Journal of Artificial Intelligence Research 74 (2022) 517-568 Submitted 01/2022; published 06/2022

Automated Reinforcement Learning (AutoRL): A Survey and Open Problems

Jack Parker-Holder* JACKPH@ROBOTS.OX.AC.UK
University of Oxford

Raghu Rajan* RAJANR@CS.UNI-FREIBURG.DE
University of Freiburg

Xingyou Song* XINGYOUSONG@GOOGLE.COM
Google Research, Brain Team

André Biedenkapp BIEDENKA@CS.UNI-FREIBURG.DE
University of Freiburg

Yingjie Miao YINGJIEMIAO@GOOGLE.COM
Google Research, Brain Team

Theresa Eimer EIMER@TNT.UNI-HANNOVER.DE
Leibniz University Hannover

Baohe Zhang ZHANGB@CS.UNI-FREIBURG.DE
University of Freiburg

Vu Nguyen VUTNGN@AMAZON.COM
Amazon Australia

Roberto Calandra RCALANDRA@FB.COM
Meta AI

Aleksandra Faust† SANDRAFAUST@GOOGLE.COM
Google Research, Brain Team

Frank Hutter† FH@CS.UNI-FREIBURG.DE
University of Freiburg & Bosch Center for Artificial Intelligence

Marius Lindauer† LINDAUER@TNT.UNI-HANNOVER.DE
Leibniz University Hannover

Abstract

The combination of Reinforcement Learning (RL) with deep learning has led to a series of impressive feats, with many believing (deep) RL provides a path towards generally capable agents. However, the success of RL agents is often highly sensitive to design choices in the training process, which may require tedious and error-prone manual tuning. This makes it challenging to use RL for new problems and also limits its full potential. In many other areas of machine learning, AutoML has shown that it is possible to automate such design choices, and AutoML has also yielded promising initial results when applied to RL. However, *Automated Reinforcement Learning* (AutoRL) involves not only standard applications of AutoML but also includes additional challenges unique to RL, that naturally produce a different set of methods. As such, AutoRL has been emerging as an important area of research in RL, providing promise in a variety of applications from RNA design to playing

 * Co-first authors † Co-last authors.

games, such as Go. Given the diversity of methods and environments considered in RL, much of the research has been conducted in distinct subfields, ranging from meta-learning to evolution. In this survey, we seek to unify the field of AutoRL, provide a common taxonomy, discuss each area in detail and pose open problems of interest to researchers going forward.

1. Introduction

In the past decade, we have seen a series of breakthroughs using Reinforcement Learning (RL, (Sutton & Barto, 2018)) to train agents in a variety of domains, such as games (Mnih et al., 2015; Berner et al., 2019; Silver et al., 2016; Vinyals et al., 2019) and robotics (OpenAI et al., 2018), with successes emerging in real world applications (Bellemare et al., 2020; Nguyen et al., 2021; Degrave et al., 2022). As such, there has been a surge of interest in the research community.

However, while RL has achieved some impressive feats, many of the headline results rely on heavily tuned implementations, which fail to generalize beyond the intended domain. Indeed, RL algorithms have been shown to be incredibly sensitive to hyperparameters and architectures of deep neural networks (Henderson et al., 2018; Andrychowicz et al., 2021; Engstrom et al., 2020), while there are a growing number of additional design choices, such as the agent's objective (Hessel et al., 2019) and update rule (Oh et al., 2020). It is tedious, expensive and potentially even error-prone for humans to manually optimize so many design choices at once. There has been significant success in other areas of machine learning (ML) with *Automated Machine Learning* (AutoML, Hutter et al. (2019)). However, these methods have not yet had a significant impact in RL, in part due to RL applications being typically challenging, due to the diversity of environments and algorithms, and the nonstationarity of the RL problem.

The goal of this survey is to present the field of *Automated Reinforcement Learning* (AutoRL), as a suite of methods automating a varying degree of the RL pipeline. AutoRL serves to tackle a variety of challenges: on the one hand, the fragility of RL algorithms hinders applications in novel fields, especially those where practitioners lack vast resources to search for optimal configurations. In many settings it may be prohibitively expensive to manually find even a moderately strong set of hyperparameters for a completely unseen problem. AutoRL has already been shown to aid in such a situation for important problems, such as designing RNA (Runge et al., 2019). On the other hand, for those with the benefit of more compute, it is clear that increasing the flexibility of the algorithm can boost performance (Xu et al., 2020; Zahavy et al., 2020; Jaderberg et al., 2017). This has already been exhibited with the famed AlphaGo agent, which was significantly improved through the use of Bayesian Optimization (BO) (Chen et al., 2018).

Methods that can be considered AutoRL algorithms were shown to be effective as early as the 1980s (Barto & Sutton, 1981). However, in recent times the prevalence of AutoML has led to the nascent application of more advanced techniques (Runge et al., 2019; Chiang et al., 2019). Meanwhile the evolutionary community has been evolving neural networks alongside their weights for decades (Stanley & Miikkulainen, 2002), with methods inspiring those proving effective for modern RL (Jaderberg et al., 2017). In addition, the recent popularity of meta-learning has led to a series of works that seek to automate the RL process (Houthooft et al., 2018; Xu et al., 2018; Kirsch et al., 2020).

In this paper, we seek to provide a taxonomy of these methods. In doing so, we hope to open up a swathe of future work through the cross-pollination of ideas, while also introducing RL researchers to a suite of techniques to improve the performance of their algorithms. We believe AutoRL has a significant part to play in aiding the potential impact of reinforcement learning, both in open-ended research and practical real-world applications, and this survey could form a starting point for those looking to harness its potential.

Furthermore, we hope to attract researchers interested in AutoML more broadly to the AutoRL community, since AutoRL poses unique challenges. In particular, RL suffers from non-stationarity, as the data the agent is training on is a function of the current policy. In addition, AutoRL also encompasses environment and algorithm design specific to RL problems. We believe these challenges will require significant future work and thus outline open problems throughout this paper.

We structure the paper as follows. In Section 2, we describe the background and notation needed to formalize the AutoRL problem and then formalize the problem and discuss various methodologies to evaluate it. We then briefly summarize various types of RL algorithms followed by a description of non-stationarity unique to the AutoRL problem. In Section 3, we discuss the various components of the AutoRL problem that need automating including the environments, algorithms, their hyperparameters and architectures. In Section 4, we provide a taxonomy and survey current AutoRL methods in subsections following this taxonomy. In Section 5, we discuss various publicly available benchmarks and their application domains. Finally, in Section 6, we discuss future directions for AutoRL.

2. Preliminaries and Notation

We begin by defining a Markov Decision Process (MDP) as a 7-tuple $(\mathcal{S}, \mathcal{A}, P, R, \rho_o, \mathcal{T}, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ describes the transition dynamics, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ describes the reward dynamics, $\rho_o : \mathcal{S} \to \mathbb{R}^+$ is the initial state distribution, $\mathcal{T}$ is the set of terminal states and $\gamma \leq 1$ is the discount factor[1].

We further define a POMDP with two additional components: $\mathcal{O}$ represents the set of observations and $\Omega : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \to \mathbb{R}^+$ describes the probability density function of an observation given a state and action. This makes it an 9-tuple $(\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Omega, \rho_o, \mathcal{T}, \gamma)$.

Note that in many cases this is not sufficient to fully define the task considered (for example, if the dynamics are sampled from a parameterized distribution, such as having a cartpole environment where the length of the pole is drawn from a distribution). Where necessary we will discuss this alongside the methods which require it. Many of the algorithms we discuss are designed to optimize agents to maximize reward in a MDP/POMDP with a fixed parameterization. However, we also discuss those which seek to learn or automate components of the environment itself.

To provide sufficient background information, we introduce the predominant classes of RL algorithms. Deep RL typically considers the problem of finding a *policy* $\pi_\theta : \mathcal{S} \to \mathcal{A}$

1. γ in many instances is treated as a hyperparameter, e.g., to make optimization tractable during training, while evaluation usually reports undiscounted rewards (i.e. $\gamma = 1$).

Notation	Meaning
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
P	Transition function
R	Reward function
$\mathcal{O}$	Observation space
ρ_o	Initial state distribution
$\mathcal{T}$	Set of terminal states
γ	Discount factor
t	Inner loop environment timestep
T	Maximum inner loop environment timestep
n	Outer loop trial index
N	Maximum outer loop trial index
V	State-value function
Q	Action-value function
J	Expected total reward function
L	Loss function
f	Objective function
θ	Neural network parameters
ϕ	Secondary neural network parameters
B	Batch size
λ	Bootstrapping hyperparameter
ζ	Underspecified components such as hyperparameters or selected algorithms
η	Subset of hyperparameters that are differentiable

Table 1: Table of notation used.

parameterized by θ (i.e. the weights of a neural network) to maximize cumulative reward:

$$\max_{\theta} J(\theta;\zeta) \text{ where } J(\theta;\zeta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t\geq 0}\gamma^t r_t\right], \tag{1}$$

where $\tau = (s_0, a_0, r_0, s_1, ...)$ is a trajectory (consisting of observed states, actions and rewards denoted by s_t, a_t and r_t at timestep t) generated via π_θ's interaction with the MDP, and $\sum_{t\geq 0}\gamma^t r_t$ is the total discounted reward, where the length of τ is usually either determined by a maximum timestep count T, or early termination when the last state is part of the terminal states $\mathcal{T}$. The main focus of this survey is optimizing ζ, which here corresponds to the *underspecified* components of the RL training pipeline. As we will discuss in Section 3, ζ may refer to anything from a single hyperparameter to an entire algorithm, which may not be known in advance.

To evaluate the success of our RL training procedure, we typically have a separate *evaluation objective* $f(\zeta, \theta)$ that refers to the performance of our agent π_θ in a given distribution of MDPs.[2] Thus, the **AutoRL problem** may be framed in terms of a general bi-level

2. Note that the distribution of MDPs might be just a single MDP depending on what the practitioner's goals are.

optimization, i.e.

$$\max_{\zeta} f(\zeta, \theta^*) \quad \text{s.t.} \quad \theta^* \in \arg\max_{\theta} J(\theta; \zeta), \tag{2}$$

where $\max_\zeta f(\zeta, \theta^*)$ can be considered the *outer loop*, while $\max_\theta J(\theta; \zeta)$ can be considered the *inner loop*, which may be equivalent to minimizing some *loss*, i.e. $\min_\theta \mathcal{L}(\theta; \zeta)$, for some methods requiring automatic differentiation. The maximization procedure usually is constrained to a *budget*, with the most common being a maximum trial count N of allowed evaluations $f(\zeta_1, \theta_1^*), ..., f(\zeta_N, \theta_N^*)$. The notation used in the paper is summarized in Table 1. **Optimizing ζ will be the main purpose of this survey**, with common definitions of ζ outlined in Section 3, while optimization methods will be outlined in Section 4.

2.1 Evaluation Methods

The most common inner-loop evaluation objective f considered is simply the original cumulative reward, i.e. $f(\zeta, \theta^*) = J(\theta^*; \zeta)$, measured after a fixed number of timesteps have been used from the training environments. However, this assumes that the evaluation MDP is specified at training time, which may not always be the case (e.g. in Sim2Real problems encountered in real world robotics), and sometimes it may be desirable, for example, to train on a different distribution of tasks to improve performance on the target task (see Section 4.7) or even a distribution of tasks. Performance under distributional shifts can also be explicitly described using metrics from the recently emerging field of RL *generalization* (Zhang et al., 2018; Kirk et al., 2021), where $f(\zeta, \theta^*)$ can be defined as the *validation* reward, i.e. the reward in the outer loop, whereas $J(\theta^*; \zeta)$ can be considered the *training* reward, i.e. the reward in the inner loop. In other cases, rather than optimizing purely reward-based metrics, properties such as efficiency can be optimized, where $f(\zeta, \theta^*)$ may be defined with respect to the policy's inference speed and number of floating point operations. Optimizing multiple objectives in such a manner comes under the scope of Multi-Objective Optimization (MOO) (Jin, 2006).[3]

Furthermore, there are various ways to evaluate the performance of the outer loop as well, many of which are used in the AutoML literature. By far the most common evaluation metric reported is the best value found so far, i.e. $\max\{f(\zeta_1, \theta_1^*), ..., f(\zeta_N, \theta_N^*)\}$, when there is a notion of a maximum budget of trials N, i.e., the hyperparameters are set N times. This budget can also be defined using multiple fidelities (Cutler et al., 2014; Kandasamy et al., 2016; Song et al., 2019), such as the sum of timesteps used over the inner loops or total wallclock time. Wallclock time is especially useful as a metric when the outer-loop optimization may not be based on a multi-trial method, but rather a differentiable optimization process, e.g. DARTS (Liu et al., 2019a). Furthermore, other metrics, such as cumulative regret (Srinivas et al., 2010), defined as $\sum_{n=1}^{N} [f(\zeta^*, \theta_{max}^*) - f(\zeta_n, \theta_n^*)]$ or trial count needed to reach a certain performance threshold for f (Cho et al., 2020), may also be used for evaluating the outer loop.

A unique challenge in regards to evaluation in AutoRL arises due to natural instabilities in training, which strongly affects the optimation procedure over ζ. In RL, the result of

3. Such Multi-Objective Optimization should not be confused with Multi-Objective Reinforcement Learning (MORL) (Yang et al., 2019; Moffaert & Nowé, 2014). The former refers to multiple objectives in the outer loop while the latter refers to a vector reward signal, e.g. $\vec{\mathbf{R}}$, in the MDP under consideration.

optimizing $J(\theta^*, \zeta)$ can vary widely per training run and as a result some of the metrics mentioned above, such as the cumulative reward, cannot be robustly measured with just a single training run. As such, it is quite common in RL to occasionally have even the perfect training setup completely fail. Normally in RL, there is a common standard of reporting aggregate results from multiple training runs or *seeds* (Henderson et al., 2018; Colas et al., 2018), usually involving the mean or median of rewards, in order to resolve this issue. However, even such metrics can be very unreliable, especially for ranking different optimization methods (Agarwal et al., 2021) where the distribution of outcomes can be incredibly wide. Agarwal et al. (2021) propose to correct this by using metrics based on robust statistics, which include the interquartile mean and the optimality gap. Such issues and improvements in robust evaluation are very important in ensuring that the outer-loop optimization remains effective; we discuss this matter throughout Section 4.

2.2 Inner Loop Optimization

We now provide a brief background on popular methods of optimizing the inner loop, i.e., θ. There are several classes of methods used to optimize θ (Sutton & Barto, 2018). At a high level, such methods may be classified into *model-free* and *model-based* methods depending on whether they use a model of the environment or not.

Model-free methods: A common class of model-free methods is *policy gradients* (Sutton et al., 1999a), which seeks to optimize $J(\theta)$ via gradient descent using the fact that

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t\geq 0} r_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right], \tag{3}$$

assuming the notion of an action distribution from the policy: $a \sim \pi_\theta(\cdot|s)$. Policy gradient algorithms have been used extensively in robotics (Haarnoja et al., 2018; Tan et al., 2018), with several large scale applications (Berner et al., 2019).

In contrast, *value-based* methods seek to learn an action-value function predicting the expected discounted cumulative reward from a given state if an action is taken (Watkins & Dayan, 1992) and policy π_θ is followed thereafter:

$$Q_\theta(s,a) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t\geq 0} r_t \middle| s_0 = s, a_0 = a\right], \tag{4}$$

for each initial state-action pair $(s,a) \in \mathcal{S}\times\mathcal{A}$. The policy therefore seeks to take the action maximizing the so-called *Q-values*, i.e. $\pi_\theta(s) \in \arg\max_a Q_\theta(s,a)$. A prototypical method based on temporal difference learning (Sutton & Barto, 2018) is Q-learning (Watkins & Dayan, 1992); in its most basic form, it maintains an estimate $Q_\theta : \mathcal{S}\times\mathcal{A}\to\mathbb{R}$ of the optimal value function, and given a sequence of transition tuples $(s_t, a_t, r_t, s_{t+1})_{t\geq 0}$, updates $Q_\theta(s_t,a_t)$ towards the target $r_t + \gamma \max_{a\in\mathcal{A}} Q_\theta(s_{t+1},a)$, for each $t \geq 0$. The canonical value-based method in deep RL is DQN (Mnih et al., 2015). Since then, there has been an explosion of interest in the field, and a swathe of improvements to this algorithm (van Hasselt et al.,

2016; Schaul et al., 2016; Wang et al., 2016; Bellemare et al., 2017; Fortunato et al., 2018). These innovations were combined in the Rainbow algorithm (Hessel et al., 2018).

Actor-critic methods (Sutton and Barto (2018)) usually combine policy gradient and value-based methods. In this case, the actor represents the policy which is learned using policy gradients based on the value function learnt by the critic (see e.g., Lillicrap, Hunt, Pritzel, Heess, Erez, Tassa, Silver, & Wierstra, 2016; Mnih, Badia, Mirza, Graves, Lillicrap, Harley, Silver, & Kavukcuoglu, 2016).

The model-free methods mentioned above can be practically organized into three components: a learner, a replay buffer, and potentially multiple actors. The *learner* performs gradient descent optimization over the *replay buffer*, which contains trajectory data collected by *actors* interacting with the MDP.

