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. Despite the effectiveness of this technique, it is interesting to investigate other different models to Gaussian Processes. In this paper an alternative to Bayesian Optimization based on the use of neural networks instead of Gaussian Processes is proposed. Since neural networks do not naturally provide an information about the quality of the estimates, the proposed method makes use of the distance between an evaluated point and the closest observed one to it for the exploration. The efficiency of the proposed approach is illustrated on two different study cases : controller design and aerodynamic design.**

I. INTRODUCTION

In many applications, 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 optimization 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 determine the next grid point, a model of the function $f(X)$ is identified based on the previous grid points $(X_i, f(X_i))_{i=1}^{k}$ 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 Optimization* (BO) [1] where the model of the function $f(X)$ takes the form of a *Gaussian Process* (GP) [2]. In this paper, we propose an alternative version of this approach where the model of the function is a neural network.

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 Optimization 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 Bayesian Optimisation 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 optimization procedure. As mentioned above, Gaussian Processes (GP) 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 $(X_i, f(X_i))_{i=1}^{k}$. Besides this estimate, the GP also provides an uncertainty region indicating the confidence we can have in the estimate. Note that this uncertainty region is certainly not an uncertainty region as the ones encountered in robust control. Indeed, there is no guarantee that the true function $f(X)$ lies in this uncertainty region. One can however say 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 maximizing a so-called *acquisition function* $A(X)$. The acquisition function $A(X)$ is a function of the GP model and of its uncertainty and its maximum (*i.e.*, the next grid point X_{k+1}) will be a point for which the GP model of $f(X)$ is high (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 this paper, the objective is to show that, instead of a GP model, a neural Network (NN) model can also be efficiently used in a smart-grid based optimization procedure. Neural networks are increasing in popularity and being applied in many fields such as system identification [8], [9]. It is clear that, unlike a GP model, a NN model is generally not accompanied by an uncertainty region. However, as mentioned above, the uncertainty of the GP can not also really be considered as an uncertainty region such as in robust control. Moreover it

[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

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. In addition, in a number of cases, a NN model can be easier to handle than a GP. For instance, basic GP are not suited to learn large quantity of samples *i.e* the number of pairs $(X_k, f(X_k))$, since GP scale cubically with the number of observations. This is mainly due to the fact that it implies solving heavy calculations on very large covariance matrices, which results into costly computations. This is indeed inconvenient in cases where BO needs to be performed using an initial large data set of available experiments. Gaussian Processes also encounter some issues for high dimensional problems. In [10], it is shown that in higher dimensions, distance metrics such as L_2 norm, used to calculate typical GP kernel matrices, become limited since they provide poor contrast between farthest and nearest neighbours. Rather, the proposed approach overcomes this drawback by applying some modifications on the proposed acquisition function. Despite the complexity of building GP in high dimensions, some studies have been achieved on how to overcome these issues. These strategies, summarized in [11], require modifications on the classic GPs for them to be employed for datasets of several dimensions. Among these strategies, dimensionality reduction approaches for high dimensional problems can be found as suggested in [12]. An alternative could also be to use other types of models to represent the function to optimize instead of Gaussian Processes as it is proposed in this paper. Only a few work can be found in the literature where other types of models are used in smart grid based optimisation procedures similar to BO. A study worth mentioning where the procedure is similar to the one discussed in this paper is presented in [13]. In the latter, Inverse Distance Weighting interpolation and Radial Basis functions are successfully used as models of the objective functions. Similarly to our procedure, the exploration is also partly based on the distance between the evaluated and observed points. Some studies have also explored the use of particular neural networks which are known to perform well with large datasets and in high dimensions both in classification and regression problems. For example in [14], Bayesian neural networks are used for BO and perform well for hyper-parameter tuning. Nevertheless, building Bayesian neural networks can be computationally intensive. In the presented approach, we propose the use of multi layer perceptrons types of NN with only one hidden layer, which require less computational cost and are sufficient in different applications. This paper will focus on two of them : Control design and aerodynamic optimization.

