Université de Montréal

Accounting for variance and hyperparameter optimization in machine learning benchmarks

par

Xavier Bouthillier

Département d'informatique et de recherche opérationelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

June 20, 2022

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Accounting for variance and hyperparameter optimization in machine learning benchmarks

présentée par

Xavier Bouthillier

a été évaluée par un jury composé des personnes suivantes :

Simon Lacoste-Julien
(président-rapporteur)

Pascal Vincent
(directeur de recherche)

Alain Tapp
(membre du jury)

Frank Hutter
(examinateur externe)

Timothée Poisot
(représentant du doyen de la FESP)

Résumé

La récente révolution de l'apprentissage automatique s'est fortement appuyée sur l'utilisation de bancs de test standardisés. Ces derniers sont au centre de la méthodologie scientifique en apprentissage automatique, fournissant des cibles et mesures indéniables des améliorations des algorithmes d'apprentissage. Ils ne garantissent cependant pas la validité des résultats ce qui implique que certaines conclusions scientifiques sur les avancées en intelligence artificielle peuvent s'avérer erronées.

Nous abordons cette question dans cette thèse en soulevant d'abord la problématique (Chapitre 5), que nous étudions ensuite plus en profondeur pour apporter des solutions (Chapitre 6) et finalement developpons un nouvel outil afin d'amélioration la méthodologie des chercheurs (Chapitre 7).

Dans le premier article, chapitre 5, nous démontrons la problématique de la reproductibilité pour des bancs de test stables et consensuels, impliquant que ces problèmes sont endémiques aussi à de grands ensembles d'applications en apprentissage automatique possiblement moins stable et moins consensuels. Dans cet article, nous mettons en évidence l'impact important de la stochasticité des bancs de test, et ce même pour les plus stables tels que la classification d'images. Nous soutenons d'après ces résultats que les solutions doivent tenir compte de cette stochasticité pour améliorer la reproductibilité des bancs de test.

Dans le deuxième article, chapitre 6, nous étudions les différentes sources de variation typiques aux bancs de test en apprentissage automatique, mesurons l'effet de ces variations sur les méthodes de comparaison d'algorithmes et fournissons des recommandations sur la base de nos résultats. Une contribution importante de ce travail est la mesure de la fiabilité d'estimateurs peu couteux à calculer mais biaisés servant à estimer la performance moyenne des algorithmes. Tel qu'expliqué dans l'article, un estimateur idéal implique plusieurs exécution d'optimisation d'hyperparamètres ce qui le rend trop couteux à calculer. La plupart des chercheurs doivent donc recourir à l'alternative biaisée, mais nous ne savions pas jusqu'à présent la magnitude de la dégradation de cet estimateur. Sur la base de nos résultats, nous fournissons des recommandations pour la comparison d'algorithmes sur des bancs de test avec des budgets de calculs limités. Premièrement, les sources de variations devraient

être randomisé autant que possible. Deuxièmement, la randomization devrait inclure le partitionnement aléatoire des données pour les ensembles d'entraînement, de validation et de test, qui s'avère être la plus importante des sources de variance. Troisièmement, des tests statistiques tel que la version du Mann-Withney U-test présenté dans notre article devrait être utilisé plutôt que des comparisons sur la simple base de moyennes afin de prendre en considération l'incertitude des mesures de performance.

Dans le chapitre 7, nous présentons un cadriciel d'optimisation d'hyperparamètres développé avec principal objectif de favoriser les bonnes pratiques d'optimisation des hyperparamètres. Le cadriciel est conçu de façon à privilégier une interface simple et intuitive adaptée aux habitudes de travail des chercheurs en apprentissage automatique. Il inclut un nouveau système de versionnage d'expériences afin d'aider les chercheurs à organiser leurs itérations expérimentales et tirer profit des résultats antérieurs pour augmenter l'efficacité de l'optimisation des hyperparamètres. L'optimisation des hyperparamètres joue un rôle important dans les bancs de test, les hyperparamètres étant un facteur confondant significatif. Fournir aux chercheurs un instrument afin de bien contrôler ces facteurs confondants est complémentaire aux recommandations pour tenir compte des sources de variation dans le chapitre 6.

Nos recommendations et l'outil pour l'optimisation d'hyperparametre offre une base solide pour une méthodologie robuste et fiable.

Mots clés: apprentissage automatique, reproductibilité, optimisation d'hyperparamètres

Abstract

The recent revolution in machine learning has been strongly based on the use of standardized benchmarks. Providing clear target metrics and undeniable measures of improvements of learning algorithms, they are at the center of the scientific methodology in machine learning. They do not ensure validity of results however, therefore some scientific conclusions based on flawed methodology may prove to be wrong.

In this thesis we address this question by first raising the issue (Chapter 5), then we study it to find solutions and recommendations (Chapter 6) and build tools to help improve the methodology of researchers (Chapter 7).

In first article, Chapter 5, we demonstrate the issue of reproducibility in stable and consensual benchmarks, implying that these issues are endemic to a large ensemble of machine learning applications that are possibly less stable or less consensual. We raise awareness of the important impact of stochasticity even in stable image classification tasks and contend that solutions for reproducible benchmarks should account for this stochasticity.

In second article, Chapter 6, we study the different sources of variation that are typical in machine learning benchmarks, measure their effect on comparison methods to benchmark algorithms and provide recommendations based on our results. One important contribution of this work is that we measure the reliability of a cheaper but biased estimator for the average performance of algorithms. As explained in the article, an ideal estimator involving multiple rounds of hyperparameter optimization is too computationally expensive. Most researchers must resort to use the biased alternative, but it has been unknown until now how serious a degradation of the quality of estimation this leads to. Our investigations provides guidelines for benchmarks on practical budgets. First, as many sources of variations as possible should be randomized. Second, the partitioning of data in training, validation and test sets should be randomized as well, since this is the most important source of variation. Finally, statistical tests should be used instead of ad-hoc average comparisons so that the uncertainty of performance estimation can be accounted for when comparing machine learning algorithms.

In Chapter 7, we present a framework for hyperparameter optimization that has been developed with the main goal of encouraging best practices for hyperparameter optimization.

The framework is designed to favor a simple and intuitive interface adapted to the workflow of machine learning researchers. It includes a new version control system for experiments to help researchers organize their rounds of experimentations and leverage prior results for more efficient hyperparameter optimization. Hyperparameter optimization plays an important role in benchmarking, with the effect of hyperparameters being a serious confounding factor. Providing an instrument for researchers to properly control this confounding factor is complementary to our guidelines to account for sources of variation in Chapter 6.

Our recommendations together with our tool for hyperparameter optimization provides a solid basis for a reliable methodology in machine learning benchmarks.

Keywords: machine learning, reproducibility, hyperparameter optimization

Contents

List of tables

List of figures

Acknowledgements

I am deeply grateful to my supervisor Pascal Vincent for supporting me very early in my academic formation, as a first year under-graduate, and throughout my whole bachelor, master and PhD. There is so much we cannot learn in books but only through our mentors. Thank you for believing in me and supporting me through this parental-PhD roller coaster.

It has been an immense pleasure and honor to collaborate with Gaël Varoquaux during the last year of my PhD. His invaluable advices, his principles and his energy had a profound influence on the progression of one of the project that is dearest to me.

I am also grateful to César Laurent and Thomas George with whom I had the pleasure to collaborate early on. I cannot thank enough César Laurent for his impromptu collaboration on my first work on reproducibility as well as Michael Noukhovitch for his amazing feedback.

During the past 4 years, I had the chance of collaborating with dozens of incredible persons on the development of the library Oríon. I am deeply indebted to Frédéric Bastien and Pascal Lamblin for their support and for sharing their wisdom on software development and open source practices. I hold great memories of an extraordinary collaboration and friendship with Christos Tsirigotis. I already miss working with Thomas Schweizer, probably the most organized and methodical person I have ever met. I am grateful to Yonggang Hu and all his team at IBM for a very fruitful collaboration. I would like to thank all the people who collaborated with me on Oríon; Guillaume Alain, Sébastien Arnold, Frédéric Bastien, Christopher Beckham, Arnaud Bergeron, Hadrien Bertrand, Olexa Bilaniuk, Steven Bocco, Olivier Breuleux, Mirko Bronzi, François Corneau-Tremblay, Pierre Delaunay, Lin Dong, Mathieu Germain, Nadhir Hassen, Reyhane Askari Hemmat, Peter Henderson, Yonggang Hu, Pascal Lamblin, Junfeng Liu, Fabrice Normandin, Michael Noukhovitch, Satya Ortiz-Gagné, Frédéric Osterrath, Irina Rish, Thomas Schweizer, Dmitriy Serdyuk, Dendi Suhubdy, Christos Tsirigotis, Sean Wagner and Chao Xue.

I will be forever thankful for the team of known and unknown collaborators who jumped on-board for a very ambitious project right in the surge of a pandemic. May 2020 was by far the craziest month of my PhD, thank you! To Pierre Delaunay who worked ceaseless on a fundamental framework for our experiments, to Mirko Bronzi who patiently setup and

executed thousands of experiments, to Assya Trofimov who persisted on finding a proper application for biology, fought her way through thousands of experiments and helped so much with figures and text, to Brennan Nichyporuk, Naz Sepah and Justin Szeto who worked and learned so much to achieve executing thousands of long experiments on a short dead-line, to Edward Raff for his continuous help with the writing of the paper and to all other collaborators, Samira Ebrahimi Kahou, Kanika Madan, Vincent Michalski, Dmitriy Serdyuk, Gaël Varoquaux, Pascal Vincent and Vikram Voleti, thank you! I am also grateful to Christian Léger who generously helped me better understand bootstrapping and statistical testing, and to Jessica Thompson for sharing insightful papers and books on research methodologies, philosophy of science and statistical testing.

I would like to thank Joumana Ghosn for three great internships at Nuance, as well as Mirko Bronzi, David Huggins Daines, Sylvain Goulet, and Xiaohua Liu.

To conclude, I would like to acknowledge collaborators on projects I eventually abandoned in favor of my work on reproducibility and hyperparameter optimization. I am thankful to Francis Dutil for joining me with César Laurent to work on regularization functions. And last but not least, it has been a pleasure to collaborate with Guillaume Lajoie, Simon Guiroy and Valentin Thomas. Thank you all very much for your precious time, it is deeply appreciated.

Finalement, tout ceci n'aurait été possible si ce n'avait été du soutient de ma mère, Lise Gaudreault et de mon père, Louis Bouthillier. Je souhaiterais aussi remercier mes beaux-parent, Monique Lefebvre et Daniel Thibodeau pour tout ce qu'ils m'ont apporté. Le plus profond et le plus sincère des remerciements à ma conjointe, Catherine Thibodeau Lefebvre, ainsi qu'à mes garçons, Raphaël et Ludovic Bouthillier. Des études doctorales en tant que père de famille sont d'abord et avant tout un travail d'équipe. Merci infiniment à toute l'équipe!

Introduction

Intelligence is an elusive concept, one that is particularly difficult to define precisely. In an attempt to both understand and simulate intelligence, a variety of school of thoughts appeared throughout the past century. For each school of thoughts, different sets of problems and different methodologies were favored. Traveling through time across these different perspectives helps understand the current deterministic view of the scientific approaches in artificial intelligence.

The first dominant view of artificial intelligence revolved around logic (Newell & Simon, 1956) based on the principle that rationality is the foundation of human intelligence, distinguishing it from other animals. Although being successful in mathematical puzzles, logic fell short of solving real world problems lacking well defined mathematical models. By the end of the 70s, knowledge-based and expert systems gained popularity among scientists by leveraging the ever increasing memory size of computers (Lindsay et al., 1993). These handcrafted systems lacked however perception and intuition, making it difficult to interact with a real world full of eluding concepts (ex: how can we define precisely what is a cat in an image?). Born around the same time as most sub-fields of artificial intelligence (Rosenblatt, 1958; Samuel, 1959), machine learning eventually blossomed in the 90s (Boser et al., 1992; Cortes & Vapnik, 1995) allowing statistical modelization of some eluding concepts.

These paradigm changes, from logic to expert systems and from expert systems to machine learning entailed tremendous changes throughout a few decades in the perception of problems and research practices. Starting from a very rational and deterministic view of problem solving, the focus of researchers gradually shifted towards more noisy real-world problems. Around the end of the 1980s, research practices of machine learning followed this trend and gradually departed from logic-based approaches in favor of statistical approaches (Langley, 2011).

Despite the adoption of statistical approaches, the research methodologies remained fairly impervious to statistical tools developed in other scientific fields such as medicine or biology. Researchers would use statistical learning theory to build and analyse machine learning methods, but the experimentations would be interpreted with little regard to their stochastic nature. According to Langley, the shifts have been partly supported by well curated problems

(benchmarks) used to guide the progress of researcher in artificial intelligence, thus somehow shielding the field from real-world stochasticity other scientists must handle.

By the 2010s, methods based on neural networks were gaining interest of researchers in the field of machine learning (Hochreiter & Schmidhuber, 1997; LeCun et al., 1998; Bengio et al., 2003; Hinton et al., 2006; Collobert et al., 2011). In 2011-2012, the works of Hinton et al. (2012) in speech recognition and Krizhevsky et al. (2012) in image classification provided unmatched improvements, literally sparkling a revolution in machine learning. While Hinton et al. (2012) and Krizhevsky et al. (2012) inspired global change of practical approaches in machine learning, the work of Zhang et al. (2016) epitomize the change of paradigm at a theoretical level, explicitly stating the change of theoretical puzzles to come with the new paradigm. With the help of graphics cards dramatically speeding up computations, larger datasets and new training methods, deep neural networks became the dominant paradigm in machine learning.

Change of paradigm comes with change in methodologies. Cross-validation which has been until then the only widely used practice to handle stochasticity due to data sampling suddenly became less popular. With larger datasets, cross-validation was believed to be less useful and rather impractical due to computational cost of training neural networks on larger datasets. This set-back in research practices led to several criticisms (Kadlec et al., 2017; Henderson et al., 2018; Lucic et al., 2018; Melis et al., 2018) as well as fueled initiatives for computational reproducibility in machine learning (Forde et al., 2018; Pineau et al., 2020; Fursin, 2020).

Main criticisms were that 1) scientific publications tended to share too little information to enable reproduction of the experiments, 2) variability of the results was not accounted for, leading to misleading benchmarking of machine learning algorithms, and 3) confounding factors such as fine-tuning of algorithm configuration were not controlled properly leading again to misleading benchmarking.

The work of this thesis positions itself in this wave of criticisms, by 1) asserting that experimental practices are problematic and would not be solved solely by sharing more information (ex: data and code sharing) although we recognise the importance of doing so, 2) studying the sources of variability in machine learning benchmarks and proposing guidelines to account for this variance in benchmarks, and 3) building a hyperparameter optimization framework to encourage better controlling the fine-tuning of algorithm configuration.

In the first article, Chapter 5, we argue against the importance of code sharing and other practices of computational reproducibility, demonstrating that current methodologies are inadequate to provide reliable benchmarks in the first place. Without depreciating the undeniable value of open science, we support that unreliability of benchmarks is not compensated by open science and that more attention should be applied to variability of benchmarks.

We then turn from criticism to analysis and problem solving in the second article, Chapter 6. We study common sources of variations in various machine learning applications and measure the reliability of comparison methods to draw conclusions on the superiority of machine learning algorithms. We shed light on important sources of variation which were typically neglected and unknowingly undermined the quality of our comparison methods.

Throughout these works I alternated my software engineering and research scientist hats, leading the development of a framework for hyperparameter optimization, Oríon, described in Chapter 7. Our goal while building Oríon was not to improve existing hyperparameter optimization algorithms, but rather to create an instrument for scientific inquiry in machine learning research. Hence our focus has been on usability to favor wide adoption by researchers, encouraging a better control over fine-tuning of algorithm configuration.

In short, these works shed light on current methodological issues, proposed guidelines to improve reliability of benchmarks and provided tools to help improve their reliability further.

Before delving into the technical contributions, I will first cover the fundamental topics required to understand the three articles. I will present important theories of philosophy of science in Chapter 1 that will help understand the importance and nuances of methodologies to conduct research in machine learning. A brief presentation of statistical inference in Chapter 2 will then provide the basics to understand the importance of statistical testing for reproducibility. I will present some parts of machine learning from an unusual angle in Chapter 3, as a discipline to automate knowledge acquisition, bridging together philosophy of science and machine learning. Finally, although hyperparameter optimization can be considered as part of machine learning, I will cover the topic in details, dedicating Chapter 4 to it. The next three chapters will cover the main contributions of my thesis. Chapter 5 will discuss shortcomings of common methodologies for reproducibility. Chapter 6 will extend on these issues to study the effect of variance on the reliability of machine learning benchmarks. Chapter 7 will provide a brief description of the open-source software Oríon for hyperparameter optimization. Finally, I will conclude my thesis discussing promising avenues for hyperparameter optimization and reproducibility in machine learning.

Chapter 1

Philosophy of Science

Beginning a thesis on the topic of machine learning with a chapter on philosophy of science is certainly unorthodox. The rationale behind this decision is that, first, scrutinizing the research methodologies in machine learning requires a remote perspective to help identifying issues and goals. Second, machine learning and science are close enough that philosophy of science can serve as an interesting viewpoint to understand some parts of machine learning.

Reproducibility can serve as a good example of a concept requiring a broader perspective to be better understood, the terminology itself being contentious (Barba, 2018; Plesser, 2018). As we shall see, there are multiple definitions of reproducibility and the goals are subject to debate (Devezer et al., 2019). Starting from the broad perspective of philosophy of science on reproducibility, we will narrow it down to the perspective of statistical inference in Chapter 2 before narrowing it down further to the perspective of machine learning and hyperparameter optimization in Chapters 3-4.

Philosophy of science and machine learning are intricately related. Not only can machine learning research be viewed as a science and thus an object of analysis for philosophy of science, but also the process of machine learning itself can be viewed as a scientific process. Science aims at acquiring knowledge through methods that can ensure the trustworthiness of this knowledge. Machine learning aims at building systems that can automatically learn to solve tasks, in other words systems that can acquire knowledge to make accurate predictions. Both machine learning as a science and the process of machine learning can be objects of analysis for philosophy of science. We will not be using philosophy of science to actively study machine learning but rather draw insightful parallels from it in Chapter 3.

We will begin our journey with Karl Popper, discussing how theories may or may not be confirmed by observations through inductive reasoning (Section 1.1). The limitations of Popper's view will lead us to Imre Lakatos and Thomas Kuhn, discussing the biases of observations and interpretations. Finally these issues will bring us to Peter Godfrey-Smith and Deborah Mayo, paving the way to a procedural and statistical view of hypothesis

testing (Section 1.3). But before crossing this bridge to statistical testing in Chapter 2, we will discuss *reproducibility* from the perspective of philosophy of science (Section 1.4).

1.1. Falsificationism and Demarcation; contradiction with observations

Observations, products of our sensory experience, is all we can rely on to assess the truth of theories about nature according to empiricism. This view of the world became particularly strong at the beginning of the 20th century. The Vienna circle, an influential group of philosophers and scientists, worked extensively on applications of logic to confirm theories on the basis of observations (Godfrey-Smith, 2009).

This effort was deemed fruitless by Karl Popper for the simple reason that the truth of a theory could not be proved using inductive reasoning, which is the only reasoning approach available to us when working solely with observations (Popper, 1934, 2005). For Popper, theories about nature could not be proven, only falsified. The principle can be easily illustrated with mathematics.

Let $\mathcal{X}$ the unknown infinite set of all possible observations consistent with an unknown phenomenon (s.a. a natural phenomenon). Let $x \in \mathcal{X}$ a single observation, and let us denote $x \in \mathcal{A}$ when x satisfies some property A (that we can verify) and $x \in \mathcal{B}$ when x satisfies some property B. We suppose that we have access to a finite set of n actual observations $\{x_1, \cdots, x_n\} \subset \mathcal{X}$. Let us consider a theory stating the following:

$$\forall x \in \mathcal{X}, x \in \mathcal{A} \implies x \in \mathcal{B}. \tag{1.1.1}$$

Proving this theory would required to prove it for all $x \in \mathcal{X}$. Verifying the truth of this statement for a finite number of observations $\{x_1, \cdots, x_n\}$, however large, cannot serve as a proof. However the statement might be more easily disproved as it would suffice to find but a single x_i such that $x_i \in \mathcal{A}$ but $x_i \notin \mathcal{B}$ to prove that the theory is wrong. This is what Popper calls falsifying, finding observations which proves a theory to be wrong.

It follows that for Popper, there are no *true* theories. There are falsified theories, and not yet falsified theories. This was for him both an attempt at prescribing how science should be and at describing how scientists work.

His prescriptions were his answer to the *demarcation problem*, that is, stating whether a theory is scientific or not. If the theory is falsifiable, then it is scientific, if it is not falsifiable, then it is not scientific. He considered this criterion to be both necessary and sufficient.

These prescriptions entail two important steps for the scientific inquiry. Researchers must first define a theory, how it could be falsified, and then seek out observations in an attempt to falsify it. We can recognise here two steps of the hypothetic-deductive method, often dogmatically known as *the scientific method* (Godfrey-Smith, 2009), so in some sense

Popper was right in his description of the methodology of scientists. It falls short however of describing many behaviors. Although the dogmatic *scientific method* dictates that a scientist must reject a falsified hypothesis, it is often not what scientists do while they continue resolutely pursuing the study of their hypothesis. To some account, something must be wrong with falsification.

1.2. Research programs and paradigms; bias in observations

Falsification is appealing by its simplicity but it is also one of its weaknesses. Scientists do not automatically reject falsified theories, and they do so rationally. Perhaps the measurement instruments are to blame, or perhaps some errors occurred during the experiments. Popper holds we should attempt falsifying everything, but this is not practically possible.

Imre Lakatos described the behavior of scientists as if they were following research programs which guided what they would attempt falsifying and what they would not (Lakatos & Musgrave, 1970; Lakatos, 1976). A *research program* is a set of theories and practices adopted by a group of researchers. It comprises a *hard core* of theories and practices and a *protective belt* of theories and practices based on the hard core. According to Lakatos, researchers will attempt falsifying the parts of the protective belt only.

A few years earlier than the work of Lakatos, Thomas Kuhn published the profoundly influential book "The Structure of Scientific Revolutions" (Kuhn, 1962) in which he defended a similar dynamic. For Kuhn, science is separated in two categories, the normal science and the scientific revolution (or crisis).

Normal science is when scientists have adopted a paradigm, similar to the concept of research program from Lakatos, and work solely within the framework defined by it. A paradigm also determines *puzzles* that scientists may work on. As normal science proceeds, some puzzles are solved and others resist until the inability to solve them leads to a crisis. During this crisis, different paradigms emerge and clash with the predominant one until the community of researchers settle on a new one and re-enter normal science possibly working on a different set of puzzles.

For Kuhn, normal science is both dogmatic and efficient. The adherence of researchers to dogmas allows them to focus on the puzzles with most potential. Obviously Kuhn trusts scientists to adhere to reasonable dogmas. They are to Kuhn what the *hard core* is to Lakatos.

So far we avoided an issue about the nature of the observations that becomes manifest with research programs and paradigms; observations and their interpretations are biased. Additionally, observations may be misleading due to confounding factors. Thus scientists are reasonably skeptical when faced with surprising observations. Falsification as proposed

by Popper does not handle these issues, nor do Lakatos and Kuhn. The latters only describe these issues.

1.3. Procedures and severe testing; probability of observations

One common difficulty in empiricism and positions alike is that the observations, deemed to be the sacrosanct objective resource to reach truth, are actually deeply impregnated with theories. On one side, observations in scientific experiments are the product of inquiries that are based on theories. On the other side, observations, be they outcome of experiments or not, are interpreted on the basis of theories and prior beliefs.

According to philosopher Peter Godfrey-Smith these issues can be handled by procedural naturalism (Godfrey-Smith, 2009). It is not sufficient to attempt falsification with observations, we must also define precise procedures to ensure the observations are sufficiently informative. Quoting Godfrey-Smith (2009, p. 214):

> *"The main idea I will defend is that we should analyze evidence, confirmation, and testing by focusing on* procedures*. If an observation provides support for a theory, that will be by virtue of the procedure that the observation was embedded within."*

A good example for this is the generic procedure of randomization. The simple fact of randomizing observations reduces confounding effects and even support discoveries of causal influences (Godfrey-Smith, 2009; Pearl & Mackenzie, 2018).

By combining a procedural approach with modeling assumptions we obtain a probabilistic description of what the observations may look like. This probabilistic description allows us to measure to what extent a theory is tested, to what extent the observations we have may be representative of all possible observations. We saw earlier that, according to Popper, accumulating observations cannot prove a theory. It may not prove it, but now we can have a sense of confidence of its level of agreement with observations. Mayo calls this improvement to inductive reasoning the *Lift-off*: "An overall inference can be more reliable and precise than its premises individually." (Mayo, 2018).

Thanks to statistical inference we can now reconcile with inductive reasoning and build a sense of confidence on which theories are best candidates based on observations. But the procedure used to obtain these observations is critical. A quote from Fisher (1938, p. 17), a prominent figure of statistical inference, provides an incisive illustration of the importance of properly designing the experimental procedures:

> *"To consult the statistician after an experiment is finished is often merely to ask him to conduct a post mortem examination. He can perhaps say what the experiment died of."*

There is little statistical inference can do with observations if the experimental procedures are flawed.

We will see in the next Chapter a statistical testing approach that is distinctively procedural, the Neyman-Pearson approach. But before moving on, let us first discuss reproducibility from the perspective of philosophy of science.

1.4. Reproducibility

Robert Boyles' air pump replication experiment is a famous historical example supporting the importance of reproducibility (Shapin & Schaffer, 1985). The existence of vacuum, a very controversial idea at the time, was supported by a repeatable experiment. An experiment alone would not have been enough to convince skeptics, but the successful reproductions of the experiments rendered the results unquestionable, or at least more difficult to question.

For Popper, reproductions of experiments are of limited utility. Falsifying a theory is what sheds light on the theory's weaknesses and where the researchers should investigate next. It is by falsifying theories that science progress. Kuhn shared a similar view, in that puzzles are what drives science forward.

But how can we falsify theories if researchers are working within a research program or a paradigm and may reject surprising results? The example of Boyle is interesting in that it combines together the view of Popper and Kuhn along with severe testing to answer this question. According to Boyle, failed reproductions are opportunities for further investigations and improvement of theories, while successful reproductions are a way to ensure the robustness of the theories with respect to "*wantonness or other deviation of Nature*" (Bishop & Gill, 2020). A corollary to the second point, is that successful reproductions are a reliable way of convincing skeptics and forcing changes of research programs or paradigms. Reproducing a scientific experiment can be both useful if it fails, to guide researchers on what to investigate, or if it succeeds, to ensure robustness of theories.

But are all replications equally useful? We can try answering this question by flipping falsificationism upside down. A successful replication is a failed attempt at falsifying a theory. It follows that for a replication to be informative, it must have a reasonable chance of failing and consequently falsifying the theory. A theory must make general predictions which are tested through reproductions. Fixing the whole universe in a given state so that the experiment is *perfectly reproducible* brings absolutely no information with respect to general predictions. It cannot falsify them. This view of *perfect reproducibility* is nevertheless common in computational sciences for which the full control over the experimental environments is possible. We will severely criticize this view in Chapter 5, contending that ensuring *perfect* reproduction does not ensure general and insightful reproductions.

In the rest of this thesis, we will concentrate on reproducibility as a mean to ensure robustness of scientific conclusions with respect to sources of variations that are considered irrelevant by researchers. We will now move down from philosophy's stratospheric perspective to statistical inference, more precisely statistical testing, before touching ground in Chapters 3-4 to cover machine learning and hyperparameter optimization.

Chapter 2

Statistical Testing

Statistical testing grounds progress in empirical science, extracting reliable findings in the face of noisy evidence. Sources of variations abound in machine learning despite its computational nature making it highly controllable in comparison to natural sciences. Researchers neglecting these sources of variations are at risk of being misled by experimental results.

In this chapter I will present the basics of the statistical approach and tools used in Chapter 6. Building upon the previous chapter, statistical inference is a tool that can be used to ensure severe testing of our theories. Perhaps failing to falsify a theory does not imply that the latter is true, but failing to falsify a theory after it has been severely tested is certainly indicative of its reliability. Our main goal using these statistical tools, will be to weed out results that could be explained by mere noise.

There are no widely accepted statistical approaches and still much debate on the topic (Mayo, 2018). Among the different approaches, those generally present at the center of debates are the Neyman-Pearson hypothesis testing, fiducial inference and Bayesian inference. The most common approach across science in general is however an unfortunate mix of the Neyman-Pearson hypothesis testing and an early version of fiducial inference, called null-hypothesis testing, which is strongly criticized by all camps.

In this thesis I will focus on Neyman-Pearson hypothesis testing as it is the most appropriate for our use case. One of the most common criticisms towards this approach is that it leads to a binary decision (McShane et al., 2019). Of two possible hypothesis $\mathcal{H}_0$ and $\mathcal{H}_A$, the test shall serve as deciding whether $\mathcal{H}_0$ or $\mathcal{H}_A$ should be taken to be true. Although the test may be highly useful for some parts of science, not all inquiries in science can be formulated as a binary question and the criticism is certainly justified. In our case however, we will use this approach on a problem that is already formulated as a binary question by researchers. Hence, we are not distorting scientific methods to satisfy the requirements of Neyman-Pearson hypothesis testing.

In Section 2.1, I will present the basics of Neyman-Pearson hypothesis testing and distinguish it from null-hypothesis testing. We will go through fundamental concepts such as the test error-rates, statistical power and confidence intervals, all illustrated with a simple Z-test.

In Section 2.2, I will describe the Mann-Whitney U test, its special formulation as a *probability index* and its relation with the Student t-test. This test plays an important role in our contributions in Chapter 6.

Finally, I will conclude this chapter with a few words on reproducibility from a statistical point of view in Section 2.3.

2.1. Neyman-Pearson Approach

Null-hypothesis testing is ubiquitous for most scientific fields and is a source of confusion (Mayo, 2018). A helpful way of understanding the Neyman-Pearson approach without confusing it with null-hypothesis testing is to follow the history of its creation.

At the beginning of the 20th century, Ronald A. Fisher had laid the ground of several fundamental principles of statistical science such as the p-value, level of significance based on mathematical models of a population and null-hypothesis (Fisher, 1922, 1932, 1936).

Figure 2.1 illustrates the p-value of a null-hypothesis as used by Fisher. Let X and Y be two random variables from normal distributions of unknown averages μ_X, μ_Y representing the efficacy of two medications, and a known equal variance σ^2. We want to verify whether the new medication Y is worth the additional production cost over X. We define the null-hypothesis $\mathcal{H}_0$ as if Y provided no improvements, thus $\mathcal{H}_0 : \mu_X - \mu_Y = 0$. For our example, we model the population as $\mathcal{N}(\mu_X - \mu_Y, 2\sigma^2)$. The p-value is the probability of obtaining the average $\mu_X - \mu_Y$ under this model.

Fisher recommended the use of the p-value as an indicator for the degree of agreement between the data and the null-hypothesis. For him, this indicator should serve as a guide for further investigations. The lower the p-value, the higher the level of significance, indicating that the null-hypothesis may be false. If we set a level of significance α, we have a measure of the rate of false positives if $\mathcal{H}_0$ is true, also known as the type I error. The use of a fixed level of significance $\alpha = 0.05$ serving as a hard threshold is specific to null-hypothesis testing and was borrowed from Neyman-Pearson approach, but lacks many important aspects of the later that we shall now see. According to Fisher, α should not be fixed.

Jerzy Neyman and Egon S. Pearson were concerned about the asymmetry of the statistical test in its binary form with a fixed α ($\mathcal{H}_0$ is either true of false). A simple example illustrates well their concerns. Suppose we set the significance level to some α. This was not recommended by Fisher, but in the end we need to make a decision whether we believe $\mathcal{H}_0$ is true or not, so let's fix α. We have that in the long run, repeating the test multiple

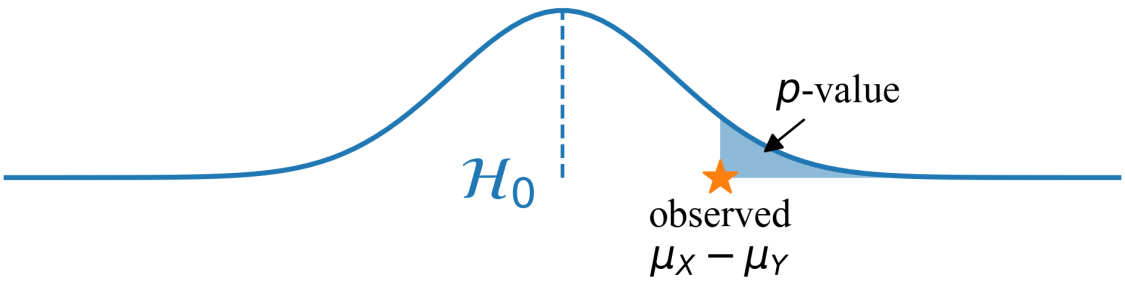


Fig. 2.1. Null-hypothesis and p-values from Fisher. The blue curve represents the null-hypothesis $\mathcal{H}_0$, the probability density of a normal distribution with its mean represented as the dashed line. The orange star represents an observed difference $\mu_X - \mu_Y$. The p-value is represented by the blue area under the curve on the right of the observed difference, which corresponds to the probability of observing values above $\mu_X - \mu_Y$ if $\mathcal{H}_0$ is true. The p-value is often misunderstood as being the probability of the null-hypothesis given the data. We are not estimating the probability of the null-hypothesis however, the null-hypothesis rather defines the probability model under which the p-value is calculated. For Fisher, the p-value is a useful way of measuring the level of agreement of the null-hypothesis with the data. The p-value being small is indicative that the null-hypothesis may be false, warranting further investigation.

times, we would reject the null hypothesis only 5% of the time if it is true. This rate is true only if $\mathcal{H}_0$ is true however. When $\mathcal{H}_0$ is false, all bets are off. It may be that under the true population, we would reject $\mathcal{H}_0$ only 5% of the time even though it is false! In such scenario the results would be uninformative as they would be equally likely under $\mathcal{H}_0$ and $\neg\mathcal{H}_0$. To avoid such scenario, Neyman and Pearson believed the test should be framed as a comparison of hypotheses such that the odds of the test could be made symmetrical.

