Neural Networks 113 (2019) 28–40

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Review

Policy search in continuous action domains: An overview

Olivier Sigaud [a,*], Freek Stulp [b]

[a] INRIA Bordeaux Sud-Ouest, équipe FLOWERS Sorbonne Université, CNRS UMR 7222, Institut des Systèmes Intelligents et de Robotique, F-75005 Paris, France
[b] German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Wessling, Germany

ARTICLE INFO

Article history:
Received 22 June 2018
Received in revised form 3 December 2018
Accepted 22 January 2019
Available online 5 February 2019

Keywords:
Policy search
Sample efficiency
Deep reinforcement learning
Deep neuroevolution

ABSTRACT

Continuous action policy search is currently the focus of intensive research, driven both by the recent success of deep reinforcement learning algorithms and the emergence of competitors based on evolutionary algorithms. In this paper, we present a broad survey of policy search methods, providing a unified perspective on very different approaches, including also Bayesian Optimization and directed exploration methods. The main message of this overview is in the relationship between the families of methods, but we also outline some factors underlying sample efficiency properties of the various approaches.

Contents

1. Introduction ... 29
   1.1. Scope and contributions ... 29
   1.2. Perspective and structure of the survey ... 29
2. Policy search without a utility model ... 30
   2.1. Truly random search ... 30
   2.2. Population-based optimization ... 30
   2.3. Evolutionary strategies ... 30
   2.4. Estimation of distribution algorithms ... 31
   2.5. Finite difference methods ... 31
   2.6. Reference to the main algorithms ... 32
   2.7. Sample efficiency analysis ... 32
3. Policy search with a model of utility in the space of policy parameters ... 32
   3.1. Bayesian optimization ... 32
   3.2. Reference to the main algorithms ... 32
   3.3. Sample efficiency analysis ... 33
4. Directed exploration methods ... 33
   4.1. Reference to the main algorithms ... 34
   4.2. Sample efficiency analysis ... 34
5. Policy search with a critic ... 34
   5.1. Exploration in parameter or state–action space ... 34
   5.2. Transient critic algorithms ... 34
   5.3. Persistent critic algorithms ... 35
   5.4. Key properties of persistent critic algorithms ... 35
      5.4.1. Accuracy and scalability: deep neural networks ... 35
      5.4.2. Stability: the target critic ... 35
      5.4.3. Sample reuse: the replay buffer ... 35
      5.4.4. Adaptive step sizes and return lengths ... 35
   5.5. Overview of deep RL algorithms ... 35

* Corresponding author.
E-mail addresses: olivier.sigaud@isir.upmc.fr (O. Sigaud), freek.stulp@dlr.de (F. Stulp).

https://doi.org/10.1016/j.neunet.2019.01.011
0893-6080/

5.6. Reference to the main algorithms ... 35
5.7. Sample efficiency analysis ... 35
6. Discussion ... 36
6.1. Building a model or not ... 36
6.2. Building a utility function model in the policy parameter space versus the state–action space ... 36
6.3. Transient versus persistent critic ... 36
6.3.1. Trading bias against variance ... 36
6.3.2. Off-policy versus on-policy updates ... 37
7. Conclusion ... 37
7.1. Future directions ... 37
7.1.1. More analyses than competitions ... 37
7.1.2. More combinations than competitions ... 37
7.1.3. Beyond single policy improvement ... 37
7.2. Final word ... 37
Acknowledgments ... 38
References ... 38

1. Introduction

Autonomous systems are systems which know what to do in their domain without external intervention. Generally, their behavior is specified through a *policy*. The policy of a robot, for instance, is defined through a controller which determines actions to take or signals to send to the actuators for any state of the robot in its environment.

Robot policies are often designed by hand, but this manual design is only viable for systems acting in well-structured environments and for well-specified tasks. When those conditions are not met, a more appealing alternative is to let the system find its own policy by exploring various behaviors and exploiting those that perform well with respect to some predefined *utility* function. This approach is called *policy search*, a particular case of *reinforcement learning* (RL) (Sutton & Barto, 1998) where actions are vectors from a continuous space. More precisely, the goal of policy search is to optimize a policy where the function relating behaviors to their utility is *black-box*, i.e. no analytical model or gradient of the utility function is available. In practice, a policy search algorithm runs the system with some policies to generate *rollouts* made of several state and action *steps* and gets the utility as a return (see Fig. 1). These utilities are then used to improve the policy, and the process is repeated until some satisfactory set of behaviors is found. In general, policies are represented with a parametric function, and policy search explores the space of *policy parameters*. For doing so, rollout and utility data are processed by policy improvement algorithms as a set of *samples*.

In the context of robotics, *sample efficiency* is a key concern. There are three aspects to sample efficiency: (1) data efficiency, i.e. extracting more information from available data (definition taken from Deisenroth and Rasmussen (2011)), (2) sample choice, i.e. obtaining data in which more information is available and (3) sample reuse, i.e. improving a policy several times by using the same samples more than once through *experience replay*.

In this paper, we provide a broad overview of policy search algorithms under the perspective of these three aspects.

1.1. Scope and contributions

Three surveys about policy search for robotics have been published in recent years (Deisenroth, Neumann, Peters, et al., 2013; Kober, Bagnell, & Peters, 2013; Stulp & Sigaud, 2013). With respect to these previous surveys, we cover a broader range of policy search algorithms, including optimization without a utility model, Bayesian optimization (BO), directed exploration methods, and deep RL. The counterpart of this breadth is that we do not give a detailed account of the corresponding algorithms nor their mathematical derivation. To compensate for this lack of details, we refer the reader to Deisenroth et al. (2013) for the mathematical derivation and description of most algorithms before 2013, and we provide carefully chosen references as needed when describing more recent algorithms.

Furthermore, we focus on the case where the system is learning to solve a *single task*. That is, we do not cover the broader domain of *lifelong*, *continual* or *open-ended* learning, where a robot must learn how to perform various tasks over a potentially infinite horizon (Thrun & Mitchell, 1995). Additionally, though a subset of policy search methods are based on RL, we do not cover recent work on RL with discrete actions such as DQN and its successors (Hessel et al., 2017; Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, et al., 2015). Finally, we restrict ourselves to the case where samples are the unique source of information for improving the policy. That is, we do not consider the interactive context where a human user can provide external guidance (Najar, Sigaud, & Chetouani, 2016), either through feedback, shaping or demonstration (Argall, Chernova, Veloso, & Browning, 2009).

1.2. Perspective and structure of the survey

The main message of this paper is as follows. In optimization, when the utility function to be optimized is known and convex, efficient convex methods can be applied (Gill, Murray, & Wright, 1981). If the function is known but not convex, a local optimum can be found using gradient descent, iteratively moving from the current point towards a local optimum by following the direction provided by the derivative of the function at this point. If the function is *black-box*, neither the function nor its analytic gradient are known. In policy gradient methods, policy parameters are only related to their utility *indirectly* through an intermediate set of observed behaviors. Given that policy search corresponds to this more difficult context, we consider five solutions:

1. searching for high utility policy parameters without building a utility model (Section 2),
2. learning a model of the utility function in the space of policy parameters and performing stochastic gradient descent (SGD) using this model (Section 3),
3. defining an arbitrary *outcome space* and using directed exploration of this outcome space for finding high utility policy parameters (Section 4),
4. doing the same as in Solution 2 in the state–action space (Section 5),
5. learning a model of the transition function of the system-environment interaction that predicts the next state given the current state and action, to generate samples without using the system, and then applying one of the above solutions based on the generated samples.

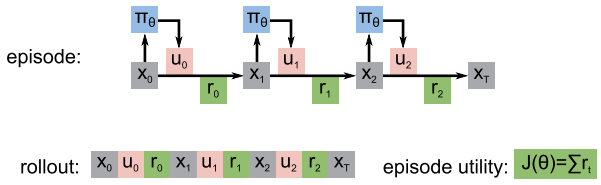

Fig. 1. Visualization of one episode, the information contained in a rollout, and the definition of the episode utility (which is also known as the *episode return* when the utility is a reward).

An important distinction in the policy search domain is whether the optimization method is episode-based or step-based (Deisenroth et al., 2013).[1] The first three solutions above are episode-based, the fourth is step-based and the fifth can be applied to all others.

Resulting from the above perspective, this survey is structured following the organization of methods depicted in Fig. 2. The rest of the paper describes the different nodes in the trees and highlights some of their sample efficiency factors. A table giving a quick reference to the main paper for each algorithm is given in the end of each section.

In Section 2 to 5, we present Solutions 1 to 4 in more detail, showing how the corresponding methods are implemented in various policy search algorithms. We do not cover Solution 5 and refer readers to Chatzilygeroudis et al. (2017) for a recent presentation of these *model-based policy search* methods. Then, in Section 6, we discuss the different elementary design choices that matter in terms of sample efficiency. Finally, Section 7 summarizes the paper and provides some perspectives about current trends in the domain.

2. Policy search without a utility model

When the function to be optimized is available but has no favorable property, the standard optimization method known as Gradient Descent consists in iteratively following the gradient of this function towards a local optimum. When the same function is only known through a model built by regression from a batch of samples, one can also do the same, but computing the gradient requires evaluations over the whole batch, which can be computationally expensive. An alternative known as Stochastic Gradient Descent (SGD) circumvents this difficulty by taking a small subset of the batch at each iteration (Bottou, 2012). Before starting to present these methods in Section 3, we first investigate a family of methods which perform policy search without learning a model of the utility function at all. They do so by sampling the policy parameter space $\boldsymbol{\Theta}$ and moving towards policy parameters $\boldsymbol{\theta}$ of higher utility $J(\boldsymbol{\theta})$.

