Master research Internship

Internship report

Implementation of a symbolic variant for the Black-DROPS algorithm

Artificial Intelligence, Systems and Control, Robotics, Neural and Evolutionary Computing, Machine Learning

Supervisor:

Author:
Vladislav Tempez

Jean-Baptiste Mouret
INRIA Larsen

Abstract: We present a extension for a reinforcement learning algorithm for robot controllers, Black-DROPS. The Black-DROPS algorithm is a model-based policy search algorithm that is designed to minimize interaction time. It builds a model of the dynamical system thanks to Gaussian processes. Its sample-based estimation and policy optimization strategy allow it to profit from parallelization. But Gaussian processes and neural networks lacks interpretability when used as policies or system models. In this internship we tried to replace Black-DROPS models and policies with symbolic regression to gain this interpretability and reduce query time. The implementation of Black-DROPS written during this internship showed that symbolic regression is unable to match the performance of Gaussian processes models and policies but are performing as well as neural networks in some cases.

Contents

1 Introduction

The developments of robotics and robots introduced important changes in industry, but there are obstacles to expand their use out of factories. The complex interactions between robot bodies and their controllers in a non-controlled environment and the variety of robot morphologies makes the design of a specific controller necessary for each situation. Moreover, the tasks they are expected to complete may be more complex than in factories and changing over time. Unfortunately, the design of a controller might be long and involve intensive testing. These challenges led to the use of machine learning to design the controllers of these robots in order to be able to design them immediately when needed with minimal human help.

In order to design these controllers, several learning approaches are commonly used. The supervised learning approach consists in giving a sampled situation and the response that the controller is expected to give to this situation [SSBD14] (e.g. a command that is should process when in a given state). This approach is limited to tasks for which it is possible to devise an algorithm that searches for the answer in the general case or for problems for which we possess a large number of samples and the corresponding answers (e.g. images and their annotations produced by captchas). Hence when there are few examples on which to learn and no way to efficiently generate more of them this approach is difficult to use.

The reinforcement learning approach [SB11] relaxes the quality of feedback given to the learning system by only providing it an estimation of how good is the answer suggested by the system to a given situation rather than the ideal answer to provide to this situation. This feedback, inspired by some teaching methods, can be interpreted as rewards or penalties given to the system and might be much easier to devise than suggesting a precise action to take.

The classical reinforcement learning approach (e.g. Q-learning, TD-learning [SB11]) is to build a map of the expected reward $r(a, x)$ received by the learning agent in state x for taking action a in order to be able to estimate the reward that future actions will earn. These predictions of future rewards allow the agent to choose actions that maximize the reward and thus, solve the task. Since this approach learns the reward for couples (a, x) that were sampled, it needs an important sampling phase to be able to predict rewards in the general case.

Another approach in reinforcement learning that intends to learn a strategy that generalizes more is to assume that there exists a function π^* (called a policy) that links current state of the model x to optimal action $a = \pi^*(x)$ and searches for this function in a parametric function space $A = \{\pi_\theta | \theta \in \mathbb{R}^p\}$. This optimal policy is searched as the policy that grant the maximum reward. In robotics the policy can constrain the robot behavior.

But even with a feedback limited to a number accounting for the quality of the action taken by the learning agent, getting the data necessary to optimize a policy can be untractable for robotics applications (e.g. in [MKS+15] more than a month of real-time interaction with the system was needed).

A solution is provided by model-based policy optimization (as opposed to model-free approaches introduced before), that is the optimization of the policy is done thanks to an estimation of its efficiency by interacting not with the system but with an internal model of the system, built during the learning process, that simulates real interactions. In robotics, most of these models are still limited to the dynamics of the robot itself and ignore environment in which the robot evolves.

One important property of the internal system models is that their predictions are often wrong. In order to improve the quality of the estimation of the policy efficiency, it has been found that the model should account for the uncertainty in the prediction they make about system dynamics

[DFR15]. Knowing about this uncertainty allows policy search process to be more robust to model errors and bridge the gap between simulation and physical systems.

The Black-DROPS algorithm [CRK$^+$17] is model-based policy search algorithm that learns robot controller using model uncertainty and an evolution strategy (namely CMA-ES[HO01]) to search for the optimal policy. This algorithm uses the popular Gaussian processes [RW06] as model. This type of model is an infinite dimensional extension of Gaussian distributions, making it a smooth model with a built in uncertainty evaluation of its predictions. However the computational cost to make these predictions is important and depending on the amount of data used to learn, thus we investigate the possibility to replace this model by models whose queries are less expensive, symbolic regression models.

Symbolic regression model are what we usually call formulas, thus, the time needed to query is independent of the data it was learned on we expect their query time to be shorter. Moreover these model are easier to interpret in the sense that we are used to use mathematical formulas to describe models in many fields.

In this internship a variant of this Black-DROPS algorithm that no longer uses Gaussian processes and replaces them with symbolic regression as models and policies was implemented. In order to search the space of mathematical formulas we use genetic programming [Koz94]. The use of bootstrap methods to assess the uncertainty of symbolic regression models was planned but not realized, as a consequence, the model used are unable to provide any uncertainty measure for their predictions. The use of lexicase selection was planned in genetic programming but instead, diversity mechanisms were used.

We evaluate this implementation on two dynamical system: a inverted pendulum and a cart-pole. We find that symbolic regression models are not as precise as Gaussian processes but are on par with neural network. We find that symbolic regression policies performs sometimes as well as neural network policies, but are longer to search for.

This report is organized as follows. Section 2 presents work related to the approach used in this internship, updates the preliminary bibliographic study, introduce the formalization of the problem and the notation used. Section 3 describes precisely the models, the algorithm and the dynamics of the systems we use. Section 4 presents the numerical results of the implemented models in Black-DROPS variants in term of reward obtained, learning time and show some of the best formulas found to model the systems. Section 5 summarizes the conclusions of this work and suggest some future extensions.

2 Bibliographic Study

The approach taken in this internship uses various tools and frameworks. This related work Section articulates these references according to the following categories. Subsection 2.1 presents a formalization of the problem of learning a model and optimizing the expected reward for a policy in reinforcement learning framework. Subsection 2.2 presents the models and tools that are used to build algorithms and policies. Subsection 2.3 deals with the challenges related to the application of these techniques to robotics and present the example of the Black-DROPS algorithm.

2.1 Problem formalization

2.1.1 Learning a dynamical model

The problem of estimating the dynamics of a system can be framed in the following way. We consider $f(x, u)$ to be the function that represent the evolution of the system, where $x \in \mathbb{R}^E$ is the state of the system at a previous step and $u \in \mathbb{R}^F$ is the control applied or action taken by the learning agent (e.g. the robot). Hence we have that

$$x_{t+1} = x_t + f(x_t, u_t) + w_t \tag{1}$$

were w_t accounts for some noise. It is often assumed to be Gaussian noise independent of the time.

Two implicit assumptions in this formulation are

- the system has no memory, its state depends only on the previous state and the control applied by the learning agent
- the system is stationary, its dynamics are invariant to time translation. i.e x does not involve time.

Learning a system model is to build an approximation of f thanks to sequence of $(x_i, u_i)_{1 \leq i \leq n}$ consecutive states of the system and the control applied to the system during transitions.

2.1.2 Finding a good policy

In the reinforcement learning framework, the reward function $r : \mathbb{R}^E \mapsto \mathbb{R}$ gives an estimation of the desirability of the state of the system. We assume that the learning agent behaves according to a policy $u = \pi(x) : \mathbb{R}^E \mapsto \mathbb{R}^F$. In order to find a good policy π we maximize the expected reward of π:

$$J_\pi = \mathbb{E}(\sum_{t=0}^{T} r(x_t)) \tag{2}$$

where $x_{t+1} = x_t + f(x_t, \pi(x_t)) + w$.

The policy π can be parametrized by some parameters $\theta \in \mathbb{R}^p$ and finding a good π is equivalent to finding a good θ in the parameter space.

2.2 Tools and models for learning

To compute an approximation of the dynamical model f or a policy π, two main aspects are to be examined. The first is about the way this approximation of f (or π) should be represented, in which set of mathematical objects we should search for it. In this subsection we present Gaussian processes, symbolic regression and neural networks as some important sets of objects that can be used to represent both models and policies. The second aspect is to choose how to search in the set of objects representing the approximation of f (or π). The second part of this subsection presents some sampling-based techniques and evolutionary algorithms that could be used to search in the set of potential representations of f or π.

2.2.1 Gaussian processes

Gaussian processes uses in machine learning is detailed in [RW06]. These models consist in distribution over functions constrained by observations of the system transitions (e.g. date samples).

More formally, a Gaussian process can be defined as a collection f_x of random variables each following a Gaussian law. These variables are not independent but they covary according to a covariance function $k(.,.)$. Hence a Gaussian process is characterized by two functions, its mean m and its covariance k.

For all x, x' we have that:

$$f_x \sim \mathcal{N}(m(x), k(x, x)) \tag{3}$$

$$\text{cov}(f_x, f'_x) = k(x, x') \tag{4}$$

To use Gaussian processes in order to build a model, they are conditioned according to the observed samples, giving a new probability distribution to predict unobserved data. The prediction of a Gaussian process for the dynamics of the system at state x^* according to observed data X is the random variable $f_{x^*|X}$ such that

$$P(f_{x^*|X} = x_0) = P(f_{x^*} = x_0|X) \tag{5}$$

From a practical point of view, the prediction for a new point x^* knowing observations $(X, Y) = (x_i, y_i = f(x_i))_{i \leq n}$ is sampled from a Gaussian law whose mean is a linear combination of (y_i) with weights determined by the covariances of the observations and x^*. Variance is also determined by these covariances. Thus, Gaussian processes can be seen as lazy learners, a more elaborate version of k-nearest neighbors using kernels.

$$\mathbb{E}(f_{x^*|(X,Y)}) = K(x^*, X)K_X^{-1}Y \tag{6}$$

and

$$\text{Var}(f_{x^*|X,Y}) = k(x^*, x^*) - K(x^*, X)K_X^{-1}K(x^*, X)^T \tag{7}$$

Where K_X is the matrix whose coefficients are $k(X_i, X_j)_{i,j}$ and $K(x*, X)$ is the vector $k(x^*, X_i)_i$. One advantage of Gaussian processes is that the evaluation of the confidence in their prediction is directly built in the model via the distribution conditioned by the observations. Gaussian processes are also a non parametric model, making them a rich and flexible class with no prior on the world. Gaussian processes can still be tailored to a specific task by modifying the shape of the covariance function k. One common shape for this covariance function is the squared exponential.

$$k(x, x') = e^{\frac{-1}{2l}||x-x'||_2^2} \tag{8}$$

Some more elaborated kernels are used in [CRK+17, DFR15] rescaling the L_2 norm according to a matrix $\Lambda = \text{Diag}(\lambda_i)$.

$$k(x_i, x_j) = \sigma_1^2 e^{-\frac{1}{2}(x_i - x_j)^T \Lambda^{-1}(x_i - x_j)} + \sigma_2^2 \delta_j^i \tag{9}$$

These scaling factors λ_i and the noise coefficients σ_1, σ_2 are hyper-parameters of Gaussian processes that can be optimized.

However Gaussian processes might be long to query since they use linear combination of the observed data at each query. In fact, a data-matrix multiplication is needed for each query (see eq.

6, 7) making queries costly to use with large number of samples. This query cost is at least $O(N^2)$ where N is the number of samples. In addition the equation defining the mean and variance (eq. 6, 7) both need a matrix inversion whose classical implementation uses Cholesky decomposition that has a $O(N^3)$ complexity. Matrix K (its inverse) uses all samples of the training set and should be computed each time the model is updated.

Gaussian processes were used for regression in other fields [Kri51] for some years. They recently where used to build models in robotics in [DFR15, CRK$^+$17]. A degenerated version without uncertainty (using only the mean) was experimented in [DFR15] in order to assess the importance of the uncertainty in the learning. It revealed necessary to keep this uncertainty to achieve satisfying results. Moreover these degenerated Gaussian processes without uncertainty were used as deterministic policy models in [DFR15, CRK$^+$17].

The strong mathematical foundations of these models also allowed the derivation of analytical approximations of distributions of the future states of the system in [DFR15]. These distributions allowed to compute an analytical approximation of the gradient of the policy to optimize according to a (differentiable) reward function.

Even with this possibility, [CRK$^+$17] decided to use a sample approach to estimate distributions of the future state of the system and to optimize the policy according to non-differentiable reward functions. It also made the estimation of the expected reward of a policy easier to parallelize and reduced computation time.

2.2.2 Symbolic regression

An alternative to Gaussian processes for modeling systems dynamics is to represent these dynamics f directly by a formula whose variables are variables of the system. The idea behind this approach is that the set of mathematical formulas is rich enough to contain a good approximation of f. The definition of these formulas may vary from arithmetic expressions (using only sum and product) to analytic expressions (using infinite series). While these formulas may be rich enough, there is no inherent way to learn these and to infer their reliability as in Gaussian processes. In addition, the richer the set of formulas is, the more difficult it is to explore it.

However, the query of these formulas needs only elementary operations (in the case of closed form formulas) which requires much less computing time. The time and memory needed to query these formulas does not increase with the observed data as the Gaussian processes do (see equations (6),(7)).

