Goal-oriented sensitivity analysis of hyperparameters in deep learning

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

Goal-Oriented Sensitivity Analysis of Hyperparameters in Deep Learning

Paul Novello[1,2,3*], Gaël Poëtte[1], David Lugato[2] and Pietro Marco Congedo[2,3]

[1]CESTA, CEA, Le Barp, 33114, France.
[2]CMAP, Ecole Polytechnique, Palaiseau, 91120, France.
[3]Platon, Inria Paris Saclay, Palaiseau, 91120, France.

*Corresponding author(s). E-mail(s): paul.novello@outlook.fr;
Contributing authors: gael.poette@cea.fr; david.lugato@cea.fr;
pietro.congedo@inria.fr;

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 study the use of goal-oriented sensitivity analysis, based on the Hilbert-Schmidt Independence Criterion (HSIC), for hyperparameter analysis and optimization. Hyperparameters live in spaces that are often complex and awkward. They can be of different natures (categorical, discrete, boolean, continuous), interact, and have inter-dependencies. All this makes it non-trivial to perform classical sensitivity analysis. We alleviate these difficulties to obtain a robust analysis index that is able to quantify hyperparameters' relative impact on a neural network's final error. This valuable tool allows us to better understand hyperparameters and to make hyperparameter optimization more interpretable. We illustrate the benefits of this knowledge in the context of hyperparameter optimization and derive an HSIC-based optimization algorithm that we apply on MNIST and Cifar, classical machine learning data sets, but also on the approximation of Runge function and Bateman equations solution, of interest for scientific machine learning. This method yields competitive and cost-effective neural networks.

Keywords: Scientific machine learning, sensitivity analysis, Hilbert-Schmidt Independance Criterion, hyperparameter optimization, interpretability

1 Introduction

Hyperparameter optimization is ubiquitous in machine learning, and especially in deep learning, where neural networks are often cluttered with many hyperparameters. For applications to real-world machine learning tasks, finding good hyperparameters carries high stakes. Indeed, hyperparameters have a strong impact on the prediction error of neural networks as well as on their cost efficiency, which is an important criterion in scientific computing.

However, the balance between prediction accuracy and cost efficiency is difficult to find. For instance, recent breakthroughs owed to deep learning can be explained, among other reasons, by the ability to construct wider and deeper networks, while the computational cost of one neural network's prediction increases quadratically with its width and linearly with the depth [1]. More generally, there are many hyperparameters in deep learning, all impacting the error and some of them impacting the execution time. It justifies carefully selecting them.

This selection can be fastidious for different reasons. i) The high number of hyperparameters by itself makes this problem challenging. ii) Their impact on error changes very often depending on the problem, so it is difficult to adopt general best practices and permanently recommend hyperparameter values for every machine learning problem. iii) Hyperparameters can be of very different natures, and have non-trivial relations, like conditionality or interactions.

In this paper, we tackle these problems by proposing a goal-oriented sensitivity analysis that we adapt and apply to complex hyperparameter search space. To this end, we select a powerful metric used for global sensitivity analysis, called Hilbert-Schmidt Independence Criterion (HSIC) [2], which is a distribution dependence measure initially used for two-sample test problem [3]. Once adapted to hyperparameter search space, HSIC gives insights into hyperparameters' relative importance for a given deep learning problem. It allows identifying which hyperparameters really matter in this situation, thereby addressing challenges i) and ii). Nonetheless, because of challenge iii), using HSIC in hyperparameter spaces is non-trivial. First, hyperparameters can be discrete (width of the neural network), continuous (learning rate), categorical (activation function), or boolean (batch normalization [4]). Second, some hyperparameter's presence is conditional to others (moments decay rates specific to Adam optimizer [5]). Third, they can strongly interact (as shown in [6]: in some cases, it is better to increase depth and width by a similar factor). The metric should be able to compare hyperparameters reliably in such situations. We introduce solutions to overcome this last challenge and illustrate them with some simple examples and with hyperparameter optimization for the approximation of Runge function.

Once adapted to such complex spaces, we show that HSIC allows to understand hyperparameter relative importance better and to focus research efforts on specific hyperparameters. We also identify hyperparameters that have an impact on execution speed but not on the error. Then, we introduce ways of

reducing the hyperparameter's range of possible values to improve the stability of the training and neural network's cost efficiency. Finally, we propose an HSIC-based optimization methodology in two steps, one focused on essential hyperparameters and the other on remaining hyperparameters. It yields competitive and cost-effective neural networks for real-world machine learning problems: MNIST, Cifar10, and the approximation of the resolution of Bateman equations. This last problem is a physical data set of interest for scientific machine learning.

2 Sensitivity analysis as a new approach to hyperparameter optimization

In this section, we describe the challenges of hyperparameter optimization. We emphasize the limits of classical hyperparameter optimization algorithms used to tackle these challenges and legitimate the approach of sensitivity analysis.

2.1 Challenges of hyperparameter optimization

Let a neural network be described by n_h hyperparameters $\mathrm{x}_1, ..., \mathrm{x}_{n_h}$ with $\mathrm{x}_i \in \mathcal{X}_i$, $i \in \{1, ..., n_h\}$ and $\boldsymbol{\sigma} = (\mathrm{x}_1, ..., \mathrm{x}_{n_h})$. We denote $F(\boldsymbol{\sigma})$ the error of the neural network on a test data set once trained on a training data set. The aim of hyperparameter optimization is to find $\boldsymbol{\sigma}^* = \underset{\boldsymbol{\sigma}}{\operatorname{argmin}} F(\boldsymbol{\sigma})$. Even if its formulation is simple, neural networks hyperparameter optimization is a challenging task because of the great number of hyperparameters to optimize, the computational cost for evaluating $F(\boldsymbol{\sigma})$ and the complex structure of hyperparameter spaces. Figure 1 gives a graphical representation of a possible hyperparameter space and illustrates its complexity. Specific aspects to point out are the following ones :

- Hyperparameters 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 neural networks configurations, e.g. `dropout_rate` is not used when `dropout = False` or `adam_beta` is only involved when `optimizer = adam`. In this case, we denote them as "conditional", otherwise we call them "main" hyperparameters.

2.2 Classical hyperparameter optimization techniques

Many techniques have been introduced to tackle the problem of hyperparameter optimization. Grid search or random search [7] uniformly explore the search space. The main difference between the two methods is that hyperparameters

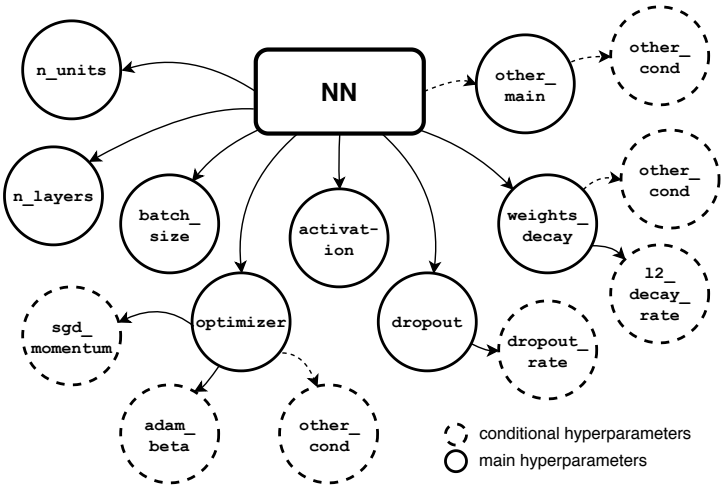


Figure 1: Example of hyperparameter space.

values are chosen on a uniform grid for a grid search. These values are deterministic, whereas, for a random search, hyperparameters values are randomly sampled from a uniform distribution in a Monte Carlo fashion. The main advantages of random search over grid search are that it allows for more efficient exploration of the hyperparameter search and that it is not constrained to a grid, so it does not suffer from the curse of dimensionality - which is a problem here since the hyperparameters can be pretty numerous. On the other hand, these two methods can be costly because they require training a neural network for each hyperparameter configuration, so exploring the search space can be computationally very expensive.

Some methods aim at reducing the cost of such searches. For instance, Successive Halving [8] and Hyperband [9] train neural networks in parallel, like in grid search or random search, and stop their training after a certain number of epochs. Then, they choose the best half of neural networks and carry on the training only for these neural networks, for the same number of epochs, and so on. This procedure allows testing more hyperparameters values for the same computational budget.

On the contrary, other methods are designed to improve the search quality with less training instances. Bayesian optimization, first introduced in [10], is based on the approximation of the loss function by a surrogate model. After an initial uniform sampling of hyperparameter 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 [11], is supposed to lead to hyperparameter configurations that will improve the error. Therefore, it focuses the computation on potentially

better hyperparameters values instead of randomly exploring the hyperparameter space. The surrogate model can be a Gaussian process [12], a kernel density estimator [13] or even a neural network [14].

Model-based hyperparameter optimization is not easily and naturally applicable to conditional or categorical hyperparameters that often appear when optimizing a neural network architecture. Such categorical hyperparameter can be the type of convolution layer for a convolutional neural network, regular convolution or depth-wise convolution [15]; and a conditional hyperparameter could be the specific parameters of each different convolution type. Neural architecture search explicitly tackles this problem. It dates back to evolutionary and genetic algorithms [16] and has been the subject of many recent works. For instance, [17] models the architecture as a graph, or [18, 19] use reinforcement learning to automatically construct representations of the search space. See [20] for an exhaustive survey of this field. Nevertheless, their implementation can be tedious, often involving numerous hyperparameters themselves.

Classical hyperparameter optimization methods handle hyperparameter optimization quite successfully. However, they are end-to-end algorithms that return one single neural network. The user does not interact with the algorithm during its execution. This lack of interactivity has many automating advantages but can bring some drawbacks. First, these methods do not give any insight on the relative importance of hyperparameters, whereas it may be of interest in the first approach to a machine learning problem. They are black boxes and not interpretable. Second, one could have other goals than the accuracy of a neural network, like execution speed or memory consumption. Some works like [19] introduce multi-objective hyperparameter optimization, but it requires additional tuning of the hyperparameter optimization algorithm itself. Finally, there may be flaws in the hyperparameter space, like a useless hyperparameter that could be dropped but is included in the search space and becomes a nuisance for the optimization. This aspect is all the more problematic since some popular algorithms, like gaussian process-based Bayesian optimization, suffer from the curse of dimensionality. We can sum up the drawbacks as lack of interpretability, difficulties in a multi-objective context, and unnecessary search space complexity.

2.3 Benefits of Sensitivity Analysis

In this work, we alleviate these concerns by mixing hyperparameter optimization with hyperparameter analysis. In other words, we construct an approach to hyperparameter optimization that relies on assessing hyperparameter's effects on the neural network's performances.

One powerful tool to analyze the effect of some input variables on the variability of a quantity of interest is sensitivity analysis [21]. Sensitivity analysis consists of studying the sensitivity of the output of a function to its inputs. We could define this function as F and its inputs as $\boldsymbol{\sigma}$. Then, it would be possible to make hyperparameter optimization benefit from characteristics of sensitivity analysis. Indeed, sensitivity analysis allows specifically:

- Analyzing the relative importance of input variables for explaining the output, which helps to answer the lack of interpretability problem. We could better understand hyperparameters' impact on the neural network error.
- Selecting practically convenient values for input variables with a limited negative impact on the output. It simplifies the multi-objective approach since we could, for instance, select values that improve execution speed with a limited impact on the neural network error.
- Identifying where to efficiently put research efforts to improve the output, which answers the unnecessary search space complexity problem. Indeed, we could focus on fewer hyperparameters to optimize by knowing which of them most impact the neural network error.