2.1. Truly random search

At one extreme, the simplest black-box optimization (BBO) method randomly searches $\boldsymbol{\Theta}$ until it stumbles on a good enough utility. We call this method "Truly random search" as the name "random search" is used in the optimization community to refer to gradient-free methods (Rastrigin, 1963). Its distinguishing feature is in its sample choice strategy: the utility of the previous $\boldsymbol{\theta}$ has no impact on the choice of the next $\boldsymbol{\theta}$.

Quite obviously, this sample choice strategy is not efficient, but it requires no assumption at all on the function to be optimized. Therefore, it is an option when this function does not show any *regularity* that can be exploited. All other methods rely on the implicit assumption that $J(\boldsymbol{\theta})$ presents some smoothness around optima $\boldsymbol{\theta}^*$, which is a first step towards using a gradient.

So globally, this method provides a proof of concept that an agent can obtain a better utility without estimating any gradient at all. Recently, other forms of gradient-free methods called *random search* though they are not truly random have been shown to be competitive with deep RL (Mania, Guy, & Recht, 2018).

The next three families of methods, population-based optimization, evolutionary strategies and estimation of distribution algorithms, are all variants of *evolutionary* methods. An overview[2] of these methods is depicted in Fig. 3.

2.2. Population-based optimization

Population-based BBO methods manage a limited population of individuals, and generate new individuals randomly in the vicinity of the previous *elite* individuals. There are several families of population-based optimization methods, the most famous being Genetic Algorithms (GAs) (Goldberg, 1989), Genetic Programming (GP) (Koza, 1992), and the more advanced NEAT framework (Stanley & Miikkulainen, 2002). In these methods, the parameter $\boldsymbol{\theta}$ corresponding to an individual is often called its *genotype* and the corresponding utility is called its *fitness*, see Back (1996) for further reading. These methods have already been combined with neural networks giving rise to *neuroevolution* (Floreano, Dürr, & Mattiussi, 2008) but, up to recently, these methods were mostly applied to small to moderate size policy representations. However, the availability of modern computational resources have made it possible to apply them to large and deep neural network representations, defining the emerging domain of *deep neuroevolution* (Petroski Such, Madhavan, Conti, Lehman, Stanley, & Clune, 2017). Among other things, it was shown that, given large enough computational resources, methods as simple as GAs can offer a competitive alternative to deep RL methods presented in Section 5, mostly due to their excellent parallelization capabilities (Conti et al., 2017; Petroski Such et al., 2017).

2.3. Evolutionary strategies

Evolutionary strategies (ES) can be seen as specific population-based optimization methods where only one individual is retained from one generation to the next. More specifically, an optimum guess is computed from the previous samples, then the next samples are obtained by adding Gaussian noise to the current optimum guess.

Moving from one optimum guess to the next implements a form of policy improvement similar to SGD, but where the gradient is approximated by averaging over samples instead of being analytically computed. Hence this method is more flexible but, since gradient approximation uses a random exploration component, it is less data efficient. However, data efficiency can be improved by reusing samples between one generation and the next when their sampling domain overlaps, a method called *importance mixing* (Sun, Wierstra, Schaul, & Schmidhuber, 2009). An improved version of importance mixing was recently proposed in Pourchot, Perrin, and Sigaud (2018), showing a large impact on sample efficiency, but not large enough to compete with deep RL methods on this aspect. Further results about importance mixing can be found in Pourchot and Sigaud (2018), showing that more investigations are necessary to better understand in which context this mechanism can be most useful.

[1] This distinction exactly matches the phylogenetic RL versus ontogenetic RL distinction in Togelius et al. (2009).

[2] A blog with dynamical visualizations and more technical details can be found at http://blog.otoro.net/2017/10/29/visual-evolution-strategies/.

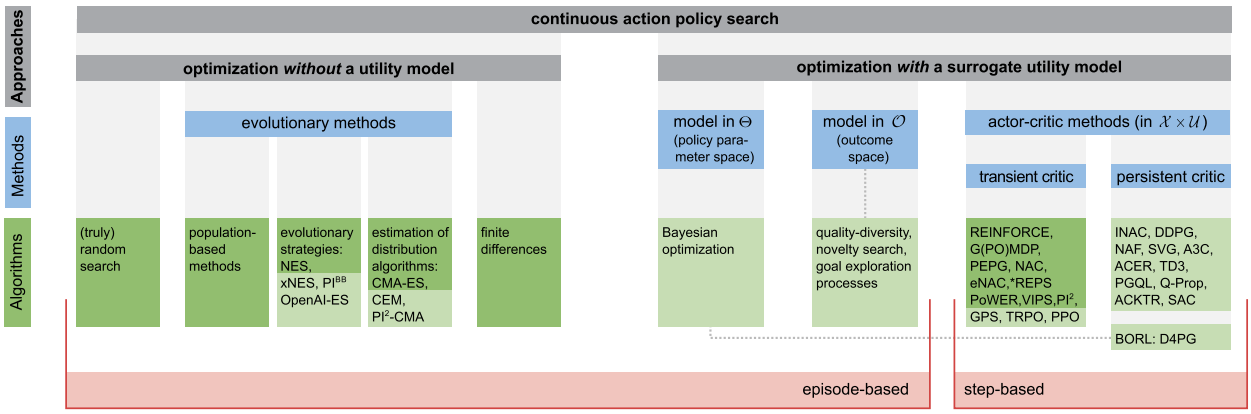

Fig. 2. Simplified classification of the algorithms covered in the paper. Θ is the space of policy parameters (see Section 2), O is an outcome space (see Section 4) and $\mathcal{X} \times \mathcal{U}$ is the state and action space (see Section 5). Algorithms not covered in Deisenroth et al. (2013) have a lighter (green) background. References to the main paper for each of these algorithms are given in a table in the end of each section. From the left to the right, algorithms are grossly ranked in order of increasing sample reuse, but methods using a utility model in Θ and O show better sample choice, resulting in competitive sample efficiency.

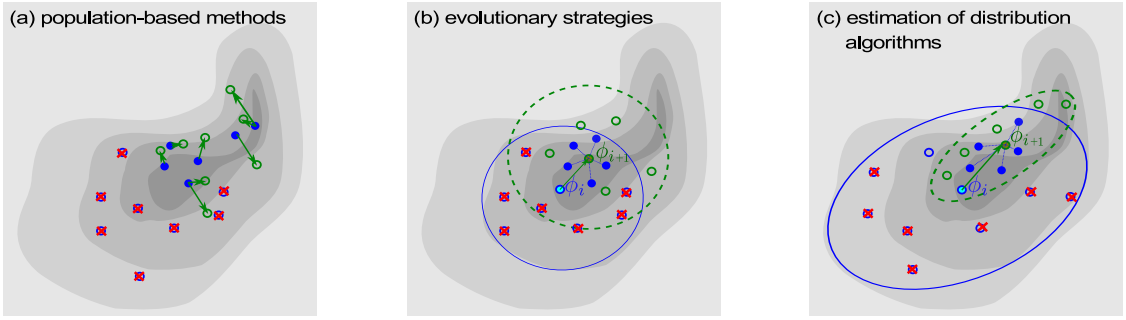

Fig. 3. One iteration of evolutionary methods. (a) Population-based methods (b) Evolutionary Strategies (c) EDAs. Blue: current generation and sampling domain. Full blue dots: samples with a good evaluation. Dots with a red cross: samples with a poor evaluation. Green: new generation and sampling domain, empty dots are not evaluated yet. In population-based methods, the next generation are offspring from several elite individuals of the previous generation. In ES, it is obtained from an optimum guess and sampling from fixed Gaussian noise. In EDAs, Gaussian noise is tuned using Covariance Matrix Adaptation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The correlation between the direction of the gradient given by SGD and the same direction for ES depends on the evolutionary algorithm. Interestingly, good ES performance can be obtained even when the correlation is not high, though this result still needs to be confirmed in the case of policy search (Zhang, Clune, & Stanley, 2017).

A specific ES implementation of deep neuroevolution where constant Gaussian noise is used at each generation was shown to compete with deep RL methods on standard benchmarks (Salimans, Ho, Chen, & Sutskever, 2017). This simple implementation generated an insightful comparison with methods based on SGD depending on various gradient landscapes, showing under which conditions ES can find better optima than SGD (Lehman, Chen, Clune, & Stanley, 2017).

Finally, instead of approximating the vanilla gradient of utility, NES (Wierstra, Schaul, Peters, & Schmidhuber, 2008) and xNES (Glasmachers, Schaul, Yi, Wierstra, & Schmidhuber, 2010) approximate its natural gradient (Akimoto, Nagata, Ono, & Kobayashi, 2010), but for doing so they have to compute the inverse of the Fisher Information Matrix, which is prohibitively expensive in large dimensions (Grondman, Busoniu, Lopes, & Babuska, 2012). We refer the reader to Pierrot, Perrin, and Sigaud (2018) for a detailed presentation of *natural gradient* and other advanced gradient descent concepts.