One important reason for which symbolic regression is used is the ability for domain experts to look at the learned formulas and understand why they were chosen and what they mean. Interpretability is further discussed in next subsection.Searching the space of formulas is often done with genetic algorithms.

Formulas as system models were successfully used in [SL09] to capture known physical invariants of mechanical systems such as pendulum or springs with no prior bias on the evolution of these systems. It proves that at least small mechanical systems dynamics can be correctly modeled by formulas. Formulas were also used as models of large-scale water dynamics in [PE08]. This work was successful in finding the qualitative importance of some variables in the studied process of evapotranspiration. However there is no comparison of this work to other state of the art techniques for evapotranspiration modeling and it is thus difficult to assess the capacity of formulas to model large scale systems dynamics.

Rather than modeling the system itself, formulas are used in [MFWE12] to model the reward given to the agent. The policy π_F using this formula F modeling the reward is:

$$\pi_F(x) = \text{argmax}_u \; F(x,u) \tag{10}$$

Even when the formulas can use variables of the *future* state of the system that they are modeling (this is possible in simulations), the corresponding policies fail to achieve an efficiency comparable to model-free policy search Fitted Q Iteration [EGW05].

Formulas can also be used directly as policies. To do that, the set of mathematical operators can be extended to programming operators, like a branching operator "if". These extended formulas are simple programs that can compute the image of policy π. These models for policies are used in [HUR17] as controller in simple robotics tasks in a continuous state-action space. This achieve efficiency comparable to a neural network policy.

In [MFWE12, HUR17, SL09, PE08] a special attention was given to the size of the formulas to keep them from acquiring unnecessary complexity. The complexity of the model in terms of number of mathematical operators and length of the formulas was found in [PE08] to increase accuracy of the model at the expense of its uncertainty. This might be explained by the well known trade-off Variance-Bias of machine learning [SSBD14]. In [SL09], polynomial approximation of operators (e.g. cos an sin) that are not allowed in the formulas are found during the search, relaxing the necessity to provide all the needed operators to a symbolic regression algorithm.

The interpretability of formulas as models of policies is studied in [DEW02] where formulas are used as policies in iterated ultimatum game. These policies are used to model human behavior according to an history of human moves.

2.2.3 Neural networks

Neural networks are widely used in machine learning [Hay94]. The base component of a neural network is a neuron. A neuron is inspired from biological neurons. It makes linear combinations of its inputs, and returns a non linear function, named *activation function*, of this combination as output. Several of these neurons are assembled in a *neural network* whose nodes are neurons and edges are weighted to provide the coefficient of the linear combination. The output y of a neuron that is linked to n other neurons with weights w_i whose outputs are x_i is:

$$y = A(\sum_i^n w_i x_i) \tag{11}$$

where A is the activation function of this neuron. Unlike the two former methods, neural networks are parametric models. For a given network, the set of its parameters is the set of the weights of its edges.

Neural networks are well known for their use in supervised learning. The weights of the edges w_i are updated through the back-propagation [RHW$^+$88] of the difference between their output $N(x)$ and the desired output y.

$$w_i = w_i + \frac{\partial ||N(x) - y||_2^2}{\partial w_i} \tag{12}$$

These neural networks can be used either to model systems as in [HUR17] where a neural network is used as baseline to optimize policies or directly as a policy as in [CRK$^+$17]. With the other components of the Black-DROPS algorithm, the neural network policy used was found to be as efficient as the state of the art on simulated and real world tasks. Unlike Gaussian processes, neural networks possess no inherent uncertainty evaluation, making their use as models of stochastic or noisy systems less stable or needing an additional mechanism to manage uncertainty.

2.2.4 Sampling-based methods

When modeling dynamics, the presence of noise can make predictions uncertain, moreover, the system itself might be stochastic. In order to manage this uncertainty about systems dynamics the estimation of expected behavior of the system can be conducted. A first technique to estimate this expected behavior is to sample the model and average the results. This falls into the field of Monte Carlo methods [KTB13]. In the domain of policy search, a particular Monte Carlo method, the PEGASUS method, is introduced in [NJ00]. PEGASUS method builds a deterministic system from a Markovian stochastic one. It replaces stochastic transitions in a Markov decision process by deterministic ones that have a supplementary parameter t accounting for the randomness. It can be seen as taking the random number generator out of the model of the system. This transformation allows to conduct policy optimization in a deterministic frame on the system and then average it for several sequences of t. The PEGASUS method ensures the theoretical convergence of this sampling method and experiments conducted in [NJ00] showed that this method led to improvement over previous results.

Black-DROPS [CRK$^+$17] takes another approach to manage this uncertainty. It assumes that the uncertainty on the evaluation of the expected reward J_π of a policy π through an uncertain model f of a system is modeled by Gaussian noise. Thus we have $G_\pi = J_\pi + w_\pi$ where w is a Gaussian noise, and G_π is the reward computed for π with the uncertain model of the system. The consequence of this assumption is that optimizing noisy evaluation of the policy G is equivalent to optimize J. This is the same technique as for stochastic gradient descent [SSBD14].

Bootstrapping [ET94] is another approach to estimate uncertainty concerning observed states X of an uncertain system. In order to estimate the distribution of a statistical quantity, for example the average reward $J_\pi(X)$ for policy π for the system in state X, new sets of states B_i, called bootstraps samples, are created by sampling X with replacement. Bootstrap does not add new information to the original observations. If X is seen as a random variable, the distribution of $J_\pi(B_i)$ is an estimation of the distribution of $J_\pi(X)$. From this distribution, uncertainty over $J_\pi(X)$ can be computed. Bootstrap is used in [PE08] to estimate the uncertainty of the formulas predicting evapotranspiration.

The optimal policy can also be found via black-box optimization techniques. The Covariance MAtrix Evolution Strategy (CMA-ES [HO01]) is an efficient black-box optimization technique. It iteratively searches the optimal parameters for a policy (or a function in general) thanks to sampling and selection. At each step, λ values $(\theta_i)_{i\leq\lambda}$ for parameters are sampled according to a multivariate Gaussian distribution centered around a mean θ_m and of covariance C. The μ best parameters according to policy reward J_{π_θ} are then selected and m is set to $m = \text{mean}_{i\leq\mu}\ \theta_i$ and C is set to $C = \text{Var}_{i\leq\mu}\ \theta_i$. Black-DROPS uses a variation of CMA-ES to optimize its policies (namely neural network and Gaussian process).

2.2.5 Genetic algorithms

Evolutionary algorithms [Hol92] are a class of black-box optimizers inspired from Darwin theory of biological evolution. A *population* of potential solutions of the problem is randomly generated. All of these *individuals* are then evaluated according to the *fitness function*, i.e. the function to optimize and the best individuals are selected. The selected individuals are then combined (*cross-over* operation) and modified (*mutation* operation) in order to produce a new population. There are many ways to select, combine and modify individuals.

This black-box optimization scheme assumes no regularity about the function to optimize, making it useful when the domain of the function is not easy to explore. One example of particular interest is the case of formulas. On the domain of formulas there is no obvious way to conduct classical optimization (such as gradient descent or convex optimization) as there is no obvious metric structure on formulas. Even the black-box optimizer CMA-ES cannot directly explore the space of formulas because it only works with parameters in a vector space.

Evolutionary algorithms can search this space as soon as a way to slightly modify the formulas and to combine them is specified. In [PE08, HUR17] the formulas modeling the evapotranspiration and the policies are represented as trees. The combination of two formulas is done by exchanging two randomly chosen sub-trees, and the mutation is done by changing an operator in a node or a value at a leaf of the tree. In [SL09] the formulas are represented as directed acyclic graphs. These graphs are converted into lists of assembly-like instructions. Combination is done by taking the first half of one formula and the second half of another.

The selection process can take several forms, in [PE08] the selection of an individual is done randomly with a probability proportional to the fitness of the individual (roulette wheel selection). In [HUR17] the individuals are selected through a tournament process where at each match of this tournament the best of two individual is chosen (tournament selection).

In the context supervised learning with several objectives to optimize, a specific selection process, Lexicase selection [HSM15] has been designed. This context might be adapted for systems where there are several constraints and none should be neglected. Naive approaches like summing, or combining into a single number, the fitness of the different objectives are sometimes insufficient. Lexicase selection attempts to select only elite individuals, i.e. individuals that show the best fitness for a given objective. The idea behind this is that these elite individuals possess a part of the solution for one objective and they could be able to spread it to the whole population. This selection is done according to a randomized list of the several objectives, keeping only the individuals that are the best for this current objective until all objectives are tested or only one individual is left. Lexicase selection showed significantly better results on several genetic programming (genetic algorithms for which individuals are programs) problems [HSM15]. One reason for this better performance is that Lexicase selection ensured better diversity of the individuals in the population. This diversity can be seen as a way to avoid to be stuck around local minimums of the function to optimize.

In [HUR17, SL09] interesting results and efficient policies are achieved, proving that genetic algorithms optimization can find good solutions.

Another approach to tackle these multiple objective is to keep every good solution until the end. An individual is said *non-dominated* if no other individual beats it on all objectives. Non-dominated individual and kept into the next generation. The algorithm NSGA-II [Deb01] uses this technique and ensures that the population is diverse enough to prevent premature convergence of the population to a local maximum. This algorithm pseudo-code follows

Algorithm 1 NSGA-II

```
Initialize population P with random individuals.
while Population is not empty do
    Extract all individuals that are non dominated
    Give these individuals the rank i
    increase i
end while
while task not solved do
    Evaluate individuals witness according to each objective
    Initialize new empty population P'
    Set n to the size of P
    Create offspring by selecting parents according to their ranking and how far they are from
    others is the ranks are equal.
    Add offspring to P
    Set counter i to 0
    while P is not empty do
        Extract all individuals that are non dominated
        Give these individuals the rank i
        if number of individuals in P'+ number of individuals of rank i < n then
            Add individuals of rank i to P'
        else
            Add the individuals of rank i that are farthest from other until P' has as many individuals
            as P'
        end if
        increase i
    end while
end while
```

If the problem defined in is not fundamentally multi-objective, the use of several initial configurations or the estimation of the fitness through several bootstraps samples can provide other objectives and with Lexicase selection only keep solutions of low uncertainty.

Another advantage of using multi-objective techniques is that they can help to prevent a phenomenon called *bloating* [DJWP01] that is wasting exploration and diversity of the population in formulas that are equivalent, e.g. adding a 0 constant to a good formula, or dividing it by one, etc. This increases the size of the formulas, making them longer to evaluate. Penalization techniques [SSBD14] aims at tackling these problems, penalizing individuals' fitness according to the size of the formula they represent can lead to premature convergence toward local maximums by excluding complex formula from the search space. The work in [DJWP01] shows that adding a simplicity criterion among the optimization objective can lead the search towards simpler individuals.

2.2.6 Learning both models and policies: Black-DROPS algorithm

The Black-DROPS algorithm [CRK$^+$17] is a learning algorithm that uses techniques discussed in this section to find controllers for robots. This algorithm uses the model-based policy search approach to reduce the interaction time and be usable in robotics. It learns a model of the system in order to use it to find good policies (that are robot controllers). It builds on the state of the art algorithm

PILCO [DFR15]. Its objectives are being as efficient as PILCO in terms of interaction time needed with the physical system, be able to use multi core hardware, manage any kind of reward function and be able to use any kind of parametric function. The three last objectives were not achieved by PILCO which relied on specific properties of its reward function and its Gaussian process policy in order to derive the analytical approximations of the expected reward.

Black-DROPS keeps the use of a Gaussian process to model the system for which it searches a policy and the estimation of its uncertainty allowed by this Gaussian model. The expected reward of a policy is optimized by the CMA-ES method (more precisely the CMA-ES variant introduced in [Han09]). This optimization method allows to optimize any parametric policy with regard to reward function that are not necessarily differentiable unlike PILCO gradient descent optimization method. It also benefits from parallelization as it is a population based method and it allows the evaluation of several policies in parallel. The estimation of the uncertainty on the policy evaluation is assumed Gaussian as described 2.2.4 allowing faster estimation of the expected reward. The whole algorithm is described by the following pseudo-code:

Algorithm 2 Black-DROPS

```
Proceed with random moves and collect observation data about system dynamics $X$ and reward $R$.    O($T$) simulator operations.
while task not solved do
    Train system model with Gaussian processes on data $X$    $O(N_d^3)$
    Train reward model with Gaussian process on data $R$    $O(N_d^2)$
    Optimize policy $\pi_\theta$ with CMA-ES method according to noisy evaluation $G_{\pi_\theta}$    $O(N_{opt} T N_d^2)$
    Try policy on robot and collect data about system $X'$ and reward $R'$.    $O(TC_\pi)$
    Add $X'$ to $X$ and $R'$ to $R$.    O(T)
end while
```

The asymptotic complexity of each step is precised on the left. N_d is the number of sampled points so far, T the number of steps in one try of the task, N_{opt} the number of optimization steps and C_π the query cost of the policy. One turn of the while loop is called and *episode*.

Experiments conducted in [CRK+17] confirm that Black-DROPS needs interaction time comparable to PILCO, achieves comparable or better performances on simulations tasks and real world systems tasks (see 3.1) while truly benefiting from the use of several cores.

2.3 Challenges for learning in robotics

The use of techniques described in the previous section is not uncommon in robotics but still experimental and several problems can show up. First the advantages of model-based policy search is detailed with regard to interaction time, and then the case of the ability to interpret the learned models is discussed.