2.4 Goal-oriented sensitivity analysis

Several types of sensitivity measures can be estimated after an initial sampling of input vectors and their corresponding output values. The 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 [22], but they only assess the contribution of variables to the output variance. Goal-oriented Sobol indices [23] or uncertainty importance measure [24] 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 $\mathcal{O}(\frac{1}{\sqrt{n_s}})$ requires $(n_h + 2) \times n_s$ sample evaluations [25], which can be prohibitive for hyperparameter analysis. Another type of metrics, called dependence measures, assesses the dependence between x_i (that can be a random vector) and the output $F(\boldsymbol{\sigma})$ [26]. 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 $\mathbb{P}_{\mathrm{x}_i}\mathbb{P}_{\mathrm{y}}$ and $\mathbb{P}_{\mathrm{x}_i,\mathrm{y}}$, where $\mathrm{x}_i \sim \mathbb{P}_{\mathrm{x}_i}$ and $\mathrm{y} = F(\boldsymbol{\sigma}) \sim \mathbb{P}_{\mathrm{y}}$, since $\mathbb{P}_{\mathrm{x}_i\mathrm{y}} = \mathbb{P}_{\mathrm{x}_i}\mathbb{P}_{\mathrm{y}}$ when x_i and y are independent. In [26], the author gives several examples of indices based on dissimilarity measures like f-divergences [27] or integral probability metrics [28]. These indices are easier and less expensive to estimate (n_s training instances instead of $(n_h + 2) \times n_s$) than variance-based measures since they only need a simple Monte Carlo design of experiment.

This work focuses on a specific dependence measure, known as the Hilbert-Schmidt Independence Criterion (HSIC). The following sections are dedicated to the description of HSIC and its adaptation to hyperparameter optimization.

3 HSIC-based goal-oriented sensitivity analysis

In this section, we recall the definition of Hilbert Schmidt Independence Criterion, how to use it in practice, and how to adapt it in order to perform goal-oriented sensitivity analysis.

3.1 From Integral Probability Metrics to Maximum Mean Discrepancy

Let $\mathbf{x}$ and $\mathbf{y}$ be two random variables of probability distribution $\mathbb{P}_\mathbf{x}$ and $\mathbb{P}_\mathbf{y}$ defined in $\mathcal{X}$. [3] show that distributions $\mathbb{P}_\mathbf{x} = \mathbb{P}_\mathbf{y}$ if and only if $\mathbb{E}[f(\mathbf{x})] - \mathbb{E}[f(\mathbf{y})] = 0$ 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) [28].

Let $\mathcal{F}$ be a class of functions, $f : \mathcal{X} \to \mathbb{R}$. An IPM γ is defined as

$$\gamma(\mathcal{F}, \mathbb{P}_\mathbf{x}, \mathbb{P}_\mathbf{y}) = \sup_{f \in \mathcal{F}} |\mathbb{E}[f(\mathbf{x})] - \mathbb{E}[f(\mathbf{y})]|. \tag{1}$$

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 [2], this choice is motivated by the capacity of RKHS to embed probability distributions efficiently. The authors define $\mu_\mathbf{x}$ such that $\mathbb{E}(f(\mathbf{x})) = \langle f, \mu_\mathbf{x} \rangle_\mathcal{H}$ as the mean embedding of $\mathbb{P}_\mathbf{x}$. Then, $\gamma_k^2(\mathbb{P}_\mathbf{x}, \mathbb{P}_\mathbf{y})$ can be written

$$\begin{aligned} \gamma_k^2(\mathbb{P}_\mathbf{x}, \mathbb{P}_\mathbf{y}) &= \|\mu_\mathbf{x} - \mu_\mathbf{y}\|_\mathcal{H}^2. \\ &= \int\int k(\boldsymbol{x}, \boldsymbol{x})(p_\mathbf{x}(\boldsymbol{x}) - p_\mathbf{y}(\boldsymbol{x}))(p_\mathbf{x}(\boldsymbol{y}) - p_\mathbf{y}(\boldsymbol{y}))d\boldsymbol{x}d\boldsymbol{x}' \\ &= \mathbb{E}[k(\mathbf{x}, \mathbf{x}')] + \mathbb{E}[k(\mathbf{y}, \mathbf{y}')] - 2\mathbb{E}[k(\mathbf{x}, \mathbf{y})], \end{aligned} \tag{2}$$

where $p_\mathbf{x}(\boldsymbol{x})dx = d\mathbb{P}_\mathbf{x}$, $p_\mathbf{y}(\boldsymbol{y})dy = d\mathbb{P}_\mathbf{y}$, $\mathbf{x}, \mathbf{x}'$ are iid (independent and identically distributed) and $\mathbf{y}, \mathbf{y}'$ are iid. After a Monte Carlo sampling of $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_{n_s}\}$ and $\{\boldsymbol{y}_1, ..., \boldsymbol{y}_{n_s}\}$, $\gamma_k^2(\mathbb{P}_\mathbf{x}, \mathbb{P}_\mathbf{y})$ can thus be estimated by $\hat{\gamma}_k^2(\mathbb{P}_\mathbf{x}, \mathbb{P}_\mathbf{y})$, with

$$\hat{\gamma}_k^2(\mathbb{P}_\mathbf{x}, \mathbb{P}_\mathbf{y}) = \sum_{j=1}^{n_s}\sum_{l=1}^{n_s} k(\boldsymbol{x}_j, \boldsymbol{x}_l) + \sum_{j=1}^{n_s}\sum_{l=1}^{n_s} k(\boldsymbol{y}_j, \boldsymbol{y}_l) - 2\sum_{j=1}^{n_s}\sum_{l=1}^{n_s} k(\boldsymbol{x}_j, \boldsymbol{y}_l), \tag{3}$$

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

3.2 The kernel choice

Equation (2) involves to choose a kernel k. In practice, k is chosen among a class of kernels that depends on a set of parameters $\mathbf{h} \in \mathbf{H}$, where $\mathbf{H}$ is a kernel parameter space. We therefore temporarily denote the kernel by $k_\mathbf{h}$. Examples of kernels are the Gaussian Radial Basis Function $k_h : (\boldsymbol{x}, \boldsymbol{y}) \to \exp(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2h^2})$ or the Matérn function $k_\mathbf{h} : (\boldsymbol{x}, \boldsymbol{y}) \to \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{\|\boldsymbol{x}-\boldsymbol{y}\|}{\eta}\right)^\nu K_\nu \left(\sqrt{2\nu}\frac{\|\boldsymbol{x}-\boldsymbol{y}\|}{\eta}\right)$, where $\mathbf{h} = \{\sigma, \nu, \eta\}$, Γ is the gamma function and K_ν is the modified Bessel

function of the second kind. In [29], 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 $\mathbb{P}_{\mathbf{x}}$ and $\mathbb{P}_{\mathbf{y}}$, the final parameter $\mathbf{h}^*$ should be chosen such that

$$\gamma^2_{k_{\mathbf{h}^*}}(\mathbb{P}_{\mathbf{x}}, \mathbb{P}_{\mathbf{y}}) = \sup_{\mathbf{h}\in\mathbf{H}} \gamma^2_{k_{\mathbf{h}}}(\mathbb{P}_{\mathbf{y}}, \mathbb{P}_{\mathbf{y}}). \tag{4}$$

The authors suggest focusing on unnormalized kernel families, like Gaussian Radial Basis Functions $\left\{k_h : (\boldsymbol{x}, \boldsymbol{y}) \to \exp(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2h^2}), h \in (0, \infty)\right\}$, also used in [26], for which they demonstrate that $\hat{\gamma}^2_{k_{\mathbf{h}^*}}(\mathbb{P}_{\mathbf{x}}, \mathbb{P}_{\mathbf{y}})$, defined as

$$\hat{\gamma}^2_{k_{\mathbf{h}^*}}(\mathbb{P}_{\mathbf{x}}, \mathbb{P}_{\mathbf{y}}) = \sup_{\mathbf{h}\in\mathbf{H}} \left[\sum_{j=1}^{n_s}\sum_{l=1}^{n_s} k_{\mathbf{h}}(\boldsymbol{x}_j, \boldsymbol{x}_l) + \sum_{j=1}^{n_s}\sum_{l=1}^{n_s} k_{\mathbf{h}}(\boldsymbol{y}_j, \boldsymbol{y}_l) - 2\sum_{j=1}^{n_s}\sum_{l=1}^{n_s} k_{\mathbf{h}}(\boldsymbol{x}_j, \boldsymbol{y}_l) \right], \tag{5}$$

is a consistent estimator of $\gamma^2_{k_{\mathbf{h}^*}}(\mathbb{P}_{\mathbf{x}}, \mathbb{P}_{\mathbf{y}})$. It is thus possible to choose $\mathbf{h}$ by maximizing $\hat{\gamma}^2_{k_{\mathbf{h}}}(\mathbb{P}_{\mathbf{x}}, \mathbb{P}_{\mathbf{y}})$ with respect to $\mathbf{h}$. Therefore, in this work, we use Gaussian Radial Basis Functions kernel. Once $\mathbf{h}^*$ is chosen, the approximation error of $\hat{\gamma}^2_{k_{\mathbf{h}^*}}(\mathbb{P}_{\mathbf{x}}, \mathbb{P}_{\mathbf{y}})$ can also be estimated like in Section 3.1. It is important to note that $\gamma^2_{k_{\mathbf{h}}}(\mathbb{P}_{\mathbf{x}}, \mathbb{P}_{\mathbf{y}})$ can be estimated in a $\mathcal{O}(n_s^2)$ computational complexity, which is not expensive given usual values of n_s in hyperparameter optimization context. The total complexity of the minimization process depends on the minimization algorithm, but since the optimization problem is low dimensional - there is never more than a handful of kernel parameters - the whole process is always cost effective. To simplify the notations, we denote $k_{\mathbf{h}^*}$ by k in the following sections.

3.3 Hilbert-Schmidt Independence Criterion (HSIC) for goal-oriented sensitivity analysis

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

$$HSIC(\mathbf{x}, \mathbf{y}) = \gamma^2_k(\mathbb{P}_{\mathbf{xy}}, \mathbb{P}_{\mathbf{x}}\mathbb{P}_{\mathbf{y}}) = \|\mu_{\mathbf{xy}} - \mu_{\mathbf{x}}\mu_{\mathbf{y}}\|_{\mathcal{G}}. \tag{6}$$

Then, HSIC measures the distance between $\mathbb{P}_{\mathbf{xy}}$ and $\mathbb{P}_{\mathbf{y}}\mathbb{P}_{\mathbf{x}}$ embedded in $\mathcal{H}$. Indeed, since $\mathbf{x} \perp \mathbf{y} \Rightarrow \mathbb{P}_{\mathbf{xy}} = \mathbb{P}_{\mathbf{y}}\mathbb{P}_{\mathbf{x}}$, the closer these distributions are, in the sense of γ_k, the more independent they are.

In [30], the authors present a goal-oriented sensitivity analysis by focusing on the sensitivity of F w.r.t. $\mathbf{x}_i$ when $\mathrm{y} = F(\mathbf{x}_1, ..., \mathbf{x}_{n_h}) \in \mathbf{Y}$, 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, $\mathrm{z} = \mathbf{1}_{\mathrm{y}\in\mathbf{Y}}$. Then,

$$HSIC(\mathbf{x}_i, \mathrm{z}) = \mathbb{P}(\mathrm{z} = 1)^2 \times \gamma^2_k(\mathbb{P}_{\mathbf{x}_i|\mathrm{z}=1}, \mathbb{P}_{\mathbf{x}_i}), \tag{7}$$

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

$$HSIC(\mathbf{x}_i, \mathrm{z}) = \mathbb{P}(\mathrm{z}=1)^2 \Big[\mathbb{E}[k(\mathbf{x}_i, \mathbf{x}_i')] + \mathbb{E}[k(\mathbf{z}, \mathbf{z}')] - 2\mathbb{E}[k(\mathbf{x}_i, \mathbf{z})] \Big], \quad (8)$$

where $\mathbf{x}_i, \mathbf{x}_i'$ are iid and $\mathbf{z}, \mathbf{z}'$ are iid. It is estimated for each $\mathbf{x}_i$ using Monte Carlo estimators denoted by $S_{\mathbf{x}_i,\mathbf{Y}}$, based on samples $\{\boldsymbol{x}_{i,1}, ..., \boldsymbol{x}_{i,n_s}\}$ from $\mathbf{x}_i \sim d\mathbb{P}_{\mathbf{x}_i}$ and corresponding $\{z_1, ..., z_{n_s}\}$ drawn from z. The estimator $S_{\mathbf{x}_i,\mathbf{Y}}$ is defined as

$$\begin{aligned} S_{\mathbf{x}_i,\mathbf{Y}} = \mathbb{P}(\mathrm{z}=1)^2 \Bigg[& \frac{1}{m^2} \sum_{j=1}^{n_s} \sum_{l=1}^{n_s} k(\boldsymbol{x}_{i,j}, \boldsymbol{x}_{i,l}) \delta(z_j = 1) \delta(z_l = 1) \\ & + \frac{1}{{n_s}^2} \sum_{j=1}^{n_s} \sum_{l=1}^{n_s} k(\boldsymbol{x}_{i,j}, \boldsymbol{x}_{i,l}) \\ & - \frac{2}{n_s m} \sum_{j=1}^{n_s} \sum_{l=1}^{n_s} k(\boldsymbol{x}_{i,j}, \boldsymbol{x}_{i,l}) \delta(z_l = 1) \Bigg], \end{aligned} \quad (9)$$

with $m = \sum_k^{n_s} \delta(z_k = 1)$ and $\delta(a) = 1$ if a is True and 0 otherwise. We use this metric in the following. This section mainly summed up the mathematics on which the sensitivity indices are based and how they are used in practice in a sensitivity analysis context. The following section is devoted to the application of HSIC in hyperparameter spaces.

