Explainable Hyperparameters Optimization using Hilbert-Schmidt Independence Criterion

Paul Novello, Gaël Poëtte, David Lugato, Pietro M Congedo

Explainable Hyperparameters Optimization using Hilbert-Schmidt Independence Criterion

Paul Novello paul.novello@cea.fr
Gaël Poëtte gael.poette@cea.fr
David Lugato david.lugato@cea.fr
CEA Cesta
15 avenue des Sablières
Le Barp, 33114, France

Pietro M. Congedo pietro.congedo@inria.fr
Inria Saclay
1 Rue Honoré d'Estienne d'Orves, 91120 Palaiseau
Palaiseau, 91120, France

Editor:

Abstract

Tackling new machine learning problems with neural networks always means optimizing numerous hyperparameters that define their structure and strongly impact their performances. In this work, we use a robust system conception approach to build explainable hyperparameters optimization. This approach is defined by the research of a parametrization of a system (a neural network) to optimize its output response (the prediction error) to an input stimulation (the test set). To that end, we study the use of Hilbert-Schmidt Independence Criterion (HSIC), a probability distribution dependence measure widely used for sensitivity analysis in robust system conception, in the context of Hyperparameters Optimization. Hyperparameters spaces can be complex and awkward, with different natures of hyperparameters (categorical, discrete, boolean, continuous), interactions and inter dependencies, which makes it non trivial to apply HSIC. We alleviate these difficulties to make HSIC applicable in that context and obtain an analysis tool that quantifies the relative impact of hyperparameters on a Neural Network's final error. Notably, we show how this knowledge allows obtaining competitive neural networks that are naturally much more cost effective.

1. Introduction

Hyperparameters Optimization is ubiquitous in Machine Learning (ML), and especially in Deep Learning (DL), where Neural Networks are often cluttered with many hyperparameters. Finding good hyperparameters is mandatory in DL applications to real world ML tasks, but can be fastidious for different reasons. First, the high number of hyperparameters by itself makes this problem challenging. Moreover, their impact on error changes very often depending on the problem, so it is difficult to enact general best practices and permanently recommend hyperparameter values for every ML problem. Finally, hyperparameters can be of very different natures, like continuous, discrete, categorical or boolean. This lead to very complex hyperparameters space, and this complexity is even more worsened by complex

relations between hyperparameters, like conditionality or interactions.

Many techniques have been introduced to tackle this problem. Random search (Bergstra and Bengio, 2012) is quite easy to implement but only relies on *a priori* sampling and does not use any knowledge about hyperparameters space. Bayesian Optimization (Shahriari et al., 2016) exploits *a posteriori* knowledge obtained after an initial random search but is less robust and may fail if too high dimension search spaces. Recently introduced Neural architecture search techniques (Kandasamy et al., 2018; Pham et al., 2018; Tan et al., 2018) have proven to be effective. Yet, their implementation can be tedious - they often involve Reinforcement Learning which itself comes with hyperparameters - and not adapted to every situations. More importantly, all these methods are black boxes: we do not have any feedback on the relative importance of hyperparameters, while this could be helpful to understand the results, to learn about NN structure for a given problem and to refine HO algorithms in order to better adapt them to a given problem.

In this work, we formalize HO as a robust system conception problem. Robust system conception aims at designing complex systems under many constraints such as budget, feasibility or operating time constraints. To achieve that, practitioners improve their knowledge of descriptive variables, very often by performing Sensibility analysis (SA). Neural networks can be seen as a complex system, and hyperparameters as descriptive variables, so SA is completely adapted to hyperparameters study. Hence, we introduce the use of this approach in the context of HO. We select a powerful metric used for SA, called Hilbert-Schmidt Independence Criterion (HSIC) Gretton et al. (2005), which is a distribution dependence measure initially used for two-sample test problem (Gretton et al., 2007). After an initial random search, HSIC value can be estimated for each hyperparameter and gives hints at their importance in the HO.

We first provide a quick overview of main HO algorithms (Section 2). Then, after a presentation of HSIC and its advantages for HO (Section 3, 4), we thoroughly study the application of HSIC in this context (Section 5). This task is highly non trivial due to the complex structure of hyperparameters space. First, hyperparameters can be discrete (width of the NN), continuous (learning rate), categorical (activation function) or boolean (batch normalization). Second, some hyperparameters presence is conditional to others (moments decay rates specific to ADAM optimizer). Third, they can strongly interact (as shown in Tan and Le (2019), in some cases, it is better increasing depth and width by a similar factor). The metric should be able to compare reliably hyperparameters in such situations. We introduce solutions to overcome these obstacles and to be able to apply HSIC in these complex situations. The efficiency of these solutions are illustrated on simple toy examples but also on more complex ML tasks. Finally, in Section 6, we highlight several possible usages of HSIC for HO. First, HSIC allows to better understand hyperparameters relative importance and to possibly focus research efforts on specific hyperparameters. We also identify that hyperparameters that have an impact on execution speed are not the only one impacting the error. Based on all this knowledge, we introduce ways of reducing hyperparameters variation range to improve stability of HO and execution speed. Finally, We construct a full HSIC based HO methodology, where we optimize hyperparameters in two steps: one for the most important hyperparameters, and another for the remaining hyperparameters. Its efficiency is validated on real world problems (MNIST, Cifar10 and

a Physical Sciences data set), where we obtain competitive errors with up to 500 less NN parameters (weights and biases) and FLOPs.

2. Overview of Hyperparameters Optimization

In this section, we provide a short review of the main HO techniques, and highlight their advantages and common drawbacks.

A first approach to optimize hyperparameters is to uniformly explore the search space. This can be done through a Grid Search (GS) or a Random Search (RS). The main difference between the two methods is that for GS, hyperparameters values are chosen on an uniform grid. These values are deterministic whereas for RS, hyperparameters values are randomly sampled from a uniform distribution, in a Monte Carlo fashion. The main advantages of RS over GS is that it allows for a more efficient exploration of the hyperparameters search , and that it is not constrained to a grid, so does not suffer from the curse of dimensionality (which is a problem here since the hyperparameters can be quite numerous). See Bergstra and Bengio (2012) for a thorough comparison of these two methods. The common costly part of these two methods is that it requires to train a NN for each hyperparameters configuration so exploring the search space can be computationally very costly.

Some methods aim at reducing the cost of such searches. For instance, Successive Halving (Jamieson and Talwalkar, 2016) and Hyperband (Li et al., 2018) trains NNs in parallel, like in GS or RS, and stop their training after a certain number of epochs. They then choose the best half of NNs and carry on the training only for these NNs, for another same number of epochs and so on. This allows testing more hyperparameters values for a same computational budget. On the contrary, the next methods are designed to improve the quality of the search with a limited number of training.

Bayesian Optimization is based on the approximation of the loss function by a surrogate model. After an initial uniform sampling of hyperparameters configurations, the surrogate model is trained on these points, and used to maximize an acquisition function. This acquisition function, often chosen to be expected improvement or upper confident bound (Shahriari et al., 2016), is supposed to lead to hyperparameters configurations that will improve the error. Therefore, it focuses the computation on potentially better hyperparameters values instead of randomly exploring the hyperparameters space. The surrogate model can be a Gaussian Process (Snoek et al., 2012), a kernel density estimator (Bergstra et al., 2011) or even a NN (Snoek et al., 2015).

One of the numerous challenges of hyperparameters selection, especially in Deep Learning, is to formalize the search space so that optimization algorithms can be conducted on it. Indeed, popular model based HO are not easily and naturally applicable for conditional or categorical hyperparameters that often appear when optimizing a NN architecture. For instance, such categorical hyperparameter can be the type of convolution layer for a Convolutional Neural Network, regular convolution or depth-wise convolution (Chollet, 2016), and a conditional hyperparameter could be the specific parameters of each different convolution type. The remaining type of HO method we describe is Neural Architeture Search. Dating back to evolutionary and genetic algorithms (Stanley and Miikkulainen, 2002), it has been the subject of many recent works. For instance, Kandasamy et al. (2018) models the

architecture as a graph, or Pham et al. (2018); Tan et al. (2018) use reinforcement learning to automatically construct representations of the search space. See Elsken et al. (2019) for an exhaustive survey of this field.

Previous methods are end to end algorithms that only require the user to specify n_h input hyperparameters. They simply return the best NN and the user does not have to interact with the algorithm during its execution. It has many automating advantages. However, this lack of interactivity can bring some drawbacks. First, these methods do not give any insights on the relative importance of hyperparameters, whereas it may be of interest in a first approach to an ML problem. These methods are black box and not explainable. Second, one could have other goals than test accuracy of a NN, like execution speed or memory consumption. Some works like Tan et al. (2018) introduce multi objective HO, but it requires additional tuning of the HO algorithm itself. Finally, there may be flaws in the hyperparameters space, e.g. a totally useless hyperparameter that could be dropped but is included in the search space and becomes a nuisance for the optimization. This is all the more problematic since some algorithms, like Gaussian Process based BO, suffer from the curse of dimensionality. The drawbacks can be summed up as:

1 Lack of explainability.

2 Difficulties in a multi objective context.

3 Unnecessary search space complexity.

Instead of applying classical HO algorithms as such, we could tackle the HO problem using a new approach that is broadly used in the engineering field, for complex systems conception, and that naturally addresses the previous problems: Robust System Optimization.

3. Problem Formalization

In this section, we emphasize the fact that HO is closely related to what is commonly called Robust System Optimisation, see for example Spagnol et al. (2018). In Robust System Optimisation, systems are defined by variables, that describe their structure, and their output response to an input stimulation. Such systems can be an engine, a wind turbine, a power plant... The aim is to find optimal values for the descriptive variables, in order to improve the response of the system. It is very similar to HO because the Neural Network can be seen as a system and hyperparameters as system descriptive variables that need to be optimized in order to improve the output response which is a metric based on the error. One powerful tool to help achieving this goal is Sensitivity Analysis (SA) (Razavi et al., 2021). SA aims at:

1 Analyzing the relative importance of variables for the response.

   $\rightarrow$ Explainability: we could explain and understand hyperparameters impact better.

2 Selecting values for input variables that facilitate the conception without affecting the response.

→ Multi-objective: we could select values that improve execution speed without hurting the error too much.

3 Identifying where to put conception efforts in order to improve the response.

→ Complexity: we could focus on less hyperparameters to optimize, by knowing which of them mostly impact the error.

SA consists in studying the effects of some input variables on an output quantity of interest. We study the sensitivity of a function $f : \mathcal{X}_1 \times ... \times \mathcal{X}_{n_h} \to \mathbb{R}$ to its inputs. We consider:

- the input vector $\boldsymbol{\sigma} = (X_1, ..., X_{n_h})$, with $X_i \in \mathcal{X}_i$, $X_i \sim d\mathbb{P}_{X_i}$ and X_i pairwise independent $\forall i \in \{1, ..., n_h\}$ ($X_i \sim d\mathbb{P}_{X_i}$ to be read X_i follows a distribution of probability measure $d\mathbb{P}_{X_i}$).
- The output vector $Y = f(\boldsymbol{\sigma}) = f(X_1, ..., X_{n_h})$.

In the context of HO, $\boldsymbol{\sigma}$ would be the vector containing a hyperparameters configuration, and f would be the error of the corresponding NN. The goal of SA, like mentioned previously, is to study the impact of each X_i, or group of $X_j, ..., X_k$ on $f(\boldsymbol{\sigma})$. More specifically, it aims at detecting significant and insignificant variables by measuring the individual (or joint) contributions of each variable X_i to the variations of $f(\boldsymbol{\sigma})$. To that end, different types of sensibility measures can be estimated after an initial sampling of n_s input vectors $\{\boldsymbol{\sigma}_1, ..., \boldsymbol{\sigma}_{n_s}\}$ and their corresponding output values, $\{f(\boldsymbol{\sigma}_1), ..., f(\boldsymbol{\sigma}_{n_s})\}$.

A first type of metric gives information about the contribution of an input variable to the output based on variance analysis. The most common metric used for that purpose are Sobol indices (Sobol, 1993) but they only assess the contribution of variables to the output variance. Goal oriented Sobol indices (Fort et al., 2016) or uncertainty importance measure (Borgonovo, 2007) construct quantities based on the output whose variance analysis gives more detailed information. However, computing these indices can be very costly since estimating them with an error of $O(\frac{1}{\sqrt{n_s}})$ requires $(n_h + 2) \times n_s$ sample evaluations (Saltelli, 2002), which can be prohibitive for HO.

Another type of metrics, called dependence measures, assesses the dependence between an input X_i variable and the output $f(\boldsymbol{\sigma})$ (Da Veiga, 2013). It relies on the claim that the more X_i is independent of $f(\boldsymbol{\sigma})$ the less important it is to explain it. Dependence measures are based on dissimilarity measures between $\mathbb{P}_{X_i}\mathbb{P}_Y$ and $\mathbb{P}_{X_i,Y}$, where $X_i \sim \mathbb{P}_{X_i}$ and $Y = f(\boldsymbol{\sigma}) \sim \mathbb{P}_Y$, since $\mathbb{P}_{XY} = \mathbb{P}_Y\mathbb{P}_X$ when X and Y are independent. In Da Veiga (2013), the author gives several examples of indices based on dissimilarity measures like f-divergences (Csizar, 1967) or integral probability metrics (Müller, 1997). These indices are easier and less expensive to estimate (n_s trainings instead of $(n_h + 2) \times n_s$) than variance based measures since they only need a simple Monte Carlo design of experiment. In the context of HO, it could be possible to compute such indices after a classical RS to assess the effect of hyperparameters on the loss, which makes it more suitable for this work. In the present paper, we use one of these metrics, Hilbert-Schmidt Independence Criterion (HSIC), that we describe and adopt in the next sections.

