Simulated Hyperparameter Optimization for Statistical Tests in Machine Learning Benchmarks

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Statistical testing grounds progress in empirical science, extracting reliable findings in the face of noisy evidence. Testing procedures require reliable estimates of the variance of the results, typically obtained via many replicates of an experiment. But machine learning experiments are expensive, and sources of variance abound: data sampling, random data augmentation, parameter initialization, etc. An important variance source which is often overlooked is due to hyperparameter optimization, a procedure typically deemed too costly to replicate many times. We alleviate this burden with a novel approach that requires only one execution of the expensive hyperparameter optimization procedure. With but a few additional trainings, we obtain useful variance estimates and confidence intervals. We use a cheap model to *simulate* subsequent hyperparameter optimizations without incurring their computational cost. We perform an extensive study of all sources of variance, fitting more than 170 000 models on 5 different tasks. Our results highlight the significant effect of randomness in hyperparameter optimization and supports the validity of our proposed simulation method.

1 Introduction: trustworthy benchmarks account for fluctuations

Evidence for a new method or algorithm A in machine learning often builds upon empirical benchmarks comparing it to prior work. The goal is to show that A produces a better score, by some chosen metric(s). Although such benchmarks are quantitative, some loopholes may easily undermine their objectivity and reproducibility. In particular, recent works have shown that uncontrolled choices in hyper-parameters lead to non-reproducible benchmarks and unfair comparisons [1–8]. Properly accounting for these may go as far as changing the conclusions for the comparison, as shown for recommender systems [9], neural architecture pruning [10], and metric learning [11].

The steady increase in number of hyper-parameters and configurations of learning pipelines calls for systematic benchmarking procedures. Yet, in parallel, the growing computational costs of models make brute-force approaches prohibitive. Indeed, robust conclusions on comparative performance of models A and B would require multiple training runs on different realization of the full learning pipelines, including hyper-parameter optimization and random seedings. Unfortunately, the computational budget of most researchers can afford only a small number of model fits [12]. Thus, many sources of variances remain not estimated via repeated experiments. This in turn amplifies the risk of conclusions built upon differences due to arbitrary factors, such as data order, rather than modeling improvements.

Current tools for the statistical evaluation of machine learning methods are based on variants of k-fold cross validation or bootstrapping [13–16], which exacerbates the already high cost of modern deep learning. We extend the work of Hothorn et al. [16] by formally describing the sources of variation that contribute to a model's performance as one of two types: algorithm intrinsic (e.g., random seed controlling weight initialization) and hyper parameter optimization process (e.g., random seed controlling hyper-parameter selection). This lets us develop a *computationally-efficient procedure* for joint hyper-parameter optimization and total variance estimation, enabling statistical testing. The simulation method we propose removes the need to run costly additional full hyperparameter optimization procedures to estimate this variance. Then we can use these results to study the *sources of variation* distinguishing between hyper-parameter optimization and spurious factors like random initialization, that influence the evaluation and conclusions. Our contributions are as follows:

- The first formal description of the whole machine-learning pipeline that makes explicit two types of sources of variance that are algorithm intrinsic and specific to hyper-parameter optimization.
- An empirical evaluation shows that the hyper-parameter optimization process is an important part of the total variance in results. To the best of our knowledge, this is the first thorough study of the variance induced by hyper-parameter optimization itself.
- We present an original low-cost approach to estimate the variance of a performance metric from a single hyperparameter optimization (by cheaply simulating subsequent hyperparameter optimizations), thus obtaining confidence intervals and enabling statistical testing.
- Using this method for statistical tests, we show that the importance of "good" and "bad" initial weights may not be as strong as previously thought.

2 Formalization of the pipelines for training and benchmarking models

Comparison of machine learning approaches is routinely done by comparing the value of a performance metric of interest they achieve. Such a value, obtained after running a complete learning pipeline on a finite dataset, should be considered the realization of a random variable. Indeed that dataset is itself a random sample from the true data distribution, and a typical learning pipeline has additional sources of randomness, as we will highlight below. A proper evaluation and comparison between approaches should thus account for the *distributions* of such metrics, and be carried out with adequate statistical tests. Procedures that probe these distributions are well-posed given a complete probabilistic description of how a realization of the metrics of interest is obtained. The framework we describe here extends in subsection 2.1 the formalism of Hothorn et al. [16] to derive biases and variances due to hyperparameter optimization procedures. Then subsection 2.2 we develop the metric to test the predictor's performance.

