Bayesian Optimisation using an Ensemble of Neural Networks

Thomas M. Pethick

***Abstract*—State-of-the-art hyperparameter optimization of machine learning algorithms has been achieved with Bayesian Optimization using Gaussian Processes (GPs). However, computational cost for this model scales cubically in the number of observations. Recent methods replaces the GP with a Bayesian linear regressor on basis functions learned by a neural network – an approach that scale linearly. The objective is to explore where this method fails and investigate whether an ensemble of neural networks improves convergence in those scenarios. The preliminary investigation suggests that an ensemble is beneficial in problems with low effective dimensionality and in spaces of sufficient dimensionality (i.e. 6 or above).**

1. Introduction

The performance of machine learning algorithms is heavily dependent on the configuration of so-called *hyperparameters*. In particular, for a deep neural network architecture this could be the number of units for each layer, the number of layers and the activation functions. We can treat the possible collection of hyperparameters as a space which we will call the *hyperparameter space*. In this framework the objective is to find the point in this space that yields the best performance. What we have done is to frame it as an optimization problem, more precisely formulated mathematically as finding the optimal configuration,

$$\boldsymbol{x}^\star = \arg\max_{\boldsymbol{x}=\mathcal{X}} f(\boldsymbol{x}). \tag{1}$$

where $\mathcal{X}$ in this particular instance is the hyperparameter space and f is a function providing a measure of performance of the machine learning algorithm for a a certain hyperparameter configuration.

This problem belongs to an interesting subclass of optimization problems. First of all, f is very expensive to evaluate. In our particular instance it is required to run the machine learning algorithm to completion to evaluate f. This could take days for a single evaluation considering the training required for recent deep learning methods. Secondly, f is considered a "black-box" in the sense that it has no special structure such as linearity or concavity that would make the problem easy and no first- or second-order derivative is observed that would similarly simplify the problem.

Several solutions have been proposed to this problem. Naive proposals include non-adaptive algorithms of which Random Sampling is one. It samples N points uniformly at random from $\mathcal{X}$ and picks the one that optimizes $f(\cdot)$. Another similarly simple algorithms is the Grid Search which instead selects the N points from a grid – more precisely the cartesian product between a finite subset of each dimension.

If we instead query f sequentially we can make an informed choice of the next point to evaluate given the already observed evaluations. These methods are commonly referred to as *Bayesian Optimization* since the way they update their belief about what point to pick based on previous observation is fundamentally Bayesian. The assumption of expensive evaluation justifies spending computational effort on making informed choices.

Numerous suggestions on how to model the belief of f exists with most methods capturing it (at least partially) using *Gaussian Processes* (GPs). Examples of this includes Spearmint (Snoek, Larochelle, and Adams 2012) with a purely GP based approach, SMAC (Hutter, Hoos, and Leyton-Brown 2011) using Random Forest and HyperOpt (Bergstra et al. 2011) using Tree-structured Parzen Estimator. These methods are computationally restricted by an expensive cost of $\mathcal{O}(n^3)$ for n observations.

More recently, methods using a Bayesian linear regressor on features from a neural network were proposed to obtain *scalable* Bayesian Optimization (Snoek et al. 2015). This was coined Deep Network for Global Optimization (DNGO).

It is a well-known fact that the random initialization of neural network weights can have a significant impact on the networks performance. To mitigate this effect it is natural to consider a common statistical method called bagging in which an ensemble of the model is aggregated. Thus, we propose an extension of DNGO that uses an ensemble of neural networks.

1.1. Contributions

Our contributions are two-fold:

- The behaviour of DNGO is badly understood. We first present a qualitative analysis based on several categorized benchmarks in order to find failure cases in which an ensemble might help.

- Secondly, we propose a generalization of DNGO which uses an ensemble of neural network and investigate in which scenarios this is advantageous.

1.2. Related work

Another extension of DNGO has been proposed in which a distribution is places on all weights thus obtaining a Bayesian neural network (Springenberg et al. 2016). Even though it similarly attempts to make the method more robust it does so through other means. It is natural to compare these methods however.

More recently it was proposed in (Li et al. 2016) that 2×Random Search (i.e. allowing to sample twice as many points) where competitive to Bayesian Optimization methods suggesting much simpler algorithms. However, experimental results in (Golovin et al. 2017) suggest that this is only the case in sufficiently high dimensions (specifically 16 or more).

1.3. Overview

The paper will be structured as follows. First the necessary background will be covered in section 2 consisting of Bayesian Optimization specifically with a gaussian processes and the neural network approach referred to as DNGO. We are then equipped in section 3 to specify the three categories of models being tested: GP, DNGO and ensembled DNGO. The results are then tested on several benchmarks and hyperparameter optimization tasks covered in Section 4. We follow up by a discussion in section 5 and close with a conclusion in section 6.

2. Background

2.1. Bayesian Optimization

Bayesian Optimization is a sequential optimization strategy well-suited for finding an optimum in few iterations. It does so by incorporating a prior belief of the objective function f and subsequently refining this through a Bayesian posterior as observations are made. The posterior now expresses our updated belief of the objective function given data. To know where to query the next observation an *acquisition function* can be induced on the statistical model that guides exploration by using the uncertainty in the posterior. Thus the next point $\boldsymbol{x}_{t+1}$ at time step t is selected by maximizing the acquisition function. Figure 1 illustrates three steps of such a process where the input space is $\mathbb{R}^1$.

To summaries the process involves two components, namely the prior over the objective function and the acquisition function. Since the acquisition function is defined based on the posterior function which arises from the prior, we will leave this topic until section 2.3.

Let $\mathcal{D}_{1:t} = \{ \boldsymbol{x}_{1:t}, y_{1:t} \}$ be the sequence of observation up to time step t in the Bayesian Optimization procedure. We combine the *prior distribution* $P(f)$ with the *likelihood function* $P(D_{1:t} \mid f)$ to obtain the *posterior distribution*,

$$P(f \mid \mathcal{D}_{1:t}) = \frac{P(\mathcal{D}_{1:t} \mid f)P(f)}{P(\mathcal{D}_{1:t})} \propto P(\mathcal{D}_{1:t} \mid f)P(f). \quad (2)$$

This is essentially what makes it Bayesian since it is an application of Bayes rule. Notice that we can drop the normalization constant given it is irrelevant for an optimization task. This posterior is sometimes also referred to as the *response surface* or *surrogate function* since it estimates the objective function.

One commonly used prior over f is the Gaussian Process covered in section 2.2. It is interesting since it yields an analytical expression for the posterior as well as for other reasons which will become apparent in the subsequent section.

For a more complete treatment and overview of BO we refer to (Shahriari et al. 2016).

2.2. Gaussian Processes

What we have done by treating our belief about f as a distribution over function and applying Bayes rule to obtain the posterior is essentially what is called *nonlinear Bayesian regression*.

A GP is one possible prior over functions $P(f)$.

Definition 2.1. A Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution (Rasmussen and Williams 2006).

This can be viewed as an extension of the multivariate gaussian distribution to infinite dimensions. The GP is a distribution over functions completely specified by its mean function m and covariance function k similarly to how a gaussian distribution is a distribution over a random variable fully specified by its mean and covariance. We write it,

$$f(\boldsymbol{x}) \sim \mathcal{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')). \quad (3)$$

To understand how it is a distribution over functions consider the GP to be defined on the continuous space $\mathbb{R}^D$ (consequently the GP assigns a random variable for every index $\boldsymbol{x}$ in the infinite index set $\{ \boldsymbol{x} \in \mathbb{R}^D \}$). Then $f(\boldsymbol{x})$ is the random variable for every location $\boldsymbol{x} \in \mathbb{R}^D$. Thus sampling from this generative model gives you one possible function over the domain $\mathbb{R}^D$.

2.2.1. Covariance Function. The covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$ (also known as the *kernel*) captures the correlation between points in the process and thus determines the structure of possible functions. Notice, however, that it specifies the similarly between the *function evaluation* of two points even though it is determined by the input. A common choice of kernel is the *Squared Exponential* (SE),

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_0^2 \exp(-\frac{1}{2l^2}|\boldsymbol{x} - \boldsymbol{x}'|^2). \quad (4)$$