In their pioneering paper, Neyman & Pearson (1933) formalized what would be latter known as the Neyman Pearson lemma, introducing the notions of type II errors, statistical power and a critical region (or effect size).

As previously mentioned, type I errors are false positives, when $\mathcal{H}_0$ is rejected while it is actually true (ex: convicting an innocent). The type II errors are false negatives. These happen when $\mathcal{H}_0$ is not rejected while it is actually false (ex: not convicting a guilty person). Table 1 illustrates the two type of errors. The statistical power is the probability of rejecting $\mathcal{H}_0$ when it is false. From Fisher we had the significance level, $p(\text{rejecting } \mathcal{H}_0 \mid \mathcal{H}_0) = \alpha$, to which Neyman-Pearson now adds the statistical power $p(\text{rejecting } \mathcal{H}_0 \mid \neg\mathcal{H}_0) = 1 - \beta$.

Table 1. Two types of errors for statistical tests. Type I error are also known as false positives (claiming an effect when there are none) and Type II errors as false negatives (claiming no effect where there is one).

Decision	$\mathcal{H}_0$ is true	$\mathcal{H}_A$ is true
Don't reject $\mathcal{H}_0$		Type II error
Reject $\mathcal{H}_0$	Type I error	

Let us reuse our previous example of X and Y representing the efficiency of two different medications, with again the null-hypothesis $\mathcal{H}_0$ that Y provides no improvements over X, $\mathcal{H}_0 : \mu_X - \mu_Y = 0$. We now add an alternative hypothesis with a minimal effect size based on the Neyman-Pearson hypothesis testing approach. To be worth the additional production cost over X, the medication represented by Y must be more efficient by at least δ, thus $\mathcal{H}_A : \mu_X - \mu_Y \geq \delta$. The value δ is known as the *expected effect size*, that is, the difference we expect to find between μ_X and μ_Y. This is a striking difference with respect to null-hypothesis testing, in that it explicitly states that differences below δ will be regarded as corroborating the null-hypothesis. In more practical terms, Y is not worth the additional cost if $\mu_X - \mu_Y < \delta$ and we therefore chose the status quo $\mathcal{H}_0$. The region between 0 and δ is also known as the critical region.

The Neyman-Pearson lemma gives us a formulation of what would be the most powerful test for a rejection region that is governed by the expected effect size δ. Thanks to this lemma we can now control both α and β, ensuring low error rates of both type I and II.

The quality of our estimation of μ_X and μ_Y depends on the variance and the number of samples. Using a preliminary estimation of the variance[1] and the Neyman-Pearson lemma we can identify what minimal number of samples should be used to achieve a statistical power of $1 - \beta$ at least. This important process is best known as the *sample size determination*.

The Neyman-Pearson approach uses a null-hypothesis to ensure statistical significance, a feature shared with null-hypothesis testing. The main differences are the control of the statistical power and the definition of an expected effect size. These differences are primordial as we discussed earlier, because we would otherwise have no guarantees when $\mathcal{H}_0$ is false.

Confidence intervals can be used to conduct a test following the Neyman-Pearson approach. In our example, the confidence interval would be given by

$$\mu_X - \mu_Y \pm z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}}, \tag{2.1.1}$$

where $z_{\frac{\alpha}{2}}$ is the standard score for a significance level of α based on a Gaussian model with known variance. This interval will cover the true difference $\mu_X - \mu_Y$ about $100(1-\alpha)\%$ of the time. If $\mu_X - \mu_Y = 0$ is within this region, then the null hypothesis is considered reasonable. Otherwise, if $\mu_X - \mu_Y \geq \delta$ intersects with the interval, then the alternative hypothesis is considered reasonable. The Neyman-Pearson approach covers two scenarios that cannot be identified with null-hypothesis testing, illustrated in Figure 2.2. The first one is when both hypotheses are not covered by the confidence interval, in which case we conclude that there may exist a difference but this difference is too small to be valuable. The second one is when both hypothesis are covered, a typical scenario with test of too small

[1]This can be done by running preliminary experiments or based on prior similar experiments in the literature.

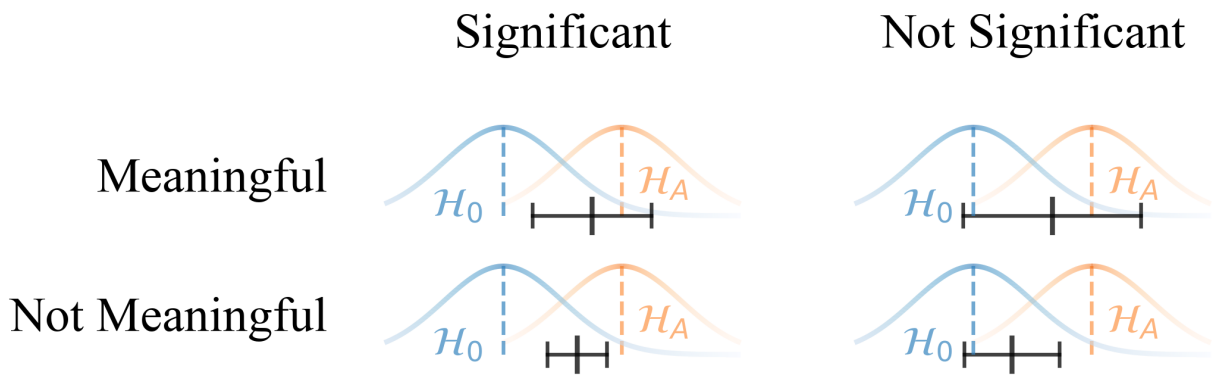


Fig. 2.2. Illustration of significant and meaningful results according to Neyman-Pearson. Given a null hypothesis $\mathcal{H}_0$ and an alternative hypothesis $\mathcal{H}_A$, there are four possible scenarios when analyzing results. Left column, confidence intervals show statistically significant results, the null hypothesis $\mathcal{H}_0$ illustrated as the dashed blue line is not included in the interval. Right column, confidence intervals show non significant results as they contain the null hypothesis. Top row, confidence intervals show statistically meaningful results as they include the alternative hypothesis $\mathcal{H}_A$ illustrated as the dashed orange line. Bottom row, confidence intervals show non meaningful results as the alternative hypothesis is not contained. Null-hypothesis testing, which focus solely on statistical significance with respect to $\mathcal{H}_0$ misses scenario at top right (not significant but meaningful) and at bottom left (significant but not meaningful). It is extremely valuable to be able to determine that a non significant result could be meaningful, as this implies that the statistical power of the test was too weak and could not yield insightful results. Likewise, it is critical to ensure that a statistically significant result is also statistically meaningful. Large sample sizes can turn tiny differences into significant results even though they are of limited practical importance.

power, in which case we can only state that the test was not powerful enough and the results are inconclusive.

To conclude this section, I would like to discuss one of the the main criticisms against the Neyman-Pearson approach. We saw that to determine the sample size we first need to define both hypotheses and the expected effect size. In other words, we must design precisely the experiments before we can design the test, which will then allow us to finally execute the experiments. For many statisticians, this seems unnecessarily stringent. For them, data is data and it should serve as evidence (Mayo, 2018). The first chapter of this thesis finds part of its purpose here to support the Neyman-Pearson approach. As we saw when discussing falsification, falsifying for the sake of it is insufficient. A sincere attempt at falsifying should be made if we were to gain trust in the results. From the point of view of procedural naturalism, the procedure is determinant for the reliability of the data. Quoting again the same passage from Godfrey-Smith (2009, p. 214)

> *"If an observation provides support for a theory, that will be by virtue of the procedure that the observation was embedded within."*

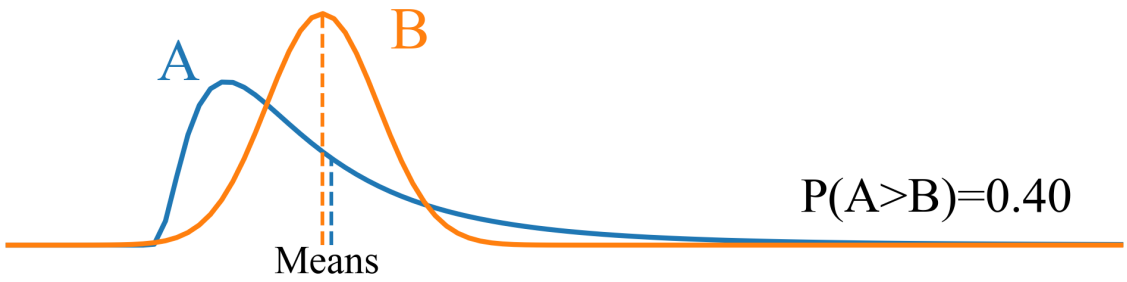


Fig. 2.3. Non-parametric Mann-Whitney U-test for non-normal distributions. Two distributions A (blue) and B (orange) are compared. Log-normal distribution A has a mean larger than the normal distribution B, both represented as dashed lines of corresponding colors. A statistical test using averages would lead to the conclusion that A is better than B, whereas a ρ-statistic would give $\mathbb{P}(A > B) = 0.4$, therefore leading to the opposite conclusion that B is greater than A. This example shows how the p-statistic can be more reliable when working with non-normal distribution. Although the mean of A is larger than the mean of B, the bulk of its probability density is on the left of B and thus samples from A will be smaller than samples from B about 60% of the time.

2.2. Mann-Whitney U test

The Student t-test is a powerful and widely applicable statistical test thanks to the omnipresence of normal distributions in our world. When unsure what distribution our random variable of interest may be coming from, it is safer to revert to non-parametric statistical tests. A non-parametric test does not assume a specific model for the data and is thus more robust. They are however less powerful than parametric ones when the data is indeed distributed according to the model of a parametric test.

The Mann-Whitney U test (Mann & Whitney, 1947) is a very simple non-parametric test for which there exists a formulation that is particularly well suited for our use case. We will not cover the details concerning the general formulation as this is not the one we will use in Chapter 6 and instead describe right away the ρ-statistic formulation.

Let X and Y be two random variable with unknown distributions, and let $x_1, \cdots, x_n$, and $y_1, \cdots, y_n$ be independent and identically distributed (i.i.d.) samples from the distributions of X and Y respectively. Let $r()$ be a function describing the ranks such as

$$r(x, y) = \begin{cases} 1, & \text{if } x > y \\ \frac{1}{2}, & \text{if } x = y \\ 0, & \text{if } x < y \end{cases} \tag{2.2.1}$$

The ρ-statistic is equal to the average of ranks over values of X and Y.

$$\rho = \frac{1}{n} \sum_i r(x_i, y_i), \tag{2.2.2}$$

Ideally, values of x and y are paired to reduce the variance which increases the statistical power of the test. The value ρ can be interpreted as the probability of a sample from X to be greater than a sample from Y, more formally $\mathbb{P}(X > Y)$.

This test can be used with the Neyman-Pearson approach by using it in combination with a confidence interval (Perme & Manevski, 2019).

We define the null-hypothesis $\mathcal{H}_0 : \mathbb{P}(X > Y) = 0.5$ and the alternative hypothesis $\mathcal{H}_A : \mathbb{P}(X > Y) > \gamma$, where γ is the expected effect size. We can use Noether's sample size determination equation to compute the sample size (Noether, 1987).

$$n \geq \left(\frac{\Phi^{-1}(1-\alpha) - \Phi^{-1}(\beta)}{\sqrt{6}(\frac{1}{2} - \gamma)} \right)^2 \tag{2.2.3}$$

Where Φ^{-1} is the inverse cumulative function of the normal distribution, α is the significance level and β corresponds to the statistical power.

One convenient property of the Mann-Whitney U test is that for a sample size of $n > 20$, the U statistic becomes approximately normally distributed. Also, Mann-Whitney U test is about 0.95 the efficiency of a t-test when applied on data that is approximately normally distributed (Lehmann, 2004, Example 3.4.3). Otherwise, it is considerably more efficient than the t-test on data that is not normally distributed.

2.3. Reproducibility

A reproducible statistical test is one that leads to the same conclusion when executed on another set of samples coming from the same distribution. Both the level of significance and the statistical power of a test are deeply related to the notion of reproducibility, as they represent the probability of drawing the right conclusion depending on whether $\mathcal{H}_0$ or $\mathcal{H}_A$ is right.

In practice, researchers tend to neglect the statistical power of tests. In 2005, physician-scientist John Ioannidis published a controversial article claiming that most published research findings are false (Ioannidis, 2005). A strong part of the argument of Ioannidis was based on the bias in favor of positive results induced by publishing incentives (also known as the file-drawer effect (Rosenthal, 1979)) and the small sample sizes used leading to statistical tests of very small power. Some statisticians have criticised the bold claims of Ioannidis (Goodman & Greenland, 2007; Jager & Leek, 2014), arguing that Ioannidis' own assumptions were not justified by data, and that correcting them accordingly leads to predictions of 8 to 17% false positive rates, much lower than the 50% reported by Ioannidis.

A few large scale replications of experiments have been carried out since the last decade in psychology and medicine in an attempt to measure the rate of false positives (Begley & Ellis, 2012; Collaboration et al., 2015; Camerer et al., 2018; Klein et al., 2018). These

initiatives total hundreds of replication attempts with more than 50% of them failing to replicate. These results tend to corroborate Ioannidis' predictions but it is hard to know to what extent these numbers are representative as there may be a selection bias in favor of smaller experiments that are easier to replicate in practice. Smaller sample sizes have lower statistical power and are thus more likely to lead to false positives. Nevertheless, these numbers are far from reassuring. Clearly, the power of a statistical test is of great importance for reproducibility and should not be neglected. Both type I and II errors should be controlled properly.

One solution that is gaining traction in various fields is preregistration (Nosek et al., 2018). The idea is to submit to a journal a description of the experiment that the researchers plan to carry out. This submission is reviewed based on the validity of the experiment design and statistical test. Once the preregistration is accepted, the experiment is carried out and the results are published, be they positive or negative. This helps ensure the use of proper statistical tests and reduces the file-drawer effect. Detractors of this method consider it too rigid and believe it harms the pace of progress in science unnecessarily (Devezer et al., 2020). In the same line of thought, it is believed that preregistration should not serve as a gate-keeping tool because not all science can be twisted into series of statistical tests (Navarro, 2019).

In machine learning, benchmarking is a central tool to measure the progress of learning algorithms. Although it serves as a controlled environment to test learning algorithms it is not devoid of stochasticity and thus requires statistical tools to draw reliable conclusions. As we will see in Chapters 5 and 6, an important part of the reproducibility issue in machine learning is due to the lack of sound statistical tests in benchmarks. We will now dive into machine learning and hyperparameter optimization in the next two chapters.

Chapter 3

Machine Learning

As we observed in Chapter 1, science is mostly based on inductive reasoning to infer general laws of nature. The same goes for machine learning. In science, we assume there exist general laws describing the observations we make. In mathematical terms, say we observe x and y, we assume there exists a function $f(x) = y$ describing the relation between x and y under some distribution for x and y (ex: x being images of cats or dogs and y being the corresponding labels `cat` or `dog`). In the attempt to discover f, scientists will posit modelizations h that approximate f. Similarly, machine learning consists of training learning algorithms to find models h that best approximate f based on some cost functions $\mathcal{L}$. While learning h, the algorithm is distilling information from observations, it is building structured knowledge. This knowledge is built into h. The structured knowledge here, is more practical than descriptive. We care about solving a task, not describing some hidden laws of nature. It does not mean however, that learning laws of nature is of no practical use, and we could therefore imagine that these laws could as well be learned by machine learning for practical purposes. I will use this analogy to science to introduce machine learning which will hopefully help clarify the epistemological considerations behind machine learning research.

We can use an analogy similar to the Little Man Computer (Englander & Wong, 2021) to understand the basics of learning algorithms. The Little Man Computer is a simple model of a computer, in which a little man emulates the operations of a computer, fetching data from memory, executing operations, storing data, etc. Let me now present the Little Scientist Computer, a pictorial machine learning algorithm.

The Little Scientist Computer may have goals of different natures, it may want to solve specific tasks, to simply find general structures in observations or to leverage observations to guide decisions. These goals are driving different learning paradigms that we will cover in Section 3.1.

Within this learning paradigm, the Little Scientist Computer will operate through a sequence of procedures to acquire reliable knowledge (Section 3.2). First, the Little Scientist

Computer is born with a set of biases, its research program in some sense[1], which will affect the wide array of possible hypotheses h it could infer from observations (Modelization phase, Section 3.2.1). Given readily available observations, or given an environment to interact with and gather observations, the Little Scientist Computer will seek a best hypothesis h to serve its goal (Optimization phase, Section 3.2.2). It will then seek to test whether its hypothesis h generalizes to unseen observations (Evaluation phase, Section 3.2.3). When faced with multiple reasonable hypotheses h, the Little Scientist Computer will be forced to select the best candidate (Model selection phase, Section 3.2.4).

Machine learning researchers generally focus on the modelization phase or optimization phase when evaluating their new learning algorithm contributions. In this thesis, I will argue that researchers should have a more holistic view of the learning algorithm to evaluate their contributions, they should evaluate the Little Scientist Computer as a whole. We will elevate ourselves by one level of abstraction in Section 3.3 and take a look at the process of creating learning algorithms. In this last section we will see how we can compare the effectiveness of different Little Scientist Computers, which will set the stage for the contributions of Chapter 5-6 and expose the main motivation for Chapter 7.

3.1. Learning Paradigms

I will present here three broad families of tasks: 1) Supervised learning, 2) Unsupervised learning and 3) Reinforcement learning. Note that although they are presented separately, they are not mutually exclusive.

Supervised learning is a predictive approach, we attempt to predict targets based on observations. Unsupervised learning on the other hand is descriptive, we attempt to uncover generic patterns in observations. Reinforcement learning is an approach for interactive environments, we attempt to learn a policy of best actions an agent should take to optimize its rewards. Reinforcement learning is a decision making approach. It can use predictive and descriptive (probability estimation) approaches but ultimately it is aimed at making decisions.

3.1.1. Supervised Learning

For any observations x, we have a corresponding target y. As presented in the introduction of this chapter, we assume that there exists a function f such that $f(x) = y$. Supervised learning aims at learning a function h that approximates f.

Typical tasks for supervised learning are regression and classification. It is a regression tasks when targets are of continuous nature – e.g. the length of a fish. When the targets are distinct classes – e.g. different species – then it is classification.

[1]Refer back to Section 1.2 for information on *Research Programs*.

Supervised learning is relatively easy to evaluate. Given an ensemble of observations (x,y) we can compute a cost function $\mathcal{L}$ that is representative of the model's performance on the observations[2]. There are generally two main problems with supervised learning however. One problem is the quality, representativeness, or reliability of the dataset. For example, an ensemble of images with a specific type of illumination may cause reliability issues if the model h obtained is used on images of different illuminations. This is a relatively harmless example, but there are many examples of such bias causing ethical issues such as increased misclassification rates based on gender and skin type (Buolamwini & Gebru, 2018) or gender biases in natural language models (Bolukbasi et al., 2016). Recently Torralba, Fergus and Freeman withdrew the 80 Million Tiny Images dataset after Prabhu & Birhane (2020) found it to contain derogatory classes and offensive images.

Another problem is the cost of generating large labeled datasets requiring significant amount of manual work to create the targets. One solution to this is semi-supervised learning, a combination of supervised learning and unsupervised learning.

3.1.2. Unsupervised Learning

We have observations x, and that is all. We assume that there exists a structure in the underlying distribution of the observations that can be learned. Unsupervised learning can aim at estimating a probability density, at compressing x, at predicting parts of the observations given the rest or at clustering x.

An example of a density estimator is the Parzen-Rosenblatt window estimator (Rosenblatt, 1956; Parzen, 1962) that we will revisit in Section 3.2.1.1. In short, a function κ is used to simulate a density around each observations x, and the average of the probability density is computed for a given x'.

The auto-encoders are an example of learning algorithm that compresses x by learning an encoding function h and a decoding function g such that $g(h(x)) \approx x$. The encoding function $h(x)$ may be a vector of smaller dimensionality than x, or a sparse vector of larger dimensionality.

Language models in natural language processing are an example of tasks where we attempt to predict parts of the observations given the rest. The language model itself is used to compute the probability of a sequence of words $p(w_1, w_2, \cdots, w_n)$, but it is trained by attempting to predict correctly $p(w_n \mid w_1, w_2, \cdots, w_{n-1})$. The term w_n can be interpreted as a target, thus the task is similar to supervised-learning. The main difference is that w_n was not labeled manually but is rather part of the basic observations x.

The main difference between classification and clustering is that there are no a-priori defined classes when clustering. Clustering can be a method to infer different classes.

[2]See Section 3.2.3 for more explanations on the evaluation of learning algorithms

Evaluating unsupervised learning is generally problematic because the metrics used to evaluate the performance of unsupervised models are surrogates. The density estimation or encoding of x are often evaluated as features in supervised-learning settings to assess their general utility. Language models for instance, are routinely evaluated on benchmarks of various supervised-learning tasks after being trained in an unsupervised manner (Wang et al., 2018).

One of the main disadvantage of supervised learning is the cost of labelling large datasets. As unsupervised learning can help learning useful features for supervised learning, a combination of both, semi-supervised learning, is a popular solution with large ensembles of data containing only a small portion of labeled examples.

3.1.3. Reinforcement Learning

With this approach, no observations are available upfront. The algorithm is an agent interacting with an environment and must explore and exploit information it acquires as it goes, in order to maximize rewards. The observations are dependent of the agent's decisions.

This paradigm is not covered directly in this thesis, but it plays an important role in reproducibility literature in machine learning due to its unstable nature and complexity (Henderson et al., 2018). The interactivity of the agent with the environment leads to more variable experiments than models trained on fixed datasets, and it can be even worse if the environment is stochastic.

We will see in Section 3.2.3 that common evaluation strategies in reinforcement learning are at odds with falsificationism and could be considered as *unscientific* by conservatives. I shall note right away that this should not serve as a depreciation for the undeniably valuable approach of reinforcement learning but should rather serve as a warning for how research is conducted.

3.2. Learning and selecting a hypothesis

To learn an approximation h of f given observations x and y, we must first define the mathematical framework within which h may be represented. This framework is called the hypothesis class $\mathcal{H}$ and will be determined by the design of the learning algorithm . This step may be viewed as the choice of assumptions that will be the substrate of possible hypotheses in $\mathcal{H}$ (Section 3.2.1).

Given a hypothesis class $\mathcal{H}$, we then search for a best model h based on the observations: this is framed as an **optimization** problem. It is during this step that knowledge is distilled from observations (Section 3.2.2).

If $\mathcal{H}$ contained every possible functions and we were able to search through all of $\mathcal{H}$, then we might learn $h = f$, but we might instead learn a function h that memorizes or

perfectly fits the finite set of observations without extracting any *general* knowledge from these observations. To evaluate the fitness of h, how well it approximates f, we must test it on unseen observations, i.e. observations that were not used for the optimization. This is the **evaluation** step where we put our knowledge to the test (Section 3.2.3).

Finally, given different initial hypothesis classes $\mathcal{H}$ or different optimizations over $\mathcal{H}$ we may end up with different h to test. In this final **model selection** step, we select the best h according to its generalization to unseen observations rather than based on its performance on initial observations (Section 3.2.4).

3.2.1. Modelization – Mathematical framework for knowledge

We assumed there exists a function $f(x) = y$ describing the relation between observations x and y. As we cannot access this function, we will attempt to model it with some function h. First, we define a set of functions, the hypothesis class $\mathcal{H}$, which we believe may contain f.

We will distinguish two main forms h may take; a parametric and a non-parametric form[3]. The parametric form is a function that contains a finite set of parameters that can be tuned. For example, let h be a polynomial such as $a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$. The ensemble of coefficients $\{a_0, a_1, \cdots, a_n\}$ are the parameters of the polynomial. Different functions in $\mathcal{H}$ corresponds to different coefficients, different parameter values. On the other hand, the non-parametric form is a function that contains no parameters or a variable number of parameters scaling with the number of examples. A simple example is the k-nearest neighbors algorithm (Altman, 1992). Given an ensemble of points X and a query point x', it computes the majority class among the k nearest points of x' in X based on a distance metric (s.a. the Euclidean distance, or the Hamming distance).

In both examples, we witness variables that are not learned; n, the degree of the polynomial and k, the number of nearest neighbors to consider. These are called hyperparameters. They are a lever that the experimenter can adjust to modify the hypothesis class $\mathcal{H}$.

For the polynomial, as we increase n, we increase the *capacity* of $\mathcal{H}$, that is its richness, the "number" of different functions h that can be found in $\mathcal{H}$. Inversely for the k-nearest neighbors algorithm, as we decrease k, we increase the capacity. Indeed, with k being equal to the number of examples, $\mathcal{H}$ will only contain a single h, a constant h that always predicts the majority class over all points.

The selection of a model h from hypothesis class $\mathcal{H}$ will be biased if $\mathcal{H}$ is too narrow and does not include f. On the other hand, the selection of a model h will have a high variance if $\mathcal{H}$ is very large, leading to selecting widely different h for slightly different sets of observations. The selection of h goes beyond modelization therefore we will not discuss

[3]There exists overlapping forms but I will omit them for simplicity as none are used in Chapters 5-7.

it further in this section. We will see in Sections 3.2.2-3.2.4 how a model h is selected in $\mathcal{H}$, how it is evaluated and compared with other h, but for now we will focus on the design of learning algorithms, the design of $\mathcal{H}$.

3.2.1.1. Parzen-Rosenblatt window estimator

A very simple and flexible learning algorithm for density estimation is the Parzen-Rosenblatt window estimator (Rosenblatt, 1956; Parzen, 1962). It uses kernel functions κ (the windows) to estimate the density around known points. Let $\mathcal{D} = \{x_1, x_2, \cdots, x_n\}$ be a set of known points independently sampled from an unknown distribution with a density f, and $x_\star$ be a query point where we want to infer $f(x_\star)$. The density function estimated by the Parzen-Rosenblatt window estimator is given by

$$h(x_\star) = \frac{1}{n} \sum_{i=1}^{n} \kappa(x_\star - x_i) \tag{3.2.1}$$

Where κ typically is a normal density function. κ can be parametrized with a width factor that can be static or adaptive. We will see in Section 4.3 of next chapter an adaptive version where the width, specifically the σ of the normal density function, is adjusted based on the distance of the closest points in the neighborhood.

As a non-parametric learning algorithm, the capacity of the Parzen-Rosenblatt window estimator will grow with the number of samples, n. The type of the kernel will affect the capacity as well, particularly if its width is very large (low capacity) or very narrow (high capacity).

3.2.1.2. Gaussian Process

The Parzen-Rosenblatt window estimator was an example of a non-parametric unsupervised learning algorithm. We now move to a non-parametric supervised learning algorithm, the Gaussian Process (Rasmussen & Williams, 2005).

A Gaussian process is a generalization of the Gaussian probability distribution that allows learning a distribution of functions of the form $h(x) = y$. It does not learn h per se, but a distribution of possible h based on observations x and y.

To better understand the mechanism of the Gaussian process, let's first build our intuition based on a Gaussian probability distribution.

Suppose we have a bivariate normal distribution for random variables X_1, X_2, with mean μ_1, μ_2 and covariance $\mathbf{\Sigma}$.

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \mathbf{\Sigma}_{11} & \mathbf{\Sigma}_{12} \\ \mathbf{\Sigma}_{21} & \mathbf{\Sigma}_{22} \end{bmatrix} \right) \tag{3.2.2}$$

From Theorem 4.3.1 (Murphy, 2012, sec. 4.3.1), the posterior conditional is given by

$$p(x_2 \mid x_1) = \mathcal{N}(x_2 \mid \mu_{2|1}, \Sigma_{2|1}) \tag{3.2.3}$$

$$\mu_{2|1} = \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(x_1 - \mu_1) \tag{3.2.4}$$

$$\Sigma_{2|1}^2 = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12} \tag{3.2.5}$$

Notice that the posterior conditional $p(x_2 \mid x_1)$ depends on the correlation between X_1 and X_2, but does not depend on observations of X_2 directly. Suppose we add a new dimension X_3 for which we have no observations. All observations so far only included values for X_1 and X_2. What we just observed about $p(x_2 \mid x_1)$ implies that if we were able to approximate some covariance for X_1, X_2 and X_3, we could estimate the posterior conditional $p(x_3 \mid x_1, x_2)$.

Elements of Σ, the correlation matrix, can be interpreted as the alignment between points i and j. To simulate this, we can resort to kernels as presented in Section 3.2.1.1. A kernel will measure some form of similarity between two points. For the sake of the example we will use a simple non-parametric squared exponential kernel, a.k.a. the Radial Basis Function kernel or Gaussian kernel.

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{K}_{ij} = e^{-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}} \tag{3.2.6}$$

This kernel $\boldsymbol{K}$ will be our approximation of a correlation $\boldsymbol{\Sigma}$ between the variables. There is a twist here however. We are concerned about representing a function $f(x) = y$, not about the distribution of some random variables X. The problem is that f is a function and to represent it we must attempt to model all of its domain, an infinity of random variables. That is why we will model the Gaussian distribution as a multivariate normal distribution of size n – the number of observations. The kernel approximation $\boldsymbol{K}$ will be based on the similarity of the inputs $\boldsymbol{x}$, assuming that the function f is smooth so that similar x lead to similar y.

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \boldsymbol{K}_{11} & \boldsymbol{K}_{12} & & \boldsymbol{K}_{1n} \\ \boldsymbol{K}_{21} & \boldsymbol{K}_{22} & \cdots & \boldsymbol{K}_{2n} \\ & \vdots & \ddots & \\ \boldsymbol{K}_{n1} & \boldsymbol{K}_{n2} & & \boldsymbol{K}_{nn} \end{bmatrix}\right) \tag{3.2.7}$$

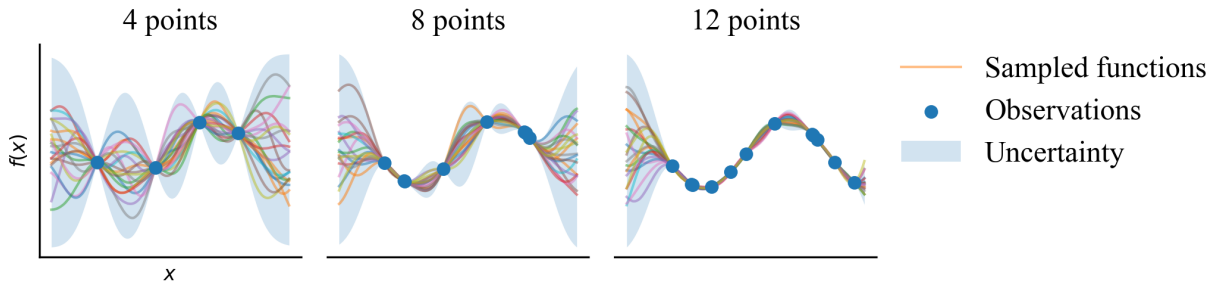


Fig. 3.1. Gaussian processes on a toy function. On each plot, a Gaussian process is fitted on a different number of observations (4, 8 and 12). X-axis is the input x and y-axis is the output of $f(x)$ and $h(x)$. Blue points are the observations, shaded area is the uncertainty (σ) of the Gaussian process, and lines of various colors are functions $h(x)$ sampled from the Gaussian process. As we gather more observations, the uncertainty of the Gaussian process decreases and all sampled functions h gets closer.

If we want to compute $h(x_\star)$, our approximation of $f(x_\star)$, we will add a new dimension to this model so that we can compute its posterior conditional.