4. Hilbert-Schmidt Independence Criterion

Hilbert Space Information Criterion (Gretton et al., 2005) (HSIC) is one of the dependance measures that can be used for SA (Da Veiga, 2013). It is built on a distance called Maximum Mean Discrepancy (MMD) (Gretton et al., 2007). In this section, we describe its construction and its advantages.

4.1 From Integral Probability Metrics to Maximum Mean Discrepancy

Let X and Y be two random variables of probability distribution $\mathbb{P}_X$ and $\mathbb{P}_Y$ defined in $\mathcal{X}$. Gretton et al. (2007) show that distributions $\mathbb{P}_X = \mathbb{P}_Y$ if and only if $\mathbb{E}_X[f(X)] - \mathbb{E}_Y[f(Y)] = 0$ for all $f \in C(\mathcal{X})$, where $C(\mathcal{X})$ is the space of bounded continuous functions on $\mathcal{X}$.

This lemma explains the intuition behind the construction of Integral Probability Metrics (IPM) (Müller, 1997). Let $\mathcal{F}$ be a class of functions, $f : \mathcal{X} \to \mathbb{R}$. An IPM γ is defined as

$$\gamma(\mathcal{F}, \mathbb{P}_X, \mathbb{P}_Y) = \sup_{f \in \mathcal{F}} (\mathbb{E}_X[f(X)] - \mathbb{E}_Y[f(Y)]). \tag{1}$$

Then, the Maximum Mean Discrepancy (MMD) can be defined as an IPM restricted to a class of functions $\mathcal{F}_{\mathcal{H}}$ defined on the unit ball of a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$ of kernel $k : \mathcal{X}^2 \to \mathbb{R}$. In Gretton et al. (2005), this choice is motivated by the capacity of RKHS to efficiently embed probability distributions. The authors define μ_X such that $\mathbb{E}_X(f(X)) = \langle f, \mu_X \rangle_{\mathcal{H}}$ as the mean embedding of $\mathbb{P}_X$. Then, $\gamma_k^2(\mathbb{P}_X, \mathbb{P}_Y)$ can be written

$$\begin{aligned} \gamma_k^2(\mathbb{P}_X, \mathbb{P}_Y) &= \|\mu_X - \mu_Y\|_{\mathcal{H}}^2. \\ &= \int\int k(x_1, x_2)(p_X(x_1) - p_Y(x_1))(p_X(x_2) - p_Y(x_2))dx_1 dx_2 \\ &= \mathbb{E}_{XX'}[k(X, X')] + \mathbb{E}_{YY'}[k(Y, Y')] - 2\mathbb{E}_{XY}[k(X, Y)], \end{aligned} \tag{2}$$

where $p_X(x)dx = d\mathbb{P}_X(x)$ and $p_Y(x)dx = d\mathbb{P}_Y(x)$. After a Monte Carlo sampling of $\{X_1, ..., X_{n_s}\}$ and $\{Y_1, ..., Y_{n_s}\}$, $\gamma_k^2(\mathbb{P}_X, \mathbb{P}_Y)$ can thus be estimated by $\hat{\gamma}_k^2(\mathbb{P}_X, \mathbb{P}_Y)$, with

$$\hat{\gamma}_k^2(\mathbb{P}_X, \mathbb{P}_Y) = \sum_{j=1}^{n_s}\sum_{l=1}^{n_s} k(X_j, X_l) + \sum_{j=1}^{n_s}\sum_{l=1}^{n_s} k(Y_j, Y_l) - 2\sum_{j=1}^{n_s}\sum_{l=1}^{n_s} k(X_j, Y_l), \tag{3}$$

and $\hat{\gamma}_k^2(\mathbb{P}_X, \mathbb{P}_Y)$ being an unbiased estimator, its standard error can be estimated using the empirical variance of $\hat{\gamma}_k^2(\mathbb{P}_X, \mathbb{P}_Y)$.

4.2 The kernel choice

Formula (2) involves to choose a kernel k. In practice, k is chosen among a class of kernels that depends on a set of parameters $\mathbf{h} \in \mathbf{H}$. We therefore denote the kernel by $k_{\mathbf{h}}$. Examples of kernels are the Gaussian Radial Basis Function $k_h : (x, y) \to \exp(-\frac{||x-y||^2}{2h^2})$ or the Matérn function $k_{\mathbf{h}} : (x, y) \to \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{||x-y||}{\eta}\right)^{\nu} K_\nu \left(\sqrt{2\nu}\frac{||x-y||}{\eta}\right)$, where $\mathbf{h} = \{\sigma, \nu, \eta\}$. In Fukumizu et al. (2009), the authors study the choice of the kernel, and more importantly of

the kernel parameters $\mathbf{h}$. They state that, for the comparison of probabilities $\mathbb{P}_X$ and $\mathbb{P}_Y$, the final parameter $\mathbf{h}^*$ should be chosen such that

$$\gamma^2_{k_{\mathbf{h}^*}}(\mathbb{P}_X, \mathbb{P}_Y) = \sup_{\mathbf{h}\in\mathbf{H}} \gamma^2_{k_{\mathbf{h}}}(\mathbb{P}_Y, \mathbb{P}_Y).$$

The authors suggest to focus on unnormalized kernel families, like Gaussian Radial Basis Functions $\left\{k_h : (x, y) \to \exp(-\frac{||x-y||^2}{2h^2}), h \in (0, \infty)\right\}$, also used in Da Veiga (2013), for which they demonstrate that $\hat{\gamma}^2_{k_{\mathbf{h}^*}}(\mathbb{P}_X, \mathbb{P}_Y)$, defined as

$$\hat{\gamma}^2_{k_{\mathbf{h}^*}}(\mathbb{P}_X, \mathbb{P}_Y) = \sup_{\mathbf{h}\in\mathbf{H}} \left[\sum_{j=1}^{n_s}\sum_{l=1}^{n_s} k_{\mathbf{h}}(X_j, X_l) + \sum_{j=1}^{n_s}\sum_{l=1}^{n_s} k_{\mathbf{h}}(Y_j, Y_l) - 2\sum_{j=1}^{n_s}\sum_{l=1}^{n_s} k_{\mathbf{h}}(X_j, Y_l) \right], \quad (4)$$

is a consistent estimator of $\gamma^2_{k_{\mathbf{h}^*}}(\mathbb{P}_X, \mathbb{P}_Y)$. It is thus possible to choose $\mathbf{h}$ by maximizing $\hat{\gamma}^2_{k_{\mathbf{h}^*}}(\mathbb{P}_X, \mathbb{P}_Y)$ with respect to $\mathbf{h}$. Therefore, in this work, we use Gaussian Radial Basis Functions kernel. Once $\mathbf{h}^*$ is chosen, $\hat{\gamma}^2_{k_{\mathbf{h}^*}}(\mathbb{P}_X, \mathbb{P}_Y)$ approximation error can also be estimated like in Section 4.1. It is important to note that both $\gamma^2_{k_{\mathbf{h}^*}}(\mathbb{P}_X, \mathbb{P}_Y)$ and optimal $\mathbf{h}$ can be estimated in a $\mathcal{O}(n_s^2)$ computational complexity, which is not expansive given usual values of n_s in HO context. To simplify the notations, we denote $k_{\mathbf{h}^*}$ by k in the following sections.

4.3 Hilbert-Schmidt Independence Criterion Definition (HSIC)

Let $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, and $\mathcal{G}$ the RKHS of kernel $k : \mathcal{X}^2 \times \mathcal{Y}^2 \to \mathbb{R}$. HSIC can be written

$$HSIC(X, Y) = \gamma^2_k(\mathbb{P}_{XY}, \mathbb{P}_Y\mathbb{P}_X) = \|\mu_{XY} - \mu_Y\mu_X\|_{\mathcal{G}}. \quad (5)$$

HSIC measures the distance between $\mathbb{P}_{XY}$ and $\mathbb{P}_Y\mathbb{P}_X$ embedded in $\mathcal{H}$. Indeed, since $X \perp Y \Rightarrow \mathbb{P}_{XY} = \mathbb{P}_Y\mathbb{P}_X$ so the closer these distributions are, in the sense of γ_k, the more independent they are.

4.4 HSIC for Hyperparameters Optimization

In Spagnol et al. (2018), the authors present a goal oriented sensitivity analysis by focusing on the sensitivity of f w.r.t. X_i when $Y = f(X_1, ..., X_{n_h}) \in \mathbf{Y}$, with $\mathbf{Y} \subset \mathbb{R}$. The sub-space $\mathbf{Y}$ is chosen based on the goal of the analysis. In the context of optimization, for instance, $\mathbf{Y}$ is typically chosen to be the best percentile of Y. To achieve this, the authors introduce a new random variable, $Z = \mathbb{1}_{Y\in\mathbf{Y}}$. Then,

$$HSIC(X_i, Z) = \mathbb{P}(Z = 1)^2 \times \gamma^2_k(\mathbb{P}_{X_i|Z=1}, \mathbb{P}_{X_i}), \quad (6)$$

so $HSIC(X_i, Z)$ measures the distance between X_i and $X_i|Z = 1$ (to be read X_i conditioned to $Z = 1$) and can be used to measure the importance of X_i to reach the sub-space $\mathbf{Y}$ with f. Using the expression of γ_k given by equation (2), its exact expression is

$$HSIC(X_i, Z) = \mathbb{P}(Z = 1)^2 \Big[\mathbb{E}_{X_iX_i'}[k(X_i, X_i')] + \mathbb{E}_{ZZ'}[k(Z, Z')] - 2\mathbb{E}_{X_iZ}[k(X_i, Z)] \Big]. \quad (7)$$

It is estimated for each X_i using Monte Carlo estimators denoted by $S_{X_i,\mathbf{Y}}$, based on samples $\{X_{i,1},...,X_{i,n_s}\}$ from $X_i \sim d\mathbb{P}_{X_i}$ and corresponding $\{Z_1,...,Z_{n_s}\}$, defined as

$$\begin{aligned} S_{\mathbf{Y},X_i} = \mathbb{P}(Z=1)^2 \Bigg[& \frac{1}{m^2} \sum_{j=1}^{n_s} \sum_{l=1}^{n_s} k(X_{i,j}, X_{i,l}) \delta(Z_j = 1) \delta(Z_l = 1) \\ & + \frac{1}{{n_s}^2} \sum_{j=1}^{n_s} \sum_{l=1}^{n_s} k(X_{i,j}, X_{i,l}) \\ & - \frac{2}{n_s m} \sum_{j=1}^{n_s} \sum_{l=1}^{n_s} k(X_{i,j}, X_{i,l}) \delta(Z_l = 1) \Bigg], \end{aligned} \tag{8}$$

with $m = \sum_k^{n_s} \delta(Z_k = 1)$ and $\delta(x) = 1$ if x is True and 0 otherwise. We will use this metric in the following. In HO, $\mathbf{Y}$ could be chosen to be the sub-space for which $f(X_1,...,X_{n_h})$ is in the best percentile p of a metric (L_2 error, accuracy,...), say $p = 10\%$. Then HSIC, and so $S_{X_i,\mathbf{Y}}$, would measure the importance of each hyperparameter X_i for obtaining the 10% best NNs.

HSIC have two advantages that make it stand out from other sensibility indices and make it particularly suitable for this work. First, expression (8) emphasizes that it is possible to estimate a HSIC using simple Monte Carlo estimation. In the context of HO, it could be possible to compute such indices after a classical RS to assess the effect of hyperparameters on the loss, even if it was not initially conducted for HSIC estimation. Second using (6), HSIC allows to easily perform goal oriented SA, i.e. to assess the importance of each hyperparameter for reaching $\mathbf{Y}$ (that can be the top percentile p of any metric measuring the error of a NN). In this section, we have mainly summed up the mathematics on which the sensitivity indices are based and how they are used in practice in a SA context. In order to apply it in HO context, some additional difficulties must be overcome.

5. Tackling complex hyperparameters spaces

So far, we have presented a metric, HSIC, that can be used to assess the independence of two probability distributions. There are still some concerns to consider before being able to apply it for Sensibility Analysis in HO. Figure 1 gives a graphical representation of a possible hyperparameters space and its structure. It emphasizes that hyperparameters may have complex relations:

- they do not live in the same measured space. Some are continuous (`weights_decay` $\in [10^{-6}, 10^{-1}]$), some are integers (`n_layers` $\in \{8,...,64\}$), others are categorical (`activation` $\in \{$`relu`$,...,$`sigmoid`$\}$), or boolean (`dropout` $\in \{True, False\}$).
- They could interact with each others. For instance `batch_size` adds variance on the objective function optimized by `optimizer`.
- Some hyperparameters are not involved for every configurations, e.g. `dropout_rate` is not used when `dropout = False` or `adam_beta` is only involved when `optimizer =`

`adam`. We call these hyperparamaters "conditional" hyperparameters and the others "main" hyperparameters.