2.1 The learning pipeline

The training procedure To simplify the exposition, we consider here the familiar setting of supervised learning on i.i.d. data (and will use classification in our experiments) but this can easily be adapted to other machine learning settings. Suppose we have access to a dataset $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$ containing n examples of (input, target) pairs that are i.i.d. samples from an unknown true data distribution $\mathcal{D}$, i.e. $S \sim \mathcal{D}^n$. The goal of a complete learning pipeline is to find a good predictor, a function $h \in \mathcal{H}$ that will have good prediction performance in expectation over $\mathcal{D}$, as evaluated by a metric of interest e. More precisely, in supervised learning, $e(h(x), y)$ is a measure of how far a prediction $h(x)$ lies from the target y associated to the input x (e.g., classification error). The goal is to find a predictor h that minimizes the *expected risk* $R_e(h, \mathcal{D}) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[e(h(x), y)]$, but since we have access only to finite datasets, all we can ever measure is an *empirical risk* $\hat{R}_e(h, S) = \frac{1}{|S|}\sum_{(x,y)\in S} e(h(x), y)$. Leveraging efficient gradient-based optimization to learn parameters θ of $h_\theta \in \mathcal{H}$ requires a differentiable optimization objective, but $\hat{R}_e(h, S)$ often is not. So in place of e, a differentiable surrogate loss ℓ is typically used for defining an optimization objective, a regularized empirical risk: $J(\theta; S, \lambda_J) = \hat{R}_\ell(h_\theta, S) + \Omega(h_\theta; \lambda_J)$, where Ω is a regularization term, controlled by hyperparameters λ_J. Let Opt be the gradient-based optimization algorithm used. It will typically have additional hyperparameters λ_O, and need a starting point for θ which we will sample from a heuristic initialization distribution $\theta^0 \sim P^{\text{init}}$. Let us denote $\widehat{\theta^*}$ the approximate minimizer found by that optimization algorithm on a set S:

$$\widehat{\theta^*}(S; \lambda_J, \lambda_O) = \mathrm{Opt}(J; \lambda_O) \approx \arg\min_\theta J(\theta; S, \lambda_J) \tag{1}$$

And keep in mind that $\widehat{\theta^*}$ is a random variable whose value will depend also on other additional random variables that we shall collectively denote ξ_O, that are sampled to determine parameter initialization, data augmentation, example ordering, etc.[1].

Tuning hyperparameters We have thus defined the training procedure to train a predictor on a data set S. But since it requires specifying hyperparameters λ_J, λ_O and possibly also architectural hyperparameters $\lambda_{\mathcal{H}}$ (that define the hypothesis class $\mathcal{H}$), a complete learning pipeline has to tune all of these. Let λ be the set of all hyperparameters we need to tune. A complete pipeline will involve a hyper-parameter optimization procedure, which will strive to find a value of λ that minimizes objective

$$r(\lambda) = \mathbb{E}_{(S^t,S^v)\sim \mathrm{sp}(S)}\left[\hat{R}_e\left(h_{\widehat{\theta^*}(S^t;\lambda)}, S^v\right)\right] \tag{2}$$

where $\mathrm{sp}(S)$ is a distribution of random splits of the data set S between training and validation subsets S^t, S^v. Ideally, hyperparameter optimization would be applied over random dataset samples from the true distribution $\mathcal{D}$, but in practice the learning pipeline is limited to S, hence the expectation over splits. An ideal hyper-parameter optimization would yield $\lambda^*(S) = \arg\min_\lambda r(\lambda)$. A concrete hyperparameter optimization algorithm HOpt will however use an average over a small number of splits (or just 1), and a limited training budget, yielding $\widehat{\lambda^*}(S) = \mathrm{HOpt}(r; S, \xi_H) \approx \lambda^*(S)$. ξ_H are random variables that yield the splitting and random exploration of the hyperparameter search procedure HOpt. These are the additional sources of variance introduced by hyperparameter optimization.

After hyperparameters have been tuned, it is customary to retrain the predictor using the full data S. So the complete learning pipeline $\mathcal{P}$ will finally return a single predictor:

$$\widehat{h^*}(S) = \mathcal{P}(S) = h_{\widehat{\theta^*}\left(S;\widehat{\lambda^*}(S)\right)} \tag{3}$$

Recall that $\widehat{h^*}(S)$ is not deterministic, as it is affected by random sources ξ_O in the training of θ (random weight initialization and example ordering, possibly random data augmentation) and ξ_H in the hyperparameter optimization. We will use ξ to denote the set of all variations sources in the learning pipeline, $\xi = \xi_H \cup \xi_O$. Thus ξ captures source of variation in the learning pipeline that are not configurable with λ.

2.2 Benchmarking: measuring the performance of a learning pipeline

The full learning procedure $\mathcal{P}$ described above yields a a model $\widehat{h^*}$. We now must define a metric that we can use to evaluate this model with statistical tests. For simplicity, we will use the same evaluation metric e on which we based hyperparameter optimization. The expected risk obtained by applying the full learning pipeline $\mathcal{P}$ to datasets $S \sim \mathcal{D}^n$ of size n is:

$$R_{\mathcal{P}}(\mathcal{D}, n) = \mathbb{E}_{S\sim\mathcal{D}^n}\left[R_e(\widehat{h^*}(S), \mathcal{D})\right] \tag{4}$$

where the expectation is also over the random sources (i.e., $\lambda \cup \xi$) that affect the learning procedure (initialization, ordering, data-augmentation) and hyperparameter optimization[2].