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \\ y_\star \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \begin{bmatrix} \boldsymbol{K}_{11} & \boldsymbol{K}_{12} & & \boldsymbol{K}_{1n} \\ \boldsymbol{K}_{21} & \boldsymbol{K}_{22} & \cdots & \boldsymbol{K}_{2n} \\ & \vdots & \ddots & \\ \boldsymbol{K}_{n1} & \boldsymbol{K}_{n2} & & \boldsymbol{K}_{nn} \end{bmatrix} & \begin{bmatrix} \boldsymbol{k}_{1\star} \\ \boldsymbol{k}_{2\star} \\ \vdots \\ \boldsymbol{k}_{n\star} \end{bmatrix} \\ \begin{bmatrix} \boldsymbol{k}_{\star 1} & \boldsymbol{k}_{\star 2} & \cdots & \boldsymbol{K}_{\star n} \end{bmatrix} & k_{\star\star} \end{bmatrix}\right) \tag{3.2.8}$$

Which can be rewritten using a compact notation.

$$\begin{bmatrix} \boldsymbol{y} \\ y_\star \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \boldsymbol{K} & \boldsymbol{k}_\star \\ \boldsymbol{k}_\star^T & k_{\star\star} \end{bmatrix}\right) \tag{3.2.9}$$

From Equation 3.2.3, we compute the posterior conditional $p(y_\star \mid \boldsymbol{y}, \boldsymbol{X}, x_\star)$

$$p(y_\star \mid \boldsymbol{y}, \boldsymbol{X}, x_\star) = \mathcal{N}(y_\star \mid \mu_\star, \Sigma_\star) \tag{3.2.10}$$

$$\mu_\star = \mathbb{E}\left[y_\star\right] = \boldsymbol{k}_\star^T \boldsymbol{K}^{-1} \boldsymbol{y} \tag{3.2.11}$$

$$\sigma_\star^2 = k_{\star\star} - \boldsymbol{k}_\star^T \boldsymbol{K}^{-1} \boldsymbol{k}_\star \tag{3.2.12}$$

This conditional is a distribution over functions $h(x)$ that represents well $f(x)$ based on the data. To get a function $h(x_\star)$, we sample from $\mathcal{N}(\mu_\star, \sigma_\star^2)$.

The Gaussian Processes have the advantage of requiring little specifications towards the nature of f. The assumptions underlying the choice of kernels will narrow the space of possible function $\mathcal{H}$ that can be represented by the Gaussian Process, but it will remain nevertheless particularly expressive. It is less true however for large dimensional data where the flexibility of the Gaussian Process will become its weakness. We will now enter the

world of parametric learning algorithms, where we will make stronger assumptions on f, at the risk of severely biasing $\mathcal{H}$ but with the potential of harnessing observations of larger dimensionality.

3.2.1.3. Neural Networks

The current excitement about artificial intelligence is strongly due to recent successes with deep neural networks in machine learning. This modelization approach dates back to the beginning of the field of artificial intelligence but it has long been an underdog. During the early 70s, many researchers incorrectly interpreted the work of Minsky & Papert (1969) as stating that neural networks could not learn a simple XOR problem, making it useless, whereas Minsky & Papert (1969) statement was limited to a single layer Perceptron. Larger neural networks were trained during the 80-90s to solve more complex problems thanks to advances in back-propagation algorithms (Rumelhart et al., 1986). Nevertheless, statistical learning theory predicted that large models would not generalize well if trained on too few observations (Vapnik, 1998). Yet, it worked. Theoreticians are still struggling to understand why so large neural networks can learn functions h close to f while they have the capacity of learning functions simply memorizing the observations. Most recent progress looks at not only the nature of $\mathcal{H}$ but also at the optimization procedures to search through $\mathcal{H}$ and the nature of the observations (Jiang et al., 2020). This subsection will focus on modelization only, optimization of neural networks will be discussed in Section 3.2.2.

A neural network, or more precisely an artificial neural network, is a structured ensemble of artificial neurons connected with weights that can process signals. Given an observation x at its input layer, a pattern of activation will emerge within the neural network eventually leading to a pattern of activation at the output layer which is interpreted as the prediction of the neural network.

In its simplest form, a neural network is a simple linear regression model.

$$h(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{x}^T \boldsymbol{W} + \boldsymbol{b} \tag{3.2.13}$$

Where x is the input, $\boldsymbol{W}$ are the weight parameters and $\boldsymbol{b}$ are the bias parameters. A basic structure of neural network is simply a stack of such functions, which we call layers.

$$
\begin{aligned}
\boldsymbol{z}^{(0)} &= \boldsymbol{x} \\
\boldsymbol{z}^{(1)} &= a^{(1)}(h^{(1)}(\boldsymbol{z}^{(0)}; \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)})) \\
\boldsymbol{z}^{(2)} &= a^{(2)}(h^{(2)}(\boldsymbol{z}^{(1)}; \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)})) \\
&\vdots \\
\hat{y} &= a^{(l)}(h^{(l)}(\boldsymbol{z}^{(l-1)}; \boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}))
\end{aligned}
\tag{3.2.14}
$$

Where $a^{(i)}$ is an activation function. Stacking linear layers is barely useful, as it is equivalent to single but larger linear layer. Activation functions turn these linear functions $h^{(i)}$ into non-linear functions. This enables retrieving higher-order statistics from the observations as the signal flows through multiple layers.

The first predominant activation function in the literature was the sigmoid function.

$$
\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.2.15}
$$

This function tends to saturate activations however which hampers the flow of the signal through the neural network, making it more difficult to train. More recently, the rectified linear-unit became the defacto activation function (Nair & Hinton, 2010).

$$
\text{relu}(z) = \max(0, z) \tag{3.2.16}
$$

Finally, the softmax function is present at the last layer of about every neural networks applied on classification tasks. It turns the activation of the last layer into a normalized form that can be interpreted as a probability.

$$
\text{softmax}(\boldsymbol{z}) = \frac{e^{z}}{\sum_j e^{z_j}} \tag{3.2.17}
$$

There exists many other activation functions that we will not cover here. Although the choice of activation function can affect the capacity of the neural network, the most important effect that we can observe and which motivated the strong adoption of the ReLU is on the optimization of the neural network. Indeed ReLUs helped stabilizing the flow of the signal through the network which facilitated training. There is indeed a strong relation between how we define $\mathcal{H}$ and how we can search through it.

The version of neural network we have seen so far is called a fully-connected feed forward neural network. At each layer, neurons are connected to every neurons of the previous layer.

In theory, large enough fully-connected feed forward neural networks are universal approximators (Cybenko, 1989; Hornik, 1991), meaning that $\mathcal{H}$ could contain any possible f.

Unfortunately we did not have the chance to witness such specimen in the wild yet. We will see in Section 3.2.2 that the difficulty of optimizing the networks may render us incapable of finding f even though $\mathcal{H}$ contains it.

There exists alternatives to fully-connected networks that can leverage assumptions on the function we aim to learn to approximate f. In most cases $\mathcal{H}$ will retain its expressivity but it will become easier to search through it.

An obvious example is the convolutional layer (Fukushima & Miyake, 1982; LeCun et al., 1989). Instead of connecting neurons to all previous ones, the convolutional layer will *convolve* a group of neurons – called a kernel – over the input. This will allow learning functions h that are invariant to translations more easily, like the position of an object on an image.

Convolutional layers can handle inputs $\boldsymbol{x}$ in the form of sequences $x_1, x_2, \cdots, x_l$ but it cannot represent directly long-term dependencies across the sequences. A solution for this are the recurrent neural networks (Rumelhart et al., 1985; Jordan, 1986) which learns a mapping $z_t = h(x_t \mid z^{t-1})$ tracking an earlier state z^{t-1}. Similarly to deep neural networks, recurrent neural networks have issues with the flow of the signal through the operations over the sequences, a problem commonly known under the name of *vanishing gradient* (Hochreiter, 1991; Bengio et al., 1994; Pascanu et al., 2013). The *long short-term memory* (LSTM) (Hochreiter & Schmidhuber, 1997) and the *gated recurrent unit* (GRU) (Cho et al., 2014; Chung et al., 2014) were proposed to attenuate this issue. Variants of recurrent neural networks still struggle however with sequences longer than typical sentences in language applications. Very recently, Vaswani et al. (2017) proposed a new architecture – the Transformer – that can handle sequences the size of documents thanks to the use of attention layers (Bahdanau et al., 2015).

Decisions in the design of the architecture of a neural network can bias $\mathcal{H}$ towards family of functions that are more representative of the data (i.e convolutional layers for computer vision), or help facilitating the search through $\mathcal{H}$ (i.e ReLU activations).

These few examples are representative of a broad set of possible modelization in machine learning. Let us now look at the optimization phase, the search in $\mathcal{H}$ for h.

3.2.2. Optimization – Acquiring Knowledge

We will focus here on optimization of neural networks, as this is the object of study in this thesis.

Both modelization and optimization are part of the learning algorithm, they are both part of the Little Scientist Computer. The Little Scientist Computer has chosen $\mathcal{H}$, that is, its research program within which it will conduct its research – the neural network architecture is chosen. The Little Scientist Computer will then search through $\mathcal{H}$ to find a hypothesis h, a model, that will minimize an evaluation function based on the observations.

Let S^t be the set of observations guiding our search for h. We will define a metric of interest $e(h(x), y)$ which will measure how far our predictions $h(x)$ are with respect to target observation y corresponding to x. The model h should minimize this metric over all observations in S^t, the *empirical risk* $\hat{R}_e(h, S^t)$.

$$\hat{R}_e(h, S^t) = \frac{1}{|S^t|} \sum_{(x,y)\in S^t} e(h(x),y) \tag{3.2.18}$$

In practice, the Little Scientist Computer will be apprehensive to find a model h that fails on unobserved data and will therefore add regularization techniques to encourage the search to visit certain regions of $\mathcal{H}$ over others. Let us denote this regularization with $\Omega(h, \lambda)$, where λ is a hyperparameter influencing the strength of the regularization. Also, it may not be possible to directly optimize the metric of interest e and we will often need to optimize a surrogate loss e' instead. We will represent this search of h as the optimization $\text{Opt}(S^t, \lambda)$.

$$\text{Opt}(S^t, \lambda) \approx \arg\min_{h\in\mathcal{H}} \hat{R}_{e'}(h, S^t) + \Omega(h, \lambda), \tag{3.2.19}$$

Alternatively, and equivalently, Ω can instead be included as an additional term in defining e' in which case we will just be minimizing $\hat{R}_{e'}$ that now includes the regularization. For notational simplicity we will thus drop Ω from now on while discussing optimization.

Neural networks are generally optimized with stochastic gradient descent or variants of it. Stochastic gradient descent is a single-order optimization technique, meaning that it only uses the gradient of the surrogate loss. Let $\boldsymbol{\theta}$ be the ensemble of all parameters of model h. A gradient descent will update the parameters based on the gradient of the empirical risk $\nabla_\theta \hat{R}_{e'}(h, S^t)$.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_\theta \hat{R}_{e'}(h, S^t) \tag{3.2.20}$$

Where η is a hyperparameter, called the learning rate, controlling the size of the steps made during optimization. For a large datasets S^t, the gradient is however needlessly time consuming to compute for all observations in S^t at once, this is why we revert to stochastic gradient descent and estimate the gradient with only a small subset of S^t, a *mini-batch*. Let $\boldsymbol{g}$ be the gradient approximation over a mini-batch, given by

$$\boldsymbol{g} = \frac{1}{m} \sum_{i=1}^{m} e'(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}). \tag{3.2.21}$$

And thus the update rule for the parameters become

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \boldsymbol{g}. \tag{3.2.22}$$

This gradient approximation leads to less reliable steps during optimization, but we will see in Section 3.2.3 that the noise induced by the approximation turns out to be beneficial in some contexts.

There exist popular alternatives or complementary methods to improve the speed of convergence of stochastic gradient descent. Momentum for instance uses a decaying linear combination of past gradient updates to speed stochastic gradient optimization (Rutishauser, 1959; Polyak, 1964; Sutskever et al., 2013). The most popular alternative to stochastic gradient descent in recent literature is Adam, which leverages running averages of the gradients and their second moment (Kingma & Ba, 2015).

So far we focused on the movements in $\mathcal{H}$ during optimization and ignored the initial conditions. The weights and biases of a neural network must be initialized at the beginning of the optimization, which conditions where we may search in $\mathcal{H}$. Glorot & Bengio (2010) studied the effect of the distribution used to randomly initialized weights and found that the following uniform distribution favored a better flow of the signal through deep neural networks with sigmoid activation functions. Let m and n be the size of input and output of a layer.

$$\boldsymbol{W}_{i,j} \sim U(-1, 1)\sqrt{\frac{6}{m+n}} \tag{3.2.23}$$

The ReLU activation shortly after became the defacto activation function and He et al. (2015) revised the distribution with

$$\boldsymbol{W}_{i,j} \sim U(-1, 1)\sqrt{\frac{2}{m}}. \tag{3.2.24}$$

The most important point to remember from this section on optimization is the stochastic nature of it. Due to this stochasticity and the fact that deep neural networks yield non-convex optimization problems, if the Little Scientist Computer was to perform two searches in $\mathcal{H}$ it would almost invariably find different hypothesis h. Both the initial point and the process, as we use random mini-batches, are noisy. We will come back to this in Section 3.3.

3.2.3. Evaluation – Testing Knowledge

Suppose we have found some model h in $\mathcal{H}$ as described in the previous section. We cannot know yet if it is truly a good model even though we selected it by optimizing the empirical risk on our observations. We do not care primarily about fitting h on the available observations, what we care about is finding an h which generalizes to any observations under the same distribution. We must put h to the test, try falsifying it in order to determine if it has the potential of being a good approximation for f, the true function we are after.

The *empirical risk* was the average value of the metric on the available observations. What we would want to have ideally is an *expected risk* over all possible observations from our distribution of interest. Let $\mathcal{D}$ be the distribution from which x,y are sampled independently. The expected risk is then given by

$$R_e(h, \mathcal{D}) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[e(h(x),y)] \tag{3.2.25}$$

Only, we do not have access to $\mathcal{D}$ and even if we had we cannot measure infinitely many prediction errors $e(h(x), y)$. Thus, we revert again to the *empirical risk*, but this time using a different set of observations S^o which were not used during the optimization. If S^o contained only observations from S^t, and $\mathcal{H}$ contained functions that can memorize S^t, then we could not *falsify* our model h. The Little Scientific Computer would be *unscientific* according to Popper's account of demarcation[4].

This separation is generally well observed in supervised and unsupervised learning but has been a source of issues in reinforcement learning (Henderson et al., 2018). There are no clear separation of environments into sets of training and testing observations because the agent is driving the generation of observations. There are nevertheless obvious cases where the environment is so large that the model never managed to observe most states and actions. One can think of the great achievement of mastering Go well enough to win against a world champion (Silver et al., 2017). We could not be sure however prior to the competition against a world champion whether AlphaGo generalized well outside its training environment. The competition has been the ultimate attempt at falsifying it.

The use of the term *falsify* can be confusing when discussing the empirical risk on testing observations. The empirical risk is not binary, either pass or fail, but rather an average performance. Falsifying here would mean finding testing observations that h is not able to predict. It will rarely be the case that h can predict correctly every testing observations, but it is also true for science in general. Theories are not simply *falsified* or *not falsified*. A choice must be made between alternative theories based on their levels of agreement with new observations. It does not mean the concept of falsification is pointless however. The point is to stress the importance of the procedural evaluation so that the results can be deemed trustworthy. We will see in the next section how choices between competing models h can be made, as well as how regularization and resampling techniques can help select best candidates h across different searches through $\mathcal{H}$.

[4]Refer back to Section 1.1 for information on *demarcation*.

3.2.4. Model selection – Choosing the best hypothesis

A choice must be made between alternative theories based on their level of agreements with new observations. What are different models h and on what basis would we choose to compare them?

3.2.4.1. Hyperparameters and regularization

First of all, defining $\mathcal{H}$ is no trivial matter. The choice of the architecture, the number of layers, the type of layers, the type of activation functions is highly arbitrary and will affect the potential models h we can obtain. These choices are part of the hyperparameters that we may optimize. This is a different optimization than the one in Section 3.2.2 however. In this case we are optimizing hyperparameters, the configuration of $\mathcal{H}$, rather than the parameters. We are defining $\mathcal{H}$ instead of searching through it. There are also other hyperparameters that will affect how we search through $\mathcal{H}$ such as the learning rate. We will say that these hyperparameters influence the *effective* $\mathcal{H}$, that is, the explorable regions of $\mathcal{H}$.

As we compare evaluation of models h on testing observations we will start *regularizing* $\mathcal{H}$ to avoid *overfitting* the training observations and favor generalization. An extreme example of *overfitting* is when h perfectly predicts training observations but fails to correctly predict any testing observations, as if h simply memorized the training observations only.

Regularization can take different forms, one being an explicit additional cost $\Omega(h, \lambda)$ as previously denoted in Equation 3.2.19. One widely used regularization cost is the L_2-regularization, also known as ridge regression when applied to linear regression models (Hoerl & Kennard, 1970). This technique limits the explorable regions of $\mathcal{H}$ by penalizing models h with too large weights norms.

Another form of regularization is noise injection. Srivastava et al. (2014) proposed for instance to randomly shut-off neurons in neural networks to limit their ability to *co-adapt*. This means neurons needed to learn to detect patterns that are more globally distributed.

Regularization can be applied as a cost or as a modification of the model, but also as a transformation on the training observations themselves – better known as data augmentation. Suppose we have a set of images each representing either a cat or a dog. We can safely assume that slightly translated version of the images would not alter the nature of the animal depicted in the images. By applying such random translation we will regularize our optimization. All h in $\mathcal{H}$ that would simply memorize training observations would be unfit in the face of transformed training observations. The *effective* $\mathcal{H}$ would thus be reduced, hence why we can classify data augmentation as a regularization.

As we modify $\mathcal{H}$ with regularization we also run the risk of missing out good approximations of f. There is a delicate balance between capacity and generalization. Because both capacity and generalization are loosely defined, finding a balance is more an art than

a science so far. The best solution available is an automated optimization of this balance using hyperparameter optimization techniques (Chaptor 4).

This leads however to another problem, we could overfit the testing observations while optimizing the hyperparameters. Let Λ be the space of possible configutations of values of hyperparameters. Say we find a best $\lambda \in \Lambda$ so that h is optimal on the testing observations. On what basis can we trust this result? For instance, λ could be defining the entire initialization state of the optimization. In such a scenario, the hyperparameter optimization could entirely bypass the optimization phase in $\mathcal{H}$ and actually directly search through $\mathcal{H}$. Although this is an extreme and unlikely scenario it helps illustrate the problem of optimizing the hyperparameters on the testing observations. We are back to case 1, we need to be able to falsify the h obtained by optimizing λ.

In order to solve this issue, we can divide the observations into three sets instead of two. There will still be a set of training and testing observations, and additionally a set of validation observations. We will search the best λ based on the evaluation of h on the validation observations. Once we have our best candidate h, we will compute the final evaluation on the testing observations.

3.2.4.2. Comparing noisy evaluations

Until now we assumed our evaluations to be reliable. They are not. We have two main problems. First, the evaluation is an approximation of the population risk on a finite set of testing observations, it is thus noisy. Second, the result of the optimization of model h with a given set of hyperparameters λ should generally be considered non-deterministic, and therefore the evaluation used to optimize the hyperparameters is noisy.

For the issue of noisy evaluation, a solution could be to partition the observations into several groups of training, validation and testing observations so that the whole process of finding the best h can be repeated several times on independent observations. This is generally unpractical because either we do not have enough observations to make several groups of reasonable sizes or the process of finding the best h is too time-consuming and could not be repeated in a reasonable amount of time.

An alternative is to use re-sampling techniques to estimate the average and standard deviation of the evaluation. The more observations we have, the more accurate this estimation will be. Cross-validation (Allen, 1974; Stone, 1974) is widely used in machine learning when available observations are not numerous enough. The recent popularity of deep neural networks and very large datasets lead to a decreased use of this method however. Our work in Chapters 5-6 demonstrates that this decrease of use is problematic.

By solving the first problem, working with subsets of data, we also solve the second one, accounting for noise in model selection. Hyperparameter optimization should be applied on average evaluations obtained with partitioned groups of observations or with re-sampling

techniques. While repeating the independent optimizations we are not only incorporating the noise coming from the sampling of the observations, but also the noise coming from the optimization process.

We now reach the end of the last operation done by the Little Scientist Computer. It has built a framework $\mathcal{H}$ (i.e. the model architecture) in which it can search for a best model h. It has searched through $\mathcal{H}$ based on the observations to find a best model h. It has evaluated its model h and then it has compared it with alternatives to select the one that generalizes best. When researchers develop new machine learning algorithms, they generally focus on either modelization, optimization or regularization. These contributions are localized but they make sense in the broader learning process. Evaluating their impact thus requires that we contextualize them, that we measure their effect on the Little Scientist Computer as a whole.

3.3. Reproducibility – Evaluating the whole procedure

Conducting research to improve machine learning algorithms does not entail producing a new h, but rather improving the procedure with which we generate h. From this point of view, the question of interest is not simply whether h is better, but rather whether the new procedure is better at producing better h. In other words, is the Little Scientist Computer improved when I change one of its components?

When machine learning researchers conduct an experiment on a given set of observation, the resulting performance evaluation is only one data point to verify the efficiency of its contributed modification to the learning algorithm. What if the same algorithm was applied on different observations from the same distribution? What if the same algorithms was applied on observations from a different distribution? What if the new optimizer was tested with a different architecture or vice-versa? What if different hyperparameters were optimized? What if a different metric was used to train the algorithm?

The more we attempt to answer these different questions the more general the conclusion on the value and usefulness of the contribution will be. To conclude this background section on machine learning, I will briefly discuss these questions and related recent works.

3.3.1. What if the same algorithm was applied on different observations from the same distribution?

There is a growing literature raising issues about the inadequacy of many common practices of machine learning researchers with respect to noise (Henderson et al., 2018; Lucic et al., 2018; Card et al., 2020). Recht et al. (2018) and Recht et al. (2019) went through the challenging task of replicating the generation of two datasets, CIFAR-10 and ImageNet in

order to measure the discrepancy of models predictions on the original examples and new examples. The model performances dropped significantly but they observed that the rankings of the model performances were relatively stable. These results are reassuring concerning the effect of data sampling, but we will see in Chapter 6 that the authors perhaps minimised the importance of ranking changes.

3.3.2. What if the same algorithms was applied on observations from a different distribution?

The seminal work of Demšar (2006) recommends statistical methods to reliably compare classification models across several datasets, that is, observations from different distributions. The method offers weak reliability when using few datasets however which is the common practice in the literature. Dror et al. (2017) proposes a promising alternative that is better suited when using few datasets.

However these are statistical methods and empirical analysis to account for noise only, not the nature of the datasets. They do not answer the question of which datasets to use. For instance, using datasets from very similar distributions would not give a lot more information than using two datasets from the same distribution. Until now, choosing datasets has been more a rule of thumb based on common practices. Measure of domain similarities being designed in theoretical work for transfer learning may be the key for objectively choosing sets of measurably diverse domains to use (Tripuraneni et al., 2020).

3.3.3. What if a different metric was used to train the algorithm?

The metric used to evaluate the models are sometimes surrogates and may influence the apparent performance of a model h. The surrogate loss called *perplexity* for instance has long been known to be an unreliable estimate of the potential of a language model (Chen et al., 1998; Hill et al., 2015; Wang et al., 2018). Another example is the different metrics used for Generative Adversarial Networks where we attempt to measure the quality of generated examples (Lucic et al., 2018). The use of multiple metrics in benchmarks (ex: SuperGLUE from Wang et al. (2019)) is important to avoid artificially biasing comparisons in favor of one model.

3.3.4. What if the new optimizer was tested with a different architecture or vice-versa?

In the recent work of Schmidt et al. (2020), dozens of different optimization techniques are applied on 8 different domains and neural network architectures. Out of the 225 experiments, no optimizer stands out as the best one. The authors state that the least they can say is that Adam (Kingma & Ba, 2015) is a reliable candidate with stable performances but it is

not the best on many benchmarks. Interestingly, the authors warn about the fact that the research field of optimization in machine learning is at the risk of being drowned by noise due to the numerous contributions with no proper evaluations accounting for the noise and the effect of hyperparameter optimization.

3.3.5. What if different hyperparameters were optimized?

Hyperparameter optimization is also a source of issues as researchers tend to use unreliable methods such as manual tuning (See survey in Appendix A). Documentation of the issue is accumulating but we have not yet seen important improvements (Kadlec et al., 2017; Melis et al., 2018; Probst et al., 2019; Schmidt et al., 2020; Sivaprasad et al., 2020).

We have seen in this chapter the different learning paradigms, the whole process of learning algorithms and what are the kinds of challenges to reliably measure contributions to machine learning researcher.

We will demonstrate some of the issues when comparing learning algorithms in Chapter 5 and provide recommendations for carrying out more reliable comparisons in Chapter 6. Before concluding the background chapters, we will next cover hyperparameter optimization in more details, together with some of the best practices.

Chapter 4

Hyperparameter Optimization

Machine learning algorithms have several hyperparameters which can significantly influence the algorithms ability to learn in some domains.

Hyperparameter optimization is a difficult task. Some hyperparameters can have a strong effect, either enabling the algorithm to reach good performances or hampering it to the point it cannot learn anything. Some other hyperparameters are tightly dependent making it difficult to optimize them manually. To make matters worse, deep neural networks are typically very expensive to train, taking several hours to several days and rendering it impractical to optimize well their hyperparameters manually.

Most importantly, hyperparameter optimization is a fundamental tool for a rigorous comparison of learning algorithms. If left uncontrolled it can be a confounding variable leading astray the researchers because of its significant impact on the achievable performance of learning algorithms.

My contribution on this topic is not to create new hyperparameter optimization algorithms but rather to build tools to democratize and encourage researchers to adopt best practices. In this chapter, I will present the different hyperparameter optimization algorithms used in my research in Chapter 5-6 or included in the hyperparameter optimization library Oríon described in Chapter 7.

4.1. Search Space

Let Λ be the set of hyperparameters of a machine learning algorithm we wish to optimize. Hyperparameters can be real numbers, integers, ordinal or categorical. For each hyperparameter we define a distribution limiting the possible values to explore during the optimization and providing a prior starting probability for possible values to try out. For instance, a learning rate may be sampled from a log-uniform distribution in the interval $(10^{-5}, 10^{-1})$, or a categorical hyperparameter such as the type of activation function may be assigned different probabilities (ex: {`'relu'`, `'sigmoid'`, `'tanh'`}, with probabilities {0.5,

0.3, 0.2}). The hyperparameters can also be *conditional* on other hyperparameters. For instance the type of optimizer to use could be a hyperparameter and the hyperparameters of these optimizer would only be *activated* when the corresponding optimizer is chosen.

In Chapter 3, we defined the empirical risk $\hat{R}_e(h, S^t)$ (Equation 3.2.18) and its optimization $\text{Opt}(S^t, \lambda)$ (Equation 3.2.19) on the training observations. Let S^{tv} be the combination of training and validation observations and $\text{sp}(S^{tv})$ be a distribution of random splits from S^{tv}. As discussed in Section 3.2.4, we would ideally compute the population risk but as we only have access to a finite set we must revert to re-sampling techniques to estimate the average performance. The best hyperparameters would thus be given by

$$\lambda^*(S^{tv}) = \arg\min_{\lambda \in \Lambda} \mathbb{E}_{(S^t, S^v) \sim \text{sp}(S^{tv})} \left[\hat{R}_e \left(\text{Opt}(S^t, \lambda), S^v \right) \right] \tag{4.1.1}$$

In practice we can only compute the empirical risk for a few train-validation sets however, leading to an approximate $\widehat{\lambda^*}(S^{tv})$

$$\widehat{\lambda^*}(S^{tv}) = \text{HOpt}(S^{tv}) \approx \lambda^*(S^{tv}) \tag{4.1.2}$$

We will see next different forms of optimization algorithm $\text{HOpt}(S^{tv})$.

4.2. Model-free Methods

The simplest methods for hyperparameter optimization are model-free. This means they are not modeling the structure of the search space to better optimize it. Either they totally ignore it as in the case of grid search and random search, or they use strategies to explore the space based on results, as in evolutionary algorithms.

4.2.1. Grid Search

Grid search is one of the most widely used algorithm for hyperparameter optimization[1]. In a grid search, the search space is discretized in a small set of possible values for each individual hyperparameter and then a Cartesian product of these values is computed. This solution can work relatively well for small search spaces of one or two hyperparameters. For larger search spaces, the number of combinations in the Cartesian product explodes, forcing the grid search to be executed on a very small number of possible values for each hyperparameter. Bergstra & Bengio (2012) raised this issue in Figure 4.1 which illustrates the inefficient coverage of the search space by grid search.

[1]See Appendix A, question 3)

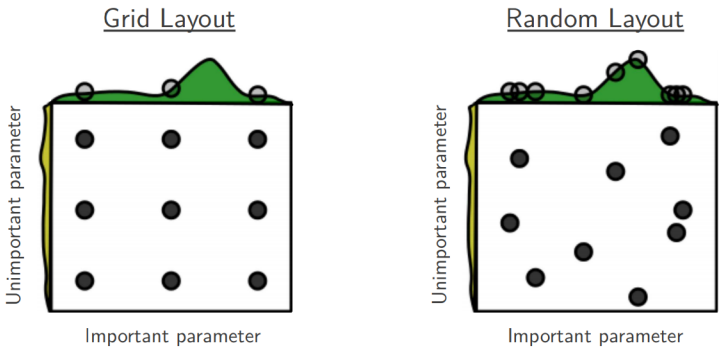


Fig. 4.1. Grid search and Random search. On each plot are 9 samples following grid layout on the left or random layout on the right. X-axis and y-axis are two different hyperparameters, one that is important, the other not. On the left and on the top of each plot is represented the average objective as a function of the corresponding hyperparameter. In this figure taken from Bergstra & Bengio (2012), the authors illustrate how grid search can be wasteful when some hyperparameters are relatively unimportant.

4.2.2. Random Search

Random search is arguably a simpler alternative than grid search. The method consists in sampling sets of hyperparameters λ based on prior distributions defined by the researcher. As illustrated by Bergstra & Bengio (2012) in Figure 4.1, random search will generally be more efficient to find best values than grid search.

Bergstra & Bengio (2012) further show that random search will find a set of hyperparameter falling in the vicinity of the optimal set λ^* with probability $1-(1-\frac{v}{V})^T$, where T is the number of hyperparameter values λ tried, v is the volume of the vicinity considered and V is the total volume of the search space. Suppose that the search space is fairly smooth and that λ in a vicinity of $\frac{v}{V} = 0.05\%$ of V would be reasonable, then you only need $T = 59$ to reach that vicinity with 95% probability.

Random search is a common baseline in hyperparameter optimization literature and is often surprisingly difficult to outcompete (Hutter et al., 2019, Sec. 1.3.1).

4.2.3. Evolutionary algorithms

Evolutionary or population-based algorithms are less commonly used in machine learning although they perform well in black-box challenges (Hutter et al., 2019, Sec. 1.3.1).

One popular method for reinforcement learning algorithm is the Population Based Training from Jaderberg et al. (2017). One of the reason for its efficiency with reinforcement learning algorithms is that it induces a schedule of hyperparameter values, adjusting them during the training, which helps stabilizing the optimization.

One of the disadvantages of all model-free methods is that they require comparatively more trials λ to explore the space. We will see now how model-based methods can alleviate this issue.

4.3. Model-based Methods

Using a surrogate model to represent the optimization landscape makes it possible to leverage the structure of the space and quickly focus optimization to interesting regions when discovered. If the landscape has large flat regions for instance, the surrogate model can identify it with few points so that the model-based method will then concentrate the exploration in depressions. Making correct assumptions for the surrogate model, for example assuming the landscape is generally smooth, will help leverage the structure of the search space if done right, otherwise it may be detrimental.

Bayesian optimization (Močkus, 1975) is a powerful model-based method for hyperparameter optimization, and for wide variety of other applications such as drug design, geological exploration and graphical design (Shahriari et al., 2015).

The general principle is to use a surrogate model to represent the whole search space and optimization landscape and to chose next points λ to evaluate based on an acquisition function which allows to balance exploration of unknown regions and exploitation of previouly visited regions of Λ that seemed promising as they had points yielding good evaluations.

Let $\{(\lambda_1, y_1), (\lambda_2, y_2), \cdots, (\lambda_t, y_t)\}$ be the set of values for λ that have been tried so far at step t with their corresponding evaluations $y = f(\lambda)$. And let $y_{\min}$ be the smallest y observed so far. A common acquisition function for Bayesian optimization is the expected improvement.

$$\mathrm{EI}(\lambda) = \mathbb{E}_{\xi}\left[\max(y_{\min} - f(\lambda), 0)\right] \tag{4.3.1}$$

Where ξ represents sources of noise in $f(\lambda)$. Assuming y to follow a normal distribution given λ, we can use a Gaussian process[2] as a surrogate model and we can get a closed-form solution to compute this expectation, yielding

$$\mathrm{EI}(\lambda) = (y_{\min} - \mu_\lambda)\Phi\left(\frac{y_{\min} - \mu_\lambda}{\sigma_\lambda}\right) + \sigma_\lambda \phi\left(\frac{y_{\min} - \mu_\lambda}{\sigma_\lambda}\right) \tag{4.3.2}$$

Where μ_λ is the average prediction of our surrogate model for λ and σ_λ is the uncertainty of the surrogate model for λ. For a Gaussian process, this is given by Equations 3.2.11 and 3.2.12.

Other acquisition functions exist such as Thompson sampling, upper confidence bound and entropy search (Shahriari et al., 2015). Figure 4.2 illustrates the sequence of steps during

[2]Refer back to Section 3.2.1.2 for more information on Gaussian processes.