Model-based methods: Alternatively, *Model-Based* Reinforcement Learning (MBRL) methods make use of models of the environment, such as the transition function P and the reward function R. This may be in the form of the *true* model provided to the agent, famously used by the AlphaGo (Silver et al., 2016) algorithm, or the model may also be *learned* during training to predict the environment dynamics, either for training a policy (Sutton, 1991; Hafner et al., 2020; Ha & Schmidhuber, 2018; Kaiser et al., 2020) or for planning (Hafner et al., 2019; Chua et al., 2018; Tassa et al., 2012; Schrittwieser et al., 2020). In addition to minimizing the losses involved for model-free methods above, this may entail further minimizing, e.g., a mean-squared error (MSE) loss or negative log likelihood (NLL) loss on learning P and R.

Planning-based approaches led to one of the most prominent successes in RL, with the AlphaGo agent making use of Monte Carlo Tree Search (MCTS, Browne et al. (2012)) guided by neural network approximators for the policy and value functions. MCTS performs a heuristic search by randomly sampling rollouts for a given state. This requires the true model of the environment. MuZero (Schrittwieser et al., 2020) takes these ideas one step further. It matches the performance of AlphaZero on Go, chess and shogi, while also performing well on Atari games with a learned model. An alternative planning approach is Model Predictive Control (MPC, Garcia et al. (1989)), which solves an optimization problem online. This has been used in Deep RL, for example, in the PETS algorithm (Chua et al., 2018) or PlaNet (Hafner et al., 2019).

In contrast, Dyna (Sutton, 1991) is a canonical MBRL algorithm that essentially treats a learned dynamics model as an environment for model-free RL. Dyna-style approaches are popular since they facilitate faster inference at deployment time (forward-pass of the policy rather than an optimization loop) and have been shown to be highly sample-efficient both from proprioceptive states (Janner et al., 2019) and pixels (Hafner et al., 2020).

2.3 Non-Stationarity of the Optimization Problem

In various domains of AI, such as evolutionary algorithms or deep learning, it is well known that some hyperparameters need to be adapted over time to achieve the best performance (see e.g., Moulines & Bach, 2011; Doerr & Doerr, 2020). A prime example for this are learning rates in deep learning. Keeping static learning rates might result in very slow learning or

even in diverging learning behavior. As deep neural networks are the commonly used method of function approximation in the current landscape of RL, it is to be expected that similar dynamic treatments of hyperparameters are needed in RL. In fact, existing optimization experiments have observed that dynamically adapting learning rates is also beneficial for model-free and model-based RL (Jaderberg et al., 2017; Zhang et al., 2021).

Additionally, the nature of RL further amplifies the potential non-stationarity of hyperparameters, even in stationary environments (Igl et al., 2021). With the trial-and-error approach to learning, an RL agent constantly (re-)generates its own training data, which is highly dependent on the current instantiation of the agent. Thus, hyperparameter settings might require very different values over the whole training run. Indeed, besides classical static optimization approaches, methods for AutoRL typically include approaches for dynamically selecting configurations. We will discuss these more in Section 4 when we introduce the current state of methods for AutoRL.

3. What Needs to be Automated?

In this section, we introduce the set of problems AutoRL seeks to solve. Figure 1 demonstrates some of the tunable components (i.e. possible definitions of ζ) in a standard RL pipeline, which range from entire algorithms to a single hyperparameter. The following subsections begin at a high level, approximately equating to the sequence of decisions that need to be made when first considering using RL to solve a problem.

3.1 How Do We Design Tasks?

When using RL for a new problem setting, designing the learning environment is usually the first problem to be considered. First of all, the user must specify a reward function $R(s, a, s')$. This is typically assumed to be provided in many RL papers and common benchmarks (Bellemare et al., 2012; Brockman et al., 2016), yet it is certainly non-trivial to be set for many real world problems. For example, for robotic manipulation, the reward could be provided once the task is completed, or given based on some intermediate goal, such as reaching the object of interest. These differences can often have a dramatic impact on the success of policy optimization (Gleave et al., 2021).

Often, it is also necessary to determine what should be provided as an observation for the agent. For example, when training from pixels, it is common to deploy a series of preprocessing steps that may be customized per environment. Indeed, one of the very first successes of deep RL made use of greyscale to simplify the learning problem and to stack consecutive frames to form the observation (Mnih et al., 2015). While such additional processing is not always needed, using raw frames directly can be demanding in terms of computation and memory requirements (Mnih et al., 2015). In addition, a large action space makes it significantly harder for an agent to begin learning; so it is often required for researchers to pick the available actions for an agent, or consider learning them (Farquhar et al., 2020).

Finally, the environments to train on can also be controlled by the designer. We discuss a wide variety of automated environment design methods in Section 4.7.

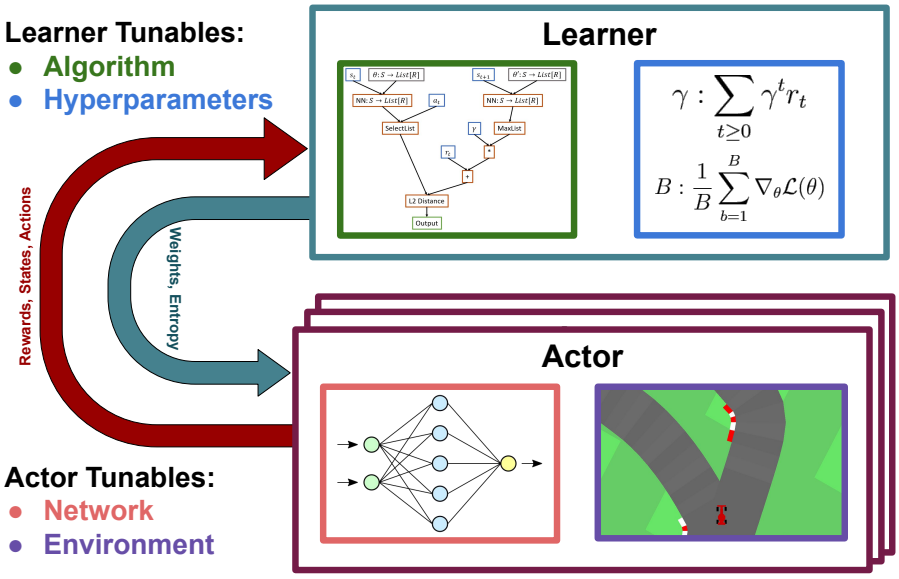


Figure 1: Components, or definitions of ζ, that have been commonly tuned in AutoRL approaches. Most pipelines consist of a learner and actors. The learner possesses tunable algorithms (e.g. Q-learning computation graph by Co-Reyes et al. (2021)) and hyperparameters (e.g.: discount factor γ and batch size B). The actors possess the network with weights, and the environment (e.g. Car-Racing by Brockman et al. (2016)).

3.2 Which Algorithm Do We Use?

Once we are able to define an MDP (or POMDP) to solve, the next question is the choice of algorithm. In cases where RL has already demonstrated strong performance, the choice may seem rather simple. For example, when playing a new Atari game, a variant of DQN can be expected to perform well. For a continuous control task from MuJoCo (Todorov et al., 2012), an actor critic algorithm such as SAC (Haarnoja et al., 2018) would likely be a strong baseline. However, even for these standard benchmarks, RL agents are known to be brittle (Henderson et al., 2018; Andrychowicz et al., 2021; Engstrom et al., 2020) and this choice is not as simple as it seems. For a completely new problem, the challenge is significantly greater, which prevents the uptake of RL for real world problems. Without expertise in the field or vast computational resources, many use cases may result in sub-optimal solutions as arbitrary algorithms are used based on success in somewhat unrelated problems.

To tackle the problem of which algorithm to use, there are multiple possible avenues. One could use a learning to learn approach (Andrychowicz et al., 2016; Chen et al., 2017) where a meta-learner learns an agent that can perform well on a set of related tasks. The agent would be trained over a distribution of tasks and learn to recognize the task during rollouts at test time. Such recognition capabilities can be achieved by a Recurrent Neural Network (RNN), for instance, that rolls out a policy depending on its state that would encode the task at hand.

One could also learn the objective function itself, where each objective function defines a new algorithm. In the most general form, RL's objective is to maximize expected cumulative reward $J(\theta; \zeta)$ in a given environment. In the context of deep RL, this objective is not differentiable with respect to model parameters; therefore proxy objectives are used in practice, which have a strong impact on the learning dynamics. Indeed, many advancements in deep RL result from improvements in the objective functions, such as double Q-learning (van Hasselt et al., 2016), distributional value functions (Bellemare et al., 2017) and yet more (Lillicrap et al., 2016; Schulman et al., 2017a; Haarnoja et al., 2018). These algorithms have fundamental differences in the objective functions, which are designed by human experts. Even smaller tweaks to known objective functions can make a big difference, e.g. Munchausen-DQN (Vieillard et al., 2020) and CQL (Kumar et al., 2020b), yet these tweaks require significant theoretical analysis and empirical experiments to validate their effectiveness.

One could also make a categorical choice between existing algorithms based on multiple evaluations. In AutoML, such a choice of algorithm can be automated through algorithm selection (Rice, 1976). In algorithm selection, a meta-learning model is used to decide which algorithm to use for an environment at hand. The so-called selector is trained based on past observed performances and features of the environment. While this alleviates the need of expert knowledge, it requires potentially many resources to gather enough performance data to learn a well-performing selector.

We discuss works related to learning algorithms in RL in Section 4.6.

3.3 What about Architectures?

Many of the large breakthroughs in machine learning have come through the introduction of new architectures (Krizhevsky et al., 2012; He et al., 2016; Vaswani et al., 2017). To automate this discovery, the field of *Neural Architecture Search* (NAS) (Elsken et al., 2019) has become an active area of research over the past few years. In contrast to supervised learning, very little attention has been paid to the design of neural architectures in RL. For tasks from proprioceptive states, it is common to use two or three hidden layer feedforward MLPs, while many works learning from pixels still make use of the convolutional neural network (CNN) architecture used in the original DQN paper, known as "Nature CNN". More recently, it has become commonplace to make use of the network proposed in the IMPALA paper (Espeholt et al., 2018), which is now referred to as "IMPALA-CNN". While the IMPALA-CNN has been shown to be a stronger architecture for vision and generalization (Cobbe et al., 2020, 2019b), there has been little research into alternatives, although some have focused on using attention modules for policies (Parisotto et al., 2020; Tang et al., 2020; Zambaldi et al., 2019).

Along with IMPALA-CNN, there is evidence that deeper and denser networks, use of different nonlinearities, as well as normalizers, such as Batch Normalization, can improve current methods across a suite of manipulation and locomotion tasks (Sinha et al., 2020; Song et al., 2020), even for the MLP setting. Kumar et al. (2020a) further expanded upon the downside of underparameterization for value-based methods. In general, there remains little conceptual understanding (and uptake) on architectural design choices and their benefits

in the RL domain. While Cobbe et al. (2020, 2019b) have shown *overparameterization* and batch normalization (Ioffe & Szegedy, 2015) effects in RL generalization, it is unclear whether they can be explained by supervised learning theories, i.e. *implicit regularization* (Neyshabur, 2017; Neyshabur et al., 2015), Neural Tangent Kernels (Jacot et al., 2018; Arora et al., 2019), complexity measures (Neyshabur et al., 2019) and landscape smoothness (Santurkar et al., 2018).

We discuss works seeking to address neural architectures in RL in a variety of sections, given that it can be addressed using many different approaches.

3.4 Last but not Least: What about Hyperparameters?

Having defined the task and selected (or learned) an algorithm and architecture, the last remaining challenge is selecting hyperparameters.[4] The most widely studied area of AutoRL is the sensitivity of RL algorithms to hyperparameters. Indeed, in one of the best-known studies, Henderson et al. (2018) found that many of the most successful recent algorithms were brittle with respect to hyperparameters, implementation, or even seed, while Islam et al. (2017) noted that it is challenging to compare benchmark algorithms from different papers given the impact of hyperparameters on performance.

One of the better understood hyperparameters is the discount factor γ, which determines the time-scale of the return. Prokhorov and Wunsch (1997), as well as Bertsekas and Tsitsiklis (1996), found that lower discount factors led to faster convergence, with the potential risk of leading to myopic policies. Singh and Dayan (1996) explored the manner in which TD learning is sensitive to the choice of its step size and eligibility trace parameters.

Another important hyperparameter, which could be possibly considered as choosing the whole algorithm itself, is the *exploration-exploitation trade-off* in RL. This trade-off is a significant determinant of performance and has been explored in several papers (Badia et al., 2020; Dabney et al., 2021; Biedenkapp et al., 2021). At a higher level, the outer loop algorithm must also consider a similar trade-off, which remains a key component of research in methods such such as Bayesian optimization and Population-Based Training and deserves special attention for an AutoRL pipeline.

Looking at specific algorithms, Andrychowicz et al. (2021) conducted an extensive investigation into design choices for on-policy actor critic algorithms. They found significant differences in performance across loss functions, architectures and even initialization schemes, with significant dependence between these choices. Obando-Ceron and Castro (2020) also explored design choices for off-policy RL, highlighting the differing performance for MSE vs. Huber loss functions, while also assessing the importance of n-step returns, which Fedus et al. (2020) and Rowland et al. (2020) also studied. Bas-Serrano et al. (2021) showed that the performance improved by ensuring the convexity in the Q-learning using logistic Bellmann error function. Furthermore, Liu et al. (2021b) showed that the choice of regularizer can also significantly impact performance.

4. We note, however, that to search for neural architectures one should already have meaningful hyperparameters. This poses a chicken-and-egg problem, which can be side-stepped by jointly searching for neural architectures and hyperparameters as, for example, done by Runge et al. (2019).

Finally, beyond the well-flagged hyperparameters, there are even significant *code level* implementation details. Henderson et al. (2018) identified this and showed that different codebases for the *same algorithm* produced markedly different results. Furthermore, Engstrom et al. (2020) investigated implementation details in popular policy gradient algorithms (PPO, Schulman et al. (2017a) and TRPO Schulman et al. (2015)), where details such as reward normalization were found to play a crucial role in RL performance. With this area being the most studied component of AutoRL, we discuss various different solution approaches to the hyperparameter optimization problem in Sections 4.1 to 4.5.

4. Methods for Automating Reinforcement Learning

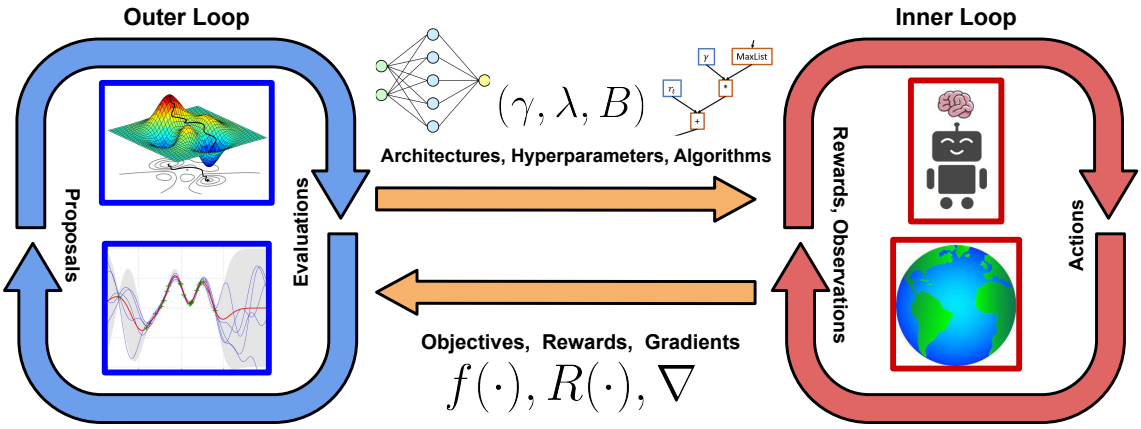


Figure 2: Most approaches can be organized in terms of an outer loop and inner loop, where the outer loop sends specifications, such as architecture specification, hyperparameters, and algorithms, while the inner loop sends back objectives, rewards, and gradients. Images from (Pérez-Cruz et al., 2013; Co-Reyes et al., 2021; Guo, 2020).

In this section, we survey methods for AutoRL, spanning a wide range of communities and encompassing a broad array of techniques. In general, most methods can be conveniently organized in terms of a combination of an *inner loop*, which consists of a standard RL pipeline, and an *outer loop*, which optimizes the agent configuration. Each loop can be optimized via either blackbox optimization or gradient-based methods, although the combination of gradients for the outer loop and blackbox for the inner loop blackbox is not possible, as the inner loop blackbox setting would make gradients unavailable. This is shown in Figure 2 and Table 2.

We summarize the taxonomy of AutoRL methods along broad classes (which often overlap) in Table 3. Each has its own pros and cons, as there are inevitably trade-offs, such as sample vs. compute efficiency. The scope for each method is also vastly different, from tuning a single hyperparameter, to learning a reward function, or an entire algorithm from scratch. In each subsection we first discuss related methods before proposing open problems.

	Benefits	Downsides
Blackbox	Flexible/No assumptions on pipeline	Can be sample-inefficient
Gradient	Scalable and sample-efficient	Inflexible/Requires differentiable pipelines

Table 2: Comparison between gradient-based and blackbox approaches.

Table 3: A high level summary of each class of AutoRL algorithms. In each case, the properties are a generalization, and specific algorithms in each class may vary.