4 Application of HSIC to hyperparameter spaces

HSIC has two advantages that make it stand out from other sensitivity indices and make it particularly suitable to hyperparameter spaces. First, equation (9) emphasizes that it is possible to estimate HSIC using simple Monte Carlo estimation. Hence, in the context of hyperparameter optimization, such indices can be estimated after a classical random search. Secondly, using equation (7), HSIC allows to perform goal-oriented sensitivity analysis easily, i.e. to assess the importance of each hyperparameter for the error to reach a given $\mathbf{Y}$. For hyperparameter analysis, $\mathbf{Y}$ can be chosen to be the sub-space for which $F(\mathrm{x}_1, ..., \mathrm{x}_{n_h})$ is in the best percentile p of a metric (L_2 error, accuracy,...), say $p = 10\%$. Then, the quantity $S_{\mathrm{x}_i,\mathbf{Y}}$ measures the importance of each hyperparameter x_i for obtaining the 10% best neural networks.

However, one cannot use HSIC as is in hyperparameter analysis. Indeed, hyperparameters do not live in the same measured space, they could interact

with each other, and some are not directly involved for each configuration. In the following sections, we suggest some original solutions to these issues.

To illustrate the performances of these solutions, we consider a toy example which is the approximation of Runge function $r : x \to \frac{1}{1+15x^2}$, $x \in [-1, 1]$ by a fully connected neural network. This approximation problem is a historical benchmark of approximation theory. We consider $n_h = 14$ different hyperparameters (see **Appendix A** for details). We randomly draw $n_s = 10000$ hyperparameter 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.

In Section 4.1, we introduce a transformation to deal with hyperparameters that do not live in the same measured space. Then, in Section 4.2 we explain how to use HSIC to evaluate hyperparameters' interactions. Finally, in Section 4.3 we deal with conditionality between hyperparameters.

4.1 Normalization of hyperparameter space

Hyperparameters can be defined in 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, it may be useful to sample hyperparameters with a non-uniform distribution (e.g. log-uniform for `learning_rate`). Doing so affects HSIC value and its interpretation, which is undesirable since this distribution choice is arbitrary and only relies on practical considerations. Let us illustrate this phenomenon in the following example.

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

$$f(\mathrm{x}_1, \mathrm{x}_2) = \begin{cases} 1 & \text{if } \mathrm{x}_1 \in [0, 1], \mathrm{x}_2 \in [0, 1], \\ 0 & \text{otherwise.} \end{cases}$$

Suppose we want to assess the importance of x_1 and x_2 for reaching the goal $f(\mathrm{x}_1, \mathrm{x}_2) = 1$ 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 $\mathrm{x}_1 \sim \mathcal{N}(1, 0.1, [0, 2])$ (normal distribution of mean 1 and variance 0.1 truncated between 0 and 2) and $\mathrm{x}_2 \sim \mathcal{U}[0, 2]$. We compute $S_{\mathrm{x}_1,\mathbf{Y}}$ and $S_{\mathrm{x}_2,\mathbf{Y}}$ with $n_s = 10000$ points and display their value in Table 1. The values of $S_{\mathrm{x}_1,\mathbf{Y}}$ and $S_{\mathrm{x}_2,\mathbf{Y}}$ are quite different. It is natural since their chosen initial distribution is different. However, this choice has nothing to do with their actual importance; it should not have any impact on the importance measure. Here, we could erroneously conclude that x_2 is more important than x_1.

This example shows that we have to make $S_{\mathrm{x}_i,\mathbf{Y}}$ and $S_{\mathrm{x}_j,\mathbf{Y}}$ comparable in order to say that hyperparameter x_i is more important than hyperparameter

	x_1	x_2
$S_{\mathrm{x},\mathbf{Y}}(\times 10^{-2})$	1.17 ± 0.05	1.55 ± 0.05

Table 1: $S_{\mathrm{x},\mathbf{Y}}$ values for x_1 and x_2

	u_1	u_2
$S_{\mathrm{x},\mathbf{Y}}(\times 10^{-2})$	1.54 ± 0.05	1.55 ± 0.05

Table 2: $S_{\mathrm{x},\mathbf{Y}}$ values for u_1 and u_2

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 $S_{\mathrm{x}_i,\mathbf{Y}}$ that are robust to the choice of $d\mathbb{P}_{\mathrm{x}_i}$. To tackle this problem, we introduce the following approach for comparing variables with HSIC. Let Φ_i be the CDF of x_i. We have that $\Phi_i(\mathrm{x}_i) = \mathrm{u}_i$, with $\mathrm{u}_i \sim \mathcal{U}[0,1]$. After an initial Monte Carlo sampling of hyperparameter x_i, which can be a random search, we can apply Φ_i to each input point to obtain u_i corresponding to x_i with u_i iid, so living in the same measured space. Yet, one must be aware that to obtain $\mathrm{u}_i \sim \mathcal{U}[0,1]$, its application is different for continuous and discrete variables:

- for continuous variables, $\Phi_i(\mathrm{x}_i)$ 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, $\Phi_i(\mathrm{x}_i)$ is not a bijection between $\mathcal{X}_i$ and $[0,1]$. Suppose that x_i is a discrete variable with p possible values $\{\mathrm{x}_i[1], ..., \mathrm{x}_i[p]\}$, each with probability w_p. Let us encode $\{\mathrm{x}_i[1], ..., \mathrm{x}_i[p]\}$ by $\{1, ..., p\}$. Then, $\Phi_i(\mathrm{x}_i) = \sum_{j=1}^{p} w_j \mathbf{1}_{[\mathrm{x}_i \leq j]}(\mathrm{x}_i)$. When Φ_i is applied as is, $\Phi_i(\mathrm{x}_i)$ is not uniform. To overcome that, one can simply use $\mathrm{u}_i = \sum_{j=1}^{p} \mathcal{U}[\sum_{k<j} w_k, \sum_{k<j+1} w_k]\delta(\mathrm{x}_i = j)$. This trick, introduced in [31], is commonly used in Monte Carlo resolution of Partial Differential Equations [32]. As a result, $\mathrm{u}_i \sim \mathcal{U}[0,1]$.

Finally, all we have to do is to sample x_i, like in a classical random search, following the distribution we want, and then apply Φ_i to obtain u_i. The corresponding HSIC estimation is $S_{\mathrm{u}_i,\mathbf{Y}}$. It only involves u_i and $\mathrm{u}_i|\mathrm{z} = 1$ and since u_i are iid, the comparison of different $S_{\mathrm{u}_i,\mathbf{Y}}$ becomes relevant. Coming back to the previous example, Table 2 displays values of $S_{\mathrm{u}_1,\mathbf{Y}}$ and $S_{\mathrm{u}_2,\mathbf{Y}}$. 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 $S_{\mathrm{u}_i,\mathbf{Y}}$ by $S_{\mathrm{x}_i,\mathbf{Y}}$ for clarity but always resort to this transformation.

Let us apply this methodology to the Runge approximation hyperparameter analysis problem. Figure 2 displays a comparison between $S_{\mathrm{x}_i,\mathbf{Y}}$ for hyperparameters of the Runge approximation problem, with $\mathbf{Y}$ the set of the 10% best neural networks. For readability, we order x_i by $S_{\mathrm{x}_i,\mathbf{Y}}$ value in the legend and 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 non-impactful because their $S_{\mathrm{x}_i,\mathbf{Y}}$ values are low. Besides, these values are lower than the error evaluation. It could be only noise, and therefore these hyperparameters can not be ordered on this basis.

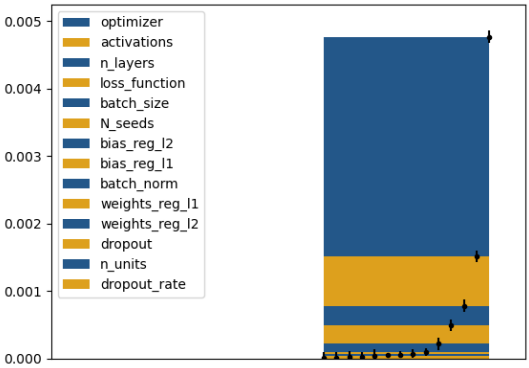


Figure 2: Comparison of $S_{\mathrm{x}_i,\mathbf{Y}}$ for hyperparameters in Runge approximation problem. The hyperparameters are ordered from the most important (top of the legend) to the least important (bottom of the legend), and their value is graphically represented in a stacked bar plot following the same order.

4.2 Interactions between hyperparameters

If $S_{\mathrm{x}_i,\mathbf{Y}}$ is low, it means that $\mathbb{P}_{\mathrm{x}_i}$ and $\mathbb{P}_{\mathrm{z}}$ are close to independent. We could want to conclude that x_i has a limited impact on Y. However, x_i may have an impact due to its interactions with some of the other hyperparameters. In other words, let x_i and x_j be two variables, it can happen that $S_{\mathrm{x}_i,\mathbf{Y}}$ and $S_{\mathrm{x}_j,\mathbf{Y}}$ are low while $S_{(\mathrm{x}_i,\mathrm{x}_j),\mathbf{Y}}$ is high.

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

$$f(\mathrm{x}_1,\mathrm{x}_2,\mathrm{x}_3) = \begin{cases} 1 & \text{if } \mathrm{x}_1 \in [0,1], \mathrm{x}_2 \in [1,2], \mathrm{x}_3 \in [0,1], \\ 1 & \text{if } \mathrm{x}_1 \in [0,1], \mathrm{x}_2 \in [0,1], \mathrm{x}_3 \in [1,2], \\ 0 & \text{otherwise.} \end{cases}$$

In that case, let $\mathbf{Y} = \{1\}$, $\forall x \in [0,2]$ we have $p_{\mathrm{x}_2|\mathrm{z}=1}(x) = p_{\mathrm{x}_2}(x)$ and $p_{\mathrm{x}_3|\mathrm{z}=1}(x) = p_{\mathrm{x}_3}(x)$. Hence, according to equation (8) we have $HSIC(\mathrm{x}_2,\mathrm{z}) = HSIC(\mathrm{x}_3,\mathrm{z}) = 0$. However, we have

$$\begin{aligned} HSIC(\mathrm{x}_1,\mathrm{z}) &= \mathbb{P}(\mathrm{z}=1)^2 \int_{[0,2]^2} k(x,x')\left[p_{\mathrm{x}_1|\mathrm{z}=1}(x) - p_{\mathrm{x}_1}(x)\right] \\ &\quad \times \left[p_{\mathrm{x}_1|\mathrm{z}=1}(x') - p_{\mathrm{x}_1}(x')\right] dx dx' \\ &= \frac{1}{8}\Big[\int_{[0,1]\times[0,1]} k(x,x')dxdx' + \int_{[1,2]\times[1,2]} k(x,x')dxdx' \\ &\quad - 2\int_{[0,1]\times[1,2]} k(x,x')dxdx'\Big], \end{aligned}$$

so for non-trivial choice of k, $HSIC(\mathrm{x}_1, \mathrm{z}) \neq 0$. 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 $\mathrm{x}_1 \in [0, 1]$, $f(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3) = 1$ if $\mathrm{x}_2 \in [1, 2]$ and $\mathrm{x}_3 \in [0, 1]$ but $f(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3) = 0$ if $\mathrm{x}_2 \in [1, 2]$ and $\mathrm{x}_3 \in [1, 2]$. This is illustrated in Figure 3, which displays the histograms of x_1 and $\mathrm{x}_1|\mathrm{z} = 1$, x_2 and $\mathrm{x}_2|\mathrm{z} = 1$, x_3 and $\mathrm{x}_3|\mathrm{z} = 1$, obtained from 10000 points $(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3)$ sampled uniformly in the definition domain of f.