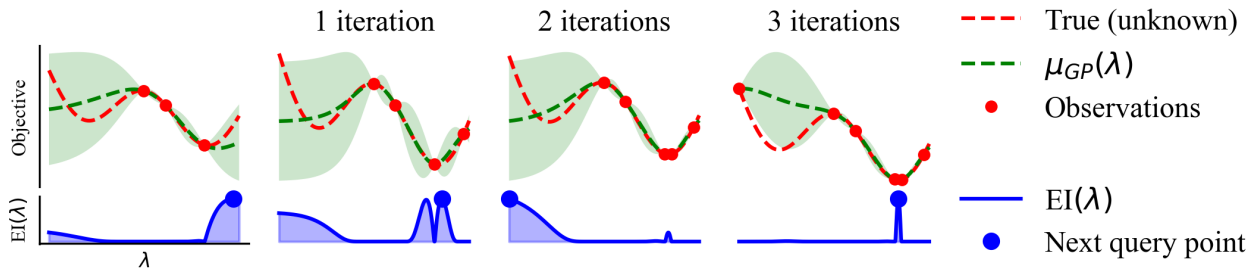


Fig. 4.2. Bayesian optimization on a toy function. In each column, in the top row the average estimation and uncertainty of the Gaussian process is represented with green curves and areas, red points are the observations and the red curve is the true toy function. The x-axis is the hyperparameter λ (input of the toy function), and the y-axis is the objective (output of the toy function). In the bottom row, the experiment improvement is represented on the y-axis (EI(λ)) with the blue curve with the selected next hyperparameter to try for next iteration as a blue point. The first column is the first Bayesian optimization step after sampling 3 random points. Next columns are the subsequent iterations. The toy function is being minimised. As more observations are being made, the uncertainty of the Gaussian process decreases. At the third iteration, the assumptions on the smoothness of the function leads the Gaussian process to predict that no likely functions would give lower objectives in unexplored regions than the current minimum found – the green area on the left is higher than the minimum found.

Bayesian optimization, finding the optimal λ, computing $f(\lambda)$ and updating the Gaussian process, repeating the tree steps multiple times.

Gaussian processes work particularly well in search spaces of small dimensionality, which is the use case of most machine learning researchers. They cannot be applied directly to non-real hyperparameters, but solutions exist to handle them. A naive solution is to work on real-valued dimensions for integers, or a one-hot encoding of categorical dimensions, and to simply round the suggested values of the Bayesian optimization algorithm. Garrido-Merchán & Hernández-Lobato (2020) proposed a modified kernel which improves the optimization of such types of hyperparameters.

Another issue of Gaussian processes for Bayesian optimization is that they do not scale well with many observations, scaling with $O(t^3)$ in computational complexity. Nevertheless, the scaling issue of Gaussian processes is minor when optimizing of hyperparameters of deep neural networks which take several hours to train. Using a random forest instead of a Gaussian process can improve Bayesian optimization if there are non-real hyperparameters or if the computational complexity is problematic (Hutter et al., 2011).

This brings us to the major issue with model-based optimization in general; It does not parallelize well. The computational complexity may not be problematic when working with deep neural networks, but the sequential nature of model-based methods is generally a drawback. Suppose evaluating λ takes one day, and that to achieve a good performance requires about 50 trials with a different λ_t. To execute Bayesian optimization, we would

first randomly sample a batch of trials, lets say 20 for the sake of the example, that we can run in parallel in a single day. This means the optimization would take about 31 days, a full month. Hyperparameter optimization algorithms must be parallelizable to be practical when working with deep neural networks.

One simple solution to parallelize Bayesian optimization is the constant liar (Chevalier & Ginsbourger, 2012). The suggested λ that were not yet evaluated are assigned a constant value, which can be $y_{\min}$, $y_{\max}$ or mean(y), and a new set λ is sampled accordingly. Another solution is to sample λ using Markov Chain Monte-Carlo instead of a closed form solution (Snoek et al., 2012).

Finally, Bergstra et al. (2011) proposed a very different approach to Bayesian optimization that can handle any type of hyperparameter and that parallelizes well; the Tree-Structured Parzen Estimator (TPE). Instead of modeling directly $p(y \mid \lambda)$, TPE uses Parzen-Rosenblatt estimators to model $\ell(\lambda) = p(\lambda \mid y < y^*)$ and $g(\lambda) = p(\lambda \mid y \geq y^*)$ where y^* is the γ-th quantile in observations $\{(\lambda_1, y_1), (\lambda_2, y_2), \cdots, (\lambda_t, y_t)\}$. This provides a density estimation of $\ell(\lambda)$ for *good regions* and $g(\lambda)$ for *bad regions*. The expected improvement acquisition function can be rewritten using $\ell(\lambda)$ and $g(\lambda)$, leading to a simple formulation that can easily be optimized.

$$\left(\gamma + \frac{g(\lambda)}{\ell(\lambda)}(1-\gamma)\right)^{-1} \tag{4.3.3}$$

The Parzen-Rosenblatt estimator $\ell(\lambda)$ is used to sample candidates and selecting among these candidates the one with lowest $\frac{g(\lambda)}{\ell(\lambda)}$. This sampling procedure enables TPE to parallelize seamlessly compared to other Bayesian optimization algorithms. The use of Parzen-Rosenblatt estimators adapted to different types of hyperparameters (real, integer, categorical) also makes it very flexible.

Model-based methods presented so far all relied on y to model the optimization landscape. They treat the evaluation process as a black-box, λ goes in, y goes out. During the training of deep neural networks there are however many statistics that can be used to guide the optimization of hyperparameters. We will open the black-box in the next section to leverage this information.

4.4. Multi-fidelity methods

When experimenting, researchers usually monitor the training progress of models to see if they are promising. This allows them to save a significant amount of time by stopping unpromising models or tweaking hyperparameters in a short feed-back loop. Hyperparameter optimization algorithms can also be sped-up significantly by using this information.

Hyperband (Li et al., 2018a), an extension of successive halving (Karnin et al., 2013), is perhaps the most used multi-fidelity algorithm due to its simplicity and efficiency. It is

based on successive halving of batch of trials to leverage low-budget evaluations of λ (i.e. small subset of the observations or training for a few epochs) to select candidates that are worth evaluating with larger budgets.

Given a training budget T (ex: total number of training iterations) and a minimal number of trials n, successive halving will sample n sets of hyperparameters λ and evaluate them with a budget of $\frac{T}{n}$ training iterations. It will then select the 50% hyperparameters λ that yielded the best evaluation and evaluate them with a budget of $\frac{2^1 T}{n}$. The algorithm repeats the cycle until reaching the budget $\frac{2^i T}{n} \geq T$ with a single trial, where i is the number of times the cycle was repeated.

One problem of this procedure is that it is sensitive to the choice of n. Dividing the budget across too many trials may lead to fidelities (budget $\frac{T}{n}$) that are too low and unreliable. For instance if the budget is the number of training epochs (the number of training passes over the full dataset), training for too few epochs may not give reliable evaluations of the hyperparameters. On the other hand, giving too large budgets at the beginning may be wasting resources on configurations of hyperparameters that are clearly non-promising.

Hyperband addresses this issue by spreading the budgets across multiple *brackets* to increase the chances that at least one of the successive halvings was executed using a good initial number of trials n. Table 1 gives an illustration of this distribution of budget across *brackets*.

Hyperband parallelizes particularly well compared to model-based algorithms, but it requires waiting for all evaluations to complete before executing the successive halving, causing serious bottlenecks. Li et al. (2020) proposed an asynchronous alternative to Hyperband, Asynchronous Successive Halving Algorithm (ASHA), that alleviates this issue by incrementally building the rungs (iterations of successive halving) based on available results. Another improvement to Hyperband is the work of Falkner et al. (2018) in which they replaced the

	bracket 4		bracket 3		bracket 2		bracket 1		bracket 0	
rung	# λ	budget	# λ	budget	# λ	budget	# λ	budget	# λ	budget
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

Table 1. Distribution of resources with Hyperband. The resources are allocated into 5 different brackets, each starting with a different minimal budget, from 1 to 81. The number of trials (# λ) is adjusted based on the minimal budgets and reduction factor (for successive halving) so that the total budget of each bracket is the same (405). The rungs are the steps of successive halving from the minimal to the maximum budgets in each bracket.

initial random search by a Bayesian optimization algorithm similar to TPE, thus combining the strengths of both Hyperband and Bayesian optimization.

The efficiency of low-fidelity algorithms strongly rely on a simple assumption; cheap noisy evaluations of λ can help us find the best λ when using a full budget. Unsurprisingly, best candidates with cheap noisy evaluations are generally not the best candidates at full budget. If it was the case, we could simply optimize the hyperparameters in a low budget regime, pick the best ones and evaluate with full budget right away. Still, these algorithms can perform well when this assumption is mildly wrong. However, in cases where the training is highly unstable, as is the case with generative adversarial networks or reinforcement learning, multi-fidelity algorithms can actually perform worse than random search.

There is thus no hyperparameter optimization algorithm that performs best overall. Researchers must be careful when choosing an algorithm, taking into consideration the size of the search space, the type of hyperparameters, the total computational budget and whether multi-fidelity can be reliable in their specific use-case.

4.5. Reproducibility

We can look at reproducibility from two different perspectives with respect to hyperparameter optimization. The first one is the reproducibility of the research works on hyperparameter optimization itself, and the second one is the role played by hyperparameter optimization in reproducibility of general machine learning research.

4.5.1. Reproducible hyperparameter optimization research

In 2019, Liam Li shared concerning results about reproducibility issues at the AutoML workshop[3]. Based on his results, the reported performances of several neural architecture search algorithms were unreproducible and in some case these were even out-performed by random search. Fortunately, benchmarks for neural architecture search were proposed at the same conference by Ying et al. (2019) later followed by a guideline of best practices (Lindauer & Hutter, 2020).

The creation of reliable benchmarks for hyperparameter optimization or neural architecture search is a remarkable enterprise. Hyperparameter optimization is the most expensive part of the whole learning pipeline as it requires training the learning algorithms multiple times. Thus, benchmarking hyperparameter optimization algorithms requires executing multiple times hyperparameter optimization, literally leading to an explosion of the computational cost of the whole benchmarking process.

One way to overcome this is to create datasets of pre-evaluated sets of hyperparameters (Klein & Hutter, 2019). This requires however discretizing real-valued hyperparameters

[3]Later published at the Conference on Uncertainty in Artificial Intelligence (Li & Talwalkar, 2020)

which will hamper the performance of some hyperparameter optimization algorithms, thus degrading the representativity of the benchmark for real-world problems. Ying et al. (2019) and Arango et al. (2021) pushed this solution very far by evaluating 5 and 6.4 million sets of hyperparameters[4]. Another solution is to train surrogate models which will represent the optimization landscape and serve as a simulated environment to benchmark hyperparameter optimization algorithms. This is what Eggensperger et al. (2015) proposed using diverse regression models and later Klein & Hutter (2019) using a multi-task Bayesian neural network (NIP) to simulate different search spaces.

The latter solution is promising, not only for hyperparameter optimization benchmarks, but for benchmarks in machine learning in general as we will see next.

4.5.2. Hyperparameter optimization for reproducible results

The second perspective to look at reproducibility with respect to hyperparameter optimization is the role it plays for reproducibility in general machine learning research. As discussed in Section 3.2.4, hyperparameter optimization is a fundamental part of model selection to benchmark machine learning algorithms.

Manual tuning of the hyperparameters is by far the least reproducible of all methods. This alone is enough to severely hamper the reproducibility of findings in machine learning. The common solution based on the idea of open-science is to share the hyperparameter values found so that another researcher can obtain the same results. This communication of hyperparameters does not solve the methodological issues as raised by Kadlec et al. (2017) and Melis et al. (2018) however, and rather perpetuates them. Suppose I optimize the hyperparameters of algorithms A and B with a budget of T and T' trials respectively, with $T' \ll T$, and then compare the resulting performance of A and B and find that A outperforms B . The outcome of this comparison is misleading: the reason A outperforms B could simply be that I better optimised its hyperparameters. Yet, if I shared these hyperparameters with other researchers, they would obtain the same result and perpetuate the methodological error. Machine learning algorithms should be compared on common grounds.

Unfortunately, sharing hyperparameters when using a proper methodology, that is, equally optimizing the hyperparameters of A and B, is also problematic. Something we did not cover yet is the stochastic nature of hyperparameter optimization itself, irrespective of the evaluation function. Obvious examples of stochastic methods are random search or any other method based on it, such as Hyperband. Even Bayesian optimization is noisy partly due to the initial stage during which random sets of hyperparameters are sampled. Let us continue with the example from last paragraph. I have learning algorithms A and B, and optimize their hyperparameters with the same budget T. A turns out to be better than

[4]For the sake of precision, the hyperparameters were specifically architecture configurations in the case of Ying et al. (2019)

B. Is it because A is actually better than B? Or is it because out of good luck I explored better regions of the hyperparameter search space for A than for B? I cannot answer this question without repeating this hyperparameter optimization multiple times. But as we discussed in last section, running hyperparameter optimization multiple times is prohibitively expensive.

This is why modeling the optimization landscape as proposed by Eggensperger et al. (2015) and Klein & Hutter (2019) could be useful. By doing so we could have estimates of the variability of the performances for slightly different hyperparameters, this way answering the question without having to execute many hyperparameter optimization. Ensuring the reliability of the surrogate model remains a critical problem as of now, and would require theoretical guarantees if we were to follow this route.

This concludes the background chapters of this thesis. The next three chapters will cover the issue of reproducibility of machine learning algorithms we just discussed (Chapter 5), analyse the sources of variations in these benchmarks to provide recommendations (Chapter 6) and present a framework for hyperparameter optimization developed during this thesis (Chapter 7).

Chapter 5

Article: Unreproducible Research is Reproducible

The article reproduced below was published as:

Xavier Bouthillier, César Laurent and Pascal Vincent "Unreproducible research is reproducible." In *International Conference on Machine Learning*, pp. 725-734. PMLR, 2019.

Context

This project stemmed following a discussion with Liam Li during NeurIPS 2018. During this conference I had been the spectator of many discussions about reproducibility which invariably revolved around open science, code sharing and dataset sharing. During this meeting with Liam Li, we discussed how these solutions were insufficient to solve the reproducibility issues faced by our community.

Recent works (Melis et al., 2018; Henderson et al., 2018; Lucic et al., 2018) had already shown how reproducibility issues could be due to improper methodology either due to unfair comparisons of algorithms or lack of accounting for noise in results. However, these results could be ignored by researchers in other fields of machine learning on the basis of the disputable metrics of the domains covered by these works or their particular instability. For instance, Melis et al. (2018) compared language models on the basis of the perplexity metric, a surrogate metric that does not strongly correlate with model performances on end-tasks (Iyer et al., 1997; Ito et al., 1999; Dudy & Bedrick, 2020; Hu et al., 2020) and is recommended to be used as a sanity-check only (Jurafsky & Martin, 2018). Lucic et al. (2018) investigated model performance comparisons between Generative Adversarial Networks by estimating the quality of generated images using the Inception Score (Salimans et al., 2016) and the Fréchet Inception Distance (Heusel et al., 2017), metrics that cannot detect overfitting according to Lucic et al. (2018). Henderson et al. (2018) demonstrated the instability of performance estimations in reinforcement learning, however this level of instability is rarely found outside

reinforcement learning. Despite all these concerning results, researchers from other sub-fields of machine learning could blame these domain-specific problems (disputable metrics, unstable environments) for the reproducibility issues and refuse changing their research practices.

I decided to take one of the most canonical tasks in machine learning, image classification on standardized datasets, to verify whether these methodological issues are general to most machine learning domains. Liam Li concurrently followed a similar path in the domain of Neural Architecture Search (Li & Talwalkar, 2020).

Contributions

This work highlights that even in fully controlled environments, improper hyperparameter optimization across baselines and lack of accounting for variance leads to unreproducible results. It further shows that even when using proper hyperparameter optimization, rankings of algorithms would not be reproducible if they do not account for variance.

Authors contributions

Xavier Bouthillier:

- Main idea.
- Implementation and execution of the experiments.
- Writing paper.

César Laurent:

- Writing code for figures.
- Significant help for writing the paper.

Pascal Vincent:

- Discussing the ideas and improving the clarity of the message and text.

Abstract. The apparent contradiction in the title is a wordplay on the different meanings attributed to the word *reproducible* across different scientific fields. What we imply is that unreproducible *findings* can be built upon reproducible *methods*. Without denying the importance of facilitating the reproduction of *methods*, we deem important to reassert that reproduction of *findings* is a fundamental step of the scientific inquiry. We argue that the commendable quest towards easy deterministic reproducibility of methods and numerical results should not have us forget the even more important necessity of ensuring the reproducibility of empirical findings and conclusions by properly accounting for essential sources of variations. We provide experiments to exemplify the brittleness of current common practice in the evaluation of models in the field of deep learning, showing that even if the results could be reproduced, a slightly different experiment would not support the findings. We hope to help clarify the distinction between *exploratory* and *empirical* research in the field of deep learning and believe more energy should be devoted to proper empirical research in our community. This work is an attempt to promote the use of more rigorous and diversified methodologies. It is not an attempt to impose a new methodology and it is not a critique on the nature of exploratory research.

Keywords: Reproducibility

5.1. Introduction

Reproducibility has been the center of heated debates in many scientific disciplines. Psychology in particular has been the focus of several large reproduction efforts, attempting to reproduce close to a hundred studies (Collaboration et al., 2015; Klein et al., 2018). These were motivated by past evidence of lack of scientific rigour, researcher biases, and fraud (Eisner, 2018).

To help counter these problems, important changes were enacted in the psychology research community in the past few years. Making data available is becoming more common, journals are publishing replication reports and preregistration of research specifications is a growing practice.

We see a similar recent trend in machine-learning: the topic of reproducibility rose to prominence at top conferences (Henderson et al., 2018), and several workshops are now focusing on that matter. Top conferences have adopted recommendations for code sharing. More tools are made available to simplify the replication of experiments reported in papers, building on new technologies such as shareable notebooks (Kluyver et al., 2016; Forde et al., 2018), containerization of operation systems, such as Docker (Merkel, 2014) and Singularity (Kurtzer et al., 2017), and open-sourcing of frameworks such as Theano (Theano Development Team, 2016), PyTorch (Paszke et al., 2017) and TensorFlow (Abadi & al., 2015).

While the type of reproduciblity enabled by these tools is a valuable first step, there has been comparatively much fewer discussion about the reproducibility of the *findings* of studies.

Three recent works (Melis et al., 2018; Henderson et al., 2018; Lucic et al., 2018) have shown that proper experimental design is capital to assert the relative performances of models. Beyond mere reproduction, these works shed light on the fundamental problem of reproducibility that cannot be addressed solely by sharing resources such as code, data and containers. The experimental design is at the core of the concept of *reproducibility of findings*.

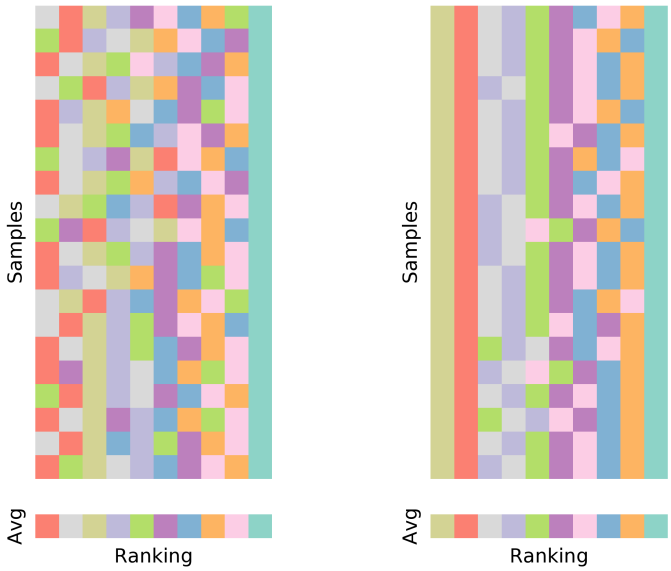
Melis et al. (2018) conducted large scale experiments in Natural Language Processing with hyper-parameter optimization procedures to compare models in an unbiased benchmark, leading to the surprising result that vanilla LSTM may be as good as recent supposedly state-of-the-art models. Lucic et al. (2018) analyzed GAN models with various experimental setups including average analysis over different initialization of models, concluding that current evaluation methods of GANs can hardly discriminate between model performances. Henderson et al. (2018) exposed the problem of high instability of results in reinforcement learning. They executed several trials over different seeds and concluded that results in reinforcement learning should include enough trials over different initialization of the model and environment to support a claim with statistical significance.

We extend on these prior works by analyzing a task which played an essential role in the development of deep learning: image classification. Its simple undisputed evaluation metric, in contrast to NLP (Melis et al., 2018) and GAN metrics (Lucic et al., 2018), guarantees that any inconsistency in results cannot be blamed on the brittleness of the evaluation metric, but only on the methodology itself. Additionally, the environment is strongly controlled, in contrast to RL (Henderson et al., 2018), making the inconsistency of results due to small controlled sources of variations even more striking.

We propose to revisit the empirical methodology behind most research papers in machine learning, *model comparisons*, from the perspective of reproducibility of methods and findings. We will first give an example to outline the problem of reproduciblity of methods and findings in section 5.2. We will then clarify the definition of reproducibility in section 5.3. In section 5.4 we will describe the design of the experiments, modeled on current practices in the field, in order to verify how easy false-positive conclusions can be generated. In section 5.5 we will present and analyse the results and discuss their implications, before highlighting some limitations of the current study in section 5.6. We will conclude with an open discussion on the differences between exploratory and empirical research in section 5.7, explaining why all forms of reproducibility deserve the attention of the community.

5.2. A problem scenario in a typical deep learning experimentation

Suppose we choose several model architectures that we want to compare for the task of image classification. We train all of them on a given dataset and then compare their



(a) (b)

Fig. 5.1. Variation in the ranking of 8 different neural network architectures (models) across multiple trials (samples). 5.1a: on MNIST digit classification; 5.1b: on CIFAR100 image classification. The eight different model architecture types are shown in different colors. Each row is a trial with a different random initialization of the models, and in each row the models are ranked from best (leftmost) to worst test accuracy. In 5.1a we see that ranking can vary greatly from one trial to another, while for a different dataset (5.1b) rankings of the same set of models can be more stable. We cannot know this however unless we train multiple times the same model. It is thus crucial to do so to ensure the robustness of the conclusions we draw based on a ranking of models.

classification accuracy on the same held-out test set. We then rank the models according to this measured evaluation metric and conclude that the one with highest accuracy is the best one on this dataset. Later on, we retrain the same models on the same dataset but obtain different numerical results, and observe that the new best model is different than in the previous experiment. How come? It turns out we forgot to seed the random number generator used to initialize the models to have reproducible results.

The usually recommended fix to this reproducibility problem is to set the seed of the random number generator to some arbitrary value, and forget about it. But why are the performances of models sensitive to it? Measurements are affected by sources of variations. The measured accuracy of a model is, for instance, affected by its initialization, the order of the data presented during training and which particular finite data sample is used for training and testing, to name but a few. Trying to fix this problem by seeding a random number generator can inadvertently limit the conclusions to this specific seed. Therefore,

simply fixing one of these sources of variations has the effect of limiting the generality of a conclusion.

We show in Figure 5.1a an example of different runs using different seeds, keeping everything else fixed, which lead to different conclusions as to the ranking of eight different types of model architectures on the MNIST dataset. We can clearly see that any conclusion based on a single trial would very likely be invalidated by other runs. It may be different for other datasets, where we could observe a behavior as shown in Figure 5.1b. However we cannot know this unless we re-run the experiment under different values of the source of variation.

What we would like to point out here, is that there are two forms of reproducibility that can interfere if we are not cautious. The reproduction of the *results* requires the conversion of a stochastic system into a deterministic one, e.g. the seeding process. While this helps reproduction of results, avoiding this source of variation altogether in experiments has the potential effect of dramatically weakening the generality of conclusions. This is at odds with the reproduction of *findings*.

5.3. Reproducibility: a confused terminology

The distinction between different types of reproducibility is not a new phenomenon (Barba, 2018), however there is no standard terminology to this day.

In this work we will use the terms proposed by Goodman et al. (2016), which avoid the ambiguity of the terms *reproducibility*, *replicability* and *repeatability*. We report here the definitions adapted to the context of computational sciences:

Methods Reproducibility: A *method* is reproducible if reusing the original code leads to the same results.

Results Reproducibility: A *result* is reproducible if a re-implementation of the method generates statistically similar values.

Inferential Reproducibility: A *finding* or a *conclusion* is reproducible if one can draw it from a different experimental setup.

In machine learning, methods reproducibility can be achieved by seeding stochastic processes, but this is insufficient to ensure results reproducibility, where one cannot e.g. rely on having the exact same implementation, execution order, and hardware. To assess results reproducibility some characterization of the probability distribution over what is measured (such as evaluation metrics) is needed. However confidence intervals are seldom provided in the deep learning literature, thus *results reproducibility* can hardly be achieved at the moment, unfortunately. Note that *methods reproducibility* can be obtained as well by producing confidence intervals instead of documenting seeds. The distinction between methods and results reproducibility lies in the presence of a step of reimplementation or reconstruction of the experimental setup.

At the other end of the reproducibility spectrum is *inferential reproducibility*, which is not about the (numerical) results, but rather the conclusions drawn. Suppose a technique performs better than the state-of-the-art for a given task on several vision datasets and fulfills results reproducibility. The authors may conclude that the technique improves the performance on that task. However, if the method later fails on another similar vision dataset, it would invalidate inferential reproducibility. The conclusion, as stated, is not reproducible. This would imply that the assumptions behind the conclusion were wrong or too vaguely stated if at all, and need to be refined: maybe the model performs better on smaller datasets, or on some particular types of images. Such refinements are critical for the advancement of science and can lead to new discoveries.

An observation we want to convey to the reader is that a major part of the current reproducibility litterature in computational science is strongly influenced by the seminal work of Claerbout & Karrenbach (1992), a work that was solely about methods reproducibility, proposing a methodology to ensure automatic regeneration of a report with its accompanying figures. Likewise, the machine learning community seems to be currently mostly referring to methods reproducibility when discussing about *reproducibility*, with the common solution proposed being code sharing.

While code sharing is a valuable practice for the community we argue that it only addresses methods reproducibility and results reproducibility at best. We will present in the next section our methodology to test how current common practice for analyzing model performance in deep learning fails to ensure inferential reproducibility.

5.4. Methodology to test the robustness of conclusions

The goal of this work is to verify the effect of sources of variations on the robustness of the conclusions drawn in the context of image classification with deep learning models, using common methodology.

To verify this, we will train several popular deep-learning models (i.e. network architectures) multiple times without fixing the initialization or the sampling order of the data and we will measure how much the ranking of the models vary due to these sources of variations.

5.4.1. Biased vs unbiased scenarios

In order to draw a faithful portrait of the current methodology of practitioners in the field, we would need to use what original authors deemed the best hyper-parameters of each model on each dataset. Unfortunately, the dataset/model matrix we might gather from the literature in this way would be too sparse, leaving us with very few datasets where we could hope to compare all (or even most) models. We will instead consider two methodologies

which are respectively worse and arguably better than most common practice. By doing so, we bound the spectrum of experimental bias that includes common practices.

The worse than common practice approach consists in selecting the optimizer hyper-parameters that are the best for one specific model on one specific dataset and apply them unchanged to all other (model,dataset) pairs. This is the most biased methodology, as it should favor the model that was used to select these hyper-parameters. This is arguably a worse practice than what we would (hopefully) observe in the field, but a reasonable lower bound of it as long as all models can be trained sufficiently well. We will refer to this as the *biased scenario*.

The better practice is to optimize the hyper-parameters for each model on each dataset independently using an appropriate validation set, while ensuring that all models had an equal budget of hyper-parameter optimization. We will call this the *unbiased scenario*.

Considered hyper-parameters include the learning rate and momentum as well as weight-decay (L2 regularization strength).

5.4.2. Experimental setup

For the benchmarking of models, we chose 10 different models of different scales: LeNet (LeCun et al., 1998), MobileNetV2 (Sandler et al., 2018), VGG11, VGG19 (Simonyan & Zisserman, 2014), ResNet18, ResNet101, PreActResNet18, PreActResNet101 (He et al., 2016b), DenseNet121 and DenseNet201 (Huang et al., 2017). We limit ourselves to common models in the field for image classification tasks. The evaluation metric of interest is the classification accuracy on the test set.

By *model* we refer to a given architecture (e.g. VGG11) i.e. a specific parameterized function form, together with its standard recommended random parameter initialization strategy. A specific set of (trained) parameter values for a given model corresponds to an *instantiation* of the model. What we are after is a qualitative estimation of which model architecture (together with its standard training procedure) performs better, not which instance. In practice one may care more about which model instance performs best, as it is the instance that is used in the end. However, in science, model architecture are the center of interest. An instance is useful as a probe to better understand a model architecture. This is why sources of variations such as the initialization should not be fixed. Conclusions on an architecture that are limited to a single instance are very weak.

5.4.2.1. Seed replicates

For each model, we sample 10 different seeds for the pseudo-random number generator used for both the initialization of the model parameters and the ordering of the data presented

by the data iterator. All models are trained for 120 epochs on the same dataset. Hyper-parameters will be selected differently in the biased and unbiased scenarios, in a way which we will explain shortly.

Following the terminology of Vaux et al. (2012), we call these runs *seed replicates*.

5.4.2.2. Dataset Replicates

Observations are likely to differ depending on the difficulty of the task, as the potential of different models will be easier to distinguish on more challenging tasks. To ensure some robustness of our conclusions to this source of variation, we will run the seed replicates on different datasets, namely MNIST (LeCun et al., 1998), SVHN (Netzer et al., 2011), CIFAR10, CIFAR100 (Krizhevsky et al., 2009), EMNIST-balanced (Cohen et al., 2017) and TinyImageNet (et al, 2019). We will call the set of seed replicates of a model on a given dataset a *dataset replicate*. We will not consider here other (less extreme) potential sources of variation in the dataset, but briefly discuss them in section 5.6.

5.4.2.3. Biased and unbiased seed replicates

As explained in subsection 5.4.2.1, the observations about the variations of performances of the models will be different in the biased and unbiased scenario.

For the biased scenario, we will pick the specific hyper-parameters provided by He et al. (2016b) in their work on ResNets. This choice should favor ResNet models in our benchmark.

For the unbiased scenario, we will optimize the hyper-parameters for each dataset replicate. To distinguish each scenario, we will call them biased and unbiased seed replicates (5.4.2.1), biased and unbiased dataset replicates (5.4.2.2).

Example to summarize the terminology: the *unbiased-scenario dataset replicate* for dataset MNIST and a given model, will be constituted of 10 *seed replicates*, each of which is a trained model instance (that was initialized with on of the 10 seeds) whose hyper-parameters were selected for best performance on the validation subset of that dataset.

The hyperparameter optimization will be executed using a slightly modified version of ASHA (Li et al., 2020)[1]. The exploration is executed until 10 different runs, each with a budget of 120 epochs, have been trained for a given pair of model and dataset. Once this threshold is reached, best hyper-parameters found are used to follow the same procedure as for the seed replicates, i.e. training the model 10 times with different initialization seeds. The hyper-parameter optimization is done based on error rate of the models on the validation set. For the analysis, we will use the test accuracy measures, as we do for the biased seed replicates. This set of 10 runs for each model are the unbiased seed replicates.

[1]With budgets of 15, 30, 60 and 120 epochs and a reduction factor of 4

5.5. Experimental results

Results are presented in two different forms. The first goal is to visualize the distribution of performance across seed replicates (5.5.1). The second goal is to visualize the stability of the model rankings when selecting single seed replicates to measure their performance (5.5.2). The variances of the model rankings are a way of measuring the likelihood that a conclusion drawn from a single seed replicate, which is common practice in the deep learning community, would hold across many replicates.

5.5.1. Performance distributions over seed replicates

We generated histograms for the seed replicates for different models on each dataset to compare the distribution of their test error rate. Figures 5.2a and 5.2b present these histograms for the biased and unbiased scenarios, respectively. Datasets are ordered based on their difficulty (measured by the average performance of all models). These plots help visualize the overlaps between the distributions of the model performances and give insight on the complexity of the different tasks.

We observe that the overlaps in distribution do not significantly increase between the unbiased and biased scenario. Since they are bounding the spectrum of common practices, we can safely assume that the current observations would also hold in a faithful simulation of common practices.

One can see that concluding which model performs best based on observations from a single initialization seed is brittle: this conclusion will often be falsified if using a different seed. This is especially true for simpler datasets (mnist, svhn, emnist), but one also sees that model ranking varies widely across datasets. Thus, results from single seed experiments on too few datasets, even if they satisfy methods reproducibility, are not sufficient to ensure inferential reproducibility. Hence our irreverent title.

5.5.2. Ranking stability over seed replicates

We then perform basic bootstrap sampling (Efron, 1992) using the seed replicates. For each dataset, we randomly select a seed replicate for each model and rank them accordingly. We do so 1000 times, and report the results as histograms of rankings aggregated over all datasets. Figures 5.3a and 5.3b contain those histograms for the biased and unbiased scenarios, respectively. Such ranking distributions makes it possible to compare model performances across several datasets.

We first note that PreActResNet models do not stand out as the best performing models in the biased scenario, although the hyper-parameters were supposed to favor them. Looking back at Figure 5.2a, we can observe that they did not outperform other models even on

CIFAR10, the dataset on which the best hyper-parameters were selected according to the literature, although they did outperform ResNets, which was the claim of He et al. (2016b).