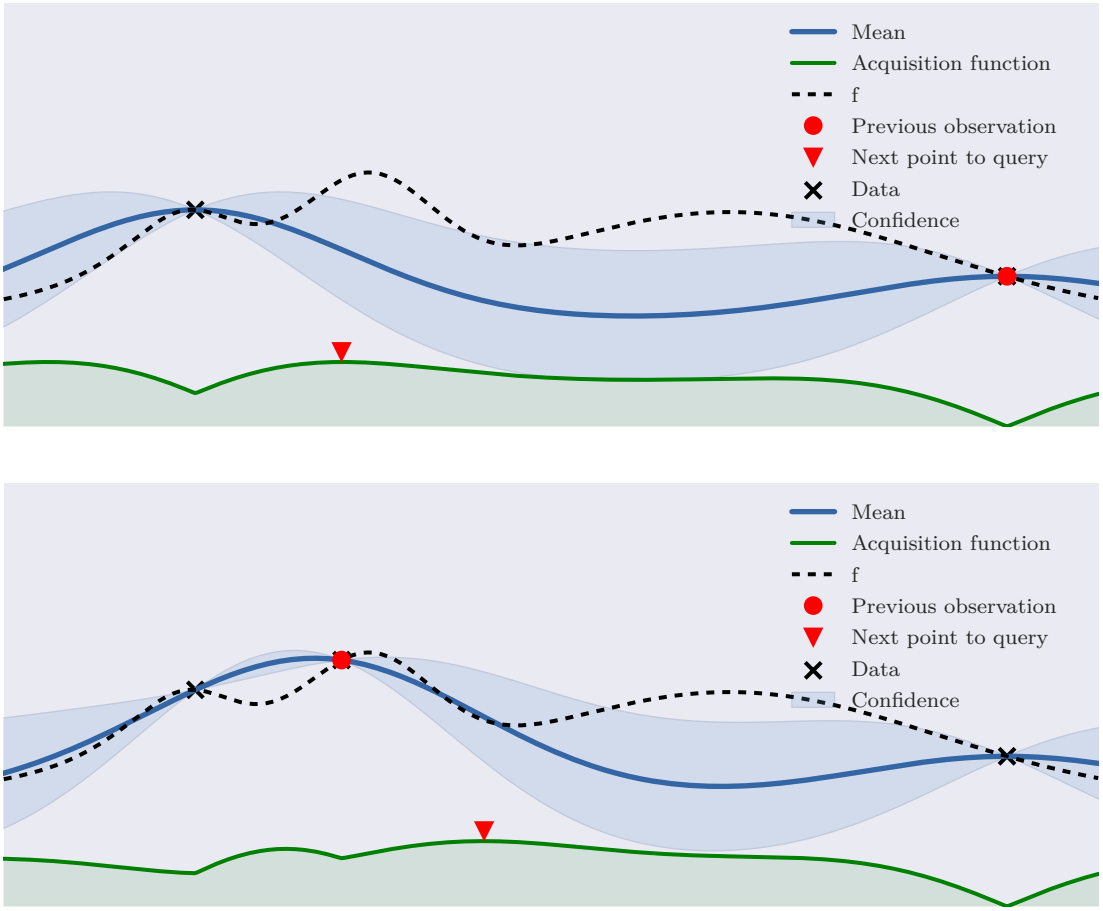


Figure 1: Two steps of Bayesian Optimization in which there are initially two observation. At every step the posterior distribution is derived and the acquisition function is maximized to determine the next point to query. Exactly how the acquisition function is connected to the mean and variance is covered section 2.3.

It has two so-called *hyperparameters*, the output variance σ_0^2 and l referred to as the *length scale*. The parameter σ_0^2 is simply a scale factor that controls how far away from the function mean value varies, and is thus neglectable considering normalization of the output. More interesting is the length scale. The effect on sampled functions from a GP based on the SE kernel using different length scales is apparent in fig. 2. Intuitively, the length scale roughly corresponds to the distance necessary to move for a change to occur in the function value. Thus samples from gaussian process prior with $l = 0.02$ fluctuates the most. Obviously, these hyperparameters have a big influence on how well we model the unknown function f. In section 2.2.4 we will cover various methods on how to learn these directly from data.

The kernel has a few characteristics.

First, notice how smoothly the sampled function varies in fig. 2. This is a consequence of every prior of SE being infinitely differentiable. This is an important property since it becomes problematic when we want to model variables that are originally discrete (see section 5).

Secondly, if we disregard the output variance, the covariance between two points specified by the above kernel is almost 1 for very close inputs but decreases as the distance grows effectively capturing that inputs close to each other have high influence on each other. This assumption is essential in allowing us to estimate which areas have potential in e.g. hyperparameter optimization.

Lastly, it should be noted that SE belongs to a whole class of kernels called *stationary kernels* since they are translationally invariant. This becomes apparent by observing that the kernel can be written as a function of the difference $|\boldsymbol{x} - \boldsymbol{x}'|$. We will return to the importance of this in section 5 as well.

Now, returning to the visualisation, the samples in fig. 2 where drawn using the *covariance matrix* which we denote by $\boldsymbol{K}$. Given a set of points X, $\boldsymbol{K}$ can be

constructed from the covariance function by element-wise application for all pairs of points,

$$\boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_t) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_t, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_t, \boldsymbol{x}_t) \end{bmatrix} \tag{5}$$

This notation will be useful when we discuss *GP regression* in section 2.2.3.

The literature on covariance function is rich including isotropic, non-stationary (such as periodic) and ways to combine kernels (Duvenaud 2014). For a thorough overview we refer to (Rasmussen and Williams 2006, ch. 5).

2.2.2. Prior Mean. So far we have only discussed the prior covariance effect on the GP by so far assuming zero mean throughout the input space. It provides a principled way of incorporating expert knowledge as done by DNGO covered in section 2.4. In practice the prior mean is mostly constant thus leaving the posterior mean to be inferred through the likelihood of the data.

2.2.3. GP Regression. For Bayesian Optimization we are interested in estimating the function value at an unobserved point based on the statistical model. This is a Bayesian regression problem and specifically with a gaussian prior it is referred to as GP regression.

As exemplified by fig. 2 it is possible to sample from the prior GP. Recall that by the definition of GP a (finite) set of points induces a joint gaussian distribution. Thus sampling from a GP prior $\mathcal{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$ is simply done through a gaussian distribution with covariance matrix $\boldsymbol{K}$ constructed from $k(\boldsymbol{x}, \boldsymbol{x}')$,

$$f \sim \mathcal{N}(\mu(\boldsymbol{x}), \boldsymbol{K}). \tag{6}$$

We will now consider how to predict a new sample from a learned posterior function, i.e. define the *predictive posterior distribution*. This will be done for only single predictions, since this is sufficient in the context of Bayesian Optimization (see (Rasmussen and Williams 2006) for generalization).

Assume we have observed $\mathcal{D}_t = \{ \boldsymbol{x}_{1:t}, \mathbf{f}_{1:t} \}$ for which we have constructed the covariance matrix $\boldsymbol{K}$ and we wish to predict f_{t+1} at $\boldsymbol{x}_{t+1}$.

To get a distribution over the prediction let us first imagine that all $\mathbf{f}_{1:t+1}$ are drawn at the same time. Again, by the property of the GP they are sampled from a joint Gaussian distribution,

$$\begin{bmatrix} \mathbf{f}_{1:t} \\ \mathrm{f}_{t+1} \end{bmatrix} \sim \mathcal{N}(\mu(\boldsymbol{x}_{1:t+1}), \begin{bmatrix} \boldsymbol{K} & \boldsymbol{k} \\ \boldsymbol{k}^T & k(\boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+1}) \end{bmatrix}), \tag{7}$$

where $\boldsymbol{k} = [k(\boldsymbol{x}_{t+1}, \boldsymbol{x}_1), \cdots, k(\boldsymbol{x}_{t+1}, \boldsymbol{x}_t)]$. The primary difference is that the covariance matrix have been expanded to capture correlation between f_{t+1} and all previous evaluations.

From here it is relatively straightforward to condition on $\mathbf{f}_{1:t}$, thus arriving at the predictive posterior distribution (see (Rasmussen and Williams 2006, A.2) for the full derivation),

$$\mathrm{f}_{t+1} \mid \boldsymbol{x}_{t+1}, \boldsymbol{x}_{1:t}, \mathbf{f}_{1:t} \sim \boldsymbol{N}(\mu_t(\boldsymbol{x}_{t+1}), \sigma_t(\boldsymbol{x}_{t+1})^2) \tag{8}$$

where

$$\mu_t(\boldsymbol{x}_{t+1}) = \boldsymbol{k}^\top \boldsymbol{K}^{-1} \mathbf{f}_{1:t} \tag{9}$$

$$\text{and } \sigma_t(\boldsymbol{x}_{t+1})^2 = k(\boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+1}) - \boldsymbol{k}^\top \boldsymbol{K}^{-1} \boldsymbol{k}. \tag{10}$$

So the predictive posterior distribution is likewise gaussian, meaning $\mu_t(\cdot)$ and $\sigma_t^2(\cdot)$ are sufficient statistics.

Returning to the Bayesian Optimization setting, our statistical model now captures a mean and variance over all points in the input space which is sufficient for defining the acquisition function.

Noisy function evaluation. In many application of Bayesian Optimization it is only possible to access f through some noisy evaluation.

We add Gaussian noise $\epsilon_i \overset{iid}{\sim} \mathcal{N}(0, \sigma^2_{\text{noise}})$ to function evaluations $y_i = \mathrm{f}_i + \epsilon_i$. Denote observations $\mathcal{D}_t = \{ \boldsymbol{x}_{1:t}, \mathbf{y}_{1:t} \}$ and the next point $\boldsymbol{x}_{t+1}$ with function evaluation $\mathbf{y}_{t+1}$.

The likelihood of $\mathbf{y}_{1:t+1}$ thus becomes,

$$\mathbf{y}_{1:t+1} \mid \mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma^2_{\text{noise}} \boldsymbol{I}). \tag{11}$$

Integrate out $\mathbf{f}$ to get the marginal likelihood,

$$P(\mathbf{y}) = \int P(\mathbf{y} \mid \mathbf{f}) P(\mathbf{f}) \, \mathrm{d}\mathbf{f} = \mathcal{N}(\mu(\boldsymbol{x}_{1:t+1}), \boldsymbol{K} + \sigma^2_{\text{noise}} \boldsymbol{I}). \tag{12}$$

From here the same method used to get (8) by conditioning on $\mathcal{D}_t$ can be applied to obtain the predictive posterior distribution,

$$\mathrm{y}_{t+1} \mid \mathbf{y}_{1:t} \sim \mathcal{N}(m_t(\boldsymbol{x}_{t+1}), \sigma_t^2(\boldsymbol{x}_{t+1})) \tag{13}$$

where

$$\mu_t(\boldsymbol{x}_{t+1}) = \boldsymbol{k}^\top (\boldsymbol{K} + \sigma^2_{\text{noise}} \boldsymbol{I})^{-1} \mathbf{y} \tag{14}$$

$$\text{and } \sigma_t^2(\boldsymbol{x}_{t+1}) = k(\boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+1}) - \boldsymbol{k}^\top (\boldsymbol{K} + \sigma^2_{\text{noise}} \boldsymbol{I})^{-1} \boldsymbol{k}. \tag{15}$$

What have changed from the noise free setting is simply the diagonal σ^2_{noise} term in the kernel and that we now observe $\boldsymbol{y}$ instead of $\boldsymbol{f}$ directly.

