Leveraging Local Variation in Data: Sampling and Weighting Schemes for Supervised Deep Learning

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

LEVERAGING LOCAL VARIATION IN DATA: SAMPLING AND WEIGHTING SCHEMES FOR SUPERVISED DEEP LEARNING

P. Novello,[1,2,3,*] G. Poëtte,[1] D. Lugato,[1] & P.M. Congedo[2,3]

[1]*CESTA, CEA, Le Barp, 33114, France*

[2]*CMAP, Ecole Polytechnique, 91120, Palaiseau, France*

[3]*Platon, Inria Paris Saclay, 91120, Palaiseau, France*

*Address all correspondence to: P. Novello, E-mail: paul.novello@outlook.fr

In the context of supervised learning of a function by a neural network, we claim and empirically verify that the neural network yields better results when the distribution of the data set focuses on regions where the function to learn is steep. We first traduce this assumption in a mathematically workable way using Taylor expansion and emphasize a new training distribution based on the derivatives of the function to learn. Then, theoretical derivations allow constructing a methodology that we call Variance Based Samples Weighting (VBSW). VBSW uses labels local variance to weight the training points. This methodology is general, scalable, cost-effective, and significantly increases the performances of a large class of neural networks for various classification and regression tasks on image, text, and multivariate data. We highlight its benefits with experiments involving neural networks from linear models to ResNet [19] and Bert [14].

KEY WORDS: *Supervised learning, importance weighting, learning theory, designs of experiments*

When a Machine Learning (ML) model is used to learn from data, the distribution of the training data set can have a substantial impact on its performance. More specifically, in Deep Learning (DL), several works have hinted at the importance of the training set. In [6,37], the authors exploit the observation that a human will benefit more from easy examples than from harder ones at the beginning of a learning task. They construct a curriculum, inducing a change in the distribution of the training data set that makes a neural network achieve better results in an ML problem. With a different approach, Active Learning [46] modifies the distribution of the training data dynamically by selecting the data points that will make the training more efficient. Finally, in Reinforcement Learning, the distribution of experiments is crucial for the agent to learn efficiently. Moreover, the challenge of finding a good distribution is not specific to ML. Indeed, in the context of Monte Carlo estimation of a quantity of interest based on a random variable, Importance Sampling owes its efficiency to the construction of a second random variable, which is used instead to improve the estimation of this quantity. [23] even make a connection between the success of likelihood ratio policy gradients and importance sampling, which shows that ML and Monte Carlo estimation, both distribution-based methods, are closely linked.

In this paper, we leverage the importance of the training set distribution to improve the performances of neural networks in supervised deep learning. We formalize supervised learning as a task which aims at approximating a function f with a model f_θ parametrized by θ using data points drawn from $X \sim d\mathbb{P}_X$, $X \in \mathcal{X}$. We build a new distribution $d\mathbb{P}_{\bar{X}}$ from the training points and their labels, based on the observation that f_θ needs more data points to approximate f on the regions where it is steep. We derive an illustrative generalization bound involving the derivatives of f that theoretically corroborates this observation. Therefore, we build $d\mathbb{P}_{\bar{X}}$ using Taylor expansion of the function f, which links the local behavior of f to its derivatives.

We first focus on the influence of using $d\mathbb{P}_{\bar{X}}$ instead of $d\mathbb{P}_X$ in simple approximation problems. To that end, we build a methodology for constructing and exploiting $d\mathbb{P}_{\bar{X}}$, that we call Taylor Based sampling (TBS). We then apply TBS to a more realistic problem based on the approximation of the solution of Bateman equations. Solving these equations is an important part of many numerical simulations of several phenomena (neutronic [8,15], combustion [9], detonic [34], computational biology [42], etc.).

Then, we study the benefits of this approach for more general machine learning problems. In these cases, exploiting $d\mathbb{P}_{\bar{X}}$ is less straightforward. Indeed, we do not know the derivatives of f, and we cannot obtain labels for new data points sampled from this distribution. To tackle these problems, we show that variance is an approximation of Taylor expansion up to a certain order. Then we leverage the link between sampling and weighting to construct a methodology called Variance Based Sample Weighting (VBSW). This methodology weights each training data point using the local variance of their neighbor labels to simulate the new distribution. We specifically investigate its application in deep learning, where we apply VBSW within the feature space of a pre-trained neural network. We validate VBSW for deep learning by obtaining performance improvements on various tasks like classification and regression of text, from Glue benchmark [51], image, from MNIST [32] and Cifar10 [29] and multivariate data, from UCI machine learning repository [*], for several models ranging from linear regression to Bert [14] or ResNet20 [19]. We also conduct analyses on the complementarity of VBSW with other weighting techniques and its robustness to label noise.

1. RELATED WORKS

This work introduces contributions that rely on different elements. First, many techniques aim to alter the training distribution to improve the prediction error of neural networks. Second, finding generalization bounds for neural networks is the goal of various works in machine learning research. Finally, the methodology of constructing a sampling distribution for statistical analysis is used for importance sampling and designs of experiments.

Modified learning distributions. Some works are dedicated to improving neural network performances by modifying the training distribution, either by weighting data points or by inducing sample selection. Active learning [46] adapts the training strategy to a learning problem by introducing an online data point selection rule. [16] uses the variational properties of Bayesian neural network to design a rule that focuses the training on points that will reduce the prediction uncertainty of the neural network. In [28], the construction of the selection rule is itself taken as a machine learning problem. See [46] for a review of more classical active learning methods. Unlike active learning, and similarly to VBSW, some other methods aim at introducing diverse *a priori* evaluations of sample importance. While curriculum learning [6,37] starts the training

[*] `http://archive.ics.uci.edu/ml`

with easier examples, self-paced learning [22,30] downscales harder examples. However, some works have proven that focusing on harder examples at the beginning of the learning could accelerate it: [48] performs hard example mining to give more importance to harder examples by selecting them primarily. This work also focuses on defining hard examples but does so with an original, mathematical way based on f derivatives and local variance. It also stands out from the aforementioned techniques for how it modifies the distribution based on this information. Indeed, it suggests and justifies that a neural network should spend more learning time on subspaces of $\mathcal{X}$ which contain harder examples.

Generalization bounds. As an argument to motivate our approach, we derive a generalization bound. The construction of Generalization bounds for the learning theory of neural networks has motivated many works (see [21] for a review). In [4,5], the authors focus on Vapnik Chervonenkis (VC) dimension, a measure that depends on the number of parameters of neural networks. [2] introduces a compression approach that aims at reducing the number of model parameters to investigate its generalization capacities. Probably Approximately Correct (PAC) Bayes analysis constructs generalization bounds using *a priori* and *a posteriori* distributions over the possible models. It is investigated, for example, in [3,40]. [39,53] links PAC-Bayes theory to the notion of sharpness of a neural network, i.e. its robustness to small perturbation. While previous works often mention the sharpness of the model, our bound includes the derivatives of f, which can be seen as an indicator of the sharpness of the function to learn. Even if it uses elements of previous works, like the Lipschitz constant of f_θ, our work does not pretend to tighten and improve the already existing generalization bounds. It only emphasizes the intuition that the neural network would need more points to capture sharper functions. In a sense, it investigates the robustness to perturbations in the input space, not in the parameter space.

Examples weighting. VBSW can be categorized as an examples weighting, or importance weighting algorithm. The idea of weighting the data set has already been explored in different ways and for various purposes. Examples weighting is used in [13] to tackle the class imbalance problem by weighting rarer, so harder examples. On the contrary, in [33] it is used to solve the noisy label problem by focusing on cleaner, so easier examples. All these ideas show that depending on the application, examples weighting can be performed in an opposed manner. Some works aim at going beyond this opposition by proposing more general methodologies. In [11], the authors use the variance of the prediction of each point throughout the training to decide whether it should be weighted or not. A meta-learning approach is proposed in [44], where the authors choose the weights after an optimization loop included in the training. VBSW stands out from the previously mentioned examples weighting methods because it does not aim at solving dataset-specific problems like class imbalance or noisy labels. It is built on a more general assumption that a model would simply need more points to learn more complicated functions. The resulting weighting scheme verifies recent findings of [52] where authors conclude that in classification, a good set of weights would put importance on points close to the decision boundary.

Importance sampling. The challenge of finding a good distribution is not specific to machine learning. Indeed, in the context of Monte Carlo estimation of a quantity of interest based on a random variable, importance sampling owes its efficiency to the construction of a second random variable, which is used instead to improve the estimation of this quantity. [23] even make a connection between the success of likelihood ratio policy gradients and importance sampling,

which shows that machine learning and Monte Carlo estimation, both distribution-based methods, are closely linked. Moreover, some previously mentioned methods use importance sampling to design the weights of the data set or to correct the bias induced by the sample selection [26]. In this work, we construct a new distribution that could be interpreted as an importance distribution. However, we weigh the data points to simulate this distribution. It does not aim at correcting a bias induced by this distribution.

Designs of experiments. Some methodologies are dedicated to the construction of data sets in the context of statistical analysis. These methodologies are called designs of experiments. In our case, the construction of a new training distribution could be seen as a design of experiments for learning. However, popular designs of experiments used for regression are either space-filling designs or model-based designs. Space-filling designs, like Latin hypercube sampling [38] or maximin designs [24], aims at spreading the learning points to cover the input space as much as possible. Model-based designs use characteristics of f_θ to adapt the training distribution. Such designs can look to maximize the entropy of the prediction [47] or minimize its uncertainty [25]. These last designs of experiments can be conducted sequentially, getting close to active learning [12,35,45]. Our methodology does not depend on f_θ, nor aims at filling the input space. Instead, its goal is to adapt the design of experiments to characteristics of f in order to reduce the prediction error.

2. LINK BETWEEN LOCAL VARIATIONS AND LEARNING

Let us first remind some basics on supervised machine learning. We formalize the supervised machine learning task as approximating a function $f : \mathbf{S} \subset \mathbb{R}^{n_i} \to \mathbb{R}^{n_o}$ with a machine learning model f_θ parametrized by θ, where $\mathbf{S}$ is a measured sub-space of $\mathbb{R}^{n_i}$ depending on the application. To this end, we are given a training data set of N points, $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\} \in \mathbf{S}$, drawn from $\mathbf{x} \sim d\mathbb{P}_\mathbf{x}$ and their point-wise values, or labels $\{f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)\}$. Parameters θ have to be found in order to minimize an integrated loss function $J_\mathbf{x}(\theta) = \mathbb{E}[L(f_\theta(\boldsymbol{x}), f(\boldsymbol{x}))]$, with L the loss function, $L : \mathbb{R}^{n_o} \times \mathbb{R}^{n_o} \to \mathbb{R}$. The data allow estimating $J_\mathbf{x}(\theta)$ by $\widehat{J_\mathbf{x}}(\theta) = \sum_{i=1}^N \omega_i L(f_\theta(\boldsymbol{x}_i), f(\boldsymbol{x}_i))$, with $\{\omega_1, ..., \omega_N\} \in \mathbb{R}$ estimation weights, generally equal to $\frac{1}{N}$. Then, an optimization algorithm is used to find a minimum of $\widehat{J_\mathbf{x}}(\theta)$ w.r.t. θ.

2.1 Illustration of the Link using Derivatives

In the following, we illustrate the intuition with a Generalization Bound (GB) that include the derivatives of f, provided that these derivatives exist. The goal of the approximation problem is to be able to generalize to points not seen during the training. The generalization error $\mathcal{J}_\mathbf{x}(\theta) = J_\mathbf{x}(\theta) - \widehat{J_\mathbf{x}}(\theta)$ thus needs to be as small as possible. Let S_i, $i \in \{1, ..., N\}$ be some sub-spaces of $\mathbf{S}$ such that $\mathbf{S} = \bigcup_{i=1}^N S_i$, $\bigcap_{i=1}^N S_i = \emptyset$, and $\boldsymbol{x}_i \in S_i$. Suppose that L is the squared L_2 error, $n_i = n_o = 1$, f is differentiable, f_θ is K_θ-Lipschitz and satisfies the conditions of Hornik theorem [20]. Provided that $|S_i| < 1$, we show that

$$\mathcal{J}_\mathrm{x}(\theta) \leq \sum_{i=1}^N (|f'(x_i)| + K_\theta)^2 \frac{|S_i|^3}{4} + \mathcal{O}(|S_i|^4), \tag{1}$$

where $|S_i|$ is the volume of S_i ($|S_i| = \int_{S_i} d\mathbb{P}_{\mathbf{x}}$). The proof can be found in **Appendix A**. We see that in the regions where $f'(\boldsymbol{x}_i)$ is high, quantity $|S_i|$ has a stronger impact on the GB. This idea is illustrated in Figure 1, which visually shows that the generalization bound increases when $|S_i|$ and $f'(\boldsymbol{x}_i)$ are high at the same time for approximating the function $f : x \to x^3$. Since $|S_i|$ can be seen as a metric for how close data points are around $\boldsymbol{x}_i$ (the smaller $|S_i|$ is, the closer $\boldsymbol{x}_i$ is to its neighbors), the GB can be reduced more efficiently by adding more points around $\boldsymbol{x}_i$ in these regions. This bound also involves K_θ, the Lipschitz constant of the neural network, which has the same impact as $f'(\boldsymbol{x}_i)$. It also illustrates the link between the Lipschitz constant and the generalization error, which has been pointed out by several works like [17], [3] and [43].