Class	Algorithm properties					What is automated?
Random/Grid Search (4.1)	†††	■	⇉	✓	≐	hyperparameters, architecture, algorithm
Bayesian Optimization (4.2)	†††	■	⇉	✓	≐	hyperparameters, architecture, algorithm
Evolutionary Approaches (4.3)	†††	■	⇉	✓	≈	hyperparameters, architecture, algorithm
Meta-Gradients (4.4)	†	∇	→	●	≈	hyperparameters
Blackbox Online Tuning (4.5)	†	■	→	✓	≈	hyperparameters
Learning Algorithms (4.6)	†††	■	⇉	●	≐	algorithm
Environment Design (4.7)	†††	■	⇉	●	≈	environment

† only uses a single trial, ††† requires multiple trials

∇ requires differentiable variables, ■ works with non-differentiable hyperparameters

⇉ parallelizable → not parallelizable

✓ works for any RL algorithm, ● works for only some classes of RL algorithms

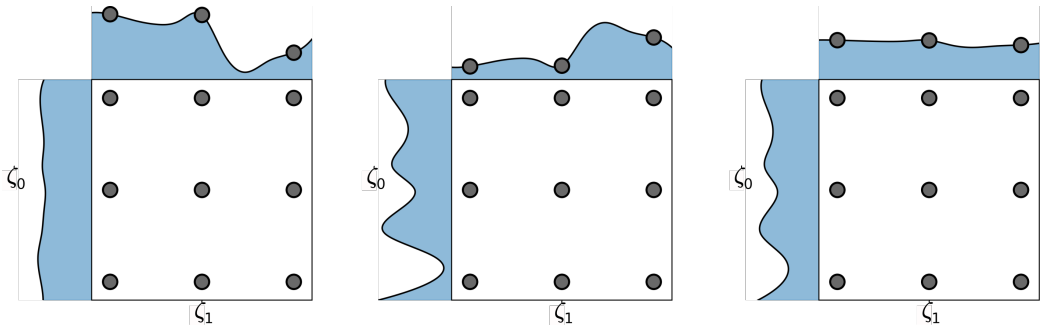
≐ static optimization, ≈ dynamic optimization

4.1 Random/Grid Search Driven Approaches

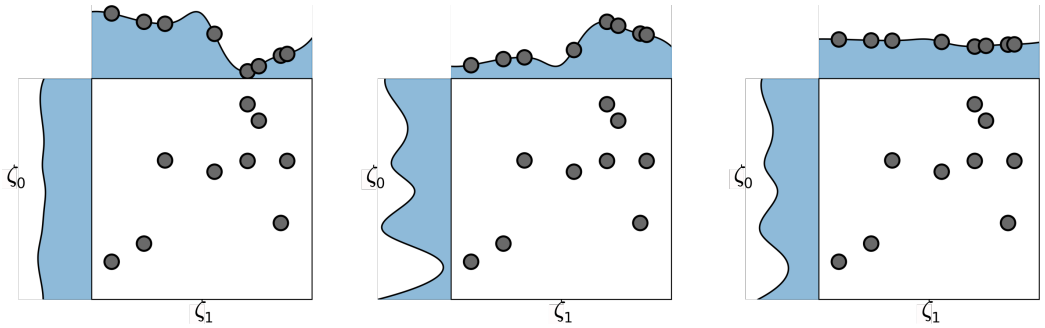
We begin by discussing the most simple methods: Random Search (Bergstra & Bengio, 2012) and Grid Search (Hutter et al., 2019). As the names imply, Random Search randomly samples hyperparameter configurations from the search space and Grid Search divides the search space into fixed grid points which are evaluated. Due to their simplicity, Random Search and Grid Search can be used to select a list of hyperparameters[5], exhaustively evaluate all of them and pick the best configuration, making them very general approaches. Indeed, Grid Search is still the most commonly used method in RL, featuring in the vast majority of cases where hyperparameters are tuned, but should not be considered as the most efficient approach.

These classical approaches do not take the potential non-stationarity of the optimization problem (recall Section 2.3) into account. We describe the issue below, using Figure 3. Assume we use these procedures to *minimize* the loss of an RL algorithm that has two continuous hyperparameters ζ_0 and ζ_1 (see Figure 3). We evaluate 9 hyperparameter settings in parallel. At $n = 0$ (First Column of Figure 3), we observe the performance of all settings as indicated by the blue shaded areas. It is clear that changes in ζ_0 result in little difference in

5. As mentioned in Table 3, these approaches are applicable to architectures and algorithms as well, but they are usually only applied to hyperparameters. They are applied to architectures in the form of selecting, e.g., number of layers of the network or the width of a layer, but we include such basic architectural choices under hyperparameters.



(a) Grid Search at $n = 0$, $n = N$ and averaged over n.



(b) Random Search at $n = 0$, $n = N$ and averaged over n.

Figure 3: The non-stationarity of the AutoRL problem poses new challenges for classical (hyperparameter) optimization methods. Generally, grid and random search keep the hyperparameters ζ_0 and ζ_1 fixed throughout a run. If the loss landscape changes during the run (see difference between leftmost and middle images) these methods optimize the average performance over time (right most images). Circles represent hyperparameters that are sampled at the beginning of the optimization procedure and the blue shaded area depicts the RL agent's loss. This figure is based on a figure for the stationary case by Bergstra and Bengio (2012).

the agents performance, whereas changes in ζ_1 result in large performance differences. While random search is likely to observe the performance of 9 unique values per hyperparameter, grid search only observes 3. At $n = N$ (Middle Column of Figure 3), we observe that the loss landscape has shifted and that now both hyperparameters have an impact on the final performance. Furthermore, while in the early phase of the optimization procedure, large values of ζ_1 were beneficial, now smaller values are preferable. Looking at the loss averaged over the whole optimization procedure (Last Column of Figure 3) we can see that this view, although often used with static configuration approaches, abstracts away a lot of information. For example, both approaches would view ζ_1 as much less important than ζ_0 even though it had an overall larger impact during the whole optimization procedure. At $n = 0$, low values for ζ_1 resulted in bad performance whereas at $n = N$, large values resulted in bad performance. If we now average the performance over time, it appears to have little to no impact. Similarly, looking only at the final performance hides a lot of valuable information

and can lead to sub-optimal hyperparameter configurations at different stages of the run. Thus, these methods might be insufficient in cases where changes in hyperparameters are required during the run to achieve the best performance.

A common way to improve the performance of Random Search is with Hyperband (Li et al., 2017), a bandit-based configuration evaluation for hyperparameter optimization. It focuses on speeding up Random Search through adaptive resource allocation and early-stopping. The hyperparameter optimization task is considered as a pure-exploration non-stochastic infinite-armed bandit problem where a predefined resource like iterations, data samples, or features is allocated to randomly sampled configurations. In particular, Hyperband uses Successive Halving (Jamieson & Talwalkar, 2016) which allocates a budget to a set of hyperparameter configurations. This is done uniformly, and after this budget is depleted, half of the configurations are rejected based on performance. The top 50% are kept and trained further with twice the budget. This process is repeated until only one configuration remains. Zhang et al. (2021) used Random Search and Hyperband for tuning hyperparameters of their MBRL algorithm. Furthermore, Zhang et al. (2021) analyzed the correlation of configuration performance across the considered budgets, and based on low correlations, they moved from static to dynamic configuration methods.

Open Problems The key drawback of random/grid search approaches is being unable to leverage the information regarding promising regions of the hyperparameter search space for making informed decisions. During the optimization, this information becomes clearer as more and more hyperparameter configurations are tried out. However, as these approaches solely focus on exploration and do not exploit this information, they typically scale poorly as the search space increases in size.

4.2 Bayesian Optimization

The next set of approaches involved are those which inherently have a notion of sequential decision making. Bayesian Optimization (BO, Mockus (1974), Jones et al. (1998), Brochu et al. (2010)) is one of the most popular approaches to date, used for industrial applications (Golovin et al., 2017; Balandat et al., 2020; Perrone et al., 2021) and a variety of scientific experimentation (Frazier & Wang, 2015; Hernández-Lobato et al., 2017; Li et al., 2018; Griffiths & Hernández-Lobato, 2020; Tran et al., 2021; van Bueren et al., 2021). For RL applications, one of the most prominent uses of BO was for tuning AlphaGo's hyperparameters, which include its core Monte Carlo Tree Search (MCTS) (Browne et al., 2012) hyperparameters and time-control settings. This led to an improvement of AlphaGo's win-rate from 50% to 66.5% in self-play games (Chen et al., 2018). In Figure 4, we demonstrate the general concept of Bayesian Optimization in the RL case.

Multi-Fidelity Algorithms: Because of the fairly large optimization overhead, standard BO methods require expensive black-box evaluations, such as training an ML algorithm to the end to observe the accuracy. This way of training is time-consuming and costly because each set up may take hours or days to run. An alternative choice to the standard BO for tuning RL hyperparameters is leveraging different fidelities (Cutler et al., 2014; Kandasamy et al., 2016; Falkner et al., 2018; Song et al., 2019), such as different number of seeds and

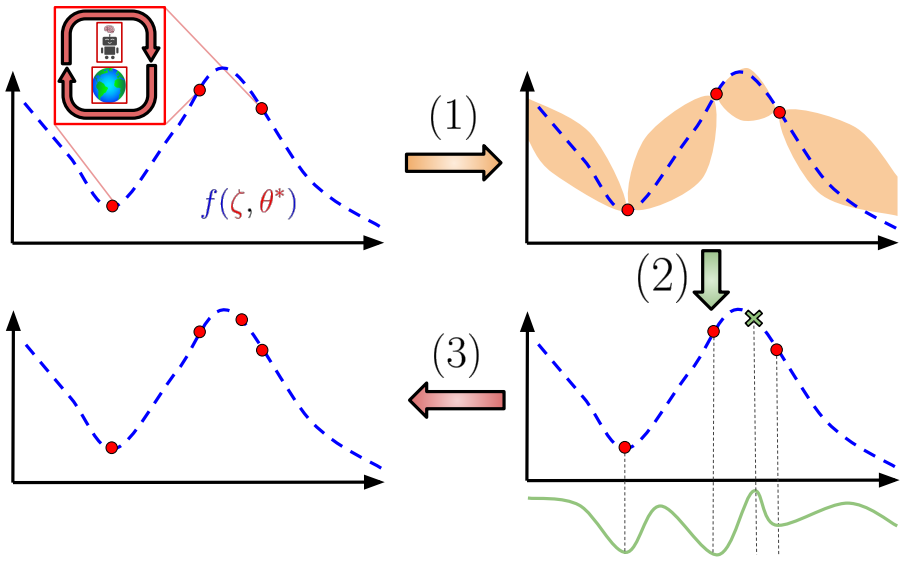


Figure 4: Bayesian Optimization, which consists of 3 main steps. (1): A regressor (commonly a Gaussian Process) is used to construct uncertainty estimates on the blackbox function $f(\cdot)$ given previous evaluation data. (2): An acquisition function is constructed from the regressor, which represents the explore-exploit tradeoff on f. (3): The argmax of the acquisition function is used for the next trial.

epochs, or runtimes. While one option is to use the model to choose both a configuration and a fidelity to evaluate it at (Kandasamy et al., 2016; Klein et al., 2017; Song et al., 2019), a simpler and more robust approach is to decide about the fidelities using Hyperband (HB) and only choose the configurations that Hyperband should consider at the beginning of each iteration with BO instead of random selection. This is the method used in the popular multi-fidelity BO method BOHB (Falkner et al., 2018), which trains BO's model for a given fidelity on the configurations that were evaluated at this fidelity so far and which can also efficiently and effectively take advantage of parallel resources. BOHB has been shown to be orders of magnitude faster than vanilla BO in some cases and to find dramatically better results of PPO on the cartpole swing-up task from Gym using seeds as a fidelity (Falkner et al., 2018). It also powered one of the first full AutoRL applications: optimizing a PPO approach that was tasked to learn to design RNA (Runge et al., 2019). In that application, BOHB was used with a runtime budget as fidelity to jointly optimize the neural architecture and hyperparameters of PPO, along with the formulation of the problem as a Markov decision process (MDP), ultimately learning an RNA design method that was over 1000 times faster than the previous state of the art. It is noteworthy that AutoRL made this project possible to be carried out by two undergraduate students with no prior experience with RL.

Exploiting additional information: Recent works have attempted to exploit internal information readily available from RL applications for improving the optimization, which we describe now. BOIL (Nguyen et al., 2020) enhances tuning performance by modelling the training curves, providing a signal to guide search. It transforms the whole training curve

into a numeric score to represent high vs low-performing curves. Then, BOIL introduces a data augmentation technique leveraging intermediate information from the training curve to inform the underlying GP model. The algorithm not only selects a hyperparameter setting to evaluate, but also how many epochs it should evaluate. Thus, BOIL makes it possible to run a larger number of cheap (but high-utility) experiments, when cost-ignorant algorithms would only be able to run a few expensive ones, resulting in greater sample-efficiency than traditional BO approaches when tuning deep RL agents. It is also possible to exploit *external* knowledge of the maximum achievable return, i.e. knowing $\max_\tau \sum_{t\geq 0} \gamma^t r_t$ to improve hyperparameter tuning in RL. This optimum value is available in advance for some RL tasks, such as the maximum reward (when $\gamma = 1$) being 200 in CartPole. To utilize such knowledge, Nguyen and Osborne (2020) proposed to (i) transform the surrogate model and (ii) select the next point using the optimum value. They showed that exploiting this external information can improve the optimization.

Empirical results: BO has been successfully used in various RL case studies. For example, Hertel et al. (2020) employed successive halving (Jamieson & Talwalkar, 2016), random search and BO and concluded that BO with a noise robust acquisition function is the best choice for hyperparameter optimization in RL tasks. Finally, Lu et al. (2021) showed it was possible to improve the performance of agents trained with offline RL when tuning hyperparameters using BO (Wan et al., 2021).

Open Problems BO-based approaches usually perform static tuning, which may not be the most effective for RL. While there are BO approaches (Parker-Holder et al., 2020a) that take the temporal nature of the optimization, especially relevant for RL, into account, these are scarce to the best of our knowledge.

4.3 Evolutionary Approaches

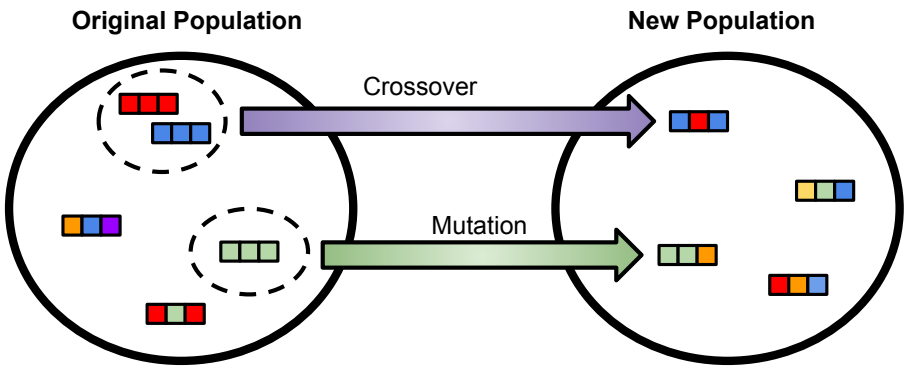


Figure 5: Most evolutionary algorithms consist of updating entire populations, usually by mechanisms such as crossover (between multiple genomes) or mutation (on a single genome).

Evolutionary approaches are widely used for various optimization tasks in different fields. These may have different ways of representing the tasks and mechanisms to mutate, recombine, evaluate and select parameters. Such mechanisms are represented in Figure 5.

Neuroevolution: While many in the deep RL community are familiar with Evolution strategies (ES, Rechenberg (1973)), a specific kind of evolutionary approach that can be used in lieu of an RL algorithm to optimize policies (Such et al., 2018; Lehman et al., 2018; Chrabąszcz et al., 2018), there remains little cross pollination with the broader evolutionary computation community. One particular subfield relevant for AutoRL is neuroevolution (Stanley et al., 2019), which has been used to evolve both weights and architectures, inspired by biological brains. In particular, the NEAT algorithm was shown to effectively evolve architectures for Pole Balancing in the early 2000s (Stanley & Miikkulainen, 2002), predating the surge of interest in deep RL. NEAT was made more expressive with the extension to HyperNEAT (Stanley et al., 2009), which was evaluated in a robotic control task in the original paper. Since then, NEAT and HyperNEAT have been evaluated in a series of control problems (Lee et al., 2013; Gomez et al., 2006; Clune et al., 2009; Risi & Stanley, 2013), even training agents to play Atari games prior to the seminal DQN paper (Hausknecht et al., 2014). More recently WANNs (Gaier & Ha, 2019) showed it was possible to use NEAT to solve RL tasks by only evolving network topology, using a single randomly initialized weight parameter.