2.4. Estimation of distribution algorithms

The standard perspective about EDAs is that they are a specific family of ES using a covariance matrix $\boldsymbol{\Sigma}$ (Larrañaga & Lozano, 2001). This covariance matrix defines a multivariate Gaussian function over $\boldsymbol{\Theta}$, hence its size is $|\boldsymbol{\theta}|^2$. Samples at the next iteration are drawn with a probability proportional to this Gaussian function. Along iterations, the ellipsoid defined by $\boldsymbol{\Sigma}$ is progressively adjusted to the top part of the hill corresponding to the local optimum $\boldsymbol{\theta}^*$.

The role of $\boldsymbol{\Sigma}$ is to control exploration. The exploration policy can be characterized as *uncorrelated* when it only updates the diagonal of $\boldsymbol{\Sigma}$ and *correlated* when it updates the full $\boldsymbol{\Sigma}$ (Deisenroth et al., 2013). The latter is more efficient in small parameter spaces but computationally more demanding and potentially inaccurate in larger spaces as more samples are required. In particular, it cannot be applied in the deep neuroevolution context where the order of magnitude of the size of $\boldsymbol{\theta}$ is between thousands and millions.

Various instances of EDAs, such as CEM, CMA-ES, PI^2-CMA, are covered in Stulp and Sigaud (2012a,b, 2013). Among them, the CMA-ES algorithm is also shown to approximate the natural gradient (Arnold, Auger, Hansen, & Ollivier, 2011). By contrast, the PI^{BB} algorithm, also described in Stulp and Sigaud (2013), is a simplification of PI^2-CMA where covariance matrix adaptation has been removed. Thus it should be considered an instance of the former ES category.

2.5. Finite difference methods

In *finite difference* methods, the gradient of utility with respect to $\boldsymbol{\theta}$ is estimated as the first order approximation of the Taylor

Table 1
Main gradient-free algorithms. Above the line, they were studied in Deisenroth et al. (2013), below they were not.

Algorithm	Main paper
CMA-ES	(Hansen & Ostermeier, 2001)
CEM	(Rubinstein & Kroese, 2004)
Finite diff.	(Riedmiller et al., 2008)
NES	(Wierstra et al., 2008)
XNES	(Glasmachers et al., 2010)
PI^{BB}	(Stulp & Sigaud, 2012b)
PI^2-CMA	(Stulp & Sigaud, 2012a)
OpenAI-ES	(Salimans et al., 2017)
Random search	(Mania et al., 2018)

expansion of the utility function. This estimation is performed by applying local perturbations to the current input. Thus these methods are derivative-free and we classify them as using no model, even if they are based on a local linear approximation of the gradient.

In finite difference methods, gradient estimation can be cast as a standard regression problem, but perturbations along each dimension of $\boldsymbol{\Theta}$ can be treated separately, which results in a very simple algorithm (Riedmiller, Peters, & Schaal, 2008). The counterpart of this simplicity is that it suffers from a lot of variance, so in practice the methods are limited to deterministic policies.

2.6. Reference to the main algorithms

See Table 1.

2.7. Sample efficiency analysis

In all gradient-free methods, sampling a vector of policy parameters $\boldsymbol{\theta}$ provides an exact information about its utility $J(\boldsymbol{\theta})$. However, the J function can be stochastic, in which case one value of $J(\boldsymbol{\theta})$ only contains partial information about the value of that $\boldsymbol{\theta}$. Anyways, sample reuse can be implemented by storing an archive of the already sampled pairs $\langle \boldsymbol{\theta}, J(\boldsymbol{\theta}) \rangle$. Each time an algorithm needs the utility $J(\boldsymbol{\theta})$ of a sample $\boldsymbol{\theta}$, if this utility is already available in the archive, it can use it instead of sampling again. In the deterministic case, using the stored value is enough. In the stochastic case, the archive may provide a distribution over values $J(\boldsymbol{\theta})$, and the algorithm may either draw a value from this distribution or sample again, depending on accuracy requirements.

Message 1: Policy search without a utility model is generally less data efficient than Stochastic Gradient Descent (SGD). Though sample reuse is technically possible without a utility model, in practice it is seldom used. Despite their lower sample efficiency in comparison to SGD, some of these methods are highly parallelizable and offer a viable alternative to deep RL provided enough computational resources.

3. Policy search with a model of utility in the space of policy parameters

As outlined in the introduction, the utility of a vector of policy parameters is only available by observing the corresponding behavior. Although no model that relates policy parameters to utilities is given, one may approximate the utility function in $\boldsymbol{\Theta}$ from these observations, by collecting samples consisting of (policy parameters, utility) pairs and using regression to infer a model of the corresponding function (see e.g. Stulp and Sigaud (2015). Such a model could either be deterministic, giving one utility per policy parameters vector, or stochastic, giving a distribution over utility values.

Once such a model is learned, one could perform gradient descent on this model. These steps could be performed sequentially (first model learning and then gradient descent) or incrementally (improving the model and performing gradient descent after each new utility observation). In the latter case, the model is necessarily *persistent*: it evolves from iteration to iteration given new information, in contrast with the sequential case where it could be *transient*, that is recomputed from scratch at each iteration.

3.1. Bayesian optimization

Though the above approach seems appealing, we are not aware of any algorithm performing what is described above in the deterministic case. A good reason for this is that utility functions are generally stochastic in $\boldsymbol{\Theta}$. Thus, algorithms which learn a model $J(\boldsymbol{\theta})$ have to learn a distribution over such models. This is exactly what Bayesian optimization (BO) does. The *distribution* over models is updated through Bayesian inference. It is initialized with a *prior*, and each new sample, considered as some new *evidence*, helps adjusting the model distribution towards a peak at the true value, whilst keeping track of the variance over models. By estimating the uncertainty over the distribution of models, BO methods are endowed with active learning capabilities, dramatically improving their sample efficiency at the cost of a worse scalability.

A BO algorithm comes with a *covariance function* that determines how the information provided by a new sample influences the model distribution around this sample. It also comes with an *acquisition function* used to choose the next sample given the current model distribution. A good acquisition function should take into account the value and the uncertainty of the model over the sampled space.

By quickly reducing uncertainty, BO implements a form of active learning. As a consequence, it is very sample efficient when the parameter space is small enough, and it searches for the global optimum, rather than a local one. However, given the necessity to optimize globally over the acquisition function, it scales poorly in the size of the parameter space. For more details, see Brochu, Cora, and De Freitas (2010).

The ROCK* algorithm is an instance of BO that searches for a local optimum instead of a global one (Hwangbo, Gehring, Sommer, Siegwart, & Buchli, 2014). It uses CMA-ES to find the optimum over the model function. By doing so, it performs natural rather than vanilla gradient optimization, but it does not use the available model of the utility function, though this could improve sample efficiency.

Bayesian optimization algorithms generally use Gaussian kernels to efficiently represent the distribution over models. However, some authors have started to note that, in the specific context of policy search, BO was not using all the information available in elementary steps of the agent. This led to the investigation of more appropriate data-driven kernels based on the Kullback–Leibler divergence between rollout density generated by two policies (Wilson, Fern, & Tadepalli, 2014).

Using BO in the context of policy search is an emerging domain (Calandra, Gopalan, Seyfarth, Peters, & Deisenroth, 2014; Lizotte, Wang, Bowling, & Schuurmans, 2007; Martinez-Cantin, Tee, & McCourt, 2017; Metzen, Fabisch, & Hansen, 2015). Furthermore, recent attempts to combine BO with reinforcement learning approaches, giving rise to the Bayesian Optimization Reinforcement Learning (BORL) framework, are described in Section 5.

3.2. Reference to the main algorithms

See Table 2.

Table 2
Main Bayesian optimization algorithms.

Algorithm	Main paper
Bayes. Opt.	(Pelikan, Goldberg, & Cantú-Paz, 1999)
ROCK*	(Hwangbo et al., 2014)

3.3. Sample efficiency analysis

Learning a model of the utility function in Θ should be more sample efficient than trying to optimize without a model, as the gradient with respect to the model can be used to accelerate parameter improvement. However, learning a deterministic model is not enough for most cases, as the true utility function is generally stochastic in Θ, and learning a stochastic model comes with an additional computational cost which impacts the scalability of the approach.

Message 2: Bayesian Optimization is BBO managing a distribution over models in the policy parameter space. Its sample efficiency benefits from active choice of samples. But as it performs global search, it does not scale well to large policy parameter spaces. Thus, it is difficult to apply to deep neural network representations.

4. Directed exploration methods

Directed exploration methods are particularly useful in tasks with *sparse rewards*, i.e. where large parts of the search space have the same utility signal. These methods have two main features. First, instead of searching directly in the policy parameter space Θ, they search in a smaller *outcome space* O (also called *descriptor space* or *behavioral space*) and learn an invertible mapping between Θ and O. Second, they all optimize a task-independent criterion called novelty or diversity which is used to efficiently cover the outcome space.

The outcome itself corresponds to properties of the observed behavior. The general intuition is that if the outcome space is properly covered by known policy parametrizations, and if utility can be easily related to outcomes, then it should be easy to find policy parameters with a high utility, even when the utility function is null for most policy parameters. Fig. 4 visualizes why it is generally more efficient to perform the search for novel solutions in a dedicated outcome space and learning a mapping from Θ to O than performing this search directly in Θ (Baranes, Oudeyer, & Gottlieb, 2014).

