Effective Reinforcement Learning through Evolutionary Surrogate-Assisted Prescription

Olivier Francon[1], Santiago Gonzalez[1,2], Babak Hodjat[1], Elliot Meyerson[1],
Risto Miikkulainen[1,2], Xin Qiu[1], and Hormoz Shahrzad[1]
[1]Cognizant Technology Solutions and [2]The University of Texas at Austin

ABSTRACT

There is now significant historical data available on decision making in organizations, consisting of the decision problem, what decisions were made, and how desirable the outcomes were. Using this data, it is possible to learn a surrogate model, and with that model, evolve a decision strategy that optimizes the outcomes. This paper introduces a general such approach, called Evolutionary Surrogate-Assisted Prescription, or ESP. The surrogate is, for example, a random forest or a neural network trained with gradient descent, and the strategy is a neural network that is evolved to maximize the predictions of the surrogate model. ESP is further extended in this paper to sequential decision-making tasks, which makes it possible to evaluate the framework in reinforcement learning (RL) benchmarks. Because the majority of evaluations are done on the surrogate, ESP is more sample efficient, has lower variance, and lower regret than standard RL approaches. Surprisingly, its solutions are also better because both the surrogate and the strategy network regularize the decision making behavior. ESP thus forms a promising foundation to decision optimization in real-world problems.

KEYWORDS

Reinforcement Learning, Decision Making, Surrogate-Assisted Evolution, Genetic Algorithms, Neural Networks

1 INTRODUCTION

Many organizations in business, government, education, and healthcare now collect significant data about their operations. Such data is transforming decision making in organizations: It is now possible to use machine learning techniques to build predictive models of behaviors of customers, consumers, students, and competitors,
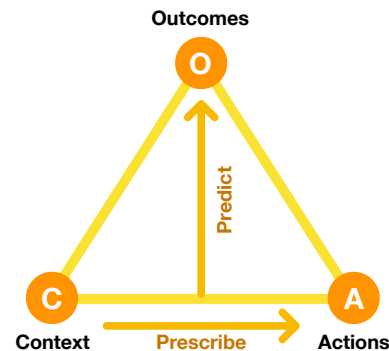
Figure 1: *Elements of ESP.* **A Predictor is trained with historical data on how given actions in given contexts led to specific outcomes. The Predictor can be any machine learning model trained with supervised methods, such as a random forest or a neural network. The Predictor is then used as a surrogate in order to evolve a Prescriptor, i.e. a neural network implementing a decision policy that results in the best possible outcomes. The majority of evaluations are done on the surrogate, making the process highly sample-efficient and robust, and leading to decision policies that are regularized and therefore generalize well.**

and, in principle, make better decisions, i.e. those that lead to more desirable outcomes. However, while prediction is necessary, it is only part of the process. Predictive models do not specify what the optimal decisions actually are. To find a good decision strategy, different approaches are needed.

The main challenge is that optimal strategies are not known, so standard gradient-based machine learning approaches cannot be used. The domains are only partially observable, and decision variables and outcomes often interact nonlinearly: For instance, allocating marketing resources to multiple channels may have a nonlinear cumulative effect, or nutrition and exercise may interact to leverage or undermine the effect of medication in treating an illness [9, 32]. Such interactions make it difficult to utilize linear programming and other traditional optimization approaches from operations research.

Instead, good decision strategies need to be found using search, i.e. by generating strategies, evaluating them, and generating new, hopefully better strategies based on the outcomes. In many domains such search cannot be done in the domain itself: For instance, testing an ineffective marketing strategy or medical treatment could be prohibitively costly; evaluating an engineering design through simulation, or a behavioral strategy in game playing could require

a prohibitive amount of computation time. However, given that historical data about past decisions and their outcomes exist, it is possible to do the search using a predictive model as a surrogate to evaluate them. Only once good decision strategies have been found using the surrogate, they are tested in the real world.

Even with the surrogate, the problem of finding effective decision strategies is still challenging. Nonlinear interactions may result in deceptive search landscapes, where progress towards good solutions cannot be made through incremental improvement: Discovering them requires large, simultaneous changes to multiple variables. Decision strategies often require balancing multiple objectives, such as performance and cost, and in practice, generating a number of different trade-offs between them is needed. Consequently, search methods such as reinforcement learning (RL), where a solution is gradually improved through local exploration, do not lend themselves well to searching solution strategies either. Further, the number of variables can be very large, e.g. thousands or even millions as in some manufacturing and logistics problems [6], making methods such as Kriging and Bayesian optimization [5, 46] ineffective. Moreover, the solution is not a single point but a strategy, i.e. a function that maps input situations to optimal decisions, exacerbating the scale-up problem further.

Keeping in mind the above challenges, an approach is developed in this paper for *Evolutionary Surrogate-Assisted Prescription* (ESP; Figure 1), i.e. for discovering effective solution strategies using evolutionary optimization. With a population-based search method, it is possible to navigate deceptive, high-dimensional landscapes, and discover trade-offs across multiple objectives [27]. The strategy is expressed as a neural network, making it possible to use state-of-the-art neuroevolution techniques to optimize it. Evaluations of the neural network candidates are done using a predictive model, trained with historical data on past decisions and their outcomes.

Elements of the ESP approach were already found effective in challenging real-world applications. In an autosegmentation version of Ascend by Evolv, a commercial product for optimizing designs of web pages [28], a neural network was used to map user descriptions to most effective web-page designs. In CyberAg, effective growth recipes for basil were found through search with a surrogate model trained with outcomes of past recipes [23]. In both cases, evolutionary search found designs that were more effective than human designs, even surprising and unlikely to be found by humans. In ESP, these elements of strategy search and surrogate modeling are combined into a general approach for decision strategy optimization. ESP is implemented as part of LEAF, i.e. Cognizant's Evolutionary AI platform, and is currently being applied to a number of business decision optimization problems.