Running time. We have obtained an analytical expression for the predictive posterior distribution thus providing well-calibrated uncertainty estimates. However, the cost of this exact inference is $\mathcal{O}(n^3)$ where n is number of observations. The cost is due to the need for inverting the covariance matrix $\boldsymbol{K}$ with size $n \times n$. By storing $\boldsymbol{K}^{-1}$ the mean and variance can be calculated in $\mathcal{O}(n^2)$ and $\mathcal{O}(n)$.

New samples in the Bayesian Optimization setting could easily be handled by adding the additional row

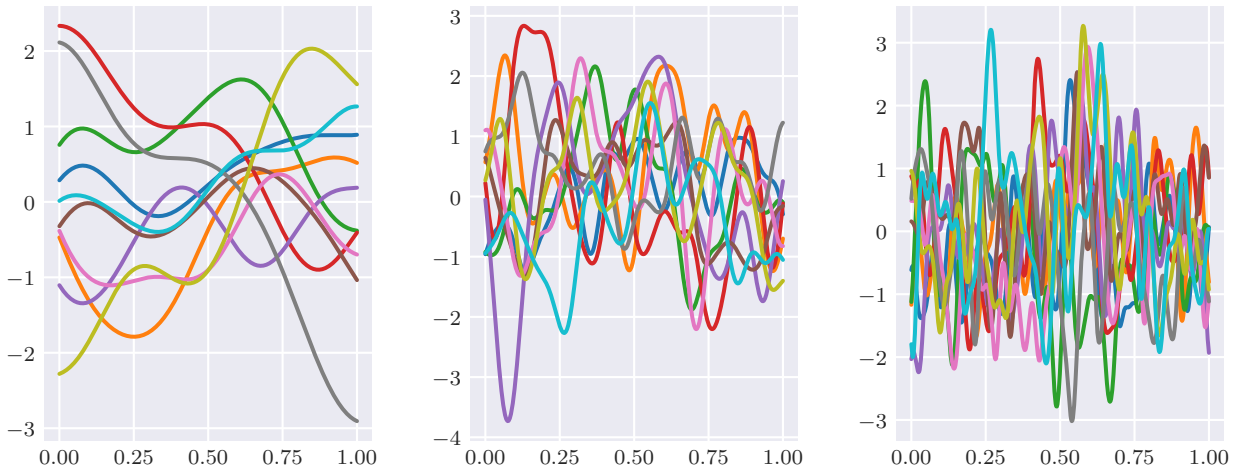


Figure 2: Each plot contains 10 sampled functions drawn form a gaussian process prior with SE covariance function using a length scale of 0.2, 0.05 and 0.02 from left to right. Notice how the length scale intuitively governs how quickly the function value changes.

and column for every update so subsequent predictions becomes $\mathcal{O}(n^2)$. However, $\boldsymbol{K}^{-1}$ would have to be recomputed everytime the hyperparameters of the kernel are updated which is desireable to do based on data as covered in section 2.2.4.

This computational constraint is the motivation for DNGO introduced in section 2.4.

2.2.4. Choosing hyperparameters. It became apparent in section 2.2.1 that the hyperparameters of the kernel have prevailing impact on the type of function the GP describes. This section will cover how to infer these hyperparameters θ from observations $\mathcal{D}_t = \{ \boldsymbol{x}_{1:t}, \mathbf{y}_{1:t} \}$.

MLE and MAP. The simplest solution to consider is where we choose the hyperparameters θ which maximizes $P(\mathbf{y}|\boldsymbol{x}_{1:t}, \theta)$,

$$\hat{\theta}_n^{MLE} = \arg\max_{\theta} P(\mathbf{y}_{1:t} \mid \boldsymbol{x}_{1:t}, \theta) \tag{16}$$

Since we are maximizing the likelihood this is referred to as the *maximum likelihood estimation* (MLE).

A natural extension to MLE includes a prior over the hyperparameters (refered to as a *hyperprior*), thus getting the *maximum a posteriori* (MAP),

$$\hat{\theta}_n^{MAP} = \arg\max_{\theta} P(\mathbf{y}_{1:t} \mid \boldsymbol{x}_{1:t}, \theta) P(\theta) \tag{17}$$

We arrive at this by first considering maximizing the posterior $P(\theta \mid \mathbf{y}_{1:t}, \boldsymbol{x}_{1:t})$ instead. Using bayes rule and dropping the normalization constant we obtain the expression,

$$\begin{aligned} P(\theta \mid \mathbf{y}_{1:t}, \boldsymbol{x}_{1:t}) &= \frac{P(\mathbf{y}_{1:t} \mid \boldsymbol{x}_{1:t}, \theta) P(\theta)}{P(\mathbf{y}_{1:t} \mid \boldsymbol{x}_{1:t})} \\ &\propto P(\mathbf{y}_{1:t} \mid \boldsymbol{x}_{1:t}, \theta) P(\theta). \end{aligned} \tag{18}$$

We can use this unnormalized posterior since the normalization constant does not vary across the parameters we are optimizing. Placing some kind of prior can be crucial, especially when few data points are available as is the case in Bayesian Optimization. Common priors include uniform priors to specify hard constraints, normal priors to suggest it lies near some nominal value and log-normal for strictly positive parameters. Notice how MLE is a special case of MAP in which a uniform prior has been placed over the hyperparameters. The specific priors used for this problemset will be discussed in section 3.5.

An analytical expression can be derived for both MLE and MAP (Rasmussen and Williams 2006) including their gradient. Thus both can be optimized using gradient decent methods like Conjugate Gradient or quasi-Newton methods. Note that in practice the *negative logarithm* of the objective is minimized since optimization problem are commonly stated as minimization problems and secondly to prevent numerical overflow.

Fully Bayesian. Both MLE and MAP provides a *point estimate* of the hyperparameters which are then used to get the predictive posterior distribution used for predicting y_{t+1} at $\boldsymbol{x}_{t+1}$. We will now consider the

so-called *fully Bayesian* treatment in which the hyperparameters are marginalized out,

$$P(\mathrm{y}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{D}_t) = \int P(\mathrm{y}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{D}_t, \theta) P(\theta|\mathcal{D}_t)\ \mathrm{d}\theta. \tag{19}$$

This integral is generally intractable so instead we approximate it by sampling θ,

$$P(\mathrm{y}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{D}_t) \approx \frac{1}{N}\sum_{\theta=1}^{N} P(\mathrm{y}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{D}_t, \theta), \tag{20}$$

in which $\{\,\theta_n\,\}_{n=1}^{N}$ is sampled from the posterior $P(\theta|\mathcal{D}_t)$. Since it is even impossible to sample directly from this posterior, Markov Chain Monte Carlo (MCMC) techniques are usually utilized to produce samples from an identical distribution in the limit of an infinitely long chain of samples. For further details on these techniques we refer to (Berg 2004).

2.2.5. Automatic Relevance Detection. Automatic Relevance Detection (ARD) is a method for multi-dimensional input useful when many dimensions are irrelevant. In the case of the SE kernel it learns an individual length scale l_d for each input dimension x_d,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\Big[-\frac{1}{2}\sum_{d=1}^{D}\Big(\frac{|x_d - x'_d|}{l_d}\Big)^2\Big]. \tag{21}$$

This way a dimension x_d can be rendered irrelevant by simply increasing l_d.

2.3. Acquisition function

Now that we have discussed the statistical model used to capture our belief about the unknown function f we return to discuss how to decide which subsequent point to evaluate.

A whole range of different acquisition functions exist e.g. probability of improvement (PI), expected improvement (EI), upper confidence bounds (UCB), which all try to trade off exploration and exploitation; that is have optima either where uncertainty is high (exploration) or where the model prediction is high (exploitation).

To optimize the acquisition function we often need to query it many time. It is therefore important that the acquisition function is much cheaper to evaluate (or approximate) than the black box f. One way of achieving this is by defining the acquisition function on a statistical model with known analytical form such as the GP.

All the above mentioned acquisitions functions do this and we will focus on the UCB defined as,

$$\alpha_{UCB}(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \kappa\sigma(\boldsymbol{x}). \tag{22}$$

Together with PI and EI it is part of a class of acquisition function guided by the principle of *being optimistic in the face of uncertainty*. That is, higher variance at a given point can only lead to an increased acquisition value. USB is especially attractive since it has a nice intuitive interpretation, follows directly from a visually plot of the underlying surrogate model (see fig. 1) and provides fast convergence in Bayesian Optimization tasks. Recently, good theoretical bounds for the *Cumulative Regret* (see section 3.6 for definition) have even been shown by careful choice of a time depended κ (Srinivas et al. 2012).

Let us turn to how we define the acquisition function when there is uncertainty about kernel hyperparameter θ. Define $\alpha(\boldsymbol{x}; \theta)$ as the acquisition function with known hyperparameter θ. We can marginalize out θ given the data $\mathcal{D}_t$ at timestep t,

$$\alpha_t(\boldsymbol{x}) = \mathbb{E}_{\theta|\mathcal{D}_t}[\alpha(\boldsymbol{x}; \theta)] = \int \alpha(\boldsymbol{x}; \theta) p(\theta|\mathcal{D}_t)\ \mathrm{d}\theta. \tag{23}$$

One simple way of dealing with this expression is to approximate $P(\theta|\mathcal{D}_t)$ by a point estimate (i.e. a Dirac distribution) at either $\hat{\theta}_t^{MLE}$ or $\hat{\theta}_t^{MAP}$ obtained as described in section 2.2.4.