So, for the method to work, the outcome space has to be defined in such a way that determining the utility corresponding to an outcome is straightforward. Generally, the outcome space is defined by an external user to meet this requirement. Nevertheless, using representation learning methods to let the agent autonomously define its own outcome space is an emerging topic of interest (Laversanne-Finot, Péré, & Oudeyer, 2018; Pere, Forestier, Sigaud, & Oudeyer, 2018).

Directed exploration methods can be split into *novelty search* (NS) (Lehman & Stanley, 2011), *quality-diversity* (QD) (Pugh, Soros, Szerlip, & Stanley, 2015) and *goal exploration processes* (GEPS) (Baranes & Oudeyer, 2010; Forestier, Mollard, & Oudeyer, 2017; Forestier & Oudeyer, 2016). The first two derive from evolutionary methods, whereas GEPS come from the developmental learning and intrinsic motivation literature.

An important distinction between them is that NS and GEPS are designed to optimize diversity only,[3] thus they do not use the utility function at all, whereas QD methods rely on multi-objective optimization methods to optimize diversity and utility at the same time.

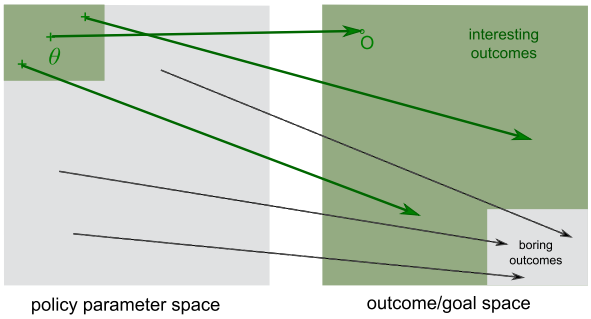

Fig. 4. A standard mapping between a policy parameter space Θ and an outcome space O. Most often, many policy parameters result in the same outcome (for instance, in the case of a robotic arm which must move a ball around, if the policy defines arm movements and the outcome space is defined as ball positions, most policy parameters will result into a static ball). In that case, sampling directly in Θ works poorly: you have to sample in such a way to efficiently cover O.

The NS approach arose from the realization that optimizing utility as a single objective is not the only option (Doncieux & Mouret, 2014). In particular, in the case of sparse or deceptive reward problems, it was shown that seeking novelty or diversity is an efficient strategy to obtain high utility solutions, even without explicitly optimizing this utility (Lehman & Stanley, 2011). The GEP approach was more inspired by thoughts on intrinsic motivations, where the goal was to have an agent achieve its own goal without an external utility signal (Forestier et al., 2017). However, researchers in evolutionary methods also realized that diversity and utility can be optimized jointly (Cuccu & Gomez, 2011), giving rise to more advanced NS and QD algorithms (Cully & Demiris, 2017; Pugh et al., 2015).

All these methods share a lot of similarities. They all start with a random search stage and, when they evaluate a policy parameter vector θ resulting in a point o in the outcome space O, they store the corresponding $\langle \theta, o \rangle$ pair in an archive. Because they use this archive for policy improvement, they all implement a form of *lazy learning*, endowing them with interesting sample efficiency properties (Aha, 1997). The archive itself can be seen as a stochastic model of the function relating Θ to O, as made particularly obvious in the *MAP-Elites* algorithm (Cully, Clune, Tarapore, & Mouret, 2015).

In more detail, the main differences between these methods lie in the way they cover the outcome space O. NS and QD methods perform undirected variations to the elite θ vectors present in the archive. More precisely, in NS, the resulting solution is just added to the archive, whereas in QD the new solution replaces a previous one if it outperforms it both in terms of diversity and utility. By contrast, GEPS choose a desired outcome o^* and modify a copy of the θ leading to the closest outcome in the archive. The choice of a desired outcome o^* can be performed randomly or using curriculum learning or learning progress concepts (Baranes & Oudeyer, 2013; Forestier et al., 2017). Similarly, the modification of θ can be performed using undirected Gaussian noise or in more advanced ways. For instance, some GEP methods build a local linear model of the mapping from Θ to O to efficiently invert it, so as to find the θ^* corresponding to the desired outcome o^* (Baranes & Oudeyer, 2013).

Thus directed exploration methods all learn a stochastic and invertible mapping between Θ and O. When they also learn a model of $J(o)$, this model is stochastic with respect to Θ, which makes them similar to BO methods. In that case, the outcome space is an intermediate space between Θ and utilities: policy

[3] Hence the dotted line in Fig. 2.

Table 3
Main directed exploration algorithms.

Algorithm	Main paper
Novelty search	(Lehman & Stanley, 2011)
Quality-Diversity	(Pugh et al., 2015)
Goal exploration	(Forestier et al., 2017)

parameters are first projected into the outcome space, and then a model of the utility function in this outcome space can be learned.

Learning a model of the utility in O shares some similarities with learning a critic in the state action space $\mathcal{X} \times \mathcal{U}$, as presented in Section 5. From this perspective, these methods can be seen as providing an intermediary family between evolutionary, BO and reinforcement learning methods. However, we shall see soon that learning a critic in the state action space $\mathcal{X} \times \mathcal{U}$ benefits from additional properties related to temporal difference learning, which limits the use of the above unifying perspective.

4.1. Reference to the main algorithms

See Table 3.

4.2. Sample efficiency analysis

The defining characteristic of all directed exploration methods is their capability to widely cover the outcome space. This provides efficient exploration, which in turn critically improves sample efficiency when combined with more standard evolutionary methods mentioned in Section 2 (Conti et al., 2017) or deep RL method mentioned in Section 5 (Colas, Sigaud, & Oudeyer, 2018).

Even though our article focuses on single-task learning, it is worth mentioning that direct exploration methods may very much improve sample efficiency in multi-task learning scenarios. This is because such methods aim at covering the (interesting) outcome space, and can thus more easily adapt when facing multiple tasks, and thus potentially multiple outcomes.

Message 3: Looking for diversity only in a user defined outcome space is an efficient way to perform exploration, and can help solve sparse or deceptive reward problems, where more standard exploration would fail. Directed exploration methods are thus useful complements to other methods covered in this survey.

5. Policy search with a critic

The previous two sections have presented methods which learn mappings from policy parameter space $\boldsymbol{\Theta}$ to utilities or outcomes. We now cover methods which learn a model of utility in the state–action space $\mathcal{X} \times \mathcal{U}$.

An important component in the RL formalization, the utility $U(\mathbf{x}, \mathbf{u})$ corresponds to the return the agent may expect from performing action $\mathbf{u}$ when it is in state $\mathbf{x}$ and then following either its current policy $\pi_{\boldsymbol{\theta}}$ or the optimal policy π^*. This quantity may also depend on a discount factor γ and a noise parameter β.

The true utility $U(\mathbf{x}, \mathbf{u})$ can be approximated with a model $\hat{U}_\eta(\mathbf{x}, \mathbf{u})$ with parameters η. Such a model is called a *critic*. A key feature is that the critic can be learned from samples corresponding to *single steps* in the rollouts of the agent, either with temporal differencing or Monte Carlo methods. Methods that approximate U by $\hat{U}_\eta$, and determine the policy parameters $\boldsymbol{\theta}$ by descending the gradient of $\boldsymbol{\theta}$ with respect to $\hat{U}_\eta$ are called *actor–critic* methods, the policy $\pi_{\boldsymbol{\theta}}$ being the actor (Deisenroth et al., 2013; Peters & Schaal, 2008b).

This actor–critic approach can be applied to stochastic and deterministic policies (Silver et al., 2014). The space of deterministic policies being smaller than the space of stochastic policies, the latter can be advantageous because searching the former is faster than searching the latter. However, a stochastic policy might be more appropriate when Markov property does not hold (Sigaud & Buffet, 2010; Williams & Singh, 1998) or in adversarial contexts (Wang, Bapst et al., 2016).

5.1. Exploration in parameter or state–action space

As mentioned in Section 3, learning a model of the utility in the space $\boldsymbol{\Theta}$ is a regression problem that is performed by sampling and exploring directly in $\boldsymbol{\Theta}$. In contrast, $\mathcal{X} \times \mathcal{U}$ cannot be sampled directly, as one does not know in advance which policy parameters will result in visiting which states and performing which action. Exploration is therefore performed in either by adding noise to the $\boldsymbol{\theta}$ (policy parameter perturbation), or adding noise to the actions the policy outputs (action perturbation). In the latter case, exploration is generally undirected and adds Gaussian noise or correlated Ornstein–Uhlenbeck noise to the actions taken by the policy. Policy parameter perturbation is done in PEPG, POWER and PI^2, and more recently to DDPG (Fortunato et al., 2017; Plappert et al., 2017), whereas action perturbation in the other algorithms presented in this paper.

All actor–critic algorithms iterate over the following three steps:

A Collect new step samples from the current policy with policy parameter perturbation or action perturbation for exploration,
B Compute a new critic $\hat{U}_\eta$ based on these samples, by determining η through a temporal difference method,
C Update the policy parameters $\boldsymbol{\theta}$ through gradient descent with respect to the critic.