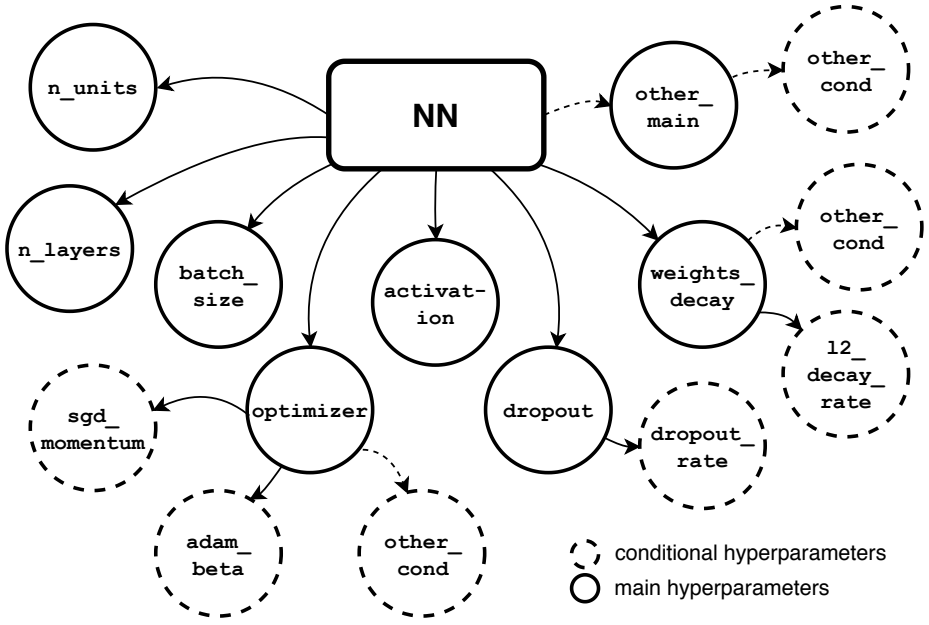


Figure 1: Example of hyperparameters space

In the following sections, we suggest original solutions to the above practical difficulties to be able to apply HSIC for Sensibility Analysis in HO. To illustrate these solutions, we consider as toy example the approximation by a Fully connected NN of Runge function $f : x \to \frac{1}{1+15x^2}$, $x \in [-1, 1]$ which is a historical benchmark of approximation theory. We consider $n_h = 14$ different hyperparameters (see **Appendix A** for details on hyperparameters space). For this problem, we randomly draw $n_s = 10000$ hyperparameters configurations, and perform the corresponding training on 11 training points. We record the test error on a test set of 1000 points. All samples are equally spaced between 0 and 1. We are aware that training NNs for 10000 different hyperparameters configurations is not realistic, but in this special toy problem it allows observing the asymptotical behaviour of HSIC estimation.

In Section 5.1, we introduce a transformation to be able to compare HSIC for hyperparameters that do not live in the same measured space. Then, in Section 5.2 we explain how to use HSIC to evaluate interactions between hyperparameters. Finally, in Section 5.3 we deal with conditionality between hyperparameters. These paragraphs are also strewn with even simpler pedagogical examples to emphasize the stakes of being able to handle the previously described problems.

5.1 Normalization of hyperparameters space

Hyperparameters can be defined on very different spaces. For instance, the activation function is a categorical variable that can be `relu`, `sigmoid` or `tanh`, `dropout_rate` is a continuous variable between 0 and 1 while `batch_size` is an integer that can go from 1 to hundreds. Moreover, for practical reasons, it may be useful to sample hyperparameters

	X_1	X_2
$S_{X,\mathbf{Y}}$	3.7×10^{-3}	4.8×10^{-3}

Table 1: $S_{X,\mathbf{Y}}$ values for X_1 and X_2

	U_1	U_2
$S_{X,\mathbf{Y}}$	4.8×10^{-3}	4.8×10^{-3}

Table 2: $S_{X,\mathbf{Y}}$ values for U_1 and U_2

with a non uniform distribution (e.g. log-uniform for `learning_rate`). This can undesirably affect HSIC and its interpretation, as we illustrate in the following example.

Example Let $f : [0,2]^2 \to \{0,1\}$ such that

$$f(X_1, X_2) = \begin{cases} 1 & \text{if } X_1 \in [0,1], X_2 \in [0,1], \\ 0 & \text{otherwise.} \end{cases}$$

Suppose we want to assess the importance of X_1 and X_2 for reaching the goal $f(X_1, X_2) = 1$ without knowing f. In the formalism of the previous section, we have $\mathbf{Y} = \{1\}$. Regarding its definition, X_1 and X_2 are equally important for f to reach $\mathbf{Y}$, due to their symmetrical effect. Let $X_1 \sim \mathcal{N}(1, 0.1, [0,2])$ (normal distribution of mean 1 and variance 0.1 truncated between 0 and 2) and $X_2 \sim \mathcal{U}[0,2]$. We compute $S_{X_1,\mathbf{Y}}$ and $S_{X_2,\mathbf{Y}}$ with $n_s = 10000$ points and display their value in Table 1. Values of $S_{X_1,\mathbf{Y}}$ and $S_{X_2,\mathbf{Y}}$ are quite different, and we could erroneously conclude that X_2 is more important than X_1.

This example shows that we have to ensure that $S_{X_i,\mathbf{Y}}$ and $S_{X_j,\mathbf{Y}}$ can be compared in order to say that hyperparameter X_i is more important than hyperparameter X_j. Indeed, if X_i and X_j do not follow the same distribution or $\mathcal{X}_i \neq \mathcal{X}_j$, it may be irrelevant to compare them directly. We need a method to obtain values for $S_{X_i,\mathbf{Y}}$ that are robust to the choice of $d\mathbb{P}_{X_i}$. To tackle this problem, we introduce a novel approach for comparing variables with HSIC. Let Φ_i be the CDF function of X_i. We have that $\Phi_i(X_i) = U_i$, with $U_i \sim \mathcal{U}[0,1]$. After an initial Monte Carlo sampling of hyperparameter X_i which can be a RS, we can apply Φ_i to each input point. Yet, one must be aware that to obtain $U_i \sim \mathcal{U}[0,1]$, its application is different for continuous and discrete variables:

- for continuous variables, $\Phi_i(X_i)$ is a bijection between $\mathcal{X}_i$ and $[0,1]$ so Φ_i can be applied on draws from X_i.
- For categorical, integer or boolean variables, $\Phi_i(X_i)$ is not a bijection between $\mathcal{X}_i$ and $[0,1]$. Suppose that X_i is a discrete variable with p possible values $\{X_i[1], ..., X_i[p]\}$, each with probability w_p. We can encode $\{X_i[1], ..., X_i[p]\}$ by $\{1, ..., p\}$. Then, $\Phi_i(X_i) = \sum_{j=1}^{p} w_j \mathbb{1}_{[X_i \leq j]}(X_i)$. When Φ_i is applied as is, $\Phi_i(X_i)$ is not uniform. To overcome that, a trick is commonly used in Monte Carlo resolution of Partial Differential Equations (Gillespie, 1976). One can simply use $U_i = \sum_{j=1}^{p} \mathcal{U}[\sum_{k<j} w_k, \sum_{k<j+1} w_k] \delta(X_i = j)$. As a result, $U_i \sim \mathcal{U}[0,1]$.

Finally, all we have to do is sampling X_i like in RS following the distribution we want, and then apply Φ_i to obtain U_i. The corresponding HSIC estimation is $S_{U_i,\mathbf{Y}}$. It only involves U_i and $U_i|Z = 1$ and since U_i are iid, the comparison of different $S_{U_i,\mathbf{Y}}$ becomes relevant. Coming back to the previous example, Table 2 displays values of $S_{U_1,\mathbf{Y}}$ and $S_{U_2,\mathbf{Y}}$. This time, the value is the same, leading to the correct conclusion that both variables are equally important. Note that in the following, we denote $S_{U_i,\mathbf{Y}}$ by $S_{X_i,\mathbf{Y}}$ for clarity but always have resort to this transformation.

Let us apply this methodology to Runge approximation HO problem. Note that in this toy example, hyperparameters are sampled uniformly and the usage of Φ_i is mostly motivated by the comparison of discrete vs continuous variables. Figure 2 displays a comparison between $S_{X_i,\mathbf{Y}}$ for hyperparameters of the Runge approximation problem, with $\mathbf{Y}$ the set of 10% best NNs. For readability, we order X_i by $S_{X_i,\mathbf{Y}}$ value in the legend and in the figure. We also display black error bars corresponding to HSIC estimation standard error. This graphic highlights that `optimizer` is by far the most important hyperparameter for this problem, followed by `activation`, `loss_function` and `n_layers`. Other hyperparameters may be considered as non impactful, because their $S_{X_i,\mathbf{Y}}$ values are low. Besides, these values are lower than the error evaluation so it could be only noise, and therefore these hyperparameters can not be ordered on this basis.

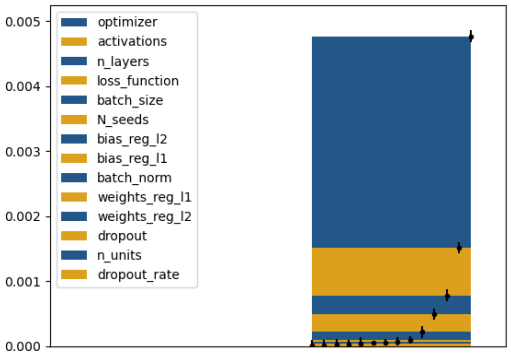


Figure 2: Comparison of $S_{X_i,\mathbf{Y}}$ for hyperparameters in Runge approximation problem.

5.2 Interactions between hyperparameters

If $S_{X_i,\mathbf{Y}}$ is low, it means that $\mathbb{P}_{X_i}$ and $\mathbb{P}_Z$ are similar (in the sense of HSIC). We could be tempted to conclude that X_i has a limited impact on Y. However, X_i may actually have an impact due to its interactions with the other hyperparameters. In other words, let X_i and X_j be two variables, it can happen that $S_{X_i,\mathbf{Y}}$ and $S_{X_j,\mathbf{Y}}$ are low while $S_{(X_i,X_j),\mathbf{Y}}$ is high. This point is illustrated in the next example.

Example For instance let $f : [0,2]^3 \to \{0,1\}$ such that

$$f(X_1,X_2,X_3) = \begin{cases} 1 & \text{if } X_1 \in [0,1], X_2 \in [1,2], X_3 \in [0,1], \\ 1 & \text{if } X_1 \in [0,1], X_2 \in [0,1], X_3 \in [1,2], \\ 0 & \text{otherwise.} \end{cases}$$

In that case, let $\mathbf{Y} = \{1\}$, $\forall x \in [0,2]$ we have $p_{X_2|Z=1}(x) = p_{X_2}(x)$ and $p_{X_3|Z=1}(x) = p_{X_3}(x)$. Hence, according to eq. 7 we have $HSIC(X_2,Z) = HSIC(X_3,Z) = 0$. However, we have

$$\begin{aligned} HSIC(X_1,Z) &= \mathbb{P}(Z=1)^2 \int_{[0,2]^2} k(x,x')\big[p_{X_1|Z=1}(x) - p_{X_1}(x)\big] \times \big[p_{X_1|Z=1}(x') - p_{X_1}(x')\big]dxdx' \\ &= \frac{1}{8}\Big[\int_{[0,1]\times[0,1]} k(x,x')dxdx' + \int_{[1,2]\times[1,2]} k(x,x')dxdx' - 2\int_{[0,1]\times[1,2]} k(x,x')dxdx'\Big], \end{aligned}$$

so for non trivial choice of k, $HSIC(X_1,Z) \neq 0$. One could deduce that X_1 is the only relevant variable for reaching $\mathbf{Y}$, but in practice it is necessary to chose X_2 and X_3 carefully as well. For instance, if $X_1 \in [0,1]$, $f(X_1,X_2,X_3) = 1$ if $X_2 \in [1,2]$ and $X_3 \in [0,1]$ but $f(X_1,X_2,X_3) = 0$ if $X_2 \in [1,2]$ and $X_3 \in [1,2]$. This is illustrated in Figure 3, which displays the histograms of X_1, $X_1|Z=1$,X_3, $X_3|Z=1$, X_3, $X_3|Z=1$, obtained from 10000

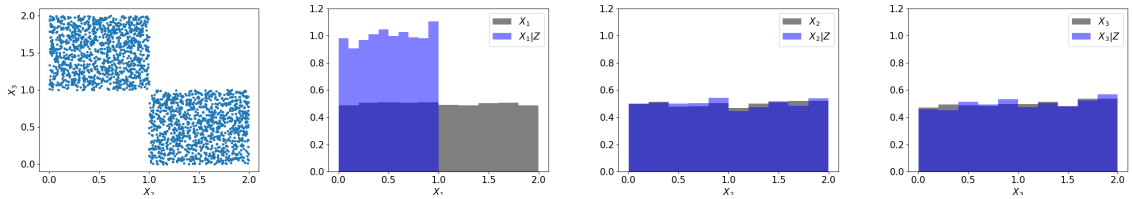


Figure 3: **From left to right: 1** - Pairs of $(X_2|Z=1, X_3|Z=1)$. **2** - Histogram of X_1 and $X_1|Z=1$. **3** - Histogram of X_2 and $X_2|Z=1$. **4** - Histogram of X_3 and $X_3|Z=1$.

points (X_1, X_2, X_3) sampled uniformly in the definition domain of f.

Histograms are the same for X_2, $X_2|Z=1$ and X_3, $X_3|Z=1$ (uniform between 0 and 2), but different for X_1, $X_1|Z=1$. Therefore, HSIC being a distance measure between X_1 and $X_1|Z=1$, it becomes intuitive that it will be high for X_1 and close to zero for X_2 and X_3, even if X_2 and X_3 are important as well because of their interaction. To assess this intuition, we compute $S_{X_1,\mathbf{Y}}$,$S_{X_2,\mathbf{Y}}$, $S_{X_3,\mathbf{Y}}$ and $S_{(X_2,X_3),\mathbf{Y}}$ after simulating f for $n_s = 2000$ points. We also computed $S_{(X_4,X_5),\mathbf{Y}}$, with X_4 and X_5 two dummy variables, uniformly distributed, to have a reference for $S_{(X_2,X_3),\mathbf{Y}}$. The results can be found in Table 3. They show that $S_{X_1,\mathbf{Y}}$ and $S_{(X_2,X_3),\mathbf{Y}}$ are of the same order while $S_{X_2,\mathbf{Y}}$, $S_{X_3,\mathbf{Y}}$ and $S_{(X_4,X_5),\mathbf{Y}}$ are two decades lower than $S_{X_1,\mathbf{Y}}$, which confirms that $S_{X,\mathbf{Y}}$ may be low while interactions are impactful.