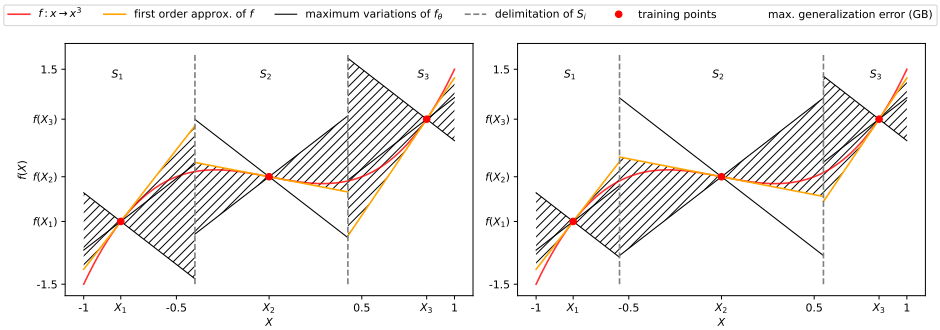


FIG. 1: Illustration of the GB. The maximum error (the GB), at order $\mathcal{O}(|S_i|^4)$, is obtained by comparing the maximum variations of f_θ, and the first order approximation of f, whose trends are given by K_θ and $f'(\boldsymbol{x}_i)$. We understand visually that because $f'(\boldsymbol{x}_1)$ and $f'(\boldsymbol{x}_3)$ are higher than $f'(\boldsymbol{x}_2)$, the GB is improved more efficiently by reducing S_1 and S_3 than S_2.

2.2 A Sampling Scheme based on Taylor Approximation

Equation (1) formalizes a link between generalization error and derivatives of f. These derivatives are expressed at order $n = 1$ for analytical reasons, but in this work we explore the use of derivatives of order $n > 1$. Using Taylor expansion at order n on f and supposing that f is n times differentiable:

$$f(\boldsymbol{x} + \boldsymbol{\epsilon}) \underset{\|\boldsymbol{\epsilon}\| \to 0}{=} \sum_{0 \leq |\boldsymbol{k}| \leq n} \boldsymbol{\epsilon}^{\boldsymbol{k}} \frac{\partial^{\boldsymbol{k}} f(\boldsymbol{x})}{\boldsymbol{k}!} + \mathcal{O}(\boldsymbol{\epsilon}^{\boldsymbol{n}}).$$

The quantity $f(\boldsymbol{x} + \boldsymbol{\epsilon}) - f(\boldsymbol{x}) = \sum_{1 \leq |\boldsymbol{k}| \leq n} \boldsymbol{\epsilon}^{\boldsymbol{k}} \frac{\partial^{\boldsymbol{k}} f(\boldsymbol{x})}{\boldsymbol{k}!} + \mathcal{O}(\boldsymbol{\epsilon}^{\boldsymbol{n}})$ gives an indication on how much f changes around $\boldsymbol{x}$. By neglecting the orders above $\boldsymbol{\epsilon}^{\boldsymbol{n}}$, it is then possible to find the regions of interest by focusing on $Df^n_{\boldsymbol{\epsilon}}$, defined as:

$$Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x}) = \sum_{1 \leq |\boldsymbol{k}| \leq n} \boldsymbol{\epsilon}^{k} \frac{(\partial^{\boldsymbol{k}} f(\boldsymbol{x}))^2}{\boldsymbol{k}!}, \tag{2}$$

Where $\boldsymbol{k}$ is a multi-index, i.e. $\boldsymbol{k} = (k_1, ..., k_{n_i})$ is a vector of n_i non negative integers, $|\boldsymbol{k}| = \sum_{i=1}^{n_i} k_i$, $\boldsymbol{k}! = k_1! \times ... \times k_{n_i}!$, $\boldsymbol{\epsilon}^{\boldsymbol{k}} = \epsilon_1^{k_1} \times ... \times \epsilon_{n_i}^{k_{n_i}}$, $\partial^{\boldsymbol{k}} = \frac{\partial^{k_1}}{\partial x_1^{k_1}} \times ... \times \frac{\partial^{k_{n_i}}}{\partial x_{n_i}^{k_{n_i}}}$. Note that $Df^n_{\boldsymbol{\epsilon}}$ is evaluated using $(\partial^{\boldsymbol{k}} f(\boldsymbol{x}))^2$ instead of $\partial^{\boldsymbol{k}} f(\boldsymbol{x})$ for derivatives not to cancel each other.

To avoid these cancellations, the absolute could have been used, but we will see in Lemma 1 that the square value ensures interesting asymptotical properties. f will be steeper and more irregular in the regions where $\boldsymbol{x} \to Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x})$ is higher. To focus the training set on these regions, one can use $\{Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x}_1), ..., Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x}_N)\}$ to construct a probability density function (pdf) and sample new data points from it.

In this part, we empirically verify that using Taylor expansion to construct a new training distribution has a beneficial impact on the performances of a neural network. To this end, we construct a methodology, that we call Taylor Based Sampling (TBS), that generates a new training data set based on the metric equation (2). To focus the training set on the regions of interest, i.e. regions of high $\{Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x}_1), ..., Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x}_N)\}$, we use this metric to construct a probability density function (pdf) - which is possible since $Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x}) \geq 0$ for all $\boldsymbol{x} \in \mathbf{S}$. It remains to normalize it but in practice it is enough considering a distribution $d\mathbb{P}_{\bar{\mathbf{x}}} \propto Df^n_{\boldsymbol{\epsilon}}$. Here, to approximate $d\mathbb{P}_{\bar{\mathbf{x}}}$ we use a Gaussian Mixture Model (GMM) with pdf $d\mathbb{P}_{\bar{\mathbf{x}},GMM}$ that we fit to $\{Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x}_1), ..., Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x}_N)\}$ using the Expectation-Maximization (EM) algorithm. N' new data points $\{\bar{\boldsymbol{x}}_1, ..., \bar{\boldsymbol{x}}_{N'}\}$, can be sampled, with $\bar{\mathbf{x}} \sim d\mathbb{P}_{\bar{\mathbf{x}},GMM}$. Finally, we obtain $\{f(\bar{\boldsymbol{x}}_1), ..., f(\bar{\boldsymbol{x}}_{N'})\}$, add it to $\{f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)\}$ and train our neural network on the whole data set.

TBS is described in Algorithm 1. **Line 1:** The parameter $\boldsymbol{\epsilon}$, the number of Gaussian distribution n_{GMM} and N' is chosen in order to avoid sparsity of $\{\bar{\boldsymbol{x}}_1, ..., \bar{\boldsymbol{x}}_{N'}\}$ over $\mathbf{S}$. **Line 2:** Without *a priori* information on f, we sample the first points uniformly in a subspace $\mathbf{S}$. **Line 3-7:** We construct $\{Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x}_1), ..., Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x}_N)\}$, and then $d\mathbb{P}_{\bar{\mathbf{x}},GMM}$ to be able to sample points accordingly. **Line 8:** Because the support of a GMM is not bounded, some points can be sampled outside $\mathbf{S}$. We discard these points and sample until all points are inside $\mathbf{S}$. This rejection method is equivalent to sampling points from a truncated GMM. **Line 9-10:** We construct the labels and add the new points to the initial data set.

Algorithm 1: Taylor Based Sampling (TBS)

Inputs: $\boldsymbol{\epsilon}$, N, N', n_{GMM}, n;
Sample $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ from $\mathbf{x} \sim \mathcal{U}(\mathbf{S})$;
for $0 \leq k \leq n$ **do**
  Compute $\{\partial^{\boldsymbol{k}} f(\boldsymbol{x}_1), ..., \partial^{\boldsymbol{k}} f(\boldsymbol{x}_N)\}$;
end
Compute $\{Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x}_1), ..., Df^n_{\boldsymbol{\epsilon}}(\boldsymbol{x}_N)\}$ using equation (2);
Approximate $d\mathbb{P}_{\bar{\mathbf{x}}} \propto Df^n_{\boldsymbol{\epsilon}}$ with a GMM using EM algorithm to obtain a density $d\mathbb{P}_{\bar{\mathbf{x}},GMM}$;
Sample $\{\bar{\boldsymbol{x}}_1, ..., \bar{\boldsymbol{x}}_{N'}\}$ using rejection method to sample inside $\mathbf{S}$;
Compute $\{f(\bar{\boldsymbol{x}}_1), ..., f(\bar{\boldsymbol{x}}_{N'})\}$;
Add $\{f(\bar{\boldsymbol{x}}_1), ..., f(\bar{\boldsymbol{x}}_{N'})\}$ to $\{f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)\}$;

2.3 Taylor based Sampling

2.3.1 Application to Simple Functions

To illustrate the benefits of TBS compared to a uniform, basic sampling (BS), we apply it to two simple functions: hyperbolic tangent and Runge function. We chose these functions because they

Sampling	L_2 error	L_∞ error
	f: Runge ($\times 10^{-2}$)	
BS	1.45 ± 0.62	5.31 ± 0.86
TBS	1.13 ± 0.73	3.87 ± 0.48
	f: tanh ($\times 10^{-1}$)	
BS	1.39 ± 0.67	2.75 ± 0.78
TBS	0.95 ± 0.50	2.25 ± 0.61

TABLE 1: Comparison between BS and TBS. The metrics used are the L_2 and L_∞ errors, displayed with a 95% confidence interval.

are differentiable and have a clear distinction between flat and steep regions. These functions are displayed in Figure 2, as well as the map $\boldsymbol{x} \to Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x})$.

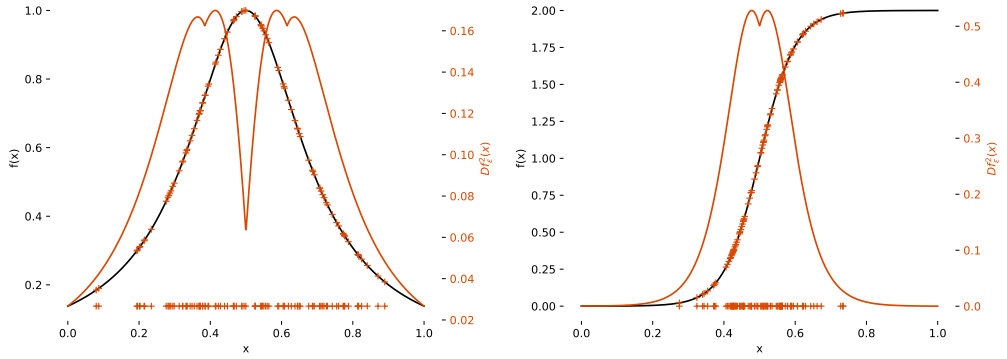


FIG. 2: Left: (left axis) Runge function w.r.t x and (right axis) $x \to Df^2_{\boldsymbol{\epsilon}}(x)$. Points sampled using TBS are plotted on the x-axis and projected on f. **Right:** Same as left, with hyperbolic tangent function.

All neural networks have been implemented in `Python`, with `Tensorflow` [1]. We use the Python package `scikit-learn` [41] to construct $d\mathbb{P}_{\tilde{\mathbf{x}},GMM}$. The network chosen for this experiment is a Multi Layer Perceptron (MLP) with one layer of 8 neurons and relu activation function, that we trained alternatively with BS and TBS using Adam optimizer [27] with the defaults tensorflow implementation hyperparameters, and Mean Squared Error loss function. We first sample $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ according to a regular grid. To compare the two methods, we add N' additional points sampled using BS to create the BS data set, and then N' other points sampled with TBS to construct the TBS data set. As a result, each data set have the same number of points $(N + N')$. We repeated the method for several values of n, n_{GMM} and $\boldsymbol{\epsilon}$, to fine tune these parameters and finally selected $n = 2$, $n_{\text{GMM}} = 3$ and $\boldsymbol{\epsilon} = 10^{-3}$.