A distinction here should be made on whether (1) the critic is discarded after step C, and must thus be learned from scratch in step B in the next iteration, or (2) the critic is persistent throughout the learning, and incrementally updated in step B. We discuss the differences between these two variants – which we denote *transient critic* and *persistent critic* respectively – in more detail in the next two sections.

5.2. Transient critic algorithms

In methods with a transient critic, Monte Carlo sampling – running a large set of episodes and averaging over the stochastic return – is used to evaluate the current policy and generate a new set of step samples. Then, determining the optimal critic parameters given these samples can be cast as a batch regression problem.

Among these methods, one must distinguish between three families: *likelihood ratio* methods such as REINFORCE (Williams, 1992) and PEPG (Sehnke et al., 2010), *natural gradient* methods such as NAC and eNAC[4] (Peters & Schaal, 2008a) and *EM-based* methods such as POWER[5] (Kober & Peters, 2009) and the variants of REPS (Peters, Mülling, & Altun, 2010). All the corresponding algorithms are well described in Deisenroth et al. (2013).

Although they derive from a different mathematical framework, likelihood ratio methods and EM-based methods are similar: they both use unbiased estimation of the gradient through Monte Carlo

[4] The critic is generally persistent in actor–critic methods, but this is not the case in NAC and eNAC.

[5] Interestingly, PI^2 can also be seen as a transient critic method, though it could in principle use a persistent one and fall into Section 5.3. This is just because using batch updates make it more stable (Deisenroth et al., 2013).

sampling and they are both mathematically designed so that the most rewarding rollouts get the highest probability.

The TRPO (Schulman, Levine, Moritz, Jordan, & Abbeel, 2015) algorithm also follows an iterative approach and can use a deep neural network representation, thus it can be classified as a deep RL method. Among other things, it uses a bound on the Kullback–Leibler divergence between policies at successive iterations to ensure safe and efficient exploration. Finally, the Guided Policy Search (GPS) algorithm (Levine & Koltun, 2013; Montgomery & Levine, 2016) is another transient critic deep RL method inspired from REINFORCE, but adding guiding rollouts obtained from simpler policies.

5.3. Persistent critic algorithms

In contrast with transient critic algorithms, persistent critic algorithms incrementally update the critic during training. Most such algorithms use an actor–critic architecture, with the notable exception of NAF (Gu, Lillicrap, Sutskever, & Levine, 2016), which does not have an explicit representation of the actor. To our knowledge, before the emergence of deep RL algorithms described below, the four iNAC algorithms were the only representative of this family (Bhatnagar, Sutton, Ghavamzadeh, & Lee, 2007).

The way to compute the critic incrementally can be named a *temporal difference* (TD) method, also named a *bootstrap* method (Sutton, 1988). They compute at each step a *temporal difference error* or *reward prediction error* (RPE) between the immediate reward predicted by the current values of the critic and the actual reward received by the agent. This RPE can then be used as a loss that the critic should minimize over iterations (Sutton & Barto, 1998).

5.4. Key properties of persistent critic algorithms

Most mechanisms that made deep actor–critic algorithms possible where first introduced in DQN (Mnih et al., 2015). Though DQN is a discrete action algorithm which is outside the scope of this survey, we briefly review its important concepts and mechanisms before listing the main algorithms in the family of continuous action deep RL methods.

5.4.1. Accuracy and scalability: deep neural networks

By using deep neural networks as approximation functions and making profit of large computational capabilities of modern clusters of computers, all deep RL algorithms are capable of addressing much larger problems than before, and to approximate gradients with unprecedented accuracy, which makes them more stable than the previous linear architectures of NAC and POWER, hence amenable to incremental updates of a persistent critic rather than recomputing a transient one.

5.4.2. Stability: the target critic

Deep RL methods introduced a *target critic* as a way to improve stability. Standard regression is the process of fitting samples to a model so as to approximate an unknown stationary function (Stulp & Sigaud, 2015). Estimating a critic through temporal difference methods is similar to regression, but the target function is not stationary: it is itself a function of the estimated critic, thus it is modified each time the critic is modified. This can result in divergence of the critic when the target function and the estimated critic are racing after each other (Baird, 1994). To mitigate this instability, one should keep the target function stationary during several updates and reset it periodically to a new function corresponding to the current critic estimate, switching from a regression problem to another. This idea was first introduced in DQN (Mnih et al., 2015) and then modified from periodic updates to smooth variations in DDPG (Lillicrap et al., 2015).

5.4.3. Sample reuse: the replay buffer

Since they are based on *value propagation*, TD methods can give rise to more sample reuse than standard regression methods, provided that these samples are saved into a *replay buffer*. Using a replay buffer is at the heart of the emergence of modern actor–critic approaches in deep RL. Actually, learning from the samples in the order in which they are collected is detrimental to learning performance and stability because these samples are not independent and identically distributed (i.i.d.). Stability is improved by drawing the samples randomly from the replay buffer and sample efficiency can be further improved by better choosing the samples, using *prioritized experience replay* (Schaul, Quan, Antonoglou, & Silver, 2015).

5.4.4. Adaptive step sizes and return lengths

Modern SGD methods provided by most machine learning libraries now incorporate adaptive step sizes, removing a difficulty with previous actor–critic algorithms such as eNAC. Another important ingredient for the success of some recent methods in the use of *n-step return*, which consists in performing temporal difference updates over several time steps, resulting in the possibility to control the bias–variance trade-off (see Section 6.3.1).

5.5. Overview of deep RL algorithms

All these favorable properties are common traits of several incremental deep RL algorithms: DDPG (Lillicrap et al., 2015), NAF (Gu, Lillicrap, Sutskever et al., 2016), PPO (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017), ACKTR (Wu, Mansimov, Liao, Grosse, & Ba, 2017), SAC (Haarnoja, Zhou, Abbeel, & Levine, 2018), TD3 (Fujimoto, van Hoof, & Meger, 2018) and D4PG (Barth-maron et al., 2018). As depicted in Fig. 2, the last one, D4PG, is an instance of Bayesian Optimization Reinforcement Learning (BORL) algorithms which derive from BO but belong to the step-based category of methods described in Section 5. These algorithms result from an effort to incorporate Bayesian computations into the deep RL framework, and correspond to a very active trend in the field. Most of these works address discrete actions (Azizzadenesheli, Brunskill, & Anandkumar, 2018; Tang & Kucukelbir, 2017), but D4PG is an exception that derives from adopting a distributional perspective on policy gradient computation, resulting in more accurate estimates on the gradient and better sample efficiency (Bellemare, Dabney, & Munos, 2017).

Finally, a few algorithms such as ACER (Wang, Bapst et al., 2016), Q-PROP (Gu, Lillicrap, Ghahramani, Turner, & Levine, 2016) and PGQL (O'Donoghue, Munos, Kavukcuoglu, & Mnih, 2016) combine properties of transient and persistent critic methods, and are captured into the more general framework of Interpolated Policy Gradient (IPG) (Gu et al., 2017). For a more detailed description of all these algorithms, we refer the reader to the corresponding papers and to a recent survey (Arulkumaran, Deisenroth, Brundage, & Bharath, 2017).

5.6. Reference to the main algorithms

See Table 4.

5.7. Sample efficiency analysis

Message 4: Being step-based, deep RL methods are able to use more information from rollouts than episode-based methods. Furthermore, using a replay buffer leads to further sample reuse.

Table 4
Main reinforcement learning algorithms. Algorithms below the line have not yet been covered in Deisenroth et al. (2013).

Algorithm	Main paper
REINFORCE	(Williams, 1992)
G(PO)MDP	(Baxter & Bartlett, 2001)
NAC	(Peters & Schaal, 2008a)
eNAC	(Peters & Schaal, 2008a)
POWER	(Kober & Peters, 2009)
PI^2	(Theodorou, Buchli, & Schaal, 2010)
REPS	(Peters et al., 2010)
PEPG	(Sehnke et al., 2010)
VIPS	(Neumann, 2011)
iNAC	(Bhatnagar et al., 2007)
GPS	(Levine & Koltun, 2013)
TRPO	(Schulman et al., 2015)
DDPG	(Lillicrap et al., 2015)
A3C	(Mnih et al., 2016)
NAF	(Gu, Lillicrap, Sutskever et al., 2016)
ACER	(Wang, Bapst et al., 2016)
Q-PROP	(Gu, Lillicrap, Ghahramani et al., 2016)
PGQL	(O'Donoghue et al., 2016)
PPO	(Schulman et al., 2017)
ACKTR	(Wu et al., 2017)
SAC	(Haarnoja et al., 2018)
TD3	(Fujimoto et al., 2018)
D4PG	(Barth-maron et al., 2018)

6. Discussion

In the previous sections we have presented methods which: (1) do *not* build a utility model; (2) learn a utility model: (2a) in the policy parameter space Θ, (2b) in an arbitrary outcome space O, (2c) in the state–action space $\mathcal{X} \times \mathcal{U}$. In this section, we come back to the sample efficiency properties of these different methods. We do so by descending the tree of design choices depicted in Fig. 2.

6.1. Building a model or not

We have outlined that policy search methods which build a model of the utility function are generally more sample efficient than methods which do not. However, the reliance of the latter to SGD can make them less robust to local optima (Lehman et al., 2017) and it has been shown recently that methods which do not build a model of utility are still competitive in terms of final performance, due to their higher parallelization capability and distinguishing properties with respect to various gradient landscapes (Petroski Such et al., 2017; Salimans et al., 2017; Zhang et al., 2017).