Hyperparameter Optimization: Evolutionary methods have also been used to search for the hyperparameters of RL algorithms. Eriksson et al. (2003) used a real-number Genetic Algorithm (GA, Michalewicz (2013)), which encodes the hyperparameters of the RL algorithm via genes in every individual of the population, to tune the hyperparameters of SARSA(λ), applying the approach to control a mobile robot. Fernandez and Caarls (2018) used GAs to tune the hyperparameters of RL algorithms in simple settings and achieved good performance by combining with automatic restarting strategies (Fukunaga, 1998) to escape from local minima. Similarly, Sehgal et al. (2019) also showed that GAs can be applied to improve the performance of DDPG with Hindsight Experience Replay (Andrychowicz et al., 2017) by tuning hyperparameter configurations. Ashraf et al. (2021) used a Whale Optimization Algorithm (WOA, Mirjalili and Lewis (2016)), which is inspired by the hunting strategies of humpback whales, to optimize the hyperparameters of DDPG in various RL tasks in order to improve the performance. Elfwing et al. proposed *online meta-learning by parallel algorithm competition* (OMPAC; (Elfwing et al., 2017, 2018)) which similarly to PBT (Jaderberg et al., 2017) proposed to use a population of RL agents that is trained in parallel. OMPAC has shown to be successful for tuning hyperparameters of RL agents learning to play Tetris and Atari games.

Multi-Fidelity Algorithms: Like with Bayesian optimization, evolutionary methods can be sped up by evaluating at different fidelities. DEHB (Awad et al., 2021) combined Hyperband with Differential Evolution (DE), yielding an approach that was up to 1000x faster than random search for the hyperparameter optimization of neural networks and 5x faster to optimize seven hyperparameters of the PPO algorithm (see Figure 9 in the paper).

Population Based Training: In the RL community *Population Based Training* (PBT, Jaderberg et al. (2017)) refers to *a specific class of methods* that has found widespread

and successful use for hyperparameter optimization but also neuroevolution for RL. PBT inherits ideas from many evolutionary approaches (Spears, 1995; Bäck, 1998; Gloger, 2004; Clune et al., 2008). PBT seeks to replicate how a human would observe experiments; it trains multiple agents in parallel and periodically replaces weaker agents with copies of stronger ones. Taking inspiration from Lamarckian Evolutionary Algorithms (Whitley et al., 1994), where parameters are inherited whilst hyperparameters are evolved, PBT seeks to "exploit" stronger agent weights, while "exploring" the hyperparameter space, typically through random perturbations. The benefit of this procedure is that it is possible to explore the hyperparameter space with the same wall clock time as a single training run, given access to parallel computational resources.

Another key benefit of PBT and evolutionary approaches is the ability to learn hyperparameter *schedules*, which were shown in the original paper to be particularly effective in RL, likely due to the non-stationarity of the problem (see: Section 2.3). As such, PBT has had a prominent role in many high profile RL publications (Schmitt et al., 2018; Liu et al., 2019b, 2021a; Espeholt et al., 2018; Jaderberg et al., 2019; Team et al., 2021; Zhang et al., 2021).

In the following we give a simplified view on how evolutionary (and PBT-like) approaches are able to produce hyperparameter schedules. For this we consider a dynamic version of random search: a version that resamples the hyperparmeters after a fixed time interval (see Figure 6). Assume we use these procedures to *minimize* the loss of an RL algorithm that has two continuous hyperparameters ζ_0 and ζ_1, as in Figure 3. With the dynamic variant of random search however, we change the hyperparameter settings on the fly. In this example, changing the hyperparameters on the fly allows to find better performing hyperparameter schedules than keeping the hyperparameters fixed throughout the run. Although taking the non-stationarity of the optimization problem into account can result in better performing agents, it poses interesting new challenges for the field of AutoRL. PBT-like and evolutionary approaches are a natural way of handling the non-stationarity, although they might require far too many resources in cases where hyperparameters need not be changed on the fly.

In recent times, there has been a series of additional improvements over PBT-style algorithms. Franke et al. (2021) proposed SEARL and showed it is possible to share experience amongst agents in an off-policy setting, leading to significant efficiency gains. Zhang et al. (2021) proposed PBT-BT which included a backtracking component to the exploit step of PBT, and evaluated the approach on a challenging MBRL setting. Finally, Dalibard and Jaderberg (2021) addressed the greediness of the exploitation phase of PBT, and showed that higher long term performance could be achieved by maintaining sub-populations, and using a metric based on expected improvement.

Open Problems One of the key challenges of evolutionary approaches is their inefficiency, often requiring thousands of CPU cores to achieve strong results. While this may be possible for well-equipped industrial labs, it renders many methods impractical for small to medium sized groups and in many practical applications. Furthermore, there remain very few cases of large-scale evolutionary approaches being applied to both hyperparameters (ζ) and neural network parameters (θ), although recent efforts such as ES-ENAS (Song et al., 2021) have attempted to do so by combining different evolutionary algorithms.

A disadvantage of PBT-style methods is the relative data inefficiency. This will be particularly important when increasing the search space beyond a handful of hyperparameters

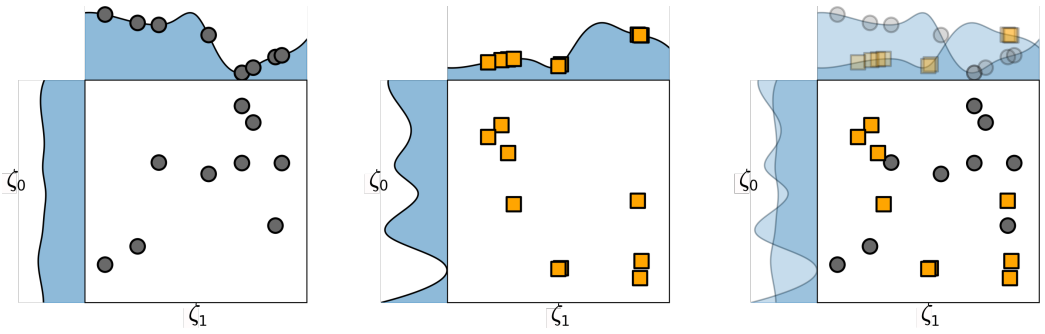


Figure 6: Dynamic random search at $t = 0$ (first column), $n = N$ (second column) and averaged over t. Dynamic random search resamples the hyperparameters ζ_0 and ζ_1 after a fixed time interval N. Circles represent hyperparameters that are sampled at the beginning ($n = 0$) of the optimization procedure, orange squares hyperparameters that were sampled at $n = N$ and the blue shaded area depicts the RL agent's loss. This figure is based on Figure 1.1 from Hutter et al. (2019).

as typically tuned with PBT. In addition, the population size is currently a meta-parameter which has a significant impact on the results; so far there has been very little research understanding how the efficacy of PBT works when this changes.

4.4 Meta-Gradients for Online Tuning

Meta-gradients provide an alternative approach to dealing with the non-stationarity of RL hyperparameters. The meta-gradient formulation is inspired by meta-learning methods, such as MAML (Finn et al., 2017), which optimize both an inner and outer loop with *gradients*. In particular, meta-gradient methods specify a subset of their (differentiable) hyperparameters to be meta-parameters η, e.g., bootstrapping, discount factor and learning rate. In the inner loop, the agent is optimized with a fixed η, taking gradient steps to minimize a (typically fixed) loss function. In the outer loop, η is optimized by taking gradient steps to minimize an outer loss function. Each specific choice of the inner and outer loss function defines a new meta-gradient algorithm.

In the original meta-gradient paper, Xu et al. (2018), used IMPALA (Espeholt et al., 2018) and set $\eta = \{\lambda, \gamma\}$, the bootstrapping hyperparameter and discount factor. When evaluated on the full suite of Atari games, the meta-gradient version improved over the baseline agent by between 30% and 80% depending on the evaluation protocol used. This work was extended to all differentiable IMPALA hyperparameters with *Self-Tuning Actor-Critic* (STAC, Zahavy et al. (2020)). STAC introduced a new loss function to handle varying hyperparameters, which allowed it to further boost performance on the same set of Atari games, as well as robotic benchmarks. In addition, Zahavy et al. (2020) also introduced a means to integrate meta-gradients with auxiliary tasks, producing STACX, which saw further gains. Interestingly, the agents learned non-trivial hyperparameter schedules which could not have been tuned by hand as they were not smoothly varying or static schedules usually used in manual tuning. More recently, Flennerhag et al. (2021) proposed *Bootstrapped*

Meta-Learning, which first bootstraps a target from the meta-learner, then seeks to match it in a meaningful space, making it possible to extend the meta-learning horizon. This results in state-of-the-art performance for model free agents on the Atari benchmark, by tuning multiple hyperparameters on the fly.

Meta-gradients have also been used to discover auxiliary tasks (Veeriah et al., 2019), discover options (Veeriah et al., 2021) and learn RL objectives online, demonstrating strong asymptotic performance (Xu et al., 2020). Importantly, (Xu et al., 2020) showed that it was more effective to meta-learn the *target* rather than the full update rule, with the FRODO algorithm capable of producing strong asymptotic performance. Finally, in the field of NAS, differentiable architecture search can also be viewed as a meta-gradient approach. The DARTS algorithm (Liu et al., 2019a) and its successors have become widely popular as well in the NAS community, due the effective use of gradient-based search. Recently this approach has been shown to be effective in RL, finding effective architectures in challenging environments (Miao et al., 2021; Akinola et al., 2021).

Open Problems Meta-gradient methods have two well-known weaknesses. First, they typically rely on their meta-parameters being initialized well, which itself is a hyperparameter optimization problem. In addition, current meta-gradient methods cannot tune non-differentiable hyperparameters, e.g., the choice of optimizer or the activation function. Nonetheless, they remain one of the most efficient approaches, and offer the potential to improve upon existing strong algorithms with known hyperparameters.

4.5 Blackbox Online Tuning

The strength of both PBT and meta-gradients is the ability to adapt hyperparameters on the fly. However, these are not the only approaches to do so. Indeed, a variety of other methods have been considered, from blackbox methods to online learning inspired approaches. This section focuses on single agent approaches to adapt on the fly in settings where the hyperparameters are not (necessarily) differentiable.

Adaptive methods for selecting hyperparameters have been prominent since the 1990s. Sutton and Singh (1994) proposed three alternative methods for adaptive weighting schemes in TD algorithms. Kearns and Singh (2000) derived upper bounds on the error of temporal-difference algorithms, and used these bounds to derive schedules for λ. Downey and Sanner (2010) used Bayesian Model Averaging to select the λ bootstrapping hyperparameter for TD methods. More recently, White and White (2016) proposed λ-greedy to adapt λ as a function of the state and achieve an approximately optimal bias-variance trade-off. Paul et al. (2019) proposed HOOF which uses random search with off-policy data to periodically select new hyperparameters for policy gradient algorithms.

Several algorithms make use of bandits to adapt hyperparameters. In a distributed setting, Schaul et al. (2019) proposed to adapt several behavioral hyperparameters, such as the degree of stochasticity of an agent, using a notion of learning progress as feedback. This idea inspired Agent57 (Badia et al., 2020) which adaptively selects from several exploration policies to become the first algorithm to achieve human level performance on all 57 games in the Arcade Learning Environment (Bellemare et al., 2012). Other methods have had

success using bandits to select the degree of exploration (Ball et al., 2020), the degree of optimism for off-policy methods (Moskovitz et al., 2021) or the amount of diversity to add to a population of agents (Parker-Holder et al., 2020b). Finally, Riquelme et al. (2019) considered adaptively switching between Temporal Difference (TD) learning and Monte Carlo (MC) policy evaluation using learned confidence intervals that detect biases of TD estimates.

Open Problems One of the challenges of these approaches is the limited scope for the search space, since the bandit algorithm must be able to explore all arms. In addition, most bandit algorithms assume the arms to be independent which may lead to decreased efficiency by removing information which is known to the algorithm designer.

4.6 Learning Reinforcement Learning Algorithms

Initial approaches to learning RL algorithms employed the learning to learn formulation. Wang et al. (2016) and Duan et al. (2016), both, use a meta-learner that updates the weight of an agent equipped with an RNN to learn over a distribution of interrelated RL tasks. The former's main interest was in structured task distributions (e.g., dependent bandits) while the latter focussed on unstructured task distributions (e.g., uniformly distributed bandit problems and random MDPs). Wang et al. (2016) even showed generalization, to some extent, beyond the exact training distribution encountered. Related to these, Rakelly et al. (2019) introduced PEARL, which adapts to new tasks by performing inference over a latent context variable on which the policy is conditioned. They use Variational Autoencoders (VAEs) (Kingma & Welling, 2014) to perform such inference. Zintgraf et al. (2020) introduced variBAD, which uses meta-learning to utilise knowledge obtained in related tasks and perform approximate inference in unknown environments to approximate Bayes-optimal behaviour. They also used VAEs perform such inference but used an RNN as the encoder.

Recently however, there has been increased interest in learning to learn or Meta-RL, which aims to automate the design of RL loss functions. A key insight is that a loss function can be viewed as a parameterizable object that can be learned from data. Instead of using a fixed loss function $\mathcal{L}(\theta)$, one can construct a family of loss functions $\mathcal{L}(\theta;\zeta)$ parameterized by ζ, and seek to maximize the expected reward $J(\theta;\zeta)$ via optimizing the surrogate loss $\mathcal{L}(\theta;\zeta)$. In what follows, we separate ζ into two cases: 1) neural loss functions where ζ encodes neural network parameters, and 2) symbolic loss functions where ζ encodes computational graphs.

Neural loss functions: In this case, the loss function is a neural network with parameters ζ which may be optimized via ES or gradient based methods. For example, in Evolved Policy Gradient (Houthooft et al., 2018), the inner loop uses gradient descent to optimize the agent's policy against a loss function provided by the outer loop. The policy's performance is used by the outer loop to evolve the loss parameterization, using ES.

Gradient based methods are closely related to meta-gradient methods, as they use second-order derivatives to optimize ζ. Bechtle et al. (2020) described a general framework for meta learning with learned loss functions. Most meta-gradient RL algorithms, as mentioned in Section 4.4, start with a human designed loss function $\mathcal{L}(\cdot)$ and modify it with parameterization $\mathcal{L}(\theta;\zeta)$ to allow inner loop and outer loop procedures. In the inner loop, gradient descent via $\nabla_\theta \mathcal{L}(\theta;\zeta)$ obtains $\theta^*(\zeta)$, which can be viewed as a function of ζ. In

the outer loop, the quality of ζ is then measured by $\mathcal{L}(\theta^*(\zeta);\zeta)$, and we can optimize ζ via gradient descent using $\nabla_\zeta \mathcal{L}(\theta^*(\zeta);\zeta)$. This approach requires computing second-order information, i.e. $\nabla_\zeta \nabla_\theta$, although usually only Jacobian-Vector products are needed for the pipeline, which are readily available in popular autodifferentation libraries (Jax (Bradbury et al., 2018), Tensorflow (Abadi et al., 2015), Pytorch (Paszke et al., 2019)).

Examples which employ the above technique include MetaGenRL (Kirsch et al., 2020), which is an extension to the DDPG actor-critic framework where the critic is trained to minimize the TD-error as usual. Meta-learning is applied to the policy update. In the original DDPG, an additional policy parameter ϕ is trained to maximize the value function $Q_\theta(s_t, \pi_\phi(s_t))$. In MetaGenRL, this is used as the outer loss $Q_\theta(s_t, \pi_{\phi^*(\zeta)}(s_t))$, and the inner loss $\mathcal{L}(\phi;\zeta)$ is modeled as an LSTM, which can be transferred to different tasks.

In Learned Policy Gradient (LPG) (Oh et al., 2020), instead of using a neural meta loss function end-to-end, an LSTM network is used to provide target policy $\hat{\pi}$ and target value $\hat{y}$. The meta loss and the task loss are defined manually. Oh et al. (2020) show that a learned update rule can also be transferred between qualitatively different tasks. More recently Kirsch et al. (2021) showed that incorporating symmetries could lead to improved transfer to unseen action and observation spaces, tasks, and environments.

Symbolic loss functions: In this case, the loss function is represented as a symbolic expression consisting of predefined primitives, akin to genetic programming. Alet et al. (2020) used a Domain Specific Language (DSL) to represent a curiosity module as a directed acyclic graph (DAG). The curiosity module provides intrinsic rewards, which are added to the loss function and can be optimized by PPO. The search space contains 52,000 valid programs and they are evaluated exhaustively with pruning techniques, such as training in a cheap environment and predicting algorithm performance ("meta-meta-RL"). Co-Reyes et al. (2021) proposed a DSL to represent the entire loss function as a DAG, and this loss function is used to train value-based agents. Unlike Alet et al. (2020), the DAG is optimized using an evolutionary algorithm called Regularized Evolution (Real et al., 2019).

Finally, regarding the categorical choice of an algorithm for RL, Laroche and Féraud (2018a) used Algorithm Selection for RL where, given an episodic task and a portfolio of off-policy RL algorithms, a UCB bandit-style meta-algorithm selects which RL algorithm is in control during the next episode so as to maximize the expected return. They evaluated their approach, among others, on an Atari game, where it improved the performance by a wide margin.

Open Problems In general, it is difficult to understand analytical properties of neural loss functions. Developing tools to empirically study these loss functions becomes crucial for explaining why they work and for understanding their generalization capabilities. For symbolic loss functions, the search space (of all possible graphs) and the search algorithm (e.g., regularized evolution) play a fundamental role, yet it is far from clear what the optimal design choices are.