The rest of the paper is structured as follows : First, the Bayesian Optimisation algorithm is presented in section III-A, then the proposed approach is presented in section III-B. Finally, the proposed method is illustrated through two applications : the design of a controller and the optimisation of a projectile's geometry in section IV.

II. Problem statement

This paper focuses on the optimisation of the form :

$$X_{opt} = \underset{X}{\operatorname{argmax}}\, f(X), \tag{1}$$

f is considered to be unknown, high dimensional and expensive (*i.e.* requires a lot of resources or time to obtain the value of $f(X)$ for a given X) to evaluate.

The goal of this paper is to answer the following problem : Given a dataset $D^k : \{X_i, f(X_i)\}_{i=1}^k$ obtained from an unknown function f, use a neural network as a model of the unknown function f in a smart grid based procedure such as Bayesian Optimisation, to search which promising candidates to evaluate to get close to X_{opt}.

III. Methodology

The proposed smart grid optimisation is inspired by a method named Bayesian Optimisation 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 which gives a promising candidate, to be close to the optimum X_{opt} of f, to evaluate on the function f. The acquisition function balances the search between exploitation (searching for the place where a model's estimate of f is high) and exploration (searching for the place where the uncertainties are the highest). Therefore, a model of the unknown function f and the uncertainties associated to each estimate of the model are needed to build the acquisition function. In this sense, all the available samples from f, $D^k : \{X_i, f(X_i)\}_{i=1}^k$, 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 distribution over functions defined by its mean $\mu_p(X)$ and a kernel matrix $K(X, X^{'})$. The Kernels are covariance matrices that measure the similarity between the samples 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 :

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

where d is the Euclidean distance and l a length scale parameter. Using Bayesian inference, the unknown function is approximated by a Gaussian distribution $\mathcal{N}(\mu(X), \sigma^2(X))$. At each iteration of the BO, the GP model is updated with the sample obtained after the maximisation of θ and the evaluation on f. 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. Note that the uncertainties provided by the GP which are used to build acquisition functions in BO depend on the chosen kernel (covariance matrix) and therefore are based on the similarity between an evaluated point and the points observed by the GP.

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 from 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 is used here to control the degree of the trade-off *i.e* when β is chosen high more importance will be given to the exploration and when it is chosen low, more importance will be given to the exploitation.

The BO algorithm using UCB as acquisition function to solve an optimisation problem (1) is described in Algorithm 1, with N_{it} the maximum number of iterations.

Algorithm 1 Bayesian Optimisation algorithm

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

1. Train Gaussian Process with available set of samples $D^{k+j} : \{X_i, f(X_i)\}_{i=1}^{k+j}$

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

$$X_{k+j} = \underset{X}{\operatorname{argmax}}\, A_j(X)$$

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

end for

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

As mentioned before, 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 NN based smart grid optimisation (NNSGO). Similarly to BO, the idea behind this method is that the point which should be evaluated as a promising candidate to be close to the optimum of the real function f should be a trade-off between the place where the estimate of f is at its optimum and where it is likely to differ from the real value of f, *i.e* the place where the uncertainties are the highest. The main difference between NNSGO and BO is that NN are employed as models of the functions rather than using probabilistic models such as GP.

1) Neural networks and a measure for their uncertainties: Neural networks (NN) are used for their flexibility and good adaptability to different sorts of problems. Many types of neural networks exist. In this paper, multi layer perceptrons (MLP), which are types of NN with fully connected layers are used as models. They provide the relation between a set of inputs X^m of dimension m associated to outputs $f(X)^l$ of dimension l, thanks to trained weights W.

A MLP needs to be trained to provide the best estimates of an unknown function. The training consists in finding the best values of weights 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 defined as

$$l = \frac{1}{k}\sum_{i=1}^{k} \|f_{nn}(X_k) - f(X_k)\|^2, \tag{4}$$

Where l is the loss function, k the number of samples, $f(X_k)$ the target output for X_k, $f_{nn}(X_k)$ the estimates of $f(X_k)$ by the NN and $\|A\| = \sqrt{A^T A}$ represents the Euclidean norm of a vector A. The most common method used to train a MLP is back-propagation which uses Gradient Descent [15] to iteratively update the weights of the NN. To use this method, the MLP needs to have differentiable activation functions such as $reLu$.