The aggregated results of Figure 5.3b tend to confirm the superiority of PreActResNets over ResNets. The superiority is however more subtle than what is shown in the original paper, with ResNets sometimes performing better (CIFAR10) or on par (CIFAR100, Tiny-ImageNet). We must note nevertheless that the models used in He et al. (2016b) were considerably deeper (110, 164 and 1001 layers) than the one used in this study (18 and 101 layers), making it impossible to compare directly our results to the original ones.

This brings us to another important observation: In our study larger ResNets and PreActResNets did not outperform their smaller counterparts, raising a doubt that larger models would here fare differently. This could be due in part to the fact that we did not perform data augmentation. Nevertheless, the same cannot be said for VGG, for which the larger model is systematically better than its smaller counterpart.

Given the relative homogeneity of the aggregated results, a more subtle measure, one for instance where we weigh performance with respect to computational complexity, would likely raise small models to prominence. We believe that a more nuanced portrait of model performances as the one presented in this study would promote such finer grained analysis.

5.6. Limitations of this work

5.6.1. Problem diversity

All experiments are confined to the problem of image classification. It is reasonable however to expect that similar observations can be made for different family of problems provided that best performing models have overlapping distribution of performances. Note that similar observations were made on the more complex tasks in NLP (Melis et al., 2018) and for GANs (Lucic et al., 2018). Our empirical contribution here is to assess the situation on what is arguably the most studied standard task for deep learning, which has a simple undisputed evaluation metric, i.e. image classification.

5.6.2. Hyper-parameter optimization challenges

Hyper-parameter optimization is not a simple task and although it can help to reduce the bias in the way hyper-parameters are chosen it might also introduce another bias for models that are easier to hyper-optimize.

It is also difficult to determine which hyper-parameters should be tuned as there are several factors that influence the training of a model. When training all models with the same optimizer for instance, even though we tune the corresponding hyper-parameters for all models, some of the models may be favored by this choice of optimizer over another. A

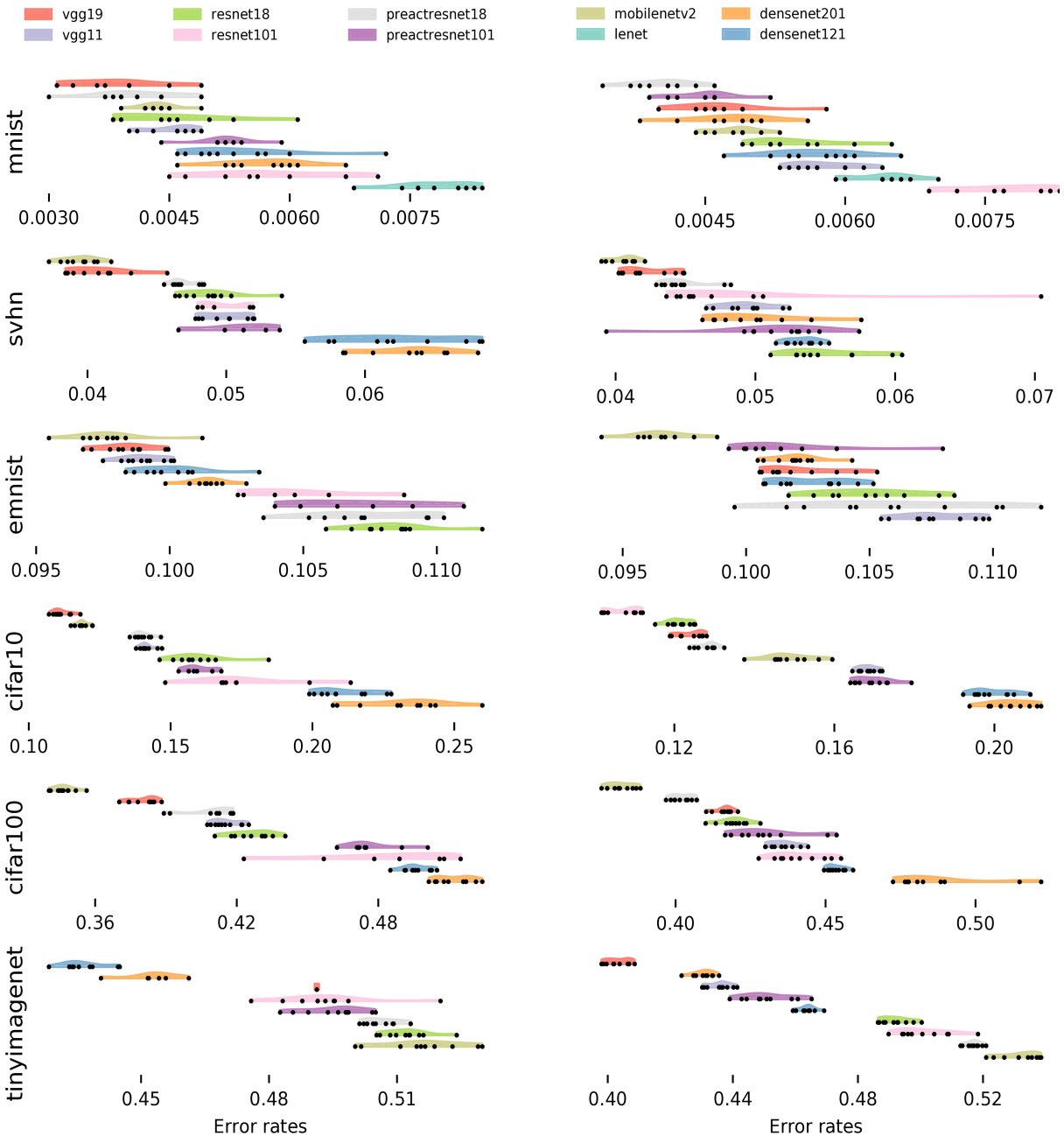
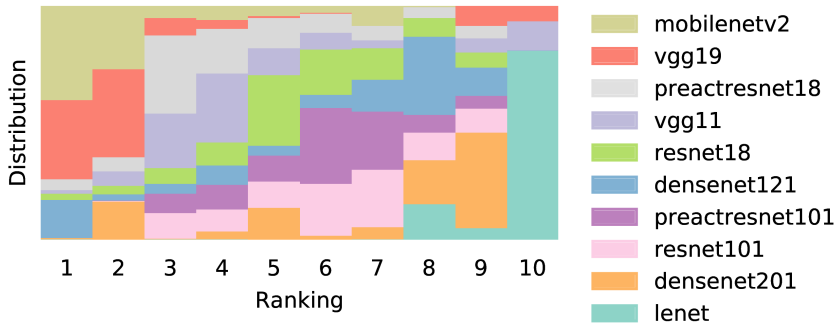
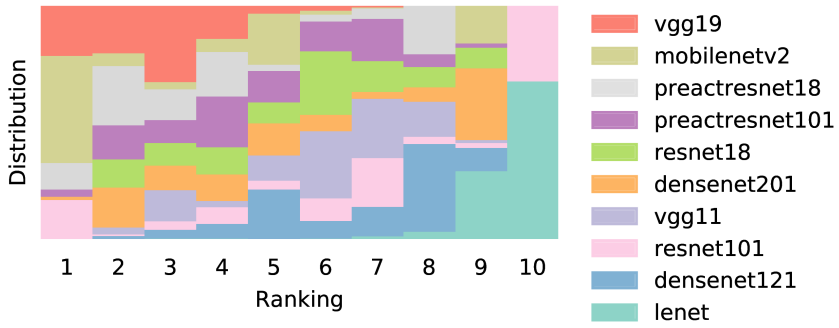


Fig. 5.2. Histograms of performances for each model when changing seeds in the biased (a) and unbiased (b) scenario. Each model is identified by a color. For each dataset, models are ordered based on their average performance. Outliers are omitted for clarity. They consist of models that would not train with the shared hyperparameters values, namely VGG19 on tinyimagenet in the biased scenario. One can see that concluding which model performs best based on observations from a single initialization seed is brittle: this conclusion will often be falsified if using a different seed. This is especially true for simpler datasets (top three), but one also sees that model ranking varies widely across datasets. Thus results from single seed experiments on too few datasets, even if they satisfy methods reproducibility, are not sufficient to ensure inferential reproducibility. This is true for both biased and unbiased scenarios.



(a) biased scenario



(b) unbiased scenario

Fig. 5.3. Stacked histograms of model rankings estimated through 1000 bootstrap samples of seeds replicates across all datasets in the biased (a) and unbiased (b) scenario. Models are ordered according to their average performance ranking over all datasets. We note three important observations. 1) PreActResNet models do not stand out as the best performing models in the biased scenario (a) although the hyper-parameters were supposed to favor them. 2) The aggregated results of (b) tend to confirm the superiority of PreActResNets over ResNets. 3) Larger ResNets and PreActResNets did not outperform their smaller counterparts, while the larger VGG is systematically better than its smaller counterpart. This can be verified for all datasets in Figure 5.2.

conclusion would only hold for the optimizer chosen, and may not hold anymore if this source of variance is introduced in the experimental design. Choosing a large set of hyper-parameters to optimize would have the advantage of increasing the robustness of the conclusions one draws. Doing so would however significantly increase the size of the search space and likely hamper the hyper-parameter optimization procedure, making it unpractical. It is worth noting that the current study required the time-equivalent of training over 7000 models for 120 epochs[2].

5.6.3. Other sources of variations

The current study is limited to the stochasticity of the data ordering on which the model is trained and to the stochasticity of the model initialization. There are two other important sources of variations that we here kept fixed.

The first one is the sampling of the datasets. It is common practice to use given datasets as a fixed set of data. There is however a source of variations in the finite sampling of a dataset from a distribution. Using a technique such as cross-validation could help integrate such variation in our experiments without requiring access to the true distribution of the data. Those would be data sampling replicates.

The second source comes from the optimization procedure of the hyper-parameters. The technique we use, ASHA, is in its very own nature stochastic as it can be seen as a sophisticated random search. To include this source of variation we would need to execute several hyper-parameter optimization procedures and average our analyses over all of them. These would be hyper-parameter optimization replicates.

5.7. Open Discussion: exploratory v.s. empirical research

Reproducibility is undeniably bound to a definition of the scientific method. Inferential reproducibility is based on concepts such as falsification from Popper (2005), statistically significant demonstration as described by Fisher (1936) or increasing confirmation as stated by Carnap (1936). From this vantage point, methods reproducibility seems but secondary, playing only an accessory role in the scientific inquiry, i.e. in the proper forming of scientific conclusions.

There have been strong debates however in the second part of the 20th century on the nature of the scientific method. Kuhn (1962) and Feyerabend (1993) amongst others have argued that *the scientific method* described by Popper does not exist. We can indeed observe

[2] 39k+ models if we do not normalize the length of training procedures. ASHA required training 30k models for 15 epochs, 7k+ models for 30 epochs and 1k+ models for 60 epochs. The seed variations required training 1k+ models for 120 epochs.

a growing number of research methods to this day, and methods such as *exploratory research* are widely used and accepted despite their weak compliance with a rigorous application of *the scientific method*. As stated by Leek (2017), limiting all scientific work to *the scientific method* would pose a risk of hampering the progress of science.

Let us clarify what we mean by empirical and exploratory research.

Empirical research: Its goal is to test an hypothesis. It aims to build a robust corpus of knowledge. It has the advantage of favoring stable progress of a scientific field. As previously outlined, *inferential reproducibility* is strongly linked to empirical research.

Exploratory research: Its goal is to explore a new subject and gather new observations. It aims to expand the research horizon with new corpus of knowledge and favors fast progress of a scientific field. *Methods and results reproducibility* have the advantage of facilitating the diffusion of knowledge and exploration by providing tools to extend existing research and are thus strongly linked to exploratory research.

A too large proportion of the research devoted to exploratory research increases the risk of seeing lines of research collapsing because of building on non-robust basis, while a too large proportion devoted to empirical research increases the risk of hampering the progress by limiting exploration. We do not know what the proper balance is. We can however easily claim that the situation of research in deep learning is currently insufficiently balanced in favor of exploratory research.

The risks of line of research collapses are slowly emerging, as suggested by recent works (Melis et al., 2018; Henderson et al., 2018; Lucic et al., 2018). Sculley et al. (2018) drew attention to the problem, controversially arguing that current methodology in deep learning research is akin to "alchemy". In light of this it is important to understand the tension between exploratory and empirical research, because although both are valuable, they do not play the same role. While Batch-Norm (Ioffe & Szegedy, 2015) was criticized by Sculley et al. (2018), we can actually use it as an example to demonstrate the importance of both research methods. Although Ioffe & Szegedy (2015) include empirical experiments in their work, it could hardly be considered as *empirical research* since the data used to build the evidence would be considered insufficient to substantially support the claims of a superior training approach due to the reduction of internal covariate shift[3]. This however does not invalidate their impactful contribution, and there is now undeniable confirmations that Batch-Norm provides improvements in large models, such as ResNets (He et al., 2016b), though likely not due to the reduction of internal covariate shift (Santurkar et al., 2018). The risk with exploratory research is that the *findings* and *conclusions* are brittle and may rest on unstated or unverified assumptions. Consequently using them as a basis for further exploratory research should be exercised with great caution. The example of Batch-Norm is

[3]According to the position of Vaux et al. (2012) on research in epidemiology, the small amount of data of most deep learning paper would not be enough to classify them as *empirical research*.

interesting here because, it was revolutionary for the construction of deeper models, and led to a significant number of works (Salimans & Kingma, 2016; Ba et al., 2016; Cooijmans et al., 2016; Arpit et al., 2016) that focused on normalizations, ahead of a good understanding of *why* Batch-Norm works. The internal covariate shift assumption was debunked much later (Santurkar et al., 2018).

Both *exploratory* and proper *empirical research* methods have their role to play in science, and progress in one should support the other. Recognizing their distinct valuable roles, instead of confusing them or arguing one is superior to the other, will certainly lead to a more rational, harmonious, and efficient development of the field, with earlier detected dead ends, and less time and effort wasted globally. Ideally, a promising exploratory work such as Ioffe & Szegedy (2015) should have led more directly to an empirical work such as Santurkar et al. (2018). In short, methods and results reproducibility will mostly help exploratory research, speeding the exploration further with readily available code, while better experimental design will help support robust conclusions as required by inferential reproducibility. This in turn will establish solid empirical ground, on which the community can build further exploration and empirical studies, with increased confidence.

5.8. Conclusion

We have highlighted the problem of reproducibility of findings due to improper experimental design and presented experiments to showcase how current practice methodologies to benchmark deep convolutional models on image classification tasks are sensitive to this. It is important to take into consideration and investigate sources of variability that *should not* affect the conclusion. As the community embraces rigorous methodologies of empirical research, we believe large scale analysis that include all important sources of variations will provide new insights that could not be discovered through current common methodologies.

Comparing models on different datasets makes it difficult to claim absolute superiority, as the rankings rarely holds across many of them, but it also provides useful information. As outlined by Sculley et al. (2018), the No Free Lunch Theorem (Wolpert et al., 1997) still applies and as such negative performances of a new model should also be reported. These negative results are crucial for the understanding of the underlying principles that make a model better than another on a set of tasks. By identifying in what situations a model fails to deliver on its promises, it becomes possible to identify the shared properties on the corresponding tasks, shedding light on the implicit biases that are shared by the model and the tasks.

Acknowledgments

We thank CIFAR, and Facebook for funding that supported this research, and Compute Canada and Microsoft Azure for computational resources. We also thank Michael Noukhovitch, Mirko Bronzi, Brady Neal, Zafarali Ahmed and Rithesh Kumar for their insightful comments.

Recent developments

The literature on reproducibility issues continues to grow since the publication of this work. New domains are covered, from recommender systems (Dacrema et al., 2019), neural architecture pruning (Blalock et al., 2020), metric learning algorithms (Musgrave et al., 2020) to transformers (Narang et al., 2021).

In the massive work of Raff (2019), more than 200 papers are reproduced in an attempt to quantify the reproducibility rates based on different criteria such as whether pseudo-code was provided or whether the hyperparameter values were shared. Surprisingly, code sharing did not stand out as an important guarantee of reproducibility. On the other hand, pseudo-code as well as readability appeared to be highly correlated with reproducibility.

In Recht et al. (2018) and Recht et al. (2019), the authors realize the colossal task of creating new CIFAR-10 and ImageNet test sets following the original procedures of both datasets. The goal was not evaluating the reproducibility of these dataset creations, but rather measuring the drift of performance measures of existing learning algorithms benchmarked on these datasets. The authors expected to observe some level of overfitting to the test sets, with some algorithms having their performance severely degraded on the new test set. What they observed however is that all algorithm performances seem proportionally affected. These results suggested that no severe overfitting was occurring, and that rankings on these datasets are reliable. Rankings may not have been totally scrambled, however there were significant shifts in these rankings with some algorithms dropping or climbing by up to 4 positions.

In Roelofs et al. (2019), a related group conducted a meta-analysis of benchmarks on the machine learning competition platform Kaggle. The general conclusion of the author was that no significant overfitting was observed on the public test sets. The authors also considered the variability of performance measures as relatively small and thus not concerning. The variability was however only measured in terms of the metric, not in terms of the ranking, and the magnitude was not compared to any reference to justify on which grounds it may be considered small. Looking at the rankings instead of metrics, a similar pattern as in Recht et al. (2018) and Recht et al. (2019) can be observed. The rankings are unstable.

Another work from the same group (Mania et al., 2019) has shown that test sets may be reused more than previously believed as long as the learning algorithms share very similar features or behaviors. They show that as long as the algorithms have very similar predictions, selecting algorithms among these is unlikely to lead to overfitting on the test sets. This work explains in part the observations of Recht et al. (2018), Recht et al. (2019) and perhaps Roelofs et al. (2019) to some degree.

Engstrom et al. (2020) further studied the replication of ImageNet from Recht et al. (2019) and identified biases in the replication. After correction of these biases only 3.6% $\pm$

1.5% of the original 11.7% ± 1.0% accuracy drop remains unexplained. It remains anyhow that the variability of the performance measure can lead to significant ranking changes and must be accounted for to obtain reliable comparisons.

Chapter 6

Article: Accounting for Variance in Machine Learning Benchmarks

The article reproduced below was published as:

Xavier Bouthillier, Pierre Delaunay, Mirko Bronzi, Assya Trofimov, Brennan Nichyporuk, Justin Szeto, Naz Sepah, Edward Raff, Kanika Madan, Vikram Voleti, Samira Ebrahimi Kahou, Vincent Michalski, Dmitriy Serdyuk, Tal Arbel, Chris Pal, Gaël Varoquaux and Pascal Vincent. "Accounting for variance in machine learning benchmarks." *Proceedings of Machine Learning and Systems*, 3 (2021).

Context

Our work on reproducibility (Bouthillier et al., 2019) shed light on the unreliability of comparisons based on single trainings but provided no guidelines on how to account for the variance in these comparisons.

So far, both seminal and more recent works tackling the question of reliability of conclusions when comparing learning algorithms were limited to data sampling (Dietterich, 1998; Nadeau & Bengio, 2000; Bouckaert & Frank, 2004; Riezler & Maxwell, 2005; Taylor Berg-Kirkpatrick & Klein, 2012; Anders Sogaard & Alonso, 2014; Dror et al., 2018; Gorman & Bedrick, 2019).

Hothorn et al. (2005) on the other hand proposed a framework for statistical tests that can incorporate all sources of variance. This framework however appeared unpractical when accounting for the variance due to hyperparameter optimization.

In order to get some measure of these practical limitations, Gaël Varoquaux and I surveyed authors of published papers at two of the most prestigious conferences in machine learning, NeurIPS (2019) and ICLR (2020) (Bouthillier & Varoquaux, 2020)[1].

Based on these measures we designed a series of experiments to evaluate the reliability of comparison methods in practical regimes.

[1]Report of the survey is available in Appendix A

Contributions

This empirical work includes three phases of experiments each providing additional insights.

(1) We measure the variance separately for the different sources of variation in the learning pipeline. Doing so we observe that weights initialization is far from the most important source of variance despite being the gold-standard in the literature to measure neural network performance variance. Variance due to hyperparameter optimization is equally important and variance due to random data splits dominates all other sources of variance.

(2) We measure the quality of mean performance estimation when varying all sources of variation or when using a practical alternative where we keep hyperparameters fixed to good defaults and vary everything else. We observe that keeping fixed hyperparameters deteriorates the reliability of the mean estimation but that varying every other sources of variation helps mitigate this degradation.

(3) Using simulations we examine the behavior of average comparison and statistical tests as a way of drawing conclusions based on ideal or practical mean performance estimations. We observe that the practical estimation affects the reliability of the comparisons, but a statistical test based on it is a significant improvement over average comparisons.

Based on these insights, we make the following recommendations:

(1) Random data splits should be favored against standard dataset splits.

(2) As many sources of variations should be randomized.

(3) Statistical tests should be used to properly account for variance when comparing learning algorithms. We suggest the use of a simple test based on Mann-Withney (Perme & Manevski, 2019).

Authors contributions

Xavier Bouthillier:

- Main idea following Bouthillier et al. (2019).
- Design of experimental pipelines with guidelines for each sub-teams.
- Coordination of the sub-teams.
- Implementation of the experimental pipelines.
- Implementation and execution of CIFAR10-VGG11 experiments.
- Writing code for figures.
- Writing paper.

Pierre Delaunay:

- Implementation of main framework for all experiments.
- Implementation of a general task queue for HPO and complex experiment pipelines.
- Help sub-teams with diverse implementations.

Mirko Bronzi:

- Implementation of Glue data loader wrapper, BERT wrapper and AdamW wrapper.
- Execution of experiments with BERT on Glue-SST2 and Glue-RTE.
- Write appendix on Glue-SST2 and GlueRTE experiments.

Assya Trofimov:

- Implementation of Data loaders for MHC and training pipeline.
- Execution of experiments of MLP trained on MHC.
- Write appendix on MHC-MLP experiments.
- Write code for figures.
- Improving general text.

Brennan Nichyporuk, Justin Szeto, Naz Sepah, Tal Arbel:

- Implementation of PascalVOC data loaders, segmentation model with ResNet18 backend, and training pipeline.
- Execution of PascalVOC-ResNet18 experiments.
- Write appendix on PascalVOC-ResNet18 experiments.

Edward Raff:

- Implementation of data loading and training pipeline for SHWEL (did not make it to final paper).
- Write code for figures.
- Improving general text.

Kanika Madan:

- Implementation of training pipeline for PPO on Mini-grid (did not make it to final paper).

Vikram Voleti, Chris Pal:

- Implementation of training pipeline for WGAN on CIFAR10. (did not make it to final paper).

Samira Ebrahimi Kahou, Vincent Michalski:

- Implementation of training pipeline for YOLO on PascalVOC. (did not make it to final paper).

Dmitriy Serdyuk:

- Implementation of training pipeline for HMM-Hybrid (MLP) on TIMIT. (did not make it to final paper).

Gaël Varoquaux:

- Contributed to ideas through many discussions.
- Write code for figures.
- Improving general text.

Pascal Vincent:

- Contributed to ideas through many discussions.
- Improving general text.
- Notations for the learning pipeline (section 2).

Abstract. Strong empirical evidence that one machine-learning algorithm A outperforms another one B ideally calls for multiple trials optimizing the learning pipeline over sources of variation such as data sampling, augmentation, parameter initialization, and hyperparameters choices. This is prohibitively expensive, and corners are cut to reach conclusions. We model the whole benchmarking process, revealing that variance due to data sampling, parameter initialization and hyperparameter choice impact markedly the results. We analyze the predominant comparison methods used today in the light of this variance. We show a counter-intuitive result that adding more sources of variation to an imperfect estimator approaches better the ideal estimator at a 51× reduction in compute cost. Building on these results, we study the error rate of detecting improvements, on five different deep-learning tasks/architectures. This study leads us to propose recommendations for performance comparisons.

Keywords: Variance, Benchmark, Hyperparameter optimization, Statistical testing

6.1. Introduction: trustworthy benchmarks account for fluctuations

Machine learning increasingly relies upon empirical evidence to validate publications or efficacy. The value of a new method or algorithm A is often established by empirical benchmarks comparing it to prior work. Although such benchmarks are built on quantitative measures of performance, uncontrolled factors can impact these measures and dominate the meaningful difference between the methods. In particular, recent studies have shown that loose choices of hyper-parameters lead to non-reproducible benchmarks and unfair comparisons (Raff, 2019, 2021; Lucic et al., 2018; Henderson et al., 2018; Kadlec et al., 2017; Melis et al., 2018; Bouthillier et al., 2019; Reimers & Gurevych, 2017; Gorman & Bedrick, 2019). Properly accounting for these factors may go as far as changing the conclusions for the comparison, as shown for recommender systems (Dacrema et al., 2019), neural architecture pruning (Blalock et al., 2020), and metric learning (Musgrave et al., 2020).

The steady increase in complexity –e.g. neural-network depth– and number of hyperparameters of learning pipelines increases computational costs of models, making brute-force approaches prohibitive. Indeed, robust conclusions on comparative performance of models A and B would require multiple training of the full learning pipelines, including hyperparameter optimization and random seeding. Unfortunately, since the computational budget of most researchers can afford only a small number of model fits (Bouthillier & Varoquaux, 2020), many sources of variances are not probed via repeated experiments. Rather, sampling several model initializations is often considered to give enough evidence. As we will show, there are other, larger, sources of uncontrolled variation and the risk is that conclusions are driven by differences due to arbitrary factors, such as data order, rather than model improvements.

The seminal work of Dietterich (1998) studied statistical tests for comparison of supervised classification learning algorithms focusing on variance due to data sampling. Following works (Nadeau & Bengio, 2000; Bouckaert & Frank, 2004) perpetuated this focus, including a series of work in NLP (Riezler & Maxwell, 2005; Taylor Berg-Kirkpatrick & Klein, 2012; Anders Sogaard & Alonso, 2014) which ignored variance extrinsic to data sampling. Most of these works recommended the use of paired tests to mitigate the issue of extrinsic sources of variation, but Hothorn et al. (2005) then proposed a theoretical framework encompassing all sources of variation. This framework addressed the issue of extrinsic sources of variation by marginalizing all of them, including the hyper-parameter optimization process. These prior works need to be confronted to the current practice in machine learning, in particular deep learning, where *1)* the machine-learning pipelines has a large number of hyper-parameters, set by uncontrolled procedures, sometimes manually, *2)* the cost of fitting a model is so high that train/validation/test splits are used instead of cross-validation, or nested cross-validation that encompasses hyper-parameter optimization (Bouthillier & Varoquaux, 2020).

In **Section 6.2**, we study the different source of variation of a benchmark, to outline which factors contribute markedly to uncontrolled fluctuations in the measured performance. **Section 6.3** discusses estimating the performance of a pipeline and its uncontrolled variations with a limited budget. In particular we discuss this estimation when hyper-parameter optimization is run only once. Recent studies emphasized that model comparisons with uncontrolled hyper-parameter optimization is a burning issue (Lucic et al., 2018; Henderson et al., 2018; Kadlec et al., 2017; Melis et al., 2018; Bouthillier et al., 2019); here we frame it in a statistical context, with explicit bias and variance to measure the loss of reliability that it incurs. In **Section 6.4**, we discuss criterion using these estimates to conclude on whether to accept algorithm A as a meaningful improvement over algorithm B, and the error rates that they incur in the face of noise.

Based on our results, we issue in **Section 6.5** the following recommendations:

1) As many sources of variation as possible should be randomized whenever possible. These include weight initialization, data sampling, random data augmentation and the whole hyperparameter optimization. This helps decreasing the standard error of the average performance estimation, enhancing precision of benchmarks.

2) Deciding of whether the benchmarks give evidence that one algorithm outperforms another should not build solely on comparing average performance but should account for variance. We propose a simple decision criterion based on requiring a high-enough probability that in one run an algorithm outperforms another.

3) Resampling techniques such as out-of-bootstrap should be favored instead of fixed held-out test sets to improve capacity of detecting small improvements.

Before concluding, we outline a few additional considerations for benchmarking in **Section 6.6**.

6.2. The variance in ML benchmarks

Machine-learning benchmarks run a complete learning pipeline on a finite dataset to estimate its performance. This performance value should be considered the realization of a random variable. Indeed the dataset is itself a random sample from the full data distribution. In addition, a typical learning pipeline has additional sources of uncontrolled fluctuations, as we will highlight below. A proper evaluation and comparison between pipelines should thus account for the *distributions* of such metrics.

6.2.1. A model of the benchmarking process that includes hyperparameter tuning

Here we extend the formalism of Hothorn et al. (2005) to model the different sources of variation in a machine-learning pipeline and that impact performance measures. In particular, we go beyond prior works by accounting for the choice of hyperparameters in a probabilistic model of the whole experimental benchmark. Indeed, choosing good hyperparameters –including details of a neural architecture– is crucial to the performance of a pipeline. Yet these hyperparameters come with uncontrolled noise, whether they are set manually or with an automated procedure.

6.2.1.1. The training procedure

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 $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ containing n examples of (input, target) pairs. These pairs are i.i.d. and sampled from an unknown data distribution $\mathcal{D}$, i.e. $S \sim \mathcal{D}^n$. The goal of a learning pipeline is to find 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* $\hat{R}_e(h, S) = \frac{1}{|S|}\sum_{(x,y)\in S} e(h(x),y)$. In practice, training with a training set S^t consists in finding a function (hypothesis) $h \in \mathcal{H}$ that minimizes a trade-off between a data-fit term – typically the empirical risk of a differentiable surrogate loss e' – with a regularization $\Omega(h, \lambda)$ that induces a preference over hypothesis functions:

$$\mathrm{Opt}(S^t, \lambda) \approx \arg\min_{h\in\mathcal{H}} \hat{R}_{e'}(h, S^t) + \Omega(h, \lambda), \tag{6.2.1}$$

where λ represents the set of hyperparameters: regularization coefficients (s.a. strength of weight decay or the ridge penalty), architectural hyperparameters affecting $\mathcal{H}$, optimizer-specific ones such as the learning rate, etc... Note that Opt is a random variable whose value will depend also on other additional random variables that we shall collectively denote ξ_O, sampled to determine parameter initialization, data augmentation, example ordering, etc.[2].

6.2.1.2. Hyperparameter Optimization

The training procedure builds a predictor given a training set S^t. But since it requires specifying hyperparameters λ, a complete learning pipeline has to tune all of these. A complete pipeline will involve a hyper-parameter optimization procedure, which will strive to find a value of λ that minimizes objective

$$r(\lambda) = \mathbb{E}_{(S^t,S^v)\sim \mathrm{sp}(S^{tv})}\left[\hat{R}_e\left(\mathrm{Opt}(S^t,\lambda), S^v\right)\right] \tag{6.2.2}$$

where $\mathrm{sp}(S^{tv})$ is a distribution of random splits of the data set S^{tv} 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 only has access to S^{tv}, hence the expectation over dataset splits. An ideal hyper-parameter optimization would yield $\lambda^*(S^{tv}) = \arg\min_\lambda r(\lambda)$. A concrete hyperparameter optimization algorithm HOpt will however use an average over a small number of train-validation splits (or just 1), and a limited training budget, yielding $\widehat{\lambda^*}(S^{tv}) = \mathrm{HOpt}(S^{tv}) \approx \lambda^*(S^{tv})$. We denoted earlier the sources of random variations in Opt as ξ_O. Likewise, we will denote the sources of variation inherent to HOpt as ξ_H. These encompass the sources of variance related to the procedure to optimize hyperparameters HOpt, whether it is manual or a search procedure which has its arbitrary choices such as the splitting and random exploration.

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

$$\widehat{h^*}(S^{tv}) = \mathcal{P}(S^{tv}) = \mathrm{Opt}(S^{tv}, \mathrm{HOpt}(S^{tv})) \tag{6.2.3}$$

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

[2]If stochastic data augmentation is used, then optimization procedure Opt for a given training set S^t has to be changed to an expectation over $\tilde{S}^t \sim P^{\mathrm{aug}}(\tilde{S}^t|S^t;\lambda_{\mathrm{aug}})$ e 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 with a random number generator.

6.2.1.3. The performance measure

The full learning procedure $\mathcal{P}$ described above yields a model $\widehat{h^*}$. We now must define a metric that we can use to evaluate the performance of 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 $S^{tv} \sim \mathcal{D}^n$ of size n is:

$$R_{\mathcal{P}}(\mathcal{D}, n) = \mathbb{E}_{S^{tv} \sim \mathcal{D}^n} \left[R_e(\widehat{h^*}(S^{tv}), \mathcal{D}) \right] \tag{6.2.4}$$

where the expectation is also over the random sources ξ that affect the learning procedure (initialization, ordering, data-augmentation) and hyperparameter optimization.

As we only have access to a single finite dataset S, the performance of the learning pipeline can be evaluated as the following expectation over splits:

$$\mu = \hat{R}_{\mathcal{P}}(S, n, n') = \mathbb{E}_{(S^{tv}, S^o) \sim \mathrm{sp}_{n,n'}(S)} \left[\hat{R}_e(\widehat{h^*}(S^{tv}), S^o) \right] \tag{6.2.5}$$

where $\mathrm{sp}_{n,n'}(S)$ is a distribution of 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 as σ^2 the corresponding variance of $\hat{R}_e(\widehat{h^*}(S^{tv}), S^o)$. The performance measures vary not only depending on how the data was split, but also on all other random factors affecting the learning procedure (ξ_O) and hyperparameters optimization (ξ_H).