2.3.1 Data efficiency

In robotics applications, the evaluation of the expected reward of a policy on the system itself can take several seconds to several minutes. If many of them have to be evaluated (as in genetic algorithms or CMA-ES) the optimization time is greatly increased. This quantity of interaction with a system to learn properly is called *data efficiency*. The less interaction needed, the higher the data efficiency. For robotics application a high data efficiency is needed since interaction time cannot extend forever. One of the objectives of PILCO [DFR15] is to reduce this interaction time.

PILCO uses a Gaussian process as internal model of the system and takes the uncertainty of this model into account. Policy estimation can be done according to this model. This allows PILCO to have a better data efficiency than other algorithms at that time. The Black-DROPS algorithm described in [CRK$^+$17] build on this to keep data efficiency and allow parallelizability of the learning process. It uses parallelizability to reduce the computation power needed outside the interactions with the world.

2.3.2 Interpretability of the learned models

The necessity to be able to understand the meaning of the learned system model is highlighted in [HUR17, MFWE12, SL09]. In [DEW02] the objective is even to copy human strategies then analyze these strategies through the analysis of the learned models. In [SL09] the objective is to rediscover known, and thus interpretable, formulas in order to apply this process to systems with unknown dynamics and find formulas directly usable by domain experts.

However, this interpretability can take many forms. The work in [Lip16] discusses different forms of interpretability. The use of symbolic regression as model relates to interpretability by being decomposable, simulatable and transparent, where Gaussian processes by their lazy learner nature may be interpreted through carefully selected examples.

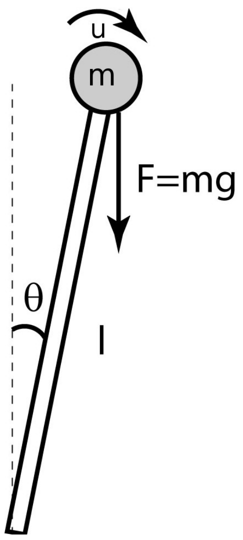

Figure 1: The inverted pendulum setup

3 Experimental setup and tasks

3.1 Evaluation tasks

In order to assess the efficiency of learning algorithms, we present some classical tasks used in many papers allowing these algorithms to be compared. A first task, the *inverted pendulum* is to control a pendulum thanks to the application of torque $u \in [-2.5, 2.5]$N.m on it. For this task the goal is to maintain the pendulum on top position. Hence, the state space is:

$$(\theta, \dot{\theta}) \in \mathbb{R}^2 \tag{13}$$

The dynamics of the pendulum system can be described by the following equations:

$$\dot{\theta} = \dot{\theta} \tag{14}$$

$$\ddot{\theta} = \frac{u - b\dot{x} - \frac{mgl \cdot \sin(\theta)}{2}}{\frac{m \cdot l^2}{2}} \tag{15}$$

where m is the mass of the pendulum, l is the length of the pendulum arm, g the gravitational constant and b a friction coefficient. The simulated pendulum has $m = 0.5kg$, $l = 0.5$m, $g = 9.81$ and $b = 0.1$. One important fact about this system is that it is under-actuated, the maximum torque does not allow to directly bring the pendulum at $\theta = \pi$, a balancing movement is needed. The reward is decided by the distance to the top position (i.e. $\theta = \pi$).

$$r(x, \dot{x}, \theta, \dot{\theta}) = \exp(-\frac{1}{2\sigma^2} \cdot ((\cos(\theta) - \cos(\pi))^2 + (\sin(\theta) - \sin(\pi))^2) \tag{16}$$

The *cart-pole swing-up* task consists in balancing an inverted pendulum set on the top of a cart by moving the cart forward and backward. The state of this task is described by 4 variables that are

$$(x, \dot{x}, \theta, \dot{\theta}) \in \mathbb{R}^4 \tag{17}$$

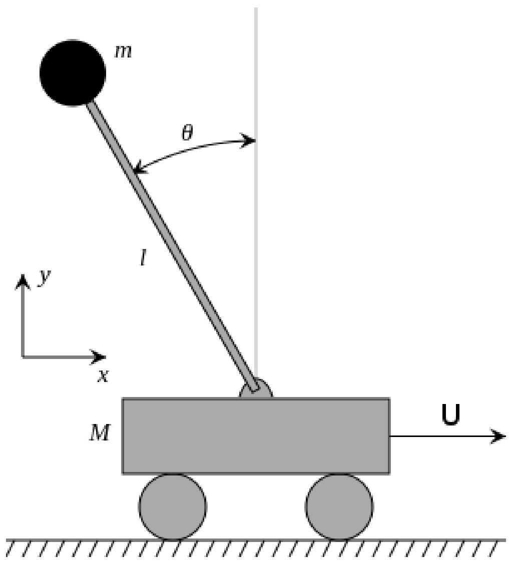

Figure 2: The cart-pole setup

where x is the cart position and θ the angle of the pendulum. The control u is a continuous application of a force $u \in [-10N, 10N]$ on the cart along the horizontal direction.

The dynamics of this system can be described by the following equations:

$$\dot{x} = \dot{x} \tag{18}$$

$$\ddot{x} = \frac{2ml \cdot \dot{\theta}^2 \cdot \sin(\theta) + 3mg \cdot \sin(\theta) \cdot \cos(\theta) + 4u - 4b \cdot \dot{x}}{4(M+m) - 3m \cdot \cos(\theta)^2} \tag{19}$$

$$\dot{\theta} = \dot{\theta} \tag{20}$$

$$\ddot{\theta} = \frac{-3ml \cdot \dot{\theta}^2 \cdot \sin(\theta) \cdot \cos(\theta) - 6(M+m)g \cdot \sin(\theta) - 6(u - b\dot{x}) \cdot \cos(\theta)}{4l(m+M) - 3ml \cdot \cos(\theta)^2} \tag{21}$$

where m is the mass of the pendulum, M is the mass of the cart, l is the length of the pendulum arm, g the gravitational constant and b a friction coefficient. The simulated cart-pole has $m = 0.5kg$, $M = 0.5$kg, $l = 0.5$m, $g = 9.81$ and $b = 0.1$.

A reward is given for an angle θ close to 0, i.e. a pendulum on top position (i.e. $\theta = \pi$).

$$r(x, \dot{x}, \theta, \dot{\theta}) = \exp(-\frac{1}{2\sigma^2} \cdot (x^2 + (\cos(\theta) - \cos(\pi))^2 + (\sin(\theta) - \sin(\pi))^2) \tag{22}$$

In order to simulate these two systems the dynamics equations were discretized to the following form:

$$\theta_{t+1} = \theta_t + \dot{\theta}_t \cdot \mathrm{dt} \tag{23}$$

$$\dot{\theta}_{t+1} = \dot{\theta}_t + \frac{u - b\dot{x}_t - \frac{mgl \cdot \sin(\theta_t)}{2}}{\frac{m \cdot l^2}{2}} \tag{24}$$

that is a Euler integration scheme. That scheme was chosen during this internship over more elaborate integration scheme (like Runge-Kutta scheme [Pre07]) in order to provide a step variation of the state of the system whose formulas are small enough to be found by symbolic regression techniques. The original integration scheme used in Black-DROPS [CRK$^+$17] experiments was more elaborate, making simulations more stable, especially for the cart-pole system which is very sensitive to small state variations. In addition, mimicking the technique used in PILCO [DFR15], the state is extended, i.e. the learning algorithm is not given directly the angle θ directly but rather a couple $(\cos(\theta), \sin(\theta))$ in order to avoid all problems due to the equivalence of angle measure modulo 2π. This solution is preferred over the use of a principal value (i.e. the corresponding value of θ in $]-\pi, \pi]$) because of the discontinuity arising at $\theta = \pi$. This extended state is used only as input, meaning that the models directly predicts variation of of θ (i.e. $\theta_{t+1} - \theta_t$) from inputs $(\cos(\theta_t), \sin(\theta_t))$.

For both these system, episodes of 4 seconds sampled à 10Hz are used, i.e. 40 samples per episode.

3.2 Genetic operators and formulas operators

The space of formulas in which models are search is defined by the following structure, formulas are trees. Leaves are composed of the input variables of the system (previous state and command) and constants: 0 to 9, the time step dt and g. Each non terminal node is a mathematical operator with as many children as the operator arity. The operators used are cos, sin, $+$, -, $*$, $/$.

Formulas for policies were only given the state as input, the same constants as for the model and the same operators. A conditional operator was added for policies to allow to switch behavior if needed (if$(a, b, c) \mapsto$ if $a > 0$ then b else c).

This set of operators was chosen so that it is able to find the correct formulas for the dynamics of the system. All the operators appearing in the system dynamics are provided, with the exception of the exponent operator that is easily replaced by $*$ since exponents are no larger than 2 in system dynamics.

The crossover operator (i.e. the way to combine the formulas of two individuals) is done by exchanging two sub-formulas chosen uniformly from the two parents. Two mutating operators are defined. The first uniformly selects an operator in a formula and replaces it by another operator of the same arity. The second one selects a sub-formula and replaces it by a newly generated random formula.

Rather than using lexicase selection as initially planned, the algorithm NSGA-II is used. This choice was made because of an already implemented version of the algorithm NSGA-II in C++ that existed in the library SFERESv2 [MD10]. This headers only C++ implementation allowed an easier integration in the Black-DROPS code. Moreover, the NSGA-II algorithm relied on diversity preserving mechanisms among the population. This diversity preservation proved useful at preventing premature convergence to local extremum [Deb01] and helping exploration [MC15, CVM18, VM18].

NSGA-II is used to learn system models with two objectives to minimize: the squared prediction error and the size of the formula (i.e. the number of nodes in its tree). The algorithm NSGA-II uses a distance between individuals as a supplementary mechanism to maintain diversity.

This distance was defined according to the data points on which each individual was making errors. Formally, for two individuals i_i, i_2, a behavior vector was defined:

$$b(i) = (1 - \frac{1}{1 + e_{i,k}})_{k \in [|1,n|]} \qquad (25)$$

where $e_{i,k}$ is the prediction error of individual i for learning sample k and n is the number of sample in the learning set. The distance between i_1 and i_2 was defined as the squared l_2 norm of these vectors:

$$d(i_1, i_2) = ||b(i_1) - b(i_2)||_2^2 \tag{26}$$

This distance can be interpreted as making two individuals distant if the sets of points on which they make prediction errors are different. The point of defining distance between individuals according to the error they make rather than their structure is again to avoid bloating. A formula and a semantically equivalent bloated version of it are considered close according to this distance but might be considered different according to distance defined directly on the structure of the formulas. Moreover, the prediction error is squashed to $[0, 1]$ in order to prevent formulas that are making one really bad prediction (e.g. prediction error going to infinity) completely different from all other formulas.

To learn policy formulas, the two objective are to maximize the reward obtained by the policy according to the model of the system dynamics and to minimize the size of the formula. A similar distance is used but the behavior vector $b(i)$ is the sequence of commands that the formula for the policy gives for an episode. The conjunction of diversity preserving mechanism and multi-objective to reduce bloat is experimental for robotics.

$$b(i) = (i(x_t))_{t\in[|0,n|]} \tag{27}$$

where n is the number of simulation steps in an episode and

$$x_{t+1} = x_t + f(x_t, i(x_t)) \tag{28}$$

starting from a given state x_0.

The two system used (3.1) are deterministic, the symbolic regression models and policies (i.e.) formulas are deterministic, hence the command sequence is fixed by the model, the policy and the initial position.

No uncertainty mechanism is implemented for symbolic regression models there was not enough time to add the bootstrapping techniques initially planned. Since the system were deterministic and no uncertainty mechanism was implemented on symbolic regression, the Gaussian processes models were used without their uncertainty prediction, using directly the mean as prediction.

3.3 Libraries used

The Black-DROPS algorithm[CRK+17] is completely inserted into the limbo [CCAM16] C++ library that offers Gaussian process and Bayesian optimization implementation.

An initial attempt was made to use the gpcxx C++ library (`https://github.com/Ambrosys/gpcxx`). This library already implements genetic programming and symbolic regression, however it lacks NSGA-II implementation and its documentation is scarce and incomplete, making it difficult to use. Hence, the library SFERESv2 [MD10] was used. This library is well documented and actively maintained. It already implements NSGA-II algorithm that is able to manage multiple objectives. However it lacks an implementation of genetic programming. An important part of this internship was dedicated to the implementation of a genetic programming extension for SFERESv2.

For comparison with neural network a wrapper interfacing the tiny-dnn library (`https://github.com/tiny-dnn/tiny-dnn`) with Black-DROPS.

The extension of SFERESv2 written during the internship will soon be integrated into the SFERESv2 library as a module.

4 Results

In order to assess the benefits of replacing model and policies by symbolic regression in Black-DROPS several parameters we compare the predictions made by these models to actual trajectories obtained in the simulator. As Black-DROPS algorithm proceeds in two phases, namely learning the model, then optimizing the policy at each episode, we look at both phase independently.

In this section we first present the efficiency of the symbolic regression implementation as models compared to Gaussian processes models and neural networks models according to prediction error. Then we look at the evaluation of symbolic regression as policies compared to Gaussian processes and neural network policies, according to the optimization time and reward obtained. Finally, we compare the efficiency and learning time of Black-DROPS with all pairs of models (Gaussian processes, symbolic regression, neural networks) and policies (Gaussian processes, symbolic regression, neural networks).