	X_1	X_2	X_3	(X_2, X_3)	(X_4, X_5)
$S_{X,\mathbf{Y}}$	5.79×10^{-3}	2.41×10^{-5}	1.79×10^{-5}	1.39×10^{-3}	1.99×10^{-5}

Table 3: $S_{X,\mathbf{Y}}$ values for variables of the experiment

Additionally, we display the $S_{(X_i,X_j),\mathbf{Y}}$ for each pairs of variable X_i and X_j on Figure 4. We can see that for variables other than X_1, $S_{(X_i,X_j),\mathbf{Y}}$ is high only for $i=2$ and $j=3$. This example shows that it is necessary to compute $S_{X,\mathbf{Y}}$ of joint variables to perceive the importance of interactions between variables.

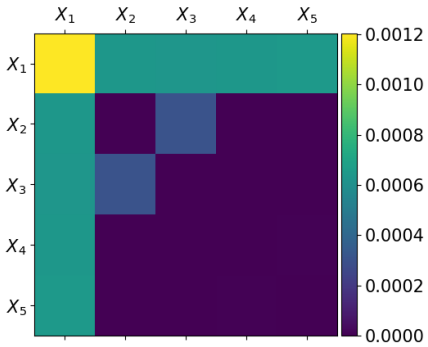
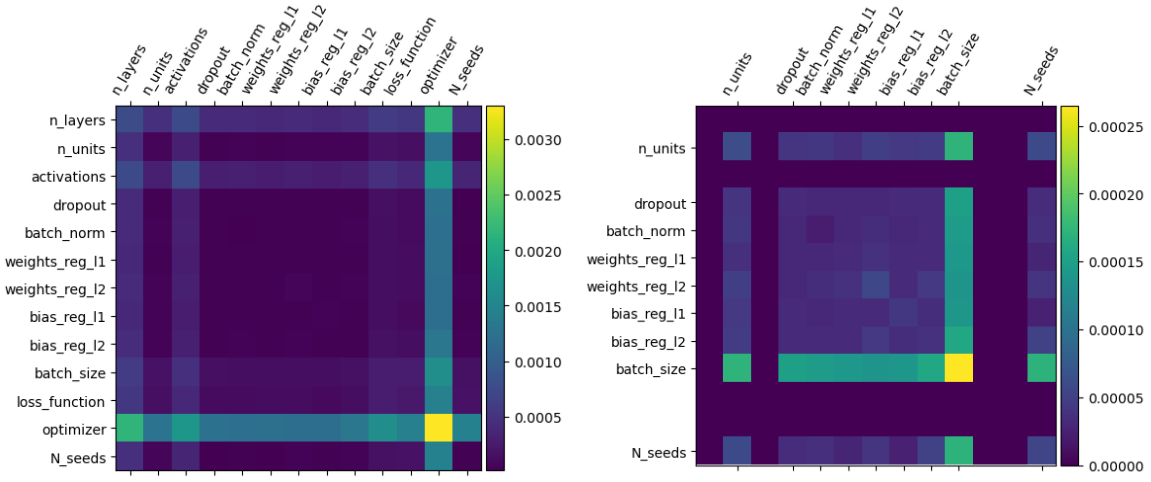


Figure 4: $S_{X,\mathbf{Y}}$ for each pairs of variable (lower diagonal).

These values are easy to interpret in this example because we know the behaviour of the underlying function f. In practice, $S_{X_1,\mathbf{Y}}$ and $S_{(X_2,X_3),\mathbf{Y}}$ can not be compared because (X_2, X_3) and X_1 do not live in the same measured space ($\mathcal{X}_2 \times \mathcal{X}_3$ and $\mathcal{X}_1$ respectively). Moreover, like we see on Figure 4, $S_{(X_i,X_j),\mathbf{Y}}$ is always the highest when $i=1$, regardless of j. In fact, if for a given variable X_i, $S_{X_i,\mathbf{Y}}$ is high, so will be $S_{(X_i,X_j),\mathbf{Y}}$ for any other variable X_j. Hence, care must be taken to only compare interactions of low $S_{X,\mathbf{Y}}$ variables with each others, and not with high $S_{X,\mathbf{Y}}$ variables. Coming back to Runge approximation example, Figure 5a displays the $S_{(X_i,X_j),\mathbf{Y}}$ for each pair of hyperparameters, and Figure 5b for each pair of hyperparameters, except for the impactful hyperparameters `optimizer`, `activation`, `n_layers` and `loss_function`.



(a) (b)

Figure 5: (a) $S_{(X_i,X_j),\mathbf{Y}}$ for each pair of hyperparameters. (b) $S_{(X_i,X_j),\mathbf{Y}}$ for each pair of hyperparameters, except for `optimizer`, `activation`, `n_layers` and `loss_function`

Figures 5a and 5b illustrate the remarks of the previous section. First, if we only look for interactions on Figure 5a, we would conclude that the most impactful hyperparameters are the only one to interact, and that they only interact with each others. Figure 5b shows that this conclusion is not true. Hyperparameter `batch_size` is the 5-th most impactful hyperparameter, and like we can see in Figure 2, is slightly above remaining hyperparameters. It is normal that $S_{(\texttt{batch\_size},X_j),\mathbf{Y}}$ is high, with X_j every other hyperparameters. However, $S_{(\texttt{batch\_size},\texttt{n\_units}),\mathbf{Y}}$ is higher, whereas `n_units` is the 13-th most impactful hyperparameter. This means that `batch_size` interacts with `n_units` in this problem, i.e. that when considered together, they contribute to explain the best results. Section 6 describes a situation where it is an important piece of information.

5.3 Conditionality between hyperparameters

Conditionality between hyperparameters, that often arises in Deep Learning, is a non trivial challenge in HO. For instance, hyperparameter "`dropout_rate`" will only be involved when hyperparameter "`dropout`" is set to `True`. Classically, two approaches can be considered. The first **(i)** splits the HO between disjoint groups of hyperparameters that are always involved together, like in Bergstra et al. (2011). Then, two separate instances of HO are created, one for the main hyperparameters, and another for `dropout_rate`. The second **(ii)** simply considers these hyperparameters as if they were always involved, even if they are not, like in Falkner et al. (2018). In that case, `dropout_rate` is always assigned a value even when `dropout` $= False$, and these dummy values are used in the optimization. First, we explain why these two approaches are not suited to our case and then we propose a third approach **(iii)**.

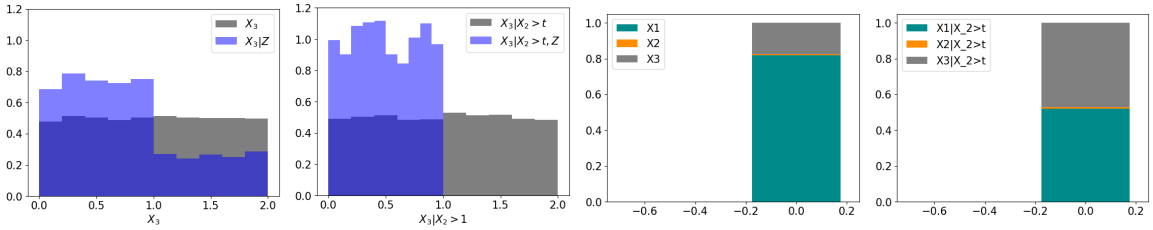
(i) The first formulation splits the hyperparameters between disjoints sets of hyperparameters whose value and presence are involved jointly in the training. In Runge approximation HO, it would mean to split the hyperparameters between two groups: {`dropout_rate`} and another containing all the others, since `dropout_rate` is the only conditional hyperparameter. This splitting approach is not suited to HSIC computation because it produces disjoints sets of hyperparameters, while we would want to measure the importance of every hyperparameter as compared to each other hyperparameter. As a result, `dropout_rate` could not be compared to any other hyperparameters.

(ii) In the second case, if we apply HSIC with the same idea, we could compute HSIC of a hyperparameter with irrelevant values coming from configurations where it is not involved. Two situations can occur. First, if a conditional variable X_i is never involved in the hyperparameters configurations that yield the p-percent best accuracies (depending on the percentile chosen), the values used for computing $S_{X_i,\mathbf{Y}}$, i.e. $X_i|Z=1$, are drawn from the initial, uniform distribution U_i. Then, $S_{X_i,\mathbf{Y}}$ will be very low, and the conclusion will be that it is not impactful for reaching the percentile, which is correct since none of the best NN have used this hyperparameter. However, if X_i is only involved in a subset of all tested hyperparameters configurations, and is really impactful in that case, $S_{X_i,\mathbf{Y}}$ would be lowered by the presence of the other artificial values of X_i drawn from the uniform distribution. In that case, we could miss its real impact. This phenomenon is illustrated on the following example.

Example. Let $f : [0,2]^3 \to \{0,1\}$ such that:

$$f(X_1,X_2,X_3) = \begin{cases} B & \text{if } X_1 \in [0,1], X_2 \in [0,t] \\ 1 & \text{if } X_1 \in [0,1], X_2 \in [1,2], X_3 \in [0,1], \\ 0 & \text{otherwise,} \end{cases}$$

With B a Bernoulli variable of parameter 0.5 and $t \in [0,2]$. Let $\mathbf{Y} = \{1\}$. In that case, X_1 plays a key role for reaching $\mathbf{Y}$, and X_3 is taken into account only when $X_2 > t$. Parameter t hence allows to control how much values of X_3 will be involved. We evaluate f on $n_s = 2000$ points uniformly distributed across $[0,2]^3$, first with $t = 1$.



(a) (b) (c) (d)

Figure 6: **(a)** - Histogram of X_3 and $X_3|Z$ **(b)** - Histogram of X_3 and $X_3|X_2 > t, Z$. **(c)** - $S_{X,\mathbf{Y}}$ for X_1; X_2 and X_3. **(d)** - $S_{X,\mathbf{Y}}$ for $X_1|X_2 > t$; $X_2|X_2 > t$ and $X_3|X_2 > t$.

Figure 6a compares the histograms of X_3 and $X_3|Z$. Figure 6b compares histograms of $X_3|X_2 > t$ and of $X_3|X_2 > t, Z$. This shows that the distribution of $X_2|Z$ is different if we choose to consider artificial values of X_3 or values of X_3 that are actually used by

$f\ (X_3|X_2 > t)$. Figures 6c and 6d show that relative values of $S_{X_1,\mathbf{Y}}$ and $S_{X_3,\mathbf{Y}}$ are quite different whether we chose to consider $X_2 > t$ or not, meaning that the conclusions about the impact of X_3 can be potentially different. To emphasize how different these conclusions can be, we compare $S_{X_1,\mathbf{Y}}$ and $S_{X_3,\mathbf{Y}}$ for different values of t. The results are displayed on Figure 7 (top row). Since the value of t controls how much artificial values there are for X_3, this demonstrates how different $S_{X_3,\mathbf{Y}}$ can be, depending on the amount of artificial points. This experiment emphasize the problem because in all cases, X_3 is equally important for reaching $\mathbf{Y}$ whereas for $t = 1.8$ we would be tempted to discard X_3.

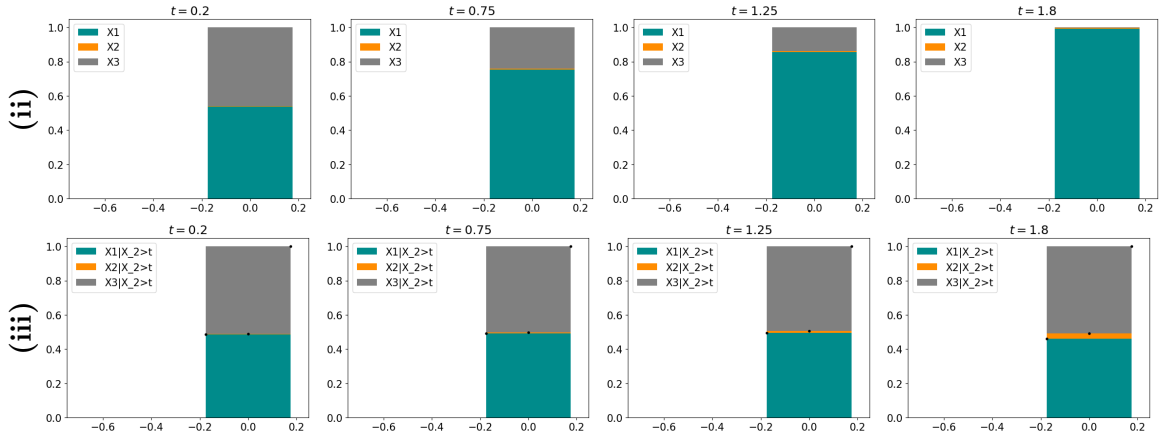
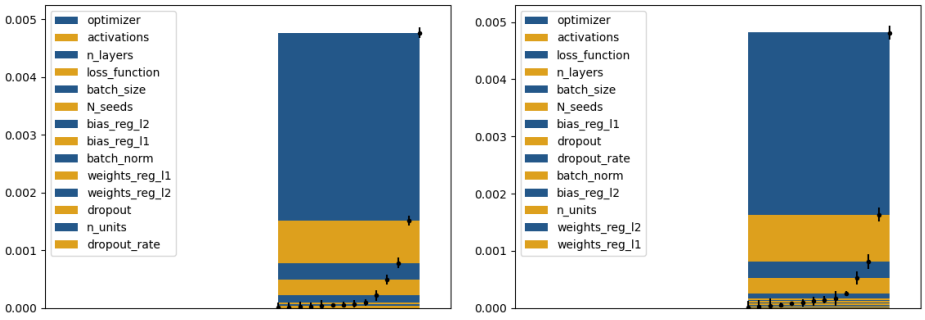


Figure 7: **Top (ii):** $S_{X,\mathbf{Y}}$ for X_1, X_2 and X_3 for different values of t. **Bottom (iii):** $S_{X,\mathbf{Y}}$ for $X_1|X_2 > t$, $X_2|X_2 > t$ and $X_3|X_2 > t$ for different values of t.