Table 1 summarizes the L_2 and the L_∞ norm of the error of $f_{\boldsymbol{\theta}}$, obtained at the end of the training phase for $N + N' = 16$, with $N = 8$. Those norms are estimated using the same test data set of 1000 points. The values are the means of the 40 independent experiments displayed with a 95% confidence interval. These results illustrate the benefits of TBS over BS. Table 1 shows that TBS does not significantly improve L_2 error, but does so for L_∞ error, which may explain the good results of VBSW for classification that we describe in Section 5. Indeed, the accuracy will not be very sensitive to small output variations for a classification task since the output is rounded to 0 or 1. However, a high error increases the risk of misclassification, which

can be limited by the reduction of L_∞.

2.3.2 Application to an ODE System

We apply TBS to a more realistic case: the approximation of the resolution of the Bateman equations, an ODE system. In this system, u is the velocity of the reacting particles. Depending on the physical field of interest, u may be distributed according to a Maxwellian distribution (dense gas with chemical reactions for example) or may be distributed according to a distribution computed by another part of the code (this is the case in general for neutronic reactions or collisions in a rarefied plasma).

$$\begin{cases} \partial_t u(t) &= v\boldsymbol{\sigma_a} \cdot \boldsymbol{\eta}(t) u(t), \\ \partial_t \boldsymbol{\eta}(t) &= v\boldsymbol{\Sigma_r} \cdot \boldsymbol{\eta}(t) u(t), \end{cases} \text{, with initial conditions } \begin{cases} u(0) = u_0, \\ \boldsymbol{\eta}(0) = \boldsymbol{\eta}_0. \end{cases},$$

with $u \in \mathbb{R}^+, \boldsymbol{\eta} \in (\mathbb{R}^+)^M, \boldsymbol{\sigma}_a^T \in \mathbb{R}^M, \boldsymbol{\Sigma}_r \in \mathbb{R}^{M\times M}$. Here, $f : (u_0, \boldsymbol{\eta}_0, t) \to (u(t), \boldsymbol{\eta}(t))$. For physical applications, M ranges from tens to thousands, but we consider the particular case $M = 1$ so that $f : \mathbb{R}^3 \to \mathbb{R}^2$, with $f(u_0, \eta_0, t) = (u(t), \eta(t))$, and $\sigma_a = \sigma_r = -0.45$. The advantage of $M = 1$ is that we have access to an analytic, cheap to compute solution for f. Of course, this particular case can also be solved using a classical ODE solver, which allows us to test it end to end. It can thus be generalized to higher dimensions ($M > 1$).

All neural network training instances have been performed in `Python`, with `Tensorflow`. We used a fully connected neural network with hyperparameters chosen using a simple grid search. The final values are: 2 hidden layers, relu activation function, and 32 units for each layer, trained with the Mean Squared Error (MSE) loss function using Adam optimization algorithm with a batch size of 50000, for 40000 epochs and on $N + N' = 50000$ points, with $N = N'$. We trained the model for $(u(t), \eta(t)) \in \mathbb{R}$, with the $N + N'$ points sampled uniformly (BS), and compared it to TBS applied on N' after a uniform sampling of N points (TBS). We did so for several values of n, n_{GMM} and $\boldsymbol{\epsilon} = \epsilon(1, 1, 1)$, to fine tune these parameters. We finally select $\epsilon = 5 \times 10^{-4}$, $n = 2$ and $n_{\text{GMM}} = 10$. The data points used in this case have been sampled with an explicit Euler scheme. Note that we used this scheme because it is a stable converging accurate scheme if the time steps for the resolution are fine enough (which we thoroughly checked). Depending on the application, other schemes could be used (faster ones, stabler ones etc.). As we here mainly aim at building a database of solution, we are not constrained by some computational restrictions. So we decided to use a very simple scheme, easy to handle which can easily produce accurate solutions, even if costly (as it is only an offline cost). This experiment has been repeated 50 times to ensure statistical significance of the results.

Table 2 summarizes the MSE, i.e. the L_2 norm of the error of f_θ and L_∞ norm, with $L_\infty(\theta) = \max_{\boldsymbol{x}\in\mathbf{S}}(|f(\boldsymbol{x}) - f_\theta(\boldsymbol{x})|)$ obtained at the end of the training phase. This last metric is important because the goal in computational physics is not only to be averagely accurate, which is measured with MSE, but to be accurate over the whole input space $\mathbf{S}$. Those norms are estimated using a same test data set of $N_{test} = 50000$ points. The values are the means of the 50 independent experiments displayed with a 95% confidence interval. These results reflect an error reduction of 6.6% for L_2 and of 45.3% for L_∞, which means that TBS mostly improves the L_∞ error of f_θ. Moreover, the L_∞ error confidence intervals do not intersect so the gain is statistically significant for this norm.

Figure 3a shows how the neural network can perform for an average prediction. Figure

Sampling	L_2 error ($\times 10^{-4}$)	L_∞ ($\times 10^{-1}$)	AEG($\times 10^{-2}$)	AEL($\times 10^{-2}$)
BS	1.22 ± 0.13	5.28 ± 0.47	-	-
TBS	1.14 ± 0.15	2.96 ± 0.37	2.51 ± 0.07	0.42 ± 0.008

TABLE 2: Comparison between BS and TBS.

3b illustrates the benefits of TBS relative to BS on the L_∞ error (Figure 2b). These 2 figures confirm the previous observation about the gain in L_∞ error. Finally, Figure 3c displays $u_0, \eta_0 \rightarrow \max_{0 \leq t \leq 10} D^n_\epsilon(u_0, \eta_0, t)$ w.r.t. (u_0, η_0) and shows that D^n_ϵ increases when $U_0 \rightarrow 0$. TBS hence focuses on this region. Note that for the readability of these plots, the values are capped to 0.10. Otherwise only few points with high D^n_ϵ are visible. Figure 3d displays $u_0, \eta_0 \rightarrow g_{\theta_{BS}}(u_0, \eta_0) - g_{\theta_{TBS}}(u_0, \eta_0)$, with $g_\theta : u_0, \eta_0 \rightarrow \max_{0 \leq t \leq 10} \|f(u_0, \eta_0, t) - f_\theta(u_0, \eta_0, t)\|_2^2$ where θ_{BS} and θ_{TBS} denote the parameters obtained after a training with BS and TBS, respectively. It can be interpreted as the error reduction achieved with TBS.

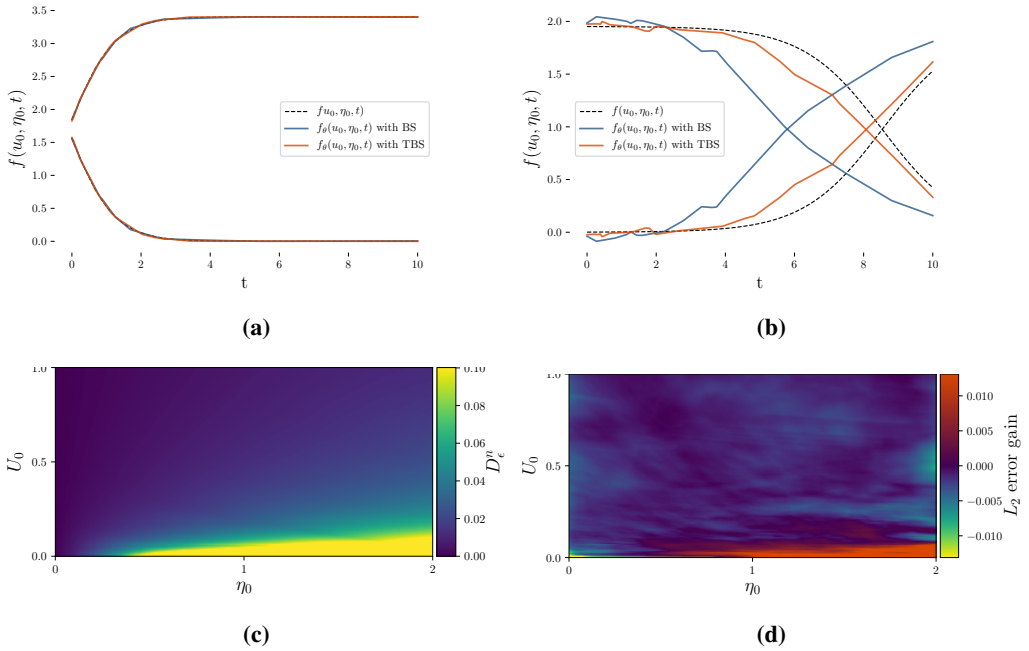


FIG. 3: **(a)** $t \rightarrow f_\theta(u_0, \eta_0, t)$ for randomly chosen (u_0, η_0), for f_θ obtained with the two samplings. **(b)** $t \rightarrow f_\theta(u_0, \eta_0, t)$ for (u_0, η_0) resulting in the highest point-wise error with the two samplings. **(c)** $u_0, \eta_0 \rightarrow \max_{0 \leq t \leq 10} D^n_\epsilon(u_0, \eta_0, t)$ w.r.t. (u_0, η_0). **(d)** $u_0, \eta_0 \rightarrow g_{\theta_{BS}}(u_0, \eta_0) - g_{\theta_{TBS}}(u_0, \eta_0)$,

The highest error reduction occurs in the expected region. Indeed, more points are sampled where D^n_ϵ is higher. The error is slightly increased in the rest of $\mathbf{S}$, which could be explained by a sparser sampling on this region. However, as summarized in Table 2, the average error loss (AEL) of TBS is around six times lower than the average error gain (AEG), with $AEG = \mathbb{E}[Z(u_0, \eta_0) 1_{Z>0}]$ and $AEL = \mathbb{E}[Z(u_0, \eta_0) 1_{Z<0}]$ where $Z(u_0, \eta_0) = g_{\theta_{BS}}(u_0, \eta_0) - g_{\theta_{TBS}}(u_0, \eta_0)$. In practice, AEG and AEL are estimated using uniform grid integration, and averaged on the 50 experiments.

3. GENERALIZATION OF TAYLOR BASED SAMPLING

The previous section empirically validated the intuition behind the construction of a new, more efficient training distribution $d\mathbb{P}_{\bar{\mathbf{x}}}$. However, this new distribution cannot always be applied as-is for two reasons. **Problem 1:** $\{Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x}_1), ..., Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x}_N)\}$ cannot be evaluated since it requires to compute the derivatives of f, and it assumes that f is differentiable, which is often not true. Moreover, the previously described setting, in which we focus on f derivatives, is not suited to classification tasks where the notion of derivatives is not straightforward. **Problem 2:** even if $\{Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x}_1), ..., Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x}_N)\}$ could be computed and new points sampled, we could not obtain their labels to complete the training data set. In this section, we alleviate this concern to be able to use insights from $d\mathbb{P}_{\bar{\mathbf{x}}}$ in practice.

3.1 From Taylor Expansion to Local Variance

To overcome **problem 1**, we construct a new metric based on statistical estimation. In this paragraph, $n_i > 1$ but $n_o = 1$. The following derivations can be extended to $n_o > 1$ by applying it to f element-wise and then taking the sum across the n_o dimensions.