4.1 Learning dynamical models

In order to evaluate the efficiency of models on meaningful data, we create a training set and a testing set from trajectories generated by two classical Black-DROPS runs of 15 episodes (i.e. 15*40*2 = 1200 data points) for each set. That is, Black-DROPS was run for 4 times 15 steps of the following loop:

1. learn a model from data
2. optimize policy according to the learned model
3. execute best policy found
4. collect state, command and reward from execution and add it to the data used to learn

and the execution of step 3 gives trajectories that are used as training and testing data to evaluate models only. This choice for training and testing data was made to ensure that the models were evaluated on parts of the state space that are important for the task to solve (i.e. keeping the pendulum on top).

Each experiment was replicated 20 times in order to get an estimation of the average performance of the performance of this model. However, the Gaussian processes and neural network models show much less variation of performance, hence variance is not visible on the plot.

In order to assess the importance of the data efficiency, the learning was conducted several times with a growing fraction of the total learning test but was tested on the entire test set. The samples constituting the smaller data set are chosen according to a simple deterministic process in order to not insert more random variations: 10 training sets are built, each containing only samples whose index $i \equiv 0[k]$ where k is the number of the training set. This means that the 1st training set will contain all samples whose index are multiple of 1, i.e. the full training set, the 2nd only even samples, i.e $\frac{1}{[}2]$ of the full training set, and the 10th will contain only samples whose index are multiple of 10, i.e. $\frac{1}{10}$ of the full training set.

NSGA-II was used with a population of 200 individuals and searched for a model during 1000 generation at each time every time it is used except if something else is specified. For each state variable the average absolute value of the difference between the predicted value and the true value was computed.

On the plots showing performances of the models, symbolic regression will be labeled "SR", Gaussian processes "GP" and neural networks "NN". On the pendulum (3.1) system all the models achieved a good precision. Figure 3 shows at top the average absolute error of prediction of the angle θ for the three types of models. The error the symbolic regression model achieves seems independent of data quantity. The variation of the performance is low here. Gaussian processes error is higher but drops to a a small error (regarding the possible values for θ) with enough data. The neural networks error is higher than the three models but still achieves a low error (<0.01) with enough data (as much data as in less than 15 episodes).

Figure 3 shows at bottom the average absolute error of prediction of the angular velocity $\dot{\theta}$ for the three types of models. The error of the symbolic regression model seems to not vary much with the quantity of data, even if it is several order of magnitude higher than for the prediction of the angle (10^{-2} compared to 10^{-5}). Both Gaussian process model and neural network model have improving performance with the quantity of data. Gaussian process model achieves an average error lower than for the prediction of the angle (10^{-4} compared to 10^{-3}). The neural network model's error is comparable to its error for angle θ prediction.

Figure 4 presents the same results than on the bottom of Figure 3 but for a symbolic regression search during 20000 generation rather than 1000. It shows that the symbolic regression model does not uses more data than the minimal amount given to it and can achieve better precision if optimized during more generations. We can also notice that the variance of this error is lower (50% of the median value compared to 100% of the median value).

The performance is not strictly monotonous with the amount of data. The reason for that may be that the process building the different training sets does not ensure that these training sets are a growing sequence (e.g. the $\frac{1}{10}$ training set contains data points that are not in the $\frac{1}{3}$), thus some important points may not be included. For both Gaussian processes model and neural networks model, the performance on the full training set is better than the performance of any smaller set. That fact support the previous hypothesis.

The symbolic regression model does not learn in a deterministic way. Hence, the 25th and 75th percentiles were displayed to assess the reliability of the process. If this learning performance does not vary much for the prediction of the angle, the prediction of the angular velocity is more spread. This non deterministic behavior might be the second reason for the non monotonous variation of performance with data quantity.

The reason for the drop in performance for symbolic regression model certainly is the different complexity of the relative equations. For angle prediction (equation 14) the formula to found consists only in one variable where for angular velocity variation (equation 15) the formula is larger and then more difficult to found by the evolutionary algorithm.

For cart-pole system (3.1) only the Gaussian Process manages to learn a correct model. Figure 5 presents the performance of the different models for the angle variables (angle position θ and angular velocity $\dot{theta}$). As for the pendulum system, for variable θ, the symbolic regression model achieves very low error and is more precise than the two other models. However, there is an important variation of its performances. For this variable, neural network model cannot go lower than an error of 0.1, and the Gaussian process model achieves a good precision.

Concerning the angular velocity $\dot{\theta}$, symbolic regression models are no longer able to learn a useful model, the error they achieve is higher than one, making future predictions largely incorrect. Neural network model achieves slightly better results with all data. Gaussian processes model is more precise and able to use the full training set to improve its predictions.

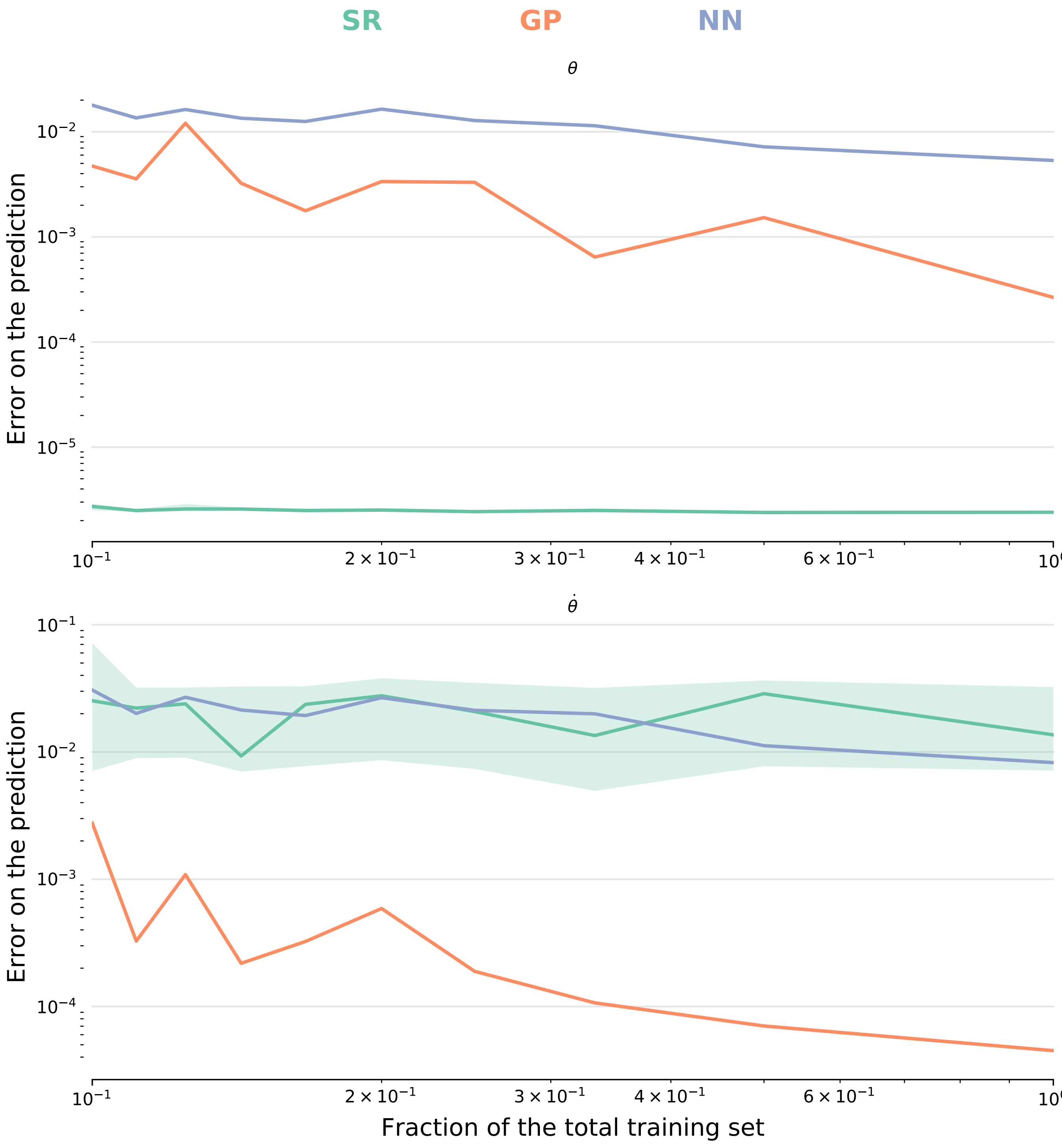

Figure 3: Average absolute error of prediction of variables θ and $\dot{\theta}$ on the pendulum system for three models types: symbolic regression (SR), Gaussian processes (GP), neural networks (NN) learning on a fraction of the whole training set. Uncertainty is given by the clearer area representing 25th and 75th percentiles over 20 replicates. Logarithmic scale for both x and y.

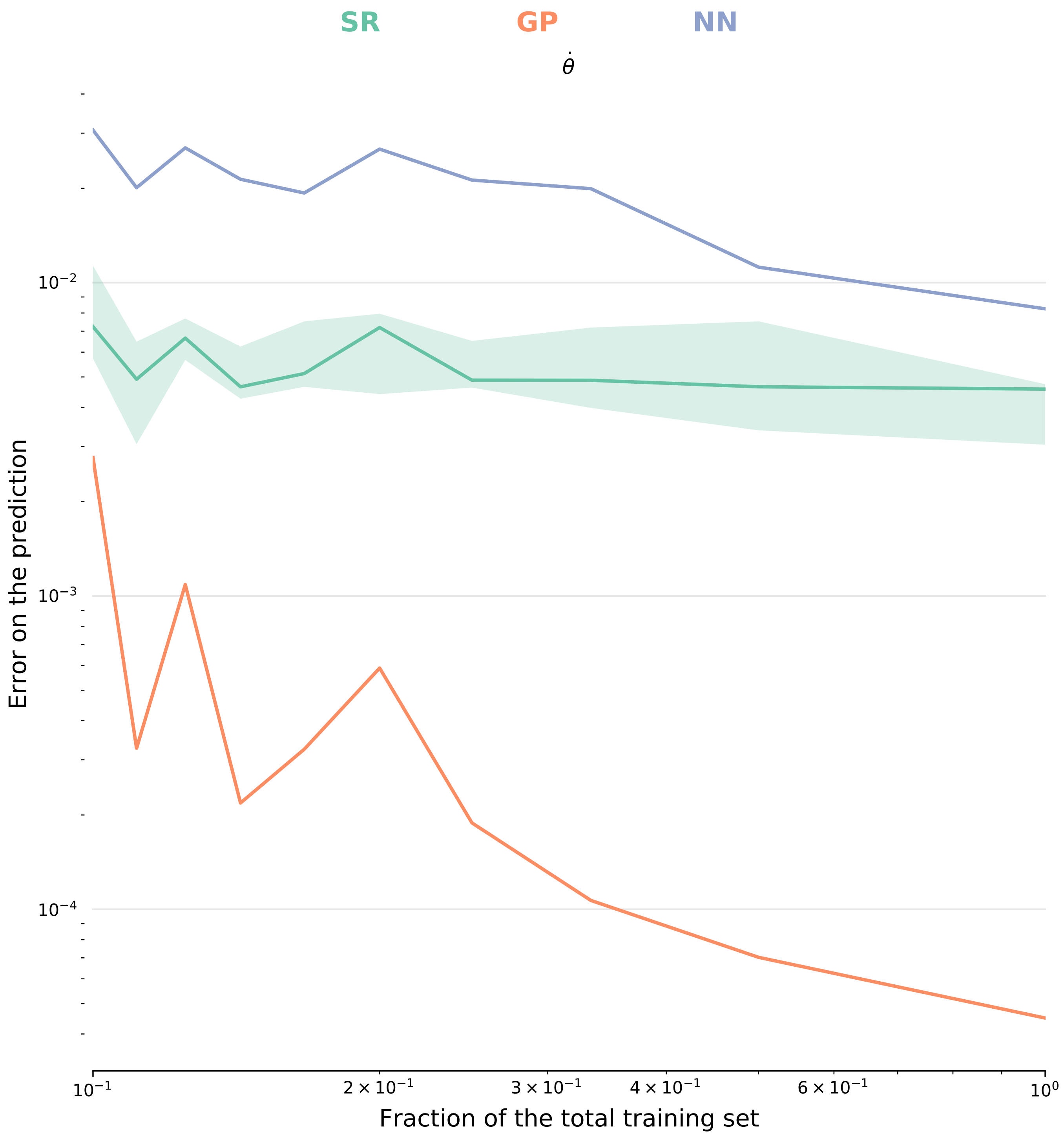

Figure 4: Error for $\dot{\theta}$ with 20 000 generations instead of 1000 as in figure 3

On this system again, symbolic regression model does not seem to benefit from the increasing quantity of data. Gaussian processes model does benefit from it, and neural network model do benefit only when the initial error is high (for $\dot{\theta}$).

Figure 6 shows similar results replacing *theta* by x and $\dot{\theta}$ by $\dot{x}$. It is however important to notice that Gaussian processes model does not benefit much from data increase in prediction of position. It has worse error with the entire training set than with only half of it.