The goal of this paper is to introduce the ESP approach in general, and to extend it further into decision strategies that consist of sequences of decisions. This extension makes it possible to evaluate ESP against other methods in RL domains. Conversely, ESP is used to formalize RL as surrogate-assisted, population based search. This approach is particularly compelling in domains where real-world evaluations are costly. ESP improves upon traditional RL in several ways: It converges faster given the same number of episodes, indicating better sample-efficiency; it has lower variance for best policy performance, indicating better reliability of delivered solutions; and it has lower regret, indicating lower costs and better safety during training. Surprisingly, optimizing against the surrogate also has a regularization effect: the solutions are sometimes more general and thus perform better than solutions discovered in the domain itself. Further, ESP brings the advantages of population-based search outlined above to RL, i.e. enhanced exploration, multiobjectivity, and scale-up to high-dimensional search spaces.

The ESP approach is evaluated in this paper in various RL benchmarks. First its behavior is visualized in a synthetic domain, illustrating how the Predictor and Prescriptor learn together to discover optimal decisions. Direct evolution (DE) is compared to evolution with the surrogate, to demonstrate how the approach minimizes the need for evaluations in the real world. ESP is then compared with standard RL approaches, demonstrating better sample efficiency, reliability, and lower cost. The experiments also demonstrate regularization through the surrogate, as well as ability to utilize different kinds of Predictor models (e.g. random forests and neural networks) to best fit the data. ESP thus forms a promising evolutionary-optimization-based approach to sequential decision-making tasks.

2 RELATED WORK

Traditional model-based RL aims to build a transition model, embodying the system's dynamics, that estimates the system's next state in time, given current state and actions. The transition model, which is learned as part of the RL process, allows for effective action selection to take place [13]. These models allow agents to leverage predictions of the future state of their environment [54]. However, model-based RL usually requires a prohibitive amount of data for building a reliable transition model while also training an agent. Even simple systems can require tens- to hundreds-of-thousands of samples [41]. While techniques such as PILCO [7] have been developed to specifically address sample efficiency in model-based RL, they can be computationally intractable for all but the lowest dimensional domains [51].

As RL has been applied to increasingly complex tasks with real-world costs, sample efficiency has become a crucial issue. Model-free RL has emerged as an important alternative in such tasks because they sample the domain without a transition model. Their performance and efficiency thus depend on their sampling and reward estimation methods. As a representative model-free off-policy method, Deep Q-Networks (DQN) [30] solves the sample efficiency issue by modeling future rewards using action values, also known as Q values. The Q-network is learned based on a replay buffer that collects training data from real-world interactions. Advanced techniques such as double Q-learning [14] and dueling network architectures [52] makes DQN competitive in challenging problems.

In terms of on-policy model-free techniques, policy gradient approaches (sometimes referred to as deep RL) leverage developments in deep neural networks to provide a general RL solution. Asynchronous Advantage Actor-Critic (A3C) [29] in particular builds policy critics through an advantage function, which considers both action and state values. Proximal Policy Optimization (PPO) [44] further makes actor-critic methods more robust with a clipped surrogate objective. Unfortunately, policy gradient techniques are typically

quite sensitive to hyperparameter settings, and require overwhelming numbers of samples. One reason is the need to train a policy neural network. While expensive, policy gradient methods have had success in simulated 3D locomotion [15, 43], Atari video games [30], and, famously, Go [45].

ESP will be compared to both DQN and PPO methods in this paper. Compared to ESP, these existing RL approaches have several limitations: First, their performance during real-world interactions cannot be guaranteed, leading to safety concerns [37]. In ESP, only elite agents selected via the surrogate model are evaluated on the real world, significantly improving safety. Second, the quality of the best-recognized policy is unreliable because it has not been sufficiently evaluated during learning. ESP solves this issue by evaluating all elite policies in the real world for multiple episodes. Third, existing RL methods rely heavily on deep neural networks. In contrast, ESP treats the Predictor as a black box, allowing high flexibility in model choices, including simpler models such as random forests that are sufficient in many cases.

Evolutionary approaches have been used in RL in several ways. For instance, in evolved policy gradients [17], the loss function against which an agent's policy is optimized is evolved. The policy loss is not represented symbolically, but rather as a neural network that convolves over a temporal sequence of context vectors; the parameters to this neural network are optimized using evolutionary strategies. In reward function search [33], the task is framed as a genetic programming problem, leveraging PushGP [47].

More significantly, entire policies can be discovered directly with many evolutionary techniques including Genetic Algorithms, Natural Evolutionary Strategies, and Neuroevolution [1, 11, 39, 48, 49, 55]. Such direct policy evolution can only leverage samples from the real-world, which is costly and risky since many evaluations have to take place in low-performing areas of the policy space. This issue in evolutionary optimization led to the development of surrogate-assisted evolution techniques [12, 18, 22]. Surrogate methods have been applied to a wide range of domains, ranging from turbomachinery blade optimization [35] to protein design [42]. ESP aims to build upon this idea, by relegating the majority of policy evaluations to a flexible surrogate model, allowing for a wider variety of contexts to be used to evolve better policies.

3 THE ESP APPROACH

The goal of the ESP approach is to find a decision policy that optimizes a set of outcomes (Figure 1). Given a set of possible contexts (or states) $\mathbb{C}$ and possible actions $\mathbb{A}$, a decision policy D returns a set of actions A to be performed in each context C:

$$D(C) = A , \tag{1}$$

where $C \in \mathbb{C}$ and $A \in \mathbb{A}$. For each such (C, A) pair there is a set of outcomes $O(C, A)$, i.e. the consequences of carrying out decision A in context C. For instance, the context might be a description of a patient, actions might be medications, and outcomes might be health, side effects, and costs. In the following, higher values of O are assumed to be better for simplicity.

In ESP, two models are employed: a Predictor P_d, and a Prescriptor P_s. The Predictor takes, as its input, context information, as well as actions performed in that context. The output of the Predictor is the resulting outcomes when the given actions are applied in the
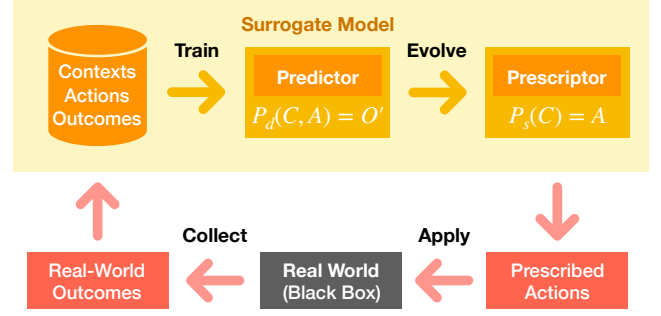

Figure 2: *The ESP Outer Loop*. **The Predictor can be trained gradually at the same time as the Prescriptor is evolved, using the Prescriptor to drive exploration. That is, the user can decide to apply the Prescriptor's outputs to the real world, observe the outcomes, and aggregate them into the Predictor's training set.**

given context. The Predictor is therefore defined as

$$P_d(C, A) = O', \tag{2}$$

such that $\sum_j L(O_j, O'_j)$ across all dimensions j of O is minimized. The function L can be any of the usual loss functions used in machine learning, such as cross-entropy or mean-squared-error, and the model P_d itself can be any supervised machine learning model such as a neural network or a random forest.

