Variance Based Samples Weighting for Supervised Deep Learning

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

Variance Based Samples Weighting for Supervised Deep Learning

Paul Novello[*†] **Gaël Poëtte**[†] **David Lugato**[†] **Pietro Marco Congedo**[*]

Abstract

Machine Learning (ML) aims at approximating functions defined on a measured space with a model. A relevant choice of distribution for the training data set can improve the performances of a given ML model. We claim and empirically justify that an ML model yields better results when the data set focuses on regions where the function to learn is steeper. We first traduce this assumption in a mathematically workable way. Then, theoretical derivations allow to construct a methodology that we call Variance Based Samples Weighting (VBSW). VBSW uses local variance of the labels to weight the training points. This methodology is general, scalable and cost effective. It is validated on Deep Learning models like Bert [10] or ResNet [14] by significantly increasing their performances for various Natural Language Processing and image classification tasks.

1 Introduction

Supervised Learning consists in learning a mapping from input points to output values, using a Machine Learning model. The model is then used on new data to perform predictions. Approximation theory is one of the approaches that can be used to tackle the learning problem. In that case, it can be formalized as approximating a function $f : \mathbf{S} \subset \mathbb{R}^{n_i} \to \mathbb{R}^{n_o}$ with an ML model f_θ parametrized by θ, where $\mathbf{S}$ is a measured a sub-space which depends on the application. Parameters θ have to be found in order to minimize an integrated loss function $J(\theta)$. In supervised learning, we are given a training data set of N points, $\{X_1, ..., X_N\} \in \mathbf{S}$, drawn from $X \sim d\mathbb{P}_X$ and their point-wise values $\{f(X_1), ..., f(X_N)\}$. These points allow to statistically estimate $J(\theta)$ by $\widehat{J}(\theta)$ and then to use optimization algorithms to find a minimum of $\widehat{J}(\theta)$ w.r.t. θ.

In such problem, the distribution of the data set, $d\mathbb{P}_X$, can have a strong impact on the performances of the model. That is why the challenge of finding a good distribution appears in all fields that involve data, especially Machine Learning (ML). More specifically, in the context of Deep Learning (DL), several works have hinted at the importance of the training set. In [6], 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 will make a Deep Neural Network (DNN) achieve better results in a learning task. With a different approach, Active Learning [33] aims at modifying dynamically the distribution of the training data, 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. Nonetheless, this challenge is not specific to ML. Indeed, Importance Sampling used in the context of statistical Monte Carlo estimation of a quantity of interest owes its efficiency to the construction of a second random variable, $\bar{X} \sim d\mathbb{P}_{\bar{X}}$ that will be used instead of $X \sim d\mathbb{P}_X$ to estimate this quantity. In, [18], the authors even make a connection between the success of likelihood ratio policy gradients and importance sampling, which shows that ML and statistical estimation are closely linked.

[*]INRIA Paris Saclay, {paul.novello,pietro.congedo}@inria.fr

[†]CEA CESTA, {paul.novello, gael.poette, david.lugato}@cea.fr

Preprint. Under review.

We exploit this idea in supervised ML. We build a distribution $d\mathbb{P}_{\bar{X}}$ based on the observation that a model f_θ needs to focus on the regions where f is steep and irregular. Then, we construct a general methodology applicable to any loss function based supervised learning problem. This methodology stands out from the other works in the way the distribution is constructed and exploited and can be used in addition to them. This point is discussed in the positioning and related works (section 7).

Contributions: first, we derive a Generalization Bound (GB) which helps to understand why focusing on the regions where f is steep can yield better results in the approximation task. We then progressively construct a metric that will be used to characterize $d\mathbb{P}_{\bar{X}}$. This leads to a new methodology, Variance Based Samples Weighting (VBSW), that preprocesses the training data points prior to the optimization. This method is general, scalable, cost effective, and applicable to any supervised ML problem that involves a loss function. We focus on DL problems, where it only requires to train a simple linear model and to plug it on top of an already trained DNN. Finally, VBSW is validated on MNIST and Cifar10 by improving the accuracy of a ResNet20 [14] and on several Natural Language Processing (NLP) tasks by improving Bert [10] results for various metrics.

2 The importance of the training set

In this section, we discuss why a ML model may need more points where f is steep. To this end, we use the derivatives of f. This allows to explain it theoretically, by deriving a GB and empirically, by constructing an algorithm that we call Taylor Based Sampling (TBS). These results justify the importance of the derivatives in the approximation problem.

2.1 A derivative based generalization bound

In the following, we show that it is possible to express a GB that includes 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 $J_G(\theta) = J(\theta) - \widehat{J}(\theta)$ thus needs to be as small as possible, where

$$J(\theta) = \int_{\mathbf{S}} L(f_\theta(x), f(x)) d\mathbb{P}_X \approx \widehat{J}(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(f_\theta(X_i), f(X_i)),$$

with L a loss function, $L : \mathbb{R}^{n_o} \times \mathbb{R}^{n_o} \to \mathbb{R}$. 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 $X_i \in S_i$. Then,

$$J(\theta) = \sum_{i=1}^{N} \int_{S_i} L(f_\theta(x), f(x)) d\mathbb{P}_X.$$

Suppose, for now, that L is the squared L_2 error, $n_i = 1$ and f is differentiable. Supposing that f_θ is K_θ-Lipschitz and using Taylor approximation, provided that $|S_i| < 1$, we show (see **Appendix B**)

$$J_G(\theta) \leq \sum_{i=1}^{N} (|f'(X_i)| + K_\theta)^2 \frac{|S_i|^3}{4} + O(|S_i|^4), \tag{1}$$

where $|S_i|$ is the volume of S_i. We see that on the regions where $f'(X_i) + K_\theta$ is higher, quantity $|S_i|$ 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 involve K_θ, the Lipschitz constant of the DNN, which has the same impact than $f'(X_i)$. It also illustrates the link between the Lipschitz constant and the generalization error, which has been pointed out by several works like, for instance, [13], [3] and [31]. We will build an algorithm using this *a priori* knowledge.

2.2 A Taylor expansion based metric

In this paragraph, we build a metric involving the derivatives of f. Using the Taylor approximation at order n on f and supposing that f is n times differentiable (multi index notation):

$$f(x + \boldsymbol{\epsilon}) \underset{\|\boldsymbol{\epsilon}\| \to 0}{=} \sum_{0 \leq |\boldsymbol{k}| \leq n} \boldsymbol{\epsilon}^{\boldsymbol{k}} \frac{\partial^{\boldsymbol{k}} f(x)}{\boldsymbol{k}!} + O(\boldsymbol{\epsilon}^{\boldsymbol{n}}). \tag{2}$$

Quantity $f(x+\boldsymbol{\epsilon})-f(x)$ gives an indication on how much f changes around x. By neglecting the orders above $\boldsymbol{\epsilon}^{\boldsymbol{n}}$ in (2), it is then possible to find the regions of interest by focusing on $Df^n_{\boldsymbol{\epsilon}}$, given by

$$Df^n_{\boldsymbol{\epsilon}}(x) = \sum_{1\leq|\boldsymbol{k}|\leq n} \frac{\|\boldsymbol{\epsilon}\|^k \cdot \|\operatorname{Vect}(\partial^{\boldsymbol{k}} f(x))\|}{\boldsymbol{k}!}. \tag{3}$$

Where $\operatorname{Vect}(\mathbf{X})$ denotes the vectorial form of a tensor $\mathbf{X}$ and $\|.\|$ the squared L_2 norm. Note that $Df^n_{\boldsymbol{\epsilon}}$ is evaluated using $\|\partial^{\boldsymbol{k}} f(X)\|$ instead of $\partial^{\boldsymbol{k}} f(X)$ for derivatives not to cancel each other. f will be steeper and more irregular in the regions where $X \to Df^n_{\boldsymbol{\epsilon}}(X)$ is higher. Note that (1) only gives indications about $n=1$. For $n>1$, the derivations for the GB were intractable but in practice, taking higher orders into account is feasible and more efficient.

