines 12-27) performs the actual exploration of the pipeline search space limited by discrepancy θ. Specifically, when it selects a new value a that is different from the one corresponding to stage i in p_{init} it decrements θ to reduce the number of changes allowed for the remaining stages (lines 21-22). Otherwise, the value for stage i remains unchanged and, therefore, the θ value is preserved (line 18-19). When SEARCH1 either has checked all stages/modules in p or consumed the discrepancy budget, it determines p's performance using the HPO function (lines 12-15).

Specifically, the HPO method performs a nested LDS traversal of the hyperparameter search space corresponding to the current pipeline p (lines 25-48). It starts with an initial hyperparameter configuration c_{init} and gradually explores the search space around it using increasing discrepancy values (up to $disc_h$). Function SEARCH2 conducts the actual exploration (in the same way as SEARCH1 does) and when the current best hyperparameter configuration cannot be improved further it returns its score as the score of pipeline p. Namely, when SEARCH2 selects a new hyperparameter value a that is different from the one corresponding to the hyperparameter at position i in c_{init} it decrements θ to reduce the number of changes allowed for the remaining hyperparameters (lines 45-46). Otherwise, the value for the hyperparameter at position i in the configuration remains unchanged and, therefore, the θ value is preserved (line 42-43). When SEARCH4 either finished checking all hyperparameters in configuration c or consumed the discrepancy budget, it determines the score of the configuration c using the TRAINANDEVAL function (lines 12-15).

For efficiency, the joint optimizer also caches the score for all trained pipelines and all hyperparameter configurations in order to avoid retraining them. In fact, caching alleviates the overhead of revisiting the same pipeline and hyperparameter configurations possibly with different discrepancy values.

Remark Finally, we note that our joint optimizer's architecture is quite flexible and can accommodate different search algorithms for pipeline optimization and for hyperparameter optimization, respectively. For example, we can perform random search (Bergstra and Bengio 2012) over the pipeline search space and optimize the hyperparameters using Bayesian Optimization (BO) (Bergstra et al. 2011). In our experiments, we also ran a version of the joint optimizer that uses LDS for pipeline optimization and Hyperopt (BO) (Bergstra 2013) for hyperparameter optimization. Furthermore, the joint optimizer can be easily parallelized and run in a distributed environment such as a ray cluster.

4 Experiments

In this section, we evaluate empirically the performance of the end-to-end AutoDO system using online model-free RL and classical control environments. In addition, we also consider offline model-free RL with datasets derived from an environment using perturbed expert policies.

4.1 Online AutoDO

For the online case, we consider the following environments: CartPole-v1 and Acrobot-v1 from OpenAI Gym[1] and InvManagement-v1 from OR-Gym[2], respectively. The first two environments correspond to discrete control problems (i.e., discrete actions) while the third environment is a continuous control problem (i.e., continuous actions). In case of discrete control, we use the following agents: PPO, A2C and DQN, while for continuous control we use PPO, A2C and SAC, respectively[3]. For each environment, we ran 10 different seeds $s \in \{1, 2, 3, \ldots, 10\}$. Therefore, the pipeline search space was relatively small having 30 pipelines while the hyperparameter search space contained between 7 and 10 hyperparameters with 5 discrete values each. Consequently, we ran the LDS based joint pipeline and hyperparameter optimizer with a maximum discrepancy value of 1. Each instantiated pipeline was trained for 50,000 timesteps and subsequently evaluated for 100 episodes to determine the score used during the search.

Table 1 summarizes the results obtained with the online AutoDO system. We show the top 3 pipelines (environment instance and agent) found for each of the three environments and compare them with the existing state of the art (SOTA) in terms of mean (episode) reward. The latter was obtained by evaluating the trained agent for 1,000 episodes on the corresponding environment. We obtained the SOTA performance from (Hubbs et al. 2020) for InvManagement-v1 and from (Custode and Iacca 2020) for CartPole-v1, respectively. For the Acrobot-v1 environment we consulted the

[1] `https://gym.openai.com/envs`

[2] `https://github.com/hubbs5/or-gym`

[3] All agents available from the `stable-baselines` library at https://github.com/DLR-RM/stable-baselines3

Environment	AutoDO Agent	AutoDO Return	SOTA
CartPole-v1-s1	DQN	**500**	
CartPole-v1-s2	PPO	**500**	**500**
CartPole-v1-s3	PPO	**500**	
Acrobot-v1-s6	PPO	-82	
Acrobot-v1-s9	PPO	-82	**-60**
Acrobot-v1-s7	PPO	-88	
InvManagement-v1-s7	A2C	**417**	
InvManagement-v1-s8	A2C	404	409.8
InvManagement-v1-s5	A2C	354	

Table 1: Results with online AutoDO on the CartPole-v1, Acrobot-v1 and InvManagement-v1 environments. We show the top 3 pipelines for each environment. Using LDS with discrepancy 1 for both pipeline and hyperparameter optimization.