The Prescriptor takes a given context as input, and outputs a set of actions:

$$P_s(C) = A , \tag{3}$$

such that $\sum_{i,j} O'_j(C_i, A_i)$ over all possible contexts i is maximized. It thus approximates the optimal decision policy for the problem. Note that the optimal actions A are not known, and must therefore be found through search.

The ESP algorithm then operates as an outer loop that constructs the Predictor and Prescriptor models (Figure 2):

(1) Train a Predictor based on historical training data;
(2) Evolve Prescriptors with the Predictor as the surrogate;
(3) Apply the best Prescriptor in the real world;
(4) Collect the new data and add to the training set;
(5) Repeat until convergence.

As usual in evolutionary search, the process terminates when a satisfactory level of outcomes has been reached, or no more progress can be made. Note that in Step 1, if no historical decision data exists initially, a random Predictor can be used. Also note that not all data needs to be accumulated for training each iteration. In domains where the underlying relationships between variables might change over time, it might be advisable to selectively ignore samples from the older data as more data is added to the training set in Step 4. It is thus possible to bias the training set towards more recent experiences.

Building the Predictor model is straightforward given a (C, A, O) dataset. The choice of algorithm depends on the domain, i.e. how much data there is, whether it is continuous or discrete, structured or unstructured. Random forests and neural networks will be demonstrated in this role in this paper. The Prescriptor model,
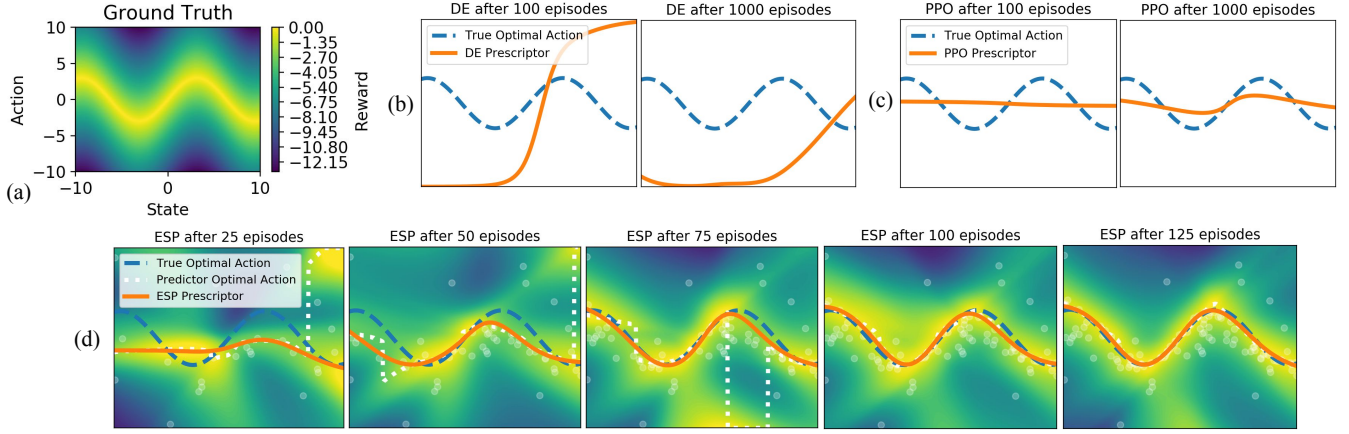
Figure 3: *Visualizing ESP Behavior.* (*a*) **True reward for every state-action pair;** (*b−c*) **After 1000 episodes, the top Prescriptors for direct evolution (DE) and PPO are still far from optimal;** (*d*) **ESP Prescriptor (orange) and Predictor (background) for several iterations. The translucent circles indicate the state-action pairs sampled so far, i.e., the samples on which the Predictor is trained. By 125 episodes, ESP has converged around the optimal Prescriptor, and the ESP Predictor has converged in the neighborhood of this optimum, showing how ESP can leverage Predictors over time to find good actions quickly. Note that the Prescriptor does not exactly match the actions the Predictor would suggest as optimal: the Prescriptor regularizes the Predictor's overfitting by implicitly ensembling the Predictors evolved against over time. For a full video of the algorithms, see http://evolution.ml/demos/espvisualization.**

in contrast, is built using neuroevolution in ESP: Neural networks because they can express complex nonlinear mappings naturally, and evolution because it is an efficient way to discover such mappings [48], and naturally suited to optimize multiple objectives [4, 10]. Because it is evolved with the Predictor, the Prescriptor is not restricted by a finite training dataset, or limited opportunities to evaluate in the real world. Instead, the Predictor serves as a fitness function, and it can be queried frequently and efficiently. In a multiobjective setting, ESP produces multiple Prescriptors, selected from the Pareto front of the multiobjective neuroevolution run.

Applying the ESP framework to RL problems involves extending the contexts and actions to sequences. The Prescriptor can be seen as an RL agent, taking the current context as input, and deciding what actions to perform in each time step. The output of the Predictor, O', can be seen as the reward vector for that step, i.e. as Q values [53] (with a given discount factor, such as 0.9, as in the experiments below). Evolution thus aims to maximize the predicted reward, or minimize the regret, throughout the sequence.

The outer loop of ESP changes slightly because in RL there is no dataset to train the Predictor; instead, the data needs to be generated by applying the current Prescriptors to the domain. An elite set of several good Prescriptors are used in this role to create a more diverse training set. The initial training set is created randomly. The loop now is:

(1) Apply the elite Prescriptors in the actual domain;
(2) Collect Q values for each time step for each Prescriptor;
(3) Train a Predictor based on data collected in Step 2;
(4) Evolve Prescriptors with the Predictor as the surrogate;
(5) Repeat until convergence.