To focus the training set on these regions, one can use $\{Df^n_{\boldsymbol{\epsilon}}(X_1),...,Df^n_{\boldsymbol{\epsilon}}(X_N)\}$ to construct a probability density function (pdf) and sample new data points from it. This sampling, that we call Taylor Based Sampling (TBS), is evaluated and validated in **Appendix A** for conciseness. The validation experiments are the approximation of the solution of an ODE system, Runge function and hyperbolic tangent function. The good results obtained with TBS confirm our observation and motivate its application to more complex DL problems.

Note that for TBS and in the following, we chose $n=2$, i.e. we use $\{Df^2_{\boldsymbol{\epsilon}}(X_1),...,Df^2_{\boldsymbol{\epsilon}}(X_N)\}$.

3 From TBS to VBSW

TBS cannot always be applied as is. The obstacles for exploiting this approach in any ML problems rely on the fact that we do not have access to f. **Problem 1:** $\{Df^2_{\boldsymbol{\epsilon}}(X_1),...,Df^2_{\boldsymbol{\epsilon}}(X_N)\}$ cannot be evaluated since it requires to compute the derivatives of f, when f is differentiable, which is often not true. **Problem 2:** even if $\{Df^2_{\boldsymbol{\epsilon}}(X_1),...,Df^2_{\boldsymbol{\epsilon}}(X_N)\}$ could be computed and $\{\bar{X}_1,...,\bar{X}_{N'}\}$ sampled, $\{f(\bar{X}_1),...,f(\bar{X}_{N'})\}$ could not be obtained to complete the training data set.

The following two parts are dedicated to the generalization of TBS. First, we construct a statistical metric related to $Df^n_{\boldsymbol{\epsilon}}$ that does not directly depends on f or its derivatives (**problem 1**). Second, we use a change of variable to simulate $d\mathbb{P}_{\bar{X}}$ using $d\mathbb{P}_X$ (**problem 2**).

3.1 From Taylor approximation to local variance

To overcome **problem 1**, we construct a 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. Let $\boldsymbol{\epsilon}\sim\mathcal{N}(0,\epsilon\mathbb{I}_{n_i})$ with $\epsilon\in\mathbb{R}^+$ and $\mathbb{I}_{n_i}$ the identity matrix of dimension n_i. We claim that

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

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

$$\widehat{Df^2_{\boldsymbol{\epsilon}}}(x) = \frac{1}{k-1}\sum_{i=1}^{k}\Big(f(x+\boldsymbol{\epsilon_i})-f(x)\Big)^2, \tag{4}$$

with $\{\boldsymbol{\epsilon}_1,...,\boldsymbol{\epsilon}_k\}$ k samples of $\boldsymbol{\epsilon}$. Note that $\widehat{Df^2}_{\epsilon}(x) \underset{k\to\infty}{\to} Var(f(x+\boldsymbol{\epsilon}))$. Since $Var(f(x+\boldsymbol{\epsilon})) = Df^2_{\boldsymbol{\epsilon}}(x)+O(|\epsilon|_2^3)$, $\widehat{Df^2}_{\epsilon}(x)$ is a biased estimator of $Df^2_{\boldsymbol{\epsilon}}(x)$, with bias $O(|\epsilon|_2^3)$. Hence, when $\epsilon\to 0$, $\widehat{Df^2}_{\epsilon}(x)$ becomes an unbiased estimator of $Df^2_{\boldsymbol{\epsilon}}(x)$.

This definition brings several advantages, based on the fact that the derivatives of f are no longer required. First, $\widehat{Df^2}$ can even be applied to non differentiable functions. Second, with (4), it is possible to compute $\widehat{Df^2}(x)$ from any set of points centered around x. Therefore, we compute $\widehat{Df^2}(X_i)$ for each $i\in\{1,...,N\}$ using the k-nearest neighbors of X_i. We obtain $\widehat{Df^2}(X_i)$ using

$$\widehat{Df^2}(X_i) = \frac{1}{k-1}\sum_{X_j\in\mathcal{S}_k(X_i)}\Big(f(X_j)-\frac{1}{k}\sum_{l=1}^{k}f(X_l)\Big)^2, \tag{5}$$

with $\mathcal{S}_k(X) = \{X_j, \|X_j - X\| < \|X_l - X\|, X_l \in (X_1, ..., X_N) \backslash \mathcal{S}_k(X)\}$, $\text{card}(\mathcal{S}_k(X)) = k$. The advantage of this formulation is that all that we need is $\{f(X_1), ..., f(X_N)\}$. In other words, the points used by $\widehat{Df^2}(X)$ are those used for the training of the DNN. Finally, while the definition of $Df^2_\epsilon(x)$ is local, the definition of $\widehat{Df^2}_\epsilon(x)$ holds for any ϵ. In the - very frequent - case where the data points are too sparse for the nearest neighbors of X_i to be considered as close to X_i, eq. (5) can still be applied. It can thus be seen as a generalization of $Df^2_\epsilon(x)$, which tends towards $Df^2_\epsilon(x)$ locally.

3.2 From sampling to weighting

In this part, we tackle **problem 2** using a probabilistic formulation of the DL optimization problem. Recall that f_θ has to be chosen in order to minimize a loss function $J(\theta)$, estimated using the training points $\{X_1, ..., X_N\}$ and their point-wise values $\{f(X_1), ..., f(X_N)\}$. The goal is to find $\theta^* = \underset{\theta}{\text{argmin}} \hat{J}(\theta)$, with $\widehat{J}(\theta) = \frac{1}{N} \sum_i L(f(X_i), f_\theta(X_i))$.

With TBS, the procedure is different. Since the training points are sampled using $Df^2_{\boldsymbol{\epsilon}}$, we no longer minimize $\widehat{J}(\theta)$, but $\widehat{J_{TBS}}(\theta) = \frac{1}{N} \sum_i L(f(\bar{X}_i), f_\theta(\bar{X}_i))$, with $\bar{X} \sim d\mathbb{P}_{\bar{X}}$ the distribution of TBS. However, $\widehat{J_{TBS}}(\theta)$ estimates

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

Let $p_X = \frac{d\mathbb{P}_X}{dx}$ and $p_{\bar{X}} = \frac{d\mathbb{P}_{\bar{X}}}{dx}$ be the pdf of X and $\bar{X}$ respectively (note that $Df^2_{\boldsymbol{\epsilon}} \propto p_{\bar{X}}$). Then,

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

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

$$\widehat{J_{TBS,2}}(\theta) = \frac{1}{N} \sum_i L(f(X_i), f_\theta(X_i)) \frac{p_{\bar{X}}(X_i)}{p_X(X_i)} \propto \frac{1}{N} \sum_i L(f(X_i), f_\theta(X_i)) \frac{Df^2_{\boldsymbol{\epsilon}}(X_i)}{p_X(X_i)}. \quad (6)$$

It is therefore possible to estimate $J_{TBS}(\theta)$ with the same points as $J(\theta)$. All we have to do is weighting these points by $w_i = \frac{Df^2_{\boldsymbol{\epsilon}}(X_i)}{p_X(X_i)}$.