Lemma 1. *Let* $\mathbf{e} \sim \mathcal{N}(0, \epsilon \boldsymbol{I}_{n_i})$ *with* $\epsilon \in \mathbf{R}^+$ *and* $\boldsymbol{I}_{n_i}$ *the identity matrix of dimension* n_i*. Let* $\boldsymbol{\epsilon} = \epsilon(1, ..., 1)$*. Then,*

$$Var(f(\boldsymbol{x} + \mathbf{e})) = Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x}) + \mathcal{O}(\|\boldsymbol{\epsilon}\|_2^3).$$

The demonstration can be found in **Appendix A**. Using the unbiased estimator of variance, we thus define new indices $\widehat{Df^2_{\boldsymbol{\epsilon}}}(\boldsymbol{x})$ by

$$\widehat{Df^2_{\boldsymbol{\epsilon}}}(\boldsymbol{x}) = \frac{1}{k-1} \sum_{i=1}^{k} \Big(f(\boldsymbol{x} + \boldsymbol{\epsilon_i}) - f(\boldsymbol{x}) \Big)^2, \tag{3}$$

with $\{\boldsymbol{\epsilon}_1, ..., \boldsymbol{\epsilon}_k\}$ k samples of $\boldsymbol{\epsilon}$. The metric $\widehat{Df^2_{\boldsymbol{\epsilon}}}(\boldsymbol{x}) \underset{k\to\infty}{\to} Var(f(\boldsymbol{x}+\boldsymbol{\epsilon}))$ and $Var(f(\boldsymbol{x}+\boldsymbol{\epsilon})) = Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x}) + \mathcal{O}(\|\boldsymbol{\epsilon}\|_2^3)$, so $\widehat{Df^2_{\boldsymbol{\epsilon}}}(\boldsymbol{x})$ is a biased estimator of $Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x})$, with bias $\mathcal{O}(\|\boldsymbol{\epsilon}\|_2^3)$. Hence, when $\boldsymbol{\epsilon} \to 0$, $\widehat{Df^2_{\boldsymbol{\epsilon}}}(\boldsymbol{x})$ becomes an unbiased estimator of $Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x})$. It is possible to compute $\widehat{Df^2_{\boldsymbol{\epsilon}}}(\boldsymbol{x})$ from any set of points centered around $\boldsymbol{x}$. Therefore, we evaluate $\widehat{Df^2_{\boldsymbol{\epsilon}}}(\boldsymbol{x}_i)$ for each $i \in \{1, ..., N\}$ using the set $\mathcal{S}_k(\boldsymbol{x}_i)$ of k-nearest neighbors of $\boldsymbol{x}_i$. We note this metric $\widehat{Df^2}(\boldsymbol{x}_i)$, where we replace $f(\boldsymbol{x}_i + \boldsymbol{\epsilon_i})$ by $f(\boldsymbol{x}_l)$, the values of f for the neighbors of $\boldsymbol{x}_i$ ($\boldsymbol{x}_l \in \mathcal{S}_k(\boldsymbol{x}_i)$) and $f(\boldsymbol{x}_i)$ by $\frac{1}{k}\sum^k_{\boldsymbol{x}_l \in \mathcal{S}_k(\boldsymbol{x}_i)} f(\boldsymbol{x}_l)$, the average of f on the neighbors of $\boldsymbol{x}_i$:

$$\widehat{Df^2}(\boldsymbol{x}_i) = \frac{1}{k-1} \sum_{\boldsymbol{x}_j \in \mathcal{S}_k(\boldsymbol{x}_i)} \Big(f(\boldsymbol{x}_j) - \frac{1}{k} \sum_{\boldsymbol{x}_l \in \mathcal{S}_k(\boldsymbol{x}_i)}^{k} f(\boldsymbol{x}_l) \Big)^2, \tag{4}$$

Equation (4) has several practical advantages. First, $\widehat{Df^2}$ can even be applied to non-differentiable functions and for classification problems, unlike equation (2). Second, the definition of $\widehat{Df^2}(\boldsymbol{x})$ does not rely on $\boldsymbol{\epsilon}$, unlike equation (3). To compute $\widehat{Df^2}$, all we need are $\{f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)\}$, the points used for the training of the neural network. In addition, equation (4) can even be applied when the data points are too sparse for the nearest neighbors of $\boldsymbol{x}_i$ to be considered as close

to $\boldsymbol{x}_i$, which is almost always the case in high dimension. It can thus be seen as a generalization of $\widehat{Df^2_{\boldsymbol{\epsilon}}}(\boldsymbol{x})$, which tends towards $Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x})$ locally.

3.2 From Sampling to Weighting

To tackle **problem 2**, recall that the goal of the training is to find $\theta^* = \underset{\theta}{\operatorname{argmin}}\ \widehat{J_{\mathbf{x}}}(\theta)$, with $\widehat{J_{\mathbf{x}}}(\theta) = \frac{1}{N}\sum_i L(f(\boldsymbol{x}_i), f_\theta(\boldsymbol{x}_i))$. With the new distribution based on previous derivations, the procedure is different. Since the training points are sampled using $\widehat{Df^2_{\boldsymbol{\epsilon}}}$, we no longer minimize $\widehat{J_{\mathbf{x}}}(\theta)$, but $\widehat{J_{\bar{\mathbf{x}}}}(\theta) = \frac{1}{N}\sum_i L(f(\bar{\boldsymbol{x}}_i), f_\theta(\bar{\boldsymbol{x}}_i))$, with $\bar{\mathbf{x}} \sim d\mathbb{P}_{\bar{\mathbf{x}}}$ the new distribution. However, $\widehat{J_{\bar{\mathbf{x}}}}(\theta)$ estimates

$$J_{\bar{\mathbf{x}}}(\theta) = \int_{\mathbf{S}} L(f(\boldsymbol{x}), f_\theta(\boldsymbol{x}))d\mathbb{P}_{\bar{\mathbf{x}}}.$$

Let $p_{\mathbf{x}}(\boldsymbol{x})d\boldsymbol{x} = d\mathbb{P}_{\mathbf{x}}$, $p_{\bar{\mathbf{x}}}(\boldsymbol{x})d\boldsymbol{x} = d\mathbb{P}_{\bar{\mathbf{x}}}$ be the pdfs of $\mathbf{x}$ and $\bar{\mathbf{x}}$ (note that $Df^2_{\boldsymbol{\epsilon}} \propto p_{\bar{\mathbf{x}}}$). Then,

$$J_{\bar{\mathbf{x}}}(\theta) = \int_{\mathbf{S}} L(f(\boldsymbol{x}), f_\theta(\boldsymbol{x}))\frac{p_{\bar{\mathbf{x}}}(\boldsymbol{x})}{p_{\mathbf{x}}(\boldsymbol{x})}d\mathbb{P}_{\mathbf{x}}.$$

The straightforward Monte Carlo estimator for this expression of $J_{\bar{\mathbf{x}}}(\theta)$ is

$$\widehat{J_{\bar{\mathbf{x}}}}(\theta) = \frac{1}{N}\sum_i L(f(\boldsymbol{x}_i), f_\theta(\boldsymbol{x}_i))\frac{p_{\bar{\mathbf{x}}}(\boldsymbol{x}_i)}{p_{\mathbf{x}}(\boldsymbol{x}_i)} \propto \frac{1}{N}\sum_i L(f(\boldsymbol{x}_i), f_\theta(\boldsymbol{x}_i))\frac{\widehat{Df^2}(\boldsymbol{x}_i)}{p_{\mathbf{x}}(\boldsymbol{x}_i)}. \quad (5)$$

Thus, $J_{\bar{\mathbf{x}}}(\theta)$ can be estimated with the same points as $J_{\mathbf{x}}(\theta)$ by weighting them with $w_i = \frac{\widehat{Df^2}(\boldsymbol{x}_i)}{p_{\mathbf{x}}(\boldsymbol{x}_i)}$.

The expression of w_i involves $p_{\mathbf{x}}$, the distribution of the data. Just like for f, we do not have access to $p_{\mathbf{x}}$. The estimation of $p_{\mathbf{x}}$ is a challenging task by itself, and standard density estimation techniques such as *K*-nearest neighbors or Gaussian Mixture density estimation led to extreme estimated values of $p_{\mathbf{x}}(\boldsymbol{x}_i)$ in our experiments. Therefore, we decided to only apply $\omega_i = \widehat{Df^2}(\boldsymbol{x}_i)$ as a first-order approximation. In practice, we re-scale the weights between 1 and m, a hyperparameter, and then divide them by their sum to avoid affecting the learning rate.

As a result, we obtain a new methodology based on weighting the training data set. We call this methodology Variance Based Sample Weighting (VBSW).

4. VARIANCE BASED SAMPLE WEIGHTING

In this part, we sum up Variance Based Sample Weighting (VBSW) to clarify its application to machine learning problems. We also study this methodology through toy experiments.

4.1 Methodology

Variance Based Samples Weighting (VBSW) is recapitulated in Algorithm 2. **Line 1:** m and k are hyperparameters that can be chosen jointly with all other hyperparameters, e.g. using a random search. Their effects and interactions are studied and discussed in Sections 4.2 and 5.4. **Line 2-3:** equation (4) is applied to compute the weights w_i that are used to weight the data set. Notations $\{(w_1, \boldsymbol{x}_1), ..., (w_N, \boldsymbol{x}_N)\}$ denote that each $\boldsymbol{x}_i$ is weighted by w_i. To perform a

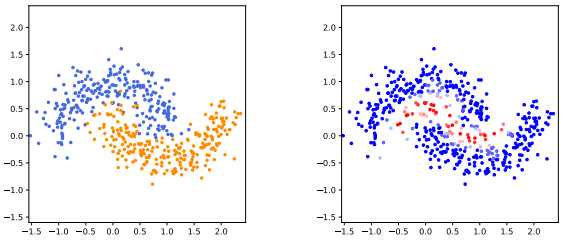
nearest-neighbors search, we use an approximate nearest neighbor search technique called hierarchical navigable small world graphs [36] implemented by nmslib [10]. **Line 4:** Train f_θ on the weighted data set.

Algorithm 2: Variance Based Samples Weighting (VBSW)

Inputs: k, m;
Compute $\{\widehat{Df^2}(\boldsymbol{x}_1), ..., \widehat{Df^2}(\boldsymbol{x}_N)\}$ using equation (4);
Construct a new training data set $\{(w_1, \boldsymbol{x}_1), ..., (w_N, \boldsymbol{x}_N)\}$;
Train f_θ on $\{(w_1, f(\boldsymbol{x}_1)), ..., (w_N, f(\boldsymbol{x}_N))\}$;

4.2 Toy Experiments & Hyperparameter Study

VBSW is studied on a Double Moon (DM) classification problem, the Boston Housing (BH) regression, and Breast Cancer (BC) classification data sets.



(a) **(b)**

FIG. 4: From left to right: **(a)** Double Moon (DM) data set. **(b)** Heat map of the value of w_i for each $\boldsymbol{x}_i$ (red is high and blue is low)

For DM, Figure 4b shows that the points with higher w_i (in red) are close to the boundary between the two classes. Indeed, in classification, VBSW can be interpreted as a local label agreement. This behavior verifies recent findings of [52] where authors conclude that in classification, a good set of weights would put importance on points close to the decision boundary.

We train a Multi-Layer Perceptron of 1 layer of 4 units, using Stochastic Gradient Descent (SGD) and binary cross-entropy loss function, on a 300 points training data set for 50 random seeds. In this experiment, VBSW, i.e. weighting the data set with w_i is compared to the baseline where no weights are applied. The results of Table 3 show the improvement obtained with VBSW.

	VBSW	baseline
DM	**99.4**, **94.44** $\pm$ 0.78	99, 92.06 $\pm$ 0.66
BH	**13.31**, **13.38** $\pm$ 0.01	14.05, 14.06 $\pm$ 0.01
BC	**99.12**, 97.6 $\pm$ 0.34	98.25, 97.5 $\pm$ 0.11

TABLE 3: **best, mean + se** for each method. The metric used is accuracy for DM and BC and Mean Squared Error for BH.

For BH data set, a linear model is trained, and for BC data set, an MLP of 1 layer and 30 units, with a train-validation split of $80\% - 20\%$. Both models are trained with Adam [27]. Since these data sets are small and the models are light, we study the effects of m and k on the error. Moreover, BH is a regression task and BC a classification task, so it allows studying the effect of hyperparameters more extensively.

For BH and BC experiments, we conduct a grid search for VBSW on the values of m and k. As a reminder, m is the ratio between the highest and the lowest weights, and k is the number of neighbor points used to compute the local variance. We train a linear model for BH and a MLP with 30 units for BC with VBSW on a grid of 20 values of m equally distributed between 2 and 100 and 20 values of k equally distributed between 10 and 50. As a result, we train the model on 400 pairs of (m, k) values and with 10 different random seeds for each pair.

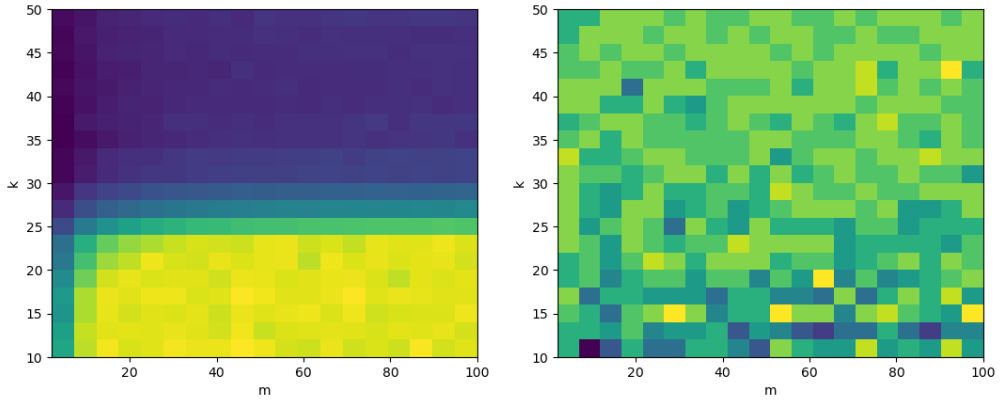


FIG. 5: Color map of the error, with respect to m and k. **Left:** BH data set, for the mean of the MSE across 10 different seeds and **right:** BC data set, for the mean of $1 - acc$ across these seeds. Blue is lower.