As we have only access to a single finite dataset S, this can be evaluated as the following empirical average:

$$\hat{R}_{\mathcal{P}}(S, n, n') \quad = \quad \widehat{\mathbb{E}^k}_{(S^{tv},S^o)\sim \mathrm{sp}_{n,n'}(S)}\left[\hat{R}_e(\widehat{h^*}(S^{tv}), S^o)\right]$$

where sp is a distribution of either random splits or bootstrap resampling of the data set S that yield sets S^{tv} (train+valid) of size n and S^o (test) of size n'. We denote by $\widehat{\mathbb{E}^k}$ an empirical average over k samples. Here, samples vary not only on how the data was split, but also on all other random factors affecting the learning procedure and hyperparameters. In the next section we gauge the effect on the training procedure of different sources of variance.

[1]If stochastic data augmentation is used, then optimization objective J for a given training set S has to be changed to an expectation over $\tilde{S} \sim P^{\mathrm{aug}}(\tilde{S}|S; \lambda_{\mathrm{aug}})$ where P^{aug} is the data augmentation distribution. This adds additional stochasticity to the optimization, as we will optimize this through samples from P^{aug} obtained thanks to a random number generator.

[2]The size n' of test sample S^o does not matter here as we are taking a true expectation.

3 Empirical evaluation of sources of variance in benchmarks

We first evaluated empirically the relative importance of common sources of variances within the standard learning pipeline, as previously described. The source of variance we investigated come either originating from the learning algorithm (collectively referred to as ξ_O) or from the hyperparameter optimization method (collectively referred to as ξ_H). We omit sources of variation due to hardware and software [17].

We performed our evaluation on a wide variety of case studies, spanning multiple application domains. Specifically, we selected i) the CIFAR10 [18] image classification with VGG11 [19], ii) PascalVOC [20] image segmentation using an FCN [21] with a ResNet18 [22] backbone pretrained on imagenet [23], iii-iv) Glue [24] SST-2 [25] and RTE [26] tasks with BERT [27] and v) peptide to major histocompatibility class I (MHC I) binding predictions with a shallow MLP. All details on default hyperparameters used and the computational environments can be found in Appendix C.

3.1 Variance from sources within the learning algorithms: ξ_O

For the sources of variance from the learning algorithm (ξ_O), we identified: i) the data sampling, ii) data augmentation procedures, iii) model initialization, iv) dropout and v) data visit order in stochastic gradient descent. We model the data-sampling variance as resulting from training the model on a finite dataset S of size n, sampled from an unknown true distribution. $S \sim \mathcal{D}^n$ is thus a random variable, and the standard source of variance considered in statistical learning. Since we have a single finite dataset in practice, we can evaluate this variance by repeatedly generating a variety of datasets of size n via bootstrap samples[3].

First, we fixed the hyperparameter configuration to a pre-selected reasonable one [4]. Then, iteratively for each of these sources of variance, we randomized the seeds 200 times, while keeping all other sources fixed to the initially chosen values . Moreover, we measured the background noise with 200 training runs with all fixed seeds.

We aggregated all the obtained individual variances due to sources from within the learning algorithms into Figure 1 for better side by side comparison. We observed that bootstrapping data stands out as the most important source of variance. In contrast, model initialization is generally representing less than 50% of the total variance, on par with data ordering.

3.2 Variance induced by hyperparameter optimization methods: ξ_H

To study the ξ_H sources of variation, we chose three of the most popular hyperparameter optimization methods: i) random search, ii) grid search and iii) Bayesian optimization. Naturally, while grid-search in itself has no random parameters, the specific choice of the parameter range is arbitrary and consequently we expect it can be an uncontrolled source of variance. We study this variance with a *noisy grid search*, in which we perturb slightly the parameter ranges (details in Appendix D).

For each of these chosen tuning methods, we held all ξ_O fixed to random values and executed 20 independent hyperparameter optimization procedures up to a budget of 200 trials. This way, all the observed variance across the hyperparameter optimization procedures is strictly due to ξ_H. It is of note that we designed the search space so that it covers the optimal hyperparameter values while still being large enough so that its borders cover suboptimal values.

Our results show that hyperparameter optimization methods induce a significant amount of variance (Figure 1). While all three methods induced varying levels of variance in performance, we observed that model initialization induced comparable variance to noisy grid search. Bootstrapping was found to be comparable on average to noisy random search. Taken together, these results motivate the need to further investigate the variance due to hyperparameter optimization as they play a sizeable role in the total variance of the learning pipeline.

[3]The more common alternative in machine learning is to use cross-validation, but the latter will tend to underestimate variance, whose good estimation is our primary goal here, hence our choice. Bootstrapping is discussed in more detail in Appendix B.

[4]The partial pair-wise dependence plots used to validate the search spaces can be found in Appendix C.

Conclusion on sources of variance Experiments show that run-to-run differences in hyperparameter optimization are a sizable factor of variation in the overall performance of a model. Hence, sampling only model-specific sources of randomness (ξ_O) is insufficient to estimate the variance in model performance in a benchmark. However, the computational cost of running multiple hyperparameter optimizations is extreme and prohibitive. In the next section, we present a simulation-based approach to reduce the cost of estimating the variance due to hyperparameter optimization.

4 Simulated hyperparameter optimization

While in the previous section we showed that hyperparameter tuning generates non-negligible variance in the model's performance, a recent study [12] showed that executing even a single sufficiently thorough hyper-parameter optimization is computationally-prohibitive for most researchers, let alone perform a whole set of independent ones.