4.7 Environment Design

Environment design is an important component for automating the learning in RL agents. From curriculum learning (Jiang et al., 2021b; Eimer et al., 2021b; Klink et al., 2020;

Matiisen et al., 2020; Sukhbaatar et al., 2018) to synthetic environment learning (Ferreira et al., 2021) and generation (Volz et al., 2018) to combining curriculum learning with environment generation (Bontrager & Togelius, 2020; Wang et al., 2019, 2020; Dennis et al., 2020), the objective here is to speed up the learning for the RL agents through environment design.

We organize algorithms performing Automated Environment Design according to the component of the environment (assumed to be a POMDP as defined in Section 2) that they try to automatically learn. This organization can also be seen in Figure 7.

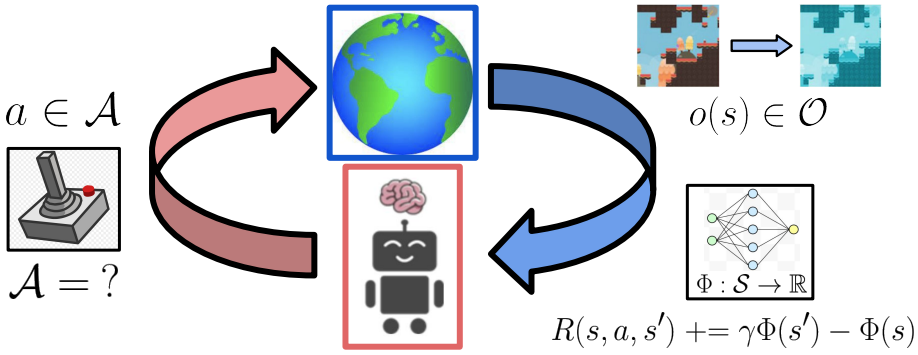


Figure 7: Representation of a typical policy + environment feedback loop, with controllable components such as the reward function R from (Ng et al., 1999), action space $\mathcal{A}$, and observation space $\mathcal{O}$ from (Raileanu et al., 2020).

Reward function, R: Zheng et al. (2018) presented a bi-level optimization approach that learns reward shaping for policy gradient algorithms using a parameterized intrinsic reward function. In an inner loop, the parameters of the agent are updated using gradient descent to increase the sum of intrinsic and extrinsic rewards, while in the outer loop, the parameters of the intrinsic reward are updated using gradient descent to increase only the extrinsic rewards. Hu et al. (2020) also presented a bi-level optimization approach which they term BiPaRS (and its variants). They use a PPO (Schulman et al., 2017b) agent in the inner loop. Furthermore, they use meta-gradients with respect to the (state-action dependent) parameters of *user-defined* reward shaping functions to learn helpful reward functions in the outer loop. Zou et al. (2019) proposed to meta-learn a potential function (Ng et al., 1999) prior over a distribution of tasks. Inspired by MAML (Finn et al., 2017), they try to adapt the meta-learned prior towards the optimal shaping function (which they derive to be equal to the optimal state-value function V). Konidaris and Barto (2006) introduced a function that preserves value information between tasks and acts as the agent's internal shaping reward. The input to this function is the part of the state space whose meaning does *not* change across tasks. This function can be used as the initial estimate for newly observed states and is further refined based on parts of the state space whose meaning *does* change across tasks. Snel and Whiteson (2010) used an evolutionary feature selection method and showed that this method chooses representations that are suitable for shaping and value functions. Their algorithm can find a shaping function in multi-task RL without

pre-specifying a separate representation. Marthi (2007) proposed automatically learning a shaped reward and a decomposed reward. Given a set of state and temporal abstractions, they create an abstracted MDP over the original MDP and learn the reward shaping using collected reward samples based on a mathematical formulation they derive.

Faust et al. (2019) investigate the effect of learning a multi-objective intrinsic reward over several RL algorithms and benchmarks with an evolutionary search. The results show that a) the utility of automated reward search correlates with the difficulty of the task, b) using simple sparse task objectives (e.g. distance travelled in MuJoCo tasks) as a fitness function leads close to identical results to using the default yet complex hand-tuned MuJoCo rewards, and c) reward search is more likely to produce better policies that the HP search on a fixed training budget and with reasonable hyperparameters.

Action Space, $\mathcal{A}$: Simplifying an environment's action space can make training significantly easier and faster. The two main approaches used to accomplish this are repeating single actions in order to reduce the number of overall decision points or to construct a new action space entirely, composed of combinations of basic actions as macro actions or options. While handcrafting has been common in action space augmentation, Sharma et al. (2017) learned the number of repetitions for a given action alongside the action selection. Biedenkapp et al. (2021) presented an extension of the idea by conditioning the amount of repetition on the predicted action itself. In automatic macro action discovery, Farahani and Mozayani (2019) showed that it is possible to combine actions into suitable macro actions by defining sub-goals to be reached in the environment; this is achieved by partitioning its transition graph. Options are similar to macro actions, but instead of executing a macro action once, each option has its own intra-option policy that is followed until a termination function defers back to the agent (Sutton et al., 1999b). Bacon et al. (2017) jointly learned both a policy across these options as well as the options themselves. Mankowitz et al. (2018) extended the idea to also learn options that are robust to model misspecification.

Observation Space, $\mathcal{O}$: Raileanu et al. (2020) proposed using a UCB bandit to select an image transformation (e.g. crop, rotate, flip) and applying it to the observations. The augmented and original observations are passed to a regularized actor-critic agent which uses them to learn a policy and value function which are invariant to the transformation.

Multiple Components, Unsupervised: We discuss here approaches that change multiple components in the environment or the whole environment itself. Most notable here are curriculum learning approaches that usually modify the state space, S, and the initial state distribution, ρ. As a result of modifying S, they naturally also change the P and R since these are functions with S a component of their domains. We begin with unsupervised methods seeking a generally robust agent, before moving to supervised approaches which typically have a target goal or task.

Wang et al. (2019) as well as Wang et al. (2020) proposed POET that generates new environments according to an environment encoding. The environments are automatically generated, either with random mutations (Wang et al., 2019) or if the new environments create a significantly different ranking of existing agents (Wang et al., 2020). These methods of generating new environments are assumed to create a diversity of environments on which to

train agents in a curriculum. Lee et al. (2020) applied the principle to controlling quadrupedal robots and found the robustness to increase significantly.

Dennis et al. (2020) proposed PAIRED that also automatically generates new environments to create a curriculum. They used a fixed environment encoding. In contrast to Wang et al. (2019, 2020), environments are chosen to maximize *regret*, defined as the difference in performance between the protagonist agent and an additional *antagonist* agent. This means the adversary is encouraged to propose the simplest environments the protagonist cannot currently solve, while provides theoretical guarantees that (at equilibrium) the protagonist follows a minimax regret strategy. Extending PAIRED, Gur et al. (2021) proposed Compositional Design of Environments (CoDE) for compostional tasks. CoDE's environment generative model builds a series of compositional tasks and environments tailored to the RL agent's skill level and makes use of a population of agents, making it possible to train agents to navigate the web. Extending the theoretical framework from PAIRED in a different direction, Jiang et al. (2021a) showed that rather than learn to *generate* new environments, it is also effective to *curate* randomly sampled ones using *Prioritized Level Replay* (PLR, Jiang et al. (2021b)). This approach maintains the theoretical properties of PAIRED while demonstrating stronger empirical performance.

Bontrager and Togelius (2020) also employed an environment generator (in addition to human created environments) and an actor-critic agent which work both cooperatively and adversarially to create a curriculum for the agent by selecting environments minimizing the agent's expected value.

Multiple Components, Supervised: If the agent's goal is solving a specific task instead of being generally more robust, it can be supported by continually progressing a simple version of the task towards this goal in a curriculum (Narvekar et al., 2020). Progression is usually tied to how well the agent currently performs in order to keep the environment difficulty at an appropriate level. A well-known example of this is OpenAI et al. (2019) who showed it is possible to use a robot hand to solve the Rubik's cube, by starting from an almost solved cube and gradually increasing the starting position complexity of both hand and cube as soon as the agent can solve the current state sufficiently well. Klink et al. (2020) applied the same principle to several physics simulation tasks, using value estimations as the progression criterion. Value estimation approaches in general have been successful for RL because their approximation of an environment's challenge level is cheap to compute (Jiang et al., 2015; Zhang et al., 2020; Eimer et al., 2021b). They are an important class in the starting state curriculum generation taxonomy proposed by Wöhlke et al. (2020). Student-teacher curriculum learning approaches can also create new task variations, e.g. by using GANs, although they only gradually increase the complexity of their distribution as the agent improves during training (Florensa et al., 2018; Matiisen et al., 2020; Turchetta et al., 2020). We can even induce a difficulty curve into environments that are immutable by using Self-Play to challenge agents with ever more difficult opponents (da Silva et al., 2019). A canonical approach is *Asymmetric Self-Play* (ASP, Sukhbaatar et al. (2018)) which proposes to use two agents: "Alice", who proposes new tasks by doing a sequence of actions and "Bob" who must undo or repeat them. ASP was also shown to be highly effective for challenging robotic manipulation tasks (OpenAI et al., 2021). Finally AMIGo (Campero et al., 2021)

and APT-Gen Fang et al. (2021) both consider settings with a fixed task, whereby the goal becomes incrementally harder as the student becomes increasingly capable.

Ferreira et al. (2021) proposed a method that learns a given target environment's transition and reward functions in order to train agents more efficiently. They framed this as a bi-level optimization problem and optimized the learnt environment (an NN) in an outer loop to maximize the reward of the agent in the inner loop. They showed that their *synthetic environments* can not only serve as a more efficient proxy for expensive target environments, but that training on them can also reduce the number of training steps.

Open Problems There are many different approaches for environment design and it is unclear which of them leads to the biggest gains in performance. This raises the possibility of: a) employing several of these methods in a single approach and studying the impact on performance and to what extent the gains are complementary; and b) evaluating the approaches on a shared benchmark. Unfortunately, there is currently also a lack of unified frameworks and shared benchmarks. To further progress towards fully automated progression through environments, more efforts like Romac et al. (2021) would be helpful in encouraging closer cooperation and better comparability.

4.8 Hybrid Approaches

Inevitably some methods do not fall into a single category. Indeed, many methods seek to exploit the strengths of different approaches, which we refer to as a *hybrid* method. In this section we define these hybrid methods as those which use more than one class of approaches from Table 3.

Chiang et al. (2019) combined evolutionary search with reward shaping to automatically select a reward function and neural architecture for navigation task. They learn a more robust policy and a hyperparameter configuration which generalizes better to unseen environments.

Jaderberg et al. (2019) achieved human-level performance in the game Quake III Arena in Capture the Flag mode, using only pixels and game points scored as input. To do this, their *FTW* agent learned reward shaping coefficients and hyperparameters jointly in an outer optimization loop. They use a bi-level optimization process with PBT, with the inner loop optimizing an IMPALA (Espeholt et al., 2018) RL agent. The outer loop is evolutionary and maintains a population of such independent RL agents which are trained concurrently from thousands of parallel matches on randomly generated environments. Each agent learns its own internal reward signal and rich representation of the world. More recently, another large-scale project saw the use of environment design, through a form of guided domain randomization, and PBT to produce "generally capable" agents in a large simulation environment (Team et al., 2021). This work attempted to achieve a more open-ended system, whereby an agent could learn to play a wide variety of games, resulting in multiple innovations, such as generational training which allowed agents to transfer behaviors across different reward functions in a PBT setup.

The Population Based Bandits (PB2 Parker-Holder et al. (2020a)) approach seeks to combine ideas from both PBT and BO. It formulates the explore step of PBT as a batch GP-bandit optimization problem and uses an upper confidence bound (UCB) acquisition

function to select new configurations. In a series of RL problems it was shown to be more sample efficient than PBT, but remains untested in larger problems or with more than a handful of hyperparameters. In addition, PB2 was recently extended to deal with mixed input hyperparameters (continuous and categorical) (Parker-Holder et al., 2021), but there has been very little work exploring kernel choice, improving the time-varying mechanism or further extending, for example, to architectures.

Open Problems In addition to the open problems for each of the individual sub-sections above, hybrid approaches face the additional open problems of how best to combine the individual approaches. Since the compute requirement of combining many methods and searching for the best combination among these could be extremely large, hybridizing approaches would require being efficiently able to prune the search space of such combinations.

5. Benchmarks

When considering different AutoRL methods, we must inevitably ask how to compare them to one another. There are no established standardized benchmarks for evaluating AutoRL methods as of yet. Instead, many works thus far have tuned components of a baseline RL method and re-used its evaluation environments, typically continuous control tasks from OpenAI Gym (Brockman et al., 2016) or discrete environments from the Arcade Learning Environment (Bellemare et al., 2012).

There is, however, an increasing emphasis on using environments that test generalization. Whiteson et al. (2009) were one of the first to propose testing agents on distributions of environments to quantify generalization in RL. The popular modern benchmark of OpenAI Procgen (Cobbe et al., 2020) is based on procedurally generated environment variations and has been used to evaluate AutoRL methods (Miao et al., 2021; Parker-Holder et al., 2021). Other procedurally generated environments include CoinRun (Cobbe et al., 2019a), MiniGrid (Chevalier-Boisvert et al., 2018), NetHack (Küttler et al., 2020), MiniHack (Samvelyan et al., 2021), Griddly (Bamford et al., 2020) or MineRL (Guss et al., 2019).

Instead of using procedural generation, Rajan and Hutter (2019) as well as Osband et al. (2020) provide simple and configurable environments with a latent causal structure. These low-level benchmarks may be used to perform experiments at a small cost at the expense of the complexity of the environments. CARL (Benjamins et al., 2021) provides similar freedom in defining environment distributions via potentially observable context features, but for more complex domains like physics simulation environments.

In addition, it can be worthwhile to consider AutoRL in more targeted, compositional tasks. Yu et al. (2019) introduced Meta-World, a benchmark that proposes 50 distinct robotic manipulation tasks and their variations to enable such learning. Alchemy (Wang et al., 2021) is another complex benchmark that also proposes a meta-distribution for RL agents which can be used to perform meta-RL to determine an underlying latent causal structure.

Learning over such distributions can not only make RL agents robust to variation but also allow them to perform few-shot learning on a distribution of environments. AutoRL for generalization (Song et al., 2019) remains an understudied problem, where it becomes a challenge to define an appropriate task for the outer loop.

Open Problems While RL benchmarks keep evolving, none of the tasks above are specific to AutoRL and thus AutoRL methods are still tested on problems with various degrees of complexity and different needs for generalization. Complicating comparability even further, AutoRL includes additional factors like configuration spaces for hyperparameters or architectures that are not standardized even across methods evaluating on the same RL benchmarks or environments. Thus specific benchmarking protocols that not only control for RL-specific environment factors but also the AutoRL setting are necessary to enable reliable comparison of different AutoRL methods. Finally, the Brax physics engine environments (Freeman et al., 2021) provide massively parallel simulation with a single GPU, which may make it possible to make rapid progress AutoRL methods.

6. Future Directions

In this section, we highlight a few specific areas which we believe will be particularly fruitful avenues for future work.

In this survey we emphasized the success of methods for AutoRL that dynamically adapt configurations during training. However, to the best of our knowledge, the non-stationarity of hyperparameters of many modern state-of-the-art algorithms has not been studied extensively. Thus, it is often not clear which hyperparameters need to be optimized dynamically and which are best optimized statically. Furthermore, it remains largely an open question if the impact of hyperparameters remains the same across environments or if different hyperparameters are important for different tasks (Eimer et al., 2021a), and why that may be the case. Methods to analyze such effects have so far only been proposed for static configuration procedures (Hutter et al., 2014; Fawcett & Hoos, 2016; Biedenkapp et al., 2017, 2018; van Rijn & Hutter, 2018) and have not yet found wide-spread use in RL.

Classical hyperparameter optimization (Hutter et al., 2019) for (un-)supervised learning considers only individual datasets when searching for well performing configurations. Similarly, when optimizing the hyperparameters of an RL agent commonly only individual environments are considered. In fact, nearly all of the discussed optimization methods throughout this paper consider only individual environments. Given the sensitivity of RL agents discussed throughout this paper (Henderson et al., 2018; Andrychowicz et al., 2021; Engstrom et al., 2020), it is to be expected that the discovered settings are not transferable to other environments. Recently however there is increasing interest in training RL agents that are capable of handling multiple (homogeneous) environments. While agents have typically used single configurations across all Atari games (Mnih et al., 2015), it remains a challenge to find robust configurations that work in a variety of settings (Eimer et al., 2021a). AutoRL approaches that tune RL agents across environments fall into the class of algorithm configuration (Hutter et al., 2009) methods, which seek to find better parameters to improve general algorithm performance. Such methods are capable of finding well performing and robust hyperparameters for a set of environments (Eggensperger et al., 2019). Recently, evaluation protocols were proposed that consider the performance of RL algorithms across a set of environments (Jordan et al., 2020; Kirk et al., 2021; Patterson et al., 2021), which may prove useful for future work in this space. Meanwhile, it also raises the problem of AutoRL-specific benchmarking, which considers different metrics such as performance improvement, data efficiency and generalization capability of the AutoRL method.

Related to the problem of algorithm configuration, algorithm selection (Rice, 1976) can be used to choose which RL algorithm to use for learning (Laroche & Féraud, 2018b). To the best of our knowledge, so far no AutoRL approach has explored the intersection of algorithm selection and algorithm configuration known as per-instance algorithm configuration (PIAC, Xu et al. (2010), Kadioglu et al. (2010)). In this framework, a selector can choose a configuration from a portfolio of well performing configurations and decides which of these to use for the environment at hand.