Here again the complexity of dynamics formulas for both velocity and angular velocity (equations 19, 21) might be the reason why symbolic regression model is unable to model correctly these variables. However, the dimension of the state space increased from 2 to 4 compared to the pendulum. This might be a reason for the higher error of the neural network model and the higher variance in the error of the symbolic regression model.

From these results we can conclude that for really simple dynamics, symbolic regression models are able to learn from few data points and achieve significantly lower error than Gaussian processes and neural network models. However when dimensionality and formula complexity increases symbolic regression models are unable to scale well, but achieve results similar to neural networks. Gaussian processes model on the other hand benefit a lot from a large training set and are able to correctly scale according to the prediction performance.

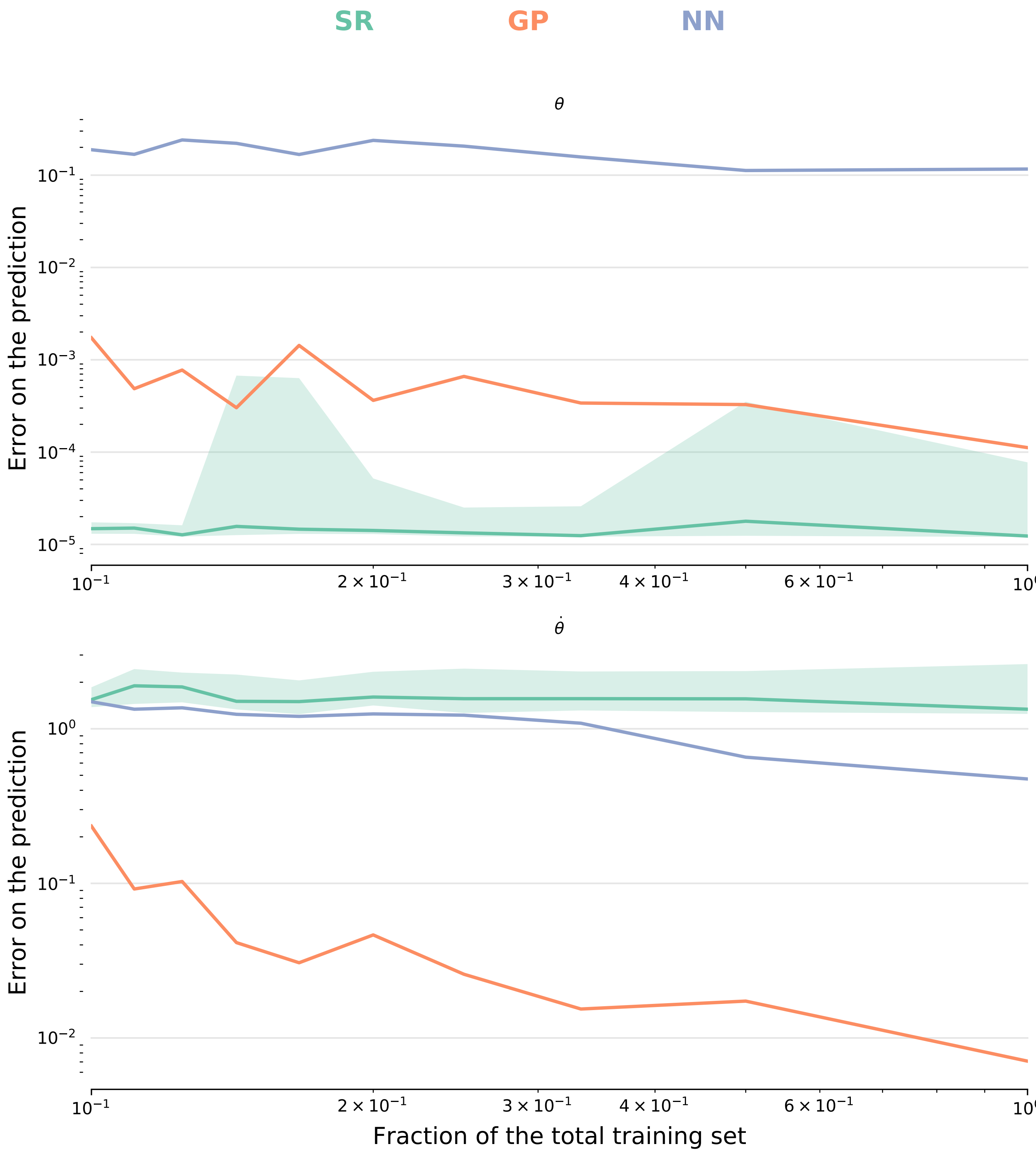

Figure 5: Average absolute error of prediction of variable θ and $\dot{\theta}$ on the cart-pole system for three models types: symbolic regression (SR), Gaussian processes (GP), neural networks (NN) learning on a fraction of the whole training set. Uncertainty is given by the clearer area representing 25th and 75th percentiles over 20 replicates. Logarithmic scale for both x and y.

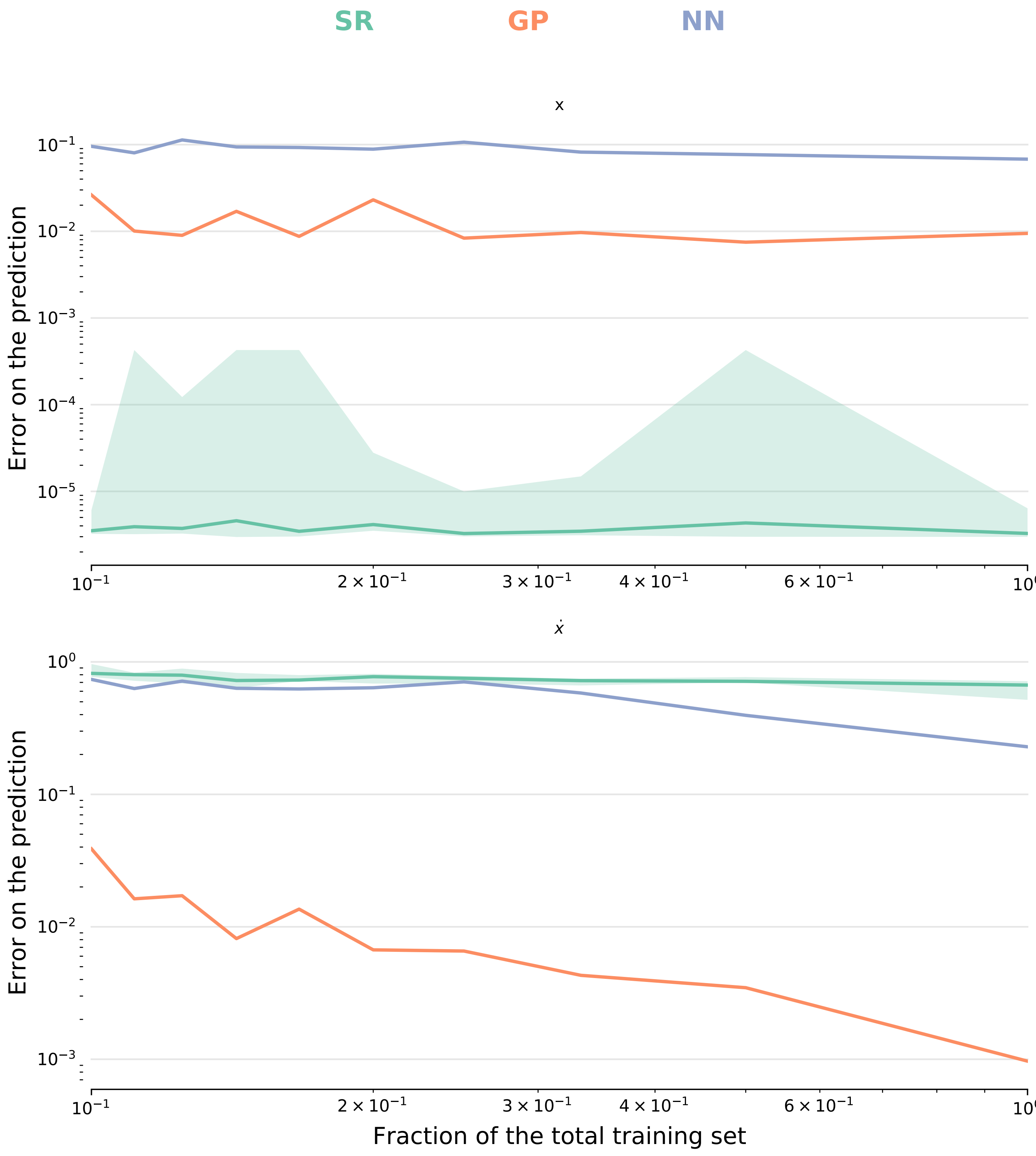

Figure 6: Average absolute error of prediction of variable x and $\dot{x}$ on the cart-pole system for three models types: symbolic regression (SR), Gaussian processes (GP), neural networks (NN) learning on a fraction of the whole training set. Uncertainty is given by the clearer area representing 25th and 75th percentiles over 20 replicates. Logarithmic scale for both x and y.

4.2 Policies

In order to assess the efficiency of the policies structure, namely Gaussian processes-based, symbolic regression-based and neural network-based policies independently of the quality of the underlying model, an ideal model is implemented to match the dynamics used by the simulator. This model uses directly the dynamics equations used in the simulator used by Black-DROPS and described in 3.1. It is used in Black-DROPS for 15 episodes following the algorithm described at 2.2.6, except that the model learning phase is skipped since the model is already ideal.

However, because of the models use a transformations of the state space (i.e. replacing θ by its cos and sin) and the simulator does not, to reuse the exact same equations of the dynamics a conversion from the couple $(\cos(\theta), \sin(\theta))$ back to θ is done inside this ideal model. This transformation is the source of some rounding errors. Errors that take the average error of this model when predicting already known trajectories (as in previous subsection) up to 10^{-5}. All the policy structures (GP,NN,SR) are thus tested on a very good model that is not entirely ideal.

As the policies are optimized thanks to CMA-ES[HO01] (for Gaussian processes policies and neural networks policies) and by genetic programming (for symbolic regression policies) that are both stochastic optimization processes, there are variations of performance among the policies found. It is important to note that neural network policies are not trained thanks to gradient descent. Computing the gradient of the error would impose to derive the reward of a whole trajectory according to the model and would need to have a differentiable reward and a differentiable model. To assess the variability of the policy optimization process, 20 replicates are computed for each policy structure.

Figure 7 shows the total reward achieved by the best policy found so far on the pendulum system. It clearly shows that neural network policies achieve faster, better and less variable policies. Gaussian processes policies are unable to converge as high as the neural network policies even after 15 episodes, their performances still have an important variance at the end of the 15 episodes. Symbolic regression policies are converging to an even lower reward and slower than all others. Since the model is already considered ideal and not being improved at each step, the improvement on the reward only comes from the exploration of the policy space that is done at each step. These episodes could be considered as restarts in the policy search.

Figure 8 shows the total reward achieved by the best policy found so far on the cart-pole system. We can observe a different outcome than for the pendulum system. The symbolic regression policies are better than previously compared to Gaussian processes policies and neural network policies. They achieve a small variability after the 15 episodes. Neural network on the other hand achieve a similar reward here but with a much larger variability than previously. Gaussian processes policies achieve low reward.

These overall low reward achieved for the cart-pole (compared to the reward achieved in Black-DROPS original paper [CRK+17]) could be caused by two things. The first is a modification on the CMA-ES algorithm on this version of the Black-DROPS code, modification that reduces the tendency of CMA-ES to explore and focuses it on local improvement. The second is the small errors that the "ideal" model has. The cart-pole system seems to be very sensitive to small variations, hence, these small error could be the cause of the mismatch between a reward predicted by the model, and real reward. The better result achieved by the symbolic regression policy could come from a important difference between the two tasks that is that the pendulum is under-actuated and needs some planning in order to get to the goal (i.e. balancing the pendulum to load enough mechanical energy) and the cart-pole is sufficiently actuated to take the pendulum attached to