Unlike GP, regular NN do not come with uncertainties. Nevertheless, it remains possible to have an idea of the points X for which the estimate $f_{nn}(X)$ of the neural network is likely to differ from the real value $f(X)$. Well trained NNs are supposed to *generalise i.e.* to predict unseen points as well as observed ones. Even so, this will still be limited by the similarity between the observed and unseen points. When a point X to be predicted is far from the observed points, then it is likely that the prediction $f_{nn}(X)$ will differ from the real value $f(X)$. This is also the case in GP, where the uncertainties are based on the similarity between the evaluated points and the observed ones, as seen in section III-A. Therefore, this idea is used to create a measure which is employed for the exploration aspect of the proposed procedure. In that respect, $\delta_{min}(X)$ is defined as the distance between a point to be evaluated X and the closest point, depending on chosen distance metric, to it from the observed points in D_X^k. It is defined as :

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

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 $f(X)$, $f_{nn}(X)$ 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 Gaussian Process. To illustrate this, 5 random samples are generated from a function $f(x)$. A GP model and a NN model, are both identified using those samples. The GP and NN models are then used to predict

unseen and seen points. The following elements are then compared:

- The standard deviation obtained from the predictions of the GP.
- The δ_{min} (5) calculated by taking into account the 5 random samples.
- The error of the NN for each sample x determined as the absolute value of the difference between the estimate of the unknown function $f_{nn}(X)$ and the target value $f(x)$: $error = |f(x) - f_{nn}(x)|$.

Fig. 1 presents the results for the samples drawn from $f(x) = xsin(x)$.

As it can be seen on Fig. 1, even if the values of both the GP standard deviation σ and δ_{min} are slightly different, their behaviour is very similar. Indeed, at observed samples (marked by black dots), the GP σ is at its lowest and so is δ_{min}. The prediction quality of the NN (error) can also be compared to δ_{min}. Indeed, δ_{min} increases when the difference between the predicted and the target value also increases.

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 $D^k : \{X_i, f(X_i)\}_{i=1}^{k}$ drawn from an unknown function f. The function f is modelled by a neural network model f_{nn} which is trained with the initial samples D^k. In this case, a similar function to the UCB acquisition function (3) is optimised to choose the next point to evaluate on f at each iteration. It is given by:

$$A(X) = \beta\delta_{min}(X) + f_{nn}(X), \tag{6}$$

Compared to the UCB (3), $f_{nn}(X)$ 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). β is a trade-off hyperparameter which balances the search between exploration and exploitation. One way to choose the value of β, is to analyze the distribution of the data across the search space. In the case where the data do not cover uniformly the search space, then β could be chosen high to increase the exploration in the spaces not covered by the data.

The main steps of the SGO are described in algorithm 2.

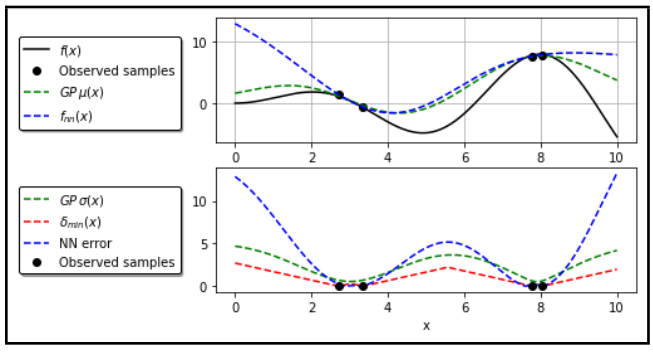

Fig. 1: Comparison between the GP σ (std), the δ_{min} and NN error

4) Distance metrics in higher dimensions: The NNSGO requires the use of distance metrics. As stated in the introduction, in higher dimensions when measuring similarities between points, distance metrics tend to provide poor contrasts between farthest and nearest neighbours to an evaluated point [10]. 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 [10], 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 δ_{min} which is evaluated is higher than 1.