Environment	AutoDO Agent	AutoDO Return	SOTA
CartPole-v1-s1	PPO	**500**	
CartPole-v1-s2	A2C	**500**	**500**
CartPole-v1-s3	A2C	**500**	
Acrobot-v1-s9	PPO	-79	
Acrobot-v1-s1	PPO	-80	**-60**
Acrobot-v1-s7	PPO	-88	
InvManagement-v1-s1	PPO	**494**	
InvManagement-v1-s4	PPO	**484**	409.8
InvManagement-v1-s7	PPO	**465**	

Table 2: Results with online AutoDO on the CartPole-v1, Acrobot-v1 and InvManagement-v1 environments. We show the top 3 pipelines for each environment. Using LDS with discrepancy 1 for pipeline search and Hyperopt (BO) for hyperparameter optimization.

current SOTA leaderboard[4]. Notice that for each pipeline, the environment instance is identified by its corresponding seed (e.g., Acrobot-v1-s6 is the Acrobot-v1 environment seeded with $s = 6$). We can see that the AutoDO agents are quite competitive and sometimes exceed SOTA on two environments (CartPole-v1 and InvManagement-v1). This is important because our agents require virtually no input from the user in terms of tuning their hyperparameter values. The relatively poor performance on the Acrobot-v1 environment compared with SOTA may be due to the number of training timesteps which could be quite small in this case.

Table 2 shows the results obtained with a version of the joint optimizer that uses LDS with discrepancy 1 for pipeline search and Bayesian Optimization based Hyperopt for hyperparameter optimization. As before, we report the top 3 pipelines found for each of the three environments. We can see that the AutoDO solutions are again competitive and sometimes significantly outperform SOTA on the CartPole-v1 and InvManagement-v1 environments, respectively. When comparing with the LDS based hyperparameter optimizer we see that the Hyperopt based one provides higher quality solutions compared with the former. The relatively poorer performance of LDS may be due to the uniform scheme used to discretize the continuous hyperparameters which does not yield good hyperparameters in this case. Therefore, improving the performance of LDS for mixed discrete and continuous hyperparameter search spaces is left as a topic of future work.

4.2 Offline AutoDO

For the offline case, we consider offline datasets derived from the InvManagement-v1 environment as follows. Following (Hubbs et al. 2020), we manually train an expert PPO policy on the InvManagement-v1 environment so that it converged to a mean reward of 490 which is competitive to the one reported in (Hubbs et al. 2020). Our policy network has 2 layers with 256 hidden units each. Subsequently, we use the trained policy to generate datasets with 10,000 episodes from the environment by injecting random actions with probability $\epsilon \in \{0.01, 0.05, 0.1, 0.2, 0.3\}$. We use 9 model-free offline agents[5]: BC, CQL, BEAR, AWAC, CRR, TD3, TD3PlusBC, PLAS and AWAC, respectively. In this case, we ran only one 80-20 training-test split (with seed 5489) per dataset and therefore traversed a small search space with only 9 pipelines. The hyperparameter search space was larger having 7 to 13 hyperparameters with both discrete and continuous values. We ran an LDS based pipeline optimizer with discrepancy of 1, and a Hyperopt (Bergstra et al. 2011) based hyperparameter optimizer with 100 iterations. Each instantiated pipeline was trained for 1,000,000 timesteps and we determined its score by off-policy offline policy evaluation (OPE) (Le, Voloshin, and Yue 2019) using 10,000 timesteps.

Table 3 summarizes the results obtained with the offline AutoDO system. We show the top 3 pipelines (or policies) found for each value of the ϵ parameter and report the mean reward (return) found when we evaluated the respective policy on the actual InvManagement-v1 environment for 1,000 episodes. We observe that as the amount of randomness in the perturbed expert policy used for generating the data decreases, the policy improves considerably and gets closer to the expert policy. However, even though we do not expect the offline policies to match or exceed their online counterparts, we demonstrate the value of our proposed AutoDO system that is not only able to produce competitive policies but require no manual tuning of parameters.

5 Conclusions

The paper presents AutoDO, a new system for end-to-end solving of sequential decision making problems under uncertainty using reinforcement learning. In contrast to existing state-of-the-art solutions, our approach not only selects the best reinforcement learning algorithm but also finds the best configuration of its hyperparameters. The system supports various flavours of reinforcement learning including

[4]https://github.com/openai/gym/wiki/Leaderboard

[5]Available from the `d3rlpy` library at https://github.com/takuseno/d3rlpy.