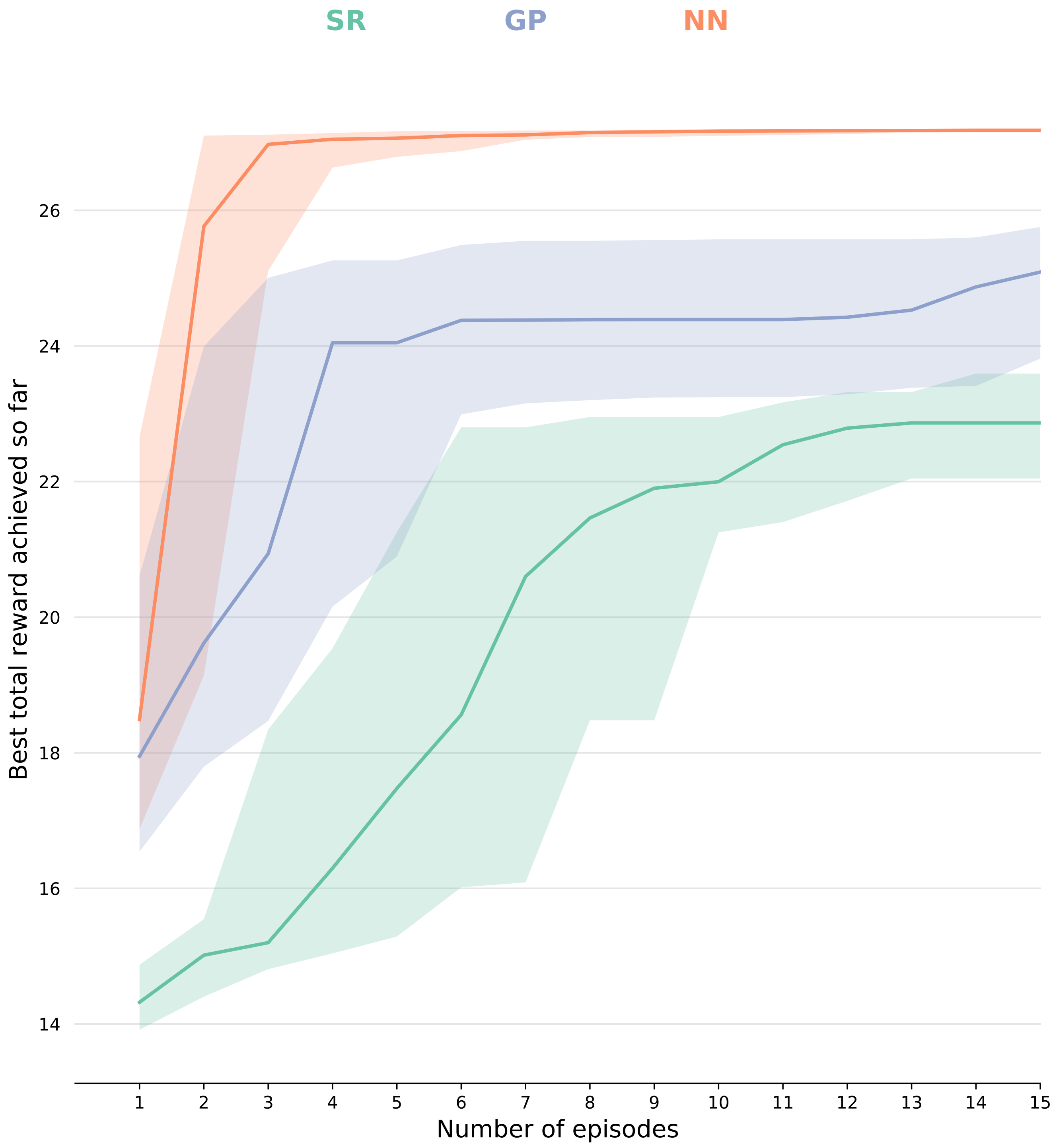

Figure 7: Total reward for an episode of Black-DROPS by the best policy known so far on the pendulum system. Shady area represents the 25th and 75th percentile of a 20 replicate distribution.

it directly on top position without balancing it. Symbolic regression policies could be better for maintaining equilibrium in a system than at reaching a difficult state. From these results we can conclude that neural network policies seems to be at least as efficient as other two policies. Symbolic regression policies and Gaussian processes policies do not fare as well as neural networks policies but achieved similar or slightly lower reward in one of the two tasks.

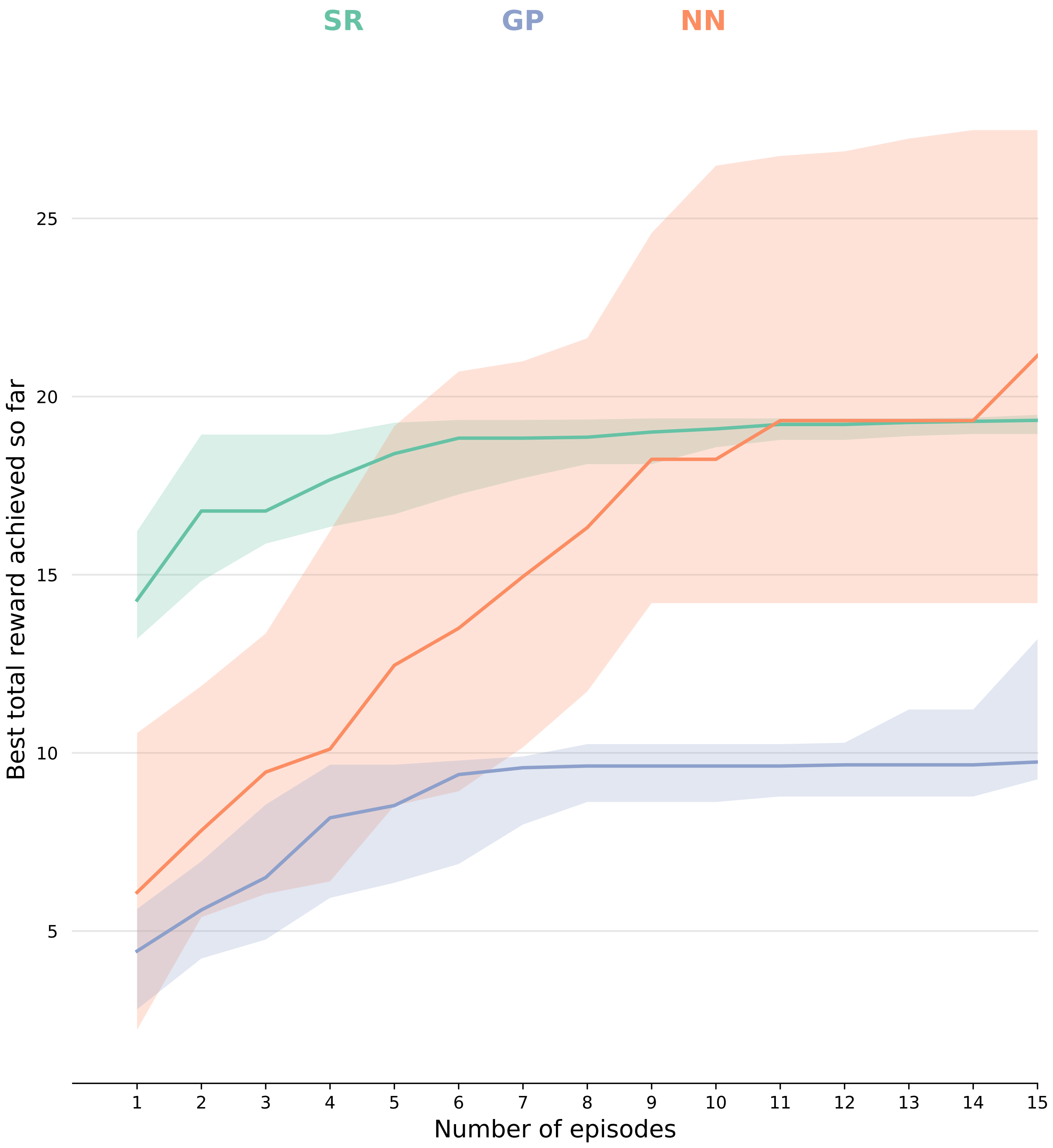

Figure 8: Total reward for an episode of Black-DROPS by the best policy known so far on the pendulum system. Shaded area represents the 25th and 75th percentile of a 20 replicate distribution.

4.3 Black-DROPS

After evaluating the performance of the models and policies independently, their interactions are yet to verify. To do that we run Black-DROPS for 15 episodes with 20 replicates for each of all the combinations of models and policy structure and compare the reward achieved and the time needed. Gaussian processes and neural network policies are still optimized with CMA-ES, neural network models learned with gradient descent and symbolic regression policies and models found with genetic programming.

Figure 9 shows the distribution of the best reward achieved after 15 episodes on the pendulum system. The two best combinations are without question neural network policy with neural network model or Gaussian processes model. These two achieve highest reward with no variability at all, confirming the high performance of neural network policies observed in previous section. Gaussian processes policies achieve a good but not optimal reward with neural network models or Gaussian processes models. However these performance are subject to an important variability. Symbolic regression policies and models achieves the lowest reward overall, with an important variability. However, for symbolic regression models, the highest reward is not achieved thanks to a neural network policy as we could expect according to their good results on the other models, but thanks to a symbolic regression policy. An hypothesis that could explain this information would be that Gaussian processes and neural network policies are overfitting a model that is not good enough, but symbolic regression policies may be constrained by their structure and be less prone to overfitt the errors of symbolic regression models.

Figure 10 shows the distribution of the best reward achieved after 15 episodes on the cart-pole system. Here again the overall reward achieved is not as high as in the original Black-DROPS paper. The modification of the CMA-ES algorithm algorithm is the only hypothesis for this. However we can still observe that only combination with Gaussian processes models achieve a non-random reward. Among these combinations, the neural network policies are achieving the higher reward, becoming close to solving the task. Gaussian processes policies and symbolic regression policies achieve similarly low reward. Symbolic regression and neural network models are unable to provide prediction good enough to find a good policy. This could be expected according to the high error achieved by these models observed on figures 6 and 5.

Figure 11 shows the time needed for the computation of the 15 episodes of Black-DROPS for each couple of models and policies. We can observe that symbolic regression policies and models take more than 10 times longer than other models. For other models and policies the time is around 250s. Figure 12 confirms that symbolic regression models are 10 to 30 times longer to learn.

Figure 13 shows the time needed to optimize policies at each step. The 25th and 75th percentiles are shown as previously. The plot confirms that symbolic regression policies take longer than other policies to optimize. We can also see that the time needed to optimize policy is approximately constant for all combinations except those that contain a Gaussian processes-based model. For these models, the time needed to optimize the policy increases rapidly with the episodes. This is explained by the fact that Gaussian processes queries are at least quadratic in the number of samples they use and this number of samples increases as the episodes go.

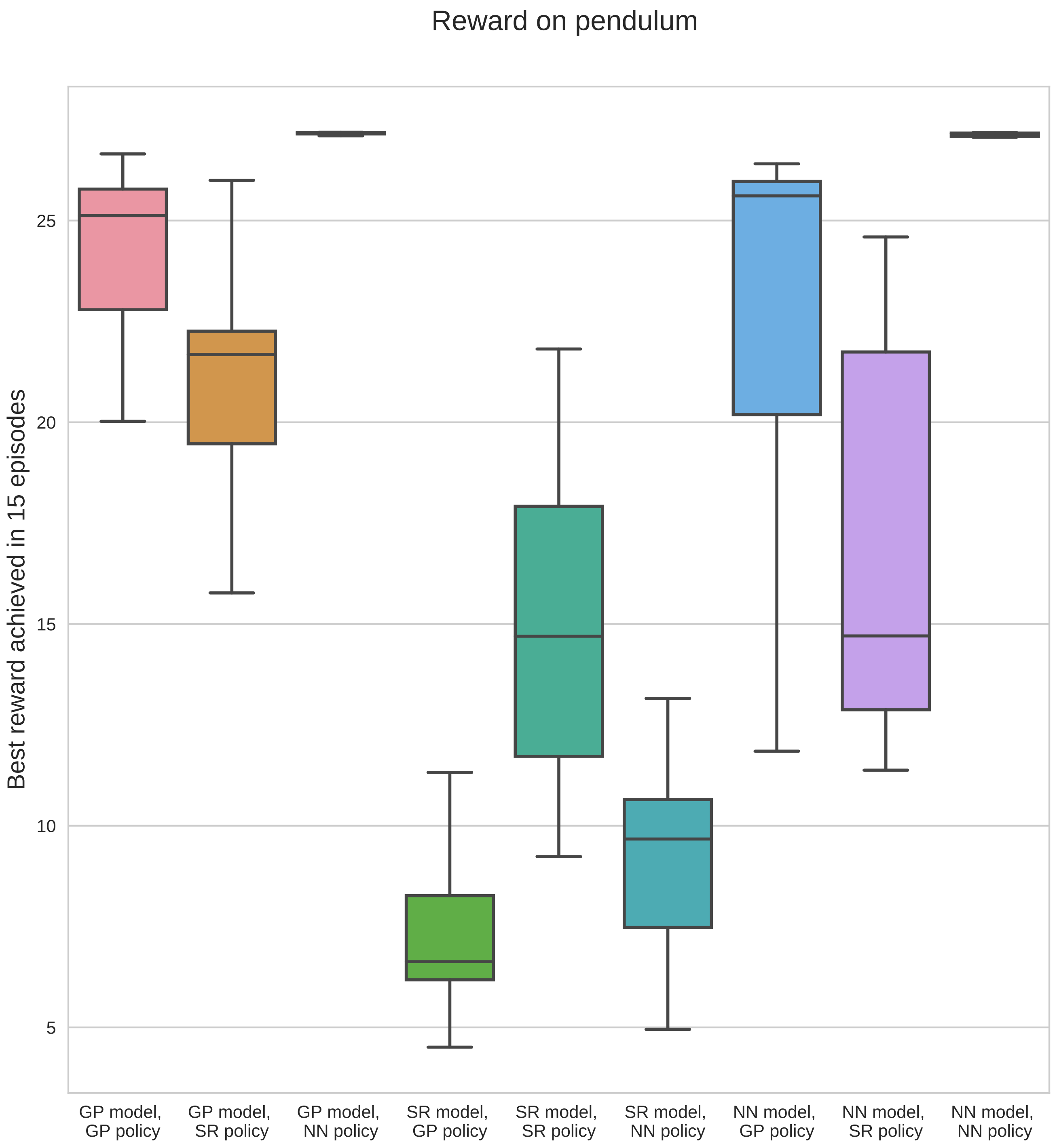

Figure 9: Distribution of the best reward achieved after 15 episodes on the pendulum system for all pairs of models and policies for 20 replicates.