4 Variance Based Samples Weighting (VBSW)

There are still two concerns before being able to construct an end to end methodology for general use in ML. First, the distribution p_X is not known so (6) can not be used as is. Second, the dimension of the input space can be extremely high (e.g. in Computer Vision), which makes the application of (5) non trivial. In this section, we tackle these concerns to construct the final methodology, VBSW.

4.1 Construction of VBSW

The expression of w_i involves $Df^2_{\boldsymbol{\epsilon}}(X_i)$, whose estimation has been the goal of the previous sections. However, it also involves p_X, the distribution of the data. Just like for f, we do not have access to p_X. The estimation of p_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 $p_X(X_i)$ in our experiments. Therefore, we decided to only apply $\omega_i = \widehat{Df^2}(X_i)$ as a first order approximation. In practice, we re-scale the weighting points to be between 1 and m, a hyperparameter. As a result, VBSW has two hyperparameters: m and k. These choices are tested on a Double Moon binary classification problem and in the Boston Housing regression data set.

For double moon, Figure 1 (c) shows that the points with highest w_i (in red) are close to the boundary between the two classes. 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 baseline where no weights are applied. Figure 1 (b) and (d) displays the decision boundary of best fit for each methods. VBSW provides a cleaner decision boundary than baseline. These pictures as well as the results of Table 1 show the improvement obtained with VBSW.

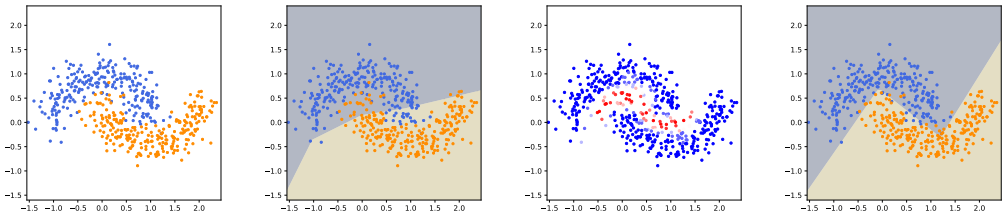

Figure 1: From left to right: **(a)** Double Moon (DM) data set. **(b)** Decision boundary with the baseline method. **(c)** Heat map of the value of w_i for each X_i (red is high and blue is low) and **(d)** Decision boundary with VBSW method

For Boston housing data set, a linear model trained with a ADAM [20] is used. Since this data set is small and the model really light, we study the effects of the choice of m and k on the performances. We train a linear model for a grid of 20×20 different values of m and k. These hyperparameters choice have an impact on the performances but no pattern or tendency seem to appear from these results so they should be chosen with a classical grid search, like any other hyperparameter. Details and visualization of this experiment can be found in **Appendix C**. The best result obtained with this study is compared to the best result of a linear model trained without VBSW in Table 1.

	VBSW	baseline
DM	**99.4**, **94.44** ± 0.78	99, 92.06 ± 0.66
BH	**13.31**, **13.38** ± 0.01	14.05, 14.06 ± 0.01

Table 1: **best, mean + se** for each method. The metric used is accuracy for Double Moon (DM) and Mean Squared Error for Boston Housing data set (BH).

4.2 VBSW for Deep Learning

We specified that the local variance could be computed using already existing points. This statement implies to find the nearest neighbors of each point. To compute the local variances, in the previous experiment, we used a KD-tree [7] data structure. It allows us to query the k nearest neighbors, with k the number of data points used to estimate the variance. Let N be the number of points and n_i their dimension, the complexity of the construction of KD-tree is $O(N(n_i + log(N)))$, which is acceptable since it is constructed once. However, in too high dimension, the cost of a search suffers from the curse of dimensionality. Moreover, in extremely high dimension spaces like image spaces, the structure of the data may be highly irregular, and the concept of nearest neighbor misleading. Thus, it may be irrelevant to evaluate $\widehat{D^2 f_\epsilon}$ directly on this data.

One of the strength of DL is to construct good representations of the data, embedded in lower dimensional latent spaces. For instance, in Computer Vision, Convolutional Neural Networks (CNN)'s deeper layers represent more abstract features. We could leverage this representational power of DNNs, and simply apply our methodology within its latent space.

Algorithm 1 Variance Based Samples Weighting (VBSW) for Deep learning

1: **Inputs:** k, m, $\mathcal{M}$
2: Train $\mathcal{M}$ on the training set $\{(\frac{1}{N}, X_1), ..., (\frac{1}{N}, X_N)\}$, $\{(\frac{1}{N}, f(X_1)), ..., (\frac{1}{N}, f(X_N))\}$
3: Construct $\mathcal{M}^*$ by removing its last layer
4: Compute $\{\widehat{Df^2}(\mathcal{M}^*(X_1)), ..., \widehat{Df^2}(\mathcal{M}^*(X_N))\}$ using (5) and a KD-tree.
5: Construct a new training data set $\{(w_1, \mathcal{M}^*(X_1)), ..., (w_N, \mathcal{M}^*(X_N))\}$
6: Train f_θ on $\{(w_1, f(X_1)), ..., (w_N, f(X_N))\}$ and add it to $\mathcal{M}^*$. The final model is $\mathcal{M}_f = f_\theta \circ \mathcal{M}^*$

Variance Based Samples Weighting (VBSW) for DL is recapitulated in Algorithm 1. Here, $\mathcal{M}$ is the initial DNN whose latent 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 .**Line 2:** The initial DNN, $\mathcal{M}$, is trained as usual. Notations $\{(\frac{1}{N}, X_1), ..., (\frac{1}{N}, X_N)\}$ is equivalent to $\{X_1, ..., X_N\}$, 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 latent space. **Line 4-5:** (5) 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.

5 Validation

We test this methodology on several data sets, in image classification and NLP, namely MNIST [25], Cifar10 [22] and some of the glue benchmark data sets [37] (RTE, STS-B and MRPC). We use respectively LeNet 5 [24], ResNet20 [14], two CNNs and Bert [10] a DNN based on bi-directional Transformers [36].

5.1 MNIST and Cifar10

For MNIST, we train 40 LeNet 5, i.e. with 40 different random seeds, and then apply VBSW for 10 different random seeds, with ADAM optimizer and categorical cross-entropy loss. We record the best value obtained from the 10 VBSW training. The same procedure is followed for Cifar10, except that we train a ResNet20 for 50 random seeds and with data augmentation. The networks have been trained on 4 Nvidia K80 GPUs. The values of the hyperparameters used can be found in **Appendix C**. 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** ± 0.01	98.99, 98.84 ± 0.01	**0.15**, **0.03** ± 0.01
Cifar10	**91.30**, **90.64** ± 0.07	91.01, 90.46 ± 0.10	**1.65**, **0.15** ± 0.04

Table 2: **best, mean + se** for each method. The metric used is accuracy. Gain per model is not to be confused with the difference between the statistics of VBSW and baseline.

The results statistics are gathered in Table 2, which also displays statistics about the gain due to VBSW for each model. Given 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 on MNIST, for all statistics and for the gain are significantly better than baseline. 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% by VBSW. Note that applying VBSW took less than 15 minutes on a laptop with an i7-7700HQ CPU. A visualization of the samples that were weighted by the highest w_i is given in Figure 2.