6.2.2. Empirical evaluation of variance in benchmarks

We conducted thorough experiments to probe the different sources of variance in machine learning benchmarks.

6.2.2.1. Cases studied

We selected i) the CIFAR10 (Krizhevsky et al., 2009) image classification with VGG11 (Simonyan & Zisserman, 2014), ii) PascalVOC (Everingham et al., 2012) image segmentation using an FCN (Long et al., 2014) with a ResNet18 (He et al., 2016a) backbone pretrained on imagenet (Deng et al., 2009), iii-iv) Glue (Wang et al., 2018) SST-2 (Socher et al., 2013) and RTE (Bentivogli et al., 2009) tasks with BERT (Devlin et al., 2018) 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 –which used $\sim$ 8 GPU years– can be found in Appendix B.4.

6.2.2.2. Variance in the learning procedure: ξ_O

For the sources of variance from the learning procedure (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

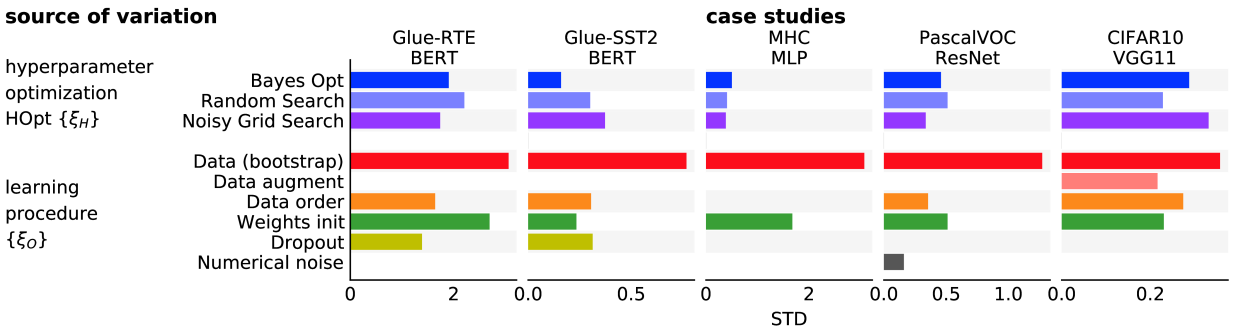


Fig. 6.1. Different sources of variation of the measured performance: across our different case studies, as a fraction of the variance induced by bootstrapping the data. For hyperparameter optimization, we studied several algorithms.

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, the standard source of variance considered in statistical learning. Since we have a single finite dataset in practice, we evaluate this variance by repeatedly generating a train set from bootstrap replicates of the data and measuring the out-of-bootstrap error (Hothorn et al., 2005)[3].

We first fixed hyperparameters to pre-selected reasonable choices[4]. Then, iteratively for each sources of variance, we randomized the seeds 200 times, while keeping all other sources fixed to initial values. Moreover, we measured the numerical noise with 200 training runs with all fixed seeds.

Figure 6.1 presents the individual variances due to sources from within the learning algorithms. Bootstrapping data stands out as the most important source of variance. In contrast, model initialization generally is less than 50% of the variance of bootstrap, on par with the visit order of stochastic gradient descent. Note that these different contributions to the variance are not independent, the total variance cannot be obtained by simply adding them up.

For classification, a simple binomial can be used to model the sampling noise in the measure of the prediction accuracy of a trained pipeline on the test set. Indeed, if the pipeline has a chance τ of giving the wrong answer on a sample, if it makes i.i.d. errors, and if performance is measured on n samples, the observed measure follows a binomial distribution of location parameter τ with n degrees of freedom. If the errors are correlated, not i.i.d., the degrees of freedom are smaller and the distribution is wider. Figure 6.2 compares standard deviations of the performance measure given by this simple binomial model to those observed when bootstrapping the data on the three classification case studies. The match between

[3]The more common alternative in machine learning is to use cross-validation, but the latter is less amenable to various sample sizes. Bootstrapping is discussed in more detail in Appendix B.2.
[4]This choice is detailed in Appendix B.4.

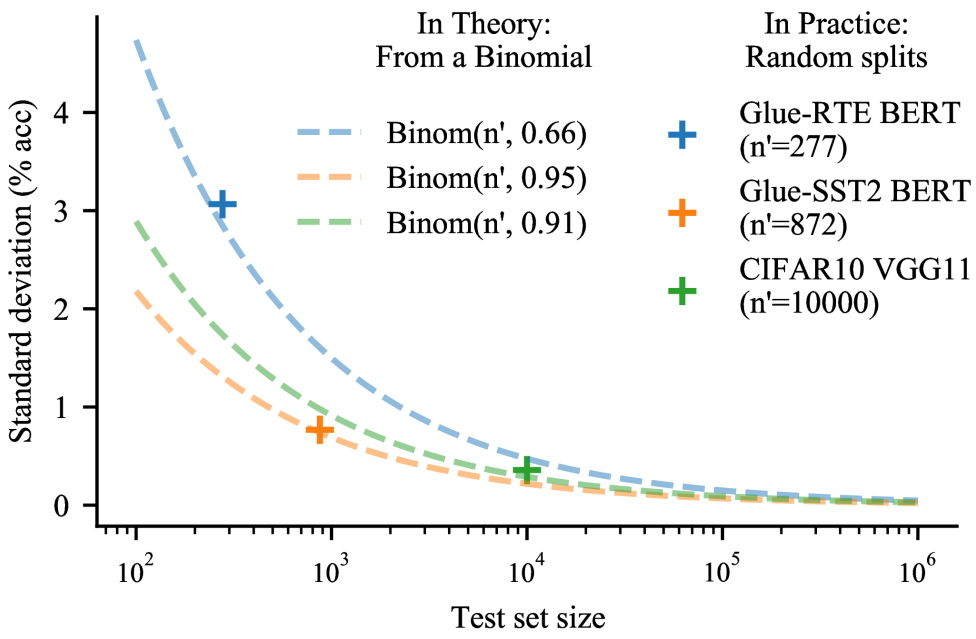


Fig. 6.2. Error due to data sampling: The dotted lines show the standard deviation given by a binomial-distribution model of the accuracy measure; the crosses report the standard deviation observed when bootstrapping the data in our case studies, showing that the model is a reasonable.

the model and the empirical results suggest that the variance due to data sampling is well explained by the limited statistical power in the test set to estimate the true performance.

6.2.2.3. Variance induced by hyperparameter optimization: ξ_H

To study the sources of variation ξ_H, we chose three of the most popular hyperparameter optimization methods: i) random search, ii) grid search, and iii) Bayesian optimization. While grid-search in itself has no random parameters, the specific choice of the parameter range is arbitrary and can be an uncontrolled source of variance (e.g., does the grid size step by powers of 2, 10, or increments of 0.25 or 0.5). We study this variance with a *noisy grid search*, perturbing slightly the parameter ranges (details in Appendix B.5).

For each of these 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. We were careful to design the search space so that it covers the optimal hyperparameter values (as stated in original studies) while being large enough to cover suboptimal values as well.

Results in figure 6.1 show that hyperparameter choice induces a sizable amount of variance, not negligible in comparison to the other factors. The full optimization curves of

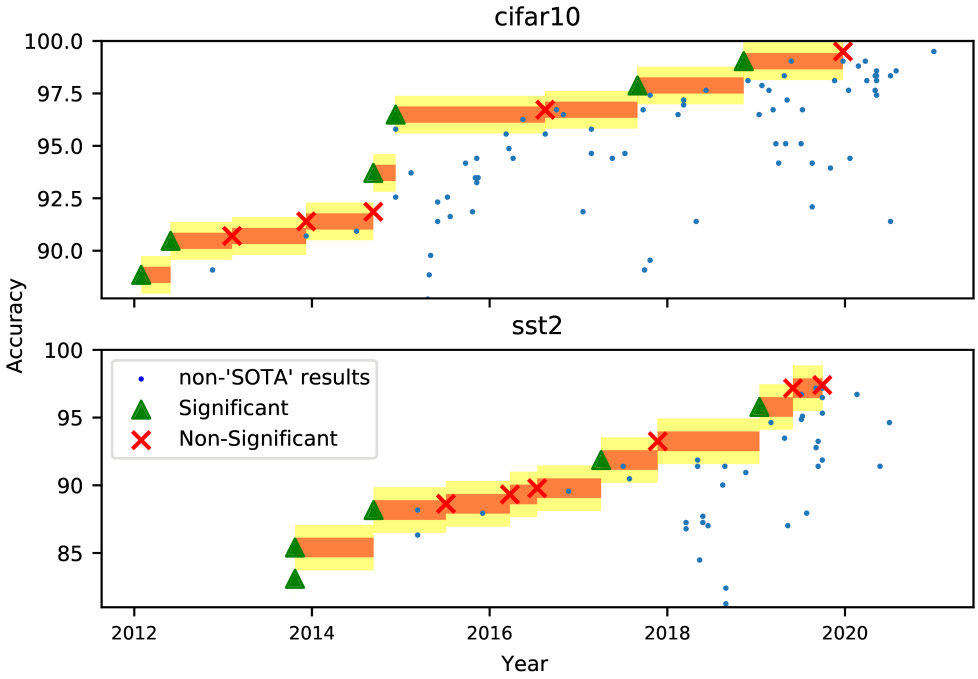


Fig. 6.3. Published improvements compared to benchmark variance The dots give the performance of publications, function of year, as reported on `paperswithcode.com`; red band shows our estimated σ, and the yellow band the resulting significance threshold. Green marks are results likely significant compared to prior 'State of the Art', and red "×" appear non-significant.

the 320 HPO procedures are presented in Appendix B.6. The three hyperparameter optimization methods induce on average as much variance as the commonly studied weights initialization. These results motivate further investigation the cost of ignoring the variance due to hyperparameter optimization.

6.2.2.4. The bigger picture: Variance matters

For a given case study, the total variance due to arbitrary choices and sampling noise revealed by our study can be put in perspective with the published improvements in the state-of-the-art. Figure 6.3 shows that this variance is on the order of magnitude of the individual increments. In other words, the variance is not small compared to the differences between pipelines. It must be accounted for when benchmarking pipelines.

6.3. Accounting for variance to reliably estimate performance $\hat{R}_{\mathcal{P}}$

This section contains 1) an explanation of the counter intuitive result that accounting for more sources of variation reduces the standard error for an estimator of $\hat{R}_{\mathcal{P}}$ and 2) an empirical measure of the degradation of expected empirical risk estimation due to neglecting HOpt variance.

We will now consider different *estimators* of the average performance $\mu = \hat{R}_{\mathcal{P}}(S,n,n')$ from Equation 6.2.5. Such estimators will use an empirical average over k (train+test) splits in place of the expectation of Equation 6.2.5, which we will denote $\hat{\mu}_{(k)}$ and $\hat{\sigma}^2_{(k)}$ the corresponding empirical variance. We will make an important distinction between an estimator which encompasses all sources of variation, the *ideal estimator* $\hat{\mu}_{(k)}$, and one which accounts only for a portion of these sources, the *biased estimator* $\tilde{\mu}_{(k)}$.

But before delving into this, we will explain why many splits help estimating the expected empirical risk ($\hat{R}_{\mathcal{P}}$).

6.3.1. Multiple data splits for smaller detectable improvements

The majority of machine-learning benchmarks are built with fixed training and test sets. The rationale behind this design, is that learning algorithms should be compared on the same grounds, thus on the same sets of examples for training and testing. While the rationale is valid, it disregards the fact that the fundamental ground of comparison is the true distribution from which the sets were sampled. This finite set is used to compute the expected empirical risk ($\hat{R}_{\mathcal{P}}$ Eq 6.2.5), failing to compute the expected risk ($R_{\mathcal{P}}$ Eq 6.2.4) on the whole distribution. This empirical risk is therefore a noisy measure, it has some uncertainty because the risk on a particular test set gives limited information on what would be the risk on new data. This uncertainty due to data sampling is not small compared to typical improvements or other sources of variation, as revealed by our study in the previous section. In particular, figure 6.2 suggests that the size of the test set can be a limiting factor.

When comparing two learning algorithms A and B, we estimate their expected empirical risks $\hat{R}_{\mathcal{P}}$ with $\hat{\mu}_{(k)}$, a noisy measure. The uncertainty of this measure is represented by the standard error $\frac{\sigma}{\sqrt{k}}$ under the normal assumption[5] of $\hat{R}_e$. This uncertainty is an important aspect of the comparison, for instance it appears in statistical tests used to draw a conclusion in the face of a noisy evidence. For instance, a z-test states that a difference of expected empirical risk between A and B of at least $z_{0.05}\sqrt{\frac{\sigma_A^2+\sigma_B^2}{k}}$ must be observed to control false

[5]Our extensive numerical experiments show that a normal distribution is well suited for the fluctuations of the risk – Figure B.3

detections at a rate of 95%. In other words, a difference smaller than this value could be due to noise alone, e.g. different sets of random splits may lead to different conclusions.

With $k = 1$, algorithms A and B must have a large difference of performance to support a reliable detection. In order to detect smaller differences, k must be increased, i.e. $\hat{\mu}_{(k)}$ must be computed over several data splits. The estimator $\hat{\mu}_{(k)}$ is computationally expensive however, and most researchers must instead use a biased estimator $\tilde{\mu}_{(k)}$ that does not probe well all sources of variance.

6.3.2. Bias and variance of estimators depends on whether they account for all sources of variation

Probing all sources of variation, including hyperparameter optimization, is too computationally expensive for most researchers. However, ignoring the role of hyperparameter optimization induces a bias in the estimation of the expected empirical risk. We discuss in this section the expensive, unbiased, ideal estimator of $\hat{\mu}_{(k)}$ and the cheap biased estimator of $\tilde{\mu}_{(k)}$. We explain as well why accounting for many sources of variation improves the biased estimator by reducing its bias.

6.3.2.1. Ideal estimator: sampling multiple HOpt

The ideal estimator $\hat{\mu}_{(k)}$ takes into account all sources of variation. For each performance measure $\hat{R}_e$, all ξ_O and ξ_H are randomized, each requiring an independent hyperparameter optimization procedure. The detailed procedure is presented in Algorithm 1. For an estimation over k splits with hyperparameter optimization for a budget of T trials, it requires fitting the learning algorithm a total of $O(k \cdot T)$ times. The estimator is unbiased, with $\mathbb{E}\left[\hat{\mu}_{(k)}\right] = \mu$.

For a variance of the performance measures $\mathrm{Var}(\hat{R}_e) = \sigma^2$, we can derive the variance of the ideal estimator $\mathrm{Var}(\hat{\mu}_{(k)}) = \frac{\sigma^2}{k}$ by taking the sum of the variances in $\hat{\mu}_{(k)} = \frac{1}{k}\sum_{i=1}^{k} \hat{R}_{ei}$. We see that with $\lim_{k\to\infty} \mathrm{Var}(\hat{\mu}_{(k)}) = 0$. Thus $\hat{\mu}_{(k)}$ is a well-behaved unbiased estimator of μ, as its mean squared error vanishes with infinitely large k:

$$\begin{aligned} \mathbb{E}[(\hat{\mu}_{(k)} - \mu)^2] &= \mathrm{Var}(\hat{\mu}_{(k)}) + (\mathbb{E}[\hat{\mu}_{(k)}] - \mu)^2 \\ &= \frac{\sigma^2}{k} \end{aligned} \tag{6.3.1}$$

Note that T does not appear in these equations. Yet it controls HOpt's runtime cost (T trials to determine $\hat{\lambda}^*$), and thus the variance σ^2 is a function of T.

6.3.2.2. Biased estimator: fixing HOpt

A computationally cheaper but biased estimator consists in re-using the hyperparameters obtained from a *single* hyperparameter optimization to generate k subsequent performance

Algorithm 1 `IdealEst`
Ideal Estimator $\hat{\mu}_{(k)}$,$\hat{\sigma}_{(k)}$

Input:
dataset S
sample size k

for i in $\{1, \cdots, k\}$ **do**
$\xi_O \sim \text{RNG}()$
$\xi_H \sim \text{RNG}()$
$S^{tv}, S^o \sim \text{sp}(S; \xi_O)$
$\widehat{\lambda^*} = \text{HOpt}(S^{tv}, \xi_O, \xi_H)$
$\widehat{h^*} = \text{Opt}(S^{tv}, \widehat{\lambda^*})$
$p_i = \hat{R}_e(\widehat{h^*}, S^o)$
end for
Return $\hat{\mu}_{(k)} = \text{mean}(p)$,
$\hat{\sigma}_{(k)} = \text{std}(p)$

Algorithm 2 `FixHOptEst`
Biased Estimator $\tilde{\mu}_{(k)}$,$\tilde{\sigma}_{(k)}$

Input:
dataset S
sample size k

$\xi_O \sim \text{RNG}()$
$\xi_H \sim \text{RNG}()$
$S^{tv}, S^o \sim \text{sp}(S; \xi_O)$
$\widehat{\lambda^*} = \text{HOpt}(S^{tv}, \xi_O, \xi_H)$
for i in $\{1, \cdots, k\}$ **do**
$\xi_O \sim \text{RNG}()$
$S^{tv}, S^o \sim \text{sp}(S; \xi_O)$
$\widehat{h^*} = \text{Opt}(S^{tv}, \widehat{\lambda^*})$
$p_i = \hat{R}_e(\widehat{h^*}, S^o)$
end for
Return $\tilde{\mu}_{(k)} = \text{mean}(p)$,
$\tilde{\sigma}_{(k)} = \text{std}(p)$

Fig. 6.4. Estimators of the performance of a method, and its variation. We represent the seeding of sources of variations with $\xi \sim \text{RNG}()$, where RNG() is some random number generator. Their difference lies in the hyper-parameter optimization step (HOpt). The ideal estimator requires executing k times HOpt, each requiring T trainings for the hyperparameter optimization, for a total of $O(k \cdot T)$ trainings. The biased estimator requires executing only 1 time HOpt, for $O(k + T)$ trainings in total.

measures $\hat{R}_e$ where only ξ_O (or a subset of ξ_O) is randomized. This procedure is presented in Algorithm 2. It requires only $O(k + T)$ fittings, substantially less than the ideal estimator. The estimator is biased with $k > 1$, $\mathbb{E}\left[\tilde{\mu}_{(k)}\right] \neq \mu$. A bias will occur when a set of hyperparameters $\widehat{\lambda^*}$ are optimal for a particular instance of ξ_O but not over most others.

When we fix sources of variation ξ to arbitrary values (e.g. random seed), we are conditioning the distribution of $\hat{R}_e$ on some arbitrary ξ. Intuitively, holding fix some sources of variations should reduce the variance of the whole process. What our intuition fails to grasp however, is that this conditioning to arbitrary ξ induces a correlation between the trainings which in turns increases the variance of the estimator. Indeed, the variance of a sum of correlated variables increases with the strength of the correlations.

Let $\text{Var}(\hat{R}_e \mid \xi)$ be the variance of the conditioned performance measures $\hat{R}_e$ and ρ the average correlation among all pairs of $\hat{R}_e$. The variance of the biased estimator is then given by the following equation.

$$\text{Var}(\tilde{\mu}_{(k)} \mid \xi) = \frac{\text{Var}(\hat{R}_e \mid \xi)}{k} + \frac{k-1}{k}\rho\text{Var}(\hat{R}_e \mid \xi) \tag{6.3.2}$$

We can see that with a large enough correlation ρ, the variance $\text{Var}(\tilde{\mu}_{(k)} \mid \xi)$ could be dominated by the second term. In such case, increasing the number of data splits k would not reduce the variance of $\tilde{\mu}_{(k)}$. Unlike with $\hat{\mu}_{(k)}$, the mean square error for $\tilde{\mu}_{(k)}$ will not decreases with k :

$$\begin{aligned}\mathbb{E}[(\tilde{\mu}_{(k)} - \mu)^2] &= \text{Var}(\tilde{\mu}_{(k)} \mid \xi) + (\mathbb{E}[\tilde{\mu}_{(k)} \mid \xi] - \mu)^2 \\ &= \frac{\text{Var}(\hat{R}_e \mid \xi)}{k} + \frac{k-1}{k}\rho\text{Var}(\hat{R}_e \mid \xi) \\ &\quad + (\mathbb{E}[\hat{R}_e \mid \xi] - \mu)^2 \end{aligned} \tag{6.3.3}$$

This result has two implications, one beneficial to improving benchmarks, the other not. Bad news first: the limited effectiveness of increasing k to improve the quality of the estimator $\tilde{\mu}_{(k)}$ is a consequence of ignoring the variance induced by hyperparameter optimization. We cannot avoid this loss of quality if we do not have the budget for repeated independent hyperoptimization. The good news is that current practices generally account for only one or two sources of variation; there is thus room for improvement. This has the potential of decreasing the average correlation ρ and moving $\tilde{\mu}_{(k)}$ closer to $\hat{\mu}_{(k)}$. We will see empirically in next section how accounting for more sources of variation moves us closer to $\hat{\mu}_{(k)}$ in most of our case studies.

6.3.3. The cost of ignoring HOpt variance

To compare the estimators $\hat{\mu}_{(k)}$ and $\tilde{\mu}_{(k)}$ presented above, we measured empirically the statistics of the estimators on budgets of $k = (1, \cdots, 100)$ points on our five case studies. The ideal estimator is asymptotically unbiased and therefore only one repetition is enough to estimate $\text{Var}(\hat{\mu}_{(k)})$ for each task. For the biased estimator we run 20 repetitions to estimate $\text{Var}(\tilde{\mu}_{(k)} \mid \xi)$. We sample 20 arbitrary ξ (random seeds) and compute the standard deviation of $\tilde{\mu}_{(k)}$ for $k = (1, \cdots, 100)$.

We compared the biased estimator `FixedHOptEst()` while varying different subset of sources of variations to see if randomizing more of them would help increasing the quality of the estimator. We note `FixedHOptEst(`k`,Init)` the biased estimator $\tilde{\mu}_{(k)}$ randomizing only the weights initialization, `FixedHOptEst(`k`,Data)` the biased estimator randomizing only data splits, and `FixedHOptEst(`k`,All)` the biased estimator randomizing all sources of variation ξ_O except for hyperparameter optimization.

We present results from a subset of the tasks in Figure 6.5 (all tasks are presented in Figure B.4). Randomizing weights initialization only (`FixedHOptEst(`k`,init)`) provides only a small improvement with $k > 1$. In the task where it best performs (Glue-RTE), it converges to the equivalent of $\hat{\mu}_{(k=2)}$. This is an important result since it corresponds to the predominant approach used in the literature today. Bootstrapping with `FixedHOptEst(`k`,Data)`

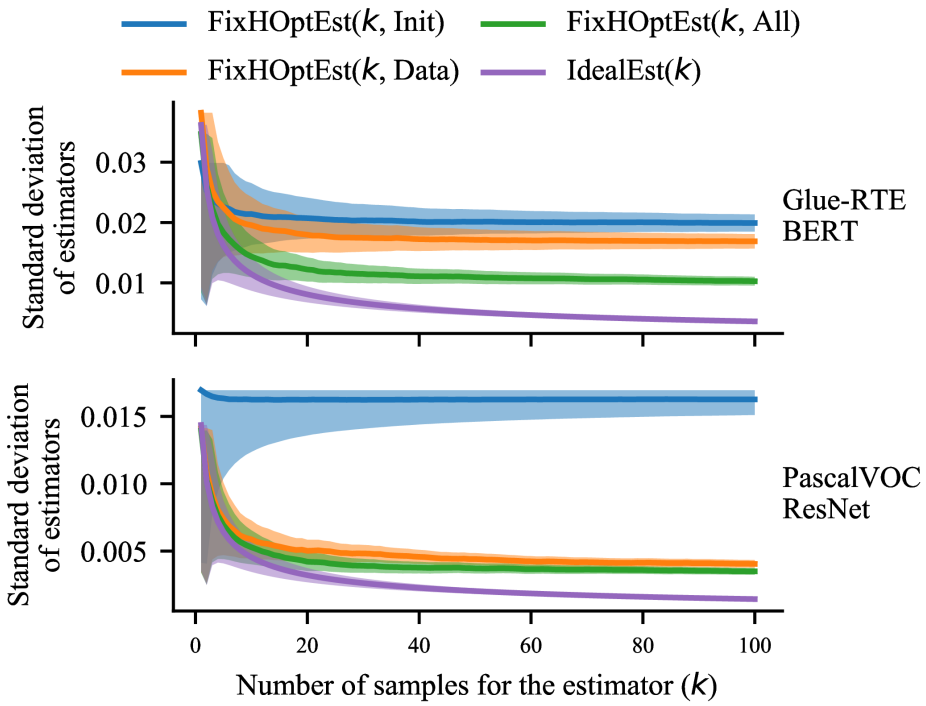


Fig. 6.5. Standard error of biased and ideal estimators with k samples. Top figure presents results from BERT trained on RTE and bottom figure a ResNet on PascalVOC. All other tasks are presented in Figure B.4. On x axis, the number of samples used by the estimators to compute the average classification accuracy. On y axis, the standard deviation of the estimators. Uncertainty represented in light color is computed analytically as the approximate standard deviation of the standard deviation of a normal distribution computed on k samples. For most case studies, **accounting for more sources of variation reduces the standard error of $\hat{\mu}_{(k)}$.** This is caused by the decreased correlation ρ thanks to additional randomization in the learning pipeline. `FixHOptEst(k, All)` provides an improvement towards `IdealEst(k)` for no additional computational cost compared to `FixHOptEst(k, Init)` which is currently considered as a good practice. **Ignoring variance from** HOpt **is harmful for a good estimation of $\hat{R}_{\mathcal{P}}$.**

improves the standard error for all tasks, converging to equivalent of $\hat{\mu}_{(k=2)}$ to $\hat{\mu}_{(k=10)}$. Still, the biased estimator including all sources of variations excluding hyperparameter optimization `FixedHOptEst(`k`,All)` is by far the best estimator after the ideal estimator, converging to equivalent of $\hat{\mu}_{(k=2)}$ to $\hat{\mu}_{(k=100)}$.

This shows that accounting for all sources of variation reduces the likelihood of error in a computationally achievable manner. `IdealEst(`$k = 100$`)` takes 1 070 hours to compute, compared to only 21 hours for each `FixedHOptEst(`$k = 100$`)`. Our study paid the high computational cost of multiple rounds of `FixedHOptEst(`k`,All)`, and the cost of `IdealEst(`k`)` for a total of *6.4 GPU years* to show that `FixedHOptEst(`k`,All)` is better than the status-quo and a satisfying option for statistical model comparisons *without* these prohibitive costs.

6.4. Accounting for variance to draw reliable conclusions

6.4.1. Criteria used to conclude from benchmarks

Given an estimate of the performance of two learning pipelines and their variance, are these two pipelines different in a meaningful way? We first formalize common practices to draw such conclusions, then characterize their error rates.

6.4.1.1. Comparing the average difference

A typical criterion to conclude that one algorithm is superior to another is that one reaches a performance superior to another by some (often implicit) threshold δ. The choice of the threshold δ can be arbitrary, but a reasonable one is to consider previous accepted improvements, e.g. improvements in Figure 6.3.

This difference in performance is sometimes computed across a single run of the two pipelines, but a better practice used in the deep-learning community is to average multiple seeds (Bouthillier & Varoquaux, 2020). Typically hyperparameter optimization is performed for each learning algorithm and then several weights initializations or other sources of fluctuation are sampled, giving k estimates of the risk $\hat{R}_e$ – note that these are biased as detailed in subsubsection 6.3.2.2. If an algorithm A performs better than an algorithm B by at least δ on average, it is considered as a better algorithm than B for the task at hand. This approach does not account for false detections and thus cannot easily distinguish between true impact and random chance.

Let $\hat{R}_e^A = \frac{1}{k}\sum_{i=1}^{k} \hat{R}_{ei}^A$ be the mean performance of algorithm A where $\hat{R}_{ei}^A$ is the empirical risk of algorithm A on the i-th split, and similarly for B. The decision whether A outperforms B is then determined by $(\hat{R}_e^A - \hat{R}_e^B > \delta)$.

The variance is not accounted for in the average comparison. We will now present a statistical test accounting for it. Both comparison methods will next be evaluated empirically using simulations based on our case studies.

6.4.1.2. Probability of outperforming

The choice of threshold δ is problem-specific and does not relate well to a statistical improvement. Rather, we propose to formulate the comparison in terms of *probability of improvement*. Instead of comparing the average performances, we compare their distributions altogether. Let $\mathbb{P}(A > B)$ be the probability of measuring a better performance for A than B across fluctuations such as data splits and weights initialization. To consider an algorithm A significantly better than B, we ask that A outperforms B *often enough*: $\mathbb{P}(A > B) \geq \gamma$. Often enough, as set by γ, needs to be defined by community standards, which we will revisit

below. This probability can simply be computed as the proportion of successes, $\hat{R}_{ei}^{A} > \hat{R}_{ei}^{B}$, where $(\hat{R}_{ei}^{A}, \hat{R}_{ei}^{B}), i \in \{1, \ldots, k\}$ are pairs of empirical risks measured on k different data splits for algorithms A and B.

$$\mathbb{P}(A > B) = \frac{1}{k} \sum_{i}^{k} I_{\{\hat{R}_{ei}^{A} > \hat{R}_{ei}^{B}\}} \tag{6.4.1}$$

where I is the indicator function. We will build upon the non-parametric Mann-Whitney U-test to produce decisions about whether $\mathbb{P}(A > B) \geq \gamma$ (Perme & Manevski, 2019) .

The problem is well formulated in the Neyman-Pearson view of statistical testing (Neyman & Pearson, 1928; Perezgonzalez, 2015), which requires the explicit definition of both a null hypothesis H_0 to control for *statistically significant* results, and an alternative hypothesis H_1 to declare results *statistically meaningful*. A *statistically significant* result is one that is not explained by noise, the null-hypothesis $H_0 : \mathbb{P}(A > B) = 0.5$. With large enough sample size, any arbitrarily small difference can be made *statistically significant*. A *statistically meaningful* result is one large enough to satisfy the alternative hypothesis $H_1 : \mathbb{P}(A > B) \geq \gamma$. Recall that γ is a threshold that needs to be defined by community standards. We will discuss reasonable values for γ in next section based on our simulations.

We recommend to conclude that algorithm A is better than B on a given task if the result is both *statistically significant* and *meaningful*. The reliability of the estimation of $\mathbb{P}(A > B)$ can be quantified using confidence intervals, computed with the non-parametric percentile bootstrap (Efron, 1982). The lower bound of the confidence interval $\mathrm{CI}_{\min}$ controls if the result is *significant* ($\mathbb{P}(A > B) - \mathrm{CI}_{\min} > 0.5$), and the upper bound of the confidence interval $\mathrm{CI}_{\max}$ controls if the result is *meaningful* ($\mathbb{P}(A > B) + \mathrm{CI}_{\max} \geq \gamma$).

6.4.2. Characterizing errors of these conclusion criteria

We now run an empirical study of the two conclusion criteria presented above, the popular *comparison of average differences* and our recommended *probability of outperforming*. We will re-use mean and variance estimates from subsection 6.3.3 with the ideal and biased estimators to simulate performances of trained algorithms so that we can measure the reliability of these conclusion criteria when using ideal or biased estimators.

6.4.2.1. Simulation of algorithm performances

We simulate realizations of the ideal estimator $\hat{\mu}_{(k)}$ and the biased estimator $\tilde{\mu}_{(k)}$ with a budget of $k = 50$ data splits. For the ideal estimator, we model $\hat{\mu}_{(k)}$ with a normal distribution $\hat{\mu}_{(k)} \sim \mathcal{N}(\mu, \frac{\sigma^2}{k})$, where σ^2 is the variance measured with the ideal estimator in our case studies, and μ is the empirical risk $\hat{R}_e$. Our experiments consist in varying the difference in μ for the two algorithms, to span from identical to widely different performance ($\mu_A >> \mu_B$).

For the biased estimator, we rely on a two stage sampling process for the simulation. First, we sample the bias of $\tilde{\mu}_{(k)}$ based on the variance $\text{Var}(\tilde{\mu}_{(k)} \mid \xi)$ measured in our case studies, $Bias \sim \mathcal{N}(0, \text{Var}(\tilde{\mu}_{(k)} \mid \xi))$. Given b, a sample of $Bias$, we sample k empirical risks following $\hat{R}_e \sim \mathcal{N}(\mu + b, \text{Var}(\hat{R}_e \mid \xi))$, where $\text{Var}(\hat{R}_e \mid \xi)$ is the variance of the empirical risk $\hat{R}_e$ averaged across 20 realizations of $\tilde{\mu}_{(k)}$ that we measured in our case studies.