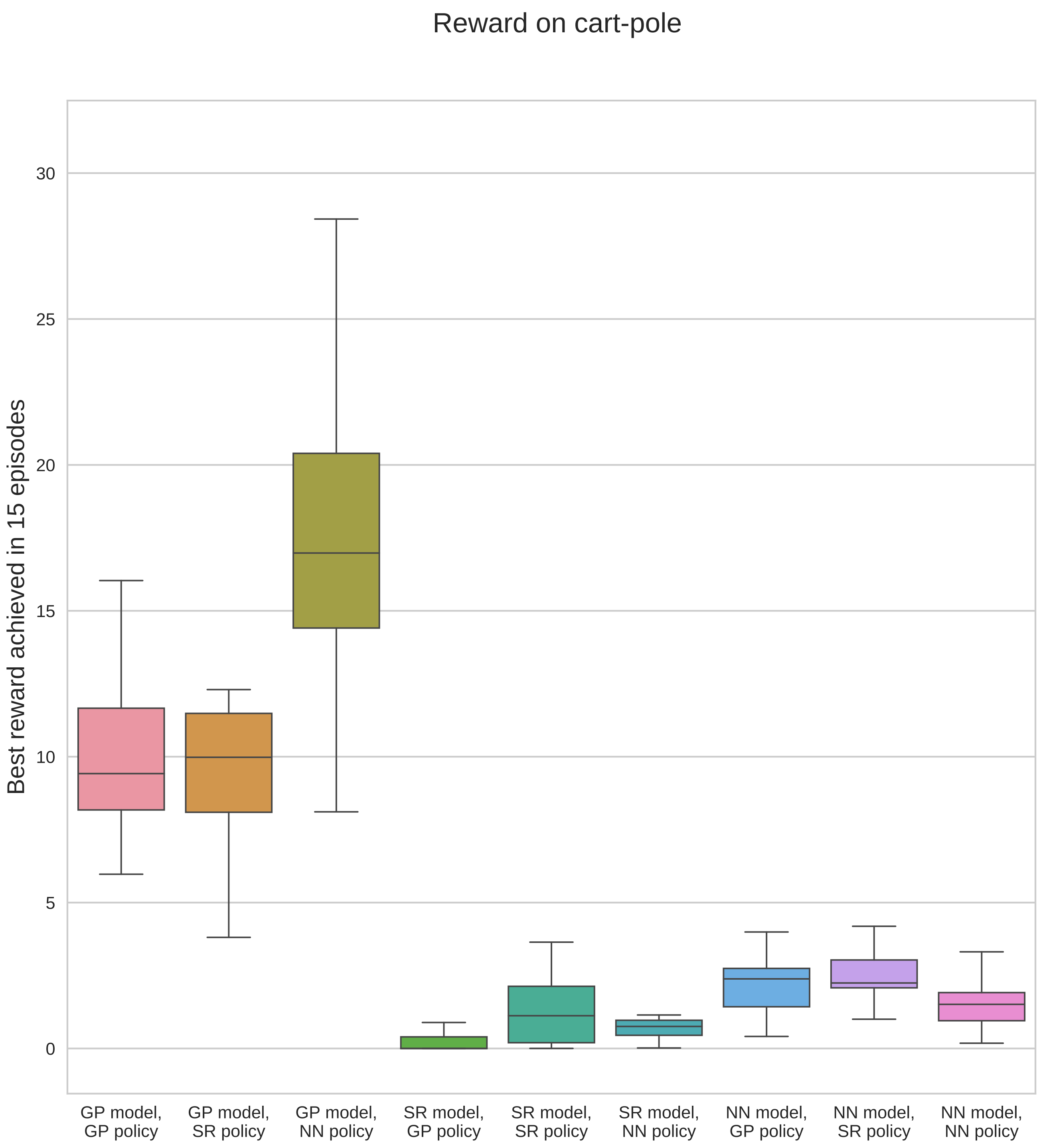

Figure 10: Distribution of the best reward achieved after 15 episodes on the cart-pole system for all pairs of models and policies for 20 replicates.

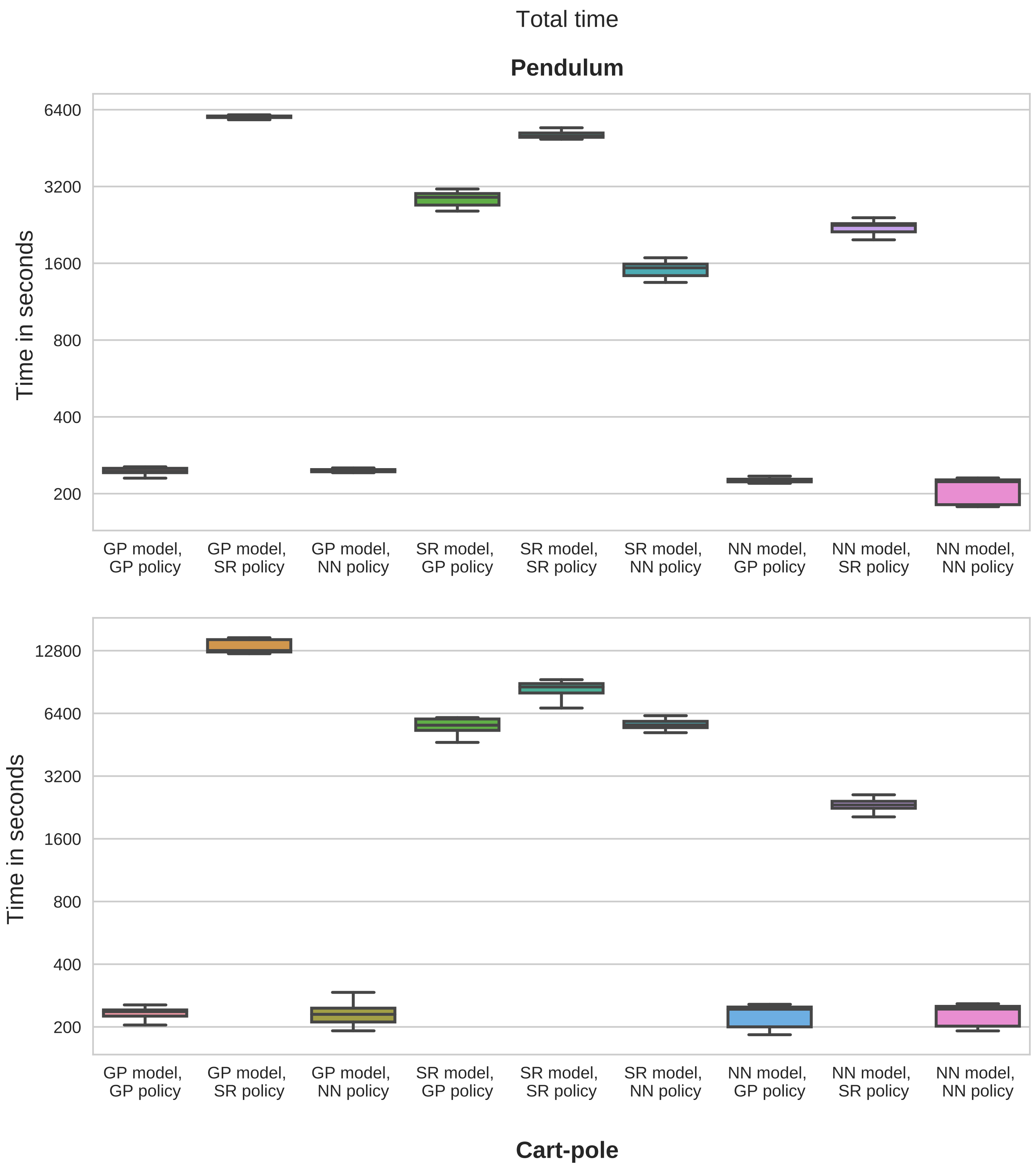

Figure 11: Distribution of the total time needed to complete 15 episodes of Black-DROPS for all pairs of models and policies for 20 replicates. Y scale is logarithmic.

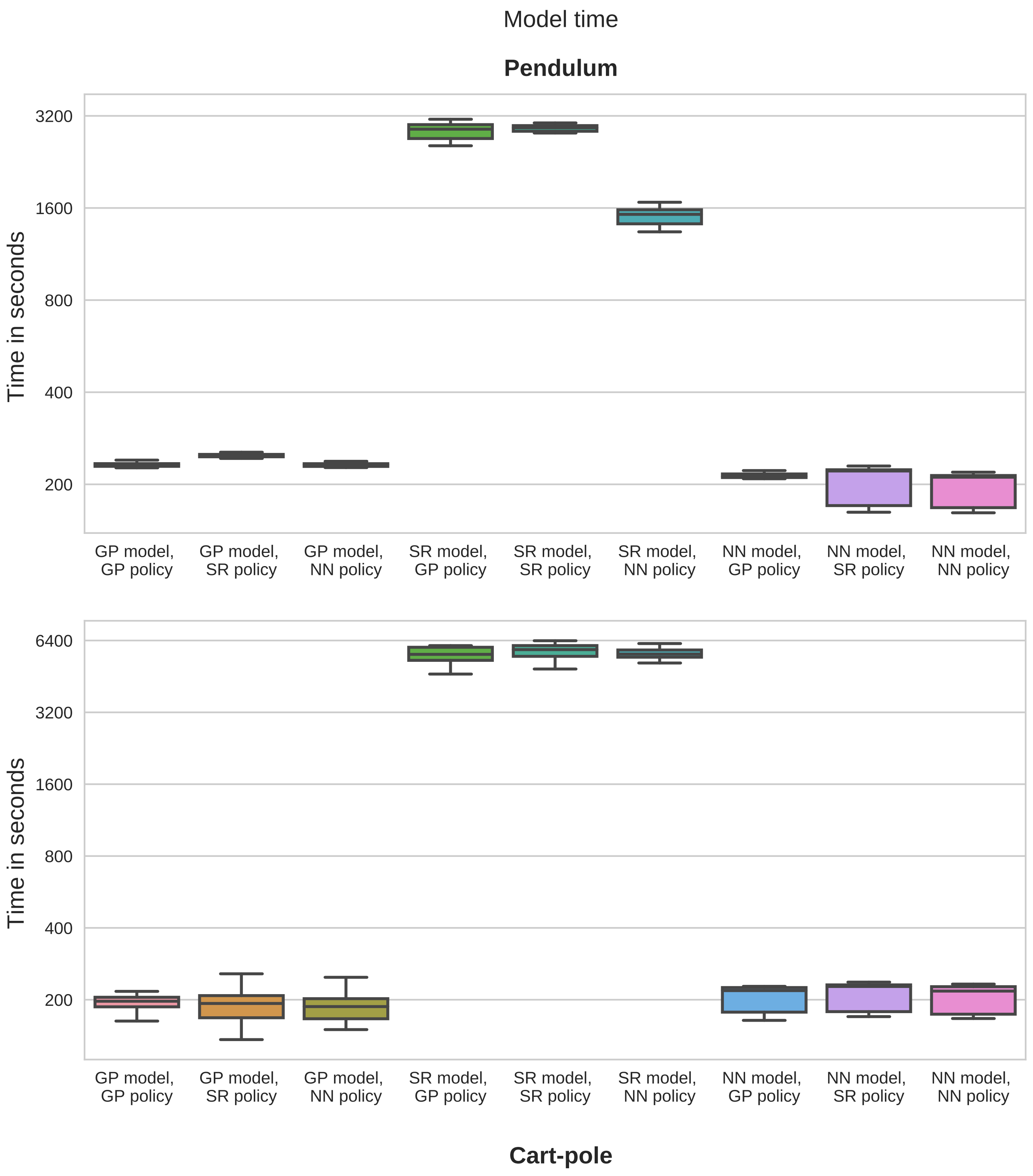

Figure 12: Distribution of the total time needed to complete 15 episodes of Black-DROPS for all pairs of models and policies for 20 replicates. Y scale is logarithmic.