One important area for research is to provide a more rigorous understanding of the impact of design choices on performance. If we can understand how each component interacts with others then we can either select more amenable combinations or design search spaces to account for this dependency. This could come from empirical investigations or theoretical analysis. Indeed, recent works such as Fedus et al. (2020), Obando-Ceron and Castro (2020) and Andrychowicz et al. (2021) have provided foundations in this area, but significant work remains to be done. Furthermore, hyperparameter importance methods and analysis tools from the AutoML community (Hutter et al., 2014; Fawcett & Hoos, 2016; Biedenkapp et al., 2017, 2018; van Rijn & Hutter, 2018) have not yet been explored in AutoRL. Increasing focus in this area is likely to have profound knock-on effects on other areas of RL, especially AutoRL.

Another important design choice that impacts performance of RL agents is that of *on-* vs *off-policy* RL. It has significant implications for algorithms (e.g., V-trace for IMPALA (Espeholt et al., 2018)). While some AutoRL approaches such as SEARL can only work in the off-policy setting, making the choice of whether to use on- or off-policy algorithms could itself be considered a hyperparameter in general and studied further under AutoRL.

Another recent area of research is *offline* RL, where agents must generalize to an online simulator or real world environment from a static dataset of experiences. Research in this space remains nascent, however, it already contains a large diversity of methods, while also introducing new challenges. For example, it is challenging to get real-world policy returns for an AutoRL algorithm, so often we must make use of off-policy evaluation (Precup et al., 2000). It is possible that in this setting there may be scope to use more "traditional" AutoML methods, or it may require completely new approaches.

In addition to the discussion so far, the majority of the work in this survey addresses *single agent* RL. However, many real-world systems are in fact inherently multi-agent (Foerster, 2018). Consider, for example, self-driving cars which need to cooperate with others to safely drive on the road. When using RL in this type of problem, there is an additional challenge of how to parameterize the different agents: whether to train in a centralized or decentralized manner, while also designing reward functions and algorithms to capture the impact of individual agent actions.

Another important sub-field of RL that was beyond the scope of the survey is Multi-Objective Reinforcement Learning (MORL). While MORL aims at optimizing a vector reward from an environment, various metrics might also need optimizing in the outer loop as mentioned in Section 2.1. For example, minimizing the memory usage of an algorithm and/or the wall-clock time, which may interfere with purely optimizing the maximum reward from the environment, may be important considerations in choosing the algorithm. Even multi-task RL where performance on different environments could be traded off with each

other could be considered an instance of MORL. Such Multi-Objective Optimization could explicitly consider the Pareto front in the outer loop optimization or could even resort to heuristic design choices to constrain the design space of an AutoRL pipeline.

7. Conclusion

This paper has introduced AutoRL by discussing a wide variety of methods to automate the RL training pipeline. Indeed, unlike supervised learning, which is usually an open-loop one-step process, RL is a complete closed-loop *system*. As such, it is likely each of the components discussed has an influence on others, and if we want to train our agents end-to-end as part of a broader system, it ultimately will require a holistic solution. The challenge is compounded in RL since evaluations in RL are almost always necessarily stochastic and (potentially much more) noisy than in supervised learning, due to various sources (e.g. policy, environment), which can be a challenge for any form of automatic tuning. However, we have presented a variety of promising directions in this survey that can help overcome the challenge and which will likely provide improvements in the coming years. It is evident that AutoRL is maturing as a field and exciting possibilities lay ahead.

Acknowledgements

Jack Parker-Holder, Raghu Rajan, and Xingyou Song are listed as first authors in alphabetical ordering of this work. Aleksandra Faust, Frank Hutter, and Marius Lindauer are listed as last authors in alphabetical ordering for their advising throughout this work.

We would like to thank Jie Tan for providing feedback on the survey, as well as Sagi Perel and Daniel Golovin for valuable discussions. Frank, André and Raghu acknowledge Robert Bosch GmbH for financial support.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.. Software available from tensorflow.org.

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., & Bellemare, M. G. (2021). Deep reinforcement learning at the edge of the statistical precipice. *CoRR*, *abs/2108.13264*.

Akinola, I., Angelova, A., Lu, Y., Chebotar, Y., Kalashnikov, D., Varley, J., Ibarz, J., & Ryoo, M. S. (2021). Visionary: Vision architecture discovery for robot learning. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, pp. 10779–10785. IEEE.

Alet, F., Schneider, M. F., Lozano-Pérez, T., & Kaelbling, L. P. (2020). Meta-learning curiosity algorithms. In *8th International Conference on Learning Representations, ICLR, Addis Ababa, Ethiopia, April 26-30*. OpenReview.net.

Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., & Zaremba, W. (2017). Hindsight experience replay. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, December 4-9, Long Beach, CA, USA*, pp. 5048–5058.

Andrychowicz, M., Denil, M., Colmenarejo, S. G., Hoffman, M. W., Pfau, D., Schaul, T., & de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., & Garnett, R. (Eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3981–3989.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., Gelly, S., & Bachem, O. (2021). What matters for on-policy deep actor-critic methods? a large-scale study. In *International Conference on Learning Representations*.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R., & Wang, R. (2019). On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems 32: NeurIPS, December 8-14, Vancouver, BC, Canada*, pp. 8139–8148.

Ashraf, N. M., Mostafa, R. R., Sakr, R. H., & Rashad, M. Z. (2021). Optimizing hyperparameters of deep reinforcement learning for autonomous driving based on whale optimization algorithm. *PLOS ONE*, *16*(6), 1–24.

Awad, N., Mallik, N., & Hutter, F. (2021). Dehb: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*.

Bäck, T. (1998). An overview of parameter control methods by self-adaptation in evolutionary algorithms. *Fundam. Inf.*, *35*(1–4), 51–66.

Bacon, P., Harb, J., & Precup, D. (2017). The option-critic architecture. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, San Francisco, California, USA*, pp. 1726–1734. AAAI Press.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., & Blundell, C. (2020). Agent57: Outperforming the atari human benchmark. In *Proceedings of the 37th International Conference on Machine Learning, ICML, 13-18 July, Virtual Event*, Vol. 119 of *Proceedings of Machine Learning Research*, pp. 507–517. PMLR.

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., & Bakshy, E. (2020). Botorch: A framework for efficient monte-carlo bayesian optimization. In *Advances in Neural Information Processing Systems 33: NeurIPS, December 6-12, virtual*.

Ball, P., Parker-Holder, J., Pacchiano, A., Choromanski, K., & Roberts, S. (2020). Ready policy one: World building through active learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML*.

Bamford, C., Huang, S., & Lucas, S. M. (2020). Griddly: A platform for AI research in games. *CoRR*, *abs/2011.06363*.

Barto, A. G., & Sutton, R. S. (1981). Goal seeking components for adaptive intelligence: An initial assessment.. Tech. rep., Massachusetts Univ Amherst Dept of Computer and Information Science.

Bas-Serrano, J., Curi, S., Krause, A., & Neu, G. (2021). Logistic q-learning. In *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS, April 13-15, Virtual Event*, Vol. 130 of *Proceedings of Machine Learning Research*, pp. 3610–3618. PMLR.

Bechtle, S., Molchanov, A., Chebotar, Y., Grefenstette, E., Righetti, L., Sukhatme, G. S., & Meier, F. (2020). Meta learning via learned loss. In *25th International Conference on Pattern Recognition, ICPR, Virtual Event / Milan, Italy, January 10-15, 2021*, pp. 4161–4168. IEEE.

Bellemare, M., Candido, S., Castro, P., Gong, J., Machado, M., Moitra, S., Ponda, S., & Wang, Z. (2020). Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, *588*, 77–82.

Bellemare, M. G., Dabney, W., & Munos, R. (2017). A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, p. 449–458. JMLR.org.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2012). The Arcade Learning Environment: An Evaluation Platform for General Agents. *CoRR*, *abs/1207.4708*.

Benjamins, C., Eimer, T., Schubert, F., Biedenkapp, A., Rosenhahn, B., Hutter, F., & Lindauer, M. (2021). Carl: A benchmark for contextual and adaptive reinforcement learning. In *NeurIPS 2021 Workshop on Ecological Theory of Reinforcement Learning*.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. In *Journal of Machine Learning Research*.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., & Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. *CoRR*, *abs/1912.06680*.

Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-Dynamic Programming*, Vol. 27.

Biedenkapp, A., Lindauer, M., Eggensperger, K., Fawcett, C., Hoos, H. H., & Hutter, F. (2017). Efficient parameter importance analysis via ablation with surrogates. In *Proceedings of the Conference on Artificial Intelligence (AAAI'17)*, pp. 773–779. AAAI Press.

Biedenkapp, A., Marben, J., Lindauer, M., & Hutter, F. (2018). CAVE: Configuration assessment, visualization and evaluation. In *Proceedings of the International Conference on Learning and Intelligent Optimization (LION)*. Springer.

Biedenkapp, A., Rajan, R., Hutter, F., & Lindauer, M. (2021). TempoRL: Learning when to act. In *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*, Vol. 139 of *Proceedings of Machine Learning Research*, pp. 914–924. PMLR.

Bontrager, P., & Togelius, J. (2020). Fully differentiable procedural content generation through generative playing networks. *CoRR*, *abs/2002.05259*.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs..

Brochu, E., Cora, V. M., & de Freitas, N. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, *abs/1012.2599*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym..

Browne, C., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Liebana, D. P., Samothrakis, S., & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intell. AI Games*, *4*(1), 1–43.

Campero, A., Raileanu, R., Kuttler, H., Tenenbaum, J. B., Rocktäschel, T., & Grefenstette, E. (2021). Learning with {amig}o: Adversarially motivated intrinsic goals. In *International Conference on Learning Representations*.

Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., Botvinick, M., & de Freitas, N. (2017). Learning to learn without gradient descent by gradient descent. In Precup, D., & Teh, Y. W. (Eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, Vol. 70 of *Proceedings of Machine Learning Research*, pp. 748–756. PMLR.

Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D., & de Freitas, N. (2018). Bayesian optimization in AlphaGo. *CoRR*, *abs/1812.06855*.

Chevalier-Boisvert, M., Willems, L., & Pal, S. (2018). Minimalistic gridworld environment for openai gym..

Chiang, H. L., Faust, A., Fiser, M., & Francis, A. G. (2019). Learning navigation behaviors end-to-end with autorl. *IEEE Robotics Autom. Lett.*, *4*(2), 2007–2014.

Cho, H., Kim, Y., Lee, E., Choi, D., Lee, Y., & Rhee, W. (2020). Basic enhancement strategies when using bayesian optimization for hyperparameter tuning of deep neural networks. *IEEE Access*, *8*, 52588–52608.

Chrabąszcz, P., Loshchilov, I., & Hutter, F. (2018). Back to basics: Benchmarking canonical evolution strategies for playing atari. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI*, pp. 1419–1426. International Joint Conferences on Artificial Intelligence Organization.

Chua, K., Calandra, R., McAllister, R., & Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems 31*, pp. 4754–4765.

Clune, J., Beckmann, B. E., Ofria, C., & Pennock, R. T. (2009). Evolving coordinated quadruped gaits with the hyperneat generative encoding. In *Proceedings of the Eleventh Conference on Congress on Evolutionary Computation*, CEC'09, p. 2764–2771. IEEE Press.

Clune, J., Misevic, D., Ofria, C., Lenski, R. E., Elena, S. F., & Sanjuán, R. (2008). Natural selection fails to optimize mutation rates for long-term adaptation on rugged fitness landscapes. *PLOS Computational Biology*, *4*, 1–8.

Co-Reyes, J. D., Miao, Y., Peng, D., Le, Q. V., Levine, S., Lee, H., & Faust, A. (2021). Evolving reinforcement learning algorithms. In *International Conference on Learning Representations*.

Cobbe, K., Hesse, C., Hilton, J., & Schulman, J. (2020). Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML, 13-18 July, Virtual Event*, Vol. 119 of *Proceedings of Machine Learning Research*, pp. 2048–2056. PMLR.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., & Schulman, J. (2019a). Quantifying generalization in reinforcement learning. In Chaudhuri, K., & Salakhutdinov, R. (Eds.), *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97 of *Proceedings of Machine Learning Research*, pp. 1282–1289. PMLR.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., & Schulman, J. (2019b). Quantifying generalization in reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML, 9-15 June, Long Beach, California, USA*, Vol. 97 of *Proceedings of Machine Learning Research*, pp. 1282–1289. PMLR.

Colas, C., Sigaud, O., & Oudeyer, P. (2018). How many random seeds? statistical power analysis in deep reinforcement learning experiments. *CoRR*, *abs/1806.08295*.

Cutler, M., Walsh, T. J., & How, J. P. (2014). Reinforcement learning with multi-fidelity simulators. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3888–3895.

da Silva, F. L., Costa, A. H. R., & Stone, P. (2019). Building self-play curricula online by playing with expert agents in adversarial games. In *8th Brazilian Conference on Intelligent Systems, BRACIS, Salvador, Brazil, October 15-18*, pp. 479–484. IEEE.

Dabney, W., Ostrovski, G., & Barreto, A. (2021). Temporally-extended ϵ-greedy exploration. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Dalibard, V., & Jaderberg, M. (2021). Faster improvement rate population based training..

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de las Casas, D., Donner, C., Fritz, L., Galperti, C., Huber, A., Keeling, J., Tsimpoukelli, M., Kay, J., Merle, A., Moret, J.-M., Noury, S., Pesamosca, F., Pfau, D., Sauter, O., Sommariva, C., Coda, S., Duval, B., Fasoli, A., Kohli, P., Kavukcuoglu, K., Hassabis, D., & Riedmiller, M. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, *602*, 414–419.

Dennis, M., Jaques, N., Vinitsky, E., Bayen, A. M., Russell, S., Critch, A., & Levine, S. (2020). Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems 33: December 6-12, virtual*.

Doerr, B., & Doerr, C. (2020). Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. In *Theory of Evolutionary Computation*, pp. 271–321. Springer.

Downey, C., & Sanner, S. (2010). Temporal difference bayesian model averaging: A bayesian perspective on adapting lambda. In *ICML*, pp. 311–318.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., & Abbeel, P. (2016). Rl$ˆ2$: Fast reinforcement learning via slow reinforcement learning. *CoRR*, *abs/1611.02779*.

Eggensperger, K., Lindauer, M., & Hutter, F. (2019). Pitfalls and best practices in algorithm configuration. *Journal of Artificial Intelligence Research (JAIR)*, *64*, 861–893.

Eimer, T., Benjamins, C., & Lindauer, M. (2021a). Hyperparameters in contextual rl are highly situational. In *NeurIPS 2021 Workshop on Ecological Theory of Reinforcement Learning*.

Eimer, T., Biedenkapp, A., Hutter, F., & Lindauer, M. (2021b). Self-paced context evaluation for contextual reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*, Vol. 139 of *Proceedings of Machine Learning Research*, pp. 2948–2958. PMLR.

Elfwing, S., Uchibe, E., & Doya, K. (2017). Online meta-learning by parallel algorithm competition. *CoRR*, *abs/1702.07490*.

Elfwing, S., Uchibe, E., & Doya, K. (2018). Online meta-learning by parallel algorithm competition. In Aguirre, H. E., & Takadama, K. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, pp. 426–433. ACM.

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *J. Mach. Learn. Res.*, *20*, 55:1–55:21.

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., & Madry, A. (2020). Implementation matters in deep RL: A case study on PPO and TRPO. In *8th*

International Conference on Learning Representations, ICLR, Addis Ababa, Ethiopia, April 26-30. OpenReview.net.

Eriksson, A., Capi, G., & Doya, K. (2003). Evolution of meta-parameters in reinforcement learning algorithm. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (Cat. No.03CH37453)*, Vol. 1, pp. 412–417 vol.1.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., & Kavukcuoglu, K. (2018). IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning, ICML, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Proceedings of Machine Learning Research, pp. 1406–1415. PMLR.

Falkner, S., Klein, A., & Hutter, F. (2018). Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pp. 1437–1446. PMLR.

Fang, K., Zhu, Y., Savarese, S., & Fei-Fei, L. (2021). Adaptive procedural task generation for hard-exploration problems. In *International Conference on Learning Representations*.

Farahani, M. D., & Mozayani, N. (2019). Automatic construction and evaluation of macro-actions in reinforcement learning. *Appl. Soft Comput.*, *82*.

Farquhar, G., Gustafson, L., Lin, Z., Whiteson, S., Usunier, N., & Synnaeve, G. (2020). Growing action spaces. In *Proceedings of the 37th International Conference on Machine Learning, ICML, 13-18 July, Virtual Event*, Vol. 119, pp. 3040–3051. PMLR.

Faust, A., Francis, A., & Mehta, D. (2019). Evolving rewards to automate reinforcement learning. In *AutoML workshop at 7th International Conference on Learning Representation*.

Fawcett, C., & Hoos, H. H. (2016). Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, *22*(4), 431–458.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., & Dabney, W. (2020). Revisiting fundamentals of experience replay. In *ICML*.

Fernandez, F. C., & Caarls, W. (2018). Parameters tuning and optimization for reinforcement learning algorithms using evolutionary computing. In *2018 International Conference on Information Systems and Computer Science (INCISCOS)*, pp. 301–305.