The evolution of Prescriptors continues in each iteration of this loop from where it left off in previous iteration. In addition, the

system keeps track of the best Prescriptor so far, as evaluated in the actual domain, and makes sure it stays in the parent population during evolution. This process discovers good Prescriptor agents efficiently, as will be described in the experiments that follow.

4 EXPERIMENTS

ESP was evaluated in three domains: Function approximation, where its behavior could be visualized concretely; Cart-pole control [2] where its performance could be compared to standard RL methods in a standard RL benchmark task; and Flappy Bird, where the regularization effect of the surrogate could be demonstrated most clearly.

The neuroevolution algorithm for discovering Prescriptors evolves weights of neural networks with fixed topologies. Unless otherwise specified, all experiments use the following default setup for evolution: candidates have a single hidden layer with bias and tanh activation; the initial population uses orthogonal initialization of layer weights with a mean of 0 and a standard deviation of 1 [40]; the population size is 100; the top 10% of the population is carried over as elites; parents are selected by tournament selection of the top 20% of candidates; recombination is performed by uniform crossover at the weight-level; there is a 10% probability of multiplying each weight by a mutation factor, where mutation factors are drawn from $\mathcal{N}(1, 0.1)$.

4.1 Visualizing ESP Behavior

This section demonstrates the behavior of ESP in a synthetic function approximation domain where its behavior can be visualized. The domain also allows comparing ESP to direct evolution in the domain, as well as to PPO, and visualizing their differences.

Problem Description. The domain has a one-dimensional context C and a one-dimensional action A, with outcome O given by the function $O = -|A - 3\sin\frac{C}{2}|$. The optimal action for each context lies on a periodic curve, which captures complexity that can arise from periodic variables such as time of day or time of year. The outcome of each action decreases linearly as the action moves away from the optimal action. Episodes in this domain consist of single action in $[-10, 10]$, which is taken in a context drawn uniformly over $[-10, 10]$. The full domain is shown in Figure 3(a).

Algorithm Setup. ESP begins by taking ten random actions. Thereafter, every iteration, ESP trains a Predictor neural network with two hidden layers of size 64 and tanh activation for 2000 epochs using the Adam optimizer [24] with default parameters to minimize MSE, and evolves Prescriptor neural networks for 20 generations against this Predictor. Then, the top Prescriptor is run in the real domain for a single episode. Prescriptors have a single hidden layer of size 32 with tanh activation; default parameters are used for evolution.

Direct Evolution (DE) was run as a baseline comparison for ESP. It consists of the exact same evolution process, except that it is run directly against the real function instead of the Predictor. That is, in each generation, all 100 candidates are evaluated on one episode from the real function.

PPO was run as an RL comparison, since it is a state-of-the-art RL approach for continuous action spaces [44]. During each iteration it was run for ten episodes, since this setting was found to perform best during hyperparameter search. PPO defaults[1] were used for the remaining hyperparameters.

Ten independent runs were performed for each algorithm. The returned policy at any time for DE and ESP is the candidate with the highest fitness in the population; for PPO it is the learned policy run without stochastic exploration.

Qualitative Results. Snapshots of the convergence behavior for each method are shown in Figures 3(b-d). After 1000 episodes, neither DE nor PPO converged near the optimal solution. On the other hand, ESP discovered the periodic nature of the problem within 50 episodes, and converged almost exactly to the optimal within 125 episodes.

The Predictor's predicted reward for each state-action pair is shown using the background colors in each snapshot of ESP. The rapid convergence of the Predictor highlights the sample efficiency of ESP, due to aggressive use of historical data (shown as translucent circles). Note, however, that the Predictor does not converge to the ground truth over the entire domain; it does so just in the neighborhood of the optimal Prescriptor. Thus, ESP avoids excessive costly exploration of low-quality actions once the structure of optimal actions has become clear.

Note also that the Prescriptor does not follow the optimal action suggested by the Predictor at every iteration exactly. Since it maps states directly to actions, the Prescriptor provides a smoothing regularization in action space that can overcome Predictor overfitting. Also, since the top ESP Prescriptors must survive across many different Predictors over time, ESP benefits from an implicit temporal ensembling, which further improves regularization.
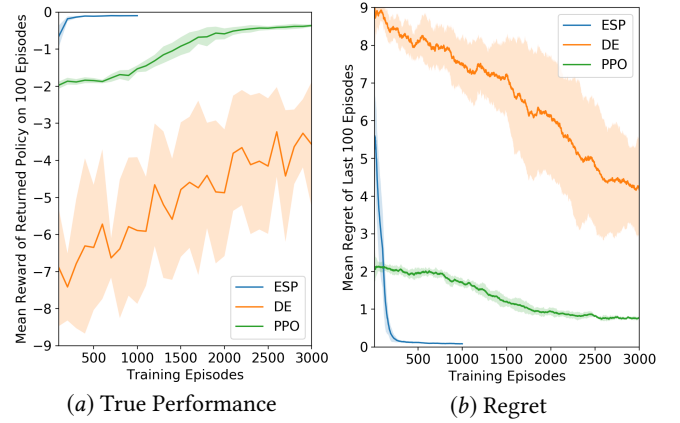
[1]https://stable-baselines.readthedocs.io/en/master/modules/ppo2.html

(*a*) True Performance   (*b*) Regret

Figure 4: *Performance in the Function Approximation Domain.* **The horizontal axis indicates the total number of real-world episodes used by training and the vertical axis the performance at that point. Ten independent runs were performed for each method. Solid lines represent the mean over 10 runs, and colored areas show the corresponding standard deviation.** (*a*) **The true performance of the returned best agent converges substantially faster with ESP.** (*b*) **ESP also operates with much lower regret than DE or PPO, converging to very low regret behavior orders-of-magnitude faster than the other approaches (the plot shows the moving average regret over the last 100 training episodes). The standard deviation of ESP is small in both metrics, attesting to the reliability of the method.**

Quantitative Results. The numerical performance results in Figure 4 confirm the substantial advantage of ESP. ESP converges rapidly to a high-performing returned solution (Figure 4(a)), while taking actions with significantly lower regret (defined as the reward difference between optimal solution and current policy) than the other methods during learning (Figure 4(b)). On both metrics, ESP converges orders-of-magnitude faster than the other approaches. In particular, after a few hundred episodes, ESP reaches a solution that is significantly better than any found by DE or PPO, even after 3,000 episodes. This result shows that, beyond being more sample efficient, by systematically exploiting historical data, ESP is able to find solutions that direct evolution or policy gradient search cannot.

