Neural networks smart grid based optimisation for expensive functions

Alain Uwadukunze, Xavier Bombois, Marion Gilson, Marie Albisser

Neural networks smart grid based optimisation for expensive functions

Alain UWADUKUNZE[1,2], Xavier BOMBOIS[3], Marion GILSON[2], Marie ALBISSER[1]

***Abstract*— Bayesian optimisation is an emerging machine learning technique known to be efficient especially for optimising functions which are expensive to evaluate. In Bayesian optimisation, a Gaussian process model of the unknown function is identified based on available data. Its estimate of the unknown function and the associated uncertainties are used to build a so-called acquisition function which does a trade-off between exploitation and exploration. The latter is then iteratively maximised to find candidates which are promising to be close to the optimum. In this paper, an alternative version of Bayesian optimisation, where the Gaussian process model is replaced by a neural network model, is proposed. As shown in the numerical illustration of this paper, this alternative version will require less computation time when facing optimisation problems with initially large data sets. Since neural networks do not naturally provide an information about the quality of the estimates, a different strategy for the exploration objective of our approach is proposed. The efficiency of the proposed approach is illustrated and compared to Bayesian optimisation on different case studies.**

I. INTRODUCTION

In many applications such as engineering design or control, one seeks to determine the maximum X_{opt} of an unknown static function $f(X)$ of a vector X of design variables. Even though $f(X)$ is unknown, this function can be evaluated for any given X (in a certain range). However, it is supposed that evaluating $f(X)$ for a given X is costly (in time and/or resources). Consequently, a gradient-based optimisation with a numerical evaluation of the gradient is not the most appropriate approach to determine X_{opt}. Instead, the literature proposes different techniques that are based on a smart gridding of the space of X. For each grid point X, the function is evaluated and a procedure determines the next grid point in a smart way. To achieve this, a model of the function $f(X)$ is identified based on the previous grid points $\{X^n, f(X^n)\}_{n=1}^k$ and this model is used to determine a point X^{k+1} (the next grid point) that is a promising candidate for (being close to) X_{opt}. A popular version of this smart grid approach is the so-called Bayesian optimisation (BO) [1] where the model of the function $f(X)$ takes the form of a Gaussian process (GP) [2]. In this paper, we will be comparing the classic BO to an alternative version, where the model of the function is a neural network (NN).

The problem described in the previous paragraph is quite classical in data-driven control applications where a controller achieving the highest level of performance is designed without requiring a model of the to-be-controlled system. Instead, an optimal controller is determined by testing different controllers (with different parameters X) on the system. It is clear that, for many real-life systems, the evaluation of a candidate can be quite costly and the to-be-tested controllers should therefore be chosen in a smart way. Bayesian optimisation is generally performed for this purpose: a model of the performance $f(X)$ of the closed loop made up of the system and the controller with parameters X is identified based on the previous experiments and this model is used to determine the parameters of the controller that has to be tested in the next experiment. Examples of application of BO in control can be found in [3], [4], [5] and [6]. Bayesian optimisation is also performed in many other applications such as in *Environmental Monitoring* as suggested in [7].

The model of the unknown function $f(X)$ is a crucial component of such smart-grid based optimisation procedures. As mentioned above, GPs are generally used for this purpose. A GP is a non-parametric identification method that allows to deduce an estimate of $f(X)$ for any point X based on data points $\{X^n, f(X^n)\}_{n=1}^k$. Besides this estimate, the GP also provides an uncertainty region indicating the confidence we can have in the estimate. This uncertainty region is deduced in a Bayesian framework and its reliability of course depends on the chosen prior assumptions and kernels used in this Bayesian framework. This means that it is not guaranteed that the unknown true function lies in the uncertainty region. However, it will be generally observed that the uncertainty will be large for points X that are far away from the data points with which the GP has been identified.

The GP model of the function $f(X)$ (and its uncertainty) is then used to determine the next point X^{k+1} for which $f(X)$ must be evaluated. This is generally done by maximising a so-called *acquisition function* $A(X)$. The acquisition function $A(X)$ is a function of the GP model and of its uncertainty. Its maximum (*i.e.*, the next grid point X^{k+1}) will be a point for which the GP model of $f(X)$ is large (exploitation) and/or for which the uncertainty of the GP is high (exploration). The acquisition function achieves the trade-off between these two objectives.

In recent studies, the use of other types of models, in procedures similar to BO, has been explored. For example in [8], Inverse Distance Weighting interpolation and Radial Basis functions are successfully used as models of the to be optimized function. The goal in this paper, will be to show that a neural network (NN) model can also be efficiently used in a smart-grid based optimisation procedure. Neural networks are increasing in popularity and being applied in

[1] French-German Research Institute, ISL, 5 rue du Géneral Cassagnou, 68300 Saint-Louis, France

[2] Université de Lorraine, CNRS, CRAN, F-54000 Nancy, France

[3] Laboratoire Ampère, Ecole Centrale de Lyon, Université de Lyon, 36 avenue Guy de Collongue, Ecully, France and Centre National de la Recherche Scientifique (CNRS), France

many fields such as system identification [9], [10]. Moreover, they are known to require less training time than regular GP when the number of training samples increases. This may prove to be useful when BO needs to be performed with an initially large dataset. It is clear that, unlike a GP model, a NN model is generally not accompanied by an uncertainty region. However, it will be shown that efficient acquisition functions can also be built when the model of $f(X)$ is a NN model by using the distance to observed samples for the exploration. It will also be illustrated, through different case studies, that using a NN in a smart grid based procedure can provide similar optimisation performances to regular BO but presents the interesting advantage of requiring less computational time than regular BO when the initial dataset available is large.