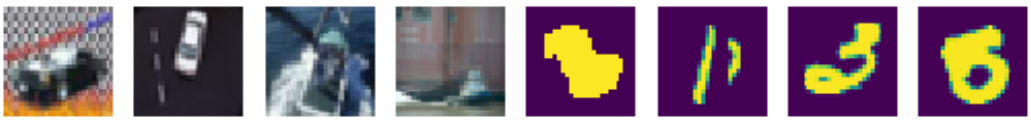

Figure 2: Samples from Cifar10 and MNIST with high w_i. Those pictures are either unusual or difficult to classify, even for a human (especially for MNIST).

5.2 Glue: RTE, STS-B and MRPC

For this application, we did not train Bert DNN, like in the previous experiments, since its purpose is to be used as is and then fine-tuned on any NLP data set. However, because of the small size of the dataset 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 latent space to apply a linear model. More specifically, we use tiny-bert [35], which is a lighter version of the initial Bert DNN. 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 model. 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 really better overall, except for Pearson Correlation in STS-B, which is slightly worse than baseline (Table 3).

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

Table 3: **best, mean + se** for each method. For RTE the metric used is accuracy (m1). For MRPC, metric 1 (m1) is accuracy and metric 2 (m2) is F1 score. For STS-B, metric 1 (m1) is Spearman correlation and metric 2 (m2) is Pearson correlation.

6 Discussion

VBSW shows improvement on the performances of these models. However, this method has to be applied carefully: a bad choice of m and k can lead to a decreased accuracy. They have to be considered as hyperparameters to optimize (they have been considered as such in our experiments). In addition, we first approximated p_X to be uniform, because we could not approximate it correctly. This still led to a an efficient methodology, but VBSW may benefit from a finer approximation of p_X. Improving the approximation of p_X is among our perspectives. Finally, the KD-tree construction struggles when the data set is too big, since the complexity of a query using the KD-tree is linear with the number of samples.

In addition to its performances boost, the advantages of VBSW are twofold. First, it validates an original view of the learning problem, that involves the local variations of f. Indeed, we started from the Taylor approximation, which is specific to derivable functions, and managed to derive a general methodology whose validation justifies this view. Second, the problem of high dimensionality and irregularity of f is alleviated by focusing on the latent space of DNN, which makes VBSW scalable. Indeed, the range of possible applications of VBSW goes from a simple linear regression to complex models such as ResNet, a very deep and complex CNN or Bert, a model based on bi-directional Transformers [36]. In theory, even if we focused on DL, it could be applied to any loss function optimization based ML model, like we did with the Boston Housing and Double Moon data sets. Its broadening to other models for larger scale applications is a direction for future works.

We only considered the cases where we have not access to f. However, there are ML applications where we do. For instance, in computationally intensive numerical simulations, for physical sciences, economics or climatology, some models are run on supercomputers and can take days to perform a prediction. Speeding up these models is a research area on its own, that gathers computer scientists, physicians and mathematicians. This speed up is important to obtain faster predictions, but also to conduct statistical analyses of the results. One way of achieving this is to use surrogate models, i.e. lighter, cost effective models that approximate the heavier, expansive model. These surrogate models are often based on learning an approximation of the full model using data points, and DNN are more and more used for that purpose. See for example works of [40], [38] or [11]. Yet, the data points being computed using the full model, it can be costly to obtain a sufficiently large data set. Our approach could be applied by sampling points from the information given by $\widehat{Df^2}$, to get more informative data and improve the accuracy of the DNN for a same training data set size.

7 Positioning and Related Works

Active Learning - Our methodology is based on the consideration that not every sample bring the same amount of information. Active learning (AL) exploits the same idea, in the sense that it adapts the training strategy to the problem by introducing a data point selection rule. In [12], the authors introduce a methodology based on Bayesian Neural Networks (BNN) to adapt the selection of points used for the training. Using the variational properties of BNN, they design a rule to focus the training on points that will reduce the prediction uncertainty of the DNN. In [21], the construction of the selection rule is taken as a ML problem itself. They formulate it as a regression to predict the error reduction of each point, based on the experience of training on synthetic data. See [33] for a review of more classical AL methods. The similarity between AL and VBSW goes beyond adapting the training to the data. Indeed, AL selects the data points, so modifies the distribution of the initial training data set. As a result, some points will have been used more often than others during the training, leading to a similar effect than weighting these points in the loss function. The main difference is that VBSW is prior to the training, and therefore the distribution of the weights can not change throughout the training. In our work, the decision rule is based on $\widehat{Df^2}$, which is fixed, unlike in [12] and [21] where

the decision rule is updated regularly. Our methodology could be applied before the training, and AL used during the training. Hence, these two approaches are not orthogonal.

Examples Weighting - VBSW can be categorized as an example weighting algorithm. The idea of weighting the data set has already been explored in different ways and for different purposes. Some of these works exploit the idea that some examples are easy to learn, and others are harder. In curriculum learning [6] and [27], a distinction is made between examples, and just like humans need to learn from easy examples first and then to focus on harder examples, a curriculum is designed for the training of DNN, leading to better results. Self paced learning [23], [17] builds on the same idea by downscaling harder examples. However, some works have proven that focusing on harder examples from the beginning of the learning could lead to faster training. In [34], hard example mining is performed to give more importance to harder examples by selecting them primarily. Example weighting is used in [9] to tackle the class imbalance problem by weighting rarer, and so harder examples. At the contrary, in [26] it is used to solve the noisy labels problem by focusing on cleaner, so easier examples. All these ideas show that depending on the application, example weighting can be performed in an opposing manner. Some works aim at moving beyond this opposition by introducing more general methodologies. In [8], the authors use the variance of the prediction of a point by f_θ throughout the training to decide whether it should be weighted or not. A meta learning approach is proposed in [32], where the authors choose the weights after an optimization nested in the training. VBSW stands out from the previously mentioned example weighting methods because it is built on a more general claim that a model will simply need more points to learn more complicated functions. Its effect is to boost the performances of a DNN, whatever the application is.

Importance Sampling - Some of the previous methods use importance sampling to design the weights of the data set or to correct the bias induced by the sample selection [19]. To obtain the expression (6), we perform a change of variable similar to importance sampling. However, it is not used for the same reasons. Here, the importance distribution chosen is voluntarily sub optimal. Indeed, it is the distribution of the training data set $d\mathbb{P}_X$, which is used to estimate a loss function integrated over the (inaccessible) distribution $d\mathbb{P}_{\tilde{X}}$.

Generalization Bound - Generalization bound approaches of the learning theory of DNN have motivated many works, most of which are reviewed in [16]. In [5], [4], the authors focus on VC-dimension, a measure which depends on the number of parameters of DNNs. [2] introduces a compression approach that aims at reducing the number of model parameters to investigate its generalization capacities. PAC-Bayes analysis constructs generalization bounds using *a priori* and *a posteriori* distributions over the possible models. In [29], the authors build on the work in [3] and adds PAC-Bayes analysis in the computation of a bound based on the Lipschitz constant of f_θ. PAC-Bayes theory is linked to the notion of sharpness of a DNN, i.e. its robustness to small perturbation in [28]. This robustness approach, to study generalization of DNNs is investigated in [39]. While sharpness of the model is often mentioned in the previous work, our generalization bound includes another element which can be seen as the sharpness of f, the function to approximate. 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. Indeed, for instance, it includes the derivatives of f, which does not always exist. But it emphasizes the intuition that the DNN will need more points to capture sharper functions. It is similar to the concept of robustness as well, but while existing works focus on the robustness to small perturbation in the parameter space, ours investigates the robustness to perturbations in the input space.