The next sections show how these advantages of ESP can be harnessed in standard RL benchmarks, focusing on the advantage of surrogate modeling over direct evolution in the domain, comparing to DQN and PPO, and demonstrating the regularization effect of the surrogate.

4.2 Comparing with Standard RL

The goal of the Cart-pole experiments was to demonstrate ESP's performance compared to direct evolution and standard RL methods.
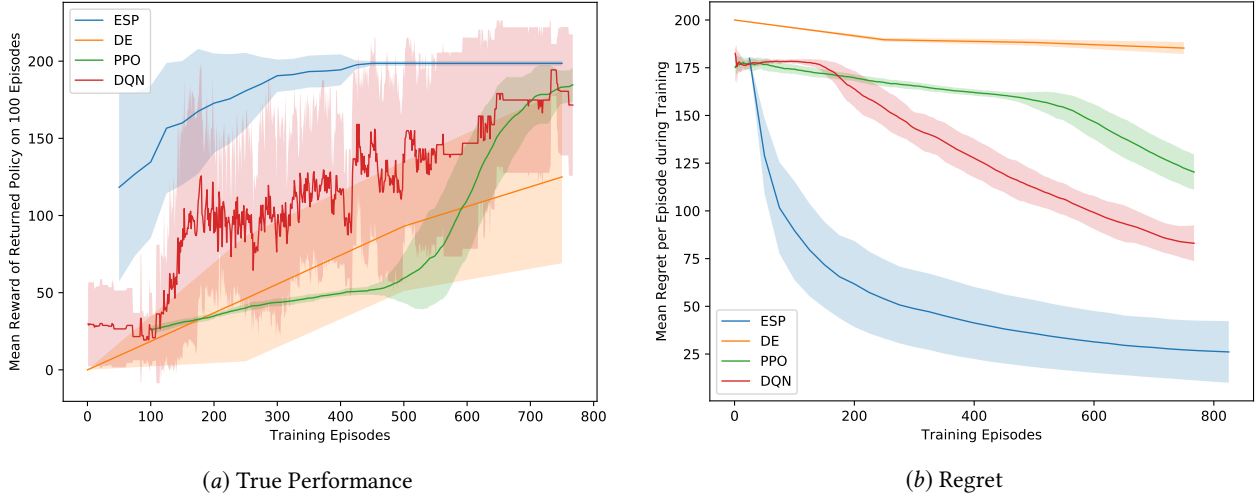
(a) True Performance

(b) Regret

Figure 5: *Performance in the Cart-Pole Domain.* **The same experimental and plotting conventions were used as in Figure 4.** (*a*) **True performance of the best policy returned by each method during the learning process. True performance is based on the average reward of 100 real-world episodes (this evaluation is not part of the training). ESP converges significantly faster and has a much lower variance, demonstrating better sample efficiency and reliability than the other methods.** (*b*) **Average regret over all past training episodes. ESP has significantly lower regret, suggesting that it has lower cost and is safer in real-world applications.**

Problem Description. The Cart-pole control domain is one of the standard RL benchmarks. In the popular CartPole-v0 implementation on the OpenAI Gym platform [3] used in the experiments, there is a single pole on a cart that moves left and right depending on the force applied to it. A reward is given for each time step that the pole stays near vertical and the cart stays near the center of the track; otherwise the episode ends.

Algorithm Setup. DE was run with a population size of 50 candidates. A candidate is a neural network with four inputs (observations), one hidden layer of 32 units with tanh activation, and two outputs (actions) with argmax activation functions. The fitness of each candidate is the average reward over five episodes in the game, where the maximum episode length is 200 time steps.

ESP runs similarly to DE, except that the fitness of each candidate is evaluated against the Predictor instead of the game. The Predictor is a neural network with six inputs (four observations and two actions), two hidden layers with 64 units each and tanh activation, and one output (the predicted discounted future reward) with tanh activation. It is trained for 1,000 epochs with the Adam optimizer [24] with MSE loss and batch size of 256.

The first Predictor is trained on samples collected from five random agents playing five episodes each. Random agents choose a uniform random action at each time step. A sample corresponds to a time step in the game and comprises four observations, two actions, and the discounted future reward. Reward is +1 on each time step, except for the last one where it is adjusted to +2,000 in case of success, −2,000 in case of failure (i.e. 10× max time steps). The discount factor is set to 0.9. The reward value is then scaled to lie between -1 and 1.

In order to be evaluated against the Predictor, a Prescriptor candidate has to prescribe an action for each observation vector from the collected samples. The action is then concatenated with the observation vector and passed to the Predictor to get the predicted future reward. The fitness of the candidate is the average of the predicted future rewards.

Every five generations, data is collected from the game from the five elites, for five episodes each. The new data is aggregated into the training set and a new Predictor is trained. The generation's candidates are then evaluated on the new Predictor with the new training data. The top elite candidate is also evaluated for 100 episodes on the game for reporting purposes only. Evolution is stopped after 160 generations, which corresponds to 800 episodes played from the game, or once an elite receives an average reward of 200 on five episodes.

In addition to DE and ESP, two state-of-the-art RL methods were implemented for comparison: double DQN with dueling network architectures [30, 52] and actor-critic style PPO [44]. The implementation and parametric setup of DQN and PPO were based on OpenAI Baselines [8]. For PPO, the policy's update frequency was set to 20, which was found to be optimal during hyperparameter search. All other parametric setups of DQN and PPO utilized default setups as recommended in OpenAI Baselines.