In simulation we vary the mean performance of A with respect to the mean performance of B so that $\mathbb{P}(A > B)$ varies from 0.4 to 1 to test three regions:

H_0 **is true:** : Not significant, not meaningful

$$\mathbb{P}(A > B) - \text{CI}_{\text{min}} \leq 0.5$$

H_0 **&** H_1 **are false (**$\not{H_0}\not{H_1}$**):** : Significant, not meaningful

$$\mathbb{P}(A > B) - \text{CI}_{\text{min}} > 0.5 \wedge \mathbb{P}(A > B) + \text{CI}_{\text{min}} \leq \gamma$$

H_1 **is true:** : Significant *and* meaningful

$$\mathbb{P}(A > B) - \text{CI}_{\text{min}} > 0.5 \wedge \mathbb{P}(A > B) + \text{CI}_{\text{min}} > \gamma$$

For decisions based on comparing averages, we set $\delta = 1.9952\sigma$ where σ is the standard deviation measured in our case studies with the ideal estimator. The value 1.9952 is set by linear regression so that δ matches the average improvements obtained from `paperswithcode.com`. This provides a threshold δ representative of the published improvements. For the probability of outperforming, we use a threshold of $\gamma = 0.75$ which we have observed to be robust across all case studies (See Appendix B.9).

6.4.2.2. Observations

Figure 6.6 reports results for different decision criteria, using the ideal estimator and the biased estimator, as the difference in performance of the algorithms A and B increases (x-axis). The x-axis is broken into three regions: 1) Leftmost is when H_0 is true (not-significant). 2) The grey middle when the result is significant, but not meaningful in our framework ($\not{H_0}\not{H_1}$). 3) The rightmost is when H_1 is true (significant and meaningful).

The single point comparison leads to the worst decision by far. It suffers from both high false positives ($\approx 10\%$) and high false negatives ($\approx 75\%$). The average with $k = 50$, on the other hand, is very conservative with low false positives ($\approx 0\%$) but very high false negatives ($\approx 94\%$). Using the probability of outperforming leads to better balanced decisions, with a high rate of false positives ($\approx 18\%$) on the left but a reasonable rate of false negatives on the right ($\approx 21\%$). When using the ideal estimator, the probability of outperforming is very close to the oracle.

False positives are often considered to be more important than false negatives since we do not want to claim false discoveries. However false negatives are just as important. With a rate of 94% false negatives, it means even if a researcher makes an important discovery, just a few would be able to reproduce it. We would be tempted to believe that the original result was a false positive if it reproduces only 5% of the time, but it can very well be a true

positive that is difficult to reproduce because of too small sample sizes. The high rate of false negatives of the average comparison is a good example of this issue. At $\mathbb{P}(A > B) = 0.75$, A is clearly superior to B, but the average comparison fails to conclude this $\approx 94\%$ of the time.

The main problem with the average comparison is the threshold. A t-test only differs from an average in that the threshold is computed based on the variance of the model performances and the sample size. It is this adjustment of the threshold based on the variance that allows better control on false negatives.

Finally, we observe that the test of probability of outperforming ($\mathbb{P}(A > B)$) controls well the error rates even when used with a biased estimator. Its performance is nevertheless impacted by the biased estimator compared to the ideal estimator. Although we cannot guarantee a nominal control, we confirm that it is a major improvement compared to the commonly used comparison method at no additional cost.

6.5. Our recommendations: good benchmarks with a budget

We now distill from the theoretical and empirical results of the previous sections a set of practical recommendations to benchmark machine-learning pipelines. Our recommendations are pragmatic in the sense that they are simple to implement and cater for limited computational budgets.

6.5.1. Randomize as many sources of variations as possible

Fitting and evaluating a modern machine-learning pipeline comes with many arbitrary aspects, such as the choice of initializations or the data order. Benchmarking a pipeline given a specific instance of these choices will not give an evaluation that generalize to new data, even drawn from the same distribution. On the opposite, a benchmark that varies these arbitrary choices will not only evaluate the associated variance (section 6.2), but also reduce the error on the expected performance as they enable measures of performance on the test set that are less correlated (6.3). This counter-intuitive phenomenon is related to the variance reduction of bagging (Breiman, 1996a; Bühlmann et al., 2002), and helps characterizing better the expected behavior of a machine-learning pipeline, as opposed to a specific fit.

6.5.2. Use multiple data splits

The subset of the data used as test set to validate an algorithm is arbitrary. As it is of a limited size, it comes with a limited estimation quality with regards to the performance

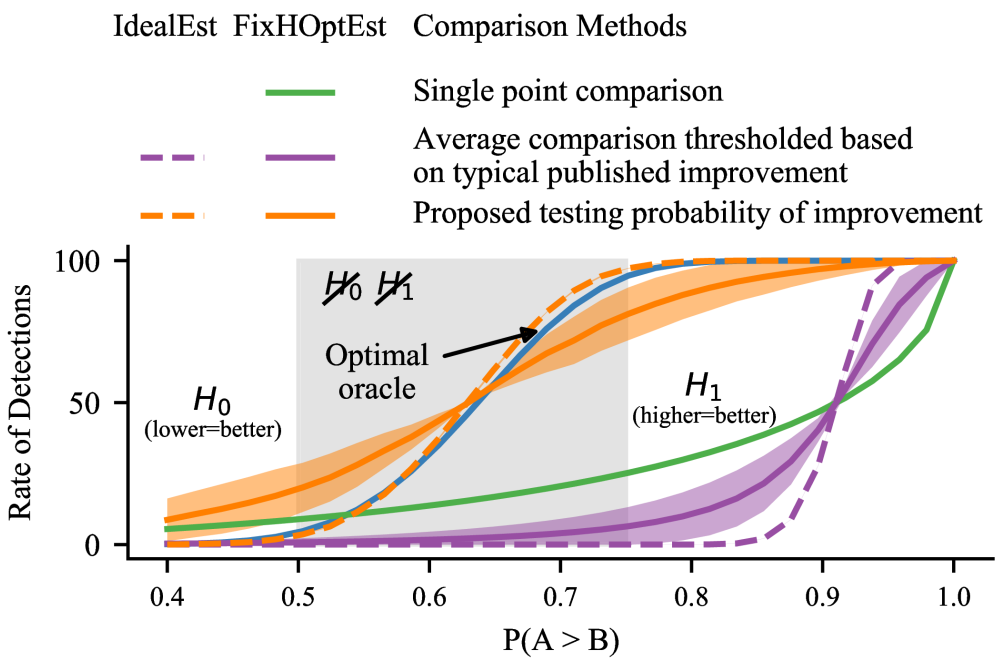


Fig. 6.6. Rate of detections of different comparison methods. x axis is the true simulated probability of a learning algorithm A to outperform another algorithm B across random fluctuations (ex: random data splits). We vary the mean performance of A with respect to that of B so that $\mathbb{P}(A > B)$ varies from 0.4 to 1. The blue line is the optimal oracle, with perfect knowledge of the variances. The single-point comparison (green line) has both a high rate of false positives in the left region ($\approx 10\%$) and a high rate of false negative on the right ($\approx 75\%$). The orange and purple lines show the results for the *average comparison method* (prevalent in the literature) and our proposed *probability of outperforming* method respectively. The solid versions are using the expensive ideal estimator, and the dashed line our 51$\times$ cheaper, but biased, estimator. The average comparison is highly conservative with a low rate of false positives ($\approx 0\%$) on the left and a high rate of false negative on the right ($\approx 94\%$), even with the expensive and exhaustive simulation. Using the probability of outperforming has high rate of false positives ($\approx 18\%$) on the left and a reasonable rate of false negatives on the right ($\approx 21\%$) even when using our biased estimator, and is close to the oracle when using the expensive estimator.

of the algorithm on wider samples of the same data distribution (figure 6.2). Improvements smaller than this variance observed on a given test set will not generalize. Importantly, this variance is not negligible compared to typical published improvements or other sources of variance (figures 6.1 and 6.3). For pipeline comparisons with more statistical power, it is useful to draw multiple tests, for instance generating random splits with a out-of-bootstrap scheme (detailed in appendix B.2).

6.5.3. Account for variance to detect meaningful improvements

Concluding on the significance –statistical or practical– of an improvement based on the difference between average performance requires the choice of a threshold that can be difficult to set. A natural scale for the threshold is the variance of the benchmark, but this variance is often unknown before running the experiments. Using the probability of outperforming $\mathbb{P}(A > B)$ with a threshold of 0.75 gives empirically a criterion that separates well benchmarking fluctuations from published improvements over the 5 case studies that we considered. We recommend to always highlight not only the best-performing procedure, but also all those within the significance bounds. We provide an example in Appendix B.3 to illustrate the application of our recommended statistical test.

6.6. Additional considerations

There are many aspects of benchmarks which our study has not addressed. For completeness, we discuss them here.

6.6.1. Comparing models instead of procedures

Our framework provides value when the user can control the model training process and source of variation. In cases where models are *given* but not under our control (e.g., purchased via API or a competition), the only source of variation left is the data used to test the model. Our framework and analysis does not apply to such scenarios.

6.6.2. Benchmarks and competitions with many contestants

We focused on comparing two learning algorithms. Benchmarks – and competitions in particular – commonly involve large number of learning algorithms that are being compared. Part of our results carry over unchanged in such settings, in particular those related to variance and performance estimation. With regards to reaching a well-controlled decision, a new challenge comes from multiple comparisons when there are many algorithms. A possible alley would be to adjust the decision threshold γ, raising it with a correction for multiple comparisons (e.g. Bonferroni) (Dudoit et al., 2003). However, as the number gets larger, the correction becomes stringent. In competitions where the number of contestants can reach hundreds, the choice of a winner comes necessarily with some arbitrariness: a different choice of test sets might have led to a slightly modified ranking.

6.6.3. Comparisons across multiple dataset

Comparison over multiple datasets is often used to accumulate evidence that one algorithm outperforms another one. The challenge is to account for different errors, in particular different levels of variance, on each dataset.

Demšar (2006) recommended Wilcoxon signed ranks test or Friedman tests to compare classifiers across multiple datasets. These recommendations are however hardly applicable on small sets of datasets – machine learning works typically include as few as 3 to 5 datasets (Bouthillier & Varoquaux, 2020). The number of datasets corresponds to the sample size of these tests, and such a small sample size leads to tests of very limited statistical power.

Dror et al. (2017) propose to accept methods that give improvements on *all* datasets, controlling for multiple comparisons. As opposed to Demšar (2006)'s recommendation, this approach performs well with a small number of datasets. On the other hand, a large number of datasets will increase significantly the severity of the family-wise error-rate correction, making Demšar's recommendations more favorable.

6.6.4. Non-normal metrics

We focused on model performance, but model evaluation in practice can include other metrics such as the training time to reach a performance level or the memory foot-print (Reddi et al., 2020). Performance metrics are generally averages over samples which typically makes them amenable to a reasonable normality assumption.

6.7. Conclusion

We showed that fluctuations in the performance measured by machine-learning benchmarks arise from many different sources. In deep learning, most evaluations focus on the effect of random weight initialization, which actually contribute a small part of the variance, on par with residual fluctuations of hyperparameter choices after their optimization but much smaller than the variance due to perturbing the split of the data in train and test sets. Our study clearly shows that these factors must be accounted for to give reliable benchmarks. For this purpose, we study estimators of benchmark variance as well as decision criterion to conclude on an improvement. Our findings outline recommendations to improve reliability of machine learning benchmarks: 1) randomize as many sources of variations as possible in the performance estimation; 2) prefer multiple random splits to fixed test sets; 3) account for the resulting variance when concluding on the benefit of an algorithm over another.

Acknowledgements

We thank CIFAR, and Facebook for funding that supported this research, and Compute Canada for computational resources. Gaël Varoquaux was partially funded via ANR-17-CE23-0018 DirtyData and DataIA MissingBigData grants.

Recent/Concurrent developments

Statistical Testing. The works of Reimers & Gurevych (2018) and Dror et al. (2019) predate our article on variance but I present them as concurrent developments as we were unaware of them at the time of publishing our work.

Reimers & Gurevych (2018) evaluate the very same statistical test (Evaluation 4) as what we call the probability of outperforming. This statistical test is a particular formulation of the Mann-Whitney U test which dates back to Mann & Whitney (1947) but is rarely used to benchmark machine learning algorithms. Reimers & Gurevych (2018) used it in a purely null hypothesis testing approach, while we recommend its use with a Neyman-Pearson approach, requiring an alternative hypothesis and a power analysis to determine the sample size.

Dror et al. (2019) evaluate a closely related statistical test in the form of stochastic dominance, a more stringent version of the probability of outperforming. This formulation being too constraining they relax it based on del Barrio et al. (2018) and recover a reasonable statistical power. This relaxed version incorporates an ϵ which can be interpreted as an alternative hypothesis, thus bringing their method closer to Neyman-Pearson although it is nowhere explicitly stated. However, for now, the statistical test based on almost stochastic-order does not provide power analysis to determine sample size and further contributions on that front would be greatly valuable. One advantage of this test is that it better preserves its statistical power on heavy-tailed distributions than the Mann-Whitney U test does.

Another important recent work with respect to statistical testing is the one of Card et al. (2020). In their article, they explain that the size of test sets in many NLP benchmarks does not allow for strong statistical power with a McNemar test (Dietterich, 1998; Dror et al., 2018). This statistical test only accounts for the variance due to the sampling of the test set however and ultimately compares predictive models rather than full learning algorithms[6]. A statistical test based on samples of performance estimation like it is done in Reimers & Gurevych (2018); Dror et al. (2019) and in our work can allow for a stronger statistical power.

Hyperparameter Optimization. Several works have studied the importance of hyperparameter optimization to properly benchmark learning algorithms with their best achievable performances (Bergstra & Bengio, 2012; Mantovani et al., 2018; Van Rijn & Hutter, 2018; Probst et al., 2019; Dodge et al., 2019; Sivaprasad et al., 2020). In most cases, the variability of the results due to hyperparameter optimization is implicitly incorporated in the analysis but its effect on benchmarking algorithms is rarely studied in depth.

Building up on Schneider et al. (2019), Schmidt et al. (2020) studies more carefully the effect of hyperparameter optimization on the reliability of optimizer benchmarking and make interesting observations on the variance due to hyperparameter optimization. The

[6]A predictive model can be seen as the output of a learning algorithm.

most relevant observations for our work can be found in Appendix C and D where the authors make 2 observations: 1) There can exist a region in the search space where small movements lead to drastically different objectives. This is particularly problematic when randomization of different sources of variation can randomly shift this region. In such case, some hyperparameter values may be optimal for some seeds and terrible for other seeds leading to a dramatically unstable benchmark. 2) Based on the first observation, they re-execute the entire benchmark to observe the stability with respect to randomization. The results confirm the instability of the ranking of the different optimization methods, making it difficult to identify a superior method even when comparing across different tasks.

Out-of-Distribution Generalisation. One of the strongest limitation of our work is our assumption of i.i.d. distributions for the training, validation and test sets. Recent works such as the one of D'Amour et al. (2020) emphasizes the importance of the representativity of benchmarks for real-task problems as well as their diversity. Synthetic problems and curated datasets are useful to preliminary benchmark learning algorithms, but the generality of the conclusions we draw based on these benchmarks is bounded by the representativity of these tasks for real-world problems. For more general conclusions, there is a serious need to extend our benchmarking guidelines for applicability to problems with distribution shifts.

Chapter 7

Oríon: Open-source software for hyperparameter optimization

The content of this chapter is heavily based on a submission to the Journal of Machine Learning, providing a quick overview of the software. The article is extended here for the purpose of providing more details.

Xavier Bouthillier, Christos Tsirigotis, Pierre Delaunay, Thomas Schweizer, François Corneau-Tremblay, Fabrice Normandin, Nadhir Hassen, Reyhane Askari Hemmat, Michael Noukhovitch, Pascal Lamblin, Frédéric Bastien, Frédéric Osterrath, Irina Rish, Lin Dong, Chao Xue, Junfeng Liu, Sean Wagner and Yonggang Hu. "Oríon: Efficient Hyperparameter Optimization with Experiment Version Control", Under review at *Machine learning Open Source Software track of Journal of Machine Learning Research*, Volume 22 (2021).

Context

During my master and the beginning of my PhD I spent a considerable amount of time on hyperparameter optimization. I was also giving some of my time to supervise students for the development of tools at LISA (later Mila) in what was called the *common code workflow* (CCW). Struggling with current alternatives, Spearmint[1] and HyperOpt[2], I suggested we include hyperparameter optimization as one of the topic to be covered by CCW. Christos Tsirigotis was about to begin an internship working on Theano (Theano Development Team, 2016), but unfortunately development of Theano was stopped a few weeks before his start because of the dominance of alternative frameworks, PyTorch and TensorFlow. Frédéric Bastien and Pascal Lamblin suggested Christos to work instead on hyperparameter optimization, which he agreed.

[1] `https://github.com/JasperSnoek/spearmint`
[2] `https://github.com/hyperopt/hyperopt`

Following a discussion with James Bergstra, former PhD student at LISA and developer of HyperOpt, we decided to start a new framework from scratch. The architecture of the framework HyperOpt seemed too difficult to extend and we wanted to focus strongly on usability.

During the beginning of the development, we conducted an informal survey at Mila, receiving answers from about 50 students[3]. The main observation stemming from the survey is that very few researchers at Mila used frameworks for hyperparameters optimization[4]. Based on the results of the survey, it appears that

(1) Researchers do not trust the algorithms' efficiency

(2) Researchers believe hyperparameter optimization requires more computational resources than manual tuning, and they do not have enough.

(3) Researchers find the libraries too disruptive, the learning curve is too steep for the potential benefits.

These results reinforced our motivation to develop a tool that would be simple to use. We favor usability over complexity and performance. When faced with complex solutions yielding small performance gains, we favor the simpler and less performant solution.

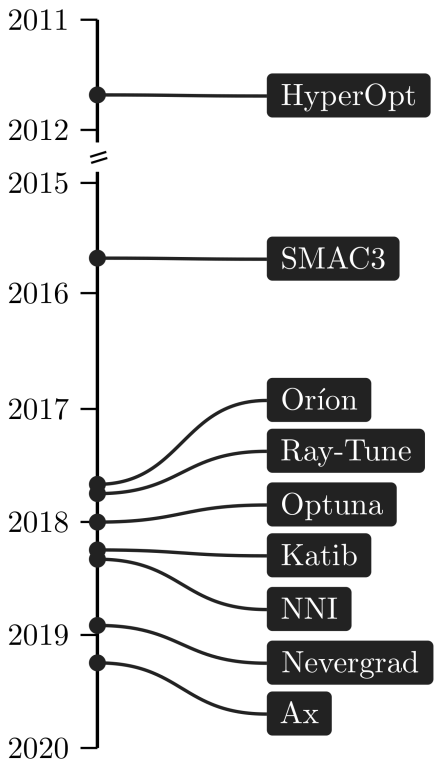


Fig. 7.1. Timeline of HPO frameworks. Today, there are many framework alternatives for hyperparameter optimization. This was not the case however at the time we started developing Oríon.

[3]50 students represented about a quarter of the laboratory at that time.

[4]We would later confirm in 2020 that very few researchers in the world-wide community of researchers in machine learning used frameworks for hyperparameter optimization (Bouthillier & Varoquaux, 2020)

Contributions

Oríon provides a simple command-line interface to turn the training script of researchers into a pipeline for hyperparameter optimization. This also allows Oríon to support any programming language for the training script as long as it can take hyperparameter values as input from the command-line and produce a json file containing the results. Alternatively, hyperparameter values can be defined in configuration files read by the training script. Oríon supports any text-based configuration file, allowing support for most use-cases from researchers.

A new experiment version control system was implemented for Oríon to adapt the hyperparameter optimization pipeline to the workflow of researchers.

To further support various use-cases, many parts of Oríon are implemented to be extendable through plug-ins. This includes algorithms, databases and visualizations.

The submitted paper, limited to 4 pages for the Machine Learning Open Source Software track of JMLR, is extended in this thesis to cover Oríon's features in more detail.

Authors Contributions

Xavier Bouthillier:

- Core developer and supervisor since the beginning of the project in Fall 2017.
- Implemented:
  - Experiment Version Control system.
  - EphemeralDB and PickledDB.
  - Parallel Coordinates, Local Parameter Importance and Partial Dependency plots.
  - Python API.
  - Grid-Search, ASHA (with François Corneau-Tremblay) and Hyperband (with Lin Dong).
- Created a template with automated test-suite for external algorithm plugins.
- Maintenance, test-suite, bug fixes, minor features.

Christos Tsirigotis:

- Internship at Mila (Fall 2017 - Winter 2018)
- Implemented most of Oríon's core:
  - Database.
  - Experiment/Trial.
  - Worker/Consumer.
  - Search space and transformations.
- Collaborated on core design decisions.
- Collaborated on writing and presenting workshop paper at ICML 2018.

Pierre Delaunay:

- Implemented:
  - Storage backend.
  - Track to support third-party storages.

Thomas Schweizer:

- Major rework of the documentation.
- Detailed documentation of the developers guide.
- Wrote scikit-learn example.
- Implemented plotting backend with regret plot.
- Designed WebAPI and implemented server app.
- Collaborated on core design decisions.

François Corneau-Tremblay:

- Internship at Mila (Summer 2018 - Summer 2019)
- Implemented:
  - Base commandline structure.
  - Commandline prompt for EVC.
  - Completed list, status, info commands.
  - Completed ASHA
- Refactored main Factory.
- Experimented for plotting backends.
- Maintenance of test-suite, bug fix, minor features.
- Collaborated on core design decisions.
- Collaborated on writing and presenting workshop paper at ICML2018

Fabrice Normandin:

- Research on warm-starting.

Nadhir Hassen:

- Research on warm-starting.

Reyhane Askari Hemmat:

- Implemented code version tracking for the EVC.
- Collaborated on core design decisions.

Michael Noukhovitch:

- Implemented parallel strategies.
- Collaborated on core design decisions.

Pascal Lamblin:

- Mentoring.

Frédéric Bastien:

- Mentoring.

Frédéric Osterrath:

- Supervision.

Irina Rish:

- Principal Investigator of the collaboration with IBM.

Lin Dong:

- Implemented:
  - Improved implementation of Hyperband.
  - Tree-Structured Parzen Estimator.
  - Benchmarking suite.
  - Search space cardinality as a stopping criterion of algorithms.

Chao Xue:

- Implemented:
  - EvolutionaryES.
  - Example of adaptive search space.

Sean Wagner:

- Coordinating the collaboration between Mila and IBM.
- Collaborated in writing the paper for JMLR.

Yonggang Hu:

- Supervising the collaboration between Mila and IBM.
- Collaborated in writing the paper for JMLR.

Abstract. Oríon is an open source framework for asynchronous distributed hyperparameter optimization. It is adapted to the workflow of machine learning researchers to allow seamless parallelization on most computational infrastructures. It includes a new version control system for experiments, which improves the organization of research projects in machine learning as well as the efficiency of hyperparameter optimization. The entire tool is built with the goals of promoting reproducibility, fair benchmarking of different machine learning models, and providing a platform for the research of black-box optimization algorithms.

Keywords: hyperparameter optimization, black-box optimization, experiment version control, reproducibility

7.1. Introduction

Hyperparameter optimization is one of the most time-consuming parts of research in machine learning. Several classes of models, such as deep neural networks take days or weeks to train, making the process of hyperparameter tuning even more time-consuming (Klein et al., 2016). Despite this, the use of automatic hyperparameter optimization tools is not widespread in the community of deep learning researchers (Bouthillier & Varoquaux, 2020). This causes a serious risk of positive bias, since research is often based on incremental improvements to state-of-the-art methods. This also contributes to the problem of reproducibility when models are highly sensitive to hyperparameter values (Islam et al., 2017; Lucic et al., 2018; Melis et al., 2018; Dodge et al., 2019).

The lack of wide-spread adoption cannot be blamed on the absence of frameworks for hyperparameter optimization as they are numerous (Akiba et al., 2019; Bergstra et al., 2015; Dewancker et al., 2016; Hutter et al., 2011; Kandasamy et al., 2019; Liaw et al., 2018; Mendels & Lahav, 2018; Olson et al., 2016b). We presume that the most important reason for the low adoption rate of these frameworks is the cognitive overhead incurred by using them. In an attempt to address this, we developed Oríon[5], based on a different approach centered on the following idea: **machine learning researchers must be viewed as users**, not as developers. From this perspective, it follows that hyperparameter optimization should be adapted to the workflow of researchers rather than imposed as an API to adapt their code to.

In order to make such an adaptation to the workflow, we propose a **non-intrusive** way of communicating with the user's script. We designed the tool to support user scripts written in any language or framework. It supports the definition of the search space using any text-based configuration files or directly using the command-line. We built it to work **asynchronously** as a way to avoid the need of setting up master and workers, and improving **resiliency**. Finally, we made it incrementally configurable for flexibility and simplicity. To further improve the research workflow, we developed a new experiment version control

[5] `https://github.com/Epistimio/orion`

system for proper experiment management and boosted hyperparameter optimization. With community development in mind, we designed Oríon to be modular and support external contributions as plug-ins. Supporting contributions is an important part of our tool, as one of our main goals is to support research in the area of hyperparameter optimization.

7.2. Black-Box Optimization API adapted to the Research Workflow

Oríon is meant to be simple to configure and operate. It requires little adaptation from the researcher and is compatible with any programming language. We illustrate the core features with a minimal example.

```
import sys
from orion.client import report_objective

x = float(sys.argv[2])
y = train_and_eval(x)

report_objective(y)
```

Suppose the user is using the script presented above to train and evaluate a model with hyperparameter `x`. The user would execute this script in the command-line, passing a value for the hyperparameter `x`.

```
$ python script.py -x 5
```

To make this script compatible with Oríon, the only modification required is to report the objective with `report_objective`. We use Python as an example because Oríon provides a convenient helper function, but the script could be of any programming language. The only requirement is that the script saves the results in a JSON file following Oríon's format standard.

```
$ orion hunt --name exp python script.py -x~'uniform(-5, 5)'
```

Running the script as it is would however only train and evaluate for a given value of `x`. To proceed with hyperparameter optimization, the user must call `orion hunt` and turn `-x 5` into `-x~'uniform(-5, 5)'`. The later, `uniform(-5, 5)`, is a prior which defines the search space that should be explored for a given hyperparameter. Oríon supports all `scipy` distributions, their discretized versions and categorical choices.

During the execution of `orion hunt`, a worker will be spawned to perform the hyperparameter optimization. The worker will connect to a database and register a new experiment with name `exp` or retrieve the corresponding experiment if it already exists. Scaling the

optimizations in parallel across multiple workers is as simple as calling `orion hunt` multiple times.

```
$ orion hunt --name exp &
$ orion hunt --name exp &
$ orion hunt --name exp &
$ orion hunt --name exp
```

As illustrated in Figure 7.2, each worker will maintain a copy of the optimization algorithm, which will be synchronized at each update.

This replication strategy simplifies the scaling up of parallel hyperparameter optimization. There is no need for a persistent master, and the scaling can be dynamic as the availability of resources evolves over time. This strategy also improves the resilience of the whole parallel optimization process. Workers can die without having any effect on the other workers. The only single point of failure is the database itself.

7.3. Features

We will details in this section four different features of Oríon that are distinctive: the various algorithms supported, the asynchronous parallelization of optimization, the experiment version control and the supports for different backends through plug-ins.

7.3.1. Algorithms

Oríon provides a core set of algorithms covering a wide array of trade-offs. Multi-fidelity algorithms Hyperband (Li et al., 2018a), ASHA (Li et al., 2020) and Evolutionary Early-Stopping (So et al., 2019) allow fast optimization when learning curves of optimized task are sufficiently monotonous. The Bayesian optimization algorithm TPE (Bergstra et al., 2011) is a powerful candidate when hyperparameters of the search space are weakly correlated and is well suited for parallel optimization. Bayesian optimization algorithms based on Gaussian Processes on the other hand are less suited for parallel optimization but can better leverage correlated hyperparameters. As a last resort, the random search method (Bergstra & Bengio, 2012) can provide good results, often surprisingly close to more sophisticated algorithms.

Native implementations in Oríon include Random Search, Hyperband, ASHA, Evolutionary Early-Stopping and TPE. Two external plugins further provide wrappers to access the Bayesian optimization algorithms of scikit-optimize (orion.algo.skopt[6]) and RoBO (Klein et al., 2017) (orion.algo.robo[7]). The RoBO wrapper also supports BOHAMIANN (Springenberg et al., 2016), DNGO (Snoek et al., 2015) and ABLR (Perrone et al., 2018), algorithms

[6] `https://pypi.org/project/orion.algo.skopt/`
[7] `https://pypi.org/project/orion.algo.robo/`

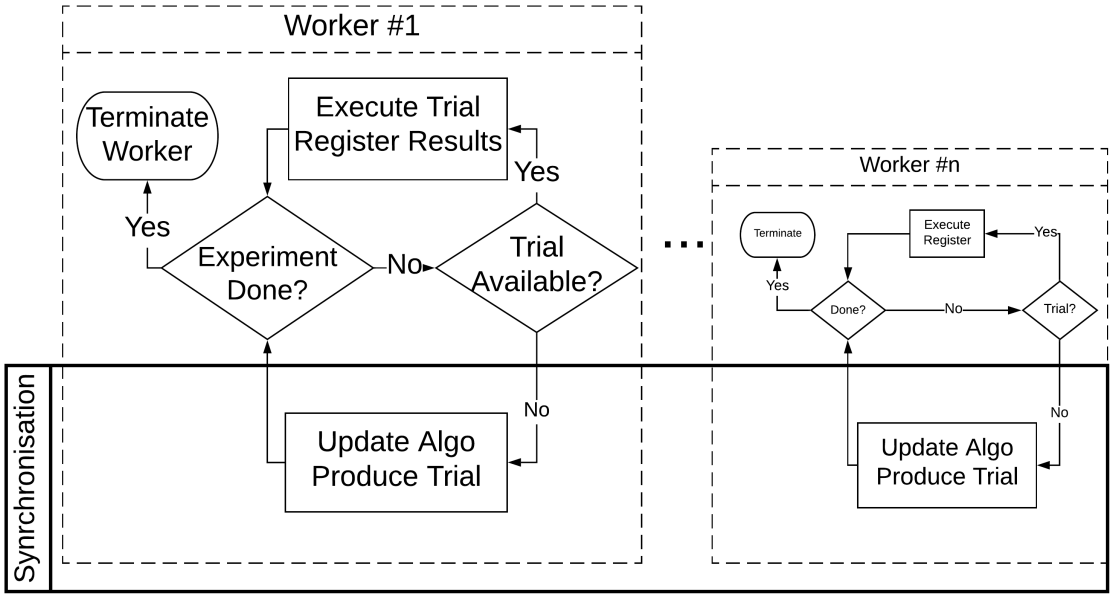


Fig. 7.2. Distributed optimization through replication of algorithm's state: The flow chart illustrates the optimization process for each worker in a pool of N workers. The algorithm state is replicated across all workers, and synchronized at the time of update to produce new trials.

that can be used for more stable (BOHAMIANN) or cheaper computations (DNGO and ABLR).

7.3.2. Parallelism and Asynchronicity

A significant overhead for many frameworks is the process of dispatching workers for concurrent black-box optimization. Most frameworks use a master-workers architecture. This implies that master's process must be deployed either by the user or by a service provider. In the latter case, an API is provided as it is needed from a SaaS (Software as a Service) delivery model. In order to avoid such third-party dependencies, we rather place the responsibility of generating trials inside the workers.

The synchronization point is the database. Sharing information between workers is not achieved by establishing interprocess communication channels. Instead it happens implicitly by reading the common history from the database. Then, workers make decisions based only on that common history. Every operation to the database is implemented in a non-blocking fashion, in order to provide a better throughput.

A *parallel strategy* (Chevalier & Ginsbourger, 2012), also known as the constant liar strategy, is integrated into workers to seamlessly handle sequential algorithms such as Bayesian Optimization. When the optimization algorithm must suggest new trials to execute, any

non-completed trial will be assigned a constant objective – min, mean or max – which will be passed to the algorithm to update it and avoid sampling duplicate trials.

7.3.3. Version control and extended optimization

It is common practice in software development to use version control systems to organize the evolution of the code. Research is no different than code development in its iterative nature. Yet, the task of organizing the research results is still far from being standardized like code version control is. There are fortunately new tools, such as Datmo (Sampat & Shabaz, 2018) and DVC (Kuprieiev, 2018), which aim to improve research organization and reproducibility in the form of version control systems.

However, there is also a lack of features compared to code version control. While the latter saves modifications rather than raw content, current tools for data version control only support tagging and snapshots. Saving modifications rather than raw content makes it possible to reapply these modifications on other contexts, with some adaptations if required. We propose to adopt such practices in our Experiment Version Control, so that trials from one experiment can be used by another one. In addition to helping organization of research projects, such a feature makes it possible to warm start the optimization process of a new experiment. It can thus improve the search efficiency if many related experiments are being executed concurrently. This reduces the time overhead of iterative hyperparameter optimization while maintaining the reproducibility of results.

Suppose a user is working on some experiment `A`. After some time, they either edit the code or want to change the configuration of the experiment. Doing so using Oríon would trigger a **branching event** and create a new experiment `A-v2`. Experiment `A-v2` is a child of `A-v1`[8]. They are connected to each other with an adapter based on the modifications applied on `A-v1` to create `A-v2`. The role of the adapter is to transform trials from `A-v1` to make them compatible with `A-v2` and vice-versa. Thanks to this adapter, it is possible to access trials from `A-v1` while optimizing `A-v2` or the opposite.

Trials belong to a single experiment. When they are fetched from many different experiments in the project tree, they are only grouped together in memory at execution time and are never saved into another experiment inside the database. One may note however, that the algorithm trained on the grouped trials is now biased by the other trials of these other experiments. To ensure full traceability, the unique IDs of the trials affecting the current state of the search algorithm are logged within each new trial at generation time as their *parent trials*.