These experiments, illustrated in Figure 5 show that the influence of m and k on the performances of the model can be different. For BH data set, low values of k clearly lead to poorer performances. Hyperparameter m seems to have less impact, although it should be chosen not too far from its lowest value, 2. For BC data set, on the contrary, the best performances are obtained for low values of k, while a high value could be chosen for m. These experiments highlight that the impact of m and k can be different between classification and regression, but it could also be different depending on the data set. Hence, we recommend considering these hyperparameters like many others involved in deep learning, selecting their values using hyperparameters optimization techniques.

It also shows that many different (m, k) pairs lead to error improvement. It suggests that the weights approximation does not have to be exact for VBSW to be effective, as stated in Section 5.4.

4.3 Cost Efficiency of VBSW

VBSW's computational burden mostly relies on the complexity of the nearest neighbor search algorithm, which is independent and can be used as a third-party algorithm. When the data set is not too large, classical techniques like KDtree [7] can be used. However, when the number of points and the dimension of the data set increase, approximate nearest neighbors searches

may be necessary to keep satisfying performances. In the previous examples, KDtree is more than sufficient. However, since we deal with more complex examples in the following, we directly use nmslib [10], an approximate nearest neighbors search library for homogeneity of the implementation.

5. VBSW FOR DEEP LEARNING

The high dimensionality of many deep learning problems makes VBSW difficult to apply in the form previously described. In this part, we adapt VBSW to such problems and study its application to various real-world learning tasks. We also study the robustness of VBSW and its complementarity with other similar techniques.

5.1 Methodology

We mentioned that local variance could be computed using already existing points. This statement implies finding the nearest neighbors of each point. In extremely high-dimensional spaces like image spaces, the curse of dimensionality makes nearest neighbors vacuous. In addition, the data structure may be highly irregular, and the concept of nearest neighbor may be misleading. Thus, it would be irrelevant to evaluate $\widehat{Df^2}$ directly on this data.

One of the strengths of deep learning is to construct good representations of the data embedded in lower-dimensional latent spaces. For instance, in Computer Vision, convolutional neural networks' deeper layers represent more abstract features. We could leverage this representational power of neural networks and simply apply our methodology within this latent feature space.

Variance Based Samples Weighting (VBSW) for deep learning is recapitulated in Algorithm 3. Here, $\mathcal{M}$ is the initial neural network whose feature space will be used to project the training data set and apply VBSW. **Line 1:** m and k are hyperparameters that can be chosen jointly with all other hyperparameters, e.g. using a random search. Their effects and interactions are studied and discussed in Sections 4.2 and 5.4. **Line 2:** The initial neural network, $\mathcal{M}$, is trained as usual. Notations $\{(\frac{1}{N}, \boldsymbol{x}_1), ..., (\frac{1}{N}, \boldsymbol{x}_N)\}$ is equivalent to $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$, because all the weights are the same ($\frac{1}{N}$). **Line 3:** The last fully connected layer is discarded, resulting in a new model $\mathcal{M}^*$, and the training data set is projected in the feature space. **Line 4-5:** equation (4) is applied to compute the weights w_i that are used to weight the projected data set. **Line 6:** The last layer is re-trained (which is often equivalent to fitting a linear model) using the weighted data set and added to $\mathcal{M}^*$ to obtain the final model $\mathcal{M}_f$. As a result, $\mathcal{M}_f$ is a composition of the already trained model $\mathcal{M}^*$ and f_θ trained using the weighted data set.

Algorithm 3: Variance Based Samples Weighting (VBSW) for deep learning

Inputs: k, m, $\mathcal{M}$;
Train $\mathcal{M}$ on the training set $\{(\frac{1}{N}, \boldsymbol{x}_1), ..., (\frac{1}{N}, \boldsymbol{x}_N)\}$;
$\{(\frac{1}{N}, f(\boldsymbol{x}_1)), ..., (\frac{1}{N}, f(\boldsymbol{x}_N))\}$;
Construct $\mathcal{M}^*$ by removing its last layer ;
Compute $\{w_1 = \widehat{Df^2}(\mathcal{M}^*(\boldsymbol{x}_1)), ..., w_N = \widehat{Df^2}(\mathcal{M}^*(\boldsymbol{x}_N))\}$ using equation (4);
Construct a new training data set $\{(w_1, \mathcal{M}^*(\boldsymbol{x}_1)), ..., (w_N, \mathcal{M}^*(\boldsymbol{x}_N))\}$;
Train f_θ on the training set of inputs $\{(w_1, \mathcal{M}^*(\boldsymbol{x}_1)), ..., (w_N, \mathcal{M}^*(\boldsymbol{x}_N))\}$ with outputs $\{f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)\}$ and add it to $\mathcal{M}^*$. The final model is $\mathcal{M}_f = f_\theta \circ \mathcal{M}^*$;

5.2 Image Classification

In this section, we study the performances of VBSW on MNIST [32] and Cifar10 [29] image classification data sets. For MNIST, we train LeNet [31], with 40 different random seeds, and then apply VBSW for 10 different random seeds, with Adam optimizer and categorical cross-entropy loss. Note that in the following, Adam is used with the default parameters of its `keras` implementation. We record the best value obtained from the 10 VBSW training. We follow the same procedure for Cifar10, except that we train a ResNet20 for 50 random seeds and with data augmentation and learning rate decay. The networks have been trained on 4 Nvidia K80 GPUs. The values of the hyperparameters used can be found in **Appendix B**. We compare the test accuracy between LeNet 5 + VBSW, ResNet20 + VBSW, and the initial test accuracies of LeNet 5 and ResNet20 (baseline) for each of the initial networks.

	VBSW	baseline	gain per model
MNIST	**99.09**, **98.87** $\pm$ 0.01	98.99, 98.84 $\pm$ 0.01	**0.15**, **0.03** $\pm$ 0.01
Cifar10	**91.30**, **90.64** $\pm$ 0.07	91.01, 90.46 $\pm$ 0.10	**1.65**, **0.15** $\pm$ 0.04

TABLE 4: **best, mean + se** for each method. The metric used is accuracy. For a model $\mathcal{M}$, the gain g for this model is given by $g = \max_{1 \leq i \leq 10} (acc(\mathcal{M}_f^i) - acc(\mathcal{M}))$ with acc the accuracy and $\mathcal{M}_f^i$ the VBSW model trained at the i-th random seed.

The results statistics are gathered in Table 4, which also displays statistics about the gain due to VBSW for each model. The results on MNIST are slightly but consistently better than for the baseline, by 0.1% for the best with up to 0.15% of accuracy gain per model. For Cifar10, we get a 0.3% accuracy improvement for the best model and up to 1.65% accuracy gain, meaning that among the 50 ResNet20s, there is one whose accuracy has been improved by 1.65% using VBSW. Note that applying VBSW took less than 15 minutes on a laptop with an i7-7700HQ CPU. A visualization of the samples weighted by the highest w_i is given in Figure 6.

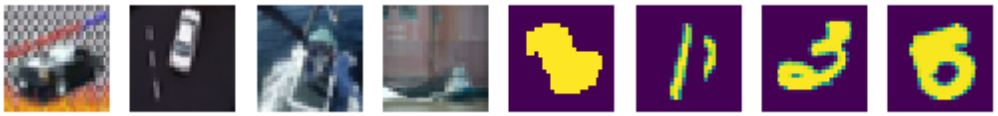


FIG. 6: Samples from Cifar10 and MNIST with high w_i. Those pictures are either unusual or difficult to classify, even for a human (especially for MNIST).

5.3 Text Classification and Regression

In this section, we study the performances of VBSW on RTE and MRPC, two text classification data sets, and STS-B, a text classification data set, extracted from the glue benchmark [51]. For this application, we use Bert, a modern neural network based on transformers [50] that is the state-of-the-art of text-based machine learning tasks. We do not pre-train Bert, like in the previous experiments, since it has been originally built for Transfer Learning purposes. Therefore, its purpose is to be used as-is and then fine-tuned on any text data set see [14]. However, because of the small size of the data set and the high number of model parameters, we chose not to fine-tune the Bert model and only to use the representations of the data sets in its feature space to

apply VBSW. More specifically, we use tiny-bert [49], which is a lighter version of the initial Bert. We train the linear model with TensorFlow to be able to add the trained model on top of the Bert model and obtain a unified model. RTE and MRPC are classification tasks, so we use binary cross-entropy loss function to train our models. STS-B is a regression task, so the model is trained with Mean Squared Error. All the models are trained with Adam optimizer. For each task, we compare the training of the linear model with VBSW and without VBSW (baseline). The results obtained with VBSW are better overall, except for Pearson Correlation in STS-B, which is slightly worse than baseline (Table 5).

	VBSW		baseline	
	m1	m2	m1	m2
RTE	**61.73**, **58.46** $\pm$ 0.15	-	61.01, 58.09 $\pm$ 0.13	-
STS-B	**62.31**, **62.20** $\pm$ 0.01	**60.99**, 60.88 $\pm$ 0.01	61.88, 61.87 $\pm$ 0.01	60.98, **60.92** $\pm$ 0.01
MRPC	**72.30**, **71.71** $\pm$ 0.03	**82.64**, **80.72** $\pm$ 0.05	71.56, 70.92 $\pm$ 0.03	81.41, 80.02 $\pm$ 0.07

TABLE 5: best, mean + se for each method. For RTE the metric used is accuracy (m1). For STS-B, metric 1 (m1) is Spearman correlation and metric 2 (m2) is Pearson correlation. For MRPC, metric 1 (m1) is accuracy and metric 2 (m2) is F1 score.

5.4 Robustness of VBSW

In this section, we assess the robustness of VBSW. First, we focus on the robustness to label noise. To that end, we train a ResNet20 on Cifar10 with four different noise levels. We randomly change the label of $p\%$ training points for four different values of p (10, 20, 30 and 40). We then apply VBSW 30 times and evaluate the obtained neural networks on a clean test set. The results are gathered in Table 6.

noise	10%	20%	30%	40%
original error	87.43	85.75	84.05	81.79
VBSW	**87.76**, 8**7.63** $\pm$ 0.01	**86.03**, **85.89** $\pm$ 0.01	**84.35**, **84.18** $\pm$ 0.02	**82.48**, **82.32** $\pm$ 0.02

TABLE 6: best, mean + se of the training of a ResNet20 on Cifar10 for different label noise levels. These results illustrate the robustness of VBSW to labels noise.

The results show that VBSW is still effective despite label noise. This specificity must be related to the robustness of VBSW with respect to the choice of hyperparameters m and k, as seen in Section 4.2. Indeed, it shows that many combinations of m and k improves the performances of the neural network, and therefore that VBSW is actually robust to error in the weights evaluation. Its robustness to label noise hence stems from its robustness to weights evaluation error, since label noise essentially hurts the accuracy of the weights evaluation.

Although VBSW is robust to label noise, note that the goal of VBSW is not to address noisy label problem, like discussed in Section 1. It may be more effective to use a sampling technique tailored specifically for this situation.