To sum up, this formulation brings important implementation advantages, because it allows to compute $S_{X_i,\mathbf{Y}}$ as if there were no conditionality, but carries a risk to miss important impacts of conditional hyperparameters, and discard them illegitimately.

(iii) In this work, we propose a splitting strategy that produces sets of hyperparameters that are involved together in the training, but are not disjoints, unlike **(i)**. Let $\mathcal{J}_k \in \{1, ..., n_h\}$ be the set of indices of hyperparameters that can be involved in a training jointly with conditional hyperparameter X_k. We define $\mathcal{G}_{X_k} = \{X_i|X_k, i \in \mathcal{J}_k\}$, the set of hyperparameters involved jointly in hyperparameters configurations when X_k is also involved. By convention, we denote the set of all main hyperparameters by $\mathcal{G}_0$. In Runge problem, `dropout_rate` is the only conditional hyperparameter, so we have two sets $\mathcal{G}_0 = \{X_1, ..., X_{n_h}\} \backslash$`dropout_rate` and $\mathcal{G}_{\texttt{dropout\_rate}} = \{X_1|\texttt{dropout\_rate}, ..., X_{n_h}|\texttt{dropout\_rate}\}$ $= \{X_1|\texttt{dropout} = \texttt{true}, ..., X_{n_h}|\texttt{dropout} = \texttt{true}\}$. It is then possible to compute $S_{X_i,\mathbf{Y}}$ for $X_i \in \mathcal{G}_0$, identify the most impactful main hyperparameters, then to compute $S_{X_i,\mathbf{Y}}$ for $X_i \in \mathcal{G}_{\texttt{dropout\_rate}}$ and to assess if `dropout_rate` is impactful by comparing it to other variables of $\mathcal{G}_{\texttt{dropout\_rate}}$. On the example problem, we can compute $S_{X_i,\mathbf{Y}}$ only for X_1, X_2 and X_3 when $X_2 > t$. This set would be $\mathcal{G}_{X_3}$ (except that X_2 is not categorical nor integer - but in that case we can consider $\bar{X}_2 = \mathbb{1}(X_2 > t)$). On the bottom row of Figure 7, $S_{X_1|X_2>t,\mathbf{Y}}$ and $S_{X_3|X_2>t,\mathbf{Y}}$ keep approximately the same values for all t, which is the correct conclusion since when X_3 is involved (i.e. $X_2 > t$) , it is as important as X_1 for reaching $\mathbf{Y}$. Coming back to Runge, Figure 8 displays $S_{X_i,\mathbf{Y}}$ for Runge approximation for

$X_i \in \mathcal{G}_{\texttt{dropout\_rate}}$, compared to the first approach where we do not care about conditionality, though in this specific case it does not change much of the conclusion that `dropout_rate` is not impactful.



(a) (b)

Figure 8: Comparison of $S_{X_i,\mathbf{Y}}$ without considering conditionality (a) and for variables $X_i \in \mathcal{G}_{\texttt{dropout\_rate}}$ (b)

In Runge example, we have only considered one conditional hyperparameter, `dropout_rate`, leading to only two groups $\mathcal{G}_0$ and $\mathcal{G}_{\texttt{dropout\_rate}}$. In another, more complex example, we could introduce additional conditional hyperparameters such as SGD's `momentum`. In that case, there would be two additional group. The group $\mathcal{G}_{\texttt{momentum}}$, that contains hyperparameters conditioned to when `momentum` is involved, but also $\mathcal{G}_{(\texttt{dropout\_rate},\texttt{momentum})}$ that contains hyperparameters conditioned to when `momentum` and `dropout_rate` are *simultaneously* involved. If the initial random search contains n_s configurations, `dropout_rate` and `momentum` are involved in $n_s/2$ configurations. HSIC estimation of hyperparameters of the groups $\mathcal{G}_{\texttt{dropout\_rate}}$ and $\mathcal{G}_{\texttt{momentum}}$ will be coarser but still acceptable. However, `dropout_rate` and `momentum` would only be involved simultaneously in $n_s/4$ configurations, which may lead to too inaccurate HSIC estimation for $\mathcal{G}_{(\texttt{dropout\_rate},\texttt{momentum})}$. This happens because `dropout_rate` and `momentum` do not depend on the same main hyperparameter. Hence to avoid this problem, we only consider groups $\mathcal{G}$ with conditional hyperparameters that depend on the same main hyperparameter. In our case, these groups are $\mathcal{G}_0$, $\mathcal{G}_{\texttt{dropout\_rate}}$ and $\mathcal{G}_{\texttt{momentum}}$.

5.4 Summary: evaluation of HSIC in HO

In this section, we summarize results of the previous discussions to provide a clear methodology for evaluating HSIC of hyperparameters in complex search spaces in Algorithm 1.

Comments on Algorithm 1. **Line 1:** one can chose any initial distribution for hyperparameters. **Line 2:** this step is a classical random search. We remind that HSIC evaluation can be applied after any random search, even if it was not initially conducted for HSIC estimation. Configurations $\boldsymbol{\sigma}_i$ are sampled from $\boldsymbol{\sigma} = (X_1, ..., X_{n_h}) \in \mathcal{H}$. **Line 3:** this step strongly benefits from parallelism. **Line 4:** the set $\mathbf{Y}$ is often taken as the p % percentile of $\{Y_1, ..., Y_{n_s}\}$, but can be any other set depending on what we want to assess. **Line 6 - 10:** the evaluation starts with main hyperparameters because they are always involved. Once most impactful main hyperparameters are selected, assess conditional ones.

Algorithm 1 Evaluation of HSIC in HO

Inputs: hyperparameters search space $\mathcal{H} = \mathcal{X}_1 \times ... \times \mathcal{X}_{n_h}$, n_s.
Sample n_s hyperparameters configurations $\{\boldsymbol{\sigma}_1, ..., \boldsymbol{\sigma}_{n_s}\}$.
Train a NN for each configuration and gather outputs $\{Y_1, ..., Y_{n_s}\}$.
Define $\mathbf{Y}$.
Construct groups $\mathcal{G}_0, ...$
for each group, starting with $\mathcal{G}_0$ **do**
    Construct U_i for every X_i using Φ_i of section 5.1.
    Compute $S_{X_i,\mathbf{Y}} := S_{U_i,\mathbf{Y}}$ using (8).
    By comparing them, select the most impactful hyperparameters.
    Check for interacting hyperparameters.
Outputs: Most impactful hyperparameters and interacting hyperparameters.

6. Applications of HSIC in HO

Now that we are able to compute and to correctly assess HSIC, we introduce possible usages of this metric in the context of HO. In this section we explore three ways of using HSIC, that are related to robust conception approach:

1 Knowledge gain: HSIC allows analysing hyperparameters, obtaining knowledge about their relative impact on error.

2 Stability: Some hyperparameters configurations can lead to dramatically high errors. A hyperparameters range reduction based on HSIC can prevent such situations.

2 Acceleration: We can choose values for less important hyperparameters that improve inference and training time.

3 Dimension reduction: We can use feature selection to construct simpler models or to focus optimization on most impactful hyperparameters.

We illustrate these points through hyperparameters studies for the training of a Fully Connected NN on MNIST and a Convolutional NN on Cifar10. We also introduce a less common benchmark which is the approximation by a Fully Connected NN of Bateman equations solution. Solving these equations is an important part of many high performance numerical simulations of several phenomenon (neutronic (Bernède and Poëtte, 2018; Dufek et al., 2013), combustion (Bisi and Desvillettes, 2006), detonic (Lucor et al., 2007), etc.). Approximating this solution with a light NN could accelerate these simulations whose execution time is often prohibitive. We believe that it is a good illustration of the need for robust conception of NNs. Details about the construction of Bateman equations data set can be found in **Appendix B** and hyperparameters space for each problem in **Appendix A**.

6.1 HSIC for Hyperparameters Analysis

This section presents a first analysis of HSIC estimation for the three benchmark datasets: MNIST, Cifar10 and Bateman equations. These evaluations are based on an initial Random Search for $n_s = 1000$ different hyperparameters configurations. The set $\mathbf{Y}$ is the 10%-best

errors percentile, so n_s is taken sufficiently large for $U_i|Z=1$ to be correctly estimated by HSIC. Indeed, if $n_s = 1000$, there will be 100 samples of $U_i|Z=1$. This Random search was conducted using 100 parallel jobs on CPU nodes for Fully Connected NNs and 24 parallel jobs on Nvidia Tesla v100 GPUs for CNNs, so the results for these configurations were obtained quite quickly, in less than two days.

6.1.1 MNIST

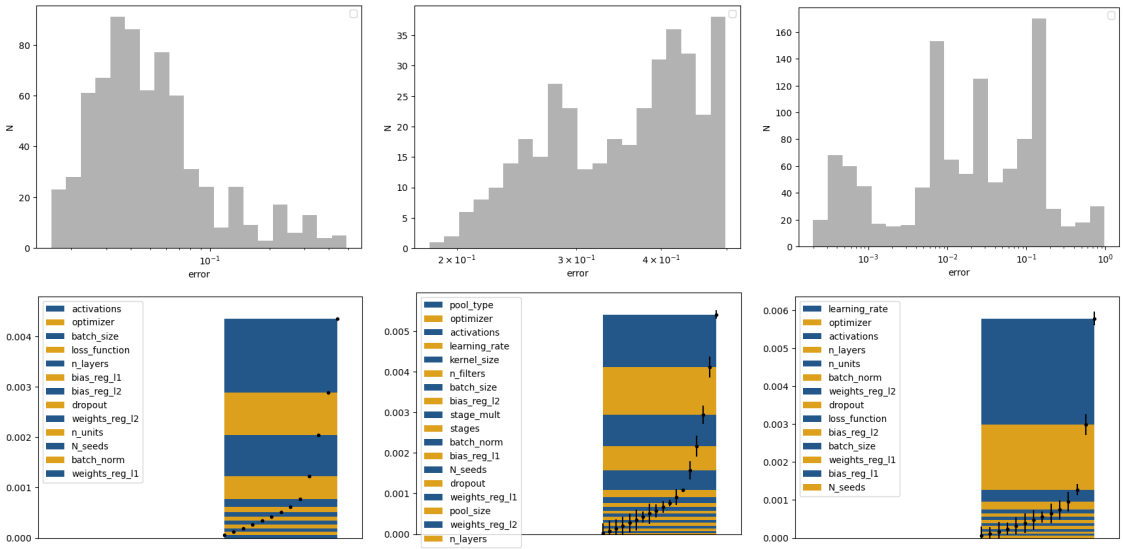
We train $n_s = 1000$ different NNs. The error is evaluated on a validation set constructed by extracting 5000 points from the training set. We keep the test set for final evaluations. We can see on Figure 9a that the accuracy goes up to $\sim 99\%(1 - \text{error})$ which is quite high for a fully connected NN on MNIST. Figure 9a also displays the values of $S_{X_i,\mathbf{Y}}$ for each hyperparameter X_i stacked vertically. Here, `activation`, `optimizer`, `batch_size` and `loss_function` have significantly high $S_{X_i,\mathbf{Y}}$. Hyperparameter `n_layers` also stands out from the remaining hyperparameter, while staying far below `loss_function` HSIC. There is one conditional group to consider, $\mathcal{G}_{\texttt{dropout\_rate}}$, and `dropout_rate` is found not to be impactful.

Interestingly, neither the depth (`n_layers`) nor the width (`n_units`) are among the most important hyperparameters. Notice that in the Random Search, we obtained a NN of depth 4 and width 340 which obtained 98.70% accuracy, while the best networks (there were two) obtained 98.82% accuracy for a depth of 10 and a width of 791 and 1403, respectively. Recall that the min-max depth were 1-10 and width were 134-1500. it means that lighter networks are capable of obtaining very good accuracy. Another interesting observation is that `loss_function` does not have the highest HSIC, meaning that Mean Squared Error allows to obtain good test errors which is counter-intuitive for a classification problem.

We plot histograms of U_i and $U_i|Z=1$ on Figure 10a for `activation` (top) and `weights_reg_l1` (bottom) with repeated sampling for categorical hyperparameters, like in Section 5.1. Note that the first and the second hyperparameters have respectively a high and low $S_{X_i,\mathbf{Y}}$. We can see that for hyperparameters with high $S_{X_i,\mathbf{Y}}$, $U_i|Z=1$ (orange for KDE, blue for histogram) is quite different from U_i (red for KDE, grey for histogram). On the contrary, for hyperparameters with low $S_{X_i,\mathbf{Y}}$ there not seems to have major differences.