6.2. Building a utility function model in the policy parameter space versus the state–action space.

Several elements speak in favor of the higher sample efficiency of learning a critic in the state–action space $\mathcal{X} \times \mathcal{U}$. First, it can give rise to more sample reuse than learning a model of the utility function in Θ. Second, learning from each step separately makes a better use of the information available from a rollout than learning from global episodes.

Furthermore, $\mathcal{X} \times \mathcal{U}$ may naturally exhibit a hierarchical structure – especially the state – which is not so obvious for Θ. As a consequence, methods that model utility in $\mathcal{X} \times \mathcal{U}$ may benefit from learning intermediate representations at different levels in the hierarchy, thus reducing the dimensionality of the policy search problem. Learning such intermediate and more compact representations is the focus of hierarchical reinforcement learning, a domain which has also been impacted by the emergence of deep RL (Bacon, Harb, & Precup, 2017; Kulkarni, Narasimhan, Saeedi, & Tenenbaum, 2016). Hierarchical reinforcement learning can also be performed off-line, which corresponds to the perspective of the DREAM project,[6] illustrated for instance in Zimmer and Doncieux (2017).

Finally, an important factor of sample efficiency is the size and structure of $\mathcal{X} \times \mathcal{U}$ with respect to Θ. In both respects, the emergence of deep RL methods using large neural networks as policy representation has changed the perspective. First, in deep RL, the size of Θ can become larger than that of $\mathcal{X} \times \mathcal{U}$, which speaks in favor of learning a critic. Second, deep neural networks seem to generally induce a smooth structure between Θ and $\mathcal{X} \times \mathcal{U}$, which facilitates learning. Finally, a utility function modeled in a larger space may suffer from fewer local minima, as more directions remain for improving the gradient (Kawaguchi, 2016).

The above conclusions might be mitigated by considering exploration. Indeed, in several surveys about policy search for robotics, policy parameter perturbation methods are considered superior to action perturbation methods (Deisenroth et al., 2013; Stulp & Sigaud, 2013). This analysis is backed-up with several mathematical arguments, but it might be true mostly when the space Θ is smaller than the space $\mathcal{X} \times \mathcal{U}$. Until recently, all deep RL methods were using action perturbation. But deep RL algorithms using policy parameter perturbation have recently been published, showing again that one can model the utility function in $\mathcal{X} \times \mathcal{U}$ while performing exploration in Θ (Fortunato et al., 2017; Plappert et al., 2017). Exploration is currently one of the hottest topics in deep RL and directed exploration methods presented in Section 4 may play a key role in this story, despite the lower data efficiency of their policy improvement mechanisms (Colas et al., 2018; Conti et al., 2017).

Message 5: There are more arguments for learning a utility model in $\mathcal{X} \times \mathcal{U}$ than in Θ, but this ultimately depends on the size of these spaces and the structure of their relationship.

6.3. Transient versus persistent critic

At first glance, having a persistent critic may seem superior to having a transient one, for three reasons. First, by avoiding to compute the critic again at each iteration, it is computationally more efficient. Second, immediate updates favor data efficiency because the policy is improved as soon as possible, which in turn helps generating better samples. Third, being based on bootstrap methods, they give rise to more sample reuse. However, these statements must be differentiated, as two factors (described below) must be taken into account.

6.3.1. Trading bias against variance

Estimating the utility of a policy in $\mathcal{X} \times \mathcal{U}$ is subject to a bias–variance compromise (Kearns & Singh, 2000). On the one hand, estimating the utility of a given policy through Monte Carlo sampling – as is generally done in transient critic approaches – is subject to a variance which grows with the length of the episodes. On the other hand, incrementally updating a persistent critic reduces variance, but may suffer from bias, resulting in potential sub-optimality, or even divergence. Instead of performing bootstrap updates of a critic over one step, one can do so over N steps. The larger N, the closer to Monte Carlo estimation, thus tuning N is a way of controlling the bias–variance compromise. For instance, while the transient critic TRPO algorithm is less sample efficient than actor–critic methods but more stable, often resulting in superior performance (Duan, Chen, Houthooft, Schulman, & Abbeel, 2016), its immediate successor, PPO, uses N steps return, resulting in a good compromise between both families (Schulman et al., 2017).

[6] http://www.robotsthatdream.eu/.

6.3.2. Off-policy versus on-policy updates

In on-policy methods such as SARSA, the samples used to learn the critic must come from the current policy, whereas in off-policy methods such as Q-LEARNING, they can come from any policy. In most transient critic methods, the samples are discarded from one iteration to the next and these methods are generally on-policy. By contrast, persistent critic methods using a replay buffer are generally off-policy.[7]

This on-policy versus off-policy distinction is related to the bias–variance compromise. Indeed, when learning a persistent critic incrementally, using off-policy updates is more flexible because the samples can come from any policy, but these off-policy updates introduce bias in the estimation of the critic. As a result, off-policy methods such as DDPG and NAF are more sample efficient because they use a replay buffer, but they are also more prone to sub-optimality and divergence. In that respect, a key contribution of ACER and Q-PROP is that they provide an off-policy, sample efficient update method which strongly controls the bias, resulting in more stability (Gu, Lillicrap, Ghahramani et al., 2016; Gu et al., 2017; Wang, Bapst et al., 2016; Wu et al., 2017). These aspects are currently the subject of intensive research but the resulting algorithms suffer from being more complex, with additional meta-parameters.

Message 6: Persistent critic methods are superior to transient critic methods in many respects, but the latter are more stable because they decorrelate the problem of estimating the utility function from the problem of descending its gradient, and they suffer from less bias.

7. Conclusion

In this paper, we have contrasted various approaches to policy search, from evolutionary methods which do not learn a model of the utility function to deep RL methods which do so in the state–action space.

In Stulp and Sigaud (2013), the authors have shown that policy search applied to robotics was shifting from actor–critic methods to evolutionary methods. Part of this shift was due to the use of open-loop DMPs (Ijspeert, Nakanishi, Hoffmann, Pastor, & Schaal, 2013) as a policy representation, which favors episode-based approaches, but another part resulted from the higher stability and efficiency of evolutionary methods by that time.

The emergence of deep RL methods has changed this perspective. It should be clear from this survey that, in the context of large problems where deep neural network representations are now the standard option, deep RL is generally more sample efficient than deep neuroevolution methods, as empirically confirmed in de Froissard de Broissia and Sigaud (2016) and Pourchot et al. (2018). The higher sample efficiency of deep RL methods, and particularly actor–critic architectures with a persistent critic, results from several mechanisms. They benefit from better approximation capability of non-linear critics and the incorporation of an adapted step size in SGD, they model the utility function in the state–action space, and they benefit from massive sample reuse by using a replay buffer. Using a target network has also mitigated the intrinsic instability of incrementally approximating a critic. However, it is important to acknowledge that incremental deep RL methods still suffer from significant instability.[8]

[7] The A3C algorithm is an incremental actor–critic method which does not use a replay buffer, and it is classified as on-policy.

[8] As outlined at https://www.alexirpan.com/2018/02/14/rl-hard.html.

7.1. Future directions

The field of policy search is currently the object of an intense race for increased performance, stability and sample efficiency. We now outline what we currently consider as promising research directions.

7.1.1. More analyses than competitions

Up to now, the main trend in the literature focuses on performance comparisons (Duan et al., 2016; Henderson et al., 2017; Islam, Henderson, Gomrokchi, & Precup, 2017; Petroski Such et al., 2017) showing that, despite their lower sample efficiency, methods which do not build a model of utility are still a competitive alternative in terms of final performance (Chrabaszcz, Loshchilov, & Hutter, 2018; Salimans et al., 2017). But stability and sample efficiency comparisons are missing and works analyzing the reasons why an algorithm performs better than another are only just emerging (Gangwani & Peng, 2017; Lehman et al., 2017; Zhang et al., 2017). By drawing an overview of the whole field and revealing some important factors behind sample efficiency, this paper was intended as a starting point towards broader and deeper analyses of the efficacy of various policy search methods.

7.1.2. More combinations than competitions

An important trend corresponds to the emergence of methods which combine algorithms from various families described above. As already noted in Section 4, directed exploration methods are often combined with evolutionary or deep RL methods (Colas et al., 2018; Conti et al., 2017). There is also an emerging trend combining evolutionary or population-based methods and deep RL methods (Jaderberg et al., 2017; Khadka & Tumer, 2018; Pourchot & Sigaud, 2018) which seem to be able to take the best of both worlds. We believe we are just at the beginning of such combinations and that this area has a lot of potential.

7.1.3. Beyond single policy improvement

Though we decided to keep *lifelong*, *continual* and *open-ended* learning outside the scope of this survey, we must mention that fast progress in policy improvement has favored an important tendency to address several tasks at the same time (Yang & Hospedales, 2014). This subfield is extremely active at the moment, with many works in multitask learning (Gangwani & Peng, 2018; Veeriah, Oh, & Singh, 2018; Vezhnevets et al., 2017), Hierarchical Reinforcement Learning (Levy, Platt, & Saenko, 2018; Nachum, Gu, Lee, & Levine, 2018) and meta reinforcement learning (Wang, Kurth-Nelson et al., 2016), to cite only a few.