Ferreira, F., Nierhoff, T., & Hutter, F. (2021). Learning synthetic environments for reinforcement learning with evolution strategies..

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML, Sydney, NSW, Australia, 6-11 August*, Vol. 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135. PMLR.

Flennerhag, S., Schroecker, Y., Zahavy, T., van Hasselt, H., Silver, D., & Singh, S. (2021). Bootstrapped meta-learning. In *arxiv*.

Florensa, C., Held, D., Geng, X., & Abbeel, P. (2018). Automatic goal generation for reinforcement learning agents. In *Proceedings of the 35th International Conference on*

Machine Learning, ICML, Stockholmsmässan, Stockholm, Sweden, July 10-15, Vol. 80 of *Proceedings of Machine Learning Research*, pp. 1514–1523. PMLR.

Foerster, J. N. (2018). *Deep multi-agent reinforcement learning.* Ph.D. thesis, University of Oxford.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., & Legg, S. (2018). Noisy networks for exploration. In *International Conference on Learning Representations*.

Franke, J. K., Koehler, G., Biedenkapp, A., & Hutter, F. (2021). Sample-efficient automated deep reinforcement learning. In *International Conference on Learning Representations*.

Frazier, P. I., & Wang, J. (2015). Bayesian optimization for materials design. *Springer Series in Materials Science*, *225*, 45–75.

Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., & Bachem, O. (2021). Brax - a differentiable physics engine for large scale rigid body simulation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Fukunaga, A. S. (1998). Restart scheduling for genetic algorithms. In *Parallel Problem Solving from Nature — PPSN V*, pp. 357–366, Berlin, Heidelberg. Springer Berlin Heidelberg.

Gaier, A., & Ha, D. (2019). Weight agnostic neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS, December 8-14, Vancouver, BC, Canada*, pp. 5365–5379.

Garcia, C. E., Prett, D. M., & Morari, M. (1989). Model predictive control: Theory and practice - A survey. *Autom.*, *25*(3), 335–348.

Gleave, A., Dennis, M. D., Legg, S., Russell, S., & Leike, J. (2021). Quantifying differences in reward functions. In *International Conference on Learning Representations*.

Gloger, B. (2004). Self-adaptive evolutionary algorithms..

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., & Sculley, D. (2017). Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17*, pp. 1487–1495. ACM.

Gomez, F., Schmidhuber, J., & Miikkulainen, R. (2006). Efficient non-linear control through neuroevolution. In *Machine Learning: ECML*, pp. 654–662, Berlin, Heidelberg. Springer Berlin Heidelberg.

Griffiths, R.-R., & Hernández-Lobato, J. M. (2020). Constrained bayesian optimization for automatic chemical design using variational autoencoders. *Chemical science*, *11*(2), 577–586.

Guo, S. (2020). An introduction to surrogate optimization: Intuition, illustration, case study, and the code..

Gur, I., Jaques, N., Miao, Y., Choi, J., Tiwari, M., Lee, H., & Faust, A. (2021). Environment generation for zero-shot compositional reinforcement learning. In *Advances in Neural Information Processing Systems*.

Guss, W. H., Codel, C., Hofmann, K., Houghton, B., Kuno, N., Milani, S., Mohanty, S., Liebana, D. P., Salakhutdinov, R., Topin, N., et al. (2019). The MineRL competition on sample efficient reinforcement learning using human priors..

Ha, D., & Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NeurIPS'18, pp. 2455–2467.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic algorithms and applications. *CoRR*, *abs/1812.05905*.

Hafner, D., Lillicrap, T., Ba, J., & Norouzi, M. (2020). Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., & Davidson, J. (2019). Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 2555–2565.

Hausknecht, M., Lehman, J., Miikkulainen, R., & Stone, P. (2014). A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, *6*(4), 355–366.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Las Vegas, NV, USA, June 27-30*, pp. 770–778. IEEE Computer Society.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep reinforcement learning that matters. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI)*, pp. 3207–3214. AAAI Press.

Hernández-Lobato, J. M., Requeima, J., Pyzer-Knapp, E. O., & Aspuru-Guzik, A. (2017). Parallel and distributed thompson sampling for large-scale accelerated exploration of chemical space. In *International conference on machine learning*, pp. 1470–1479. PMLR.

Hertel, L., Baldi, P., & Gillen, D. L. (2020). Quantity vs. quality: On hyperparameter optimization for deep reinforcement learning. *CoRR*, *abs/2007.14604*.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., & Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning..

Hessel, M., van Hasselt, H., Modayil, J., & Silver, D. (2019). On inductive biases in deep reinforcement learning..

Houthooft, R., Chen, Y., Isola, P., Stadie, B. C., Wolski, F., Ho, J., & Abbeel, P. (2018). Evolved policy gradients. In *Advances in Neural Information Processing Systems 31: NeurIPS, December 3-8, Montréal, Canada*, pp. 5405–5414.

Hu, Y., Wang, W., Jia, H., Wang, Y., Chen, Y., Hao, J., Wu, F., & Fan, C. (2020). Learning to utilize shaping rewards: A new approach of reward shaping. In *Advances in Neural Information Processing Systems 33: NeurIPS, December 6-12, virtual*.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31st International Conference on Machine Learning, (ICML'14)*, pp. 754–762. Omnipress.

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009). Paramils: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*, 267–306.

Hutter, F., Kotthoff, L., & Vanschoren, J. (Eds.). (2019). *Automated Machine Learning - Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer.

Igl, M., Farquhar, G., Luketina, J., Boehmer, W., & Whiteson, S. (2021). Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML, Lille, France, 6-11 July*, Vol. 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456. JMLR.org.

Islam, R., Henderson, P., Gomrokchi, M., & Precup, D. (2017). Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *CoRR*, *abs/1708.04133*.

Jacot, A., Hongler, C., & Gabriel, F. (2018). Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS, December 3-8, Montréal, Canada*, pp. 8580–8589.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castañeda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K., & Graepel, T. (2019). Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, *364*(6443), 859–865.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., & Kavukcuoglu, K. (2017). Population based training of neural networks. *CoRR*, *abs/1711.09846*.

Jamieson, K. G., & Talwalkar, A. (2016). Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS, Cadiz, Spain, May 9-11*, Vol. 51 of *JMLR Workshop and Conference Proceedings*, pp. 240–248. JMLR.org.

Janner, M., Fu, J., Zhang, M., & Levine, S. (2019). When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*.

Jiang, L., Meng, D., Zhao, Q., Shan, S., & Hauptmann, A. G. (2015). Self-paced curriculum learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, Austin, Texas, USA*, pp. 2694–2700. AAAI Press.

Jiang, M., Dennis, M., Parker-Holder, J., Foerster, J., Grefenstette, E., & Rocktäschel, T. (2021a). Replay-guided adversarial environment design. In *Advances in Neural Information Processing Systems*.

Jiang, M., Grefenstette, E., & Rocktäschel, T. (2021b). Prioritized level replay. In *The International Conference on Machine Learning*.

Jin, Y. (Ed.). (2006). *Multi-Objective Machine Learning*, Vol. 16 of *Studies in Computational Intelligence*. Springer.

Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, *13*(4), 455–492.

Jordan, S., Chandak, Y., Cohen, D., Zhang, M., & Thomas, P. (2020). Evaluating the performance of reinforcement learning algorithms. In *Proceedings of the 37th International Conference on Machine Learning*, Vol. 119 of *Proceedings of Machine Learning Research*, pp. 4962–4973. PMLR.

Kadioglu, S., Malitsky, Y., Sellmann, M., & Tierney, K. (2010). ISAC - instance-specific algorithm configuration. In *Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI'10)*, pp. 751–756. IOS Press.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., & Michalewski, H. (2020). Model based reinforcement learning for Atari. In *International Conference on Learning Representations*.

Kandasamy, K., Dasarathy, G., Oliva, J. B., Schneider, J., & Poczos, B. (2016). Gaussian process bandit optimisation with multi-fidelity evaluations. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., & Garnett, R. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 29. Curran Associates, Inc.

Kearns, M. J., & Singh, S. P. (2000). Bias-variance error bounds for temporal difference updates. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, COLT '00, p. 142–147, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Kingma, D. P., & Welling, M. (2014). Auto-encoding variational bayes. In Bengio, Y., & LeCun, Y. (Eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Kirk, R., Zhang, A., Grefenstette, E., & Rocktäschel, T. (2021). A survey of generalisation in deep reinforcement learning. *CoRR*, *abs/2111.09794*.

Kirsch, L., Flennerhag, S., van Hasselt, H., Friesen, A. L., Oh, J., & Chen, Y. (2021). Introducing symmetries to black box meta reinforcement learning. *CoRR*, *abs/2109.10781*.

Kirsch, L., van Steenkiste, S., & Schmidhuber, J. (2020). Improving generalization in meta reinforcement learning using learned objectives. In *8th International Conference on Learning Representations, ICLR, Addis Ababa, Ethiopia, April 26-30*. OpenReview.net.

Klein, A., Falkner, S., Bartels, S., Hennig, P., & Hutter, F. (2017). Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Proceedings of the AISTATS conference*.

Klink, P., D'Eramo, C., Peters, J., & Pajarinen, J. (2020). Self-paced deep reinforcement learning. In *Advances in Neural Information Processing Systems 33: NeurIPS, December 6-12, virtual*.

Konidaris, G. D., & Barto, A. G. (2006). Autonomous shaping: knowledge transfer in reinforcement learning. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML), Pittsburgh, Pennsylvania, USA, June 25-29*, Vol. 148 of *ACM International Conference Proceeding Series*, pp. 489–496. ACM.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25:*, pp. 1106–1114.

Kumar, A., Agarwal, R., Ghosh, D., & Levine, S. (2020a). Implicit under-parameterization inhibits data-efficient deep reinforcement learning..

Kumar, A., Zhou, A., Tucker, G., & Levine, S. (2020b). Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems 33: NeurIPS, December 6-12, virtual*.

Küttler, H., Nardelli, N., Miller, A. H., Raileanu, R., Selvatici, M., Grefenstette, E., & Rocktäschel, T. (2020). The NetHack Learning Environment. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.

Laroche, R., & Féraud, R. (2018a). Reinforcement learning algorithm selection. In *6th International Conference on Learning Representations, ICLR, Vancouver, BC, Canada, April 30 - May 3, Conference Track Proceedings*. OpenReview.net.

Laroche, R., & Féraud, R. (2018b). Reinforcement learning algorithm selection. In *6th International Conference on Learning Representations, ICLR, Vancouver, BC, Canada, April 30 - May 3, Conference Track Proceedings*. OpenReview.net.

Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., & Hutter, M. (2020). Learning quadrupedal locomotion over challenging terrain. *Sci. Robotics*, *5*(47), 5986.

Lee, S., Yosinski, J., Glette, K., Lipson, H., & Clune, J. (2013). Evolving gaits for physical robots with the hyperneat generative encoding: The benefits of simulation.. pp. 540–549.

Lehman, J., Chen, J., Clune, J., & Stanley, K. O. (2018). Es is more than just a traditional finite-difference approximator..

Li, C., Santu, R., Gupta, S., Nguyen, V., Venkatesh, S., Sutti, A., Leal, D. R. D. C., Slezak, T., Height, M., Mohammed, M., et al. (2018). Accelerating experimental design by incorporating experimenter hunches. In *18th IEEE International Conference on Data Mining, ICDM*, pp. 257–266. IEEE.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. In *Journal Machine Learning Research*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR, San Juan, Puerto Rico, May 2-4, Conference Track Proceedings*.

Liu, H., Simonyan, K., & Yang, Y. (2019a). DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR, New Orleans, LA, USA, May 6-9*. OpenReview.net.

Liu, S., Lever, G., Merel, J., Tunyasuvunakool, S., Heess, N., & Graepel, T. (2019b). Emergent coordination through competition. In *7th International Conference on Learning Representations, ICLR, New Orleans, LA, USA, May 6-9*. OpenReview.net.

Liu, S., Lever, G., Wang, Z., Merel, J., Eslami, S. M. A., Hennes, D., Czarnecki, W. M., Tassa, Y., Omidshafiei, S., Abdolmaleki, A., Siegel, N. Y., Hasenclever, L., Marris, L., Tunyasuvunakool, S., Song, H. F., Wulfmeier, M., Muller, P., Haarnoja, T., Tracey, B. D., Tuyls, K., Graepel, T., & Heess, N. (2021a). From motor control to team play in simulated humanoid football. *CoRR*, *abs/2105.12196*.

Liu, Z., Li, X., Kang, B., & Darrell, T. (2021b). Regularization matters in policy optimization - an empirical study on continuous control. In *International Conference on Learning Representations*.

Lu, C., Ball, P. J., Parker-Holder, J., Osborne, M., & Roberts, S. (2021). Revisiting design choices in offline model based reinforcement learning. In *RL for Real Life Workshop at ICML*.

Mankowitz, D. J., Mann, T. A., Bacon, P., Precup, D., & Mannor, S. (2018). Learning robust options. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI)*, pp. 6409–6416. AAAI Press.

Marthi, B. (2007). Automatic shaping and decomposition of reward functions.. ICML '07, p. 601–608, New York, NY, USA. Association for Computing Machinery.

Matiisen, T., Oliver, A., Cohen, T., & Schulman, J. (2020). Teacher-student curriculum learning. *IEEE Trans. Neural Networks Learn. Syst.*, *31*(9), 3732–3740.

Miao, Y., Song, X., Peng, D., Yue, S., Brevdo, E., & Faust, A. (2021). RL-DARTS: differentiable architecture search for reinforcement learning. *CoRR*, *abs/2106.02229*.

Michalewicz, Z. (2013). *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media.

Mirjalili, S., & Lewis, A. (2016). The whale optimization algorithm. *Advances in Engineering Software*, *95*, 51–67.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In Balcan, M., & Weinberger, K. Q. (Eds.), *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, Vol. 48 of *JMLR Workshop and Conference Proceedings*, pp. 1928–1937. JMLR.org.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–33.

Mockus, J. (1974). On bayesian methods for seeking the extremum. In *Optimization Techniques, IFIP Technical Conference, Novosibirsk, USSR, July 1-7*, Vol. 27 of *Lecture Notes in Computer Science*, pp. 400–404. Springer.

Moffaert, K. V., & Nowé, A. (2014). Multi-objective reinforcement learning using sets of pareto dominating policies. *J. Mach. Learn. Res.*, *15*(1), 3483–3512.

Moskovitz, T., Parker-Holder, J., Pacchiano, A., & Arbel, M. (2021). Deep reinforcement learning with dynamic optimism. In *Advances in Neural Information Processing Systems*.

Moulines, E., & Bach, F. (2011). Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, Vol. 24. Curran Associates, Inc.

Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., & Stone, P. (2020). Curriculum learning for reinforcement learning domains: A framework and survey. *J. Mach. Learn. Res.*, *21*, 181:1–181:50.

Neyshabur, B. (2017). Implicit regularization in deep learning. *CoRR*, *abs/1709.01953*.

Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y., & Srebro, N. (2019). The role of over-parametrization in generalization of neural networks. In *7th International Conference on Learning Representations, ICLR, New Orleans, LA, USA, May 6-9*. OpenReview.net.

Neyshabur, B., Tomioka, R., & Srebro, N. (2015). In search of the real inductive bias: On the role of implicit regularization in deep learning. In *3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA, May 7-9, Workshop Track Proceedings*.

Ng, A. Y., Harada, D., & Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML), Bled, Slovenia, June 27 - 30*, pp. 278–287. Morgan Kaufmann.

Nguyen, V., Orbell, S., Lennon, D. T., Moon, H., Vigneau, F., Camenzind, L. C., Yu, L., Zumbühl, D. M., Briggs, G. A. D., Osborne, M. A., et al. (2021). Deep reinforcement learning for efficient measurement of quantum devices. *npj Quantum Information*, *7*(1), 1–9.

Nguyen, V., & Osborne, M. A. (2020). Knowing the what but not the where in bayesian optimization. In *Proceedings of the 37th International Conference on Machine Learning, ICML, 13-18 July, Virtual Event*, Vol. 119 of *Proceedings of Machine Learning Research*, pp. 7317–7326. PMLR.

Nguyen, V., Schulze, S., & Osborne, M. A. (2020). Bayesian optimization for iterative learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Obando-Ceron, J. S., & Castro, P. S. (2020). Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *Deep Reinforcement Learning Workshop, NeurIPS*.

Oh, J., Hessel, M., Czarnecki, W. M., Xu, Z., van Hasselt, H., Singh, S., & Silver, D. (2020). Discovering reinforcement learning algorithms. In *Advances in Neural Information Processing Systems 33*.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., & Zhang, L. (2019). Solving rubik's cube with a robot hand. *CoRR*, *abs/1910.07113*.

OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J. W., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., & Zaremba, W. (2018). Learning dexterous in-hand manipulation. *CoRR*, *abs/1808.00177*.

OpenAI, O., Plappert, M., Sampedro, R., Xu, T., Akkaya, I., Kosaraju, V., Welinder, P., D'Sa, R., Petron, A., de Oliveira Pinto, H. P., Paino, A., Noh, H., Weng, L., Yuan, Q., Chu, C., & Zaremba, W. (2021). Asymmetric self-play for automatic goal discovery in robotic manipulation..

Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvári, C., Singh, S., Van Roy, B., Sutton, R., Silver, D., & van Hasselt, H. (2020). Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*.