However, since the uncertainty in the response surface is crucial to guide exploration in Bayesian Optimization, it is desirable to also incorporate the uncertainty about θ. Thus we approximate the marginalization by sampling θ,

$$\mathbb{E}_{\theta|\mathcal{D}_t}[\alpha(x; \theta)] \approx \frac{1}{N}\sum_{i=1}^{N} \alpha(\boldsymbol{x}; \theta_i). \tag{24}$$

with N samples from the posterior $P(\theta|\mathcal{D}_t)$ using MCMC as in section 2.2.4. This effectively averages over the samples similarly to section 2.2.4 but maps the posterior statistics through the acquisition function before averaging.

2.4. DNGO

This section will describe the method proposed in (Snoek et al. 2015) for making Bayesian Optimization scalable. The problem it addresses is that the running time at every step in GP based Bayesian Optimization is cubic in the number of observations (see section 2.2.3). This becomes prohibitively large when using a parallel Bayesian Optimization approach (Snoek, Larochelle, and Adams 2012) which allows them to make many (expensive) function evaluations than previously possible with the common sequential approach.

The proposed solution proceeds in two steps in which DNGO first learns a set of basis function using a neural network. Subsequently it fits a Bayesian linear regressor to the input transformed by the neural network to capture the uncertainty.

To make this model precise denote the output of the neural network $\boldsymbol{\phi}(\cdot) = [\phi_1(\cdot), \cdots, \phi_D(\cdot)]^\top$ where D is the number of basis functions. Given an input set $\{\,\boldsymbol{x}_i\,\}_{i=1}^{n}$

we can define what is commonly referred to as the *design matrix*,

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_1(\boldsymbol{x}_1) & \cdots & \phi_d(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{x}_n) & \cdots & \phi_d(\boldsymbol{x}_n) \end{bmatrix}. \tag{25}$$

The network weights are trained by adding a linear layer and then training on a dataset $\{ (\boldsymbol{x}_i, \mathrm{y}_i) \}_{i=1}^n$ using backpropagation and stocastic gradient descent with momentum. Exactly what dataset will be covered in section 3.2.

Now, given observation $\mathcal{D}_t = \{ \boldsymbol{x}_{1:t}, \mathbf{f}_{1:t} \}$ at step t in the Bayesian Optimization procedure it constructs the design matrix by applying the neural network. It then fits a Bayesian linear regressor providing predictive statistics as follows,

$$\mu(\boldsymbol{x}) = \mathbf{m}^\top \boldsymbol{\phi}(\boldsymbol{x}) \tag{26}$$

$$\sigma^2(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{K}^{-1} \boldsymbol{\phi}(\boldsymbol{x}) + 1/\beta, \tag{27}$$

where

$$\mathbf{m} = \beta \boldsymbol{K}^{-1} \boldsymbol{\Phi}^\top \mathbf{y} \tag{28}$$

$$\boldsymbol{K} = \beta \boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \boldsymbol{I}\alpha \tag{29}$$

Notice the similarity with the predictive mean (14) and variance (15) for GP. The primary difference is that $\boldsymbol{K}$ now grows with the size of the output dimensionality of the neural network. So the running time becomes $\mathcal{O}(n)$ instead of $\mathcal{O}(n^3)$ for n observations.

2.4.1. network architecture. The original paper specifies a network of three layers with 50 units in each using the ReLU activation function in the first two layers and TanH in the final layer.

3. Method

This section will cover the three models we are comparing, GP, DNGO and DNGO with an ensemble of neural networks, including details on how they are compared.

There are many choices to make both for the GP approach and the DNGO architecture. This section summaries the choices and the reasons behind them. For further detail we refer to the specific implementation at (Thomas M. Pethick 2018).

3.1. GP

The primary decision in GP is the choice of kernel. Here we will compare DNGO against a GP using the SE kernel. This is done for its smoothness assumption, simplicity and experimentally verified usefulness in Bayesian Optimization tasks (Snoek, Larochelle, and Adams 2012; Wang et al. 2013; Bergstra et al. 2011). Hyperpriors for this kernel are discussed in section 3.5.

A critical reader might question why more complicated kernels like the Matérn (Rasmussen and Williams 2006) were not considered. We decide to use the simplest kernel that performs well, which it is expected to do especially on the benchmark functions used, since most are smooth.

3.2. DNGO

The original network architecture as described in section 2.4.1 has been kept except for the use of the Adam optimizer (Kingma and Ba 2014) instead of SGD.

It was not clear how they trained the neural network. Thus we will experiment with three different approaches:

- training on a fixed set $\mathcal{D}_m$ of m randomly sampled observations;
- retraining on all $\mathcal{D}_t$ observations at every timestep t without resetting weights, and;
- retraining on all $\mathcal{D}_t$ observations at every timestep t but resetting the weight.

Results are covered in section 4 in which testing of hyperparameters such as weight decay and number of epochs are also included.

3.3. Ensemble

By introducing an ensemble of neural networks we obtain a set of values for the acquisition function at a given point – one for each neural network. This is because for all N neural networks we fit a Bayesian linear regressor separately from which the acquisition function is evaluated.

The acquisition function $\alpha(\boldsymbol{x})$ is then defined as some *aggregate* over this set of values,

$$\alpha(\boldsymbol{x}) = \mathsf{A}(\{ \alpha(\boldsymbol{x}); \theta_i \}_{i=1} N). \tag{30}$$

where $\mathsf{A} : \mathbb{R}^N \to \mathbb{R}$ is the aggregator function and θ_i denotes both the hyperparameters of the Bayesian linear regressor and the weights of i'th neural network.

Apart from varying N we will consider different method of aggregating this set of values. Specifically *mean*, *median* and *max*, as these are naturally expected to make very different exploration-exploitation tradeoffs.

3.4. Normalization

Both the input and output space are normalized to have mean 0 and variance 1 in each dimension. The primary benefit is that hyperpriors in the GP setting and the neural network become independent of the scale of the space. This allows us to specify useful problem-independent hyperpriors.

3.5. Hyperpriors

By normalizing to a fixed interval we can allow ourselves to give high prior weight to certain volumes of the hyperparameter space. That is because e.g. a length scale above 1 would assume almost no change throughout the input space (recall that the length scale is approximately the distance required for a change to happen).

Inspired by (Swersky, Snoek, and Adams 2014) we choose the three hyperpriors for the GP to be,

$$\begin{aligned} l &\sim \text{lognormal}(0,1) \\ \sigma_0^2 &\sim \text{lognormal}(0,1) \\ \sigma_{\text{noise}}^2 &\sim \text{horseshoe}(0.1) \end{aligned} \tag{31}$$

Similarly for the Bayesian linear regressor the following priors are picked,

$$\begin{aligned} \alpha &\sim \text{lognormal}(0,1) \\ \beta &\sim \text{horseshoe}(0.1) \end{aligned} \tag{32}$$

The log-normal prior ensures positive values with a mean 1 thus preventing unnecessarily large values from being considered. The horseshoe prior (described in detail in (Carvalho, Polson, and Scott 2009)) puts high weight on 0. Its Cauchy-like tail allows for strong observed values to remain large a posteriori but simultaneously allows for severe shrinkage for zero element. We can thus completely disregard noise in a noiseless setting.

For guidelines on hyperpriors for gaussian processes we recommend the user guide to Stan (Stan Development Team 2017).

3.6. Performance Evaluation

We evaluate and compare the performance with two regret based metrices: *Simple Regret* and *Cumulative Regret*.

Simple Regret measures how far the current best point is from the true optimum. To make it precise, define the optimal point as $\boldsymbol{x}^\star = \arg\max_{\boldsymbol{x}\in\mathcal{X}} = f(\boldsymbol{x})$ and the current best at round t as $\hat{\boldsymbol{x}}_t = \arg\max_{\boldsymbol{x}\in\boldsymbol{x}_{1:t}} f(\boldsymbol{x})$. Then Simple Regret as used in (Snoek et al. 2015; Springenberg et al. 2016; Golovin et al. 2017) is,

$$r_t^{\text{simple}} = |f(\boldsymbol{x}^\star) - f(\hat{\boldsymbol{x}}_t)|. \tag{33}$$

With this measure it is possible to compare how well different models have performed at a given round. However, it does not give a clear indication of the exploration-exploitation trade-off being made.

To captures this Cumulative Regret is naturally considered. Given the true optimum point $\boldsymbol{x}^\star$ and observation choice $\boldsymbol{x}_t$ at round t *instantanous regret* is defined as $r_t^{\text{instant}} = |f(\boldsymbol{x}^\star) - f(\boldsymbol{x}_t)|$. Cumulative Regret is defined as the sum of the instantaneous regret over all rounds (Srinivas et al. 2012),

$$R_T^{\text{cumulative}} = \sum_{t=1}^{T} r_t^{\text{instant}}. \tag{34}$$

When comparing models a model A is said to outperform a model B if the average Simple Regret across all runs in the final round is smaller for A than B. This is especially used in section 4.

4. Experiments

For comparing and testing optimization it is beneficial to have benchmark functions that are fast to evaluate and have known global optima. It is inherently problematic to decide on these synthetic function as the motivation for Bayesian Optimization is that f is a black-box function. The benchmarks might thus suffer from design biases – common is a placement of the optimum at the domain midpoint. However, they are still useful to understand the behavior of different methods. In (Dewancker et al. 2016) it is attempted to avoid the design bias by introducing a collection of benchmark categories based on a broad range of characteristics. We will focus on four categories that show interesting differences between GP, DNGO and DNGO ensemble:

- **Oscillatory** being a simple form of non-stationarity in which $f(x) = S(X) + T(x)$ where T introduces a long-range trend while S has a shorter scale oscillatory behaviour.
- **Mostly boring** which can be seen as a type of non-stationarity having small gradient in most of the domain except a very small region in which the optimum is found.
- **Nonsmooth** by having discontinuous derivatives on manifolds of at least one dimension.
- **Discrete values** being a specifically interesting instance of nonsmooth function in the application of hyperparameter optimization.

The specific benchmarks picked are a subset from both (Eggensperger et al. 2013) and (Dewancker et al. 2016).