IV. Applications

In this section it is illustrated through two different study cases, that the proposed NNSGO can be used as an alternative to other different smart grid based optimization procedures such as Bayesian Optimization. The first study case is a classical control problem which consists in determining the optimal controller in a particular situation while the second one consists in using the same technique but into an aerodynamic domain. In the latter, the objective is to find an optimal geometry for a projectile.

The neural network models are built using *Scikit-learn* [16], a famous Python machine learning library which offers the possibility to build simple multi layer perceptrons regression models. 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 Sequential Least Square Programming method in its *Scipy* [17]

Algorithm 2 NN Smart grid optimisation algorithm

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

1. Train neural network model f_{nn} with available set of samples $D^{k+j} : \{X_i, f(X_i)\}_{i=1}^{k+j}$

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

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

$$X_{k+j} = \underset{X}{\operatorname{argmax}}\, A_j(X)$$

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

end for

The final point to evaluate on f is :

$$X_{k+N_{it}} = \underset{X}{\operatorname{argmax}}\, f_{nn}(X)$$

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 $D^{k+N_{it}}$

implementation. *Scipy* is also used to construct the transfer functions used in the controller design case.

A. Controller design example

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

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

$$G(s) = \frac{10}{(s+10)(s+1)} \tag{7}$$

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

$$K(s,\theta) = \frac{k(p_1+s)(p_2+s)}{s(s+4.2)} \tag{8}$$

The controller is designed to track a reference model M_{ref} defined as :

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

Since the system's transfer function $G(s)$ 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, we propose to use the NNSGO algorithm to directly estimate the controller parameters. A Bayesian Optimisation algorithm will also be performed 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 minimized by the NNSGO and BO, in order to estimate the optimal controller, is defined as :

$$V(\theta) = \sum_{n=1}^{200} (y_{ref}(n) - y(\theta, n))^2, \tag{10}$$

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 parameter vector θ.

The optimal controller parameters are thus estimated by solving the following optimisation problem :

$$\theta_{opt} = \underset{\theta}{\operatorname{argmin}}\, V(\theta), \tag{11}$$

To solve this optimization problem, the NNSGO is performed using algorithm 2 with the following parameters :

- The number of iterations is set to 300.
- 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 first controller to evaluate is chosen randomly
- The maximum of the acquisition function provides the next controller to evaluate and is given as :

$$A(\theta) = -\hat{V}(\theta) + \beta\delta_{min}(\theta), \tag{12}$$

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 parameters :

- The number of iterations is set to 300.
- A Gaussian Process with an RBF kernel is used.
- The first controller to evaluate is also chosen randomly
- The maximum of the acquisition function provides the next controller to evaluate corresponds to the UCB (3) and is given as :

$$A(\theta) = -\mu(\theta) + \beta\sigma(\theta), \tag{13}$$

where $\mu(\theta)$, is the Gaussian Process estimate of $V(\theta)$, $\sigma(\theta)$ the standard deviation provided by the GP 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 300 iterations of NNSGO using algorithm 2, the best controller parameters among the tested ones are $\theta_{NNSGO} = [1, 9.1, 0.93]$ with $V(\theta_{NNSGO}) = 0.02$. The best controller found with BO after performing algorithm 1 for 300 iterations is $\theta_{BO} = [1, 7.9, 1]$ with $V(\theta_{BO}) = 0.03$. Therefore, the NNSGO proposed procedure performs similarly to the BO one for this study case. Moreover, both methods provide controller parameters, which are very close to $\theta_{opt} = [0.9, 10, 1]$.

Fig. 2 presents the obtained response with the best controller found θ_{NNSGO} and θ_{BO}, using NNSGO and BO respectively, compared to the target response associated to M_{ref}. As it can be seen, the response of the system when θ_{NNSGO} is used as the controller parameters is close to the target response. Therefore, the procedure can be used as an alternative to BO for this controller design problem.