8 Conclusion

Our work is based by the observation that, in supervised learning, a function f is more difficult to approximate by a ML model in the regions where it is steeper. We mathematically traduced this intuition and derived a generalization bound to justify it. Then, we constructed an original method, Variance Based Samples Weighting (VBSW), that uses the variance of the training samples to weight the training data set and boosts the model performances. VBSW is simple to use and to implement, because it only requires to compute statistics on the input space. In Deep Learning, applying VBSW on the data set projected in the latent space of an already trained DNN allows to boost its performance by simply re training its last layer. This method is applicable to any loss function based supervised learning problem, scalable, cost effective, and validated on several applications such as glue benchmark with Bert, for NLP and Cifar10 with a ResNet20, for image classification.

Appendix A: Taylor Based Sampling (TBS)

First, we recall the formula for the $\{Df_{\boldsymbol{\epsilon}}^n(X_1), ..., Df_{\boldsymbol{\epsilon}}^n(X_N)\}$.

$$Df_{\boldsymbol{\epsilon}}^n(x) = \sum_{1 \leq |\boldsymbol{k}| \leq n} \frac{\|\boldsymbol{\epsilon}\|^k \cdot \| \operatorname{Vect}(\partial^{\boldsymbol{k}} f(x))\|}{\boldsymbol{k}!}. \tag{7}$$

To focus the training set on the regions of interest, i.e. regions of high $\{Df_{\boldsymbol{\epsilon}}^n(X_1), ..., Df_{\boldsymbol{\epsilon}}^n(X_N)\}$, we use these metrics to construct a probability density function (pdf). This is possible since $Df_{\boldsymbol{\epsilon}}^n(x) \geq 0$ for all $x \in \mathbf{S}$. It remains to normalize it but in practice it is enough considering a distribution $d \propto Df_{\boldsymbol{\epsilon}}^n$. Here, to approximate d we use a Gaussian Mixture Model (GMM) with pdf d_{GMM} that we fit to $\{Df_{\boldsymbol{\epsilon}}^n(X_1), ..., Df_{\boldsymbol{\epsilon}}^n(X_N)\}$ using the Expectation-Maximization (EM) algorithm. N' new data points $\{\bar{X}_1, ..., \bar{X}_{N'}\}$, can be sampled, with $\bar{X} \sim d_{\text{GMM}}$. Finally, we obtain $\{f(\bar{X}_1), ..., f(\bar{X}_{N'})\}$, add it to $\{f(X_1), ..., f(X_N)\}$ and train our DNN on the whole data set.

Taylor Based Sampling (TBS)

Our methodology, which we call Taylor Based Sampling (TBS) is described in Algorithm 2. **Line 1:** The choice criterion of $\boldsymbol{\epsilon}$, the number of Gaussian distribution n_{GMM} and N' is to avoid sparsity of $\{\bar{X}_1, ..., \bar{X}_{N'}\}$ 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 $\{Df_{\boldsymbol{\epsilon}}^n(X_1), ..., Df_{\boldsymbol{\epsilon}}^n(X_N)\}$, and then d 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 2 Taylor Based Sampling (TBS)

1: **Inputs:** $\boldsymbol{\epsilon}$, N, N', n_{GMM}, n
2: Sample $\{X_1, ..., X_N\}$, with $X \sim \mathcal{U}(\mathbf{S})$
3: **for** $0 \leq k \leq n$ **do**
4: Compute $\{\partial^{\boldsymbol{k}} f(X_1), ..., \partial^{\boldsymbol{k}} f(X_N)\}$
5: **end for**
6: Compute $\{Df_{\boldsymbol{\epsilon}}^n(X_1), ..., Df_{\boldsymbol{\epsilon}}^n(X_N)\}$ using (7)
7: Approximate $d \sim D_{\boldsymbol{\epsilon}}$ with a GMM using EM algorithm to obtain a density d_{GMM}
8: Sample $\{\bar{X}_1, ..., \bar{X}_{N'}\}$ using rejection method to sample inside $\mathbf{S}$
9: Compute $\{f(\bar{X}_1), ..., f(\bar{X}_{N'})\}$
10: Add $\{f(\bar{X}_1), ..., f(\bar{X}_{N'})\}$ to $\{f(X_1), ..., f(X_N)\}$

Application to simple functions

In order to illustrate the benefits of Taylor based sampling (TBS) compared to an uniform, basic sampling (BS), we apply it to two simple functions: hyperbolic tangent and Runge function. We chose these functions because they are differentiable and have a clear distinction between flat and steep regions. These functions are displayed in Figure 3, as well as the map $x \to Df_{\boldsymbol{\epsilon}}^n(x)$.

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

Table 4: Comparison between BS and TBS. The metrics used are the L_2 and L_∞ errors, displayed with a 95% confidence interval

All DNN have been implemented in `Python`, with `Tensorflow` [1]. We use the Python package `scikit-learn` [30], and more specifically the `GaussianMixture` class to construct d_{GMM}. The network chosen for this experiment is a Multi Layer Perceptron (MLP) with one layer of 8 neurons, with Relu activation function, that we trained alternatively with BS and TBS using Adam optimizer [20] with the defaults tensorflow implementation hyperparameters, and Mean Squared Error loss function. We first sample $\{X_1, ..., X_N\}$ 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

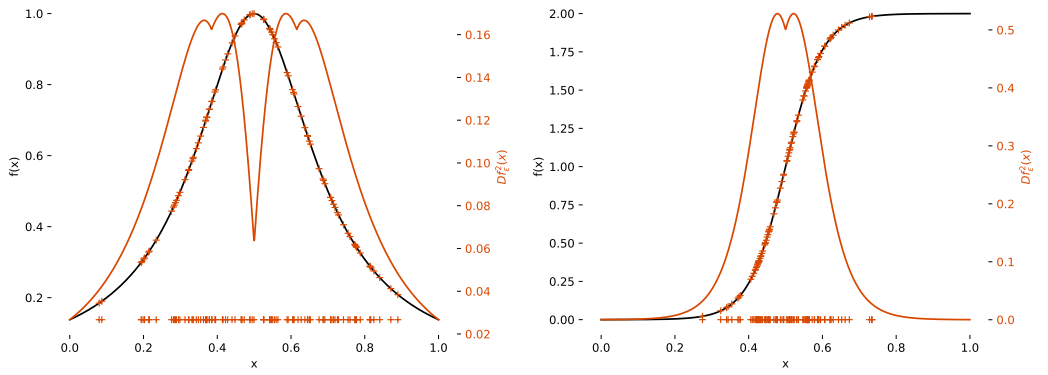

Figure 3: **Left:** (left axis) Runge function w.r.t x and (right axis) $x \to Df^2_{\boldsymbol{\epsilon}}(x)$. Points sampled using TBS are plotted on the x-axis and projected on f. **Right:** Same as left, with hyperbolic tangent function.

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 $\boldsymbol{\epsilon}$, and finally selected $n = 2$, $n_{\text{GMM}} = 3$ and $\boldsymbol{\epsilon} = 10^{-3}$.