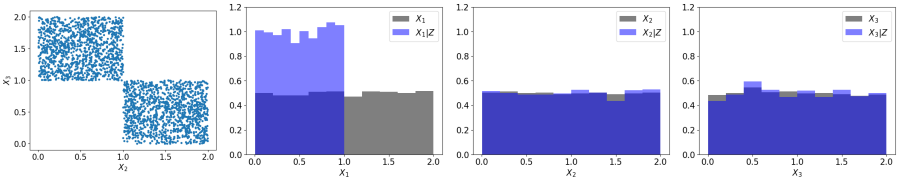


Figure 3: **From left to right: 1** - Pairs of $(\mathrm{x}_2|\mathrm{z} = 1, \mathrm{x}_3|\mathrm{z} = 1)$. **2** - Histogram of x_1 and $\mathrm{x}_1|\mathrm{z} = 1$. **3** - Histogram of x_2 and $\mathrm{x}_2|\mathrm{z} = 1$. **4** - Histogram of x_3 and $\mathrm{x}_3|\mathrm{z} = 1$.

Histograms are the same for x_2, $\mathrm{x}_2|\mathrm{z} = 1$ and x_3, $\mathrm{x}_3|\mathrm{z} = 1$ (uniform between 0 and 2), but different for x_1, $\mathrm{x}_1|\mathrm{z} = 1$. Therefore, HSIC being a distance measure between x_1 and $\mathrm{x}_1|\mathrm{z} = 1$, 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 $S_{\mathrm{x}_1,\mathbf{Y}}$, $S_{\mathrm{x}_2,\mathbf{Y}}$, $S_{\mathrm{x}_3,\mathbf{Y}}$ and $S_{(\mathrm{x}_2,\mathrm{x}_3),\mathbf{Y}}$ after simulating f for $n_s = 2000$ points. We also compute $S_{(\mathrm{x}_4,\mathrm{x}_5),\mathbf{Y}}$, with x_4 and x_5 two dummy variables, uniformly distributed, to have a reference for $S_{(\mathrm{x}_2,\mathrm{x}_3),\mathbf{Y}}$. The results can be found in Table 3. They show that $S_{\mathrm{x}_1,\mathbf{Y}}$ and $S_{(\mathrm{x}_2,\mathrm{x}_3),\mathbf{Y}}$ are of the same order while $S_{\mathrm{x}_2,\mathbf{Y}}$, $S_{\mathrm{x}_3,\mathbf{Y}}$ and $S_{(\mathrm{x}_4,\mathrm{x}_5),\mathbf{Y}}$ are two decades lower than $S_{\mathrm{x}_1,\mathbf{Y}}$, which confirms that $S_{\mathrm{x},\mathbf{Y}}$ may be low while interactions are impactful.

	x_1	x_2	x_3	$(\mathrm{x}_2, \mathrm{x}_3)$	$(\mathrm{x}_4, \mathrm{x}_5)$
$S_{\mathrm{x},\mathbf{Y}}$	1.51×10^{-2}	6.26×10^{-6}	1.64×10^{-5}	3.47×10^{-3}	3.70×10^{-6}

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

Additionally, we display the $S_{(\mathrm{x}_i,\mathrm{x}_j),\mathbf{Y}}$ for each pair of variable x_i and x_j on Figure 4. We can see that for variables other than x_1, $S_{(\mathrm{x}_i,\mathrm{x}_j),\mathbf{Y}}$ is high only for $i = 2$ and $j = 3$. This example shows that it is necessary to compute $S_{\mathrm{x},\mathbf{Y}}$ of joint variables to perceive the importance of interactions between variables.

The values are easy to interpret in this example because we know the behavior of the underlying function f. In practice, $S_{\mathrm{x}_1,\mathbf{Y}}$ and $S_{(\mathrm{x}_2,\mathrm{x}_3),\mathbf{Y}}$ can not be compared because $(\mathrm{x}_2, \mathrm{x}_3)$ 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, $S_{(\mathrm{x}_i,\mathrm{x}_j),\mathbf{Y}}$ is always the highest when $i = 1$, regardless of j. In fact, if for a given variable x_i,

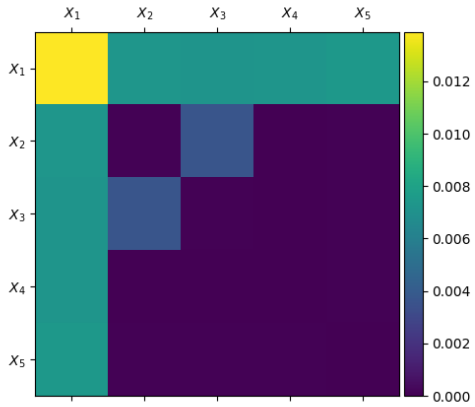
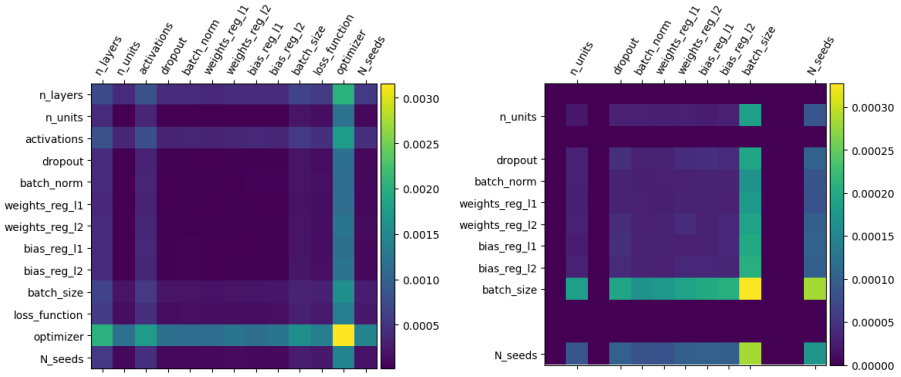


Figure 4: $S_{\mathrm{x},\mathbf{Y}}$ for each pair of variable.

$S_{\mathrm{x}_i,\mathbf{Y}}$ is high, so will be $S_{(\mathrm{x}_i,\mathrm{x}_j),\mathbf{Y}}$ for any other variable x_j. Hence, care must be taken to only compare interactions of low $S_{\mathrm{x},\mathbf{Y}}$ variables with each others, and not with high $S_{\mathrm{x},\mathbf{Y}}$ variables. Coming back to Runge approximation example, Figure 5a displays the $S_{(\mathrm{x}_i,\mathrm{x}_j),\mathbf{Y}}$ 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) $S_{(\mathrm{x}_i,\mathrm{x}_j),\mathbf{Y}}$ for each pair of hyperparameters. (b) $S_{(\mathrm{x}_i,\mathrm{x}_j),\mathbf{Y}}$ for each pair of hyperparameters, except for `optimizer`, `activation`, `n_layers` and `loss_function`. The grid can be read symmetrically with respect to the diagonal.

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 the remaining hyperparameters. It is normal that $S_{(\texttt{batch\_size},\mathrm{x}_j),\mathbf{Y}}$ 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.

4.3 Conditionality between hyperparameters

Conditionality between hyperparameters, which often arises in Deep Learning, is a non-trivial challenge in hyperparameter optimization. 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 hyperparameter optimization between disjoint groups of hyperparameters that are always involved together, like in [13]. Then, two separate instances of hyperparameter optimization are created, one for the main hyperparameters and another for `dropout_rate`. The second **(ii)** considers these hyperparameters as if they were always involved, even if they are not, like in [33]. 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. 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 a training instance. In Runge approximation hyperparameter analysis, since `dropout_rate` is the only conditional hyperparameter, it would mean to split the hyperparameters between two groups: {`dropout_rate`} and another containing all the others. 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 and compare it to each other hyperparameter. Here, `dropout_rate` could not be compared to any other hyperparameters.

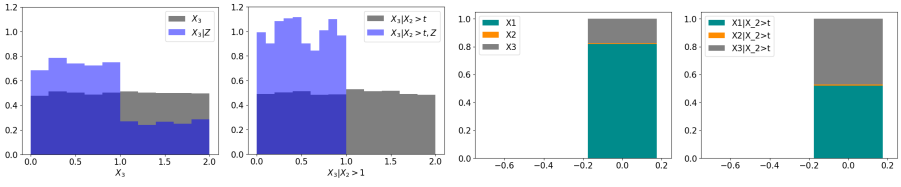
(ii) In the second case, if we apply HSIC with the same idea, we could compute HSIC of a hyperparameter with irrelevant values coming from configurations where it is not involved. Two situations can occur. First, if a conditional variable x_i is never involved in the hyperparameter configurations that yield the p-percent best accuracies (depending on the percentile chosen), the values used for computing $S_{\mathrm{x}_i,\mathbf{Y}}$, i.e. $\mathrm{x}_i|\mathrm{z}=1$, are drawn from the initial, uniform distribution u_i. Then, $S_{\mathrm{x}_i,\mathbf{Y}}$ 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 neural networks have used this hyperparameter. However, if x_i

is only involved in a subset of all tested hyperparameter configurations and is impactful in that case, $S_{\mathrm{x}_i,\mathbf{Y}}$ 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 actual impact. The following example illustrates this phenomenon.

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

$$f(\mathrm{x}_1,\mathrm{x}_2,\mathrm{x}_3) = \begin{cases} B & \text{if } \mathrm{x}_1 \in [0,1], \mathrm{x}_2 \in [0,t] \\ 1 & \text{if } \mathrm{x}_1 \in [0,1], \mathrm{x}_2 \in [t,2], \mathrm{x}_3 \in [0,1], \\ 0 & \text{otherwise,} \end{cases}$$

With B a Bernoulli variable of parameter 0.5 and $t \in [0,2]$ (so that $S_{\mathrm{x}_2,\mathbf{Y}}$ is low). 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 $\mathrm{x}_2 > t$. In these cases, it is as important as x_1 for reaching $\mathbf{Y}$ and we would like to retrieve this information. Parameter t allows controlling how many 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 $\mathrm{x}_3|\mathrm{z} = 1$ **(b)** - Histogram of x_3 and $\mathrm{x}_3|\mathrm{x}_2 > t, \mathrm{z}$. **(c)** - $S_{\mathrm{x},\mathbf{Y}}$ for x_1; x_2 and x_3. **(d)** - $S_{\mathrm{x},\mathbf{Y}}$ for $\mathrm{x}_1|\mathrm{x}_2 > t$; $\mathrm{x}_2|\mathrm{x}_2 > t$ and $\mathrm{x}_3|\mathrm{x}_2 > t$.

Figure 6a compares the histograms of x_3 and $\mathrm{x}_3|\mathrm{z} = 1$. Figure 6b compares histograms of $\mathrm{x}_3|\mathrm{x}_2 > t$ and of $\mathrm{x}_3|\mathrm{x}_2 > t, \mathrm{z}$. This shows that the distribution of $\mathrm{x}_3|\mathrm{z} = 1$ is different if we choose to consider artificial values of x_3 or values of x_3 that are actually used by f ($\mathrm{x}_3|\mathrm{x}_2 > t$). Figures 6c and 6d show that relative values of $S_{\mathrm{x}_1,\mathbf{Y}}$ and $S_{\mathrm{x}_3,\mathbf{Y}}$ are quite different whether we chose to consider $\mathrm{x}_2 > t$ 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 $S_{\mathrm{x}_1,\mathbf{Y}}$ and $S_{\mathrm{x}_3,\mathbf{Y}}$ 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 $S_{\mathrm{x}_3,\mathbf{Y}}$ can be, depending on the amount of artificial points. This experiment emphasizes 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.