The rest of the paper is structured as follows : First, the BO algorithm is presented in section III-A, then the proposed approach is presented in section III-B. Finally, the proposed method is illustrated through different applications in section IV.

II. Problem statement

This paper focuses on optimisation problems of the form:

$$X_{opt} = \arg\max_{X} f(X), \tag{1}$$

where $X \in \mathbb{R}^N$ with N the dimension of the inputs and $f(X) \in \mathbb{R}$. f is considered to be unknown and expensive to evaluate (*i.e.* requires a lot of resources or time to obtain the value of $f(X)$ for a given X).

For the optimisation problem (1), this paper has the following objective: given a dataset containing k samples: $D^k : \{X^n, f(X^n)\}_{n=1}^k$, obtained from an unknown function f, use a neural network as a model of the unknown function f in a smart grid based optimisation procedure such as BO, to search which promising candidates to evaluate to get close to X_{opt}.

III. Methodology

The proposed smart grid optimisation is inspired of BO which is briefly recalled here.

A. Bayesian optimisation

Bayesian optimisation aims at finding an optimum X_{opt} of the problem (1). The optimum is searched by iteratively maximising a so called *acquisition function* $A(X)$. The maximisation of $A(X)$ yields a promising candidate to evaluate on f in order to get close to X_{opt}. The acquisition function balances the search between exploitation (searching for the place where a model's estimate of f is large) and exploration (searching for the place where the uncertainties are the largest). Therefore, a model of the unknown function f and its uncertainty are needed to build the acquisition function. In this sense, all the available samples from f, $D^k : \{X^n, f(X^n)\}_{n=1}^k$, with k the number of samples, are used to construct a probabilistic model of $f(X)$ capable of providing uncertainties. The most commonly used probabilistic models are Gaussian processes (GP) [2] which are non-parametric regression models. A GP assumes a prior Gaussian distribution over an unknown function f defined by a prior mean $\mu_p(X)$ and Kernel matrix $K(X, X^{'})$. Kernels are covariance matrices which measure the similarity between points using distance metrics. An example of a frequently used kernel is the *Radial Basis Function* (RBF), defined as follows for two points X_a and X_b :

$$K(X_a, X_b) = \exp\left(-\frac{d(X_a, X_b)^2}{2l^2}\right), \tag{2}$$

where d is the Euclidean distance and l a length scale parameter. When samples from f, $D^k : \{X^n, f(X^n)\}_{n=1}^k$, become available, the GP model is updated through Bayesian inference and provides a Gaussian distribution $\mathcal{N}(\mu(X), \sigma^2(X))$ over the unknown function f. This process is repeated at each iteration of BO as new samples become available. The mean $\mu(X)$ is the best estimation of $f(X)$ and is used for the exploitation. The variance $\sigma^2(X)$ provides the uncertainties on f for each point X and is used for the exploration.

Several acquisition functions for BO can be found in the literature, as suggested in [1]. Among them are : Probability of Improvement, Upper Confidence Bound and Expected Improvement. Since the proposed approach is inspired of the Upper Confidence Bound (UCB) acquisition function (see III-B for comparison), the paper focuses mainly on this technique. This acquisition function is perhaps the most straightforward to understand and is defined as follows :

$$UCB(X) = \beta\sigma(X) + \mu(X), \tag{3}$$

where $\sigma(X)$ and $\mu(X)$ are the standard deviation and predictive mean of the GP, respectively. The trade-off between exploitation and exploration is achieved by finding the point X for which σ and μ are both the highest. The parameter β can be used to control the degree of the trade-off. When β is chosen high more importance will be given to the exploration objective and when it is chosen low, more importance will be given to the exploitation objective. However, the value of β can also be chosen more appropriately to, *e.g.*, bound some cumulative regret as suggested in [11].

The BO algorithm using UCB as acquisition function to solve an optimisation problem (1), given an initial dataset D^k, is described in Algorithm 1. Note that if the dataset D^k is large, *i.e.* k is large, the first step of Algorithm 1 which involves identifying a new GP model at each iteration of BO may be time consuming.

Depending on the application or available data, one may want to use other models to represent the unknown functions. As a result, an alternative to BO where neural networks are used as models of the unknown function is proposed in the following section.