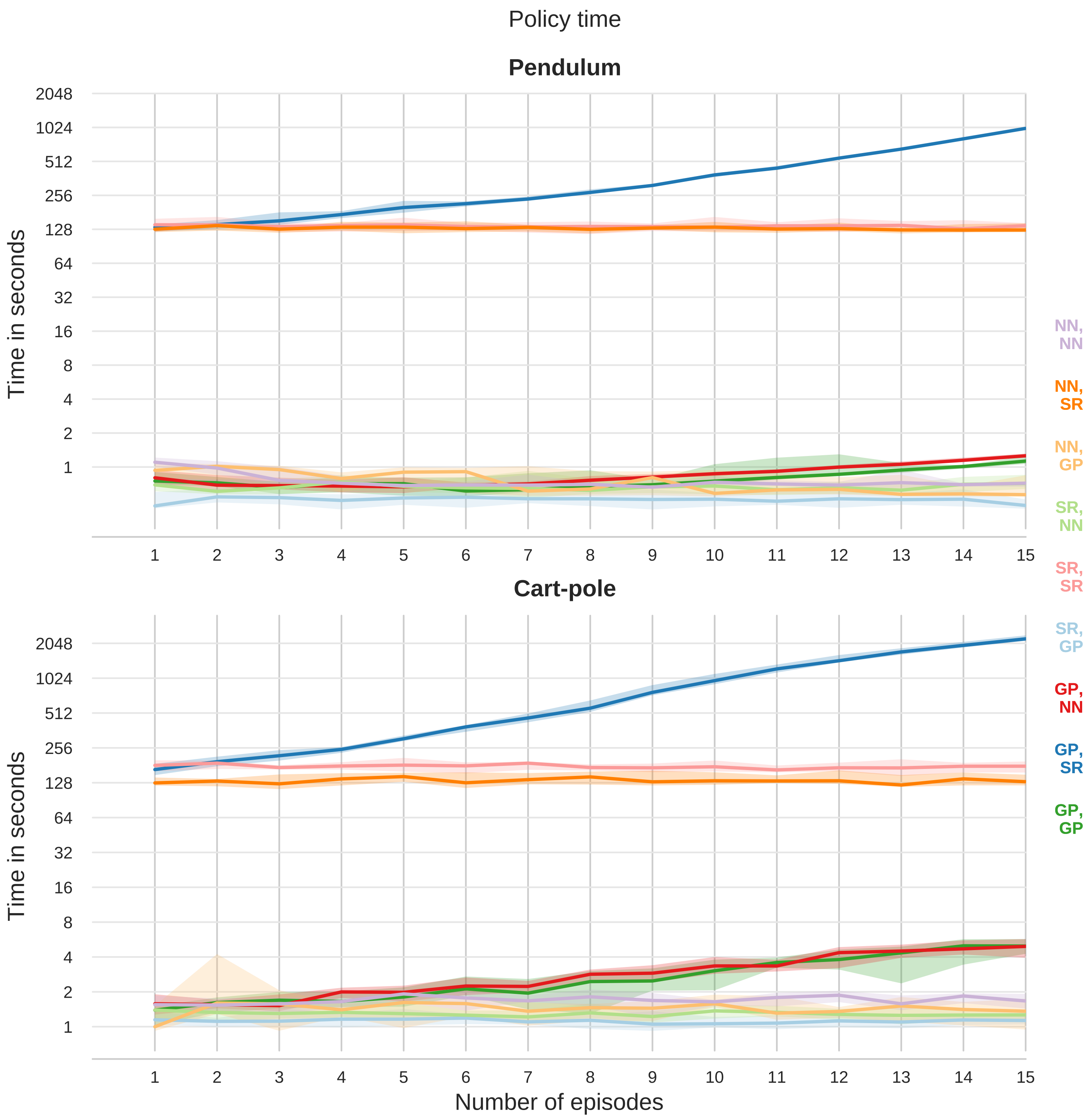

Figure 13: Time needed to optimize policy during each of 15 episodes for all pairs of models and policies. Y scale is logarithmic.

4.4 Found formulas

One reason to try to use symbolic regression as a model or policy is the claim that these model are easier to understand and interpretable 2.3.2. These symbolic regression models were found very good to predict easy variables (e.g. x and θ for pendulum and cart-pole). One example of a found formula for the variation of the angle θ in the pendulum system is:

$$\Delta t \times \dot{\theta} + \frac{\Delta t(\dot{\theta} + \Delta t)}{8 \times 9 \times g^3(\frac{\dot{\theta}}{3} - 6)} \tag{29}$$

What we can see is that it gets the correct formula ($\Delta t \times \dot{\theta}$) plus a second term that is small (the denominator is around 70 000).

For the variation of the position in the cart-pole system one formula found is:

$$\dot{x} \times \sin(\Delta t) - \sin(0) \tag{30}$$

in this formula, the first part is not correct (it should be $\Delta t \times \dot{x}$) but since Δt is close to 0, sine behavior is close to the identity (Taylor expansion of sine around 0 at first order gives identity). The second part is literally equal to 0. Now if we look formulas found for variables that are more difficult to predict but for which prediction still achieve a not so high error, like $\dot{\theta}$ for the pendulum with 20000 generations (see figure 4) we can see this:

$$\sin(\frac{u}{3}) - \frac{\cos(\theta)}{\cos(\sin(\cos(\frac{\sin(\frac{u}{\cos(7)}) - \sin(\frac{\sin(\cos(\theta))}{\cos(\sin(\theta))})}{3})))} \tag{31}$$

This formula is more difficult to explain or understand, even when knowing the actual dynamics (formula 15). Even if this formula is complex, and the link to the true dynamics is difficult to see, its average error is low enough to be kept as model. Because the prediction error of symbolic regression models is really high for the cart-pole, the found formulas for cart-pole difficult variables (i.e. $\dot{x}, \dot{\theta}$) are of little interest.

Symbolic regression policies, even if there is no "true" formula to compare with, do achieve significant reward if their model is good enough, that is shown by the evaluation on the "ideal" model and later by the good reward achieved by the combination Gaussian process model, symbolic regression policy. The following formula is one of the best formulas found for the cart-pole system (with the "ideal" model).

$$\sin((\text{if}(x > 0), \text{then } 0, \text{else } \cos(7 \times \sin(\theta)^2 \times \max(\cos(\theta), \Delta t))) + \sin(\sin(\cos(\theta)))) \tag{32}$$

In this formula again we can read multiple composition of cos and sin that are really difficult to explain. In addition, the branching condition checks the sign the position x and the system is completely symmetric around $x = 0$.

Formulas found for the pendulum policy are not easier to read. The following is one of the shortest:

$$\frac{\cos(\theta)}{\sin(\sin(-4 + \text{if}(\dot{\theta} > 0), \text{then } (8 \times \sin(\sin(\cos(8)))), \text{else}(g)))} \tag{33}$$

The multiple composition of cos and sin are still there and some are even applied to constants. The branching condition could make sense has it outputs values of different sign (the "then" output is negative) depending on the sign of the angular velocity. The later subtraction of 4 to this output shift to output to a choice between -4 and $+5$ for the denominator. Many other formulas found are even more convoluted, making the interpretability claimed not as immediate as human written formulas would be, but still much more accessible to read than a neural network would be. These formulas may not be directly the good policy but can provide interesting ideas on which parameters to control the system like in 33.

5 Conclusion

During this internship I implemented a symbolic regression variant for the Black-DROPS algorithm. The goal was to replace Gaussian processes dynamical models by another model less computationally expensive to query and easier to interpret and investigate to advantages of using genetic programming technique to find good policies. To implement this variant, I wrote a extension to the evolutionary algorithm library SFERESv2 [MD10] to do genetic programming and tree manipulation.

We found that symbolic regression models are good to predict "easy dynamics" but are unable to beat Gaussian processes when the dynamics are slightly more complex than trivial. It was impossible to learn a good model of the cart-pole system. However, symbolic regression models are on par with neural networks for dynamics prediction in the context of micro data. Symbolic regression models are certainly less expensive to query than Gaussian processes, but the time needed to learn symbolic regression processes is more than one order of magnitude higher than for Gaussian processes and neural network models. The interpretability that was expected of symbolic regression models was not easy to use as the formulas found contains many irrelevant terms even when it would be possible to find a small and precise formula.

Symbolic regression policies proved to be not as good as neural network policies, but they achieved sometimes similar reward to Gaussian processes policies. As for the models, the symbolic regression policies take much longer to optimize than the two other policy structures (neural networks and Gaussian processes). The interpretability was relative as some formulas found are partly meaningful and some others do show behavior and branching criteria with no link to the symmetries of the task. However, it is still easier to read and write these formulas than to read or write the behavior of a neural network.

These difficulties to find good symbolic regression models and policies might be caused by two main reasons. The first is the very large size of the sets of formulas and hence the difficulty to explore it enough to find a good policy in a limited time, moreover, formulas that are syntactically close may have really different behavior, making the exploration even more difficult. The second is that there is no uncertainty mechanism as in Gaussian processes helping the policy search to not overfitt the model errors. The inability to find interpretable and meaningful policies and model may be caused by the bloating for which the implemented prevention mechanisms mechanism were not sufficient.

5.1 Future works

It may be possible to tackle the issues mentioned with some extensions of this implementation. In order to allow easier exploration, the use of local optimization might help to find good constant [AA15, LCDS16, TP01]. An approach to add optimal constant finding for each individual by gradient descent might speed up the exploration by letting genetic programming find the structure of the formula and gradient descent find correct value for the constants composing this formula. The analytical computation of the gradient of the error is possible thanks to the use of the chain rule as it is done for neural networks. Encoding some priors[JB15] into formula generation might be another way to go, for example to prevent multiple trigonometry composition. The automatic simplification of some parts of the formula, for example replacing sub-trees that contain no variable (and thus are constant) by a constant equal to the value of this sub-tree, would reduce the size of the set of formulas making exploration easier as well as output simpler formulas, helping the interpretability.

The initial mechanism of uncertainty that was planned but not implemented because of a lack of time might still improve the robustness of symbolic regression models. As the symbolic regression models seems to not need a lot of data, this use of bootstraps sets that contain small fraction of the whole training set could be done with lesser risks of inadequate learning.

Alternatively, to manage noise over the observed state, each variable may be associated to a variation range that would propagate at formula evaluation and provide a interval of possible output values.

It might be interesting to examine more carefully the impact of the integration scheme over the ability for symbolic regression to find reliable models.

References

References

[AA15] Aliasghar Arab and Alireza Alfi. An adaptive gradient descent-based local search in memetic algorithm applied to optimal controller design. *Information Sciences*, 299:117–142, 2015.

[CCAM16] A. Cully, K. Chatzilygeroudis, F. Allocati, and J.-B. Mouret. Limbo: A flexible high-performance library for gaussian processes modeling and data-efficient optimization. *Preprint*, 2016.

[CRK$^+$17] Konstantinos Chatzilygeroudis, Roberto Rama, Rituraj Kaushik, Dorian Goepp, Vassilis Vassiliades, and Jean-Baptiste Mouret. Black-Box Data-efficient Policy Search for Robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2017.

[CVM18] Konstantinos Chatzilygeroudis, Vassilis Vassiliades, and Jean-Baptiste Mouret. Reset-free trial-and-error learning for robot damage recovery. *Robotics and Autonomous Systems*, 100:236–250, 2018.

[Deb01] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[DEW02] John Duffy and Jim Engle-Warnick. Using symbolic regression to infer strategies from experimental data. In *Evolutionary computation in Economics and Finance*, pages 61–82. Springer, 2002.

[DFR15] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423, 2015.

[DJWP01] Edwin D De Jong, Richard A Watson, and Jordan B Pollack. Reducing bloat and promoting diversity using multi-objective methods. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 11–18. Morgan Kaufmann Publishers Inc., 2001.

[EGW05] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

[ET94] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[Han09] Nikolaus Hansen. Benchmarking a bi-population cma-es on the bbob-2009 function testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2389–2396. ACM, 2009.

[Hay94] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[HO01] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[Hol92] John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

[HSM15] Thomas Helmuth, Lee Spector, and James Matheson. Solving uncompromising problems with lexicase selection. *IEEE Transactions on Evolutionary Computation*, 19(5):630–643, 2015.

[HUR17] Daniel Hein, Steffen Udluft, and Thomas A Runkler. Interpretable policies for reinforcement learning by genetic programming. *arXiv preprint arXiv:1712.04170*, 2017.

[JB15] Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428, 2015.

[Koz94] John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

[Kri51] Daniel G Krige. A statistical approach to some basic mine valuation problems on the witwatersrand. *Journal of the Southern African Institute of Mining and Metallurgy*, 52(6):119–139, 1951.

[KTB13] Dirk P Kroese, Thomas Taimre, and Zdravko I Botev. *Handbook of Monte Carlo methods*, volume 706. John Wiley & Sons, 2013.

[LCDS16] William La Cava, Kourosh Danai, and Lee Spector. Inference of compact nonlinear dynamic models by epigenetic local search. *Engineering Applications of Artificial Intelligence*, 55:292–306, 2016.

[Lip16] Zachary Chase Lipton. The mythos of model interpretability. *CoRR*, abs/1606.03490, 2016.

[MC15] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

[MD10] J.-B. Mouret and S. Doncieux. SFERESv2: Evolvin' in the multi-core world. In *Proc. of Congress on Evolutionary Computation (CEC)*, pages 4079–4086, 2010.

[MFWE12] Francis Maes, Raphael Fonteneau, Louis Wehenkel, and Damien Ernst. Policy search in a space of simple closed-form formulas: Towards interpretability of reinforcement learning. In *International Conference on Discovery Science*, pages 37–51. Springer, 2012.

[MKS$^+$15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[NJ00] Andrew Y Ng and Michael Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc., 2000.

[PE08] Kamban Parasuraman and Amin Elshorbagy. Toward improving the reliability of hydrologic prediction: Model structure uncertainty and its quantification using ensemble-based genetic programming framework. *Water Resources Research*, 44(12), 2008.

[Pre07] William H Press. *Numerical recipes 3rd edition: The art of scientific computing.* Cambridge university press, 2007.

[RHW$^+$88] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[RW06] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[SB11] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[SL09] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms.* Cambridge university press, 2014.

[TP01] Alexander Topchy and William F Punch. Faster genetic programming based on local gradient search of numeric leaf values. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 155–162. Morgan Kaufmann Publishers Inc., 2001.

[VM18] Vassilis Vassiliades and Jean-Baptiste Mouret. Discovering the elite hypervolume by leveraging interspecies correlation. *arXiv preprint arXiv:1804.03906*, 2018.