To circumvent this issue, we propose to leverage the information acquired from a single hyperparameter optimization to cheaply simulate subsequent hyperparameter optimizations instead of running many independent ones.

4.1 Simulation with a surrogate model

To simulate the hyperparameter optimization, we first build a surrogate model for $r(\lambda)$. Let $\{\lambda^i, r(\lambda^i)\}_{i=1}^{T}$ represent the history of trials from a single hyperparameter optimization procedure carried out with method $\mathrm{HOpt}(S)$. We fit a surrogate model $\hat{r}(\lambda)$ to this data, which will enable computing a fast approximation of $r(\lambda)$. The simulated hyperparameter optimization simply consists in replacing $r(\lambda)$ by $\hat{r}(\lambda)$ when performing *all subsequent* hyperparameter optimizations. So instead of searching for $\widehat{\lambda^*}(S) = \mathrm{HOpt}(r; S, \xi_H)$ we will be using the much cheaper $\widetilde{\lambda^*}(S) = \mathrm{HOpt}(\hat{r}; S, \xi_H)$. Going forward, we will use the notation ˆ for estimated values, and ˜ for simulated ones.

To approximate $r(\lambda)$ we use a Gaussian Process Regression (GPR) as our surrogate $\tilde{r}(\lambda)$. GPRs are widely used in Bayesian Optimization of hyperparameters [28, 29] for their desirable properties in estimating uncertainty and sample efficiency. For its configuration, we use a Matern52 kernel and integrate over its hyperparameter using Markov chain Monte Carlo as presented by Snoek et al. [29]. The mean prediction over the sample of the Markov chain Monte Carlo is used as the prediction of $\hat{r}(\lambda)$. We refer the reader to Forrester and Keane [30] for a detailed review of the use of surrogate models in simulations, and the pros/cons of other alternatives to modeling $\tilde{r}(\lambda)$.

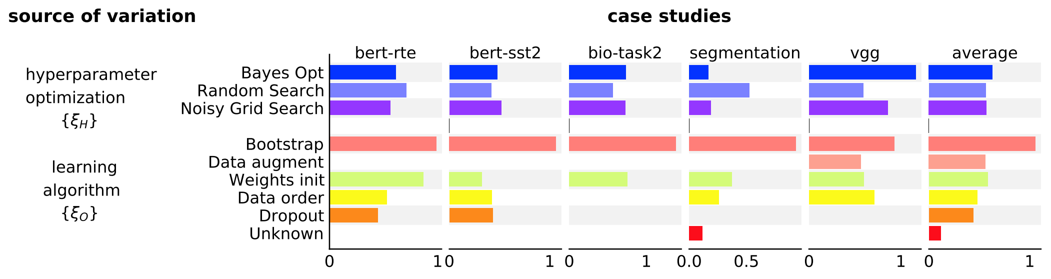

Figure 1: Proportions of standard deviations of test metrics attributable to common sources of variation with respect to full standard deviation. Group above are sources of variation ξ_H specific to hyperparameter optimization algorithms while gloup below are sources of variation ξ_o intrinsic to the learning algorithms. For each row of the group ξ_H, we fix all sources of variation of ξ_o and randomize ξ_H, executing hyperparameter optimization with the respective algorithms for a budget of 100 trials. For each row of the group ξ_o, we vary a single source of variation, fix all the rest to random seeds and set hyperparameter to good performing ones. The standard deviation of the test performance over 20 and 200 trainings is reported for ξ_H and ξ_O respectively, normalized by total standard deviation obtained when varying ξ_H and ξ_O altogether (using random search). Bootstrap stands out as the most important source of variation. Weights init on the other hand, the source of variation most commonly used to estimate standard deviation of performances, accounts for less than 50% of the total standard deviation in most tasks.

4.2 Empirical evaluation of simulated hyperparameter optimization

We investigate empirically the performance of our proposed estimator $\hat{r}(\lambda)$ by comparing the variance of the empirical risk $\hat{R}$ using the real random variable $\widehat{\lambda^*}(S)$ or the simulated $\widetilde{\lambda^*}(S)$.

For the simulation to be useful in statistical tests we must verify two points: i) The mean estimate has a small bias and standard error and ii) the variance estimation is close to the real one. To estimate the variance in the simulation, we compared it to two other replicate estimates: i) the *unbiased replicates* and ii) the *biased replicates ignoring* ξ_H. The *unbiased replicates* are what we consider the reference measures; they are obtained by running the full hyperparameter optimization procedure and are, consequently, the gold standard values we are attempting to estimate via the simulation. They come however with a prohibitive computational cost. The *biased replicates ignoring* ξ_H represent a common way of estimating the variance where the hyperparameters are kept fixed.

We first executed the *unbiased* procedure 100 times for reference. We then compute 20 mean and variance estimates with the cheap methods for sample sizes of 100. Note that estimating the mean with a sample size of 100 costs 100, 000 trainings with the unbiased procedure while costing only 200 trainings with the cheap methods.