We will not try to aggregate performance over several benchmark as is the original idea behind (Dewancker et al. 2016). This is primarily because a larger test set then the one picked is required. We are not interested in overall performance but rather in providing a preliminary investigation of the characteristics of DNGO and the ensembled DNGO. However, we do test it in a real hyperparameter optimization setting for a machine learning task as covered in section 4.5.

The models are tested on some higher dimensional problems but we have chosen to restrict these to 10 dimensions so optimization of the acquisition function would not become a problem. A characteristic of acquisition functions is that they are mostly boring (Lyu

et al. 2018). This is a problem in high dimension where optimization of the acquisition function from some set of random initialization cannot escape the large flat region. This could easily become the limiting factor for the model and render a comparison useless.

For the 8D problems and higher (including machine learning tasks) we chose to initialize the Bayesian Optimization procedure with 100 random samples similarly to what is done in (Lyu et al. 2018).

All plots will adhere to a specific naming convention. There are four different models *GP*, *DNGO fixed*, *DNGO retrain* and *DNGO retrain-reset* of which the last three refers to the different methods of training the neural network as described in section 3.2.

MCMC will be added as suffix if the fully Bayesian approach is used. For the DNGO methods a prefix is added indicating the size of the ensemble if one such is used.

As an example *10 × DNGO fixed MCMC* specifies an ensemble of 10 DNGO networks for which the neural network is trained on a fixed dataset and MCMC is used to sample hyperparameters for the Bayesian linear regressor.

In section 4.1 and section 4.2 we experiment with different hyperparameters of the DNGO architecture which are subsequently fixed for the remaining experiments. In section 4.3 the promising case of embedded function are presented. Then the four categories of benchmarks are considered in section 4.4 and hyperparameter optimization in its own section section 4.5.

The experiments can be reproduced using the accompanying source code at (Thomas M. Pethick 2018).

4.1. Hyperparameter Settings for DNGO

We compared different configurations of number of training epoch and l_2 regularization for the network evaluated on Branin, Hartmann3 and Hartmann6 from (Eggensperger et al. 2013). The model was rather robust with regards to these parameters so it was decided to use 1,000 epochs for pragmatic reasons and a small l_2 as suggested by (Snoek et al. 2015) of 10^{-4}.

4.2. Training

We investigated the effect of the size of the initial random samples for the DNGO with a trained network of fixed size. This was done on Hartmann3 as it had previously been shown to yield small incremental improvements per round with low variance which made a good candidate for method comparison.

Increasing the sample size yielded faster convergence and, in the case of 200 samples also an improved optimum (see section A.2). That the sample size sets an upper bound for how well f can be modeled also shows across Branin, Hartmann3 and Hartmann6 when compared with the other training strategies from section 3.2. *DNGO retrain-reset* performs better than *DNGO fixed* with the latter seemingly stuck. Thus we have decided to continue further experiments with DNGO retrain-reset which is computationally more expensive (though not asymptotically!) but which can update the weights based on incoming observations.

4.3. Embeddings

Low effective dimensionality is known to be a common characteristic of hyperparameter optimization (Bergstra and Bengio 2012; Chen, Castro, and Krause 2012; Wang et al. 2013). In particular, a function of two variables is said to have low effective dimensionality if it can be approximated by a function of one variable. Thus we consider embedding the objective function f into a higher dimensional space: the additional dimensions have no effect on the function value.

The Simple Regret result is plotted in fig. 3 for the embedding of the SinOne benchmark in a 2-, 3-, and 4-dimensional space. In all cases the ensembled DNGO does better than DNGO and the DNGO better than GP when averaging over runs. We will return to an explanation of this in the discussion.

It should be noted that the methods were indistinguishable from random sampling at higher dimensions.

4.4. Benchmarks

The benchmarks were divided into four categories. However the most interesting observations are between different categories and has to do with dimensionality.

DNGO consistently outperforms GP on smaller dimensional problems if they are nonsmooth or oscillatory (see e.g. Alpine01 and Griewank in section A.3 respectively). For Alpone01 this most likely comes down to the choice of the infinitely differentiable SE kernel for the GP however, since it performs significantly better on the smooth Branin and Hartmann3 functions. Further investigation of Griewank showed that the GP only captured the long-range trend of T (as defined in section 4).

Griewank provides an interesting insight into the ensemble in that you should not just blindly throw computational power after regularizing through an ensemble. In this benchmark the ensemble performed *worse* than DNGO. The reason becomes apparent when plotting the sequence of observations made in the 2D space over time. The ensemble tends to get stuck in a local optimum whereas DNGO explores a bigger part of the space. This reinforced the intuition that an averaging ensemble would decrease random exploration which in some instances could be useful to some degree. See section A.3 for details and fig. 5 for a similar effect.

There was no significant improvement by using an ensemble of DNGO in the low dimensions discussed so far. However in higher dimensions an ensemble improved performance in different ways even though it did not significantly outperform GP across all benchmarks. Firstly, it led to faster convergence for nonsmooth Corana 4D

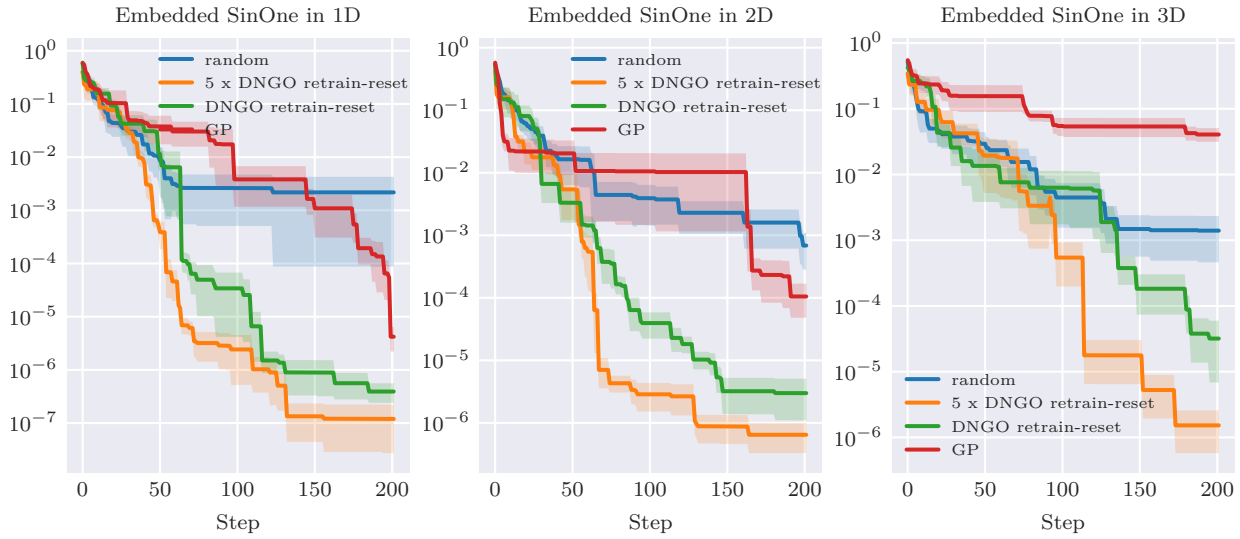


Figure 3: All plots shows the average Simple Regret over 10 runs with a 1/4 standard derivation confidence interval. Note how the ensembled DNGO consistently outperforms the DNGO. See section A.1 for accompanying Cumulative Regret plots.

function and oscillatory Dolan 5D function. More interestingly it outperformed DNGO on the mostly boring Hartmann6 and LennardJones6 as well as the higher dimensional Rosenbrock 8D (see fig. 4).

We have to note though that these results most likely have been influenced by optimizing the network primarily on Hartmann3 and Branin. This is problematic because we might overfit on lower dimensional spaces. One could imagine that an improvement to the network could have made the gain achieved in higher dimensions by an ensemble insignificant.

4.5. Machine Learning Tasks

We tried the methods on three machine learning tasks from (Eggensperger et al. 2018):

- **Logisitic Regression on MNIST** tuning learning rate, l_2 regularization, batch size and dropout rate of which one is discrete.
- **Convolution Neural Network (CNN) for CIFAR-10** tuning learning rate, batch size and the number of units in each of the three layers. Four parameters are discrete.
- **Fully Connected Network for MNIST** tuning 10 parameters of which 3 are discrete.

As stated, some of the parameters are discrete. We deal with it by relaxing the problem to $\mathbb{R}$ in all dimensions and do rounding before evaluation of f.

For both CNN and the fully connected network there was no noticeable different from random sampling for any of the tested methods.

For the logistic regression all Bayesian Optimization models did on average better than random sampling. However, they all converged to the same suggested optimal value. Thus it did not provide any insight into the behavior of the ensemble. For further detail we refer to the appendix section A.4.

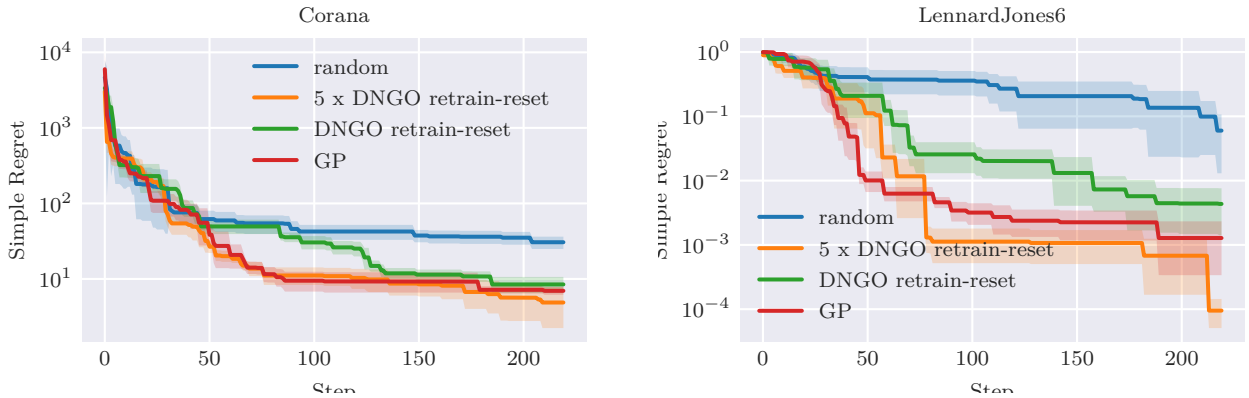
5. Discussion

This section will start by making observation about the behavior of DNGO in section 5.1. Then section 5.2 follows up on results from section 4 by commenting on why they differ from the original paper. The motivation behind DNGO is subsequently discussed in section 5.3 and, finally, different approaches to performance evaluation is considered in section 5.4.