B. Proposed approach using neural networks

In this section, an alternative approach to BO is proposed. We refer to this method as Neural Network Smart Grid Optimisation (NNSGO). Similarly to BO, the idea behind this method is that the most promising point to evaluate on the real function f should be a trade-off between the

Algorithm 1 Bayesian optimisation algorithm

for $j = 0, ..., N_{it}$, with N_{it} the maximum number of iterations, **do**

1. Train GP with available set of samples. D^{k+j} : $\{X^n, f(X^n)\}_{n=1}^{k+j}$

2. Build acquisition function :

$$A_j(X) = \beta\sigma_j(X) + \mu_j(X)$$

3. The next point X_{k+j} to evaluate on f is

$$X^{k+j} = \arg\max_X A_j(X)$$

4. Evaluate X^{k+j} on f and add $(X^{k+j}, f(X^{k+j}))$ to the training samples D^k.

end for

return X with the highest $f(X)$ from $D^{k+N_{it}}$

exploitation and exploration objectives. The main difference between NNSGO and BO is that NN are employed as models of the functions rather than using probabilistic models such as GPs. The great advantage of neural networks resides in their ability to handle different types of problems while requiring less computational time to learn large datasets (*i.e.* datasets D^k, where the number of observations k is large) than other machine learning algorithms, such as GPs. Many different types of NN exist. In this paper, multi layer perceptrons (MLP), which are NN with fully connected layers are used as models. They provide the relation between a set of inputs X, where $X \in \mathbb{R}^N$ with N the dimension of the inputs, associated to outputs $f(X)$, where $f(X) \in \mathbb{R}$, thanks to trained parameters known as weights and bias.

A MLP needs to be trained to provide the best estimates of an unknown function. The training consists in finding the best values of parameters that minimise an identification criterion commonly known as loss function for NNs. The loss function which is frequently used in regression to train the MLP is the Mean Squared Error (*MSE*) defined as

$$MSE = \frac{1}{k}\sum_{n=1}^{k} \|\hat{f}(X^n) - f(X^n)\|^2, \tag{4}$$

where k is the number of samples, $f(X^n)$ the target output for X^n, $\hat{f}(X^n)$ the estimate of $f(X^n)$ by the NN and $\|A\| = \sqrt{A^T A}$ represents the Euclidean norm of a vector A. The most commonly used method to train a MLP is back-propagation which uses Gradient Descent [12] to iteratively update the parameters of the NN. Using this method requires the MLP to have differentiable activation functions such as *reLu* or *tanH*.

1) Exploration strategy for NNSGO: Unlike GPs, regular NN do not come with a measure of their uncertainties. Therefore, the exploration when using NN has to be achieved in a different way to classic BO. In the proposed procedure, we suggest to perform the exploration with the objective of covering regions not already covered by the dataset (and therefore not already observed by the NN). The rationale behind this proposal is that when a point X to be predicted is far from the observed points, then it is likely that the prediction $\hat{f}(X)$ will differ from the real value $f(X)$. This is also the case in GP, where the exploration is performed using the uncertainties which are based on the similarity between the evaluated points and the observed ones. Therefore, this idea is used to create a measure which is employed for the exploration objective of the proposed procedure. In that respect, $\delta_{min}(X)$ is defined as the Euclidean distance between a point X and the closest point to it from the points in the dataset D_X^k. It is defined as :

$$\delta_{min}(X) = \min_{X^n \in D_X^k} \|X - X^n\|. \tag{5}$$

This means that $\delta_{min}(X)$ is small when evaluating a point close to the observed points and increases when evaluating a point far from the observed ones. When $\delta_{min}(X)$ is low, it is likely that the estimate of $f(X)$, $\hat{f}(X)$ is close to the real value $f(X)$ and vice-versa.

2) Illustration of the comparison between δ_{min} and the GP uncertainties: The measure δ_{min} can be compared to the uncertainties provided by a GP model. To illustrate this, k = 5000 random samples are generated from a function $f(X)$ where $f(X)$ is the 12 dimensional form of the Rosenbrock function defined as :

$$f(X) = \sum_{i=1}^{N-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2], \tag{6}$$

with $X = (x_1, x_2..., x_N) \in \mathbb{R}^N$ and N = 12. The optimisation of this function using the proposed procedure will be performed in section IV-C. The goal of this illustration is to visualize how the NN performs and how the GP standard deviation behaves when the value of δ_{min} increases. For this purpose, a GP and a NN model of the 12 dimensional Rosenbrock function, are identified using the generated samples. In this example, training the GP model recquires more computational time than training the NN model. This will be further discussed in section IV-C. A new dataset D_{new}^l containing $l = 500$ new samples $\notin D^k$ is generated. The GP and NN models are then used to predict the 500 unseen points from D_{new}^l. The following elements are then calculated:

- The standard deviation $\sigma(X)$ for $X \in D_{new}$, note that this value is multiplied by 4 for better visualization in Fig. 1.
- The $\delta_{min}(X)$ for $X \in D_{new}^l$ and computed with (5) where D_X^k is the set containing the $k = 5000$ initial points.
- The error of the NN for each point $X \in D_{new}^l$ determined as the absolute value of the difference between the estimate of the Rosenbrock function by the NN $\hat{f}(X)$ and the actual value $f(X)$: $error = |\hat{f}(X) - f(X)|$.