Finally, because we focused on these elementary aspects, we have left aside the emerging topic of state representation learning (Jonschkowski & Brock, 2014; Lesort, Díaz-Rodríguez, Goudou, & Filliat, 2018; Raffin, Höfer, Jonschkowski, Brock, & Stulp, 2016) or using auxiliary tasks for improving deep RL (Jaderberg, Mnih, Czarnecki, Schaul, Leibo, Silver, et al., 2016; Riedmiller et al., 2018; Shelhamer, Mahmoudieh, Argus, & Darrell, 2016). The impact of these methods should be made clearer in the future.

7.2. Final word

As we have highlighted in the article, research in policy search and deep RL moves at a very high pace. Therefore, forecasting future trends, as we have done above, is risky, and even attempts to analyze the factors underlying current trends may be quickly outdated, but this also what makes this such an exciting research field.

Acknowledgments

Olivier Sigaud was supported by the European Commission, within the DREAM project, and has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement BXMATH[93] 640891. Freek Stulp was supported by the project "Reduced Complexity Models" (ZT-I-0010), funded by Helmholtz Association of German Research Centers. We thank David Filliat, Nicolas Perrin and Pierre-Yves Oudeyer for their feedback on this article.

References

Aha, D. W. (1997). Editorial. In *Lazy learning* (pp. 7–10). Springer.

Akimoto, Y., Nagata, Y., Ono, I., & Kobayashi, S. (2010). Bidirectional relation between CMA evolution strategies and natural evolution strategies. In *International conference on parallel problem solving from nature* (pp. 154–163). Springer.

Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, *57*, 469–483.

Arnold, L., Auger, A., Hansen, N., & Ollivier, Y. (2011). *Information-geometric optimization algorithms: A unifying picture via invariance principles*: *Tech. rep.*, INRIA Saclay.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866.

Azizzadenesheli, K., Brunskill, E., & Anandkumar, A. (2018). Efficient exploration through Bayesian deep Q-networks. arXiv preprint arXiv:1802.04412.

Back, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press.

Bacon, P. -L., Harb, J., & Precup, D. (2017). The option-critic architecture. In *AAAI* (pp. 1726–1734).

Baird, L. C. (1994). Reinforcement learning in continuous time: Advantage updating. In *Proceedings of the international conference on neural networks*. Orlando, FL.

Baranes, A., & Oudeyer, P. -Y. (2010). Intrinsically motivated goal exploration for active motor learning in robots: A case study. In *IEEE/RSJ international conference on intelligent robots and systems*. Taipei, Taiwan, Province Of China: IEEE.

Baranes, A., & Oudeyer, P. -Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, *61*(1), 49–73.

Baranes, A. F., Oudeyer, P. -Y., & Gottlieb, J. (2014). The effects of task difficulty, novelty and the size of the search space on intrinsically motivated exploration. *Frontiers in Neuroscience*, *8*, 317.

Barth-maron, G., Hoffman, M., Budden, D., Dabney, W., Horgan, D., TB, D., et al. (2018). Distributional policy gradient. In *ICLR* (pp. 1–16).

Baxter, J., & Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, *15*, 319–350.

Bellemare, M. G., Dabney, W., & Munos, R. (2017). A distributional perspective on reinforcement learning. arXiv preprint arXiv:1707.06887.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., & Lee, M. (2007). Incremental natural actor-critic algorithms. In *Advances in neural information processing systems*. MIT Press.

Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade* (pp. 421–436). Springer.

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. arXiv preprint arXiv:1012.2599.

Calandra, R., Gopalan, N., Seyfarth, A., Peters, J., & Deisenroth, M. P. (2014). Bayesian Gait optimization for bipedal locomotion. In *International conference on learning and intelligent optimization* (pp. 274–290). Springer.

Chatzilygeroudis, K., Rama, R., Kaushik, R., Goepp, D., Vassiliades, V., & Mouret, J. -B. (2017). Black-box data-efficient policy search for robotics. arXiv preprint arXiv:1703.07261.

Chrabaszcz, P., Loshchilov, I., & Hutter, F. (2018). Back to basics: Benchmarking Canonical evolution strategies for playing atari. arXiv preprint arXiv:1802.08842.

Colas, C., Sigaud, O., & Oudeyer, P. -Y. (2018). GEP-PG: Decoupling exploration and exploitation in deep reinforcement learning algorithms. arXiv preprint arXiv:1802.05054.

Conti, E., Madhavan, V., Such, F. P., Lehman, J., Stanley, K. O., & Clune, J. (2017). Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. arXiv preprint arXiv:1712.06560.

Cuccu, G., & Gomez, F. (2011). When novelty is not enough. In *European conference on the applications of evolutionary computation* (pp. 234–243). Springer.

Cully, A., Clune, J., Tarapore, D., & Mouret, J. -B. (2015). Robots that can adapt like animals. *Nature*, *521*(7553), 503–507.

Cully, A., & Demiris, Y. (2017). Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*.

de Froissard de Broissia, A., & Sigaud, O. (2016). Actor-critic versus direct policy search: a comparison based on sample complexity. arXiv preprint arXiv:1606.09152.

Deisenroth, M. P., Neumann, G., Peters, J., et al. (2013). A survey on policy search for robotics. *Foundations and Trends® in Robotics*, *2*(1–2), 1–142.

Deisenroth, M., & Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International conference on machine learning* (pp. 465–472).

Doncieux, S., & Mouret, J. -B. (2014). Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, *7*(2), 71–93.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., & Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. arXiv preprint arXiv:1604.06778.

Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, *1*(1), 47–62.

Forestier, S., Mollard, Y., & Oudeyer, P. -Y. (2017). Intrinsically motivated goal exploration processes with automatic curriculum learning. arXiv preprint arXiv:1708.02190.

Forestier, S., & Oudeyer, P. -Y. (2016). Overlapping waves in tool use development: a curiosity-driven computational model. In *Joint ieee international conference on development and learning and epigenetic robotics (ICDL-EpiRob)*.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., et al. (2017). Noisy networks for exploration. arXiv preprint arXiv:1706.10295.

Fujimoto, S., van Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. arXiv preprint arXiv:1802.09477.

Gangwani, T., & Peng, J. (2017). Genetic policy optimization. arXiv preprint arXiv:1711.01012.

Gangwani, T., & Peng, J. (2018). Policy optimization by genetic distillation. In *ICLR 2018*.

Gill, P. E., Murray, W., & Wright, M. H. (1981). *Practical optimization*. Academic Press.

Glasmachers, T., Schaul, T., Yi, S., Wierstra, D., & Schmidhuber, J. (2010). Exponential natural evolution strategies. In *Proceedings of the 12th annual conference on genetic and evolutionary computation* (pp. 393–400). ACM.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison Wesley.

Grondman, I., Busoniu, L., Lopes, G. A., & Babuska, R. (2012). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*(6), 1291–1307.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., & Levine, S. (2016). Q-prop: Sample-efficient policy gradient with an off-policy critic. arXiv preprint arXiv:1611.02247.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., Schölkopf, B., & Levine, S. (2017). Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. arXiv preprint arXiv:1706.00387.

Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016). Continuous deep Q-learning with model-based acceleration. arXiv preprint arXiv:1603.00748.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290.

Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, *9*(2), 159–195.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2017). Deep reinforcement learning that matters. arXiv preprint arXiv:1709.06560.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., et al. (2017). Rainbow: Combining improvements in deep reinforcement learning. arXiv preprint arXiv:1710.02298.

Hwangbo, J., Gehring, C., Sommer, H., Siegwart, R., & Buchli, J. (2014). ROCK*: Efficient black-box optimization for policy learning. In *IEEE-RAS international conference on humanoid robots* (pp. 535–540). IEEE.

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., & Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. *Neural Computation*, *25*(2), 328–373.

Islam, R., Henderson, P., Gomrokchi, M., & Precup, D. (2017), Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control, In *Proceedings of the ICML 2017 workshop on reproducibility in machine learning*.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., et al. (2017). Population based training of neural networks. arXiv preprint arXiv:1711.09846.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., et al. (2016). Reinforcement learning with unsupervised auxiliary tasks. arXiv preprint arXiv:1611.05397.

Jonschkowski, R., & Brock, O. (2014). State representation learning in robotics: Using prior knowledge about physical interaction. In *Proceedings of robotics, science and systems*.

Kawaguchi, K. (2016). Deep learning without poor local minima. In *Advances in neural information processing systems* (pp. 586–594).

Kearns, M. J., & Singh, S. P. (2000). Bias-variance error bounds for temporal difference updates. In *COLT* (pp. 142–147).

Khadka, S., & Tumer, K. (2018). Evolutionary reinforcement learning. arXiv preprint arXiv:1805.07917.

Kober, J., Bagnell, J., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, *32*(11), 1238–1274.

Kober, J., & Peters, J. (2009). Learning motor primitives for robotics. In *IEEE international conference on robotics and automation* (pp. 2112–2118). IEEE.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Kulkarni, T. D., Narasimhan, K. R., Saeedi, A., & Tenenbaum, J. B. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. arXiv preprint arXiv:1604.06057.

Larrañaga, P., & Lozano, J. A. (2001). *Estimation of distribution algorithms: A new tool for evolutionary computation*: *Vol. 2*, Springer Science & Business Media.