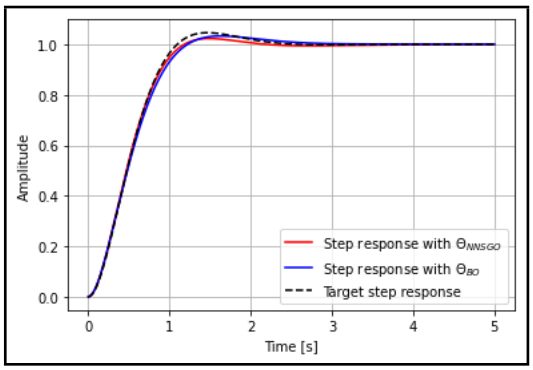

Fig. 2: Target step response vs Step response based on the controllers θ_{NNSGO} and θ_{BO}

B. Aerodynamic design

The aim of the second example is to illustrate the interest of using the NNSGO method in an aerodynamic framework. This study consists in finding the optimal dimensions of a particular type of projectile named *fin stabilized*. The optimal dimensions are obtained by minimizing a criteria associated to aerodynamic coefficients (coefficients associated to the forces and moments applied to a projectile during its flight [18]). There are no readily available physical models that give the relationship between the aerodynamic coefficients and the projectile's dimensions. 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 critical to find a smart way to optimize the geometry by doing the least possible tests. We therefore, propose to use the presented NNSGO procedure to this aerodynamic optimization.

To perform this study, we are using a database containing different values of the projectile geometry dimensions noted X, for different Mach numbers noted m associated to different aerodynamic coefficients. The number of available samples available is 2600. They are generated from an available simulation tool to test the efficiency of the procedure for this application.

The criteria to minimize to find the optimal dimensions in this study, is the average value of the drag coefficient C_{A_0} over 8 values of the Mach number m between 2 and 5. 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.

The description of the dimensions X of the projectile and their search boundaries are detailed in the table I. The unit *caliber* corresponds to the dimension with respect to the size of the diameter of the projectile's body.

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 optimization problem :

$$X_{opt} = \underset{X}{\operatorname{argmin}}\, f_o(X). \tag{15}$$

The optimisation problem (15) could be analysed geometrically. Indeed, the projectiles with the lowest drag (without considering any stability constraints) are the ones which have the lowest possible values for each dimensions. In this case, X_{opt} will be the lowest possible dimensions in our search boundaries 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 (15). The goal is to see if we are able, using the proposed procedure, to find configurations close to X_{opt} in a limited budget. These configurations found using the NNSGO procedure should also have a lower $f_o(X)$, than the one with the lowest f_o in the initial database.

A multi layer perceptron model with 1 hidden layer and 128 neurons in each layer with a *reLu* activation function on each neuron is identified with the available database. It takes as inputs the geometry parameters X and the Mach number m. Its ouput is the the aerodynamic coefficient C_{A_0}. This model provides an estimation of the static relationship between X, m and C_{A_0} : $\hat{F}(X, m) = \hat{C}_{A_0}$.

The maximum of the acquisition function provides the next geometry to evaluate and is given as :

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

where $\hat{f}_o(X)$ is the estimated average value of C_{A_0} over 8 Mach numbers using the predictions of the MLP for C_{A_0} : $\hat{f}_o(X) = \frac{1}{n}\sum_{l=1}^{8} \hat{C}_{A_0}(X, m_l)$. β is equal to 1. Algorithm 2 is ran for 20 iterations. Each new configuration is evaluated on the simulator used to generate the database.

In the initial dataset, the projectile with the lowest f_o, noted X_D, has a $f_o(X_D) = 0.28$ and the optimal projectile X_{opt} has a $f_o(X_{opt}) = 0.24$. Among the 20 tested projectiles, the one with the lowest f_o is obtained when $X_{best} = (10, 10, 0.5, 1.6, 0.9)$ for $f_o(X_{best}) = 0.25$.