Fig. 1 presents the GP standard deviation (σ) and the NN error w.r.t to δ_{min} for different tested points. On this figure, the value on the vertical axis of each orange dot

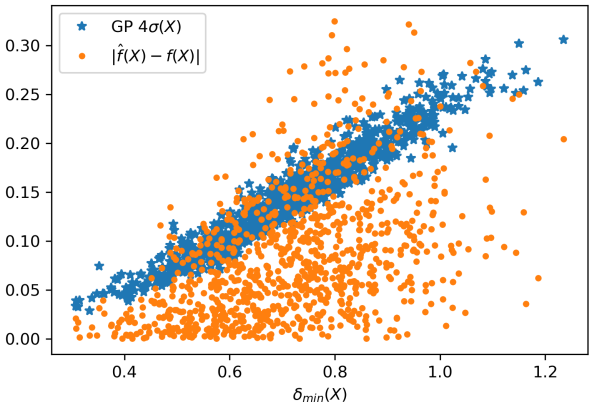

Fig. 1: Value of GP σ (represented by the blue stars) and NN error (represented by the orange dots) w.r.t δ_{min} (horizontal axis).

represents the error on the NN prediction ($|\hat{f}(X) - f(X)|$) of an unseen tested point X and the corresponding value on the horizontal axis, represents the value of δ_{min} for that point. The value on the vertical axis of each blue star represents the GP standard deviation σ of an unseen tested point X and the corresponding value on the horizontal axis, represents the value of δ_{min} for that point. We can observe that the points X for which $\delta_{min}(X)$ is low (*i.e* points in regions well covered by the dataset) are relatively well predicted by the NN. The GP standard deviation $\sigma(X)$ is also low for these points. However, we see that when $\delta_{min}(X)$ increases, more and more points tend to be poorly predicted by the NN as the error increases. The GP standard deviation is also greater for points where $\delta_{min}(X)$ is large. Therefore, by using the measure $\delta_{min}(X)$ for the exploration objective of our procedure, we will test in regions where the model is likely to perform poorly. This is essential, because if the optimum is in those regions and therefore poorly predicted, without the exploration it may never be tested. This will also allow to improve the quality of the model in areas not already covered by the dataset.

3) Acquisition function for the NNSGO: δ_{min} will therefore be used to construct the acquisition function for NNSGO. For this purpose, the problem (1) is still considered with initial samples $D^k : \{X^n, f(X^n)\}_{n=1}^k$ drawn from an unknown function f. The function f is modelled by a NN model $\hat{f}(X)$ which is trained with the initial samples D^k. We then choose the following expression for the acquisition function $A(X)$ in the NNSGSO procedure:

$$A(X) = \beta\delta_{min}(X) + \hat{f}(X). \tag{7}$$

Compared to the UCB (3), $\hat{f}(X)$ plays the role of the GP's μ and serves for the exploitation purpose, δ_{min} serves the exploration purpose just like σ in (3) and β is a trade-off hyper-parameter which balances the search between exploration and exploitation. First of all, β must be chosen in a way that both the exploration and exploitation terms have the same order of magnitude. Afterwards, the value of β can be increased or decreased to permit more exploration or more exploitation, respectively. In our procedure, the first $N_{it} - 1$ points to evaluate ,with N_{it} the maximum number of iterations, are chosen by achieving the trade-off between the exploration and exploitation objectives (*i.e.* by maximisation of the acquisition function (7)). Since the real function can only be evaluated once after the final iteration, performing exploration in the objective of improving the model is no longer useful. The last point to evaluate is thus, chosen solely through the exploitation objective *i.e.* by finding the point X for which the NN estimate of f is the largest. The main steps of the NNSGO are described in algorithm 2.

4) Distance metrics in higher dimensions: The NNSGO requires the use of distance metrics. However, in higher dimensions when measuring similarities between points, they tend to provide poor contrasts between farthest and nearest neighbours to an evaluated point [13]. This makes the proposed procedure less efficient in higher dimensions. Nevertheless, two solutions can be considered in order to overcome this issue :

- Use fractional distance metrics which give better contrast between farthest and nearest neighbours to an evaluated point, as defined in [13], instead of the Euclidean distance.
- Increase the term δ_{min} to the power α (with $\alpha > 1$) to increase the contrast. Note that this approach will only be efficient if the evaluated δ_{min} is greater than 1.