5.5 Complementarity of VBSW

Existing techniques based on dataset processing can be used jointly with VBSW, by applying the first technique during the initial training of the neural network and then applying VBSW on

its feature space. To illustrate this specificity, we compare VBSW with the recently introduced Active Bias (AB) [11] and transfer-learning-based curriculum learning (TCL) [18]. AB dynamically weights the samples based on the variance of the probability of prediction of each point throughout the training, and TCL creates a curriculum based on sample difficulty evaluated on previously trained neural networks. Here, we study the effects of AB and TCL combined with VBSW for the training of a ResNet20 on Cifar10. Table 7 gathers the results of experiments for different baselines: vanilla, for regular training with Adam optimizer, AB / TCL for training with AB / TCL, VBSW for the application of VBSW on top of regular training, and VBSW + AB / VBSW + CL for initial training with AB / TCL and the application of VBSW. Unlike in Section 5.2, we do not use data augmentation nor learning rate decay in order to simplify the experiments.

	accuracy (%)	VBSW gpm
vanilla	75.88, 74.55 ± 0.11	-
AB	76.33, 75.14 ± 0.09	-
TCL	78.54, 77.46 ± 0.07	-
VBSW	76.57, 74.94 ± 0.10	0.94, 0.40 ± 0.03
AB + VBSW	76.60, 75.33 ± 0.09	0.40, 0.14 ± 0.02
TCL + VBSW	79.86, 78.71 ± 0.09	2.19, 1.26 ± 0.08

TABLE 7: Best, mean + se of the training of 60 ResNet20s on Cifar10 for vanilla, VBSW, AB and AB + VBSW. The gain per model (gpm) g is defined by $g = \max_{1 \leq i \leq 10} (acc(\mathcal{M}_f^i) - acc(\mathcal{M}))$ with acc the accuracy and $\mathcal{M}_f^i$ the VBSW model trained at the i-th random seed.

The accuracy obtained with VBSW is quite similar to AB. While TCL yields better results than VBSW alone, the best accuracy is obtained when they are used jointly. Overall, the best neural networks are obtained when AB and TCL are used along with VBSW (AB + VBSW and TCL + VBSW), which demonstrates the complementarity of VBSW with other dataset processing techniques. Note that VBSW works much better when applied to a neural network initially trained with TCL. It means that TCL creates neural network features particularly suited to VBSW. This lead might be explored in future works.

6. DISCUSSION AND PERSPECTIVES

By studying the training distribution of the neural network, we explored a practical and classical question that naturally arises when performing surrogate modeling for approximating computer codes: how to construct the training set? We found that exploring this question led to findings that are also relevant for approximation theory, which is an important component of machine learning.

Hence, the results obtained in this paper are impactful both for machine learning in numerical simulations and machine learning in general.

6.1 Impact for Numerical Simulations

This work comes from the observation that, on our approximation problems, neural networks are more efficient when more data are sampled where the function to learn is steeper. It is an attempt to formalize this observation and to construct a workable methodology out of it. As a result, the

methodologies for constructing the distribution $d\mathbb{P}_{\tilde{\mathbf{x}}}$ can be used as new, principled designs of experiments.

In the context of numerical simulations, once $d\mathbb{P}_{\tilde{\mathbf{x}}}$ is constructed, it is possible to sample new data from it. It alleviates **Problem 2**, described in Section 3.2. In theory, **Problem 1** is also solved since we could have access to the derivatives - either by instrumenting the code with automatic differentiation if we have access to its implementation or by estimating them with finite differences. However, the implementation of automatic differentiation can be tedious, and if the computer code is slow and high dimensional, finite differences may be unaffordable. In that case, it is possible to use a third methodology based on the approximation of $\{Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x}_1), ..., Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x}_N)\}$ using local variance, like VBSW, and the sampling of new points, like TBS.

Finally, the method allows improving the error of neural networks without increasing the computational cost of their prediction. This achievement is of interest when they are intended to accelerate numerical simulations.

6.2 Impact for Machine Learning

VBSW is validated on several tasks, complementary with other training distribution modification frameworks, and robust to noise. It makes it quite versatile. Moreover, the problem of high dimensionality and irregularity of f, which often arises in deep learning problems, is alleviated by focusing on the latent space of neural networks. This makes VBSW scalable. As a result, VBSW can be applied to complex neural networks such as ResNet, or Bert, for various machine learning tasks.

The experiments support an original view of the learning problem that involves the local variations of f. The studies of Section 2.2, that use the derivatives of the function to learn to sample a more efficient training data set, support this approach as well. This view is also bolstered up by conclusions of [52]. VBSW allows extending this original view to problems where the derivatives of f are not accessible and sometimes not defined. Indeed, VBSW comes from Taylor expansion, which is specific to differentiable functions, but in the end, it can be applied regardless of the properties of f.

Finally, this method is cost-effective. In most cases, it allows to quickly improve the performances of a neural network using a regular CPU. It is better than carrying on entirely new training with a wider and deeper neural network.

6.3 Further Studies

Although VBSW uses theoretically justified approximations concerning TBS, the actual effect of these approximations should be more thoroughly investigated. For instance, we could further study the impact of not explicitly using $p_{\mathbf{x}}$, the data distribution, in the weights definitions; the convergence of the estimator of $Df^2_{\boldsymbol{\epsilon}}$, and in which context it is adequately approximated; and a more generic derivative-based generalization bound. In addition, VBSW demonstrated intriguing behaviors, like its impressive synergy with TCL [18], which would deserve more attention.

7. CONCLUSION

This work is based on the observation that, in supervised learning, a function f is more difficult to approximate by a neural network in the regions where it is steep. We mathematically traduced this intuition, derived a generalization bound to illustrate it, and a methodology, Taylor Based

Sampling, to test it empirically. In order to be able to use these insights for machine learning problems where f is not available, we constructed a weighting scheme, Variance Based Samples Weighting (VBSW) that uses the variance of the training samples' labels to weight the training data set. VBSW is simple to use and implement because it only requires computing statistics on the input space. In Deep Learning, applying VBSW on the data set projected in an already trained neural network feature space allows reducing its error by simply re-training its last layer. Although specifically investigated in deep learning, this method applies to any loss-function-based supervised learning problem and is scalable, cost-effective, robust, and versatile. It is validated on several applications, such as glue benchmark with bert for text classification and regression, and Cifar10 with ResNet for image classification.

APPENDIX A. APPENDIX A: PROOFS

APPENDIX A.1 Illustration of the Link using Derivatives

(Section 2.1)

We look at approximating $f : \boldsymbol{x} \to f(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}^{n_i}$, $f(\boldsymbol{x}) \in \mathbb{R}^{n_o}$ with a NN $f_{\boldsymbol{\theta}}$. The goal of the approximation problem can be seen as being able to generalize to points not seen during the training. We thus want the generalization error $\mathcal{J}_{\mathbf{x}}(\boldsymbol{\theta})$ to be as small as possible. Given an initial data set $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ drawn from $\mathbf{x} \sim d\mathbb{P}_{\mathbf{x}}$ and $\{f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)\}$, and the loss function L being the squared L_2 error, recall that the integrated error $J_{\mathbf{x}}(\boldsymbol{\theta})$, its estimation $\widehat{J_{\mathbf{x}}}(\boldsymbol{\theta})$ and the generalization error $\mathcal{J}_{\mathbf{x}}(\boldsymbol{\theta})$ can be written:

$$\begin{aligned} J_{\mathbf{x}}(\boldsymbol{\theta}) &= \int_{\mathbf{S}} \|f(\boldsymbol{x}) - f_{\boldsymbol{\theta}}(\boldsymbol{x})\| d\mathbb{P}_{\mathbf{x}}, \\ \widehat{J_{\mathbf{x}}}(\boldsymbol{\theta}) &= \frac{1}{N} \sum_{i=1}^{N} \|f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - f(\boldsymbol{x}_i)\|, \\ \mathcal{J}_{\mathbf{x}}(\boldsymbol{\theta}) &= J_{\mathbf{x}}(\boldsymbol{\theta}) - \widehat{J_{\mathbf{x}}}(\boldsymbol{\theta}), \end{aligned} \tag{A1}$$

where $\|.\|$ denotes the squared L_2 norm. In the following, we find an upper bound for $\mathcal{J}_{\mathbf{x}}(\boldsymbol{\theta})$. We start by finding an upper bound for $J_{\mathbf{x}}(\boldsymbol{\theta})$ and then for $\mathcal{J}_{\mathbf{x}}(\boldsymbol{\theta})$ using equation (A1).

Let S_i, $i \in \{1, ..., N\}$ be some sub-spaces of a bounded space $\mathbf{S}$ such that $\mathbf{S} = \bigcup_{i=1}^{N} S_i$, $\bigcap_{i=1}^{N} S_i = \emptyset$, and $\boldsymbol{x}_i \in S_i$. Then,

$$\begin{aligned} J_{\mathbf{x}}(\boldsymbol{\theta}) &= \sum_{i=1}^{N} \int_{S_i} \|f(\boldsymbol{x}) - f_{\boldsymbol{\theta}}(\boldsymbol{x})\| d\mathbb{P}_{\mathbf{x}}, \\ J_{\mathbf{x}}(\boldsymbol{\theta}) &= \sum_{i=1}^{N} \int_{S_i} \|f(\boldsymbol{x}_i + \boldsymbol{x} - \boldsymbol{x}_i) - f_{\boldsymbol{\theta}}(\boldsymbol{x})\| d\mathbb{P}_{\mathbf{x}}. \end{aligned}$$

Suppose that $n_i = n_o = 1$ ($\boldsymbol{x}$ becomes x and $\mathbf{x}$ becomes x) and f twice differentiable. Let $|\mathbf{S}| = \int_{\mathbf{S}} d\mathbb{P}_{\mathrm{x}}$. The volume $|\mathbf{S}| = 1$ since $d\mathbb{P}_{\mathrm{x}}$ is a probability measure, and therefore $|S_i| < 1$ for all $i \in \{1, ..., N\}$. Using Taylor expansion at order 2, and since $|S_i| < 1$ for all $i \in \{1, ..., N\}$

$$J_{\mathrm{x}}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \int_{S_i} \|f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2} f''(x_i)(x - x_i)^2 - f_{\boldsymbol{\theta}}(x) + \mathcal{O}((x - x_i)^3)\| d\mathbb{P}_{\mathrm{x}}.$$

To find an upper bound for $J(\boldsymbol{\theta})$, we can first find an upper bound for $|A_i(x)|$, with $A_i(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2} f''(x_i)(x - x_i)^2 - f_{\boldsymbol{\theta}}(x) + \mathcal{O}((x - x_i)^3)$.

NN $f_{\boldsymbol{\theta}}$ is $K_{\boldsymbol{\theta}}-$Lipschitz, so since $\mathbf{S}$ is bounded (so are S_i), for all $x \in S_i$, $|f_{\boldsymbol{\theta}}(x) - f_{\boldsymbol{\theta}}(x_i)| \leq K_{\boldsymbol{\theta}} |x - x_i|$. Hence,

$$f_\theta(x_i) - K_\theta|x - x_i| \leq f_\theta(x) \leq f_\theta(x_i) + K_\theta|x - x_i|,$$
$$-f_\theta(x_i) - K_\theta|x - x_i| \leq -f_\theta(x) \leq -f_\theta(x_i) + K_\theta|x - x_i|,$$
$$f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2}f''(x_i(x - x_i)^2) - f_\theta(x_i) - K_\theta|x - x_i| + \mathcal{O}((x - x_i)^3)$$
$$\leq A_i(x) \leq f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2}f''(x_i)(x - x_i)^2 - f_\theta(x_i) + K_\theta|x - x_i| + \mathcal{O}((x - x_i)^3),$$
$$A_i(x) \leq f(x_i) - f_\theta(x_i) + f'(x_i)(x - x_i) + \frac{1}{2}f''(x_i)(x - x_i)^2 + K_\theta|x - x_i| + \mathcal{O}((x - x_i)^3).$$

And finally, using triangular inequality,

$$\boxed{A_i(x) \leq |f(x_i) - f_\theta(x_i)| + |f'(x_i)||x - x_i| + \frac{1}{2}|f''(x_i)||x - x_i|^2 + K_\theta|x - x_i| + \mathcal{O}(|x - x_i|^3).}$$