Results are presented in Figure 2. Across all tasks, the common *biased* estimation, varying all ξ_o and ignoring ξ_H, proved to be a good estimator of the mean with small bias and standard error. The latter tended however to under-estimate the variance while the *simulated* estimation tended to over-estimate it. All-in-all, using the *biased* estimation of the mean and the average of the *biased* and *simulated* estimation of the standard deviation appears to be a good middle-ground.

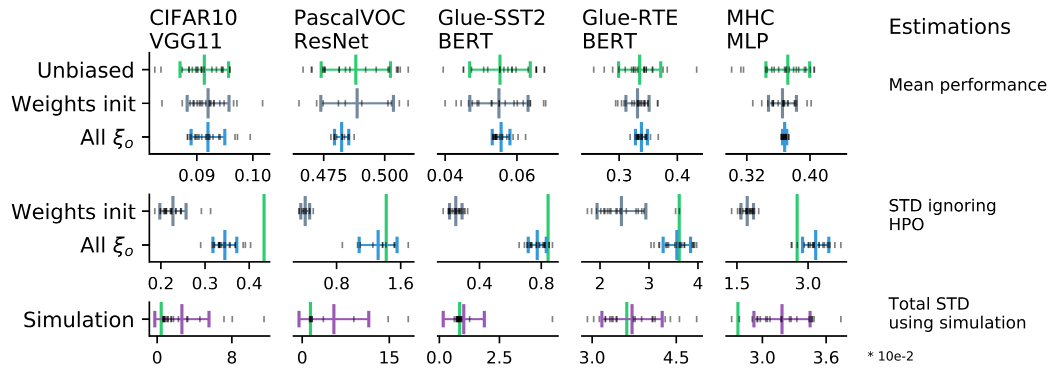

Figure 2: **Mean plot (top row)**: Estimated test metric on different tasks with 3 approaches. Typically a practitionner will use computation ressources to get but a single black tick. **Unbiased:** (= standard practice) each tick is the metric obtained after a single full hyperparameter optimization with a budget of 100 trainings (each black tick an *unbiased* estimate of the mean). **Weights-init:** 1 tick = averaged metric from *subsequent* 100 trainings keeping hyperparameters fixed, varying weights init only. **All ξ_0:** 1 tick = averaged metric from *subsequent* 100 trainings keeping hyperparameters fixed, varying all other random factors. Notice that the latter have reduced variance at the cost of introducing bias.
STD plots (middle and bottom row): Estimated standard deviation (STD) of test metric. Green vertical bar marks gold standard (STD of 100 full HPOs = width of horizontal green bar on first row of Mean-plot). *All* shown methods do a *single* full HPO with the actual model. **Weights-init:** 1 tick = STD from *subsequent* 100 trainings keeping hyperparameters **All ξ_0:** 1 tick = STD from *subsequent* 100 trainings keeping hyperparameters fixed, varying all other random factors. **Simulation:** total STD estimated by simulating 100 hyperparameter optimizations, varying all other random factors. We see that variance estimation ignoring the effect of hyperparameter optimization tends to under-estimate the true variance, while simulations tend to over-estimate it.

4.3 Case study: Are some initializations actually better or worse?

As an example of how to use our framework to provide more robust conclusions we will try answering the question: given a specific set of initial weights θ_0^g found to be *good*, and another set θ_0^b observed as *bad*, how robust are these findings, in particular with regards to other uncontrolled factors ξ? To find the good and bad initial weights θ_0^g and θ_0^b, we reuse the results from section 3. Recall that we trained 200 models by varying only one source of variation at a time, so for the experiments where only θ was altered, we take the best and worst results to become θ_0^g and θ_0^b.

The null hypothesis H_0 is that there is no difference, and the alternative hypothesis is that there exist a difference of at least δ. We set δ to be 0.2 times the magnitude initially found. More formally, $\delta = \frac{p_{\theta_0^g} - p_{\theta_0^b}}{5}$. Our data is approximately normal, so we use a paired t-test over ξ_o to increase the statistical power. All differences will be computed on the same pair of random ξ_o (bootstrapping of the dataset, data order, dropout, data transformation, etc, depending on the task). Let $p_{\theta_0^g \xi_o^i}$ and $p_{\theta_0^b \xi_o^i}$ be the performances after training with the worst and best initializations respectively using the same random state ξ_o^i. The paired difference is then $d_i = p_{\theta_0^g \xi_o^i} - p_{\theta_0^b \xi_o^i}$. We do not pair on the hyperparameter as the hyperparameter optimization procedures will be independent for $p_{\theta_0^g}$ and $p_{\theta_0^b}$. We compute the sample size using our results from Section 4.2 to estimate the standard deviation $\hat{\sigma}$[5].

For each θ_0^g and θ_0^b, we perform four steps as described in Figure 3 to estimate a confidence interval. If the confidence covers the value 0, we do not reject the null hypothesis and conclude that best initializations θ_0^g were not significantly better than θ_0^b under all other sources of variation. We cannot say that they are the same, but at least that the difference is not larger than 0.2 times the one initially found when selecting them.

Results for all tasks are presented in Table 1. For all tasks, the test is inconclusive and we cannot conclude that the best initialization lead to significantly better results than the worst initialization. If there is a difference we consider it to be $\leq \delta = 0.2$ times the original observed difference.