IV. Applications

In this section, it is illustrated through different case studies, that the proposed NNSGO can be used as an alternative to other different smart grid based optimisation procedures such as BO. The first study case is a classical control problem which consists in determining the optimal controller in a

Algorithm 2 NN Smart grid optimisation algorithm

for $j = 0, ..., N_{it} - 1$ **do**

1. Train neural network model $\hat{f}(X)$ with available set of samples $D^{k+j} : \{X^n, f(X^n)\}_{n=1}^{k+j}$

2. Build acquisition function $A_j(X) = \beta\delta_{min}(X) + \hat{f}(X)$

3. The next point X^{k+j} to evaluate on f is

$$X^{k+j} = \arg\max_X A_j(X)$$

4. Evaluate X^{k+j} on f and add $(X^{k+j}, f(X^{k+j}))$ to the training samples D^{k+j}.

end for

The final point to evaluate on f is :

$$X^{k+N_{it}} = \arg\max_X \hat{f}(X)$$

Evaluate $X^{k+N_{it}}$ on f and add $(X^{k+N_{it}}, f(X^{k+N_{it}}))$ to the training sample $D^{k+N_{it}}$.

return X with the highest $f(X)$ from $D^{k+N_{it}}$

particular situation while the second one consists in using the same technique but into an aerodynamic domain. In the latter, the objective will be to find an optimal geometry for a projectile. The final case is on the optimisation of the Rosenbrock function, in a case where the initially available dataset is large. Note that the three case studies are minimisation problems but will be presented as maximisation problems to match the theory presented in the previous sections. For all these case studies, the NNSGO is compared to BO. Before performing the procedures, a preliminary study is carried out to find the structures of the NN and of the GP models that best fit the data using *e.g* a grid search approach [14]. The number of iterations is chosen depending on the study case and more particularly the cost to evaluate each function to optimize (these functions are considered expensive to evaluate, therefore only a limited number of iterations is performed). Finally, the values of β in algorithm 1 (BO) and algorithm 2 (NNSGO) are chosen to have the same order of magnitude between the exploration and exploitation terms.

The NN and GP models are built using *Scikit-learn* [15], a famous Python machine learning library which offers the possibility to build simple MLP regression models and GPs. For more advanced NN models, tools such as *PyTorch* or *Keras* could also be used. The optimisation of different acquisition functions is achieved using the Limited-memory BFGS [16] method in its *Scipy* [17] implementation. *Scipy* is also used to construct the transfer functions used in the controller design case. Simulations are performed on a PC equipped with a 12th Gen Intel CORE i7-12700H processor with 14 cores and 64 GB of RAM.

A. Controller design example

In this section, the NNSGO procedure is applied to design a controller for an unknown system. We consider the same study case as in [3] but without including the safety constraints.

The unknown system is considered to be a linear system described by the following transfer function :

$$G(s) = \frac{10}{(s+10)(s+1)}. \tag{8}$$

The goal of this study is to find an optimal controller $K(s,\theta)$, parametrized with a parameter vector $\theta[k, p_1, p_2]^T$, in a closed-loop framework. The controller $K(s,\theta)$ has the following transfer function :

$$K(s,\theta) = \frac{k(p_1+s)(p_2+s)}{s(s+4.2)}, \tag{9}$$

and is designed to track a reference noted y_{ref}. This reference corresponds to the unit step response of the model M_{ref} defined as :

$$M_{ref}(s) = \frac{9}{s^2+4.2s+9}. \tag{10}$$

Since the transfer function of the system $G(s)$ is considered to be unknown, the optimal controller parameters can not be accessed directly but have to be obtained through an optimisation algorithm. In the following, the NNSGO algorithm is performed to directly estimate the controller parameters. A BO approach will also be carried out to compare the results with NNSGO.

To illustrate the methods presented in this paper the following experiment is conducted :

- A unit step response of 5 seconds is applied to the closed-loop system.
- From this step response, $n = 200$ samples are collected at a sampling rate of $T_s = 0.025s$.

The criterion to be maximised by the NNSGO and BO, in order to estimate the optimal controller, is defined as:

$$V(\theta) = \sum_{n=1}^{200} (y_{ref}(n) - y(\theta, n))^2, \tag{11}$$

where $y_{ref}(n)$ is the desired output at sample point n, and $y(\theta, n)$ is the obtained output at sample point n with a given parameter vector θ. The optimal controller parameters are thus estimated by solving the following optimisation problem:

$$\theta_{opt} = \arg\max_{\theta} -V(\theta). \tag{12}$$

Unlike in BO where the optimisation can be performed without any initial samples, neural networks need to be provided with a sufficient number of samples in order to efficiently estimate the value of a function. To solve this optimisation problem using the NNSGO, 50 controller parameters θ are chosen randomly to initialize the NN model. Among these controllers, the parameters θ_D which perform the best (*i.e.* for which $V(\theta)$ is the lowest) are for $V(\theta_D) = 0.7$. The procedure is then performed using algorithm 2 with the following considerations:

- The number of iterations is set to 250.
- The neural network model used is a multi layer perceptron with 1 hidden layer with 32 neurons and a *reLu* activation function on each neuron.
- The maximum of the acquisition function provides the next controller to evaluate and is given as :

$$A(\theta) = -\hat{V}(\theta) + \beta\delta_{min}(\theta), \tag{13}$$

  where $\hat{V}(\theta)$, is the multi layer perceptron's estimate of $V(\theta)$ and $\beta = 1$.