Laversanne-Finot, A., Péré, A., & Oudeyer, P. -Y. (2018). Curiosity driven exploration of learned disentangled goal spaces. arXiv preprint arXiv:1807.01521.

Lehman, J., Chen, J., Clune, J., & Stanley, K. O. (2017). Es is more than just a traditional finite-difference approximator. arXiv preprint arXiv:1712.06568.

Lehman, J., & Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, *19*(2), 189–223.

Lesort, T., Díaz-Rodríguez, N., Goudou, J. -F., & Filliat, D. (2018). State representation learning for control: An overview. arXiv preprint arXiv:1802.04181.

Levine, S., & Koltun, V. (2013). Guided policy search. In *Proceedings of the 30th international conference on machine learning* (pp. 1–9).

Levy, A., Platt, R., & Saenko, K. (2018). Hierarchical reinforcement learning with hindsight. arXiv preprint arXiv:1805.08180.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

Lizotte, D. J., Wang, T., Bowling, M. H., & Schuurmans, D. (2007). Automatic gait optimization with Gaussian process regression. In *IJCAI, vol. 7* (pp. 944–949).

Mania, H., Guy, A., & Recht, B. (2018). Simple random search provides a competitive approach to reinforcement learning. arXiv preprint arXiv:1803.07055.

Martinez-Cantin, R., Tee, K., & McCourt, M. (2017). Policy search using robust Bayesian Optimization. In *Neural information processing systems (NIPS) workshop on acting and interacting in the real world: Challenges in robot learning*.

Metzen, J. H., Fabisch, A., & Hansen, J. (2015). Bayesian optimization for contextual policy search. In *Proceedings of the second machine learning in planning and control of robot motion workshop*. Hamburg.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., et al. (2016). Asynchronous methods for deep reinforcement learning. arXiv preprint arXiv:1602.01783.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Montgomery, W. H., & Levine, S. (2016). Guided policy search via approximate mirror descent. In *Advances in neural information processing systems* (pp. 4008–4016).

Nachum, O., Gu, S., Lee, H., & Levine, S. (2018). Data-efficient hierarchical reinforcement learning. arXiv preprint arXiv:1805.08296.

Najar, A., Sigaud, O., & Chetouani, M. (2016). Training a robot with evaluative feedback and unlabeled guidance signals. In *25th IEEE international symposium on robot and human interactive communication* (pp. 261–266). IEEE.

Neumann, G. (2011). Variational inference for policy search in changing situations. In *Proceedings of the 28th international conference on machine learning* (pp. 817–824).

O'Donoghue, B., Munos, R., Kavukcuoglu, K., & Mnih, V. (2016). Combining policy gradient and Q-learning. arXiv preprint arXiv:1611.01626.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999). Boa: the bayesian optimization algorithm. In *Proceedings of the 1st annual conference on genetic and evolutionary computation. vol. 1* (pp. 525–532). Morgan Kaufmann Publishers Inc.

Pere, A., Forestier, S., Sigaud, O., & Oudeyer, P. -Y. (2018). Unsupervised learning of goal spaces for intrinsically motivated goal exploration. In *International conference on learning representations*, arXiv preprint arXiv:1803.00781.

Peters, J., Mülling, K., & Altun, Y. (2010). Relative entropy policy search. In *AAAI* (pp. 1607–1612). Atlanta.

Peters, J., & Schaal, S. (2008a). Natural actor-critic. *Neurocomputing*, *71*(7–9), 1180–1190.

Peters, J., & Schaal, S. (2008b). Reinforcement learning of motor skills with policy gradients. *Neural networks*, *21*(4), 682–697.

Petroski Such, F., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., & Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv preprint arXiv:1712.06567.

Pierrot, T., Perrin, N., & Sigaud, O. (2018). First-order and second-order variants of the gradient descent: a unified framework. arXiv preprint arXiv:1810.08102.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., et al. (2017). Parameter space noise for exploration. arXiv preprint arXiv:1706.01905.

Pourchot, A., Perrin, N., & Sigaud, O. (2018). Importance mixing: Improving sample reuse in evolutionary policy search methods. arXiv preprint arXiv:1808.05832.

Pourchot, A., & Sigaud, O. (2018). CEM-RL: Combining evolutionary and gradient-based methods for policy search. arXiv preprint arXiv:1810.01222.

Pugh, J. K., Soros, L., Szerlip, P. A., & Stanley, K. O. (2015). Confronting the challenge of quality diversity. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation* (pp. 967–974). ACM.

Raffin, A., Höfer, S., Jonschkowski, R., Brock, O., & Stulp, F. (2016). *Unsupervised learning of state representations for multiple tasks*: *Tech. rep.*, University of Berlin.

Rastrigin, L. (1963). The convergence of the random search method in the extremal control of a many parameter system. *Automation and Remote Control*, *24*(10), 1337–1342.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Van de Wiele, T., et al. (2018). Learning by playing-solving sparse reward tasks from scratch. arXiv preprint arXiv:1802.10567.

Riedmiller, M., Peters, J., & Schaal, S. (2008). Evaluation of Policy Gradient Methods and Variants on the Cart-Pole Benchmark. In *IEEE International symposium on approximate dynamic programming and reinforcement learning*.

Rubinstein, R., & Kroese, D. (2004). *The cross-entropy method: A unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. Springer-Verlag.

Salimans, T., Ho, J., Chen, X., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. arXiv preprint arXiv:1511.05952.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). Trust region policy optimization. *CoRR*, abs/1502.05477.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., & Schmidhuber, J. (2010). Parameter-exploring policy gradients. *Neural Networks*, *23*(4), 551–559.

Shelhamer, E., Mahmoudieh, P., Argus, M., & Darrell, T. (2016). Loss is its own reward: Self-supervision for reinforcement learning. arXiv preprint arXiv:1612.07307.

Sigaud, O., & Buffet, O. (2010). *Markov decision processes in artificial intelligence*. iSTE - Wiley.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 30th international conference in machine learning*.

Stanley, K. O., & Miikkulainen, R. (2002). Efficient evolution of neural network topologies. In *Evolutionary computation, 2002. CEC'02. Proceedings of the 2002 congress on, vol. 2* (pp. 1757–1762). IEEE.

Stulp, F., & Sigaud, O. (2012a). Path integral policy improvement with covariance matrix adaptation. In *Proceedings of the 29th international conference on machine learning* (pp. 1–8). Edinburgh, Scotland.

Stulp, F., & Sigaud, O. (2012b). *Policy improvement methods: Between black-box optimization and episodic reinforcement learning*: *Tech. rep.*, hal-00738463.

Stulp, F., & Sigaud, O. (2013). Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn Journal of Behavioral Robotics*, *4*(1), 49–61.

Stulp, F., & Sigaud, O. (2015). Many regression algorithms, one unified model: A review. *Neural Networks*, *69*, 60–79.

Sun, Y., Wierstra, D., Schaul, T., & Schmidhuber, J. (2009). Efficient natural evolution strategies. In *Proceedings of the 11th annual conference on genetic and evolutionary computation* (pp. 539–546). ACM.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

Tang, Y., & Kucukelbir, A. (2017). Variational deep Q network. arXiv preprint arXiv:1711.11225.

Theodorou, E., Buchli, J., & Schaal, S. (2010). A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research (JMLR)*, *11*, 3137–3181.

Thrun, S., & Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and autonomous systems*, *15*(1–2), 25–46.

Togelius, J., Schaul, T., Wierstra, D., Igel, C., Gomez, F., & Schmidhuber, J. (2009). Ontogenetic and phylogenetic reinforcement learning. *Künstliche Intelligenz*, *23*(3), 30–33.

Veeriah, V., Oh, J., & Singh, S. (2018). Many-goals reinforcement learning. arXiv preprint arXiv:1806.09605.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., et al. (2017). Feudal networks for hierarchical reinforcement learning. arXiv preprint arXiv:1703.01161.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., et al. (2016). Sample efficient actor-critic with experience replay. arXiv preprint arXiv:1611.01224.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., et al. (2016). Learning to reinforcement learn. arXiv preprint arXiv:1611.05763.

Wierstra, D., Schaul, T., Peters, J., & Schmidhuber, J. (2008). Natural evolution strategies. In *IEEE congress on evolutionary computation* (pp. 3381–3387). IEEE.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3–4), 229–256.

Williams, J. K., & Singh, S. P. (1998). Experimental results on learning stochastic memoryless policies for partially observable markov decision processes. In *NIPS* (pp. 1073–1080).

Wilson, A., Fern, A., & Tadepalli, P. (2014). Using trajectory data to improve Bayesian optimization for reinforcement learning. *Journal of Machine Learning Research (JMLR)*, *15*(1), 253–282.

Wu, Y., Mansimov, E., Liao, S., Grosse, R., & Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. arXiv preprint arXiv:1708.05144.

Yang, Y., & Hospedales, T. M. (2014). A unified perspective on multi-domain and multi-task learning. arXiv preprint arXiv:1412.7489.

Zhang, X., Clune, J., & Stanley, K. O. (2017). On the relationship between the openAI evolution strategy and stochastic gradient descent. arXiv preprint arXiv:1712.06564.

Zimmer, M., & Doncieux, S. (2017). Bootstrapping q-learning for robotics from neuro-evolution results. *IEEE Transactions on Cognitive and Developmental Systems*.