5.1. DNGO Characteristic

We note several characteristics of the DNGO model. First it seems generally worse at exploiting than GP. This is illustrated with the Branin function in fig. 5. This shows that when DNGO is exploiting (i.e. querying some local area) it does so in a relatively random fashion.

This could be explained by the second behavior which was observed, namely sudden fluctuations. That is, high peaks in areas without observations which even had low variance. Because of these it was chosen to use *mean* as ensemble aggregator as it would behave as a regularizer. *Max* seemed to increase exploration but this was not thoroughly studied. However, investigating how different quantile might effect the exploration-exploitating trade-off could be an interesting future avenue to pursue. This is especially interesting considering the failure case of an ensemble in section 4.4.



(a) Example showing the faster convergence for ensembled DNGO over DNGO on nonsmooth function.

(b) Example showing that an ensemble improves on mostly boring functions in sufficently high dimensions (i.e. above 6).

Figure 4: Both plots shows the average Simple Regret over 10 runs with a 1/4 standard derivation confidence interval.

Concerning embedding functions, they provided a promising case over GP and indicated a class in which an ensemble might be especially useful. However, the method is not competitive with specialized techniques like ARD covered in section 2.2.5. Neither does it scale to the same level of dimensions like REMBO (Wang et al. 2013) which specifically exploits problems with low effective dimensionality. Since similar assumption have not been made about the problem the hope is that ensembled DNGO usefulness will generalize to a broader class of problems.

5.2. DNGO Deviations

The results obtained with DNGO on benchmarks from (Eggensperger et al. 2013) are worse than in the original paper (Snoek et al. 2015). We propose two possible reasons for this.

Firstly, (Snoek et al. 2015) put a quadratic prior on their mean thus incorporating expert knowledge as explained in section 2.2.2. We have purposely left this prior out to make methods more comparable on the benchmark suite since (Eggensperger et al. 2013) do in fact suffer from design bias of placing the optimum in the midpoint of the domain (Dewancker et al. 2016).

Secondly, they optimized their architecture using Bayesian Optimization on an unknown training set where we instead hand-tuned a reimplementation of DNGO. Considering that we are interested in comparing DNGO with its ensembled equivalent, this is not so problematic.

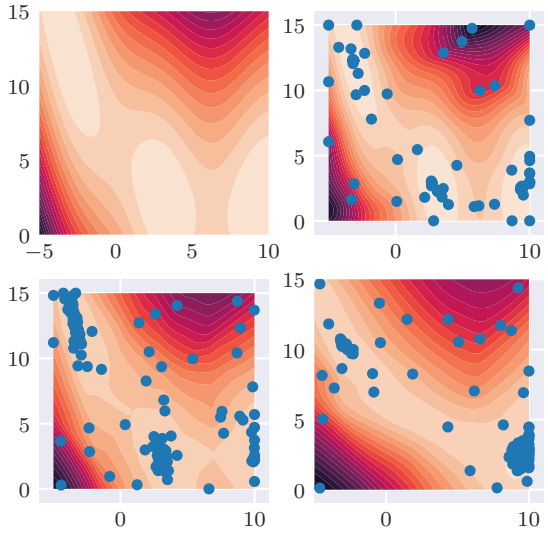


Figure 5: These plots shows 200 observations of the Branin function using three different models each plotting the associated acquisition function. Notice how the regularizing effect of the ensemble *prevents* it from exploring the third optimum. *Top left*: the Branin function. *Top right*: GP. *Bottom left*: DNGO retrain-reset. *Bottom right*: 5 × DNGO retrain-reset.

5.3. Alternative Extensions

The need for a scalable method was based on parallisability – we can obtain many more samples if we sample in parallel. The original paper (Snoek et al. 2015) evaluates a maximum of 2500 points in their biggest experiment, however. This is within the bounderies of what is already computational feasible with GPs.

This prompts the question of whether DNGO is useful because of it scalability or due to the possiblity of capturing non-stationarity. The latter would seem to be the case considering the relatively positive results for

DNGO on the oscillatory benchmarks. However, these could also be because the GP generally have troubles in high dimensions.

Either way, relaxing the requirement of scalability could open up interesting approaches. One method is to jointly train the Bayesian linear regressor and neural network weights as referred to as Deep Kernel Learning by (Wilson et al. 2016). Instead of only considering Bayesian linear regressors (which can be seen as a GP with a linear kernel) it considers more expressive kernels like the SE kernel and spectral mixture kernel. This method is even made to scale linearly using KISS-GP (Wilson and Nickisch 2015).

Another interesting way to combine GPs and neural networks is the recently proposed Neural Processes (Garnelo et al. 2018). They bear resemblance to variational autoencoders and provide an efficient way of learning a distribution over functions.

5.4. Performance Evaluation

We have provided a qualititive analysis on a small selection of benchmarks. For a more thorough testing, some aggregate of performance over several benchmarks should be considered. A simple approach used in (Golovin et al. 2017) is to normalize using Simple Regret for random search and average over all benchmarks. The ranking in (Dewancker et al. 2016) is more suphisticated and considers which method converges faster if there is no significant difference in performance. Most importantly, though, is to mitigate design bias, as was attempted in (Dewancker et al. 2016).

When considering performance in Bayesian Optimization the computational budget is another important metric. These are very practical methods for which even constants (which are usually disregarded in asymptotic analysis) relate directly to a very real expense. An experimental comparison between DNGO and GP on Hartmann6 can already been found in (Snoek et al. 2015). DNGO is asymptotically much faster which shows even after only 300 rounds on Hartmann6. Even though a test has not been conducted for the introduced ensembled method, it clearly only scales the budget by a factor k in the worst case where k is the size of the ensemble.

6. Conclusion

This work has investigated the DNGO and ensembled DNGO and a collection of benchmark and three machine learning tasks. The machine learning tasks turned out to be a fruitless attempt. However the for embedded synthesised function and for sufficently high dimensional mostly boring benchmark the ensemble deamed a promising case. However, to confirm this, more thorough testing is needed across multiple benchmarks and with a statistically significant number of runs. This would also allow for comparison with existing methods such as BOHAMIANN (Springenberg et al. 2016) and by applying deep kernel learning (Wilson et al. 2016) to Bayesian Optimization. Lastly, the effect of the ensemble aggregator was apparent, and this suggests a deeper investigation of its effect by looking into the effect of various quantiles.

References

Berg, Bernd A. (2004). "Introduction to Markov Chain Monte Carlo Simulations and their Statistical Analysis". In: *arXiv:cond-mat/0410490*. arXiv: cond-mat/0410490. URL: http://arxiv.org/abs/cond-mat/0410490.

Bergstra, James and Yoshua Bengio (2012). "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13 (Feb), pp. 281–305.

Bergstra, James S et al. (2011). "Algorithms for hyper-parameter optimization". In: *Advances in neural information processing systems*, pp. 2546–2554.

Carvalho, Carlos M, Nicholas G Polson, and James G Scott (2009). "Handling sparsity via the horseshoe". In: *Artificial Intelligence and Statistics*, pp. 73–80.

Chen, Bo, Rui Castro, and Andreas Krause (2012). "Joint optimization and variable selection of high-dimensional Gaussian processes". In: *arXiv preprint arXiv:1206.6396*.

Contal, Emile, Vianney Perchet, and Nicolas Vayatis (2014). "Gaussian process optimization with mutual information". In: *International Conference on Machine Learning*, pp. 253–261.

Dewancker, Ian et al. (2016). "A Stratified Analysis of Bayesian Optimization Methods". In: *arXiv:1603.09441 [cs, stat]*. arXiv: 1603.09441. URL: http://arxiv.org/abs/1603.09441.

Duvenaud, David (2014). "Automatic model construction with Gaussian processes". PhD thesis. University of Cambridge.

Eggensperger, Katharina et al. (2013). "Towards an empirical foundation for assessing bayesian optimization of hyperparameters". In: *NIPS workshop on Bayesian Optimization in Theory and Practice*. Vol. 10, p. 3.

Eggensperger, Katharina et al. (2018). *HPOlib2*. URL: https://github.com/automl/HPOlib2.

Garnelo, Marta et al. (2018). "Neural Processes". In: *arXiv:1807.01622 [cs, stat]*. arXiv: 1807.01622. URL: http://arxiv.org/abs/1807.01622.

Golovin, Daniel et al. (2017). "Google Vizier: A Service for Black-Box Optimization". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '17*. the 23rd ACM SIGKDD International Conference. Halifax, NS, Canada: ACM Press, pp. 1487–1495. ISBN: 978-1-4503-4887-4. DOI: 10.1145/3097983.3098043. URL: http://dl.acm.org/citation.cfm?doid=3097983.3098043 (visited on 09/10/2018).

Hutter, Frank, Holger H. Hoos, and Kevin Leyton-Brown (2011). "Sequential model-based optimization for general algorithm configuration". In: *International Conference on Learning and Intelligent Optimization*. Springer, pp. 507–523.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]*. arXiv: 1412 . 6980. URL: http://arxiv.org/abs/1412.6980.