The NNSGO is compared to BO which is performed using algorithm 1 with the following considerations :

- The first controller to evaluate is chosen randomly
- The number of iterations is set to 300. The 300 iterations are chosen here to have the exact same number of observed points as with the NNSGO procedure (50 initial samples + 250 tested).
- A GP model with an RBF kernel (2) is used.
- The maximum of the acquisition function which provides the next controller to evaluate corresponds to the UCB (3) where $\mu(\theta)$ is the GP estimate of $-V(\theta)$, $\sigma(\theta)$ the standard deviation provided by the GP model and $\beta = 1$.

The great advantage of these two approaches (NNSGO and BO) is that the optimal controller parameters are estimated without requiring the identification of a model for the process to be controlled. Indeed, the MLP and GP both model the static relationship between the parameter vector θ and the performance of the controller on the model: $V(\theta)$.
After performing 250 iterations of NNSGO using algorithm 2, the best controller parameters among the tested ones are $\theta_{NNSGO} = [1, 8, 1.1]^T$ with $V(\theta_{NNSGO}) = 0.02$. By using the NNSGO procedure, we are able to find a controller θ_{NNSGO} for which the value of V is 98% lower than the V of θ_D (the controller with the lowest value of V in the initial dataset). The best controller found with BO after performing algorithm 1 for 300 iterations is $\theta_{BO} = [1, 8.4, 0.98]^T$ with $V(\theta_{BO}) = 0.01$. The time to perform the optimisation for both approaches is of approximately 5 seconds each (Note that this time can slightly vary depending on the time that *Scipy* takes to calculate the response of a particular tested controller). In this case study, BO performs better than the NNSGO procedure since it finds a controller with a smaller value of V than the one found using the NNSGO procedure in the same optimisation duration. However, the controller provided by the NNSGO procedure remains close to the optimal controller parameters which are $\theta_{opt} = [0.9, 10, 1]$.
Fig. 2 presents the obtained response with the best controllers found using NNSGO (θ_{NNSGO}) and using BO (θ_{BO}), compared to the target response associated to M_{ref} (10). As it can be seen, the response of the system when θ_{NNSGO} is used as the controller parameters is also close to the target response. Therefore, the procedure can be used as an alternative to BO for this controller design problem.

B. Aerodynamic design

The aim of the second example is to illustrate that the NNSGO method can also be used for other applications such as aerodynamic design. This study consists in finding the optimal dimensions of a particular type of projectile named *fin stabilized*, which minimise a criteria associated to aerodynamic coefficients (coefficients associated to the forces and moments applied to a projectile during its flight. More information about the case study can be found in [18]). There are no readily available physical models that give the relationship between the aerodynamic coefficients and the dimensions of the projectile. Nevertheless, it remains possible to find the coefficients associated to a certain geometry of a projectile. This is usually achieved through experimental tests (free flight or wind tunnel tests) or numerical simulations. Both possibilities used to quantify the coefficients come at an expensive cost (time or resources needed). Therefore, it is crucial to find a smart way to optimize the geometry by performing the least possible tests. We therefore, propose to use the presented NNSGO procedure to this aerodynamic optimisation. The optimisation is also performed using BO for comparison.
To perform this study, we have access to a database containing different values of the projectile geometry dimensions noted X, (where $X \in \mathbb{R}^N$ with $N = 5$) for different Mach numbers (Ratio between the speed of an object moving through a fluid and the speed of sound in the same fluid) associated to different aerodynamic coefficients. The number of available samples is 2592. They are generated from a simulation tool to test the efficiency of the procedure for this application.
The criteria to optimise to find the optimal dimensions in this study, is the average value of the drag coefficient, noted C_{A_0}, over 8 values of the Mach number between 2 and 5, contained in the vector M. It is denoted as : $f_o(X)$ (for objective function) and is given by :

$$f_o(X) = \frac{1}{8} \sum_{l=1}^{8} C_{A_0}(X, M_l), \tag{14}$$

where M_l is the l^{th} Mach number in the vector M. Using the available samples, the average drag $f_o(X)$ of each configuration X, is computed. This provides a dataset of different configurations X directly associated to the average value of the drag coefficient $f_o(X)$. This new dataset contains 324 samples.
The description of the dimensions X of the projectile and the search domain $\mathcal{X}$ are detailed in table I. The unit *caliber* corresponds to the dimension with respect to the diameter of the body of the projectile.