Now, $\|.\|$ being the squared L_2 norm:

$$J_{\mathrm{x}}(\theta) = \sum_{i=1}^{N} \int_{S_i} \|f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2}f''(x_i)(x - x_i)^2 - f_\theta(x) + \mathcal{O}(|x - x_i|^3)\| d\mathbb{P}_{\mathrm{x}},$$

$$J_{\mathrm{x}}(\theta) \leq \sum_{i=1}^{N} \int_{S_i} \Bigg[\Big(|f(x_i) - f_\theta(x_i)|\Big) + \Big(|f'(x_i)||x - x_i| + \frac{1}{2}|f''(x_i)||x - x_i|^2 + K_\theta|x - x_i|\Big) + \mathcal{O}(|x - x_i|^3) \Bigg]^2 d\mathbb{P}_{\mathrm{x}},$$

$$= \sum_{i=1}^{N} \int_{S_i} \Bigg[|f(x_i) - f_\theta(x_i)|^2 + 2|f(x_i) - f_\theta(x_i)|\Big(|f'(x_i)||x - x_i| + \frac{1}{2}|f''(x_i)||x - x_i|^2 + K_\theta|x - x_i|\Big) + \Big[\Big(|f'(x_i)||x - x_i|\Big) + \Big(\frac{1}{2}|f''(x_i)||x - x_i|^2 + K_\theta|x - x_i|\Big)\Big]^2 + \mathcal{O}(|x - x_i|^3) \Bigg] d\mathbb{P}_{\mathrm{x}},$$

$$= \sum_{i=1}^{N} \int_{S_i} \Bigg[|f(x_i) - f_\theta(x_i)|^2 + 2|f(x_i) - f_\theta(x_i)|\Big(|f'(x_i)||x - x_i| + \frac{1}{2}|f''(x_i)||x - x_i|^2 + K_\theta|x - x_i|\Big) + \Big[|f'(x_i)|^2|x - x_i|^2 + 2K_\theta|f'(x_i)||x - x_i|^2 + K_\theta^2|x - x_i|^2\Big] + \mathcal{O}(|x - x_i|^3) \Bigg] d\mathbb{P}_{\mathrm{x}},$$

$$= \sum_{i=1}^{N} \int_{S_i} \Bigg[|f(x_i) - f_\theta(x_i)|^2 + 2|f(x_i) - f_\theta(x_i)|\Big(|f'(x_i)||x - x_i| + \frac{1}{2}|f''(x_i)||x - x_i|^2 + K_\theta|x - x_i|\Big) + \Big(|f'(x_i)| + K_\theta\Big)^2|x - x_i|^2 + \mathcal{O}(|x - x_i|^3) \Bigg] d\mathbb{P}_{\mathrm{x}}.$$

Hornik's theorem [20] states that given a norm $\|.\|_{p,\mu} =$ such that $\|f\|_{p,\mu}^p = \int_{\mathbf{S}} |f(x)|^p d\mu(x)$, with $d\mu$ a probability measure, for any ϵ, there exists θ such that for a Multi Layer Perceptron, f_θ, $\|f(x) - f_\theta(x)\|_{p,\mu}^p < \epsilon$,

This theorem grants that for any ϵ, with $d\mu = \sum_{i=1}^N \frac{1}{N}\delta(x - x_i)$, there exists θ such that

$$\begin{cases} \|f(x) - f_\theta(x)\|_{1,\mu}^1 = \displaystyle\sum_{i=1}^N \frac{1}{N}|f(x_i) - f_\theta(x_i)| \leq \epsilon, \\ \|f(x) - f_\theta(x)\|_{2,\mu}^2 = \displaystyle\sum_{i=1}^N \frac{1}{N}\big(f(x_i) - f_\theta(x_i)\big)^2 \leq \epsilon. \end{cases} \tag{A2}$$

Let's introduce i^* such that $i^* = \text{argmin}\,|S_i|$. Note that for any $i \in \{1, ..., N\}$, $\mathcal{O}(|S_i^*|^4)$ is $\mathcal{O}(|S_i|^4)$. Now, let's choose ϵ such that ϵ is $\mathcal{O}(|S_i^*|^4)$. Then, equation (A2) implies that

$$\begin{cases} |f(x_i) - f_\theta(x_i)| = \mathcal{O}(|S_i|^4), \\ \big(f(x_i) - f_\theta(x_i)\big)^2 = \mathcal{O}(|S_i|^4), \\ \widehat{J}_{\mathrm{x}}(\theta) = \|f(x) - f_\theta(x)\|_{2,\mu}^2 = \mathcal{O}(|S_i|^4). \end{cases}$$

Thus, we have $\mathcal{J}_{\mathrm{x}}(\theta) = J_{\mathrm{x}}(\theta) - \widehat{J}_{\mathrm{x}}(\theta) = J_{\mathrm{x}}(\theta) + \mathcal{O}(|S_i|^4)$ and therefore,

$$\mathcal{J}_{\mathrm{x}}(\theta) \leq \sum_{i=1}^N \int_{S_i} \Big[\Big(|f'(x_i)| + K_\theta\Big)^2 |x - x_i|^2 d\mathbb{P}_{\mathrm{x}}\Big] + \mathcal{O}(|S_i|^4).$$

Finally,

$$\boxed{\mathcal{J}_{\mathrm{x}}(\theta) \leq \sum_{i=1}^N (|f'(x_i)| + K_\theta)^2 \frac{|S_i|^3}{3} + \mathcal{O}(|S_i|^4).} \tag{A3}$$

We see that on the regions where $f'(x_i) + K_\theta$ is higher, quantity $|S_i|$ (the volume of S_i) has a stronger impact on the GB. Then, since $|S_i|$ can be seen as a metric for the local density of the data set (the smaller $|S_i|$ is, the denser the data set is), the Generalization Bound (GB) can be reduced more efficiently by adding more points around x_i in these regions. This bound also involves K_θ, the Lipschitz constant of the NN, which has the same impact as $f'(x_i)$. It also illustrates the link between the Lipschitz constant and the generalization error, which has been pointed out by several works like, for instance, [17], [3] and [43].

APPENDIX A.2 Problem 1: Unavailability of Derivatives

(Section 3.1)

In this paragraph, we consider $n_i > 1$ but $n_o = 1$. The following derivations can be extended to $n_o > 1$ by applying it to f element-wise. Let $\mathbf{e} \sim \mathcal{N}(0, \epsilon \boldsymbol{I}_{n_i})$ with $\epsilon \in \mathbb{R}^+$, $\mathbf{e} = (\epsilon_1, ..., \epsilon_{n_i})$, i.e. $\epsilon_i \sim \mathcal{N}(0, \epsilon)$ and $\boldsymbol{\epsilon} = \epsilon(1, ..., 1)$. Using Taylor expansion on f at order 2 gives

$$f(\boldsymbol{x} + \mathbf{e}) = f(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \cdot \mathbf{e} + \frac{1}{2}\mathbf{e}^T \cdot \mathbb{H}_x f(\boldsymbol{x}) \cdot \mathbf{e} + \mathcal{O}(\|\mathbf{e}\|_2^3),$$

with $\nabla_x f$ and $\mathbb{H}_x f(\boldsymbol{x})$ the gradient and the Hessian of f w.r.t. $\boldsymbol{x}$. We now compute $Var(f(X+\mathbf{e}))$ and make $Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x}) = \epsilon\|\nabla_x f(\boldsymbol{x})\|^2_F + \frac{1}{2}\epsilon^2\|\mathbb{H}_{\boldsymbol{x}} f(\boldsymbol{x})\|^2_F$ appear in its expression to establish a link between these two quantities:

$$\begin{aligned} Var(f(\boldsymbol{x}+\mathbf{e})) &= Var\Big(f(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} f(\boldsymbol{x})\cdot\mathbf{e} + \frac{1}{2}\mathbf{e}^T\cdot\mathbb{H}_x f(\boldsymbol{x})\cdot\mathbf{e} + \mathcal{O}(\|\mathbf{e}\|_2^3)\Big), \\ &= Var\Big(\nabla_{\boldsymbol{x}} f(\boldsymbol{x})\cdot\mathbf{e} + \frac{1}{2}\mathbf{e}^T\cdot\mathbb{H}_x f(\boldsymbol{x})\cdot\mathbf{e}\Big) + \mathcal{O}(\|\boldsymbol{\epsilon}\|_2^3). \end{aligned}$$

Since $\epsilon_i \sim \mathcal{N}(0,\epsilon)$, $\boldsymbol{x} = (x_1, ..., x_{n_i})$ and with $\frac{\partial^2 f}{\partial x_i x_j}(\boldsymbol{x})$ the cross derivatives of f w.r.t. x_i and x_j,

$$\begin{aligned} \nabla_{\boldsymbol{x}} f(\boldsymbol{x})\cdot\mathbf{e} + \frac{1}{2}\mathbf{e}^T\cdot\mathbb{H}_x f(\boldsymbol{x})\cdot\mathbf{e} &= \sum_{i=1}^{n_i}\epsilon_i\frac{\partial f}{\partial x_i}(\boldsymbol{x}) + \frac{1}{2}\sum_{j=1}^{n_i}\sum_{k=1}^{n_i}\epsilon_j\epsilon_k\frac{\partial^2 f}{\partial x_j x_k}(\boldsymbol{x}), \\ Var\Big(\nabla_{\boldsymbol{x}} f(\boldsymbol{x})\cdot\mathbf{e} + \frac{1}{2}\mathbf{e}^T\cdot\mathbb{H}_x f(\boldsymbol{x})\cdot\mathbf{e}\Big) &= Var\Big(\sum_{i=1}^{n_i}\epsilon_i\frac{\partial f}{\partial x_i}(\boldsymbol{x}) + \frac{1}{2}\sum_{j=1}^{n_i}\sum_{k=1}^{n_i}\epsilon_j\epsilon_k\frac{\partial^2 f}{\partial x_j x_k}(\boldsymbol{x})\Big), \\ &= \sum_{i_1=1}^{n_i}\sum_{i_2=1}^{n_i} Cov\Big(\epsilon_{i_1}\frac{\partial f}{\partial x_{i_1}}(\boldsymbol{x}), \epsilon_{i_2}\frac{\partial f}{\partial x_{i_2}}(\boldsymbol{x})\Big), \\ &\quad + \frac{1}{4}\sum_{j_1=1}^{n_i}\sum_{k_1=1}^{n_i}\sum_{j_2=1}^{n_i}\sum_{k_2=1}^{n_i} Cov\Big(\epsilon_{j_1}\epsilon_{k_1}\frac{\partial^2 f}{\partial x_{j_1}x_{k_1}}(\boldsymbol{x}), \epsilon_{j_2}\epsilon_{k_2}\frac{\partial^2 f}{\partial x_{j_2}x_{k_2}}(\boldsymbol{x})\Big) \\ &\quad + \sum_{i=1}^{n_i}\sum_{j=1}^{n_i}\sum_{k=1}^{n_i} Cov\Big(\epsilon_i\frac{\partial f}{\partial x_i}(\boldsymbol{x}), \epsilon_j\epsilon_k\frac{\partial^2 f}{\partial x_j x_k}(\boldsymbol{x})\Big), \\ &= \sum_{i_1=1}^{n_i}\sum_{i_2=1}^{n_i}\frac{\partial f}{\partial x_{i_1}}(\boldsymbol{x})\frac{\partial f}{\partial x_{i_2}}(\boldsymbol{x}) Cov\Big(\epsilon_{i_1}, \epsilon_{i_2}\Big) \\ &\quad + \frac{1}{4}\sum_{j_1=1}^{n_i}\sum_{k_1=1}^{n_i}\sum_{j_2=1}^{n_i}\sum_{k_2=1}^{n_i}\frac{\partial^2 f}{\partial x_{j_1}x_{k_1}}(\boldsymbol{x})\frac{\partial^2 f}{\partial x_{j_2}x_{k_2}}(\boldsymbol{x}) Cov\Big(\epsilon_{j_1}\epsilon_{k_1}, \epsilon_{j_2}\epsilon_{k_2}\Big) \\ &\quad + \sum_{i=1}^{n_i}\sum_{j=1}^{n_i}\sum_{k=1}^{n_i}\frac{\partial f}{\partial x_i}(\boldsymbol{x})\frac{\partial^2 f}{\partial x_j x_k}(\boldsymbol{x}) Cov\Big(\epsilon_i, \epsilon_j\epsilon_k\Big). \end{aligned}$$

In this expression, three quantities have to be assessed : $Cov(\epsilon_{i_1}, \epsilon_{i_2})$, $Cov(\epsilon_i, \epsilon_j\epsilon_k)$ and $Cov(\epsilon_{j_1}\epsilon_{k_1}, \epsilon_{j_2}\epsilon_{k_2})$.

First, since $(\epsilon_1, ..., \epsilon_{n_i})$ are i.i.d.,

$$Cov\Big(\epsilon_{i_1}, \epsilon_{i_2}\Big) = \begin{cases} Var(\epsilon_i) = \epsilon \text{ if } i_1 = i_2 = i, \\ 0 \text{ otherwise.} \end{cases}.$$

To assess $Cov(\epsilon_i, \epsilon_j\epsilon_k)$, three cases have to be considered.