Parisotto, E., Song, H. F., Rae, J. W., Pascanu, R., Gülçehre, Ç., Jayakumar, S. M., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., Botvinick, M., Heess, N., & Hadsell, R. (2020). Stabilizing transformers for reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML, 13-18 July, Virtual Event*, Vol. 119 of *Proceedings of Machine Learning Research*, pp. 7487–7498. PMLR.

Parker-Holder, J., Nguyen, V., Desai, S., & Roberts, S. (2021). Tuning Mixed Input Hyperparameters on the Fly for Efficient Population Based AutoRL. In *Advances in Neural Information Processing Systems*, Vol. 34.

Parker-Holder, J., Nguyen, V., & Roberts, S. J. (2020a). Provably efficient online hyperparameter optimization with population-based bandits. *Advances in Neural Information Processing Systems*, *33*.

Parker-Holder, J., Pacchiano, A., Choromanski, K., & Roberts, S. (2020b). Effective diversity in population-based reinforcement learning. In *Advances in Neural Information Processing Systems 33*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., & Garnett, R. (Eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc.

Patterson, A., Neumann, S., White, A. M., Kumaraswamy, R., & White, M. (2021). The cross-environment hyperparameter setting benchmark for reinforcement learning..

Paul, S., Kurin, V., & Whiteson, S. (2019). Fast efficient hyperparameter tuning for policy gradients..

Pérez-Cruz, F., Vaerenbergh, S. V., Murillo-Fuentes, J. J., Lázaro-Gredilla, M., & Santamaría, I. (2013). Gaussian processes for nonlinear signal processing: An overview of recent advances. *IEEE Signal Process. Mag.*, *30*(4), 40–50.

Perrone, V., Shen, H., Zolic, A., Shcherbatyi, I., Ahmed, A., Bansal, T., Donini, M., Winkelmolen, F., Jenatton, R., Faddoul, J. B., Pogorzelska, B., Miladinovic, M., Kenthapadi, K., Seeger, M. W., & Archambeau, C. (2021). Amazon sagemaker automatic model tuning: Scalable gradient-free optimization. In Zhu, F., Ooi, B. C., & Miao, C. (Eds.), *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pp. 3463–3471. ACM.

Precup, D., Sutton, R. S., & Singh, S. P. (2000). Eligibility traces for off-policy policy evaluation. In Langley, P. (Ed.), *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pp. 759–766. Morgan Kaufmann.

Prokhorov, D., & Wunsch, D. (1997). Adaptive critic designs. *IEEE Transactions on Neural Networks*, *8*(5), 997–1007.

Raileanu, R., Goldstein, M., Yarats, D., Kostrikov, I., & Fergus, R. (2020). Automatic data augmentation for generalization in deep reinforcement learning. *CoRR*, *abs/2006.12862*.

Rajan, R., & Hutter, F. (2019). Mdp playground: Meta-features in reinforcement learning. In *NeurIPS Deep RL Workshop*.

Rakelly, K., Zhou, A., Finn, C., Levine, S., & Quillen, D. (2019). Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *Proceedings of the 36th International Conference on Machine Learning, ICML, 9-15 June, Long Beach, California, USA*, Vol. 97 of *Proceedings of Machine Learning Research*, pp. 5331–5340. PMLR.

Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*, pp. 4780–4789. AAAI Press.

Rechenberg, I. (1973). Evolutionsstrategie : Optimierung technischer systeme nach prinzipien der biologischen evolution..

Rice, J. R. (1976). The algorithm selection problem. In *Advances in computers*, Vol. 15, pp. 65–118. Elsevier.

Riquelme, C., Penedones, H., Vincent, D., Maennel, H., Gelly, S., Mann, T. A., Barreto, A., & Neu, G. (2019). Adaptive temporal-difference learning for policy evaluation with per-state uncertainty estimates. In *Advances in Neural Information Processing Systems*, Vol. 32.

Risi, S., & Stanley, K. O. (2013). Confronting the challenge of learning a flexible neural controller for a diversity of morphologies. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, p. 255–262, New York, NY, USA. Association for Computing Machinery.

Romac, C., Portelas, R., Hofmann, K., & Oudeyer, P. (2021). Teachmyagent: a benchmark for automatic curriculum learning in deep RL. In *Proceedings of the 38th Interna-*

tional Conference on Machine Learning, ICML, 18-24 July, Virtual Event, Vol. 139 of *Proceedings of Machine Learning Research*, pp. 9052–9063. PMLR.

Rowland, M., Dabney, W., & Munos, R. (2020). Adaptive trade-offs in off-policy learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. PMLR.

Runge, F., Stoll, D., Falkner, S., & Hutter, F. (2019). Learning to design RNA. In *7th International Conference on Learning Representations, ICLR, New Orleans, LA, USA, May 6-9*. OpenReview.net.

Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Kuttler, H., Grefenstette, E., & Rocktäschel, T. (2021). Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. In *Advances in Neural Information Processing Systems 31: NeurIPS, December 3-8, Montréal, Canada*, pp. 2488–2498.

Schaul, T., Borsa, D., Ding, D., Szepesvari, D., Ostrovski, G., Dabney, W., & Osindero, S. (2019). Adapting behaviour for learning progress. *CoRR*, *abs/1912.06910*.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. In *4th International Conference on Learning Representations, ICLR, San Juan, Puerto Rico, May 2-4, Conference Track Proceedings*.

Schmitt, S., Hudson, J. J., Zídek, A., Osindero, S., Doersch, C., Czarnecki, W. M., Leibo, J. Z., Küttler, H., Zisserman, A., Simonyan, K., & Eslami, S. M. A. (2018). Kickstarting deep reinforcement learning. *CoRR*, *abs/1803.03835*.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, *588*(7839), 604–609.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., & Moritz, P. (2015). Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning, ICML, Lille, France, 6-11 July*, Vol. 37 of *JMLR Workshop and Conference Proceedings*, pp. 1889–1897. JMLR.org.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017a). Proximal policy optimization algorithms..

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017b). Proximal policy optimization algorithms. *CoRR*, *abs/1707.06347*.

Sehgal, A., La, H. M., Louis, S. J., & Nguyen, H. (2019). Deep reinforcement learning using genetic algorithm for parameter optimization. *CoRR*, *abs/1905.04100*.

Sharma, S., Lakshminarayanan, A. S., & Ravindran, B. (2017). Learning to repeat: Fine grained action repetition for deep reinforcement learning. In *5th International Conference on Learning Representations, ICLR, Toulon, France, April 24-26, Conference Track Proceedings*. OpenReview.net.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*, 484–489.

Singh, S., & Dayan, P. (1996). Analytical mean squared error curves in temporal difference learning. In *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96, p. 1054–1060, Cambridge, MA, USA. MIT Press.

Sinha, S., Bharadhwaj, H., Srinivas, A., & Garg, A. (2020). D2RL: deep dense architectures in reinforcement learning. In *Deep Reinforcement Learning Workshop, NeurIPS*, Vol. abs/2010.09163.

Snel, M., & Whiteson, S. (2010). Multi-task evolutionary shaping without pre-specified representations. In *Genetic and Evolutionary Computation Conference, GECCO, Proceedings, Portland, Oregon, USA, July 7-11*, pp. 1031–1038. ACM.

Song, J., Chen, Y., & Yue, Y. (2019). A general framework for multi-fidelity bayesian optimization with gaussian processes. In Chaudhuri, K., & Sugiyama, M. (Eds.), *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, Vol. 89 of *Proceedings of Machine Learning Research*, pp. 3158–3167. PMLR.

Song, X., Choromanski, K., Parker-Holder, J., Tang, Y., Peng, D., Jain, D., Gao, W., Pacchiano, A., Sarlós, T., & Yang, Y. (2021). ES-ENAS: combining evolution strategies with neural architecture search at no extra cost for reinforcement learning. *CoRR*, *abs/2101.07415*.

Song, X., Du, Y., & Jackson, J. (2019). An empirical study on hyperparameters and their interdependence for RL generalization..

Song, X., Jiang, Y., Tu, S., Du, Y., & Neyshabur, B. (2020). Observational overfitting in reinforcement learning. In *8th International Conference on Learning Representations, ICLR, Addis Ababa, Ethiopia, April 26-30*. OpenReview.net.

Spears, W. M. (1995). Adapting crossover in evolutionary algorithms. In McDonnell, J. R., Reynolds, R. G., & Fogel, D. B. (Eds.), *Proceedings of the Fourth Annual Conference on Evolutionary Programming, EP 1995, San Diego, CA, USA, March 1-3, 1995*, pp. 367–384. A Bradford Book, MIT Press. Cambridge, Massachusetts.

Srinivas, N., Krause, A., Kakade, S. M., & Seeger, M. W. (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, Haifa, Israel*, pp. 1015–1022. Omnipress.

Stanley, K., Clune, J., Lehman, J., & Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, *1*.

Stanley, K. O., D'Ambrosio, D. B., & Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, *15*(2), 185–212.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies.. *10*(2).

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., & Clune, J. (2018). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning..

Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., & Fergus, R. (2018). Intrinsic motivation and automatic curricula via asymmetric self-play. In *6th International Conference on Learning Representations, ICLR, Vancouver, BC, Canada, April 30 - May 3, Conference Track Proceedings*. OpenReview.net.

Sutton, R., & Singh, S. P. (1994). On step-size and bias in temporal-difference learning. In *Center for Systems Science, Yale University*, pp. 91–96.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, *2*(4), 160–163.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (Second edition). The MIT Press.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999a). Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, p. 1057–1063, Cambridge, MA, USA. MIT Press.

Sutton, R. S., Precup, D., & Singh, S. P. (1999b). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, *112*(1-2), 181–211.

Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., & Vanhoucke, V. (2018). Sim-to-real: Learning agile locomotion for quadruped robots. In Kress-Gazit, H., Srinivasa, S. S., Howard, T., & Atanasov, N. (Eds.), *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*.

Tang, Y., Nguyen, D., & Ha, D. (2020). Neuroevolution of self-interpretable agents. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

Tassa, Y., Erez, T., & Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pp. 4906–4913. IEEE.

Team, O. E. L., Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., McAleese, N., Bradley-Schmieg, N., Wong, N., Porcel, N., Raileanu, R., Hughes-Fitt, S., Dalibard, V., & Czarnecki, W. M. (2021). Open-ended learning leads to generally capable agents. *CoRR*, *abs/2107.12808*.

Todorov, E., Erez, T., & Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS'12)*, pp. 5026–5033. IEEE.

Tran, M., Nguyen, V., Bruce, R., Crockett, D., Formenti, F., Phan, P., Payne, S., & Farmery, A. (2021). Simulation-based optimisation to quantify heterogeneity of specific ventilation and perfusion in the lung by the inspired sinewave test. *Scientific reports*, *11*(1), 1–10.

Turchetta, M., Kolobov, A., Shah, S., Krause, A., & Agarwal, A. (2020). Safe reinforcement learning via curriculum induction. In *Advances in Neural Information Processing Systems 33: NeurIPS, December 6-12, virtual*.

van Bueren, N., Reed, T., Nguyen, V., Sheffield, J., van der Ven, S., Osborne, M., Kroesbergen, E., & Kadosh, R. C. (2021). Personalized closed-loop brain stimulation for effective neurointervention across participants. *PLoS Computational Biology*, *17*.

van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, Phoenix, Arizona, USA*, pp. 2094–2100. AAAI Press.

van Rijn, J. N., & Hutter, F. (2018). Hyperparameter importance across datasets. In Guo, Y., & Farooq, F. (Eds.), *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pp. 2367–2376. ACM.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*.

Veeriah, V., Hessel, M., Xu, Z., Rajendran, J., Lewis, R. L., Oh, J., van Hasselt, H. P., Silver, D., & Singh, S. (2019). Discovery of useful questions as auxiliary tasks. In *Advances in Neural Information Processing Systems*, Vol. 32.

Veeriah, V., Zahavy, T., Hessel, M., Xu, Z., Oh, J., Kemaev, I., van Hasselt, H., Silver, D., & Singh, S. (2021). Discovery of options via meta-learned subgoals. *CoRR*, *abs/2102.06741*.

Vieillard, N., Pietquin, O., & Geist, M. (2020). Munchausen reinforcement learning. In *Advances in Neural Information Processing Systems 33: NeurIPS, December 6-12, virtual*.

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

Volz, V., Schrum, J., Liu, J., Lucas, S. M., Smith, A. M., & Risi, S. (2018). Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, Kyoto, Japan, July 15-19*, pp. 221–228. ACM.

Wan, X., Nguyen, V., Ha, H., Ru, B., Lu, C., & Osborne, M. A. (2021). Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces. In Meila, M., & Zhang, T. (Eds.), *Proceedings of the 38th International Conference on Machine Learning*, Vol. 139 of *Proceedings of Machine Learning Research*, pp. 10663–10674. PMLR.

Wang, J. X., King, M., Porcel, N., Kurth-Nelson, Z., Zhu, T., Deck, C., Choy, P., Cassin, M., Reynolds, M., Song, H. F., Buttimore, G., Reichert, D. P., Rabinowitz, N. C., Matthey, L., Hassabis, D., Lerchner, A., & Botvinick, M. (2021). Alchemy: A structured task distribution for meta-reinforcement learning. *CoRR*, *abs/2102.02926*.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., & Botvinick, M. (2016). Learning to reinforcement learn. *CoRR*, *abs/1611.05763*.

Wang, R., Lehman, J., Clune, J., & Stanley, K. O. (2019). POET: open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, Prague, Czech Republic, July 13-17*, pp. 142–151. ACM.

Wang, R., Lehman, J., Rawal, A., Zhi, J., Li, Y., Clune, J., & Stanley, K. O. (2020). Enhanced POET: open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *Proceedings of the 37th International Conference on Machine Learning, ICML, 13-18 July, Virtual Event*, Vol. 119 of *Proceedings of Machine Learning Research*, pp. 9940–9951. PMLR.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML, New York City, NY, USA, June 19-24*, Vol. 48 of *JMLR Workshop and Conference Proceedings*, pp. 1995–2003. JMLR.org.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), 279–292.

White, M., & White, A. (2016). A greedy approach to adapting the trace parameter for temporal difference learning. In *AAMAS*.

Whiteson, S., Tanner, B., Taylor, M. E., & Stone, P. (2009). Generalized domains for empirical evaluations in reinforcement learning..

Whitley, D., Gordon, V. S., & Mathias, K. (1994). Lamarckian evolution, the baldwin effect and function optimization. In *Parallel Problem Solving from Nature — PPSN III*.

Wöhlke, J., Schmitt, F., & van Hoof, H. (2020). A performance-based start state curriculum framework for reinforcement learning. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, pp. 1503–1511. International Foundation for Autonomous Agents and Multiagent Systems.

Xu, L., Hoos, H., & Leyton-Brown, K. (2010). Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI'10)*, pp. 210–216. AAAI Press.

Xu, Z., van Hasselt, H., & Silver, D. (2018). Meta-gradient reinforcement learning. In *Advances in Neural Information Processing Systems 31: NeurIPS, December 3-8, Montréal, Canada*, pp. 2402–2413.

Xu, Z., van Hasselt, H. P., Hessel, M., Oh, J., Singh, S., & Silver, D. (2020). Meta-gradient reinforcement learning with an objective discovered online. In *Advances in Neural Information Processing Systems*, Vol. 33, pp. 15254–15264.

Yang, R., Sun, X., & Narasimhan, K. (2019). A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., & Garnett, R. (Eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 14610–14621.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., & Levine, S. (2019). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*.

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

Zambaldi, V. F., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D. P., Lillicrap, T. P., Lockhart, E., Shanahan, M., Langston, V., Pascanu, R., Botvinick, M., Vinyals, O., & Battaglia, P. W. (2019). Deep reinforcement learning with relational inductive biases. In *7th International Conference on Learning Representations, ICLR, New Orleans, LA, USA, May 6-9*. OpenReview.net.

Zhang, B., Rajan, R., Pineda, L., Lambert, N. O., Biedenkapp, A., Chua, K., Hutter, F., & Calandra, R. (2021). On the importance of hyperparameter optimization for model-based reinforcement learning. In *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS, April 13-15, Virtual Event*, Vol. 130 of *Proceedings of Machine Learning Research*, pp. 4015–4023. PMLR.

Zhang, C., Vinyals, O., Munos, R., & Bengio, S. (2018). A study on overfitting in deep reinforcement learning. *CoRR, abs/1804.06893*.

Zhang, Y., Abbeel, P., & Pinto, L. (2020). Automatic curriculum learning through value disagreement. In *Advances in Neural Information Processing Systems 33: NeurIPS, December 6-12, virtual*.

Zheng, Z., Oh, J., & Singh, S. (2018). On learning intrinsic rewards for policy gradient methods. In *Advances in Neural Information Processing Systems 31: NeurIPS, December 3-8, Montréal, Canada*, pp. 4649–4659.

Zintgraf, L. M., Shiarlis, K., Igl, M., Schulze, S., Gal, Y., Hofmann, K., & Whiteson, S. (2020). Varibad: A very good method for bayes-adaptive deep RL via meta-learning. In *8th International Conference on Learning Representations, ICLR, Addis Ababa, Ethiopia, April 26-30*. OpenReview.net.

Zou, H., Ren, T., Yan, D., Su, H., & Zhu, J. (2019). Reward shaping via meta-learning. *CoRR, abs/1901.09330*.