AutoDO Agent	ϵ	AutoDO Return
CRR	0.3	248
BEAR	0.3	224
CQL	0.3	223
PLAS	0.2	305
CRR	0.2	294
TD3	0.2	279
BC	0.1	282
CQL	0.1	276
PLAS	0.1	272
TD3	0.05	618
CQL	0.05	342
BEAR	0.05	326
TD3	0.01	396
BC	0.01	331
PLAS	0.01	322

Table 3: Results with offline AutoDO on datasets created from InvManagement-v1 environment. We show the top 3 agents for each value of $\epsilon \in \{0.01, 0.05, 0.1, 0.2, 0.3\}$.

online and offline as well as model-free and model-based algorithms. The complex search space of possible RL algorithms and hyperparameters is traversed efficiently by a joint optimizer that combines advanced search strategies such as limited discrepancy search and Bayesian optimization in a very effective manner. We experiment with classical control benchmarks as well as a more realistic inventory management problem. Our results show that the proposed AutoDO system is competitive and often outperforms existing state-of-the-art in terms of the quality of the solutions found.

References

Bergstra, J.; and Bengio, Y. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(1): 281–305.

Bergstra, J. S.; Bardenet, R.; Bengio, Y.; and Kegl, B. 2011. Algorithms for Hyper-Parameter Optimization. In *NeuIPS*, 2546–2554.

Bergstra, Y. D. C. D. D., J. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, 115–123.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. .

Custode, L. L.; and Iacca, G. 2020. Evolutionary learning of interpretable decision trees. *CoRR*, abs/2012.07723.

Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J. T.; Blum, M.; and Hutter, F. 2015. Auto-sklearn: Efficient and robust automated machine learning. In *NeurIPS*, 2962–2970.

Franke, J. K. H.; Köhler, G.; Biedenkapp, A.; and Hutter, F. 2021. Sample-Efficient Automated Deep Reinforcement Learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Harvey, W.; and Ginsberg, M. 1995. Limited Discrepancy Search. In *IJCAI*, 607–613.

Hubbs, C. D.; Perez, H. D.; Sarwar, O.; Sahinidis, N. V.; Grossmann, I. E.; and Wassick, J. M. 2020. OR-Gym: A Reinforcement Learning Library for Operations Research Problem. *CoRR*, abs/2008.06319.

Jaderberg, M.; Dalibard, V.; Osindero, S.; Czarnecki, W. M.; Donahue, J.; Razavi, A.; Vinyals, O.; Green, T.; Dunning, I.; Simonyan, K.; Fernando, C.; and Kavukcuoglu, K. 2017. Population Based Training of Neural Networks. *CoRR*, abs/1711.09846.

Janner, M.; Fu, J.; Zhang, M.; and Levine, S. 2019. When to Trust Your Model: Model-Based Policy Optimization. *CoRR*, abs/1906.08253.

Kiran, B. R.; Sobh, I.; Talpaert, V.; Mannion, P.; Al Sallab, A. A.; Yogamani, S.; and Pérez, P. 2021. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*.

Korf, R. E. 1996. Improved Limited Discrepancy Search. In *AAAI*, 286–291.

Kotthoff, L.; Thornton, C.; Hoos, H.; Hutter, F.; and Leyton-Brown, K. 2017. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 18(1): 826–830.

Le, H. M.; Voloshin, C.; and Yue, Y. 2019. Batch Policy Learning under Constraints. *CoRR*, abs/1903.08738.

Levine, S.; Kumar, A.; Tucker, G.; and Fu, J. 2020. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *CoRR*, abs/2005.01643.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR*, abs/1312.5602.

Mohr, F.; Wever, M.; and Hüllermeier, E. 2018. ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8-10): 1495–1515.

Olson, R. S.; Bartley, N.; Urbanowicz, R. J.; and Moore, J. H. 2016. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In Friedrich, T.; Neumann, F.; and Sutton, A. M., eds., *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, 485–492. ACM.

Sutton, R.; and Barto, A. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

Wei, T.; Wang, Y.; and Zhu, Q. 2017. Deep reinforcement learning for building HVAC control. In *Proceedings of the 54th annual design automation conference 2017*, 1–6.

Zahavy, T.; Xu, Z.; Veeriah, V.; Hessel, M.; Oh, J.; van Hasselt, H.; Silver, D.; and Singh, S. 2020. A Self-Tuning Actor-Critic Algorithm. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.