[5]In practice an estimation of the variance may not be available beforehand. See Appendix F for an example using a stopping rule instead of a sample size calculation.

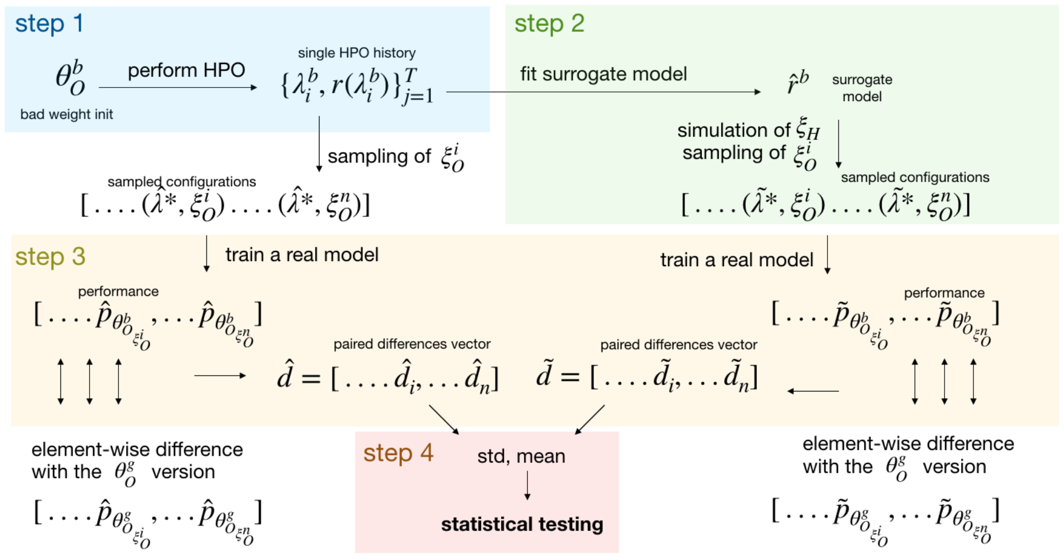
step 1
θ_O^b
bad weight init
perform HPO
single HPO history
$\{\lambda_i^b, r(\lambda_i^b)\}_{j=1}^T$
sampling of ξ_O^i
sampled configurations
$[\ldots.(\hat{\lambda}^*, \xi_O^i)\ldots.(\hat{\lambda}^*, \xi_O^n)]$
step 2
fit surrogate model
$\hat{r}^b$
surrogate model
simulation of ξ_H
sampling of ξ_O^i
sampled configurations
$[\ldots.(\tilde{\lambda}^*, \xi_O^i)\ldots.(\tilde{\lambda}^*, \xi_O^n)]$
step 3
train a real model
train a real model
performance
$[\ldots.\hat{p}_{\theta_{O\,\xi_O^i}^b}, \ldots\hat{p}_{\theta_{O\,\xi_O^n}^b}]$
paired differences vector
$\hat{d} = [\ldots.\hat{d}_i, \ldots\hat{d}_n]$
paired differences vector
$\tilde{d} = [\ldots.\tilde{d}_i, \ldots\tilde{d}_n]$
performance
$[\ldots.\tilde{p}_{\theta_{O\,\xi_O^i}^b}, \ldots\tilde{p}_{\theta_{O\,\xi_O^n}^b}]$
element-wise difference with the θ_O^g version
$[\ldots.\hat{p}_{\theta_{O\,\xi_O^i}^g}, \ldots\hat{p}_{\theta_{O\,\xi_O^n}^g}]$
step 4
std, mean
statistical testing
element-wise difference with the θ_O^g version
$[\ldots.\tilde{p}_{\theta_{O\,\xi_O^i}^g}, \ldots\tilde{p}_{\theta_{O\,\xi_O^n}^g}]$

Figure 3: Case study analysis pipeline. Only analysis steps for θ_O^b are shown; full analysis mirrors all steps for θ_O^g. **1)** We first optimize the hyper-parameters using a budget of T trials. **2)** We fit a Gaussian Process Regressor to HPO history and sample n pairs of $(\tilde{\lambda}^i, \xi_o^i)$, sampling at the same time n pairs of $(\hat{\lambda}^*, \xi_o^i)$ on the right. **3)** We train the models on the sampled configurations and compute the differences between the corresponding pairs of $(p^i_{\theta_0^g}, p^i_{\theta_0^b})$. **4)** We take the mean of $\hat{d}$ on the left as the mean estimate and the average of the STDs from $\hat{d}$ and $\tilde{d}$ as the STD estimate.

Table 1: Are best initializations found in a pool of 200 still generally better than all other initializations if we randomly vary ξ? No.