- If $i = j = k$, because $\mathbb{E}[\epsilon_i^3] = 0$,
$$\begin{aligned} Cov(\epsilon_i, \epsilon_j\epsilon_k) &= Cov(\epsilon_i, \epsilon_i^2), \\ &= \mathbb{E}[\epsilon_i^3] - \mathbb{E}[\epsilon_i]\mathbb{E}[\epsilon_i^2], \\ &= 0. \end{aligned}$$

- If $i = j$ or $i = k$ (we consider $i = k$, and the result holds for $i = j$ by commutativity),
$$\begin{aligned} Cov(\epsilon_i, \epsilon_j\epsilon_k) &= Cov(\epsilon_i, \epsilon_i\epsilon_j), \\ &= \mathbb{E}[\epsilon_i^2\epsilon_j] - \mathbb{E}[\epsilon_i]\mathbb{E}[\epsilon_i\epsilon_j], \\ &= \mathbb{E}[\epsilon_i^2]\mathbb{E}[\epsilon_j], \\ &= 0. \end{aligned}$$
- If $i \neq j$ and $i \neq k$, ϵ_i and $\epsilon_j\epsilon_k$ are independent and so $Cov(\epsilon_i, \epsilon_j\epsilon_k) = 0$.

Finally, to assess $Cov(\epsilon_{j_1}\epsilon_{k_1}, \epsilon_{j_2}\epsilon_{k_2})$, four cases have to be considered:

- If $j_1 = j_2 = k_1 = k_2 = i$,
$$\begin{aligned} Cov(\epsilon_{j_1}\epsilon_{k_1}, \epsilon_{j_2}\epsilon_{k_2}) &= Var(\epsilon_i^2), \\ &= 2\epsilon^2. \end{aligned}$$
- If $j_1 = k_1 = i$ and $j_2 = k_2 = j$, $Cov(\epsilon_{j_1}\epsilon_{k_1}, \epsilon_{j_2}\epsilon_{k_2}) = Cov(\epsilon_i^2, \epsilon_j^2) = 0$ since ϵ_i^2 and ϵ_j^2 are independent.
- If $j_1 = j_2 = j$ and $k_1 = k_2 = k$,
$$\begin{aligned} Cov(\epsilon_{j_1}\epsilon_{k_1}, \epsilon_{j_2}\epsilon_{k_2}) &= Var(\epsilon_j\epsilon_k), \\ &= Var(\epsilon_j)Var(\epsilon_k), \\ &= \epsilon^2. \end{aligned}$$
- If $j_1 \neq k_1, j_2$ and k_2,
$$\begin{aligned} Cov(\epsilon_{j_1}\epsilon_{k_1}, \epsilon_{j_2}\epsilon_{k_2}) &= \mathbb{E}[\epsilon_{j_1}\epsilon_{k_1}\epsilon_{j_2}\epsilon_{k_2}] - \mathbb{E}[\epsilon_{j_1}\epsilon_{k_1}]\mathbb{E}[\epsilon_{j_2}\epsilon_{k_2}], \\ &= \mathbb{E}[\epsilon_{j_1}]\mathbb{E}[\epsilon_{k_1}\epsilon_{j_2}\epsilon_{k_2}] - \mathbb{E}[\epsilon_{j_1}]\mathbb{E}[\epsilon_{k_1}]\mathbb{E}[\epsilon_{j_2}\epsilon_{k_2}], \\ &= 0. \end{aligned}$$

All other possible cases can be assessed using the previous results, commutativity and symmetry of Cov operator. Hence,

$$\begin{aligned} Var\Big(\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \cdot \mathbf{e} + \frac{1}{2}\mathbf{e}^T \cdot \mathbb{H}_x f(\boldsymbol{x}) \cdot \mathbf{e}\Big) =& \sum_{i_1=1}^{n_i}\sum_{i_2=1}^{n_i} \frac{\partial f}{\partial x_{i_1}}(\boldsymbol{x})\frac{\partial f}{\partial x_{i_2}}(\boldsymbol{x}) Cov\Big(\epsilon_{i_1}, \epsilon_{i_2}\Big) \\ &+ \frac{1}{4}\sum_{j_1=1}^{n_i}\sum_{k_1=1}^{n_i}\sum_{j_2=1}^{n_i}\sum_{k_2=1}^{n_i} \frac{\partial^2 f}{\partial x_{j_1}x_{k_1}}(\boldsymbol{x})\frac{\partial^2 f}{\partial x_{j_2}x_{k_2}}(\boldsymbol{x}) Cov\Big(\epsilon_{j_1}\epsilon_{k_1}, \epsilon_{j_2}\epsilon_{k_2}\Big), \\ =& \sum_{i=1}^{n_i} \epsilon\frac{\partial f^2}{\partial x_i}(\boldsymbol{x}) + \frac{1}{2}\sum_{j=1}^{n_i}\sum_{k=1}^{n_i} \epsilon^2\frac{\partial^2 f^2}{\partial x_j x_k}(\boldsymbol{x}), \\ =& \epsilon\|\nabla_x f(\boldsymbol{x})\|_F^2 + \frac{1}{2}\epsilon^2\|\mathbb{H}_{\boldsymbol{x}} f(\boldsymbol{x})\|_F^2, \\ =& Df_{\boldsymbol{\epsilon}}^2(\boldsymbol{x}). \end{aligned}$$

And finally,

$$\boxed{Var(f(\boldsymbol{x} + \mathbf{e})) = Df_{\boldsymbol{\epsilon}}^2(\boldsymbol{x}) + \mathcal{O}(\|\boldsymbol{\epsilon}\|_2^3)} \tag{A4}$$

If we consider $\widehat{Df^2}_{\epsilon}(\boldsymbol{x})$ as defined in equation (2), on section* 2.2 of the main document, $\widehat{Df^2}_{\epsilon}(\boldsymbol{x}) \underset{k\to\infty}{\to} Var(f(\boldsymbol{x} + \boldsymbol{\epsilon}))$. Since $Var(f(\boldsymbol{x} + \boldsymbol{\epsilon})) = Df_{\boldsymbol{\epsilon}}^2(\boldsymbol{x}) + \mathcal{O}(\|\boldsymbol{\epsilon}\|_2^3)$, $\widehat{Df^2}_{\epsilon}(\boldsymbol{x})$ is

a biased estimator of $Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x})$, with bias $\mathcal{O}(\|\boldsymbol{\epsilon}\|_2^3)$. Hence, when $\epsilon \to 0$, $\widehat{Df^2}_{\epsilon}(\boldsymbol{x})$ becomes an unbiased estimator of $Df^2_{\boldsymbol{\epsilon}}(\boldsymbol{x})$.

APPENDIX B. APPENDIX B: HYPERPARAMETERS SPACES

The values chosen for the hyperparameters experiments are gathered in Table B1. For Adam optimizer hyperparameters, we kept the default values of Keras implementation. We chose these hyperparameters after simple grid searches.

Experiment	m	k	learning rate	batch size	epochs	optimizer	random seeds
double moon	100	20	1×10^{-3}	100	10000	SGD	50
Boston housing	8	35	5×10^{-4}	404	50000	Adam	10
Breast Cancer	50	35	5×10^{-2}	455	250000	Adam	10
MNIST	40	20	1×10^{-3}	25	25	Adam	40
Cifar10	40	20	1×10^{-3}	25	25	Adam	50
RTE	20	10	3×10^{-4}	8	10000	Adam	50
STS-B	30	30	3×10^{-4}	8	10000	Adam	50
MRPC	75	25	3×10^{-4}	16	10000	Adam	50

TABLE B1: Hyperparameters values for experiments.

REFERENCES

1. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
2. Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 254–263, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.
3. Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6240–6249. Curran Associates, Inc., 2017.
4. Peter L. Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks. *Journal of Machine Learning Research*, 20(63):1–17, 2019.
5. Peter L. Bartlett, Vitaly Maiorov, and Ron Meir. Almost linear vc dimension bounds for piecewise polynomial networks. In *Proceedings of the 11th International Conference on Neural Information Processing Systems*, NIPS'98, page 190–196, Cambridge, MA, USA, 1998. MIT Press.
6. Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, 2009. ACM.
7. Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

8. Adrien Bernède and Gaël Poëtte. An unsplit monte-carlo solver for the resolution of the linear boltzmann equation coupled to (stiff) bateman equations. *Journal of Computational Physics*, 354:211–241, 02 2018.
9. M. Bisi and L. Desvillettes. From reactive boltzmann equations to reaction–diffusion systems. *Journal of Statistical Physics*, 124(2):881–912, Aug 2006.
10. Leonid Boytsov and Bilegsaikhan Naidan. Engineering efficient and effective non-metric space library. In Nieves R. Brisaboa, Oscar Pedreira, and Pavel Zezula, editors, *Similarity Search and Applications - 6th International Conference, SISAP 2013, A Coruña, Spain, October 2-4, 2013, Proceedings*, volume 8199 of *Lecture Notes in Computer Science*, pages 280–293. Springer, 2013.
11. Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1002–1012. Curran Associates, Inc., 2017.
12. David A. Cohn. Neural Network Exploration Using Optimal Experiment Design. *Neural Networks*, 9(6):1071–1083, August 1996.
13. Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-balanced loss based on effective number of samples. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
14. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.
15. Jan Dufek, Dan Kotlyar, and Eugene Shwageraus. The stochastic implicit euler method – a stable coupling scheme for monte carlo burnup calculations. *Annals of Nuclear Energy*, 60:295 – 300, 10 2013.
16. Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1183–1192. JMLR.org, 2017.
17. Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael Cree. Regularisation of neural networks by enforcing lipschitz continuity. 04 2018.
18. Guy Hacohen and Daphna Weinshall. On the power of curriculum learning in training deep networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2535–2544. PMLR, 09–15 Jun 2019.
19. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.
20. Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.
21. Daniel Jakubovitz, Raja Giryes, and Miguel R. D. Rodrigues. Generalization error in deep learning. *CoRR*, abs/1808.01174, 2018.
22. Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander G. Hauptmann. Self-paced curriculum learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 2694–2700. AAAI Press, 2015.
23. Tang Jie and Pieter Abbeel. On a connection between importance sampling and the likelihood ratio policy gradient. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta,

editors, *Advances in Neural Information Processing Systems 23*, pages 1000–1008. Curran Associates, Inc., 2010.

24. M.E. Johnson, L.M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. *Journal of Statistical Planning and Inference*, 26(2):131–148, October 1990.
25. Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, Dec 1998.
26. Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *ICML*, 2018.
27. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.
28. Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Learning active learning from data. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4225–4235. Curran Associates, Inc., 2017.
29. Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.
30. M. P. Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1189–1197. Curran Associates, Inc., 2010.
31. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
32. Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.
33. Tongliang Liu and Dacheng Tao. Classification with noisy labels by importance reweighting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(3):447–461, March 2016.
34. D. Lucor, C. Enaux, H. Jourdren, and P. Sagaut. Stochastic design optimization: Application to reacting flows. *Computer Methods in Applied Mechanics and Engineering*, 196(49):5047 – 5062, 2007.
35. David J. C. MacKay. Information-Based Objective Functions for Active Data Selection. *Neural Computation*, 4(4):590–604, July 1992.
36. Yu A. Malkov and D. A. Yashunin. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, April 2020.
37. Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher–student curriculum learning. *IEEE Transactions on Neural Networks and Learning Systems*, 31:3732–3740, 2020.
38. M. D. McKay, R. J. Beckman, and W. J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2):239, May 1979.
39. Behnam Neyshabur, Srinadh Bhojanapalli, David Mcallester, and Nati Srebro. Exploring generalization in deep learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5947–5956. Curran Associates, Inc., 2017.
40. Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. A PAC-bayesian approach to spectrally-normalized margin bounds for neural networks. In *International Conference on Learning Representations*, 2018.
41. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
42. Benoit Perthame. *Transport Equations in Biology*. 01 2007.

43. Haifeng Qian and Mark N. Wegman. L2-nonexpansive neural networks. In *International Conference on Learning Representations*, 2019.
44. Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. *CoRR*, abs/1803.09050, 2018.
45. S. Seo, M. Wallat, T. Graepel, and K. Obermayer. Gaussian process regression: Active data selection and test point rejection. In *Proceedings of the International Joint Conference on Neural Networks*, 2000.
46. Burr Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.
47. M. C. Shewry and H. P. Wynn. Maximum entropy sampling. *Journal of Applied Statistics*, 14(2):165–170, January 1987.
48. Abhinav Shrivastava, Abhinav Gupta, and Ross B. Girshick. Training region-based object detectors with online hard example mining. *CoRR*, abs/1604.03540, 2016.
49. Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962v2*, 2019.
50. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.
51. Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.
52. Da Xu, Yuting Ye, and Chuanwei Ruan. Understanding the role of importance weighting for deep learning. In *International Conference on Learning Representations*, 2021.
53. Huan Xu and Shie Mannor. Robustness and generalization. *Machine Learning*, 86(3):391–423, Mar 2012.