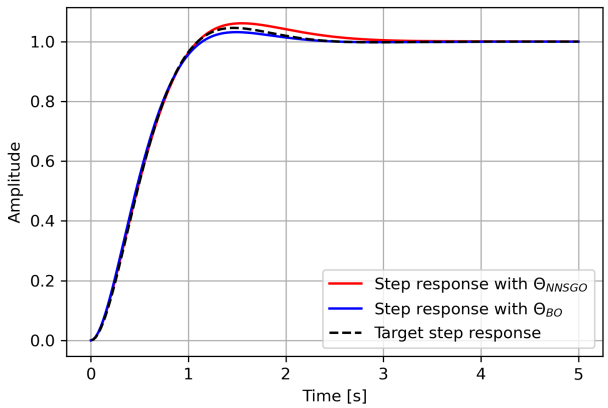

Fig. 2: Target step response (in black) compared to the step response based on the controllers found using the NNSGO approach (θ_{NNSGO} in red) and BO (θ_{BO} in blue)

TABLE I: Boundaries for each design parameter X of the geometry

Dimension X	Minimum	Maximum	unit
X_1 : Total Length	10	25	caliber
X_2 : Nose Angle	10	34.5	degrees
X_3 : Fins Height	0.5	2.5	caliber
X_4 : Fins Width	0.5	1.72	caliber
X_5 : Position of fins	0	1	caliber

This leads to the following optimisation problem :

$$X_{opt} = \arg\max_{X \in \mathcal{X}} -f_o(X). \quad (15)$$

The optimisation problem (15) could be analysed geometrically. Indeed, the projectiles with the lowest drag are the ones which have the lowest possible values for each dimensions. In this case, X_{opt} will be the lowest possible values for each X in our search domain $\mathcal{X}$ which are : $X_{opt} = (10, 10, 0.5, 0.5, 0)$. For the purpose of the study, it is considered that this optimal projectile is unknown and can not be found using a geometrical analysis. Algorithm 2 is therefore used to solve the optimisation problem (16). The goal is to see if we are able, using a smart grid based procedure, to find configurations close to X_{opt} in a limited budget.

To solve the optimisation problem (16), the NNSGO is performed using algorithm 2 with the following considerations:

- The number of iterations is set to 20.
- The neural network model used is a multi layer perceptron with 1 hidden layer with 32 neurons and a *reLu* activation function on each neuron.
- The maximum of the acquisition function provides the next point X to evaluate and is given as :

$$A(X) = -\hat{f}_o(X) + \beta\delta_{min}(X), \quad (16)$$

  where $\hat{f}_o(X)$, is the multi layer perceptron's estimate of $f(X)$ and $\beta = 1$.

The NNSGO is compared to BO which is performed using algorithm 1 with the following considerations :

- The number of iterations is set to 20.
- A GP with a Matern kernel [2] is used.
- The maximum of the acquisition function which provides the next controller to evaluate corresponds to the UCB (3) where $\mu(X)$ is the GP estimate of $-f_o(X)$, $\sigma(X)$ the standard deviation provided by the GP model and $\beta = 2$.

In the initial dataset, the projectile with the lowest value of the average drag f_o, noted X_D, has a $f_o(X_D) = 0.28$ and the optimal projectile X_{opt} has a $f_o(X_{opt}) = 0.24$. After testing 20 projectile configurations X using the NNSGO, we are able to find the optimal configuration X_{opt}. This is also the case for BO where the optimal configuration is among the 20 tested projectiles. Moreover, the time needed to run 20 iterations of both methods (including the time to test the configurations on the simulator) is 219 seconds and 221 seconds, for BO and NNSGO respectively. Therefore, for this study case, both methods perform similarly.

The smart grid based procedures are then compared to a brute force optimisation. In the latter, 60 projectiles are drawn randomly from a uniform distribution in the search domain. The values of f_o are then computed with the simulator used to generate the database employed in the study. The goal is to see, if the NNSGO and BO procedures perform *better i.e.* find configurations with a lower f_o, than when generating randomly the configurations to evaluate. Among the 60 tested projectiles using the brute force approach, the configuration X with the lowest average drag has an average drag $f_o(X) = 0.28$. Therefore, the use of the smart grid based procedure is justified in this study case. Indeed, we are not able to find the optimal configurations by increasing the dataset with 60 configurations using a brute force approach whereas by increasing the dataset with 3 times less (20) configurations using the NNSGO or BO, the optimal configuration is found.

C. Optimisation with a larger initial dataset

This section follows on from section III-B.2: the objective is to perform the optimisation of the 12 dimensional form of the Rosenbrock function $f(X)$ (6). This function has one global minimum at $X = (1, 1, ..., 1)$ for $f(X) = 0$. The optimisation is performed here in the search domain $\mathcal{X} = \{X = (x_1, .., x_{12})^T | -2.048 < x_i < 2.048 (i = 1, ..., 12)\}$ where $X \in \mathbb{R}^{12}$.

For the purpose of the study it is considered that the function is unknown and expensive (in time and resources) to evaluate. However, we have access to a dataset containing 5000 samples (pairs $\{X^n, f(X^n)\}_{n=1}^k$ with $k = 5000$). This dataset is generated randomly in a corner of the search domain which does not include the global minimum to motivate the use of a smart grid based approach.

The goal of this section will be to find the optimum X_{opt} of the following optimisation problem :

$$X_{opt} = \arg\max_{X \in \mathcal{X}} -f(X). \quad (17)$$

To solve this optimisation problem, the NNSGO is performed using algorithm 2 with the following considerations :

- The number of iterations is set to 100.
- The NN model used is a multi layer perceptron with 1 hidden layer with 256 neurons and a *reLu* activation function on each neuron.
- The maximum of the acquisition function provides the next point X to evaluate and is given as :

$$A(X) = -\hat{f}(X) + \beta\delta_{min}(X), \quad (18)$$

  where $\hat{f}(X)$, is the multi layer perceptron's estimate of $f(X)$ and $\beta = 1$.