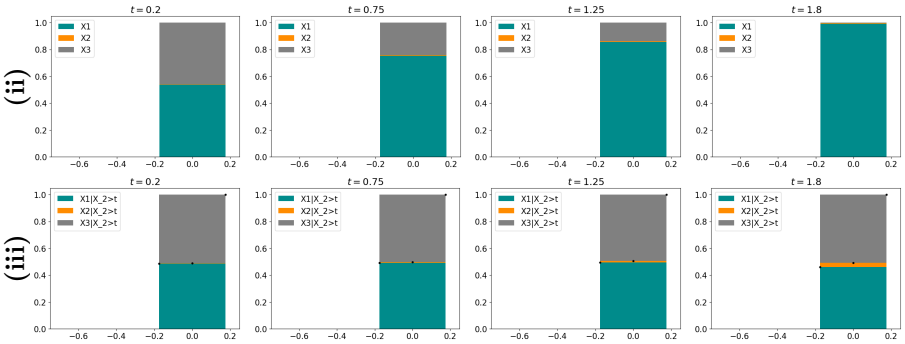
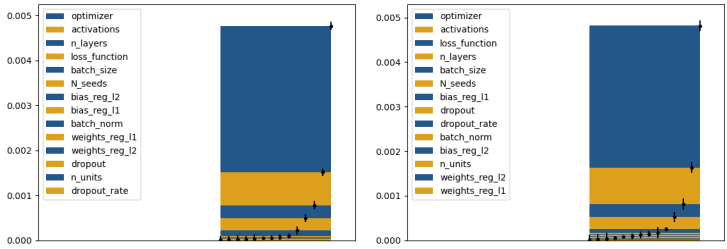


Figure 7: **Top (ii):** $S_{\mathrm{x},\mathbf{Y}}$ for x_1, x_2 and x_3 for different values of t. **Bottom (iii):** $S_{\mathrm{x},\mathbf{Y}}$ for $\mathrm{x}_1|\mathrm{x}_2 > t$, $\mathrm{x}_2|\mathrm{x}_2 > t$ and $\mathrm{x}_3|\mathrm{x}_2 > t$ for different values of t.

To sum up, this formulation brings significant implementation advantages because it allows computing $S_{\mathrm{x}_i,\mathbf{Y}}$ as if there were no conditionality. However, it carries a risk to miss essential 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 $\mathcal{G}_{\mathrm{x}_k} = \{\mathrm{x}_i|\mathrm{x}_k, i \in \mathcal{J}_k\}$, the set of hyperparameters involved jointly in hyperparameter 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 $\mathcal{G}_0 = \{\mathrm{x}_1, ..., \mathrm{x}_{n_h}\} \setminus$ `dropout_rate` and $\mathcal{G}_{\texttt{dropout\_rate}} = \{\mathrm{x}_1|\texttt{dropout\_rate}, ..., \mathrm{x}_{n_h}|\texttt{dropout\_rate}\}$ $= \{\mathrm{x}_1|\texttt{dropout} = \texttt{true}, ..., \mathrm{x}_{n_h}|\texttt{dropout} = \texttt{true}\}$. It is then possible to compute $S_{\mathrm{x}_i,\mathbf{Y}}$ for $\mathrm{x}_i \in \mathcal{G}_0$, identify the most impactful main hyperparameters, then to compute $S_{\mathrm{x}_i,\mathbf{Y}}$ for $\mathrm{x}_i \in \mathcal{G}_{\texttt{dropout\_rate}}$ 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 $S_{\mathrm{x}_i,\mathbf{Y}}$ only for x_1, x_2 and x_3 when $\mathrm{x}_2 > t$. This set would be $\mathcal{G}_{\mathrm{x}_3}$ (except that x_2 is not categorical nor integer - but in that case we can consider $\bar{X}_2 = \mathbf{1}(\mathrm{x}_2 > t)$). On the bottom row of Figure 7, $S_{\mathrm{x}_1|\mathrm{x}_2>t,\mathbf{Y}}$ and $S_{\mathrm{x}_3|\mathrm{x}_2>t,\mathbf{Y}}$ keep approximately the same values for all t, which is the correct conclusion since when x_3 is involved (i.e. $\mathrm{x}_2 > t$) , it is as important as x_1 for reaching $\mathbf{Y}$. Coming back to Runge, Figure 8 displays $S_{\mathrm{x}_i,\mathbf{Y}}$ for Runge approximation for $\mathrm{x}_i \in \mathcal{G}_{\texttt{dropout\_rate}}$, 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.

In Runge example, we have only considered one conditional hyperparameter, which is `dropout_rate`, leading to only two groups $\mathcal{G}_0$ and $\mathcal{G}_{\texttt{dropout\_rate}}$. For another, more complex example, we could introduce additional conditional hyperparameters such as SGD's `momentum`. In that case, there would



(a) (b)

Figure 8: Comparison of $S_{\mathrm{x}_i,\mathbf{Y}}$ for variables $\mathrm{x}_i \in \mathcal{G}_0$ (a) and for variables $\mathrm{x}_i \in \mathcal{G}_{\texttt{dropout\_rate}}$ (b)

be two additional groups. 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 $\mathcal{G}_{\texttt{momentum}}$.

4.4 Summary: evaluation of HSIC in hyperparameter analysis

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

Comments on Algorithm 1. Line 1: one can choose any initial distribution for hyperparameters. **Line 2:** this step is a classical random search. Recall 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 $\boldsymbol{\sigma} = (\mathrm{x}_1, ..., \mathrm{x}_{n_h}) \in \mathcal{H}$. **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, we assess the conditional ones.

Algorithm 1 Evaluation of HSIC in hyperparameter analysis

Inputs: hyperparameter search space $\mathcal{H} = \mathcal{X}_1 \times ... \times \mathcal{X}_{n_h}$, n_s.
Sample n_s hyperparameter configurations $\{\boldsymbol{\sigma}_1, ..., \boldsymbol{\sigma}_{n_s}\}$.
Train a neural network for each configuration and gather outputs $\{Y_1, ..., Y_{n_s}\}$.
Define $\mathbf{Y}$.
Construct conditional 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 4.1.
    Compute $S_{\mathrm{x}_i,\mathbf{Y}} := S_{\mathrm{u}_i,\mathbf{Y}}$ using equation (9).
    By comparing them, select the most impactful hyperparameters.
    Check for interacting hyperparameters.
end for
Outputs: Most impactful hyperparameters and interacting hyperparameters.

Remark 1 The value of $S_{\mathrm{x}_i,\mathbf{Y}}$ strongly depends on the initial distribution chosen for x_i. Indeed, if the distribution only spans values of x_i that yield good prediction error, $S_{\mathrm{x}_i,\mathbf{Y}}$ will be low. Conversely, if it spans good values but also includes absurd values, $S_{\mathrm{x}_i,\mathbf{Y}}$ will be higher. Hence, without *a priori* knowledge, we recommend to select a large range of values for each x_i

5 Experiments

Now that we can compute and correctly assess HSIC, we introduce possible usages of this metric in the context of hyperparameter analysis. In this section, we explore three benefits that we can draw from HSIC based hyperparameter analysis.

- Interpretability: HSIC allows analyzing hyperparameters, obtaining knowledge about their relative impact on error.
- Stability: Some hyperparameter configurations can lead to dramatically high errors. A hyperparameters range reduction based on HSIC can prevent such situations.
- Acceleration: We can choose values for less important hyperparameters that improve inference and training time.

We illustrate these points through hyperparameter analysis when training a fully connected neural network on MNIST and a convolutional neural network on Cifar10. We also study the approximation by a fully connected neural network of Bateman equations solution. Details about the construction of Bateman equations data set can be found in **Appendix C** and hyperparameter spaces and conditional groups $\mathcal{G}_0, ...$ for each problem in **Appendix A**.

5.1 Hyperparameter analysis

This section presents a first analysis of the estimated value of HSIC for the three benchmark data sets: MNIST, Cifar10, and Bateman equations. These evaluations are based on an initial random search for $n_s = 1000$ different hyperparameter configurations. The set $\mathbf{Y}$ is the 10%-best errors percentile, so n_s is taken sufficiently large for HSIC to be correctly estimated. Indeed, if $n_s = 1000$, there will be 100 samples of $\mathrm{u}_i|\mathrm{z} = 1$. For every data set, we extract 10% of the training data to construct a validation set to evaluate z. We keep a test set for evaluating neural networks obtained after hyperparameter optimization described in Section 6. This random search was conducted using 100 parallel jobs on CPU nodes for fully connected neural networks and 24 parallel jobs on Nvidia Tesla V100 and P100 GPUs for convolutional neural networks, so the results for these configurations were obtained quite quickly, in less than two days.

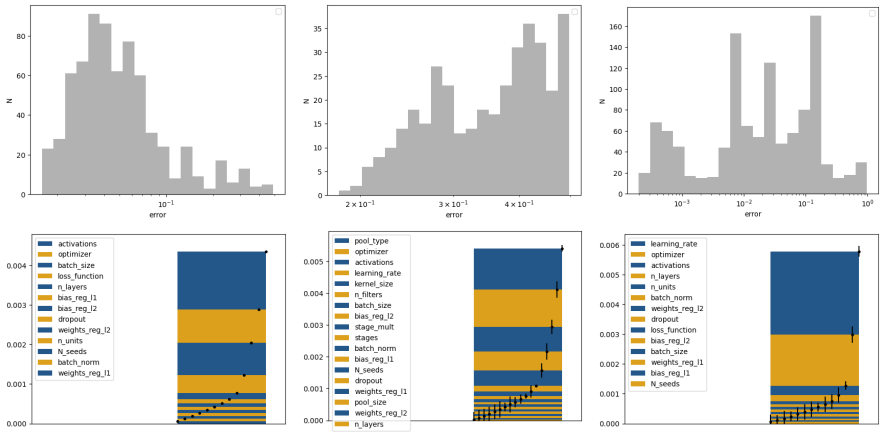
Note that for each data set, graphical comparison of $S_{\mathrm{x}_i,\mathbf{Y}}$ for conditional groups $\mathcal{G}_0, \ldots$ is displayed in **Appendix B**, for conciseness and clarity.

5.1.1 MNIST

We train $n_s = 1000$ different neural networks. 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 neural network on MNIST. Figure 9a also displays the values of $S_{\mathrm{x}_i,\mathbf{Y}}$ for each hyperparameter x_i stacked vertically. Here, `activation`, `optimizer`, `batch_size` and `loss_function` have significantly high $S_{\mathrm{x}_i,\mathbf{Y}}$. 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.

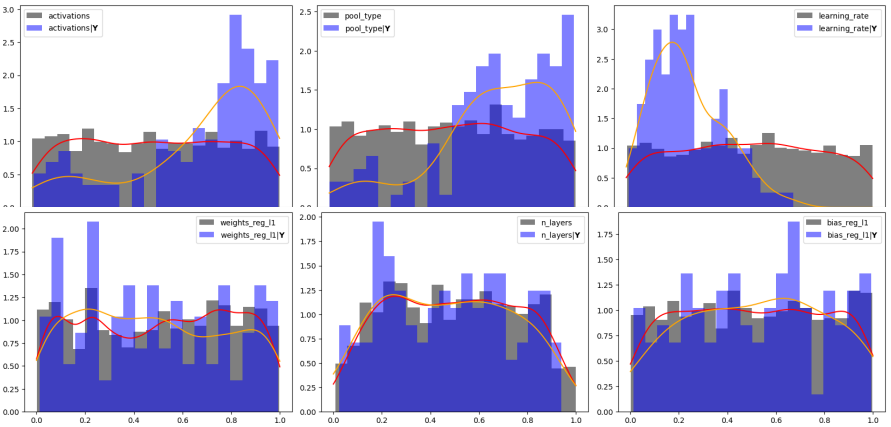
Interestingly, neither the depth (`n_layers`) nor the width (`n_units`) are among the most important hyperparameters. Notice that the random search yields a neural network of depth 4 and width 340 which obtained 98.70% accuracy, while the best networks (there were two) obtained 98.82% accuracy for a depth of 10 and a width of 791 and 1403, respectively. Recall that the min-max depth was 1-10 and width was 134-1500. It means that lighter networks are capable of obtaining competitive accuracy. Another interesting observation is that `loss_function` does not have the highest HSIC, meaning that Mean Squared Error allows obtaining good test errors, which is surprising for a classification problem.