Table 4 summarizes the L_2 and the L_∞ norm of the error of f_θ, obtained at the end of the training phase for $N + N' = 16$, with $N = 8$. Those norms are estimated using a 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 4 shows that TBS slightly degrades the L_2 error of the DNN, but improves its L_∞ error. This may explain the good results of VBSW for classification. Indeed, for a classification task, the accuracy will not be very sensitive to small output variations, since the output is rounded to 0 or 1. However, a high error can induce a misclassification, and the reduction in L_∞ norm limits this risk.

Application to an ODE system

We apply TBS to a more realistic case: the approximation of the resolution of the Bateman equations, which is an ODE system :

$$\begin{cases} \partial_t u(t) &= v\boldsymbol{\sigma_a} \cdot \boldsymbol{\eta}(t)u(t), \\ \partial_t \boldsymbol{\eta}(t) &= v\boldsymbol{\Sigma_r} \cdot \boldsymbol{\eta}(t)u(t), \end{cases} \text{, with initial conditions } \begin{cases} u(0) = u_0, \\ \boldsymbol{\eta}(0) = \boldsymbol{\eta_0}. \end{cases}$$

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. We consider the particular case $M = 1$ so that $f : \mathbb{R}^3 \to \mathbb{R}^2$, with $f(u_0, \eta_0, t) = (u(t), \eta(t))$. 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 DNN trainings have been performed in `Python`, with `Tensorflow` [1]. We used a fully connected DNN 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 first trained the model for $(u(t), \eta(t)) \in \mathbb{R}$, with an uniform sampling, that we call basic sampling (BS) $(N' = 0)$, and then with TBS for several values of n, n_{GMM} and $\boldsymbol{\epsilon} = \epsilon(1, 1, 1)$, to be able to find good values. 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. This experiment has been repeated 50 times to ensure statistical significance of the results.

Table 5 summarizes the MSE, i.e. the L_2 norm of the error of f_θ and L_∞ norm, with $L_\infty(\theta) = \max_{X\in\mathbf{S}}(|f(X) - f_\theta(X)|)$ 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.

Table 5: Comparison between BS and TBS

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	$\mathbf{1.14} \pm 0.15$	$\mathbf{2.96} \pm 0.37$	2.51 ± 0.07	0.42 ± 0.008

Figure 1a shows how the DNN can perform for an average prediction. **Figure 1b** 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 2a** displays $u_0, \eta_0 \rightarrow \max_{0 \leq t \leq 10} D^n_{\boldsymbol{\epsilon}}(u_0, \eta_0, t)$ w.r.t. (u_0, η_0) and shows that $D^n_{\boldsymbol{\epsilon}}$ increases when $U_0 \rightarrow 0$. TBS hence focuses on this region. Note that for the readability of this plots, the values are capped to 0.10. Otherwise only few points with high $D^n_{\boldsymbol{\epsilon}}$ are visible. **Figure 2b** 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. The highest error reduction occurs in the expected region. Indeed more points are sampled where $D^n_{\boldsymbol{\epsilon}}$ 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 1**, the average error loss (AEL) of TBS is around six times lower than the the average error gain (AEG), with $AEG = \mathbb{E}_{u_0,\eta_0}(Z \mathbb{1}_{Z>0})$ and $AEL = \mathbb{E}_{u_0,\eta_0}(Z \mathbb{1}_{Z<0})$ 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.

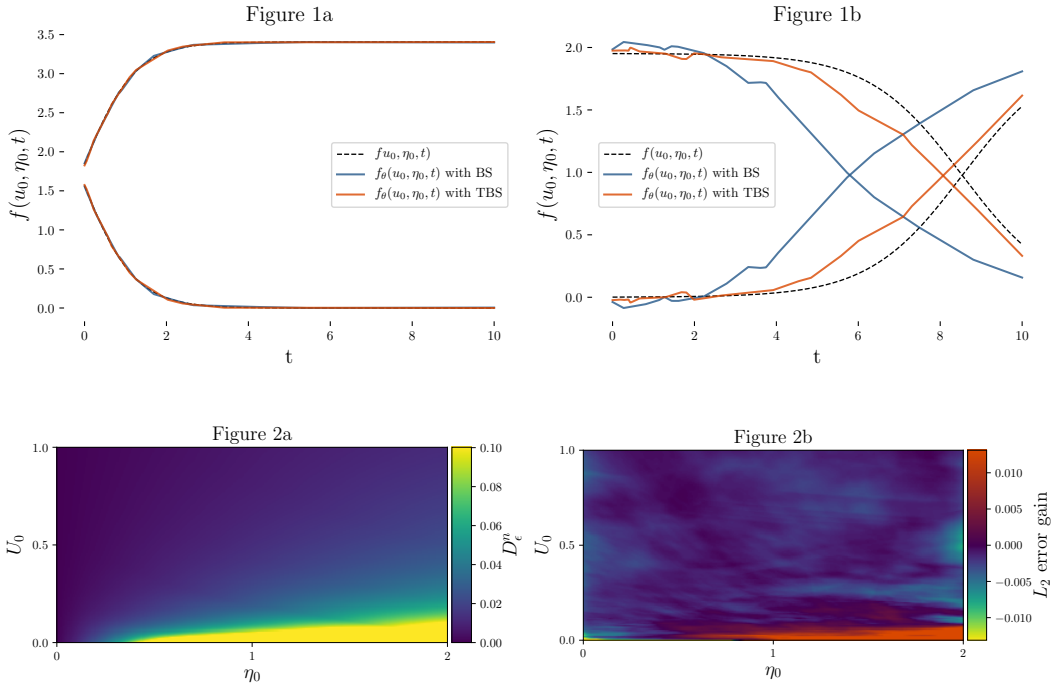

Figure 4: **1a:** $t \rightarrow f_\theta(u_0, \eta_0, t)$ for randomly chosen (u_0, η_0), for f_θ obtained with the two samplings. **1b:** $t \rightarrow f_\theta(u_0, \eta_0, t)$ for (u_0, η_0) resulting in the highest point-wise error with the two samplings. **2a:** $u_0, \eta_0 \rightarrow \max_{0 \leq t \leq 10} D^n_{\boldsymbol{\epsilon}}(u_0, \eta_0, t)$ w.r.t. (u_0, η_0). **2b:** $u_0, \eta_0 \rightarrow g_{\theta_{BS}}(u_0, \eta_0) - g_{\theta_{TBS}}(u_0, \eta_0)$,

Appendix B: Demonstrations

A derivative based generalization bound (2.4)

We want to approximate $f : x \to f(x)$, $x \in \mathbb{R}^{n_i}$, $f(x) \in \mathbb{R}^{n_o}$ with a DNN f_θ. 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 $J_G(\theta)$ to be as small as possible. Given an initial data set $\{X_1, ..., X_N\}$ drawn from $X \sim d\mathbb{P}_X$ and $\{f(X_1), ..., f(X_N)\}$, and the loss function L being the squared L_2 error, recall that the integrated error $J(\theta)$, its estimation $\widehat{J}(\theta)$ and the generalisation error $J_G(\theta)$ can be written:

$$\begin{aligned} J(\theta) &= \int_{\mathbf{S}} \|f(x) - f_\theta(x)\| d\mathbb{P}_X, \\ \widehat{J}(\theta) &= \frac{1}{N} \sum_{i=1}^{N} \big(f_\theta(X_i) - f(X_i)\big)^2, \\ J_G(\theta) &= J(\theta) - \widehat{J}(\theta). \end{aligned} \tag{8}$$

Where $\|.\|$ denotes the squared L_2 norm. In the following, we find an upper bound for $J_G(\theta)$. We start by finding an upper bound for $J(\theta)$ and then for $J_G(\theta)$ using (8).