The NNSGO is compared to BO which is performed using algorithm 1 with the following parameters :

- The number of iterations is set to 100.
- A GP with a Matern kernel [2] is used.
- The maximum of the acquisition function which provides the next controller to evaluate corresponds to the UCB (3) where $\mu(X)$ is the GP estimate of $-f(X)$, $\sigma(X)$ the standard deviation provided by the GP model and $\beta = 2$.

TABLE II: Optimisation results

Method used	Optimisation time	Value of optimum
NNSGO	40 seconds	$f(X_{NNSGO}) = 40$
BO	1 hour	$f(X_{BO}) = 38$

Table II presents the values of f for the optimum points found using the NNSGO procedure (noted X_{NNSGO}) and BO (noted X_{BO}), as well as the time to run 100 iterations of both procedures. As shown in this table, both methods perform similarly in terms of optimums found. In the initial dataset, the point X_D, for which the value of the Rosenbrock function is the lowest is for the value $f(X_D) = 226$. By increasing the dataset with 100 new points using the two smart grid based approaches (BO and NNSGO), we are able to find points X for which the value of $f(X)$ is 84% smaller than the original smallest value in the dataset.

However, for this example the NNSGO has a clear advantage in terms of the computational time needed to perform 100 iterations. This is due to the fact that neural networks scale better in training time than regular GPs when the size of the dataset increases. In this particular case, the NN takes 8 seconds to train whereas the GPs take 30 seconds. Furthermore, after each iteration of the NNSGO, the previously trained model parameters are re-used as an initialization for the new model. This further decreases the training time of the updated NN models. This is achieved by using the *warm start* option in scikit-learn multi layer perceptron regressor implementation. Without using this option, the optimisation when using the NNSGO procedure takes 12 minutes, which is still 6 times less than when using the regular BO, to perform 100 iterations and similar optimums are found. (Note that this option is not available for GP regression models in scikit-learn).

V. Conclusions and perspectives

In this study, an alternative method to Bayesian optimisation for the optimisation of costly unknown functions is proposed. It is based on the use of neural networks instead of Gaussian processes. This method uses the distance between points, instead of the usual uncertainties provided by Gaussian processes, to search for a potential optimum to be tested next on the unknown function. The proposed procedure is tested on different case studies and provides competitive results to regular Bayesian optimisation. It is also illustrated that when the initial dataset contains a large number of samples, the optimisation procedure using the proposed approach will require less time to perform a certain number of iterations than the classic Bayesian optimisation algorithm. The next step will be to improve our solution by extending it to optimisation problems under constraints.

References

[1] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[2] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer school on machine learning*, pp. 63–71, Springer, 2003.

[3] X. Bombois and M. Forgione, "Control design via Bayesian Optimization with safety constraints," *Conference on Control Technology and Applications*, 2022.

[4] S. Bansal, R. Calandra, T. Xiao, S. Levine, and C. J. Tomlin, "Goal-driven dynamics learning via bayesian optimization," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 5168–5173, 2017.

[5] S. Kuindersma, R. Grupen, and A. Barto, "Variational bayesian optimization for runtime risk-sensitive control," *Robotics: Science and Systems VIII*, pp. 201–208, 2012.

[6] R. Guzman, R. Oliveira, and F. Ramos, "Heteroscedastic bayesian optimisation for stochastic model predictive control," *IEEE Robotics and Automation Letters*, vol. 6, no. 1, pp. 56–63, 2020.

[7] R. Marchant and F. Ramos, "Bayesian optimisation for intelligent environmental monitoring," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 2242–2249, IEEE, 2012.

[8] A. Bemporad, "Global optimization via inverse distance weighting and radial basis functions," *Computational Optimization and Applications*, vol. 77, no. 2, pp. 571–595, 2020.

[9] C. Andersson, A. H. Ribeiro, K. Tiels, N. Wahlström, and T. B. Schön, "Deep convolutional networks in system identification," in *2019 IEEE 58th conference on decision and control (CDC)*, pp. 3670–3676, IEEE, 2019.

[10] M. Forgione and D. Piga, "dynonet: A neural network architecture for learning dynamical systems," *International Journal of Adaptive Control and Signal Processing*, vol. 35, no. 4, pp. 612–626, 2021.

[11] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *arXiv preprint arXiv:0912.3995*, 2009.

[12] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[13] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," in *International conference on database theory*, pp. 420–434, Springer, 2001.

[14] P. Liashchynskyi and P. Liashchynskyi, "Grid search, random search, genetic algorithm: a big comparison for nas," *arXiv preprint arXiv:1912.06059*, 2019.

[15] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[16] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on scientific computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[17] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[18] M. Albisser, *Identification of aerodynamic coefficients from free flight data*. PhD thesis, Université de Lorraine, 2015.