We plot histograms of u_i and $\mathrm{u}_i|\mathrm{z} = 1$ on Figure 10a for `activation` (top) and `weights_reg_l1` (bottom) with repeated sampling for categorical hyperparameters, like in Section 4.1. Note that the first and the second hyperparameters have respectively a high and low $S_{\mathrm{x}_i,\mathbf{Y}}$. We can see that for hyperparameters with high $S_{\mathrm{x}_i,\mathbf{Y}}$, $\mathrm{u}_i|\mathrm{z} = 1$ (orange for KDE, blue for histogram) is quite different from u_i (red for KDE, gray for histogram). On the contrary, for hyperparameters with low $S_{\mathrm{x}_i,\mathbf{Y}}$ there not seems to have major differences.



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

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



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

Figure 10: Representation of $\mathrm{u}_i|\mathrm{z}=1$ (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) $S_{\mathbf{x}_i,\mathbf{Y}}$

5.1.2 Cifar10

We train $n_s = 1000$ different convolutional neural networks. After the initial random search, the best validation error is 81.37%. Note that the histogram of Figure 9b is truncated because many hyperparameter configurations led to diverging errors. Here, `pool_type`, `optimizer`, `activation`, `learning_rate` and `kernel_size` have the highest $S_{\mathbf{x}_i,\mathbf{Y}}$, followed by `n_filters`. Half of these hyperparameters are specific to convolutional neural networks, 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 $S_{\mathbf{x}_i,\mathbf{Y}}$ comparisons for every group 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.

The depth (`n_layers`) is the less important hyperparameters. Here, the random search yields a neural network of depth 4 and width 53, with 3 stages (meaning that the neural network is widened 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 as for MNIST: increasing the size of the network is not the only efficient way to improve its accuracy.

We plot histograms of u_i and $\mathrm{u}_i|\mathrm{z}=1$ 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. An explanation could be that configurations lead to out-of-memory errors or are so long to train that 1000 other neural networks with different configurations have 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 $\mathrm{u}_i|\mathrm{z}=1$.

5.1.3 Bateman equations

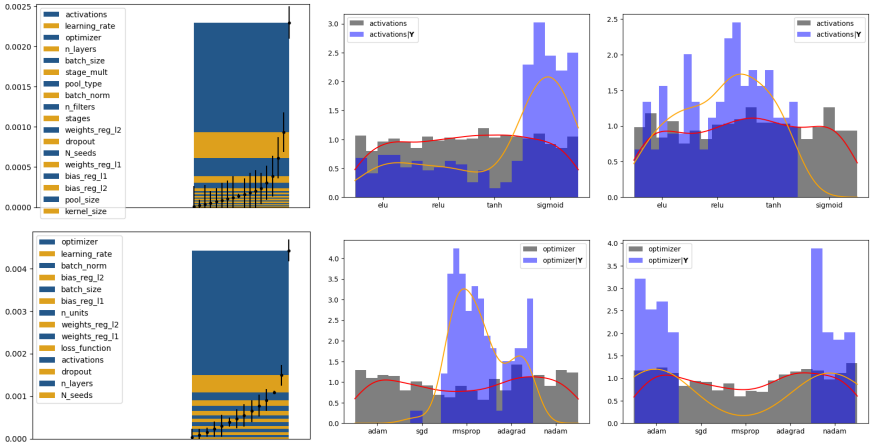
For Bateman equations, mean squared error goes down to 2.90×10^{-5}. Like for Cifar10, the histogram of Figure 9b is truncated because many hyperparameter configurations led to diverging errors. For this problem, `learning_rate`, `optimizer`, `activations` and `n_layer` can be considered as impactful. Conditional groups are also listed in **Appendix A**. Three conditional hyperparameters are important: `beta_2`, the second 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 $S_{\mathbf{x}_i,\mathbf{Y}}$ and `n_units` belongs to less impactful hyperparameters. We perform the same analysis as for MNIST and Cifar10 and quote that the best neural network has depth 5 and width 470 while another neural network of depth 5 and width 62 reaches 3.74×10^{-5} validation error.

We plot histograms of u_i and $\mathrm{u}_i|\mathrm{z}=1$ 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 $\mathrm{u}_i|\mathrm{z}=1$ seems more natural. This once again illustrates the differences of u_i and $\mathrm{u}_i|\mathrm{z}=1$ for hyperparameters with high and low $S_{\mathbf{x}_i,\mathbf{Y}}$.

5.2 Modification of hyperparameters distribution to improve training 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. However, HSIC formalism and our adaptation to hyperparameter analysis allow us to choose any $\mathbf{Y}$. In the previous section, for Cifar10 and Bateman, we truncated histograms of Figure 9b because many hyperparameter configurations led to diverging errors. It is possible to understand why by choosing $\mathbf{Y}$ as the set of the 10% worst errors. Then, HSIC can be applied to assess the importance of each hyperparameter towards the worst errors.



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

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

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

adapt 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 becomes tedious. Thanks to HSIC, we know directly which histograms to look at and how to rank hyperparameters when it is not visually clear-cut.

5.3 Interval reduction for continuous or integer hyperparameters that affect execution speed

One common conclusion of $S_{\mathrm{x}_i,\mathbf{Y}}$ values for the last three machine learning 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. It naturally raises the question of how to bias the hyperparameter optimization towards such models. Multi-objective hyperparameter optimization algorithms have already been successfully applied, like in [19] 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 $S_{\mathrm{x}_i,\mathbf{Y}}$ to reduce the hyperparameter space search in order to obtain more cost-effective neural networks. The most simple way to achieve that goal is to select values that improve execution speed for hyperparameters which have low $S_{\mathrm{x}_i,\mathbf{Y}}$ 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 $S_{\mathrm{x}_i,\mathbf{Y}}$ value, we may not be able to apply the previous idea. In that case, we can use HSIC in a different way to still achieve our goal, for integer or continuous hyperparameters (such as `n_layers`, `n_units`, or `kernel_size`). Note that most of the time, for these hyperparameters, a too low or high value will increase the error or the execution speed, respectively. We would like to choose a value which is as low as possible without hurting the error too much. Suppose that x_i = `n_layers` $\in \{a, ..., b\}$ and that $S_{\mathrm{x}_i,\mathbf{Y}}$ is high, so that `n_layers` is among the most important hyperparameters. It is likely that $S_{\mathrm{x}_i,\mathbf{Y}}$ is high because a is too small. One could therefore compute $S_{\mathrm{x}_i|\mathrm{x}_i\in\{a+c,...,b\},\mathbf{Y}}$ for $c \in \{1, ..., b-a\}$, starting with $c = 1$ until $S_{\mathrm{x}_i|\mathrm{x}_i\in\{a+c,...,b\},\mathbf{Y}}$ becomes low. Then, hyperparameter `n_layers` can be replaced by `n_layers`$|$`n_layers` $\in \{a+c, ..., b\}$, which has a low HSIC, and whose value can hence be set to $a + c$.

To illustrate this, let us come back to Runge data set. We first focus on this example because we have been able to train $n_s = 10000$ different neural networks so the methodology can be tested with limited noise. In Figure 12, $S_{\mathrm{x}_i|\mathrm{x}_i\in\{a+c,...,b\},\mathbf{Y}}$ is plotted with respect to c, where x_i = `n_layers`. We see that $S_{\mathrm{x}_i|\mathrm{x}_i\in\{a+c,...,b\},\mathbf{Y}}$ decreases until `n_layers` = 3, after which the tendency is not statistically significant. Choosing `n_layers` = 3 makes `n_layers` belong to

the less important hyperparameters so it is a good trade-off value for execution speed and accuracy.

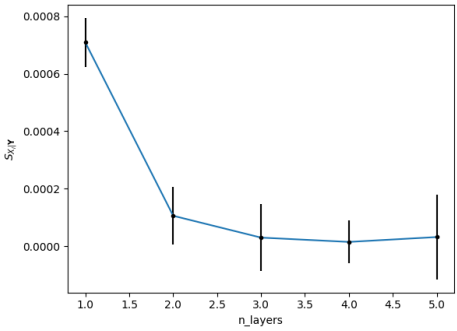


Figure 12: $S_{\mathrm{x}_i|\mathrm{x}_i\in\{a+c,\ldots,b\},\mathbf{Y}}$ w.r.t. c for `n_layers` in Runge. The error bars traduce the standard estimation error.

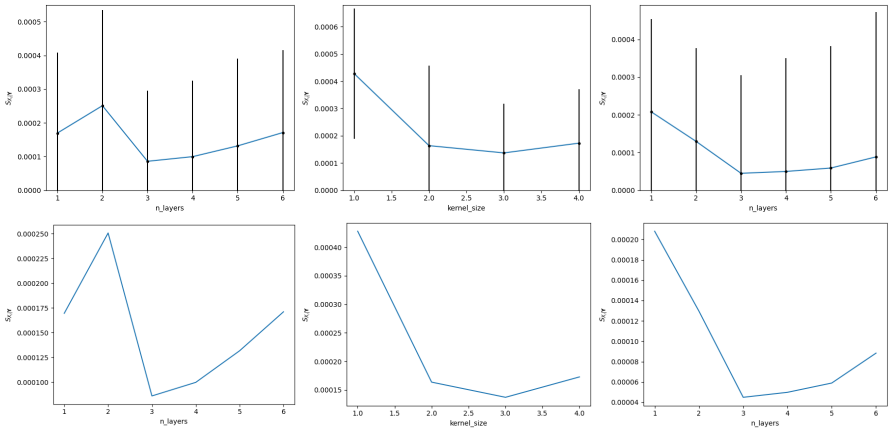
We apply this methodology to MNIST, Cifar10, and Bateman problems in Figure 13. When plotting these curves, too high values of c have to be discarded since the more c increases, the less points there are to compute $S_{\mathrm{x}_i|\mathrm{x}_i\in\{a+c,\ldots,b\},\mathbf{Y}}$. It could explain the strange behavior of the plots at the right of the axis of Figure 13, and the widening of error bars for `n_layers` $= 5$ in Figure 12.

Note that in Figure 13, error bars are much larger than in Figure 12. Indeed, in these cases, $S_{\mathrm{x}_i|\mathrm{x}_i\in\{a+c,\ldots,b\},\mathbf{Y}}$ are evaluated with 10 times less points. Hence, one must be careful with their interpretation. First, we are far from the asymptotical regime under which estimation error is gaussian for so few estimation points. It explains why error bars can go below 0 whereas the value to estimate is a distance. Therefore, these bars only indicate how spread the error is. Second, since it turns out that the error is very spread, the trade-off value must be chosen with caution by taking this statistic into account. In this manuscript, we rely on a human eye to qualitatively chose this value, but in future work, we should study the use of statistical tests.

Finally, these plots suggests that we could set `n_layers` $= 3$ for MNIST, `kernel_size` $= 3$ for Cifar10 and `n_layers` $= 3$ for Bateman without affecting the error too much. Once the hyperparameter space has been reduced to improve neural networks execution time, it is possible to apply any classical hyperparameter optimization algorithm.

6 Optimization by focusing on impactful hyperparameters

One of the most successful and widely used hyperparameter optimization algorithms is Gaussian Processes-based Bayesian Optimization, which we denote



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

Figure 13: $S_{\mathbf{x}_i|\mathbf{x}_i\in\{a+c,\ldots,b\},\mathbf{Y}}$ w.r.t. c for (a) `n_layers` in MNIST, (b) `kernel_size` in Cifar10 and (c) `n_layers` in Bateman with error bars (**top**) and without error bars (**bottom**). The error bars traduce the standard estimation error.

GPBO by convenience. 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.

In [34], the authors introduce the use of HSIC for feature selection and in [30], HSIC based feature selection is used in the context of optimization. The idea is to compute $S_{\mathbf{x}_i,\mathbf{Y}}$ for each variable involved in the optimization and to discard low $S_{\mathbf{x}_i,\mathbf{Y}}$ variables from it. More specifically, we fix the discarded variables to an arbitrary value, and then the optimization algorithm is applied only in the dimension of the high $S_{\mathbf{x}_i,\mathbf{Y}}$ variables.