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 $X_i \in S_i$. Then,

$$\begin{aligned} J(\theta) &= \sum_{i=1}^{N} \int_{S_i} \|f(x) - f_\theta(x)\| d\mathbb{P}_X, \\ J(\theta) &= \sum_{i=1}^{N} \int_{S_i} \|f(X_i + x - X_i) - f_\theta(x)\| d\mathbb{P}_X. \end{aligned}$$

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

$$J(\theta) = \sum_{i=1}^{N} \int_{S_i} \|f(X_i) + f'(X_i)(x - X_i) + \frac{1}{2} f''(X_i)(x - X_i)^2 - f_\theta(x) + O((x - X_i)^3)\| d\mathbb{P}_X.$$

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

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

$$\begin{aligned} & f_\theta(X_i) - K_\theta|x - X_i| \leq f_\theta(x) \leq f_\theta(X_i) + K_\theta|x - X_i|, \\ & - f_\theta(X_i) - K_\theta|x - X_i| \leq -f_\theta(x) \leq -f_\theta(X_i) + K_\theta|x - X_i|, \\ & f(X_i) + f'(X_i)(x - X_i) + \frac{1}{2} f''(X_i(x - X_i)^2) - f_\theta(X_i) - K_\theta|x - X_i| + O((x - X_i)^3) \\ & \leq A_i(x) \leq f(X_i) + f'(X_i)(x - X_i) + \frac{1}{2} f''(X_i)(x - X_i)^2 - f_\theta(X_i) + K_\theta|x - X_i| + O((x - X_i)^3), \\ & A_i(x) \leq f(X_i) - f_\theta(X_i) + f'(X_i)(x - X_i) + \frac{1}{2} f''(X_i)(x - X_i)^2 + K_\theta|x - X_i| + O((x - X_i)^3). \end{aligned}$$

And finally, using triangular inequality,

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

Now, suppose that $\|.\|$ is the squared L_2 norm. Then,

$$J(\theta) = \sum_{i=1}^{N} \int_{S_i} \|f(X_i) + f'(X_i)(x - X_i) + \frac{1}{2} f''(X_i)(x - X_i)^2 - f_\theta(x) + O(|x - X_i|^3)\| d\mathbb{P}_X,$$

$$J(\theta) \leq \sum_{i=1}^{N} \int_{S_i} \left[\Big(|f(X_i) - f_\theta(X_i)| \Big) + \Big(|f'(X_i)||x - X_i| + \frac{1}{2}|f''(X_i)||x - X_i|^2 + K_\theta |x - X_i| \Big) + O(|x - X_i|^3) \right]^2 d\mathbb{P}_X,$$

$$\begin{aligned} J(\theta) \leq \sum_{i=1}^{N} \int_{S_i} \Bigg[& |f(X_i) - f_\theta(X_i)|^2 + 2|f(X_i) - f_\theta(X_i)| \Big(|f'(X_i)||x - X_i| + \frac{1}{2}|f''(X_i)||x - X_i|^2 + K_\theta |x - X_i| \Big) \\ & + \left[\Big(|f'(X_i)||x - X_i| \Big) + \Big(\frac{1}{2}|f''(X_i)||x - X_i|^2 + K_\theta |x - X_i| \Big) \right]^2 + O(|x - X_i|^3) \Bigg] d\mathbb{P}_X, \end{aligned}$$

$$\begin{aligned} J(\theta) \leq \sum_{i=1}^{N} \int_{S_i} \Bigg[& |f(X_i) - f_\theta(X_i)|^2 + 2|f(X_i) - f_\theta(X_i)| \Big(|f'(X_i)||x - X_i| + \frac{1}{2}|f''(X_i)||x - X_i|^2 + K_\theta |x - X_i| \Big) \\ & + \Big[|f'(X_i)|^2 |x - X_i|^2 \\ & + 2|f'(X_i)||x - X_i| \big(\frac{1}{2}|f''(X_i)||x - X_i|^2 + K_\theta |x - X_i| \big) \\ & + \big(\frac{1}{2}|f''(X_i)||x - X_i|^2 + K_\theta |x - X_i| \big)^2 \Big] \\ & + O(|x - X_i|^3) \Bigg] d\mathbb{P}_X, \end{aligned}$$

$$\begin{aligned} J(\theta) \leq \sum_{i=1}^{N} \int_{S_i} \Bigg[& |f(X_i) - f_\theta(X_i)|^2 + 2|f(X_i) - f_\theta(X_i)| \Big(|f'(X_i)||x - X_i| + \frac{1}{2}|f''(X_i)||x - X_i|^2 + K_\theta |x - X_i| \Big) \\ & + \Big[|f'(X_i)|^2 |x - X_i|^2 + 2K_\theta |f'(X_i)||x - X_i|^2 + K_\theta^2 |x - X_i|^2 \Big] + O(|x - X_i|^3) \Bigg] d\mathbb{P}_X, \end{aligned}$$

$$\begin{aligned} J(\theta) \leq \sum_{i=1}^{N} \int_{S_i} \Bigg[& |f(X_i) - f_\theta(X_i)|^2 + 2|f(X_i) - f_\theta(X_i)| \Big(|f'(X_i)||x - X_i| + \frac{1}{2}|f''(X_i)||x - X_i|^2 + K_\theta |x - X_i| \Big) \\ & + \Big(|f'(X_i)| + K_\theta \Big)^2 |x - X_i|^2 + O(|x - X_i|^3) \Bigg] d\mathbb{P}_X. \end{aligned}$$

Hornik's theorem [15] 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 all ϵ, there exists θ such that $\|f(x) - f_\theta(x)\|_{p,\mu}^p < \epsilon$.

Let $\epsilon_i = O(|S_i|^3)$ for all $i \in \{1, ..., N\}$, $p = 1$ and $d\mu = \frac{1}{N} \sum_{i=1}^{N} \delta(x - X_i)$. Then, $\widehat{J}(\theta) = O(|S_i|^4)$ for all $i \in \{1, ..., N\}$ and $J(\theta) = J_G(\theta) + O(|S_i|^4)$. Hence,

$$J_G(\theta) \leq \sum_{i=1}^{N} \int_{S_i} \Big[(|f'(X_i)| + K_\theta)^2 |x - X_i|^2 d\mathbb{P}_X \Big] + O(|S_i|^4),$$

because $\int_{S_i} \Big[|f(X_i) - f_\theta(X_i)|^2 + 2|f(X_i) - f_\theta(X_i)| \Big(|f'(X_i)||x - X_i| + \frac{1}{2}|f''(X_i)||x - X_i|^2 + K_\theta |x - X_i| \Big) \Big] d\mathbb{P}_X = O(|S_i|^4)$. Finally,

$$\boxed{J_G(\theta) \leq \sum_{i=1}^{N} (|f'(X_i)| + K_\theta)^2 \frac{|S_i|^3}{3} + O(|S_i|^4).} \tag{9}$$

where $|S_i|$ is the volume of S_i. We see that on the regions where $f'(X_i) + K_\theta$ is higher, quantity $|S_i|$ 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

involve K_θ, the Lipschitz constant of the DNN, which has the same impact than $f'(X_i)$. It also illustrates the link between the Lipschitz constant and the generalization error, which has been pointed out by several works like, for instance, [13], [3] and [31].