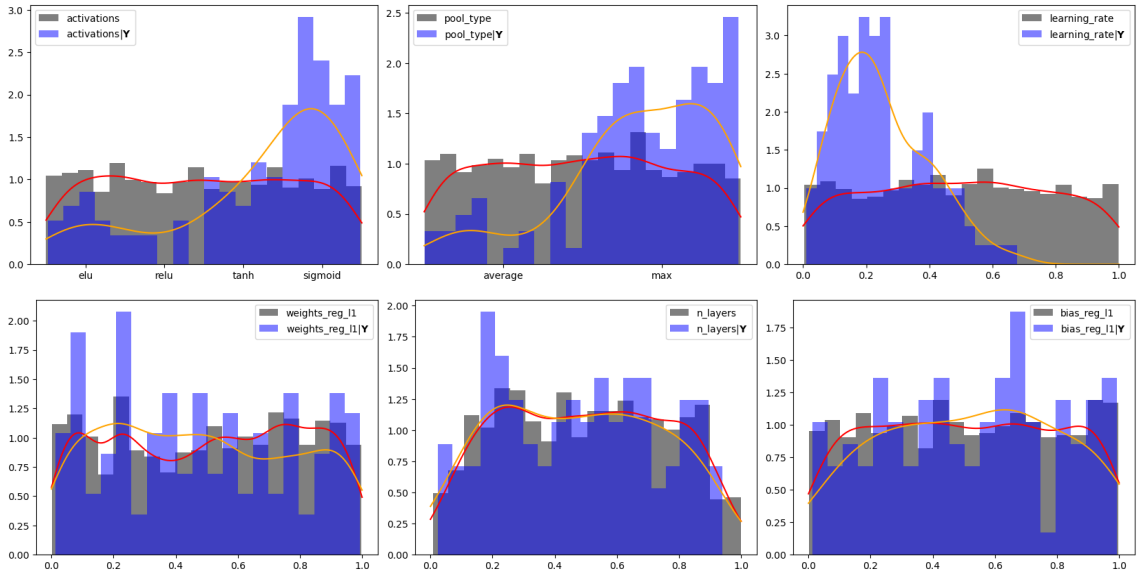
6.1.2 Cifar10

We train $n_s = 1000$ different CNNs. The error is evaluated on a validation set constructed by extracting 5000 points from the training set. We keep the test set for final evaluations in Section 6.4. After the initial Random Search, the best validation error is 81.37% and the corresponding test error is 81.80%. Note that the histogram of Figure 9b is truncated because many hyperparameters configurations led to diverging errors. Here, `pool_type`, `optimizer`, `activation`, `learning_rate` and `kernel_size` have the highest $S_{X_i,\mathbf{Y}}$, followed by `n_filters`. Half of these hyperparameters are specific to CNNs, which validates the impact of these layers on classification tasks for image data. The conditional groups are listed in **Appendix A**. We do not show $S_{X_i,\mathbf{Y}}$ comparisons for every groups for clarity of the article and simply report that one conditional hyperparameter `centered`, which triggers centered RMSPROP if this value is chosen for `optimizer`, is also found to be impactful.



(a) MNIST (b) Cifar10 (c) Bateman

Figure 9: (top) Histograms of the initial random sampling of configurations and (bottom) comparison of $S_{X_i,\mathbf{Y}}$ for every main hyperparameters.



(a) MNIST (b) Cifar10 (c) Bateman

Figure 10: Representation of $U_i|Z=1$ (orange for KDE and blue for histogram) and U_i (red for KDE and grey for histogram), for hyperparameters X_i with high (top) and low (bottom) $S_{X_i,\mathbf{Y}}$

The depth (`n_layers`) is the less important hyperparameters. Notice that in the Random Search, we obtained a NN of depth 4 and width 53, with 3 stages (meaning that the NN is widen 3 times), which obtained 80.70% validation accuracy, while the best networks obtained 81.37% accuracy for a depth of 6 and 48 but 4 stages. The conclusion is the same than for MNIST: increasing the size of the network is not the only way to improve its accuracy.

We plot histograms of U_i and $U_i|Z=1$ on Figure 10b for `pool_type` (top) and `n_layers` (bottom) like in the previous section. The histograms of `n_layers` are interesting because even the histogram of U_i does not seem uniform. It may be explained by configurations leading to out of memory errors or so long to train that 1000 other NNs with different configurations had already been trained meanwhile. It also explains why its HSIC is so low. Still, the conclusions that `n_layers` has a limited impact is valid since there is no major differences between U_i and $U_i|Z=1$.

6.1.3 Bateman equations

For Bateman equations, we use a training set of 100000 points and a validation set of 10000 points. Like for Cifar10, we keep a test set of 20000 points for final evaluations. The error is evaluated on the validation set. Mean squared error goes down to 2.90×10^{-5}. Like for Cifar10, the histogram of Figure 9b is truncated because many hyperparameters configurations led to diverging errors. For this problem, `learning_rate`, `optimizer`, `activations` and `n_layer` can be considered as impactful. Conditional groups are also listed in **Appendix A**. Three conditional hyperparameters are important: `beta_1`, the first moment decay coefficient of ADAM and NADAM, `nesterov`, that triggers Nesterov's momentum in SGD and `centered`, described previously.

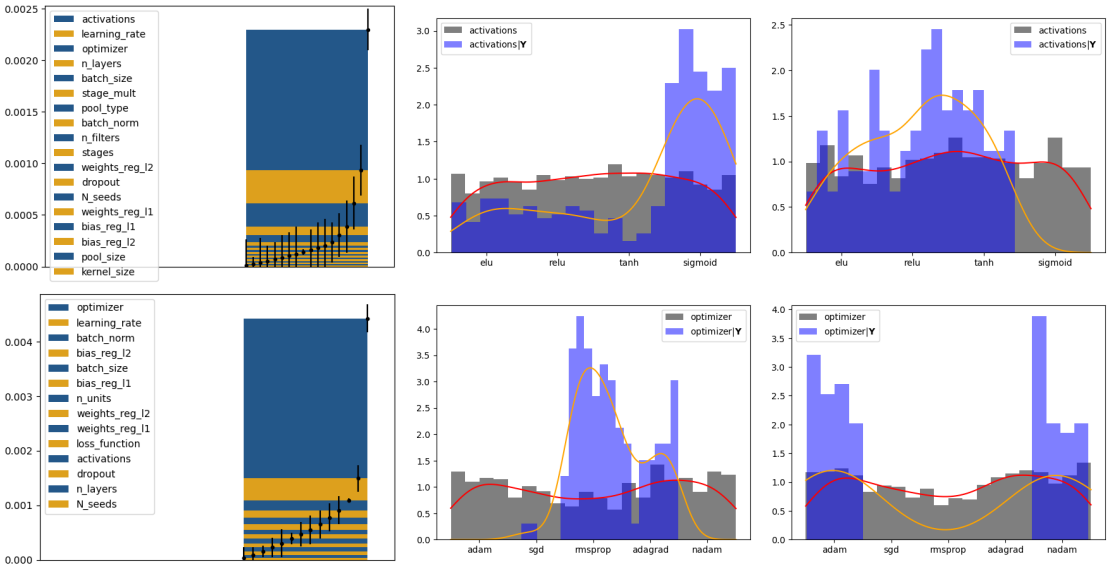
HSIC for `n_layers` is still the lowest of the significant $S_{X_i,\mathbf{Y}}$ and `n_units` belongs to less impactful hyperparameters. We perform the same analysis than for MNIST and Cifar10 and quote that the best NN has depth 5 and width 470 while another NN of depth 5 and width 62 reaches 3.74×10^{-5} validation error.

We plot histograms of U_i and $U_i|Z=1$ on Figure 10c for `learning_rate` (top) and `bias_reg_l1` (bottom). Histograms of `learning_rate` is interesting because this hyperparameter is continuous so the distribution $U_i|Z=1$ seems more natural. This once again illustrates the differences of U_i and $U_i|Z=1$ for hyperparameters with high and low $S_{X_i,\mathbf{Y}}$.

6.2 Modification of hyperparameters distribution to improve stability

Up to now, we only considered $\mathbf{Y}$ to be the 10% best error percentile, which is natural since we want to understand the impact of hyperparameters towards good errors. But HSIC formalism and our adaptation to HO allows to chose any $\mathbf{Y}$. We saw on previous Section that for Cifar10 and Bateman, histograms of Figure 9b are truncated because many hyperparameters configurations led to diverging errors. It is possible to understand why by choosing $\mathbf{Y}$ as the set of 10% worst errors. Then, HSIC can be applied to assess the importance of each hyperparameters towards the worst errors.

Figure 11b shows $S_{X_i,\mathbf{Y}}$ comparisons, for Cifar10 and Bateman, when $\mathbf{Y}$ is the set of the 10% worst errors. In that case, $S_{X_i,\mathbf{Y}}$ measures how detrimental bad values of X_i can be for the NN error. For Cifar10, `activation` is shown to be the main responsible of highest errors. If we plot the histogram of `activation`$|\mathbf{Y}$, we can clearly see that `sigmoid` is a bad value in the sense that most of the worst NN use this activation function. If we come back to $\mathbf{Y}$ being the set of 10% best NNs, we see that none of the best NNs have `sigmoid` as activation function. By itself, this kind of knowledge is valuable because it improves our understanding of hyperparameters impact. This also directly brings some practical benefits: in that case we could reasonably discard `sigmoid` from the hyperparameter space and therefore adapt



(a) $S_{X_i,\mathbf{Y}}$, $\mathbf{Y}$ = 10% worst errors (b) $X_i|\mathbf{Y}$, $\mathbf{Y}$ = 10% worst errors (c) $X_i|\mathbf{Y}$, $\mathbf{Y}$ = 10% best errors

Figure 11: **Top**: Cifar10. **Bottom:** Bateman. **(a)** Comparison of $S_{X_i,\mathbf{Y}}$ when $\mathbf{Y}$ is the set of the 10% worst errors. **(b)** Histogram of $X_i|\mathbf{Y}$ when $\mathbf{Y}$ is the set of 10% worst errors, with $X_i =$ `activations` for Cifar10 and $X_i =$ `optimizer` for Bateman. **(c)** Histogram of $X_i|\mathbf{Y}$ when $\mathbf{Y}$ is the set of 10% best errors, with $X_i =$ `activations` for Cifar10 and $X_i =$ `optimizer` for Bateman.

the distribution of `activation` to improve stability. The same reasoning can be applied to Bateman, with $X_i =$ `optimizer`, for `adagrad` and `rmsprop` optimizers.

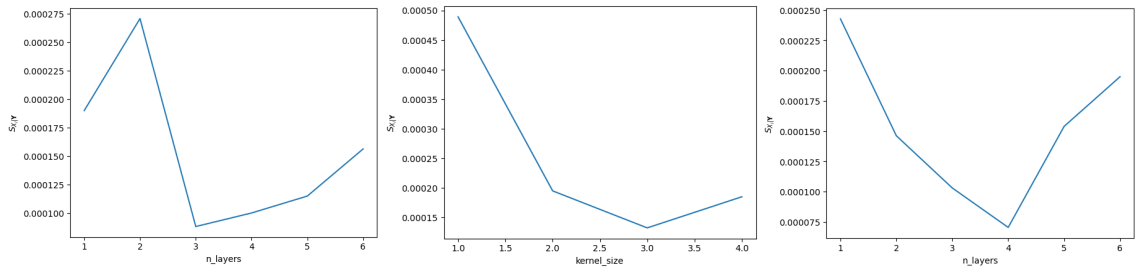
Note that we could have drawn the previous conclusions by directly looking at histograms as represented in Figure 11b and 11c. However, when the number of hyperparameters grows, the number of histograms to look at and to visually evaluate grows as well and the analysis can become tedious. Thanks to HSIC, we know directly which histograms to look, and how to rank hyperparameters when it is not visually clear cut.

6.3 Interval reduction for hyperparameters that affect execution speed

One common conclusion of $S_{X_i,\mathbf{Y}}$ values for the last three ML problems is that one does not have to set high values for hyperparameters that affect execution speed, such as `n_units`, `n_layers`, or `n_filters`, in order to obtain competitive models. This naturally raises the question of how to bias the hyperparameters optimization towards such models. Multi objective HO algorithms have already been successfully applied, like in Tan et al. (2018) for instance, but these algorithms are black-boxes and involve tuning additional hyperparameters for the multi objective loss function.

In our case, we can use information from $S_{X_i,\mathbf{Y}}$ to reduce the hyperparameters space search in order to obtain more cost effective NNs. The most simple way to achieve that goal is to select values that improve execution speed for hyperparameters that have low $S_{X_i,\mathbf{Y}}$ values. For MNIST, it would mean for instance to choose `n_units` = 128, for Cifar10, `n_layers` = 3 or for Bateman, `n_units` = 32.

However, if all hyperparameters that affect execution speed are important, i.e. they have high $S_{X_i,\mathbf{Y}}$ value, we may not be able to apply the previous idea. In that case, we can use HSIC in another way to still achieve our goal. Most of the time, hyperparameters value can be easily linked to good or bad execution time. A larger `n_units`, `n_layers`, or `kernel_size` will always hurt execution time. Suppose that $X_i = \texttt{n\_units} \in [a, b]$ and that $S_{X_i,\mathbf{Y}}$ is high, so that n_{units} is among the most important hyperparameters. Very often, a too low value for `n_units` can significantly hurt the accuracy. It is likely that $S_{X_i,\mathbf{Y}}$ is high because a is too small. One could therefore compute $S_{X_i|X_i\in[a+c,b],\mathbf{Y}}$ for $c \in [1, b-a]$, starting with $c = 1$ until $S_{X_i|X_i\in[a+c,b],\mathbf{Y}}$ becomes low. Then, hyperparameter `n_units` can be replaced by $\texttt{n\_units}|\texttt{n\_units} \in [a + c, b]$, which has a low HSIC, and whose value can be set close to $a + c$ like in the previous paragraph. This methodology is illustrated on Figure 12, where $S_{X_i|X_i\in[a+c,b],\mathbf{Y}}$ is plotted with respect to c.