Results. Figure 5(a) shows how the true performance of the best policy returned by ESP, DE, PPO, and DQN changes during the learning process in CartPole-v0. For ESP and DE, the elite candidate that has the best real-world fitness is selected as the best policy so far. For DQN and PPO, whenever the moving average reward of the past 100 episodes of training is increased, the best policy will be updated using the most recent policy. One hundred additional
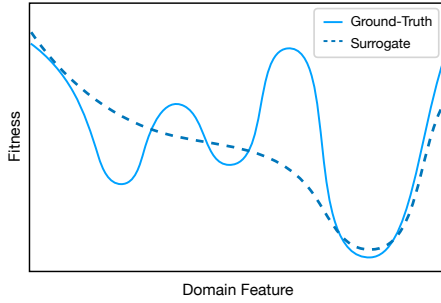
Figure 6: *Surrogate Approximation of the Fitness Landscape.* **The fitness in the actual domain may be deceptive and nonlinear, for instance single actions can have large consequences. The surrogate learns to approximate such a landscape, thereby creating a surrogate landscape that is easier to search and optimize.**

real-world episodes were used to evaluate the best policies (these evaluations are not part of the training).

ESP converges significantly faster than the other methods, implying better sample-efficiency during learning. Moreover, the variance of the true performance becomes significantly smaller for ESP after an early stage, while all other algorithms have high variances even during later stages of learning. This observation demonstrates that the solutions delivered by ESP are highly reliable.

Figure 5(b) shows the average regret for training processes of all algorithms in CartPole-v0. ESP has significantly lower regret during the entire learning process, indicating not only lower costs but also better safety in real-world interactions.

4.3 Regularization Through Surrogate Modeling

Whenever a surrogate is used to approximate a fitness function, there is a risk that the surrogate introduces false optima and misleads the search [21] (for an inspiring collection of similar empirical phenomena, reference Lehman et al. [25].) ESP mitigates that risk by alternating between actual domain evaluations and the surrogate. However, the opposite effect is also possible: Figure 6 shows how the surrogate may form a more regularized version of the fitness than the real world, and thereby make it easier to learn policies that generalize well [18, 34].

Problem Description. Flappy Bird is a side-scroller game where the player controls a bird, attempting to fly it between columns of pipes without hitting them by performing flapping actions at carefully chosen times. This experiment is based on a PyGame [50] implementation of this game, running at a speed of 30 frames per second. The goal of the game is to finish ten episodes of two minutes, or 3,600 frames each, through random courses of pipes. A reward is given for each frame where the bird does not collide with the boundaries or the pipes; otherwise the episode ends. The score of each candidate is the average reward over the ten episodes.

Algorithm Setup. Both DE and ESP were setup in a similar way as in the preceding sections. DE had a population of 100 candidates,
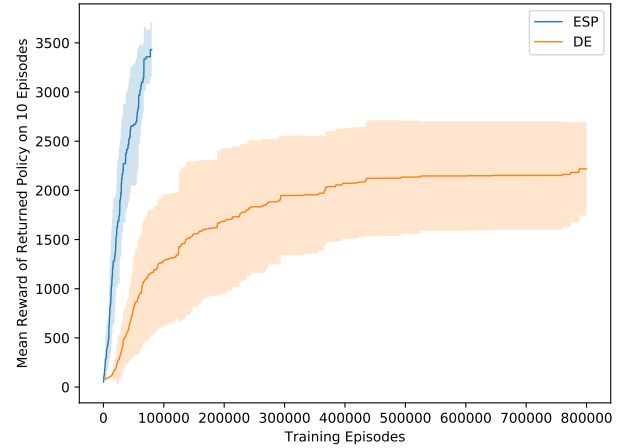

Figure 7: *Performance in the Flappy Bird Domain.* **The same experimental and plotting conventions were used as in Figure 5(a), except true performance of the best policy returned is based on the average reward of 10 real-world episodes during the training. ESP discovers a policy that solves the task very quickly, whereas DE cannot discover it even though it is run an order of magnitude longer. This result is likely due to the regularization effect that the surrogate provides as shown in Figure 6.**

each a neural network with eight inputs (observations), one hidden layer of 128 nodes with tanh activation, and two outputs (actions) with argmax activation. The ESP Predictor was a random forest with 100 estimators, approximating reward values for state-action pairs frame by frame. The state-action pairs were collected with the ten best candidates of each generation running ten episodes on the actual game, for the total of a hundred episodes per generation.

Results. Figure 7 shows how the true performance of the best policy returned by ESP and DE improved during the learning process. The elite candidate that has the best real-world fitness was selected as the best policy so far. In about 80,000 episodes, ESP discovered a policy that solved the task, i.e. was able to guide the bird through the entire course of pipes without hitting any of them. It is interesting that DE converged to a suboptimal policy even though it was run an order of magnitude longer. This result is likely due to the regularization effect illustrated in Figure 6. Direct evolution overfits to the nonlinear effects in the game, whereas the surrogate helps smooth the search landscape, thereby leading evolution to policies that perform better.

5 DISCUSSION AND FUTURE WORK

The results in this paper show that the ESP approach performs well in sequential decision making tasks like those commonly used as benchmarks for RL. Compared to direct evolution and state-of-the-art RL methods, it is highly sample efficient, which is especially important in domains where exploring with the real world is costly. Its solutions are also reliable and safe, and the complexity of its models can be adjusted depending on the complexity of the data and

task. These advantages apply to ESP in general, including decision strategies that are not sequential, which suggests that it is a good candidate for improving decision making in real-world applications, including those in business, government, education, and healthcare.

When ESP is applied to such practical problems, the process outlined in Section 3 can be extended further in several ways. First, ESP can be most naturally deployed to augment human decision making. The Presciptor's output is thus taken as advice, and the human decision maker can modify the actions before applying them. These actions and their eventual outcomes are still captured and processed in Step 4 of the ESP process, and thus become part of the learning (Figure 2). Second, to support human decision making, an uncertainty estimation model such as RIO [36] can be applied to the Predictor, providing confidence intervals around the outcome O'. Third, the continual new data collection in the outer loop makes it possible to extend ESP to uncertain environments and to dynamic optimization, where the objective function changes over time [18, 19, 57]. By giving higher priority to new examples, the Predictor can be trained to track such changing objectives. Fourth, in some domains, such as those in financial services and healthcare industries that are strongly regulated, it may be necessary to justify the actions explicitly. Rather than evolving a Prescriptor as a neural network, it may be possible to evolve rule-set representations [16] for this role, thus making the decision policy explainable. Such extensions build upon the versatility of the ESP framework, and make it possible to incorporate the demands of real-world applications.