Li, Lisha et al. (2016). "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: *arXiv:1603.06560 [cs, stat]*. arXiv: 1603 . 06560. URL: http://arxiv.org/abs/1603.06560.

Lyu, Wenlong et al. (2018). "Batch Bayesian Optimization via Multi-objective Acquisition Ensemble for Automated Analog Circuit Design". In: *International Conference on Machine Learning*, pp. 3312–3320.

Rasmussen, Carl Edward and Christopher K. I. Williams (2006). *Gaussian processes for machine learning*. Adaptive computation and machine learning. OCLC: ocm61285753. Cambridge, Mass: MIT Press. 248 pp. ISBN: 978-0-262-18253-9.

Shahriari, B. et al. (2016). "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1, pp. 148–175. ISSN: 0018-9219. DOI: 10.1109/JPROC.2015.2494218.

Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams (2012). "Practical Bayesian Optimization of Machine Learning Algorithms". In: *arXiv:1206.2944 [cs, stat]*. arXiv: 1206.2944. URL: http://arxiv.org/abs/1206.2944.

Snoek, Jasper et al. (2015). "Scalable Bayesian Optimization Using Deep Neural Networks". In: *arXiv:1502.05700 [stat]*. arXiv: 1502 . 05700. URL: http://arxiv.org/abs/1502.05700.

Springenberg, Jost Tobias et al. (2016). "Bayesian optimization with robust bayesian neural networks". In: *Advances in Neural Information Processing Systems*, pp. 4134–4142.

Srinivas, Niranjan et al. (2012). "Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design". In: *IEEE Transactions on Information Theory* 58.5, pp. 3250–3265. ISSN: 0018-9448, 1557-9654. DOI: 10.1109/TIT.2011.2182033. arXiv: 0912.3995. URL: http://arxiv.org/abs/0912.3995.

Stan Development Team (2017). *Stan Modeling Language Users Guide and Reference*. URL: http://mc-stan.org/.

Swersky, Kevin, Jasper Snoek, and Ryan Prescott Adams (2014). "Freeze-Thaw Bayesian Optimization". In: *arXiv:1406.3896 [cs, stat]*. arXiv: 1406 . 3896. URL: http://arxiv.org/abs/1406.3896.

Thomas M. Pethick (2018). *Ensembled DNGO Source Code*. URL: https : / / github . com / tmpethick / ensembled-dngo.

Wang, Ziyu et al. (2013). "Bayesian Optimization in a Billion Dimensions via Random Embeddings". In: *arXiv:1301.1942 [cs, stat]*. arXiv: 1301 . 1942. URL: http://arxiv.org/abs/1301.1942.

Wilson, Andrew and Hannes Nickisch (2015). "Kernel interpolation for scalable structured Gaussian processes (KISS-GP)". In: *International Conference on Machine Learning*, pp. 1775–1784.

Wilson, Andrew Gordon et al. (2016). "Deep kernel learning". In: *Artificial Intelligence and Statistics*, pp. 370–378.

Appendix A.
Results

Where otherwise noted, plots shows the average Simple Regret over 10 runs with a 1/4 standard derivation confidence interval. The naming convention for models is described in section 4.

A.1. Embedding

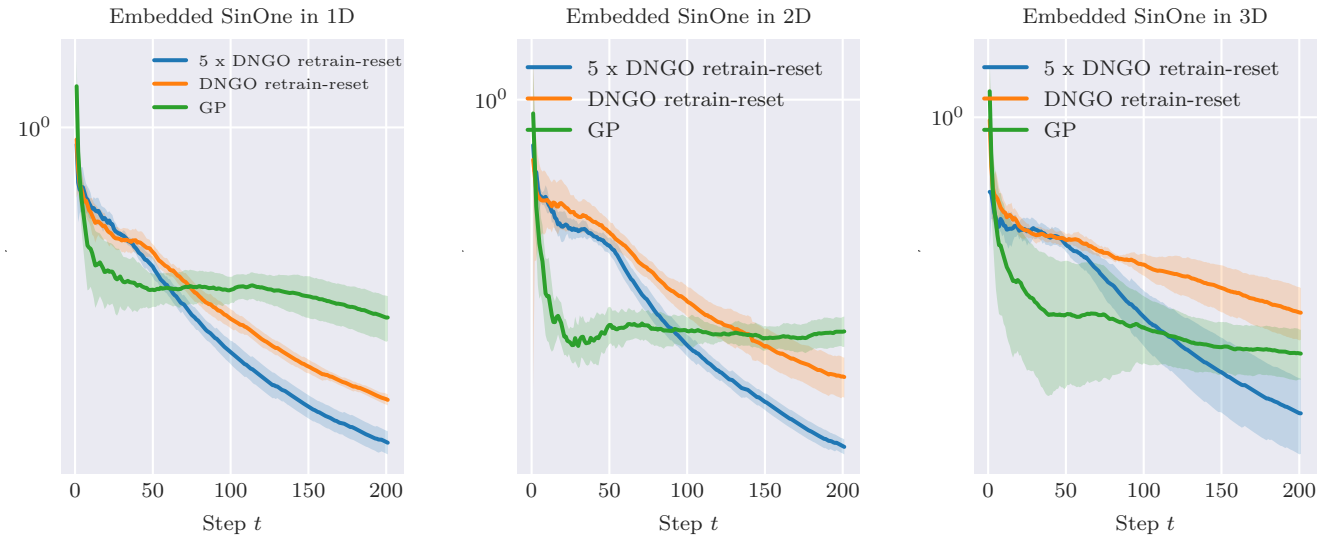


Figure 6: For every embedded SinOne the Cumulative Regret is normalized by the round t and plotted similarly to (Contal, Perchet, and Vayatis 2014).

A.2. Training

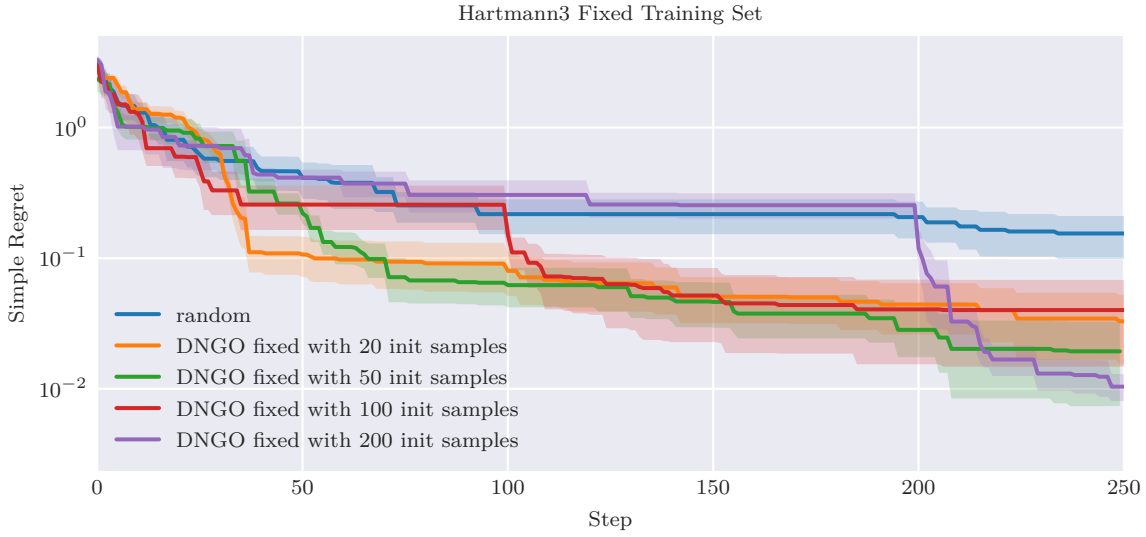
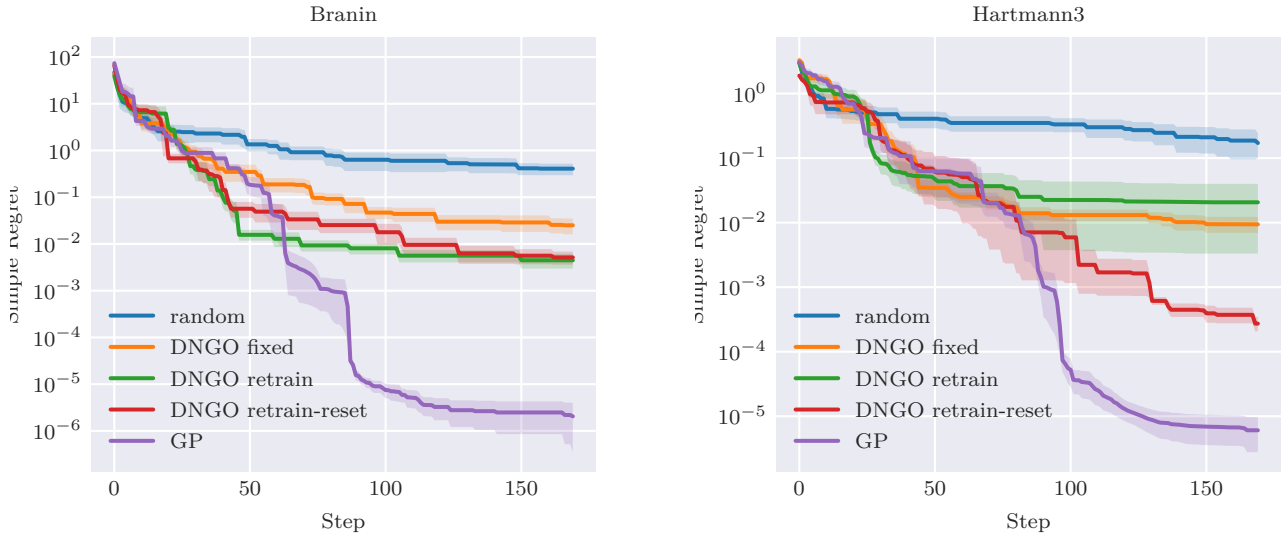


Figure 7: This plots shows Simple Regret for DNGO fixed with different initial samples sizes as described in section 3.2.