(a) MNIST (b) Cifar10 (c) Bateman

Figure 12: $S_{X_i|X_i\in[a+c,b],\mathbf{Y}}$ w.r.t. c for (a) `n_layers` in MNIST, (b) `kernel_size` in Cifar10 and (c) `n_layers` in Bateman.

This plot highlights that in each case, $S_{X_i,\mathbf{Y}}$ decreases until it reaches a value that would make the corresponding hyperparameter belong to the less important hyperparameters. It suggests that we could set `n_layers` $= 3$ for MNIST, `kernel_size` $= 3$ for Cifar10 and `n_layers` $= 4$ for Bateman without affecting the error too much. Once hyperparameters space has been reduced to improve NNs execution time, it is possible to apply any classical HO algorithm.

One of the most successful and widely used HO algorithm is Gaussian Processes based Bayesian Optimization (GPBO). However, this algorithm is known to struggle in too high dimensions. In the case of Cifar10, choosing values for hyperparameters that affect execution time would still lead to a space of dimension 20, which is quite large to apply GPBO. Another application of HSIC in HO, described in the next section could help overcoming this problem.

6.4 Two step HO: focusing on impactful hyperparameters

In Song et al. (2007), the authors introduce the use of HSIC for feature selection and in Spagnol et al. (2018), HSIC based feature selection is used in the context of optimization. The idea is to compute $S_{X_i,\mathbf{Y}}$ for each variable involved in the optimization, and to discard low $S_{X_i,\mathbf{Y}}$ variables from it. More specifically, the discarded variables are fixed to an arbitrary value, and then the optimization algorithm is applied only in the dimension of the high $S_{X_i,\mathbf{Y}}$ variables.

This methodology is particularly suited for HO. In this work, we have emphasized the ability of HSIC to identify most important hyperparameters. This allows performing relevant HSIC driven hyperparameters selection, which can overcome the problem of optimization in too high dimensional hyperparameters space. We go farther and present a two step optimization. We do not stop to the optimization of most relevant hyperparameters, but also fine tune less important hyperparameters in a second optimization step. As a result, the problematic optimization in high dimension is split into two easier optimization steps:

1 Optimization in the reduced, yet impactful hyperparameters space, which has reasonable dimension. It allows applying GPBO despite the initially large dimension of the hyperparameters space. At the end of this step, optimal values for most impactful hyperparameters are selected.

2 Optimization on the remaining dimensions. In our case, GPBO can be reasonably applied in this space, but note that we might have hyperparameters spaces whose initial dimension is so high that after the first step, the remaining dimensions to optimize could still be too numerous for GPBO to be performed. In that case, other, less refined but more robust HO algorithms (like RS or Tree Parzen Estimators (Bergstra et al., 2011)) could be applied, which would not be so much of a problem since remaining hyperparameters are less impactful.

For the first step, values have to be chosen for less impactful hyperparameters that are not involved in the optimization. In Spagnol et al. (2018), the authors choose the values yielding the best output after the initial RS. Here, the value selection method that aims at improving execution speed, introduced in Section 6.3, integrates perfectly with this two step optimization. Following this method brings two advantages:

- We can obtain more cost effective NNs, if we keep these values through the two optimization step.

- If we do not care so much about execution speed but only look for accuracy, still fixing these values during the first optimization step allows to improve the training speed and so global HO time.

The rest of the low $S_{X_i,\mathbf{Y}}$ hyperparameters value can be set as those of the hyperparameters configuration yielding the best error. There is one last attention point: one have to be careful about interactions between low $S_{X_i,\mathbf{Y}}$ hyperparameters. If two low HSIC hyperparameters X_i and X_j are found to interact, like discussed in section 5.2, and X_i has an impact on execution speed, the value of X_j must be chosen so that value of the pair (X_i, X_j) is close to a value yielding a good NN error. The two step optimization is summarized in Algorithm 2.

We evaluate this two step optimization on our three data sets. For each of these, we consider 4 baselines:

- RS: The result of the Random Search of 1000 configurations plus 200 additional configurations for a total of $n_s = 1200$ points.

Algorithm 2 Two step Optimization

Inputs: hyperparameters search space $\mathcal{H} = \mathcal{X}_1 \times ... \times \mathcal{X}_{n_h}$, n_s
Apply Algorithm 1: "Evaluation of HSIC in HO".
Perform interval reduction
Select values for less impactful hyperparameters that improve execution speed, taking care of interaction.
// `Step 1:`
Apply GPBO to the most impactful hyperparameters.
// `Step 2:`
if goal = accuracy and execution speed **then**
    Keep the optimal values of step 1 and the values of less impactful hyperparameters that improve execution speed. Apply GPBO to the remaining dimensions.
else if goal = accuracy only **then**
    Keep the optimal values of step 1. Apply GPBO to the remaining dimensions.

- full GPBO: Gaussian Processes based Bayesian Optimization, conducted on the full hyperparameters space, without any analyses based on HSIC. We initialize the optimization with 50 random configurations and perform the optimization for 50 iterations (enough to reach convergence).

- TS-GPBO (accuracy): Two-Step GPBO described in Algorithm 2, with goal = accuracy. Step 1 and 2 are ran for 25 iterations.

- TS-GPBO (accuracy + speed): Two-Step GPBO described in Algorithm 2, with goal = accuracy and execution speed.

For each of these baselines, the *test* error is reported (the metric is accuracy for MNIST and Cifar10 and MSE for Bateman), as well as the number of parameters of the best models and their FLOPs. RS (ran using 100 parallel jobs for MNIST and Bateman and 24 for Cifar10) took between 2 and 3 days depending on the data set, full GPBO between 3 and 4 days and TS-GPBO between 3 and 4 days as well ($2-3$ days for the initial RS and 1 day for the two steps of GPBO). Time measure is coarse because not all the training has been conducted on the same architectures (Sandy Bridge CPUs, Nvidia Tesla V100 and Nvidia Tesla P100 GPUs), even within a same baseline, for cluster accessibility reasons. Nonetheless, these approximate time measures demonstrate that full GPBO and TS-GPBO take approximately the same amount of time. This is explained by step 1 of TS-GPBO which always choose values for non optimized hyperparameters that improve execution speed and training time. As a result, step 1 is quite fast. In addition, experiments show that step 2 usually converges faster, in terms of number of evaluations, than full GPBO to the reported minimum, perhaps because the optimal values found during step 1 make step 2 begin close to an optimum. The results of 5 repetitions (except for RS) of each baseline can be found in Table 4.

Results show that except for Cifar10, TS+GPBO yields very competitive NNs while having far less parameters and FLOPs. For MNIST, TS+GPBO model has ≈ 66 and 3 times less parameters and FLOPs than full GPBO and RS. For Bateman these factors are 482 and 380. This huge factor may be explained by an oversized initial hyperparameter search

data set	baseline	test metric	params	MFLOPs
MNIST	RS	98.36	436,147	871
-	full GPBO	**98.42** $\pm$ 0.05	10,271,367	20,534
-	TS-GPBO (accuracy + speed)	**98.42** $\pm$ 0.02	**151,306**	**307**
Cifar10	RS	81.8	99,444,880	1,832,615
-	full GPBO	**82.73** $\pm$ 1.45	71,111,761	1,441,230
-	TS-GPBO (accuracy)	**82.60** $\pm$ 0.58	9,604,539	650,269
-	TS-GPBO (accuracy + speed)	79.34 $\pm$ 0.15	**9,281,258**	**178,621**
Bateman	RS	**1.99** $\times 10^{-4}$	1,259,140	2,516
-	full GPBO	2.94 $\pm$ 0.42 $\times 10^{-4}$	1,588,215	3,173
-	TS-GPBO (accuracy + speed)	3.49 $\pm$ 0.31 $\times 10^{-4}$	**3,291**	**7**

Table 4: Results of HO for Random Search (RS), Gaussian Processes based Bayesian Optimization on full hyperparameters space (full GPBO) and Two-Steps Gaussian Processes based Bayesian Optimization (TS-GPBO). The mean $\pm$ standard deviation across 5 repetitions is displayed for the test metric. For the number of parameters and FLOPs, the maximum value obtained across repetitions is reported because it illustrates the worst scenario that can happen for execution speed, and how much our method prevents it.

space. Still, a reasonable size for the search space cannot be found *a priori* and our method makes HO robust to such bad *a priori* choices. Note that for these cases, we only reported results of TS-GPBO (accuracy + speed), because the results of this baseline was already very good and TS-GPBO (accuracy) did not bring significant improvement. For the special case of Cifar10, TS-GPBO (accuracy) and (accuracy + speed) both find a model which has 11 and 9 times less parameters than RS and full GPBO. TS-GPBO (accuracy) finds a model which has ≈ 3 and 2 less FLOPs than RS and full GPBO while these factors are 10 and 8 for (accuracy + speed). Full GPBO and TS-GPBO (accuracy) achieve comparable accuracy but the standard deviation for full GPBO is 2.5 times higher than for TS-GPBO (accuracy), which demonstrates the robustness of TS-GPBO (accuracy). Even if execution time is not an explicitly desired output of TS-GPBO (accuracy), the first step of TS-GPBO, which selects values that improve execution time, seems to bias the optimization towards more cost effective models, as the final number of parameters and FLOPs shows. All these good results have been allowed thanks to information given by HSIC analysis. Hence, not only TS-GPBO outputs good and cost effective models, but it also outputs a better knowledge of hyperparameters interaction in these ML problems, as opposed to RS and full GPBO which are black-boxes.

7. Discussion

Hyperparameters modelling choice. HSIC is a powerful tool which is widely used for SA or as dependence measure. Its application to HO required some work, especially regarding the complex structure of hyperparameters space. To achieve this goal, some modelling choices have been made, such as the application of Φ_{X_i} to map hyperparameter X_i to a uniform random variable. The good results obtained in Section 6.4 not only validate the usage of information given by HSIC for HO, but also this modelling choice.

Automating Two Step Gaussian Process based Bayesian Optimization. In this work, we presented methodologies for exploiting HSIC information that involved human intervention. Indeed, someone have to actively decide which hyperparameter to consider as more or less important. Nevertheless, one advantage of HSIC is that it is a scalar metric. One could construct an HSIC based HO by setting a threshold above which hyperparameters are considered as important, leading to an end-to-end automatic, yet explainable HO algorithm. Though the idea of threshold is used in Spagnol et al. (2018), and variable selection is explored in Song et al. (2007), their application to HO have not been studied in this paper and could be part of future works.

Other dependence measures. In this work, we used HSIC as a dependence measure. Our derivations for its application to HO still hold for any other dependence measure sharing the same properties than HSIC, though studies of different dependence measures is beyond the scope of this paper.

Global HO speed up. We presented some ways of using HSIC in HO, but this paper mostly emphasized the possibility to exploit it in order to find lighter models. We are aware that execution speed is not always a goal for ML practitioners. Still, ML practitioner are always concerned about training speed. TS-GPBO (accuracy) demonstrated the possibility to use HSIC to improve training speed without hurting the final accuracy so even if final execution speed is not a goal, TS-GPBO made it interesting to use HSIC for that purpose. To go farther, it would even be possible to apply parallel GPBO like described in Snoek et al. (2012), or to use Hyperband on the initial RS, since HSIC computation only relies on the error of the p-% best NNs.

Further execution time improvement. One advantage of execution time improvement obtained thanks to HSIC is that it relies on the conception of the NN. Therefore, further improvement could be made by applying other techniques like quantization or weights pruning.

8. Conclusion

Hyperparameters optimization is a very important step of machine learning applications and ordinarily only returns one optimal hyperparameters configuration, in a black-box fashion. Using an approach based on sensitivity analysis, we show that we can make Hyperparameters optimization explainable. In particular, we adapt Hilbert Schmidt Independence Criterion, a statistical dependence tool used in sensitivity analysis, to hyperparameters spaces that can be complex and awkward due to the different nature of hyperparameters (continuous or categorical) and their interactions and inter-dependencies. Their use in hyperparameters optimization allows constructing new optimization methodologies, based on two step bayesian optimization where each step is applied on a relevant subset of hyperparameters and other hyperparameters values are chosen in principled ways. These methodologies allows finding optimal values for hyperparameters that improve execution speed as well as test error.

References

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

Adrien Bernède and Gaël Poëtte. An unsplit monte-carlo solver for the resolution of the linear boltzmann equation coupled to (stiff) bateman equations. *Journal of Computational Physics*, 354:211–241, 02 2018.

M. Bisi and L. Desvillettes. From reactive boltzmann equations to reaction–diffusion systems. *Journal of Statistical Physics*, 124(2):881–912, Aug 2006.

E. Borgonovo. A new uncertainty importance measure. *Reliability Engineering & System Safety*, 92(6):771 – 784, 2007.

François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

I. Csizar. Information-type measures of difference of probability distributions and indirect observation. *Studia Scientiarum Mathematicarum Hungarica*, 2:229–318, 1967.

Sébastien Da Veiga. Global sensitivity analysis with dependence measures. *Journal of Statistical Computation and Simulation*, 85, 11 2013.

Jan Dufek, Dan Kotlyar, and Eugene Shwageraus. The stochastic implicit euler method – a stable coupling scheme for monte carlo burnup calculations. *Annals of Nuclear Energy*, 60:295 – 300, 10 2013.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Jean-Claude Fort, Thierry Klein, and Nabil Rachdi. New sensitivity analysis subordinated to a contrast. *Communications in Statistics - Theory and Methods*, 45(15):4349–4364, 2016.