Although the Predictor does not have to be perfect, and its approximate performance can even lead to regularization as was discussed in Section 4.3, it is sometimes the bottleneck in building an application of ESP. In the non-sequential case, the training data may not be sufficiently complete, and in the sequential case, it may be difficult to create episodes that run to successful conclusion early in the training. Note that in this paper, the Predictor was trained with targets from discounted rewards over time. An alternative approach would be to incrementally extend the time horizon of Predictors by training them iteratively [38]. Such an approach could help resolve conflicts between Q targets, and thereby help in early training. Another approach would be to make the rewards more incremental, or evolve them using reward function search [17, 33]. It may also be possible to evaluate the quality of the Predictor directly, and adjust sampling from the real world accordingly [20].

An interesting future application of ESP is in multiobjective domains. In many real-world decision-making domains, there are at least two conflicting objectives: performance and cost. As an evolutionary approach, ESP lends itself well to optimizing multiple objectives [4, 10]. The population forms a Pareto front, and multiple Prescriptors can be evolved to represent the different tradeoffs. Extending the work in this paper, it would be illuminating to compare ESP in such domains to recent efforts in multiobjective RL [26, 31, 56], evaluating whether there are complementary strengths that could be exploited.

6 CONCLUSION

ESP is a surrogate-assisted evolutionary optimization method designed specifically for discovering decision strategies in real-world applications. Based on historical data, a surrogate is learned and used to evaluate candidate policies with minimal exploration cost. Extended into sequential decision making, ESP is highly sample efficient, has low variance, and low regret, making the policies reliable and safe. Surprisingly, the surrogate also regularizes decision making, making it sometimes possible to discover good policies even when direct evolution fails. ESP is therefore a promising approach to improving decision making in many real world applications where historical data is available.

REFERENCES

[1] Ackley, D. and Littman, M. 1991. Interactions between learning and evolution. *Artificial Life II* 10 (1991), 487–509.

[2] Barto, A. G., Sutton, R. S., and Anderson, C. W. 1983. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13 (1983), 834–846.

[3] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. 2016. OpenAI Gym. *CoRR* abs/1606.01540 (2016).

[4] Coello Coello, C. A. 1999. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *International Journal of Knowledge and Information Systems* 1 (1999), 269–308.

[5] Cressie, N. 1990. The origins of kriging. *Mathematical Geology* 22, 3 (1990), 239–252.

[6] Deb, K. and Myburgh, C. 2017. A population-based fast algorithm for a billion-dimensional resource allocation problem with integer variables: Breaking the Billion-Variable Barrier in Real-World. *European Journal of Operational Research* 261 (2017), 460–474.

[7] Deisenroth, M. and 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 (ICML) (ICMLâĂŹ11)*. 465–472.

[8] Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. 2017. OpenAI Baselines. https://github.com/openai/baselines. (2017).

[9] Dreer, L. E. and Linley, A. 2017. Behavioral Medicine: Nutrition, Medication Management, and Exercise. In *Practical Psychology in Medical Rehabilitation*, M. Budd, S. Hough, S. Wegener, and W. Stiers (Eds.). Springer.

[10] Emmerich, M. and Deutz, A. 2018. A tutorial on multiobjective optimization: Fundamentals and evolutionary methods. *Natural Computation* 17 (2018), 585âĂŞ–609.

[11] Gomez, F., Schmidhuber, J., and Miikkulainen, R. 2006. Efficient non-linear control through neuroevolution. In *Proceedings of the European Conference on Machine Learning*. Springer, 654–662.

[12] Grefenstette, J. J. and Fitzpatrick, J. M. 1985. Genetic search with approximate function evaluations. In *Proceedings of the International Conference on Genetic Algorithms and their Applications*. 112–120.

[13] Ha, D. and Schmidhuber, J. 2018. Recurrent World Models Facilitate Policy Evolution. In *Advances in Neural Information Processing Systems 32 (NIPSâĂŹ18)*. Curran Associates Inc., Red Hook, NY, USA, 2455âĂŞ2467.

[14] Hasselt, H. V. 2010. Double Q-learning. In *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (Eds.). Curran Associates, Inc., 2613–2621.

[15] Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S., and others, . 2017. Emergence of locomotion behaviours in rich environments. *arXiv:1707.02286* (2017).

[16] Hodjat, B., Shahrzad, H., Miikkulainen, R., Murray, L., and Holmes, C. 2018. PRETSL: Distributed Probabilistic Rule Evolution for Time-Series Classification. In *Genetic Programming Theory and Practice XIV*. Springer, 139–148.

[17] Houthooft, R., Chen, Y., Isola, P., Stadie, B., Wolski, F., Ho, O. J., and Abbeel, P. 2018. Evolved policy gradients. In *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 5400–5409.

[18] Jin, Y. 2011. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation* 1, 2 (06 2011), 61–70. https://doi.org/10.1016/j.swevo.2011.05.001

[19] Jin, Y. and Branke, J. 2005. Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation* 9, 3 (June 2005), 303–317. https://doi.org/10.1109/TEVC.2005.846356

[20] Jin, Y., Husken, M., and Sendhoff, B. 2003. Quality measures for approximate models in evolutionary computation. In *Proceedings of the Bird of a Feather Workshop, Genetic and Evolutionary Computation Conference (GECCO)*. 170–173.

[21] Jin, Y., Olhofer, M., and Sendhoff, B. 2000. On Evolutionary Optimization with Approximate Fitness Functions. 786–793.

[22] Jin, Y., Wang, H., Chugh, T., Guo, D., and Miettinen, K. 2019. Data-Driven Evolutionary Optimization: An Overview and Case Studies. *IEEE Transactions on Evolutionary Computation* 23 (2019), 442–458.

[23] Johnson, A. J., Meyerson, E., de la Parra, J., Savas, T. L., Miikkulainen, R., and Harper, C. B. 2019. Flavor-Cyber-Agriculture: Optimization of plant metabolites in an open-source control environment through surrogate modeling. *PLOS ONE* (2019). https://doi.org/10.1371/journal.pone.0213918.

[24] Kingma, D. P. and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv:1412.6980* (2014).