This methodology is particularly suited to hyperparameter optimization. In this work, we have emphasized the ability of HSIC to identify the most important hyperparameters. It allows performing relevant HSIC driven hyperparameter selection, which can overcome optimization in too high dimensional hyperparameter spaces. We go further and present a two-step optimization. We optimize the 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 hyperparameter space, which has reasonable dimension. It allows applying GPBO despite the initially large dimension of the hyperparameter space. At the end of this step, optimal values are selected for the most impactful hyperparameters.
2 Optimization on the remaining dimensions. In our case, GPBO can be reasonably applied in this space, but note that we might have hyperparameter spaces whose initial dimension is so high that after the first step, the remaining

dimensions to optimize could still be too numerous to perform GPBO. In that case, less refined but more robust hyperparameter optimization algorithms (like random search or Tree Parzen Estimators [13]) 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 [30], the authors choose the values yielding the best output after the initial random search. Here, the value selection method that aims at improving execution speed, introduced in Section 5.3, integrates perfectly with this two-step optimization. Following this method brings two advantages. First, we can obtain more cost-effective neural networks if we keep these values through the two optimization step. Second, if we do not care so much about execution speed but only look for accuracy, still fixing these values during the first optimization step improves the training speed and thereby global hyperparameter optimization time.

The rest of the low $S_{\mathrm{x}_i,\mathbf{Y}}$ hyperparameters value can be set as those of the hyperparameter configuration yielding the best error. There is one last attention point: one has to be careful about interactions between low $S_{\mathrm{x}_i,\mathbf{Y}}$ hyperparameters. If two low HSIC hyperparameters x_i and x_j are found to interact, like discussed in section 4.2, and x_i has an impact on execution speed, the value of x_j must be chosen so that value of the pair $(\mathrm{x}_i, \mathrm{x}_j)$ is close to the value of the hyperparameter configuration of a low error neural network. The two-step optimization is summarized in Algorithm 2.

Algorithm 2 Two-step Optimization

Inputs: hyperparameter search space $\mathcal{H} = \mathcal{X}_1 \times ... \times \mathcal{X}_{n_h}$, n_s
Apply Algorithm 1: "Evaluation of HSIC in hyperparameter analysis".
Perform interval reduction (for cost efficiency and stability), as in Sections 5.3 and 5.2.
Select values for less impactful hyperparameters that improve execution speed, taking care of interaction, like discussed in Section 4.2.
// `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.
end if

We evaluate this two-step optimization on our three data sets. For each of these, we consider 4 baselines. For each of these baselines, we report the *test* error (the metric is accuracy for MNIST and Cifar10 and MSE for Bateman), the number of parameters of the best models, and their FLOPs.

- Random search: The result of the random search of 1000 configurations plus 200 additional configurations for a total of $n_s = 1200$ points.
- Full GPBO: Gaussian Processes-based Bayesian Optimization, conducted on the full hyperparameter space, without any analysis based on HSIC. We initialize the optimization with 50 random configurations and perform the optimization for 50 iterations (enough to reach convergence).
- TS-GPBO (acc): Two-Step GPBO described in Algorithm 2, with goal = accuracy. HSIC are estimated using a first random search of $n_s = 1000$ points. Steps 1 and 2 are run for 25 iterations.
- TS-GPBO (acc + speed): Two-Step GPBO described in Algorithm 2, with goal = accuracy and execution speed.

Random search (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 random search 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 the same baseline, for cluster accessibility reasons.

We chose the number of total model evaluations for each baseline to obtain approximately the same total execution time. The differences between the number of evaluations, despite identical total execution time, can be explained by different factors. First, the random search can be fully executed in parallel, while GPBO is sequential. Second, step 1 of TS-GPBO always chooses values for non-optimized hyperparameters that improve execution speed and training time. As a result, step 1 is quite fast. Besides, experiments show that step 2 usually converges faster, in terms of the 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 random search) of each baseline can be found in Table 4.

Results show that except for Cifar10, TS+GPBO yields very competitive neural networks while having far fewer parameters and FLOPs. For MNIST, TS+GPBO model has ≈ 66 and 41 times fewer parameters and FLOPs than full GPBO and random search. For Bateman, these factors are 482 and 380. An oversized initial hyperparameter search space could explain such a high factor. Still, a reasonable size for the search space cannot be found *a priori*, and our method makes hyperparameter optimization 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 were already satisfying, and TS-GPBO (accuracy) did not bring significant improvement. For the particular case of Cifar10, TS-GPBO (accuracy) and (accuracy + speed) both find a

data set	baseline	test metric	params	MFLOPs
MNIST	RS	98.36	6,267,103	12,709 (×41)
-	full GPBO	**98.42** ± 0.05	10,271,367	20,534 (×67)
-	TS-GPBO (acc + speed)	**98.42** ± 0.02	**151,306**	**307** (×1)
Cifar10	RS	81.8	99,444,880	1,832,615 (×11)
-	full GPBO	**82.73** ± 1.45	71,111,761	1,441,230 (×8)
-	TS-GPBO (acc)	**82.60** ± 0.58	9,604,539	650,269 (×4)
-	TS-GPBO (acc + speed)	79.34 ± 0.15	**9,281,258**	**178,621** (×1)
Bateman	RS	**1.99** $\times 10^{-4}$	1,259,140	2,516 (×359)
-	full GPBO	2.94 ± 0.42 $\times 10^{-4}$	1,588,215	3,173 (×453)
-	TS-GPBO (acc + speed)	3.49 ± 0.31 $\times 10^{-4}$	**3,291**	**7** (×1)

Table 4: Results of hyperparameter optimization for Random Search (RS), Gaussian Processes based Bayesian Optimization on the full hyperparameter space (full GPBO) and Two-Steps Gaussian Processes based Bayesian Optimization (TS-GPBO). The mean ± 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.

model which has 11 and 9 times fewer parameters than random search and full GPBO. TS-GPBO (accuracy) finds a model with ≈ 3 and 2 fewer FLOPs than random search 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 results have been allowed thanks to information given by HSIC analysis. Hence, TS-GPBO outputs competitive and cost-effective models but also grants a better knowledge of hyperparameters interaction in these machine learning problems, as opposed to random search and full GPBO.

7 Discussion and Perspectives

Many techniques have already been introduced to handle hyperparameter optimization, but they often suffer from a lack of interpretability and interactivity. In this work, we tackled these problems by proposing an HSIC based goal-oriented global sensitivity analysis applied to hyperparameter search spaces. We showcased how we can use this information by improving the stability of training instances and the cost efficiency of trained networks. We also introduced an interpretable hyperparameter optimization methodology that yields competitive and cost-effective neural networks based on feature selection.

7.1 Impact for scientific machine learning

These findings are of interest to the machine learning community. Though the presented methodologies can be taken as contributions by themselves, they should also be understood as demonstrations that HSIC based goal-oriented global sensitivity analysis is interesting and valuable for hyperparameter optimization. In the end, an important outcome of this work was to make an insightful tool, HSIC, ready for use in hyperparameter optimization.

This work also impacts scientific computing since it tackles the trade-off between accuracy and cost-efficiency of neural networks. Indeed, we obtained lighter networks without significantly affecting the error, which is the ideal goal for high-performance computing.

7.2 Other comments

Other points can be made regarding the presented results and the potential follow-up work. They can be grouped into the following topics:

Hyperparameters modeling choice. HSIC is a powerful tool that is widely used for sensitivity analysis as a dependence measure. Its application to hyperparameter optimization required some work, especially regarding the complex structure of hyperparameter space. To achieve this goal, we made some modeling choices, such as applying Φ_{x_i} to map hyperparameter x_i to a uniform random variable. The good results obtained in Section 6 validate not only the usage of information given by HSIC for hyperparameter analysis but also this modeling 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 has to actively decide which hyperparameter deserves to be considered as more or less impactful. Nevertheless, one advantage of HSIC is that it is a scalar metric. One could construct an HSIC based hyperparameter optimization by setting a threshold above which hyperparameters are considered impactful. It would lead to an end-to-end automatic yet interpretable hyperparameter optimization algorithm. Though [30] use the idea of a threshold, its application to hyperparameter optimization has not been studied in this paper and could be part of future works.

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

Hyperparameter optimization speed up. We presented some ways of using HSIC in hyperparameter optimization, but this paper mainly emphasized

the possibility of exploiting it in order to find lighter models. We are aware that execution speed is not always a goal for machine learning practitioners. Still, machine learning practitioners are always concerned about training speed. The first step of 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. It would even be possible to go further and to apply parallel GPBO like described in [12], or to use Hyperband on the initial random search since HSIC computation only relies on the error of the p-% best neural networks.

Further execution time improvement. One advantage of execution time improvement obtained thanks to HSIC is that it only relies on choices for the conception of the neural network. Therefore, additional improvements could be made by applying other techniques like quantization, weights pruning, or multi-objective hyperparameter optimization.

8 Conclusion

Hyperparameter optimization is a very important step of machine learning applications, ordinarily conducted in a black-box fashion. Using an approach based on goal-oriented global sensitivity analysis, we show that we can make hyperparameter optimization more interpretable. In particular, we adapt Hilbert Schmidt Independence Criterion, a statistical dependence measure used in sensitivity analysis, to hyperparameter spaces that can be complex and awkward due to the different nature of hyperparameters (continuous or categorical) and their interactions and inter-dependencies. Its use for hyperparameter analysis is demonstrated on various case studies. In particular, it allows constructing an original and interpretable two-step hyperparameter optimization methodology based on feature selection that improves neural networks' execution speed as well as test error.

References

[1] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning, (2016). http://www.deeplearningbook.org

[2] Gretton, A., Bousquet, O., Smola, A., Schölkopf, B.: Measuring statistical dependence with hilbert-schmidt norms. In: Proceedings of the 16th International Conference on Algorithmic Learning Theory. ALT'05, pp. 63–77. Springer, Berlin, Heidelberg (2005). https://doi.org/10.1007/11564089_7. https://doi.org/10.1007/11564089_7

[3] Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B., Smola, A.J.: A kernel method for the two-sample-problem. In: Schölkopf, B., Platt, J.C., Hoffman, T. (eds.) Advances in Neural Information Processing Systems 19, (2007). http://papers.nips.cc/paper/3110-a-kernel-method-for-the-two-sample-problem.pdf

[4] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 448–456. PMLR, Lille, France (2015). https://proceedings.mlr.press/v37/ioffe15.html

[5] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (Poster) (2015). http://arxiv.org/abs/1412.6980

[6] Tan, M., Le, Q.: EfficientNet: Rethinking model scaling for convolutional neural networks. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 6105–6114. PMLR, Long Beach, California, USA (2019). http://proceedings.mlr.press/v97/tan19a.html

[7] Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research **13**(10), 281–305 (2012)

[8] Jamieson, K., Talwalkar, A.: Non-stochastic best arm identification and hyperparameter optimization. In: Gretton, A., Robert, C.C. (eds.) Proceedings of the 19th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 51, pp. 240–248. PMLR, Cadiz, Spain (2016). http://proceedings.mlr.press/v51/jamieson16.html

[9] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research **18**(185), 1–52 (2018)

[10] Mockus, J.: On bayesian methods for seeking the extremum. In: Proceedings of the IFIP Technical Conference, pp. 400–404. Springer, Berlin, Heidelberg (1974)

[11] Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., de Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE **104**, 148–175 (2016)

[12] Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2. NIPS'12, pp. 2951–2959. Curran Associates Inc., Red Hook, NY, USA (2012)

[13] Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R.S., Bartlett, P.L., Pereira, F., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 24, (2011). http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

[14] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., Adams, R.: Scalable bayesian optimization using deep neural networks. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 2171–2180. PMLR, Lille, France (2015). http://proceedings.mlr.press/v37/snoek15.html

[15] Chollet, F.: Xception: Deep learning with depthwise separable convolutions. CoRR **abs/1610.02357** (2016) https://arxiv.org/abs/1610.02357

[16] Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002). https://doi.org/10.1162/106365602320169811

[17] Kandasamy, K., Neiswanger, W., Schneider, J., Póczos, B., Xing, E.P.: Neural architecture search with bayesian optimisation and optimal transport. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. NIPS'18, pp. 2020–2029. Curran Associates Inc., Red Hook, NY, USA (2018)

[18] Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. Proceedings of Machine Learning Research, vol. 80, pp. 4095–4104. PMLR, Stockholmsmässan, Stockholm Sweden (2018). http://proceedings.mlr.press/v80/pham18a.html

[19] Tan, M., Chen, B., Pang, R., Vasudevan, V., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. CoRR **abs/1807.11626** (2018) https://arxiv.org/abs/1807.11626

[20] Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. Journal of Machine Learning Research **20**(55), 1–21 (2019)