Kenji Fukumizu, Arthur Gretton, Gert R. Lanckriet, Bernhard Schölkopf, and Bharath K. Sriperumbudur. Kernel choice and classifiability for rkhs embeddings of probability distributions. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1750–1758. Curran Associates, Inc., 2009.

Daniel T Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403 – 434, 1976.

Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *Proceedings of the 16th International Conference on Algorithmic Learning Theory*, ALT'05, page 63–77, Berlin, Heidelberg, 2005. Springer-Verlag.

Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J. Smola. A kernel method for the two-sample-problem. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 513–520. MIT Press, 2007.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 240–248, Cadiz, Spain, 09–11 May 2016. PMLR.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 2020–2029, Red Hook, NY, USA, 2018. Curran Associates Inc.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

D. Lucor, C. Enaux, H. Jourdren, and P. Sagaut. Stochastic design optimization: Application to reacting flows. *Computer Methods in Applied Mechanics and Engineering*, 196(49):5047 – 5062, 2007.

Alfred Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, 29(2):429–443, 1997.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Saman Razavi, Anthony Jakeman, Andrea Saltelli, Clémentine Prieur, Bertrand Iooss, Emanuele Borgonovo, Elmar Plischke, Samuele Lo Piano, Takuya Iwanaga, William Becker, Stefano Tarantola, Joseph H.A. Guillaume, John Jakeman, Hoshin Gupta, Nicola Melillo, Giovanni Rabitti, Vincent Chabridon, Qingyun Duan, Xifu Sun, Stefán Smith, Razi Sheikholeslami, Nasim Hosseini, Masoud Asadzadeh, Arnald Puy, Sergei Kucherenko, and Holger R. Maier. The future of sensitivity analysis: An essential discipline for systems modeling and policy support. *Environmental Modelling & Software*, 137:104954, 2021.

Andrea Saltelli. Making best use of model evaluations to compute sensitivity indices. *Computer Physics Communications*, 145(2):280 – 297, 2002.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104:148–175, 2016.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, page 2951–2959, Red Hook, NY, USA, 2012. Curran Associates Inc.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2171–2180, Lille, France, 07–09 Jul 2015. PMLR.

Ilya M. Sobol. Sensitivity estimates for nonlinear mathematical models. *MMCE*, (1):407–414, 1993.

Le Song, Alex Smola, Arthur Gretton, Karsten M. Borgwardt, and Justin Bedo. Supervised feature selection via dependence estimation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 823–830, New York, NY, USA, 2007. Association for Computing Machinery.

Adrien Spagnol, Rodolphe Le Riche, and Sébastien Da Veiga. Global sensitivity analysis for optimization with variable selection. *SIAM/ASA J. Uncertain. Quantification*, 7:417–443, 2018.

Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002.

Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018.

Appendix A: Hyperparameters spaces

In this section, we describe hyperparameters spaces used for each problem in this paper. Note that hyperparameter `n_seeds` denotes the number of random repetitions of the training for each hyperparameters configuration. If a conditional hyperparameter X_j is only involved for some specific values of a main hyperparameter X_i, it is displayed with an indent on tab lines below that of X_i, with the value of X_i required for X_j to be involved in the training.

Runge & MNIST

For Runge & MNIST, only Fully Connected Neural Networks are trained, and the width (`n_units`) is the same for every layer.

hyperparameter	type	values for Runge	values for MNIST
`n_layers`	integer	$\in \{1, ..., 10\}$	same
`n_units`	integer	$\in \{7, ..., 512\}$	$\in \{128, ..., 1500\}$
`activation`	categorical	`elu`, `relu`, `tanh` or `sigmoid`	same
`dropout`	boolean	`true` or `false`	same
`yes:dropout_rate`	continuous	$\in [0, 1]$	same
`batch_norm`	boolean	`true` or `false`	same
`weights_reg_l1`	continuous	$\in [1 \times 10^{-6}, 0.1]$	same
`weights_reg_l2`	continuous	$\in [1 \times 10^{-6}, 0.1]$	same
`bias_reg_l1`	continuous	$\in [1 \times 10^{-6}, 0.1]$	same
`bias_reg_l2`	continuous	$\in [1 \times 10^{-6}, 0.1]$	same
`batch_size`	integer	$\in \{1, ..., 11\}$	$\in \{1, ..., 256\}$
`loss_function`	categorical	L_2 error or L_1 error	L_2 error or crossentropy
`optimizer`	categorical	`adam`, `sgd`, `rmsprop` or `adagrad`	same
`n_seeds`	integer	$\in \{1, ..., 40\}$	$\in \{1, ..., 10\}$

Table 5: Hyperparameters values for Runge & MNIST

Conditional groups: (see **(iii)** of Section 5.3) $\mathcal{G}_0$ and $\mathcal{G}_{\texttt{dropout\_rate}}$

Bateman

For Bateman, only Fully Connected Neural Networks are trained, and the width (`n_units`) is the same for every layer.

hyperparameter	type	values for Bateman
`n_layers`	integer	$\in \{1, ..., 10\}$
`n_units`	integer	$\in \{7, ..., 512\}$
`activation`	categorical	`elu`, `relu`, `tanh` or `sigmoid`
`dropout`	boolean	`true` or `false`
`yes:dropout_rate`	continuous	$\in [0, 1]$
`batch_norm`	boolean	`true` or `false`
`learning_rate`	continuous	$\in [1 \times 10^{-6}, 1 \times 10^{-2}]$
`weights_reg_l1`	continuous	$\in [1 \times 10^{-6}, 0.1]$
`weights_reg_l2`	continuous	$\in [1 \times 10^{-6}, 0.1]$
`bias_reg_l1`	continuous	$\in [1 \times 10^{-6}, 0.1]$
`bias_reg_l2`	continuous	$\in [1 \times 10^{-6}, 0.1]$
`batch_size`	integer	$\in \{1, ..., 500\}$
`loss_function`	categorical	L_2 error or L_1 error
`optimizer`	categorical	`adam`, `sgd`, `rmsprop`, `adagrad` or `nadam`
`adam:amsgrad`	boolean	`true` or `false`
`adam, nadam:1st_moment_decay`	continuous	$\in [0.8, 1]$
`adam, nadam:2nd_moment_decay`	continuous	$\in [0.8, 1]$
`rmsprop:centered`	boolean	`true` or `false`
`sgd:nesterov`	boolean	`true` or `false`
`sgd, rmsprop:momentum`	continuous	$\in [0.5, 0.99]$
`n_seeds`	integer	$\in \{1, ..., 10\}$

Table 6: Hyperparameters values for Bateman

Conditional groups: (see **(iii)** of Section 5.3) $\mathcal{G}_0$, $\mathcal{G}_{\texttt{dropout\_rate}}$, $\mathcal{G}_{\texttt{amsgrad}}$, $\mathcal{G}_{\texttt{centered}}$, $\mathcal{G}_{\texttt{nesterov}}$, $\mathcal{G}_{\texttt{momentum}}$ and
$\mathcal{G}_{(\texttt{1st\_moment},\texttt{2nd\_moment})}$

Cifar10

For Cifar10, we use Convolutional Neural Networks, whose width increases with the depth according to hyperparameters `stages` and `stage_mult`. The first layer has width `n_filters`, and then, `stages` -1 times, the network is widen by a factor `stage_mult`. For instance, a NN with `n_filters` $= 20$, `n_layers` $= 3$, `stages` $= 3$ and `stage_mult` $= 2$ will have a first layer with 20 filters, a second layer with `n_filters` $\times$ *stage_mult* $= 40$ filters, and a third layer with `n_filters` $\times$ `stage_mult`$^{\texttt{stages}-1} = 60$ filters.

hyperparameter	type	values for Bateman
`n_layers`	integer	$\in \{3, ..., 12\}$
`n_filters`	integer	$\in \{16, ..., 100\}$
`stages`	integer	$\in \{1, 4\}$
`stage_mult`	continuous	$\in [1, 3]$
`kernel_size`	integer	$\in \{1, 5\}$
`pool_size`	integer	$\in \{2, 5\}$
`pool_type`	categorical	`max` or `average`
`activation`	categorical	`elu`, `relu`, `tanh` or `sigmoid`
`dropout`	boolean	`true` or `false`
`yes:dropout_rate`	continuous	$\in [0, 1]$
`batch_norm`	boolean	`true` or `false`
`learning_rate`	continuous	$\in [1 \times 10^{-6}, 1 \times 10^{-2}]$
`weights_reg_l1`	continuous	$\in [1 \times 10^{-6}, 0.1]$
`weights_reg_l2`	continuous	$\in [1 \times 10^{-6}, 0.1]$
`bias_reg_l1`	continuous	$\in [1 \times 10^{-6}, 0.1]$
`bias_reg_l2`	continuous	$\in [1 \times 10^{-6}, 0.1]$
`batch_size`	integer	$\in \{10, ..., 128\}$
`loss_function`	categorical	L_2 error or crossentropy
`optimizer`	categorical	`adam`, `sgd`, `rmsprop`, `adagrad` or `nadam`
`adam:amsgrad`	boolean	`true` or `false`
`adam, nadam:1st_moment_decay`	continuous	$\in [0.8, 1]$
`adam, nadam:2nd_moment_decay`	continuous	$\in [0.8, 1]$
`rmsprop:centered`	boolean	`true` or `false`
`sgd:nesterov`	boolean	`true` or `false`
`sgd, rmsprop:momentum`	continuous	$\in [0.5, 0.99]$
`n_seeds`	integer	$\in \{1, ..., 10\}$

Table 7: Hyperparameters values for Bateman

Conditional groups: (see **(iii)** of Section 5.3) $\mathcal{G}_0$, $\mathcal{G}_{\texttt{dropout\_rate}}$, $\mathcal{G}_{\texttt{amsgrad}}$, $\mathcal{G}_{\texttt{centered}}$, $\mathcal{G}_{\texttt{nesterov}}$, $\mathcal{G}_{\texttt{momentum}}$ and
$\mathcal{G}_{(\texttt{1st\_moment},\texttt{2nd\_moment})}$

Appendix B - Construction of Bateman data set

Bateman data set is based on the resolution of the Bateman equations, which is an ODE system modelling multi species reactions:

$$\partial_t \boldsymbol{\eta}(t) = \boldsymbol{\Sigma_r}\big(\boldsymbol{\eta}(t)\big) \cdot \boldsymbol{\eta}(t), \text{ with initial conditions } \boldsymbol{\eta}(0) = \boldsymbol{\eta_0},$$

and $\boldsymbol{\eta} \in (\mathbb{R}^+)^M$, $\boldsymbol{\Sigma}_r \in \mathbb{R}^{M\times M}$. Here, $f : (\boldsymbol{\eta_0}, t) \to \boldsymbol{\eta}(t)$ and we are interested in $\boldsymbol{\eta}(t)$, which is the concentration of each of the species S_k, with $k \in \{1, ..., M\}$. For physical applications, M ranges from tens to thousands. We consider the particular case $M = 11$. Matrix $\boldsymbol{\Sigma_r}\big(\boldsymbol{\eta}(t)\big)$ depends on reaction constants. Here, 4 reactions are considered and each reaction p has constant σ_p.

$$\begin{cases} (1) : S_1 + S_2 \to S_3 + S_4 + S_6 + S_7, \\ (2) : S_3 + S_4 \to S_2 + S_8 + S_{11}, \\ (3) : S_2 + S_{11} \to S_3 + S_5 + S_9, \\ (4) : S_3 + S_{11} \to S_2 + S_5 + S_6 + S_{10}, \end{cases}$$

with $\sigma_1 = 1$, $\sigma_2 = 5$, $\sigma_3 = 3$ and $\sigma_4 = 0.1$. To obtain $\boldsymbol{\Sigma_r}\big(\boldsymbol{\eta}(t)\big)$, we have to consider the species one by one. Here we give an example of how to construct the second row of $\boldsymbol{\Sigma_r}\big(\boldsymbol{\eta}(t)\big)$. The other rows are built the same way. Given the reaction equations :

$$\partial_t \eta_2 = -\sigma_1\eta_1\eta_2 + \sigma_2\eta_3\eta_4 - \sigma_3\eta_2\eta_{11} + \sigma_4\eta_3\eta_{11},$$

because S_2 disappears in reactions (1) and (3) involving S_1 and S_{11} as other reactants at rate σ_1 and σ_3, respectively, and appears in reactions (2) and (4) involving S_3, S_4 and S_3, S_{11} as reactants, at rate σ_2 and σ_4 respectively. Hence, the second row of $\boldsymbol{\Sigma_r}\big(\boldsymbol{\eta}(t)\big)$ is

$$[0,\ -\sigma_1\eta_1,\ 0,\ \sigma_2\eta_3,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ -\sigma_3\eta_2 + \sigma_4\eta_3],$$

with $\boldsymbol{\eta}(t)$ denoted by $\boldsymbol{\eta}$ to simplify the equation and η_i the i-th component of $\boldsymbol{\eta}$. To construct the training, validation and test data sets, we sample uniformly $(\boldsymbol{\eta}_0, t) \in [0, 1]^{12} \times [0, 5]$ 130000 times. We denote these samples $(\boldsymbol{\eta}_0, t)_i$ for $i \in \{1, ..., 130000\}$. Then, we apply a first order Euler solver with a time step of 10^{-3} to compute $f((\boldsymbol{\eta}_0, t)_i)$. As a result, NN's input is $(\boldsymbol{\eta}_0, t)$ and NN's output is $f((\boldsymbol{\eta}_0, t))$.