Task	min	max	max-min	δ	CI
VGG11-CIFAR10	9.23	10.4	1.2	0.24	(-0.33, 0.13)
ResNet-PascalVOC	45.1	47.7	2.6	0.52	(-1.34, 1.23)
BERT-SST2	4.85	5.98	1.13	0.23	(-0.01, 0.14)
BERT-RTE	35.4	48.7	13.3	2.67	(-0.88, 2.22)
MLP-MHC	32.5	42.4	9.9	1.98	(-2.08, 0.87)

5 Limitations

Smoothness We expect the effect of the hyperparameters to be relatively smooth on the final performance of learning algorithms. This means small changes of hyperparameter values should not affect significantly the final performance. If this assumption is violated, the simulation will either greatly over-estimate the variance or only provide bad samples which in turn would systematically lead to poor performances and a small variance; both these cases lead to useless estimates.

Representativity The surrogate model requires a good exploration of the search space in order to represent it properly. Although good methods for hyperparameter optimization generally balance exploration and exploitation, they may not exploit the same regions on every execution. An algorithm attempting exploitation in a region that was not explored in the original data used to fit the surrogate model may lead to very inaccurate simulations.

Homoscedasticity All variances were analyzed for a single default configuration. This is because we assumed the variance due to one source of variation to be constant across different configurations. While this is not generally true, the results of the simulations nevertheless suggest that sources of variations we studied are roughly homoscedastic in good performing regimes.

6 Discussion

Due to limited computational resources, many studies perform only one thorough hyper-parameter optimization for each algorithm they are considering, and present results from that single run. Our results have shown that conclusions drawn in this fashion may not be robust, especially when the effect size is small. Consider our results from VGG, which show that the results of the model will vary by ≥ 1 whole percentage point for 20% of hyper-parameter optimization runs. Since many papers propose improvements of $\leq 1\%$, this means there is a large risk of falsely concluding that a new method is better than existing methods when using a single hyper-parameter run. This is not to say that small improvements do not have value, but that we need to be more principled in detecting and verifying these improvements. Otherwise unreliable and unreproducible findings threaten to slow progress in machine learning.

Another way to view this problem, motivated by our framework, is that the entire process of optimizing and training a model produces just *one* sample. This single sample gives us information about the relative quality of competing methods. But to make a conclusion that one method is better than another with a sample size of one would be a statistical test with low power. In this light, we hope the impact is more clear that we have identified a critical area from improvement in the quality of how we compare methods.

7 Conclusion

We presented a formalization of the whole learning pipeline which includes the hyperparameter optimization procedure, clarifying the role of hyperparameter optimization to measure the bias and variance it induces on performance metrics. Our study highlighted the important effect of many sources of variation, including hyperparameter optimization, clearly showing that it shouldn't be ignored as it typically is. To provide cheap estimate of the variance due to hyperparameter optimization we proposed the use of a simulation to upper bound the real variance and the use of a biased procedure to lower bound it. Finally, we presented a case study to demonstrate the use of our mean and variance estimation to conduct statistical tests with confidence intervals.

Broader Impact

Statistical testing is omnipresent in science in general, being strongly connected to epistemolgy. Likewise, the method presented in this work could be applied to a large part of machine learning research and its applications. Its use would provide more reliable estimates for generalization, which is especially important for applications on real world data. Real world problems are often shifting distributions however, and more mork would be needed to extend our method beyond i.i.d. samples from fixed distributions. Beyond research, these tools could be used as well by the industry to support their products, either hardware through benchmarks such as MLPerf [31] or services of diverse machine learning applications.

Because of its computational cost, deep learning is out of reach or pose a serious engineering challenge for many small enterprises. As described in the introduction, the use of proper statistical testing methods is also problematic because of this computational cost. Although we do propose a method to reduce its cost, using statistical methods for testing models is still more expensive than common manual tuning with simple point estimates in machine learning research. By emphasising the importance of rigorously asserting model performances through statistical tests and bringing the research community to adopt such methods, this could apply additional pressure on computational ressources, and increase the associated environmental impact.

If our method is adopted it is possible that this may slow the apparent pace of progress. Work performed by researchers may fail to produce a significant result under newer testing, and thus not be submitted/published. Our hope is that it will help establish a greater scientific understanding addressing a need for improved "empirical rigor" has been previously noted [32]. This too is not without risk, as negative results are equally valuable, but publication has long suffered from a positive result bias. That a new method provided no benefit, or possibly negative benefit, is often valuable information that is lost in today's process. The amount of valuable work lost to this bias may increase if our approach is adopted. If such did occur, we would hope it could be remediated by creating avenues for such work in the future.

Statistical testing methods are blind to their applications. The same way they can be used to conclude misleading results based on experiments of small sample sizes in medical trials, they could be used to misleadingly support the quality new methods or products. Statistical testing is complex and is hazardous to turn into a simple checklist that is followed blindly. It cannot be separated from its philosophical, epistemological nature. Similarly to in-situ philosophers who serves as consultants for ethical concerns, there is a need for in-situ statisticians. Bad use of statistical methods has been shown to be often due to misunderstandings rather than bad intentions.

References

[1] E. Raff, "A Step Toward Quantifying Independently Reproducible Machine Learning Research," in *NeurIPS*, 2019. [Online]. Available: http://arxiv.org/abs/1909.06674

[2] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are gans created equal? a large-scale study," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 700–709.