[21] Razavi, S., Jakeman, A., Saltelli, A., Prieur, C., Iooss, B., Borgonovo, E., Plischke, E., Lo Piano, S., Iwanaga, T., Becker, W., Tarantola, S., Guillaume, J.H.A., Jakeman, J., Gupta, H., Melillo, N., Rabitti, G., Chabridon, V., Duan, Q., Sun, X., Smith, S., Sheikholeslami, R., Hosseini, N., Asadzadeh, M., Puy, A., Kucherenko, S., Maier, H.R.: The future of sensitivity analysis: An essential discipline for systems modeling and policy support. Environmental Modelling & Software **137**, 104954 (2021). https://doi.org/10.1016/j.envsoft.2020.104954

[22] Sobol, I.M.: Sensitivity estimates for nonlinear mathematical models. MMCE (1), 407–414 (1993)

[23] Fort, J.-C., Klein, T., Rachdi, N.: New sensitivity analysis subordinated to a contrast. Communications in Statistics - Theory and Methods **45**(15), 4349–4364 (2016) https://arxiv.org/abs/https://doi.org/10.1080/03610926.2014.901369. https://doi.org/10.1080/03610926.2014.901369

[24] Borgonovo, E.: A new uncertainty importance measure. Reliability Engineering & System Safety **92**(6), 771–784 (2007). https://doi.org/10.1016/j.ress.2006.04.015

[25] Saltelli, A.: Making best use of model evaluations to compute sensitivity indices. Computer Physics Communications **145**(2), 280–297 (2002). https://doi.org/10.1016/S0010-4655(02)00280-1

[26] Da Veiga, S.: Global sensitivity analysis with dependence measures. Journal of Statistical Computation and Simulation **85** (2013). https://doi.org/10.1080/00949655.2014.945932

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

[28] Müller, A.: Integral probability metrics and their generating classes of functions. Advances in Applied Probability **29**(2), 429–443 (1997). https://doi.org/10.2307/1428011

[29] Fukumizu, K., Gretton, A., Lanckriet, G.R., Schölkopf, B., Sriperumbudur, B.K.: Kernel choice and classifiability for rkhs embeddings of probability distributions. In: Bengio, Y., Schuurmans, D., Lafferty, J.D., Williams, C.K.I., Culotta, A. (eds.) Advances in Neural Information Processing Systems 22, (2009). http://papers.nips.cc/paper/3750-kernel-choice-and-classifiability-for-rkhs-embeddings-of-probability-distributions.pdf

[30] Spagnol, A., Riche, R.L., Da Veiga, S.: Global sensitivity analysis for optimization with variable selection. SIAM/ASA J. Uncertain. Quantification **7**, 417–443 (2018)

[31] Kruskal, J.B.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika **29**(1), 1–27 (1964). https://doi.org/10.1007/BF02289565. Accessed 2021-10-12

[32] Gillespie, D.T.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. Journal of Computational Physics **22**(4), 403–434 (1976). https://doi.org/10.1016/0021-9991(76)90041-3

[33] Falkner, S., Klein, A., Hutter, F.: BOHB: Robust and efficient hyperparameter optimization at scale. Proceedings of Machine Learning Research, vol. 80, pp. 1437–1446. PMLR, Stockholmsmässan, Stockholm Sweden (2018). http://proceedings.mlr.press/v80/falkner18a.html

[34] Song, L., Smola, A., Gretton, A., Borgwardt, K.M., Bedo, J.: Supervised feature selection via dependence estimation. In: Proceedings of the 24th International Conference on Machine Learning. ICML '07, pp. 823–830. Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1273496.1273600. https://doi.org/10.1145/1273496.1273600

Statements & Declarations

Ethics approval and Consent to participate :

All authors approve the Committee on Publication Ethics guidelines. They consent to participate.

Consent for publication :

All authors give explicit consent to submit and they obtained consent from the responsible authorities at the institutes where the work has been carried out, the CEA, Inria and the Ecole Polytechnique.

Availability of data and materials :

The source code is available at https://github.com/paulnovello/goal-oriented-ho.

Competing interests :

All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

Fundings :

The research has been supported by the institutes where the authors are affiliated : the CEA, Inria and the Ecole Polytechnique.

Authors' contributions :

All authors whose names appear on the submission

- made substantial contributions to the conception or design of the work; or the acquisition, analysis, or interpretation of data; or the creation of new software used in the work
- drafted the work or revised it critically for important intellectual content
- approved the version to be published
- agree to be accountable for all aspects of the work in ensuring that questions related to the accuracy or integrity of any part of the work are appropriately investigated and resolved.

Appendix A Hyperparameters spaces

In this section, we describe hyperparameters spaces used for each problem in this chapter. Note that hyperparameter `n_seeds` denotes the number of random repetitions of the training for each hyperparameter 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.

A.1 Runge and MNIST

For Runge and 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 A1: Hyperparameters values for Runge & MNIST

Conditional groups: (see **(iii)** of Section 4.3) $\mathcal{G}_0$ and $\mathcal{G}_{\texttt{dropout\_rate}}$

A.2 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 A2: Hyperparameters values for Bateman

Conditional groups: (see **(iii)** of Section 4.3) $\mathcal{G}_0$, $\mathcal{G}_{\texttt{dropout\_rate}}$, $\mathcal{G}_{\texttt{amsgrad}}$, $\mathcal{G}_{\texttt{centered}}$, $\mathcal{G}_{\texttt{nesterov}}$, $\mathcal{G}_{\texttt{momentum}}$ and $\mathcal{G}_{(\texttt{1st\_moment},\texttt{2nd\_moment})}$

A.3 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 neural network 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 Cifar10
`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 A3: Hyperparameters values for Cifar10

Conditional groups: (see **(iii)** of Section 4.3) $\mathcal{G}_0$, $\mathcal{G}_{\texttt{dropout\_rate}}$, $\mathcal{G}_{\texttt{amsgrad}}$, $\mathcal{G}_{\texttt{centered}}$, $\mathcal{G}_{\texttt{nesterov}}$, $\mathcal{G}_{\texttt{momentum}}$ and $\mathcal{G}_{(\texttt{1st\_moment,2nd\_moment})}$

Appendix B HSICs for conditional hyperparameters

B.1 MNIST

For MNIST, there is only one conditional hyperparameter, `dropout_rate`, so only one conditional group to consider in order to assess the importance of conditional hyperparameters.

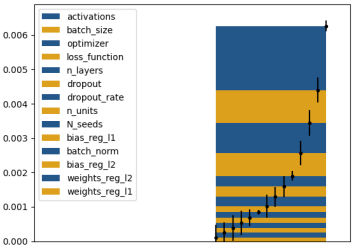
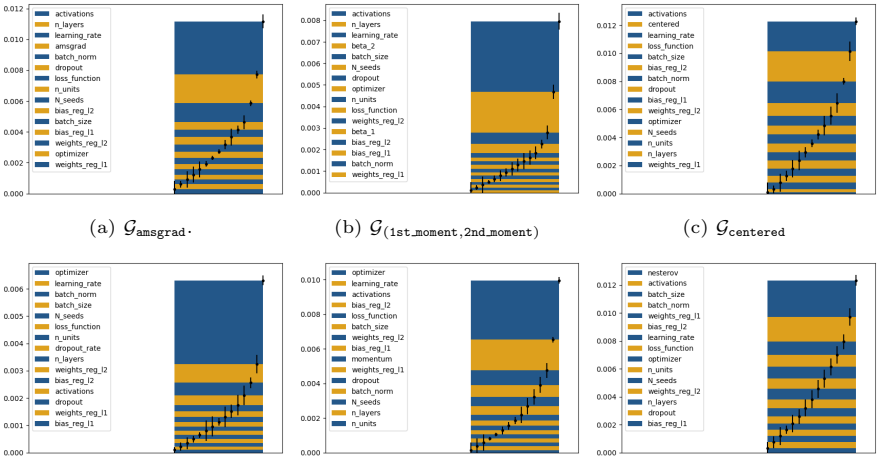


Figure B1: HSICs for $\mathcal{G}_{\texttt{dropout\_rate}}$ of MNIST hyperparameter analysis. Conditional hyperparameter `dropout_rate` is not impactful.

B.2 Bateman

For Bateman, there are seven conditional hyperparameter, `amsgrad`, `1st_moment` (`beta_1`), `2nd_moment` (`beta_2`), `dropout_rate`, `centered`, `momentum`, and `nesterov`. Six conditional groups, specified in Figure B2, have to be considered in order to assess their importance.



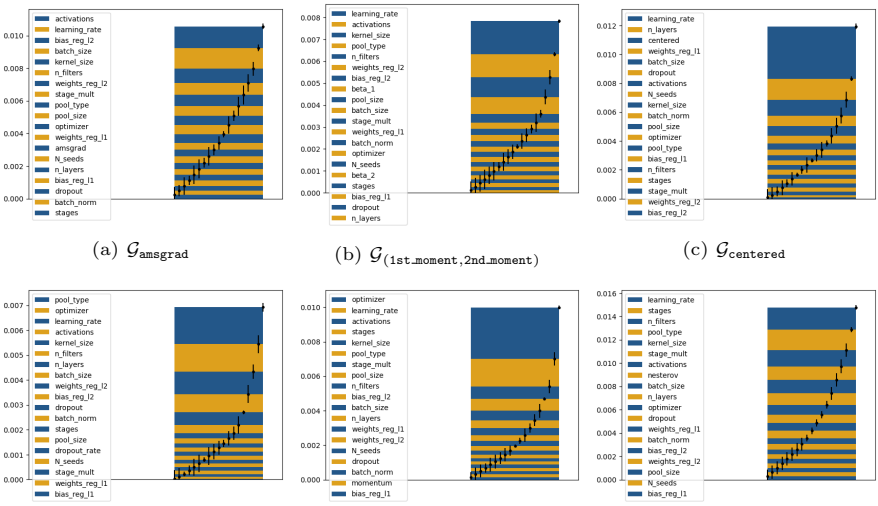
(a) $\mathcal{G}_{\texttt{amsgrad}}$. (b) $\mathcal{G}_{(\texttt{1st\_moment},\texttt{2nd\_moment})}$ (c) $\mathcal{G}_{\texttt{centered}}$

(d) $\mathcal{G}_{\texttt{dropout\_rate}}$ (e) $\mathcal{G}_{\texttt{momentum}}$ (f) $\mathcal{G}_{\texttt{nesterov}}$

Figure B2: HSICs for conditional groups of Bateman hyperparameter analysis. (a): `amsgrad` is not impactful (it is in the estimation noise), (b): `1st_moment` is not impactful but `2nd_moment` is the fourth most impactful hyperparameter of this group, (c): `centered` is the second most impactful hyperparameter of this group, (d): `dropout_rate` is not impactful, (e): `momentum` is not impactful, (f): `nesterov` is the most impactful hyperparameter of this group.

B.3 Cifar10

For Cifar10, there are seven conditional hyperparameter, `amsgrad`, `1st_moment` (`beta_1`), `2nd_moment` (`beta_2`), `dropout_rate`, `centered`, `momentum`, and `nesterov`. Six conditional groups, specified in Figure B3, have to be considered in order to assess their importance.



(a) $\mathcal{G}_{\texttt{amsgrad}}$ (b) $\mathcal{G}_{(\texttt{1st\_moment},\texttt{2nd\_moment})}$ (c) $\mathcal{G}_{\texttt{centered}}$

(d) $\mathcal{G}_{\texttt{dropout\_rate}}$ (e) $\mathcal{G}_{\texttt{momentum}}$ (f) $\mathcal{G}_{\texttt{nesterov}}$

Figure B3: HSICs for conditional groups of cifar10 hyperparameter analysis. (a): `amsgrad` is not impactful, (b): `1st_moment`, `2nd_moment` are not impactful, (c): `centered` is the third most impactful hyperparameter of this group, (d): `dropout_rate` is not impactful, (e): `momentum` is not impactful, (f): `nesterov` is not impactful.

Appendix C Construction of Bateman data set

Bateman data set is based on the resolution of the Bateman equations, which is an ODE system modeling 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 $\boldsymbol{\Sigma_r}\big(\boldsymbol{\eta}(t)\big)$, the species have to be considered 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, neural network's input is $(\boldsymbol{\eta}_0, t)$ and neural network's output is $f((\boldsymbol{\eta}_0, t))$.