From Taylor approximation to local variance (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 $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \epsilon \mathbb{I}_{n_i})$ with $\epsilon \in \mathbb{R}^+$ and $\boldsymbol{\epsilon} = (\epsilon_1, ..., \epsilon_{n_i})$, i.e. $\epsilon_i \sim \mathcal{N}(0, \epsilon)$. Using Taylor approximation on f at order 2 gives

$$f(x + \boldsymbol{\epsilon}) = f(x) + \nabla_x f(x) \cdot \boldsymbol{\epsilon} + \frac{1}{2} \boldsymbol{\epsilon}^T \cdot \mathbb{H}_x f(x) \cdot \boldsymbol{\epsilon} + O(\|\boldsymbol{\epsilon}\|_2^3).$$

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

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

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

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

In this expression, we have to assess three quantities: $Cov(\epsilon_{i_1}, \epsilon_{i_2})$, $Cov(\epsilon_i, \epsilon_j \epsilon_k)$ 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,

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

And finally,

$$\boxed{Var(f(x+\epsilon)) = Df_\epsilon^2(x) + O(\|\epsilon\|_2^3)} \tag{10}$$

If we consider $\widehat{Df^2}_\epsilon(x)$ as defined in (4), on section 3.1 of the main document, $\widehat{Df^2}_\epsilon(x) \underset{k\to\infty}{\to} Var(f(x+\boldsymbol{\epsilon}))$. Since $Var(f(x+\boldsymbol{\epsilon})) = Df_{\boldsymbol{\epsilon}}^2(x) + O(|\epsilon|_2^3)$, $\widehat{Df^2}_\epsilon(x)$ is a biased estimator of $Df_{\boldsymbol{\epsilon}}^2(x)$, with bias $O(|\epsilon|_2^3)$. Hence, when $\epsilon \to 0$, $\widehat{Df^2}_\epsilon(x)$ becomes an unbiased estimator of $Df_{\boldsymbol{\epsilon}}^2(x)$.

Appendix C: Hyperparameters

This appendix is split into two parts. First, we describe the results of the experiments on the hyperparameters search of Boston Housing data set (section 4.1.). The second part is a list of final hyperparameters values chosen for the experiments of the main paper.

Experiments on Boston Housing data set

For Boston Housing experiment, we conduct a grid search for VBSW on the values of m and k. We train a linear model 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 results, we train the model on 400 pairs of (m, k) values, and with 10 different random seeds for each pair.

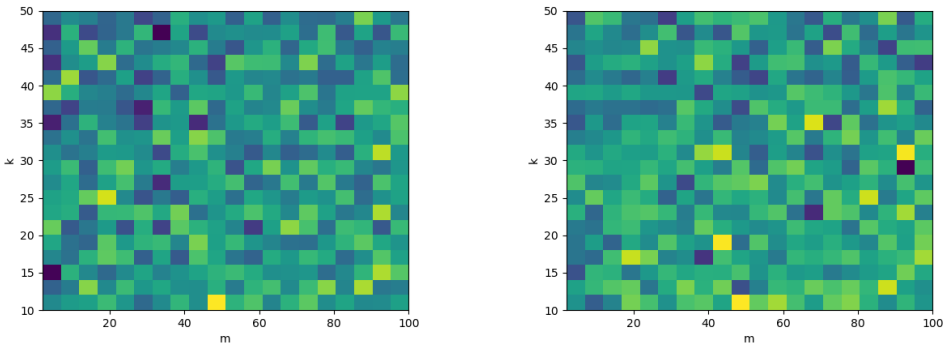

Figure 5: Color map of MSE, with respect to m and k. **Left:** for the mean of the MSE accross 10 different seeds and **right:** for the min of the MSE across these seeds. Blue is lower.

In Figure 5 even if the error is lower for small m and k above 40, there is still a great amount of noise, and no specific values of m and k which are clearly better. For this case, we could maybe narrow the search regions to $k \in [30, 50]$ and $m \in [2, 40]$. However as we can see in Table 6, values outside of these regions can be chosen as best values (e.g. m value for MRPC). Therefore, a classical hyperparameter search should be used to chose the values of m and k.

Paper hyperparameters values

The values chosen for the hyperparameters of the paper experiments are gathered in Table 6. 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
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 6: Paper experiments hyperparameters values

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang. Stronger generalization bounds for deep nets via a compression approach. In J. Dy and A. 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] P. L. Bartlett, D. J. Foster, and M. 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] P. L. Bartlett, N. Harvey, C. Liaw, and A. Mehrabian. Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks. *Journal of Machine Learning Research*, 20(63):1–17, 2019.

[5] P. L. Bartlett, V. Maiorov, and R. 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] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, 2009.

[7] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[8] H.-S. Chang, E. Learned-Miller, and A. 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.

[9] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie. Class-balanced loss based on effective number of samples. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. 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.

[11] J. Feng, Q. Teng, X. He, and X. Wu. Accelerating multi-point statistics reconstruction method for porous media via deep learning. *Acta Materialia*, 159:296–308, 2018.

[12] Y. Gal, R. Islam, and Z. 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, 2017.

[13] H. Gouk, E. Frank, B. Pfahringer, and M. Cree. Regularisation of neural networks by enforcing lipschitz continuity. 04 2018.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[15] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[16] D. Jakubovitz, R. Giryes, and M. R. D. Rodrigues. Generalization error in deep learning. *CoRR*, abs/1808.01174, 2018.

[17] L. Jiang, D. Meng, Q. Zhao, S. Shan, and A. 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.

[18] T. Jie and P. 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.

[19] A. Katharopoulos and F. Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *ICML*, 2018.

[20] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[21] K. Konyushkova, R. Sznitman, and P. 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.

[22] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research).

[23] M. P. Kumar, B. Packer, and D. 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.

[24] 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.

[25] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[26] T. Liu and D. Tao. Classification with noisy labels by importance reweighting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(3):447–461, Mar. 2016.

[27] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman. Teacher-student curriculum learning, 2017.

[28] B. Neyshabur, S. Bhojanapalli, D. Mcallester, and N. 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.

[29] B. Neyshabur, S. Bhojanapalli, and N. Srebro. A PAC-bayesian approach to spectrally-normalized margin bounds for neural networks. In *International Conference on Learning Representations*, 2018.

[30] 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.

[31] H. Qian and M. N. Wegman. L2-nonexpansive neural networks. In *International Conference on Learning Representations*, 2019.

[32] M. Ren, W. Zeng, B. Yang, and R. Urtasun. Learning to reweight examples for robust deep learning. *CoRR*, abs/1803.09050, 2018.

[33] B. Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[34] A. Shrivastava, A. Gupta, and R. B. Girshick. Training region-based object detectors with online hard example mining. *CoRR*, abs/1604.03540, 2016.

[35] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962v2*, 2019.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. 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.

[37] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.

[38] N. Winovich, K. Ramani, and G. Lin. Convpde-uq: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *Journal of Computational Physics*, 394:263 – 279, 2019.

[39] H. Xu and S. Mannor. Robustness and generalization. *Machine Learning*, 86(3):391–423, Mar 2012.

[40] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56 – 81, 2019.