Fig. 3 presents the best projectile X_{best} found among the 20 iterations compared to the best one in the dataset and the known optimum. As shown on this figure, by applying the procedure described in this paper, it is possible to find a configuration with a lower C_{A_0}, for different Mach numbers, than those in the initial dataset. Moreover, the best projectile found has a mean value of C_{A_0} close to the optimum one.

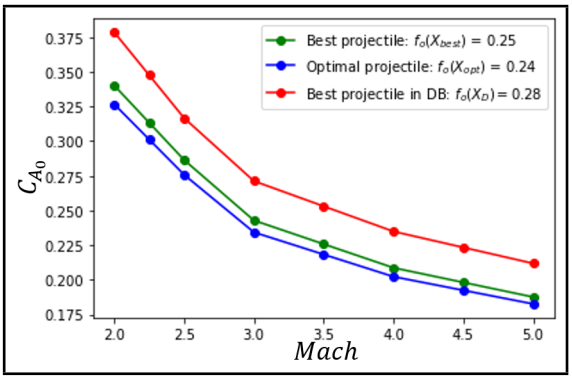

Fig. 3: C_{A_0} for best geometry in the database X_D vs the one found using the proposed procedure X_{best} vs the known optimal projectile X_{opt}

The NNSGO procedure is then compared to a brute force optimisation. In the latter, 60 projectiles are drawn randomly from a uniform distribution in the search space. The values of f_o are then computated with the simulator used to generate the database employed in the study. The goal is to see, if the procedure performs *better i.e.* finds configurations with a lower f_o, than when generating randomly the configurations to evaluate.

Fig. 4 presents the values of $f_o(X)$, found using the brute force optimisation (red bars) and using the NNSGO procedure (green bar). As it can be seen on this figure 4, among the 60 tested projectiles using the brute force approach, none of them has a lower f_o than the best configuration obtained using the NNSGO procedure, where only 20 configurations were tested. Therefore, the proposed procedure provides better results than a brute force procedure in this study case.

V. Conclusions and Perspectives

In this study, an alternative 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 the evaluated point and the closest observed one to it, instead of uncertainties to search for a potential optimum to be tested next on the unknown function. The proposed procedure is tested on two different study cases and provides efficient results in both domains. The next step will be to improve our solution by extending it to optimization problems under constraints.

VI. Acknowledgement

This work is funded by the French-German Research Institute of Saint-Louis (ISL). The database used for the study is provided by ISL.

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.

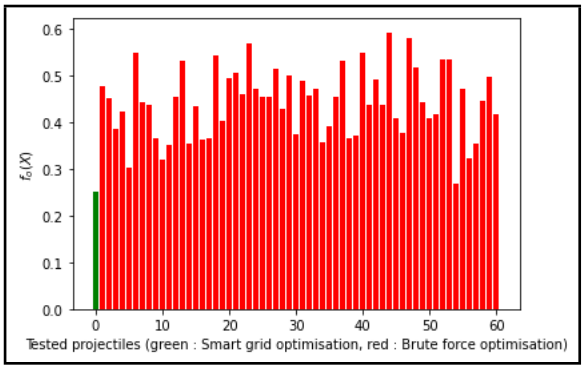

Fig. 4: Smart grid optimisation (green) vs Brute force optimisation (red)

[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] 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.

[9] 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.

[10] 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.

[11] M. Binois and N. Wycoff, "A survey on high-dimensional gaussian process modeling with application to bayesian optimization," *arXiv preprint arXiv:2111.05040*, 2021.

[12] R. Tripathy, I. Bilionis, and M. Gonzalez, "Gaussian processes with built-in dimensionality reduction: Applications to high-dimensional uncertainty propagation," *Journal of Computational Physics*, vol. 321, pp. 191–223, 2016.

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

[14] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams, "Scalable bayesian optimization using deep neural networks," in *International conference on machine learning*, pp. 2171–2180, PMLR, 2015.

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

[16] 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.

[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.