[3] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[4] R. Kadlec, O. Bajgar, and J. Kleindienst, "Knowledge base completion: Baselines strike back," in *Proceedings of the 2nd Workshop on Representation Learning for NLP*, 2017, pp. 69–74.

[5] G. Melis, C. Dyer, and P. Blunsom, "On the state of the art of evaluation in neural language models," *ICLR*, 2018.

[6] X. Bouthillier, C. Laurent, and P. Vincent, "Unreproducible research is reproducible," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 725–734. [Online]. Available: http://proceedings.mlr.press/v97/bouthillier19a.html

[7] N. Reimers and I. Gurevych, “Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging,” in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 338–348. [Online]. Available: https://www.aclweb.org/anthology/D17-1035

[8] K. Gorman and S. Bedrick, “We need to talk about standard splits,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 2786–2791. [Online]. Available: https://www.aclweb.org/anthology/P19-1267

[9] M. F. Dacrema, P. Cremonesi, and D. Jannach, “Are we really making much progress? A Worrying Analysis of Recent Neural Recommendation Approaches,” in *Proceedings of the 13th ACM Conference on Recommender Systems - RecSys ’19*. New York, New York, USA: ACM Press, 2019, pp. 101–109. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3298689.3347058

[10] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, “What is the State of Neural Network Pruning?” in *Proceedings of Machine Learning and Systems 2020*, 2020, pp. 129–146.

[11] K. Musgrave, S. Belongie, and S.-N. Lim, “A Metric Learning Reality Check,” *arXiv*, 2020. [Online]. Available: http://arxiv.org/abs/2003.08505

[12] X. Bouthillier and G. Varoquaux, “Survey of machine-learning experimental methods at NeurIPS2019 and ICLR2020,” Inria Saclay Ile de France, Research Report, Jan. 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02447823

[13] R. R. Bouckaert and E. Frank, “Evaluating the replicability of significance tests for comparing learning algorithms,” in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2004, pp. 3–12.

[14] T. G. Dietterich, “Approximate statistical tests for comparing supervised classification learning algorithms,” *Neural computation*, vol. 10, no. 7, pp. 1895–1923, 1998.

[15] C. Nadeau and Y. Bengio, “Inference for the generalization error,” in *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen, and K. Müller, Eds. MIT Press, 2000, pp. 307–313.

[16] T. Hothorn, F. Leisch, A. Zeileis, and K. Hornik, “The design and analysis of benchmark experiments,” *Journal of Computational and Graphical Statistics*, vol. 14, no. 3, pp. 675–699, 2005.

[17] M. Crane, “Questionable answers in question answering research: Reproducibility and variability of published results,” *Transactions of the Association for Computational Linguistics*, vol. 6, pp. 241–252, 2018. [Online]. Available: https://www.aclweb.org/anthology/Q18-1018

[18] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Tech. Rep., 2009.

[19] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[20] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results,” http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[21] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” 2014.

[22] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.

[24] 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," 2019, in the Proceedings of ICLR.

[25] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of EMNLP*, 2013, pp. 1631–1642.

[26] L. Bentivogli, I. Dagan, H. T. Dang, D. Giampiccolo, and B. Magnini, "The fifth PASCAL recognizing textual entailment challenge," 2009.

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[28] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2019, in press, available at http://automl.org/book.

[29] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 2951–2959.

[30] A. I. Forrester and A. J. Keane, "Recent advances in surrogate-based optimization," *Progress in aerospace sciences*, vol. 45, no. 1-3, pp. 50–79, 2009.

[31] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou *et al.*, "Mlperf inference benchmark," *arXiv preprint arXiv:1911.02549*, 2019.

[32] D. Sculley, J. Snoek, A. Rahimi, and A. Wiltschko, "Winner's Curse? On Pace, Progress, and Empirical Rigor," in *ICLR Workshop track*, 2018. [Online]. Available: https://openreview.net/pdf?id=rJWF0Fywf

[33] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *Proceedings of International Conference on Machine Learning 2014 (ICML 2014)*, Jun. 2014, pp. 754–762.

[34] B. Efron, "Bootstrap methods: Another look at the jackknife," *Ann. Statist.*, vol. 7, no. 1, pp. 1–26, 01 1979. [Online]. Available: https://doi.org/10.1214/aos/1176344552

[35] ——, *The jackknife, the bootstrap, and other resampling plans*. Siam, 1982, vol. 38.

[36] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, vol. abs/1910.03771, 2019.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[38] A. Morcos, H. Yu, M. Paganini, and Y. Tian, "One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 4933–4943. [Online]. Available: http://papers.nips.cc/paper/8739-one-ticket-to-win-them-all-generalizing-lottery-ticket-initializations-across-datasets-and-optimizers.pdf

[39] H. Yu, S. Edunov, Y. Tian, and A. S. Morcos, "Playing the lottery with rewards and multiple languages: lottery tickets in RL and NLP," *arXiv*, 2019.

[40] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations (ICLR)*, 2019.

[41] W. W. Hauck and S. Anderson, "A new statistical procedure for testing equivalence in two-group comparative bioavailability trials," *Journal of Pharmacokinetics and Biopharmaceutics*, vol. 12, no. 1, pp. 83–91, 1984.