[25] Lehman, J., Clune, J., Misevic, D., Adami, C., Beaulieu, J., Bentley, P. J., Bernard, S., Beslon, G., Bryson, D. M., Chrabaszcz, P., Cheney, N., Cully, A., Doncieux, S., Dyer, F. C., Ellefsen, K. O., Feldt, R., Fischer, S., Forrest, S., Frénoy, A., Gagné, C., Goff, L. K. L., Grabowski, L. M., Hodjat, B., Hutter, F., Keller, L., Knibbe, C., Krcah, P., Lenski, R. E., Lipson, H., MacCurdy, R., Maestre, C., Miikkulainen, R., Mitri, S., Moriarty, D. E., Mouret, J., Nguyen, A., Ofria, C., Parizeau, M., Parsons, D. P., Pennock, R. T., Punch, W. F., Ray, T. S., Schoenauer, M., Shulte, E., Sims, K., Stanley, K. O., Taddei, F., Tarapore, D., Thibault, S., Weimer, W., Watson, R., and Yosinksi, J. 2018. The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities. *arXiv:1803.03453* (2018).

[26] Liu, C., Xu, X., and Hu, D. 2015. Multiobjective Reinforcement Learning: A Comprehensive Overview. *IEEE Transactions on Systems, Man, and Cybernetics* 45 (2015), 385–398.

[27] Miikkulainen, R. 2019. Creative AI Through Evolutionary Computation. In *Evolution in Action: Past, Present and Future*, Banzhaf et al. (Ed.). Springer, New York.

[28] Miikkulainen, R., Brundage, M., Epstein, J., Foster, T., Hodjat, B., Iscoe, N., Jiang, J., Legrand, D., Nazari, S., Qiu, X., Scharff, M., Schoolland, C., Severn, R., and Shagrin, A. In Press. Ascend by Evolv: AI-Based Massively Multivariate Conversion Rate Optimization. *AI Magazine* (In Press).

[29] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML) (ICMLâĂŹ16)*. 1928–1937.

[30] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., and others, . 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[31] Mossalam, H., Assael, Y. M., Roijers, D. M., and Whiteson, S. 2016. Multi-Objective Deep Reinforcement Learning. *arXiv:1610.02707* (2016).

[32] Naik, P. A., Raman, K., and Winer, R. S. 2005. Planning Marketing-Mix Strategies in the Presence of Interaction Effects. *Marketing Science* 24 (2005), 25–34.

[33] Niekum, S., Barto, A. G., and Spector, L. 2010. Genetic programming for reward function search. *IEEE Transactions on Autonomous Mental Development* 2, 2 (2010), 83–90.

[34] Ong, Y., Nair, P., and Keane, A. 2003. Evolutionary Optimization of Computationally Expensive Problems via Surrogate Modeling. *AIAA Journal* (04 2003). https://doi.org/10.2514/2.1999

[35] Pierret, S. and Braembussche, Van den R. 1999. Turbomachinery blade design using a Navier-Stokes solver and Artificial Neural Network. *Journal of Turbomachinery* 121, 2 (1999), 326–332.

[36] Qiu, X., Meyerson, E., and Miikkulainen, R. 2020. Quantifying Point-Prediction Uncertainty in Neural Networks via Residual Estimation with an I/O Kernel. In *Proceedings of the Eighth International Conference on Learning Representations (ICLR).*

[37] Ray, A., Achiam, J., and Amodei, D. 2019. Benchmarking Safe Exploration in Deep Reinforcement Learning. (2019). https://cdn.openai.com/safexp-short.pdf

[38] Riedmiller, M. 2005. Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the European Conference on Machine Learning*. Springer, 317–328.

[39] Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv:1703.03864* (2017).

[40] Saxe, A. M., Mcclelland, J. L., and Ganguli, S. 2014. Exact solutions to the nonlinear dynamics of learning in deep linear neural network. In *Proceedings of the Second International Conference on Learning Representations (ICLR).* Citeseer.

[41] Schmidhuber, J. and Huber, R. 1991. Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems* 2, 01n02 (1991), 125–134.

[42] Schneider, G., Schuchhardt, J., and Wrede, P. 1994. Artificial neural networks and simulated molecular evolution are potential tools for sequence-oriented protein design. *Bioinformatics* 10, 6 (1994), 635–645.

[43] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. 2016. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the Fourth International Conference on Learning Representations (ICLR).*

[44] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017).

[45] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., and others, . 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484.

[46] Snoek, J., Larochelle, H., and Adams, R. P. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2951–2959.

[47] Spector, L., Goodman, E., Wu, A., B. Langdon, W., Voigt, m. H., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M., Burke, E., and Kaufmann Publishers, M. 2001. Autoconstructive Evolution: Push, PushGP, and Pushpop. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)* (05 2001).

[48] Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. 2019. Designing Neural Networks through Evolutionary Algorithms. *Nature Machine Intelligence* 1, 1 (2019), 24âĂŞ35.

[49] Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. 2017. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv:1712.06567* (2017).

[50] Tasfi, N. 2016. PyGame Learning Environment. https://github.com/ntasfi/PyGame-Learning-Environment. (2016).

[51] Wahlström, N., Schön, T. B., and Deisenroth, M. P. 2015. From pixels to torques: Policy learning with deep dynamical models. *arXiv:1502.02251* (2015).

[52] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. 2016. Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML) (ICMLâĂŹ16)*, Vol. 48. JMLR.org, 1995âĂŞ2003.

[53] Watkins, C. J. C. H. 1989. *Learning from delayed rewards*. Ph.D. Dissertation. Cambridge University.

[54] Werbos, P. J. 1987. Learning how the world works: Specifications for predictive networks in robots and brains. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, NY.*

[55] Whiteson, S. 2012. Evolutionary computation for reinforcement learning. In *Reinforcement Learning*. Springer, 325–355.

[56] Yang, R., Sun, X., and Narasimhan, K. 2019. A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 14610–14621.

[57] Yu, X., Jin, Y., Tang, K., and Yao, X. 2010. Robust optimization over time âĂŤ A new perspective on dynamic optimization problems. In *IEEE Congress on Evolutionary Computation*. 1–6. https://doi.org/10.1109/CEC.2010.5586024