A.3. Benchmarks

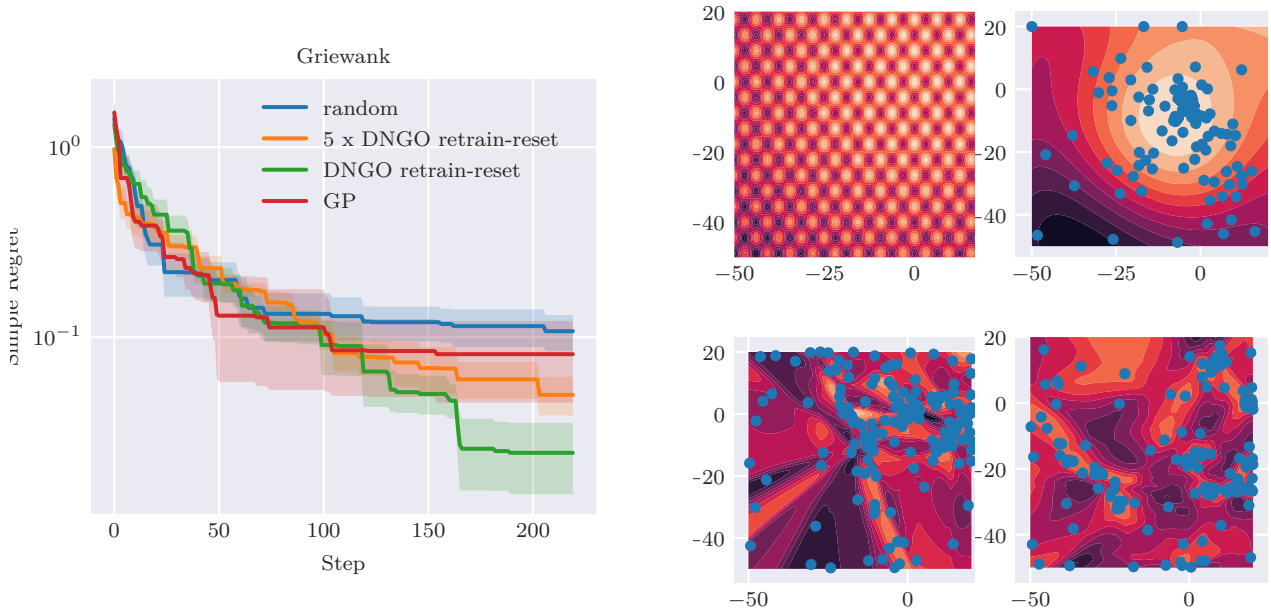


Figure 8: These plots shows 200 observations of the Griekwank function using three different models each plotting the associated acquisition function. The left plot shows the Simple Regret. The right plot have four parts. *Top left*: the Griekwank function. *Top right*: GP. *Bottom left*: DNGO retrain-reset. *Bottom right*: 5 × DNGO retrain-reset.

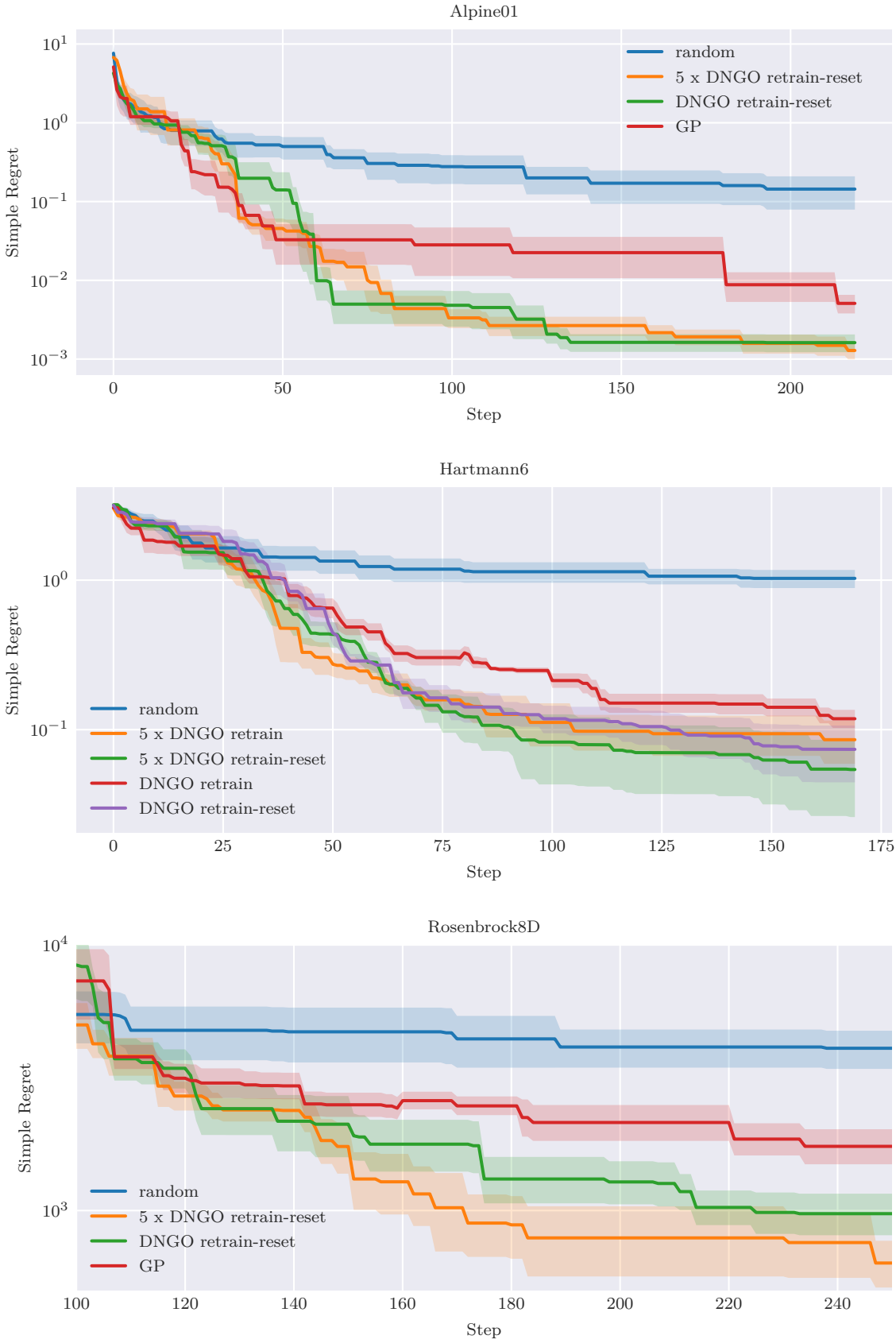
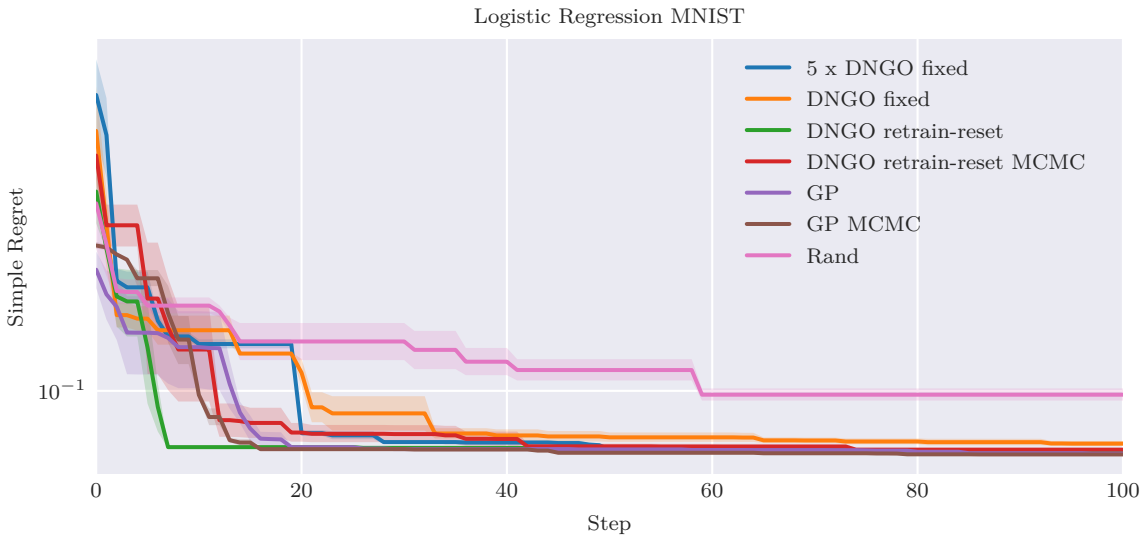


Figure 9: Shows the nonsmooth Alpine01 from (Dewancker et al. 2016) and Hartmann6 and Rosenbrock8D from (Eggensperger et al. 2013).

A.4. Machine Learning Hyperparameter Optimization Tasks

Method	Logistic regression	Fully Connected Network	CNN
5 × DNGO fixed	0.069	–	–
DNGO fixed	0.072	–	–
DNGO retrain-reset	0.069	0.0125	0.897
DNGO retrain-reset MCMC	0.069	–	–
GP	0.068	0.0119	0.897
GP MCMC	0.067	–	–
Random	0.098	–	0.897

TABLE 1: Logistic Regression indicates average over 5 runs while the rest was run once. Some cells are empty because it was decided not to waste computational resources on a fruitless endevour.



A.5. Miscellaneous