[8]All new experiments are assigned version 1 by default.

7.3.4. Backends and Plug-ins

Oríon is built to favor community development. Many components are built to be extensible by plug-ins. For example, optimization algorithms can be defined inside a different Python distribution package and be made discoverable by Oríon through the use of Python entry points. This means that a user or a researcher working with a stable or an experimental optimization algorithm can keep their codebase separated from Oríon's codebase, while also being able to use the framework with it. This set of plug-ins can quickly be expanded by the community due to the simple and modular nature of Oríon optimization algorithm interfaces. This also facilitates sharing algorithm implementations while keeping it independent of the core repository so that researchers retain visibility and to favor proper citations[9].

The *plug-inable* parts of Oríon are the storage/database, the algorithms and the parallel strategies. Databases currently supported are a built-in in-memory DB (`EphemeralDB`), a built-in file-based DB (`PickledDB`) and a persistent and performant DB based on `pymongo` (`MongoDB`).

To facilitate the creation of plug-ins for algorithms, we created a template using `cookiecutter`[10]. This template automatically build a minimal working algorithm plug-in, with documentation, and a complete continuous integration pipeline including unit-tests. The algorithm unit-tests provides an extensive suite of tests that allows researchers to easily verify whether their algorithm will execute properly with Oríon.

7.4. Related

An overview of the ecosystem of open-source hyperparameter optimization libraries is presented in Table 1. Oríon is one of the rare frameworks to support any programming language, to be non-intrusive and be compatible with any scheduling system. Katib (George et al., 2020) offers most of these features as well but its strong dependence to Kubernetes severely hampers its usability on academic computational resources.

7.5. Conclusion

Thanks to the Experiment Version Control system we introduced and the features which enable the hierarchy and the modularity of algorithms, experiments and their trials, Oríon is a simple but powerful experimentation platform. Its intuitive and flexible user interface, seamless and fast integration with any research code, as well as its distributed and asynchronous approach, make Oríon an accessible and versatile tool for creating precise and organized work.

[9]Instead of citations being attributed to the framework solely.
[10]`https://github.com/Epistimio/cookiecutter-orion.algo`

	Ax	HyperOpt	Katib	Nevergrad	NNI	Optuna	Ray-Tune	SMAC3	Orion
Languages supported	Py.	Py.	Any	Py. & R	Py.	Py.	Py.	Py.	Any
Disruptive	Low	Mild	Low	Low	Low	High	Mild	High	Low
Scheduling	Any	Any	K8	Any	K8 AML	Any	K8 Slurm	Part.	Any
Auto-scalability	No	Part.	Yes	Part.	Yes	Part.	Yes	Part.	Part.
Fault tolerance	No	Part.	Yes	No	Part.	Part.	Yes	No	Part.
Visualizations	High	Min	Min	Min	Good	High	Min	No*	High
Algorithms									
Bayesian Optimization	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Bandit	Yes	No	Yes	Yes*	Yes	Yes	Yes	Yes	Yes
Evolutionary	No	No	Yes	Yes*	Yes	Yes	Yes	No	Yes
Strategies for efficiency									
Conditional	No	Part.	No	No	Part.	Yes	Yes	Yes	No
Multi-fidelity	Yes	No	Yes	No	Part.	Part.	Part.	Yes	Yes
Warm-starting	Part.	Part.	No	Yes	No	Part.	Part.	Part.	Yes

Table 1. Comparison of features in open-source hyperparameter optimization libraries.

Contributors

In alphabetical order:

Guillaume Alain, Frédéric Bastien, Christopher Beckham, Arnaud Bergeron, Hadrien Bertrand, Olexa Bilaniuk, Steven Bocco, Xavier Bouthillier, Olivier Breuleux, Mirko Bronzi, François Corneau-Tremblay, Pierre Delaunay, Lin Dong, Reyhane Askari Hemmat, Peter Henderson, Yonggang Hu, Pascal Lamblin, Junfeng Liu, Michael Noukhovitch, Satya Ortiz-Gagné, Frédéric Osterrath, Irina Rish, Thomas Schweizer, Dmitriy Serdyuk, Dendi Suhubdy, Christos Tsirigotis, Sean Wagner and Chao Xue.

Acknowledgments

We thank Quebec Minister of Economy and Innovation (MEI), and Canada Foundation for Innovation (FCI) for grants supporting this project. We additionally thank Centers for Advanced Studies (CAS) for enabling a fruitful collaboration between Mila and IBM on Oríon. Finally we are grateful to Angelos Katharopoulos, Ryan Turner, Philip Paquette, Olexa Bilaniuk, Nasim Rahaman, Mathieu Germain, Massimo Caccia, Breandan Considine, Zhouhan Lin, Dmitriy Serdyuk and Jie Fu for the insightful discussions and comments.

Recent developments

Since the beginning of the development of Oríon many other open-source frameworks have appeared in the ecosystem. To name a few, Ray-Tune, Optuna, Katib, NNI, Nevergrad and Ax have gained substantial visibility. So far no frameworks emerged as the dominating ones. Closely related to hyperparameter optimization, the libraries TPOT (Olson et al., 2016a) and auto-sklearn (Matthias Feurer & Hutter, 2015, 2020) became two of the most popular libraries for automating the whole machine learning pipeline (Auto-ML) during the past few years. These Auto-ML libraries are more targeted for non-experts however, whereas Oríon and other hyperparameter optimization frameworks are more targeted for machine learning researchers.

There have been a few important algorithms published during the development of Oríon that have not been integrated yet in it; Improvements to Hyperband such as BOHB (Falkner et al., 2018) or DEBH (Awad et al., 2021), evolutionary algorithms such as Population Based Training (Jaderberg et al., 2017) or hybrid algorithms such as Heteroscedastic Evolutionary Bayesian Optimization (Cowen-Rivers et al., 2020). Population Based Training in particular, is an algorithm that would be beneficial to integrate in Oríon for researchers in the field of reinforcement learning.

During the past few years, research on hyperparameter optimization has strongly transitioned to the subfields of Neural Architecture Search (Hutter et al., 2019). We decided to avoid Neural Architecture Search in the development of Oríon until now for the sake of simplicity and general applicability, it is however a very exciting and promising field. While the deep learning *paradigm* replaced hand-made features with learned ones, Neural Architecture Search seeks to replace hand-made neural network architectures with learned ones. This requires opening the black-box during optimization, exposing the possible architectures in the search space for the optimization algorithms. The only framework referenced here which integrated Neural Architecture Search so far is NNI (`https://github.com/microsoft/nni`).

Optimizing the architecture of neural networks can dramatically increase the size and complexity of the search space. The pressure on computational cost of Neural Architecture Search stimulates further many research directions to estimate model performances more efficiently; Lower fidelities (Chrabaszcz et al., 2017; Zela et al., 2018; Zoph et al., 2018; Real et al., 2019), learning curve extrapolation (Baker et al., 2017; Liu et al., 2018), weights sharing (e.g. initialization using previous best solutions) (Wei et al., 2016; Cai et al., 2018a,b; Elsken et al., 2017, 2019) or even generative curriculum learning to train faster (Such et al., 2020). These solutions could hopefully be reused to also reduce the computational complexity of the expensive ideal estimator discussed in Chapter 6, thus not only helping to improve Neural Architecture Search but also the methodology of machine learning researchers.

On the same topic, Neural Architecture Search exacerbates methodological issues as discussed at the end of Chapter 4, by introducing or scaling the number of factors that must be accounted for when estimating the efficiency of a new method. Hopefully, the awareness of these issues and proposed solutions from the community of researchers in this subfield (Ying et al., 2019; Dong & Yang, 2019; Li & Talwalkar, 2020) will eventually lead to improved methodologies that can benefit the field of machine learning more globally.

Conclusion

The past decade saw prodigious improvements in the field of machine learning. The focus of researchers massively shifted to deep neural networks, methodologies changed and fundamental statistical learning theories were unsettled (Zhang et al., 2016). This thesis addressed questions about these new methodologies, about their reliability in the face of noisy data and proposed recommendations.

In first article, Chapter 5, we criticized a view of *perfect reproducibility*, presented as *methods reproducibility*, demonstrating empirically that even the most stable and consensual benchmarks can mislead researchers if noise is not accounted for properly.

We built upon this work for the second article, Chapter 6, in which we studied the different sources of variations ignored by *perfect reproducibility*. We pushed the analysis further and measured the effect of using cheaper but biased estimators to compare the average performance of the learning algorithms. Based on these results we provided recommendations on the sources of variations that should ideally be accounted for, in particular data sampling, and on the type of comparison methods to use.

Finally, we described a framework for hyperparameter optimization in Chapter 7, an attempt to favor better practices for hyperparameter tuning in machine learning benchmarks, another important part of the reproducibility issues.

I will now discuss interesting directions going forward, starting with hyperparameter optimization followed by reproducibility.

Hyperparameter optimization

One of the main difficulty for the adoption of hyperparameter optimization algorithms is the computational cost of large scale hyperparameter tuning. Researchers rely on prior experience and real-time monitoring to find reasonable hyperparameters in a relatively short amount of time. Unfortunately this practice is unreliable as discussed in Section 4.5, and should only be used to produce preliminary results.

Multi-fidelity methods (Section 4.4) can help lowering the computational cost by leveraging real-time monitoring of objectives, but they generally require more experiments than most researchers are willing to execute based on our survey (Appendix A).

Transfer learning is a very promising direction to solve this issue. For specific scenarios such as training the same algorithm on different datasets, best hyperparameters for one dataset can be inferred from results on the other datasets (Bardenet et al., 2013; Yogatama & Mann, 2014; Joy et al., 2016; Kim et al., 2017; Perrone et al., 2018). On more diverse search spaces that can vary across tasks, large amount of prior data can be used to learn good general defaults (Feurer et al., 2015; Bennani-Smires et al., 2018; Pfisterer et al., 2018). This will be a challenging problem requiring a measure of similarity between tasks in order to make sense of what tasks should be used or not as a basis of knowledge (Tripuraneni et al., 2020). With a proper use of knowledge base to warm-start hyperparameter optimization, we could potentially speedup this optimization enough to allow multiple runs of hyperparameter optimization, thus allowing to account for variance due to hyperparameter optimization on a practical computational budget. This would make the ideal estimator discussed in Chapter 6 affordable, consequently significantly improving the reliability of machine learning benchmark.

Neural architecture search is currently a trending research topic with tremendous potential. The wide adoption of deep neural networks ended an era of feature engineering such as SIFT (Lowe, 1999). It is very compelling to think that Neural Architecture Search could have a similar impact on neural network architecture engineering. To fully blossom, NAS will need to face the challenges to design proper evaluation methodologies as outlined in Section 4.5. Fortunately the future looks promising in that regard (Ying et al., 2019; Lindauer & Hutter, 2020).

Finally, there has been a need already for analytical and visualization tools for hyperparameter optimization but it becomes even more necessary as we embrace neural architecture search with significantly larger and more complex search spaces. Researchers need to better understand the dynamics of the hyperparameters. While it is possible to do so manually when working with only two or three hyperparameters, tools are necessary to distill information from larger search spaces and enable an intuitive view (Hutter et al., 2014; Biedenkapp et al., 2018). I believe visualizations tools will be key to a wider adoption of hyperparameter optimization algorithms from researchers.

Reproducibility

The reproducibility challenge, which invites hundreds of researchers to attempt reproducing recently published works through reimplementation or ablation studies, is certainly the most inspiring initiative towards adressing reproducibility issues (Pineau et al., 2020). It is

an exceptional laboratory to shed light on poor practices causing reproducibility issues and to praise good practices. Additionally, it is a remarkable educational opportunity for young researchers, as they familiarize themselves with the conference publication process including reviewing procedures.

Some issues raised do not have simple solutions however and require investigations that go beyond the reproducibility challenge. An example of such work is the one in Chapter 6 where we sought to evaluate the reliability of comparison methods in order to provide recommendations. There remains nevertheless considerable investigations to be done such as applying the same kind of analysis as in Chapter 6 to tasks that violate the assumption of independent and identically distributed observations (ex: Reinforcement learning), or finding methods to account for variance more efficiently and more reliably.

Reproducibility goes beyond the question of reliability of comparison methods however. Not all progress can be captured by benchmarks. The latter are highly curated problems that only represent a tiny fraction of the reality we wish to interact with (Bolukbasi et al., 2016; Buolamwini & Gebru, 2018; Mitchell et al., 2019). In order to support reproducible scientific conclusions, there is a need for experimentations on a variety of datasets that are representative of the problem targeted by the conclusion. We cannot claim an algorithm to be better than other algorithms at recognizing digits in general and test on only one dataset. There is also a need for more ablation studies to support general claims on modifications affecting many parts of the learning pipeline. This goes back to the discussion about reproducibility from the standpoint of philosophy of science in Chapter 1: "*Falsifying a theory is what sheds light on the theory's weaknesses and where the researchers should investigate next*". Finding failure modes of otherwise successful learning algorithms on subsets of datasets or subsets of ablation studies should be one of the main driver of machine learning researchers. Anyhow, finding these failure modes with confidence requires a proper methodology that accounts for variance and handles confounding variables such as hyperparameters.

Failed reproductions are opportunities for further investigations and successful reproductions are a way to assert the robustness of our scientific conclusions.

For machine learning research to progress more rapidly, experiments should not be designed only to succeed, but also to fail in ways to provide further insight.

References

Abadi, Martín and al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`.

Akiba, Takuya, Sano, Shotaro, Yanase, Toshihiko, Ohta, Takeru, and Koyama, Masanori. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 2623–2631, 2019.

Allen, David M. The relationship between variable selection and data agumentation and a method for prediction. *technometrics*, 16(1):125–127, 1974.

Altman, Naomi S. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

Anders Sogaard, Anders Johannsen, Barbara Plank Dirk Hovy and Alonso, Hector Mart'inez. What's in a p-value in nlp? In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pp. 1–10. Association for Computational Linguistics, 2014.

Arango, Sebastian Pineda, Jomaa, Hadi S, Wistuba, Martin, and Grabocka, Josif. Hpo-b: A large-scale reproducible benchmark for black-box hpo based on openml. *arXiv preprint arXiv:2106.06257*, 2021.

Arpit, Devansh, Zhou, Yingbo, Kota, Bhargava, and Govindaraju, Venu. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In *International Conference on Machine Learning*, pp. 1168–1176, 2016.

Awad, Noor, Mallik, Neeratyoy, and Hutter, Frank. Dehb: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. *arXiv preprint arXiv:2105.09821*, 2021.

Ba, Jimmy Lei, Kiros, Jamie Ryan, and Hinton, Geoffrey E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. In Bengio, Yoshua and LeCun, Yann (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1409.`

0473.

Baker, Bowen, Gupta, Otkrist, Raskar, Ramesh, and Naik, Nikhil. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

Bakshy, Eytan, Dworkin, Lili, Karrer, Brian, Kashin, Konstantin, Letham, Benjamin, Murthy, Ashwin, and Singh, Shaun. Ae: A domain-agnostic platform for adaptive experimentation. In *Workshop on System for ML*, 2018.

Barba, Lorena A. Terminologies for reproducible research. *arXiv preprint arXiv:1802.03311*, 2018.

Bardenet, Rémi, Brendel, Mátyás, Kégl, Balázs, and Sebag, Michele. Collaborative hyperparameter tuning. In *International Conference on Machine Learning*, pp. 199–207, 2013.

Begley, C Glenn and Ellis, Lee M. Raise standards for preclinical cancer research. *Nature*, 483(7391):531–533, 2012.

Bengio, Yoshua, Simard, Patrice, and Frasconi, Paolo. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Bengio, Yoshua, Ducharme, Réjean, Vincent, Pascal, and Janvin, Christian. A neural probabilistic language model. *The journal of machine learning research*, 3:1137–1155, 2003.

Bennani-Smires, Kamil, Musat, Claudiu, Hossmann, Andreea, and Baeriswyl, Michael. Gitgraph-from computational subgraphs to smaller architecture search spaces. 2018.

Bentivogli, Luisa, Dagan, Ido, Dang, Hoa Trang, Giampiccolo, Danilo, and Magnini, Bernardo. The fifth PASCAL recognizing textual entailment challenge. 2009.

Bergstra, James and Bengio, Yoshua. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13:281–305, 2012. ISSN 1532-4435. doi: 10.1162/153244303322533223.

Bergstra, James, Bardenet, Rémi, Bengio, Yoshua, and Kégl, Balázs. Algorithms for Hyper-Parameter Optimization. *Advances in Neural Information Processing Systems (NIPS)*, pp. 2546–2554, 2011. ISSN 10495258. doi: 2012arXiv1206.2944S. URL https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf.

Bergstra, James, Yamins, Daniel, and Cox, David. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pp. 115–123. PMLR, 2013.

Bergstra, James, Komer, Brent, Eliasmith, Chris, Yamins, Dan, and Cox, David D. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015.

Biedenkapp, A., Marben, J., Lindauer, M., and Hutter, F. CAVE: Configuration assessment, visualization and evaluation. In *Proceedings of the International Conference on Learning and Intelligent Optimization (LION'18)*, 2018.

Bishop, Dorothy and Gill, Eoin. Robert boyle on the importance of reporting and replicating experiments. *Journal of the Royal Society of Medicine*, 113(2):79–83, 2020.

Blalock, Davis, Gonzalez Ortiz, Jose Javier, Frankle, Jonathan, and Guttag, John. What is the State of Neural Network Pruning? In *Proceedings of Machine Learning and Systems 2020*, pp. 129–146. 2020.

Bolukbasi, Tolga, Chang, Kai-Wei, Zou, James, Saligrama, Venkatesh, and Kalai, Adam. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. *arXiv preprint arXiv:1607.06520*, 2016.

Boser, Bernhard E, Guyon, Isabelle M, and Vapnik, Vladimir N. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, 1992.

Bouckaert, Remco R and Frank, Eibe. Evaluating the replicability of significance tests for comparing learning algorithms. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 3–12. Springer, 2004.

Bouthillier, Xavier and Varoquaux, Gaël. Survey of machine-learning experimental methods at NeurIPS2019 and ICLR2020. Research report, Inria Saclay Ile de France, January 2020. URL `https://hal.archives-ouvertes.fr/hal-02447823`.

Bouthillier, Xavier, Laurent, César, and Vincent, Pascal. Unreproducible research is reproducible. In Chaudhuri, Kamalika and Salakhutdinov, Ruslan (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 725–734, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL `http://proceedings.mlr.press/v97/bouthillier19a.html`.

Breiman, Leo. Bagging predictors. *Machine learning*, 24(2):123–140, 1996a.

Breiman, Leo. Out-of-bag estimation. 1996b.

Bühlmann, Peter, Yu, Bin, et al. Analyzing bagging. *The Annals of Statistics*, 30(4):927–961, 2002.

Buolamwini, Joy and Gebru, Timnit. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pp. 77–91. PMLR, 2018.

Cai, Han, Chen, Tianyao, Zhang, Weinan, Yu, Yong, and Wang, Jun. Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018a.

Cai, Han, Yang, Jiacheng, Zhang, Weinan, Han, Song, and Yu, Yong. Path-level network transformation for efficient architecture search. In *International Conference on Machine Learning*, pp. 678–687. PMLR, 2018b.

Camerer, Colin F, Dreber, Anna, Holzmeister, Felix, Ho, Teck-Hua, Huber, Jürgen, Johannesson, Magnus, Kirchler, Michael, Nave, Gideon, Nosek, Brian A, Pfeiffer, Thomas, et al. Evaluating the replicability of social science experiments in nature and science between

2010 and 2015. *Nature Human Behaviour*, 2(9):637–644, 2018.

Canty, Angelo J, Davison, Anthony C, Hinkley, David V, and Ventura, Valérie. Bootstrap diagnostics and remedies. *Canadian Journal of Statistics*, 34(1):5–27, 2006.

Card, Dallas, Henderson, Peter, Khandelwal, Urvashi, Jia, Robin, Mahowald, Kyle, and Jurafsky, Dan. With little power comes great responsibility. *arXiv preprint arXiv:2010.06595*, 2020.

Carnap, Rudolf. Testability and meaning. *Philosophy of science*, 3(4):419–471, 1936.

Chen, Stanley F, Beeferman, Douglas, and Rosenfeld, Roni. Evaluation metrics for language models. 1998.

Chevalier, Clément and Ginsbourger, David. Fast Computation of the Multi-points Expected Improvement with Applications in Batch Selection. working paper or preprint, October 2012. URL `https://hal.archives-ouvertes.fr/hal-00732512`.

Cho, Kyunghyun, van Merriënboer, Bart, Bahdanau, Dzmitry, and Bengio, Yoshua. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-4012. URL `https://www.aclweb.org/anthology/W14-4012`.

Chrabaszcz, Patryk, Loshchilov, Ilya, and Hutter, Frank. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Chung, Junyoung, Gulcehre, Caglar, Cho, Kyunghyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

Claerbout, Jon F and Karrenbach, Martin. Electronic documents give reproducible research a new meaning. In *SEG Technical Program Expanded Abstracts 1992*, pp. 601–604. Society of Exploration Geophysicists, 1992.

Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and van Schaik, André. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

Collaboration, Open Science et al. Estimating the reproducibility of psychological science. *Science*, 349(6251), 2015.

Collobert, Ronan, Weston, Jason, Bottou, Léon, Karlen, Michael, Kavukcuoglu, Koray, and Kuksa, Pavel. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011.

Cooijmans, Tim, Ballas, Nicolas, Laurent, César, Gülçehre, Çağlar, and Courville, Aaron. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

Cortes, Corinna and Vapnik, Vladimir. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

Cowen-Rivers, Alexander I, Lyu, Wenlong, Wang, Zhi, Tutunov, Rasul, Jianye, Hao, Wang, Jun, and Ammar, Haitham Bou. Hebo: Heteroscedastic evolutionary bayesian optimisation. *arXiv preprint arXiv:2012.03826*, 2020.

Cybenko, George. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Dacrema, Maurizio Ferrari, Cremonesi, Paolo, and Jannach, Dietmar. 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*, pp. 101–109, New York, New York, USA, 2019. ACM Press. ISBN 9781450362436. doi: 10.1145/3298689.3347058. URL `http://dl.acm.org/citation.cfm?doid=3298689.3347058`.

D'Amour, Alexander, Heller, Katherine, Moldovan, Dan, Adlam, Ben, Alipanahi, Babak, Beutel, Alex, Chen, Christina, Deaton, Jonathan, Eisenstein, Jacob, Hoffman, Matthew D, et al. Underspecification presents challenges for credibility in modern machine learning. *arXiv preprint arXiv:2011.03395*, 2020.

del Barrio, Eustasio, Cuesta-Albertos, Juan A, and Matrán, Carlos. An optimal transportation approach for assessing almost stochastic order. In *The Mathematics of the Uncertain*, pp. 33–44. Springer, 2018.

Demšar, Janez. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Devezer, Berna, Nardin, Luis G, Baumgaertner, Bert, and Buzbas, Erkan Ozge. Scientific discovery in a model-centric framework: Reproducibility, innovation, and epistemic diversity. *PloS one*, 14(5):e0216125, 2019.

Devezer, Berna, Navarro, Danielle J, Vandekerckhove, Joachim, and Ozge Buzbas, Erkan. The case for formal methodology in scientific reform. *Royal Society open science*, 8(3): 200805, 2020.

Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, and Toutanova, Kristina. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

Dewancker, Ian, McCourt, Michael, Clark, Scott, Hayes, Patrick, Johnson, Alexandra, and Ke, George. A strategy for ranking optimization methods using multiple criteria. In *Workshop on Automatic Machine Learning*, pp. 11–20, 2016.

Dietterich, Thomas G. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

Dodge, Jesse, Gururangan, Suchin, Card, Dallas, Schwartz, Roy, and Smith, Noah A. Show your work: Improved reporting of experimental results. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2185–2194,

Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1224. URL https://www.aclweb.org/anthology/D19-1224.

Dong, Xuanyi and Yang, Yi. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2019.

Dror, Rotem, Baumer, Gili, Bogomolov, Marina, and Reichart, Roi. Replicability analysis for natural language processing: Testing significance with multiple datasets. *Transactions of the Association for Computational Linguistics*, 5:471–486, 2017.

Dror, Rotem, Baumer, Gili, Shlomov, Segev, and Reichart, Roi. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1383–1392, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1128. URL https://www.aclweb.org/anthology/P18-1128.

Dror, Rotem, Shlomov, Segev, and Reichart, Roi. Deep dominance - how to properly compare deep neural models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2773–2785, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1266. URL https://www.aclweb.org/anthology/P19-1266.

Dudoit, Sandrine, Shaffer, Juliet Popper, and Boldrick, Jennifer C. Multiple hypothesis testing in microarray experiments. *Statistical Science*, pp. 71–103, 2003.

Dudy, Shiran and Bedrick, Steven. Are some words worth more than others? In *Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems*, pp. 131–142, 2020.

Efron, B. Bootstrap methods: Another look at the jackknife. *Ann. Statist.*, 7(1):1–26, 01 1979. doi: 10.1214/aos/1176344552. URL https://doi.org/10.1214/aos/1176344552.

Efron, Bradley. *The jackknife, the bootstrap, and other resampling plans*, volume 38. Siam, 1982.

Efron, Bradley. Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics*, pp. 569–593. Springer, 1992.

Efron, Bradley and Tibshirani, Robert J. *An introduction to the bootstrap*. CRC press, 1994.

Eggensperger, Katharina, Hutter, Frank, Hoos, Holger, and Leyton-Brown, Kevin. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

Eisner, DA. Reproducibility of science: Fraud, impact factors and carelessness. *Journal of molecular and cellular cardiology*, 114:364–368, 2018.

Elsken, Thomas, Metzen, Jan-Hendrik, and Hutter, Frank. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

Elsken, Thomas, Metzen, Jan Hendrik, and Hutter, Frank. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning*

Representations, 2019.

Englander, Irv and Wong, Wilson. *The architecture of computer hardware, systems software, and networking: An information technology approach*. John Wiley & Sons, 2021.

Engstrom, Logan, Ilyas, Andrew, Santurkar, Shibani, Tsipras, Dimitris, Steinhardt, Jacob, and Madry, Aleksander. Identifying statistical bias in dataset replication. In *International Conference on Machine Learning*, pp. 2922–2932. PMLR, 2020.

et al, Li. Cs231n: Convolutional neural networks for visual recognition, 2019. URL `http://cs231n.stanford.edu/`.

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

Falkner, Stefan, Klein, Aaron, and Hutter, Frank. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pp. 1437–1446. PMLR, 2018.

Feurer, Matthias, Springenberg, Jost Tobias, and Hutter, Frank. Initializing bayesian hyperparameter optimization via meta-learning. In *AAAI*, pp. 1128–1135, 2015.

Feyerabend, Paul. *Against method*. Verso, 1993.

Fisher, Ronald A. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222(594-604):309–368, 1922.

Fisher, Ronald A. Presidential address. *The Indian Journal of Statistics*, 4(1):14–17, 1938.

Fisher, Ronald Aylmer. *Statistical methods for research workers*. Oliver and Boyd, 1932.

Fisher, Ronald Aylmer. *Design of Experiments*, volume 1. British Medical Journal Publishing Group, 1936.

Forde, Jessica, Bussonnier, Matthias, Fortin, Félix-Antoine, Granger, Brian, Head, Tim, Holdgraf, Chris, Ivanov, Paul, Kelley, Kyle, Pacer, M, Panda, Yuvi, et al. Reproducing machine learning research on binder. 2018.

Fukushima, Kunihiko and Miyake, Sei. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pp. 267–285. Springer, 1982.

Fursin, Grigori. Enabling reproducible ML and Systems research: the good, the bad, and the ugly, August 2020. URL `https://doi.org/10.5281/zenodo.4005773`.

Garrido-Merchán, Eduardo C and Hernández-Lobato, Daniel. Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes. *Neurocomputing*, 380:20–35, 2020.

George, Johnu, Gao, Ce, Liu, Richard, Liu, Hou Gang, Tang, Yuan, Pydipaty, Ramdoot, and Saha, Amit Kumar. A scalable and cloud-native hyperparameter tuning system. *arXiv preprint arXiv:2006.02085*, 2020.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Godfrey-Smith, Peter. *Theory and reality*. University of Chicago Press, 2009.

Goodman, Steven and Greenland, Sander. Assessing the unreliability of the medical literature: A response to" why most published research findings are false". 2007.

Goodman, Steven N., Fanelli, Daniele, and Ioannidis, John P. A. What does research reproducibility mean? *Science Translational Medicine*, 8(341):341ps12–341ps12, 2016. ISSN 1946-6234. doi: 10.1126/scitranslmed.aaf5027.

Gorman, Kyle and Bedrick, Steven. We need to talk about standard splits. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2786–2791, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1267. URL `https://www.aclweb.org/anthology/P19-1267`.

Hansen, Nikolaus. The cma evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pp. 75–102. Springer, 2006.

Hansen, Nikolaus and Ostermeier, Andreas. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Henderson, Peter, Islam, Riashat, Bachman, Philip, Pineau, Joelle, Precup, Doina, and Meger, David. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Henikoff, Steven and Henikoff, Jorja G. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

Heusel, Martin, Ramsauer, Hubert, Unterthiner, Thomas, Nessler, Bernhard, and Hochreiter, Sepp. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Hill, Felix, Reichart, Roi, and Korhonen, Anna. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, 2015.

Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George E, Mohamed, Abdel-rahman, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

Hinton, Geoffrey E, Osindero, Simon, and Teh, Yee-Whye. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Hochreiter, Sepp. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hoerl, Arthur E and Kennard, Robert W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Hornik, Kurt. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

Hothorn, Torsten, Leisch, Friedrich, Zeileis, Achim, and Hornik, Kurt. The design and analysis of benchmark experiments. *Journal of Computational and Graphical Statistics*, 14(3):675–699, 2005.

Hu, Jennifer, Gauthier, Jon, Qian, Peng, Wilcox, Ethan, and Levy, Roger. A systematic assessment of syntactic generalization in neural language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 1725–1744, 2020.

Huang, Gao, Liu, Zhuang, Van Der Maaten, Laurens, and Weinberger, Kilian Q. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269. IEEE, 2017.

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

Hutter, Frank, Hoos, Holger H, and Leyton-Brown, Kevin. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pp. 507–523. Springer, 2011.

Hutter, Frank, Kotthoff, Lars, and Vanschoren, Joaquin (eds.). *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2019. In press, available at http://automl.org/book.

Ioannidis, John PA. Why most published research findings are false. *PLoS medicine*, 2(8): e124, 2005.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Islam, Riashat, Henderson, Peter, Gomrokchi, Maziar, and Precup, Doina. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

Ito, Akinori, Kohda, Masaki, and Ostendorf, Mari. A new metric for stochastic language model evaluation. In *Sixth European Conference on Speech Communication and Technology*, 1999.

Iyer, Rukmini, Ostendorf, Mari, and Meteer, Marie. Analyzing and predicting language model improvements. In *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, pp. 254–261. IEEE, 1997.

Jaderberg, Max, Dalibard, Valentin, Osindero, Simon, Czarnecki, Wojciech M, Donahue, Jeff, Razavi, Ali, Vinyals, Oriol, Green, Tim, Dunning, Iain, Simonyan, Karen, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Jager, Leah R and Leek, Jeffrey T. An estimate of the science-wise false discovery rate and application to the top medical literature. *Biostatistics*, 15(1):1–12, 2014.

Jiang, Yiding, Neyshabur, Behnam, Mobahi, Hossein, Krishnan, Dilip, and Bengio, Samy. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJgIPJBFvH`.

Jordan, MI. Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986. Technical report, California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, 1986.

Joy, Tinu Theckel, Rana, Santu, Gupta, Sunil Kumar, and Venkatesh, Svetha. Flexible transfer learning framework for bayesian optimisation. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 102–114. Springer, 2016.

Jurafsky, Daniel and Martin, James H. Speech and language processing (draft). *preparation [cited 2020 June 1] Available from: https://web. stanford. edu/˜ jurafsky/slp3*, 2018.

Jurtz, Vanessa, Paul, Sinu, Andreatta, Massimo, Marcatili, Paolo, Peters, Bjoern, and Nielsen, Morten. Netmhcpan-4.0: improved peptide–mhc class i interaction predictions integrating eluted ligand and peptide binding affinity data. *The Journal of Immunology*, 199(9):3360–3368, 2017.

Kadlec, Rudolf, Bajgar, Ondrej, and Kleindienst, Jan. Knowledge base completion: Baselines strike back. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pp. 69–74, 2017.

Kandasamy, Kirthevasan, Vysyaraju, Karun Raju, Neiswanger, Willie, Paria, Biswajit, Collins, Christopher R., Schneider, Jeff, Poczos, Barnabas, and Xing, Eric P. Tuning Hyperparameters without Grad Students: Scalable and Robust Bayesian Optimisation with Dragonfly. *arXiv preprint arXiv:1903.06694*, 2019.

Karnin, Zohar, Koren, Tomer, and Somekh, Oren. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*, pp. 1238–1246. PMLR,

2013.

Kim, Jungtaek, Kim, Saehoon, and Choi, Seungjin. Learning to transfer initializations for bayesian hyperparameter optimization. *ArXiv*, abs/1710.06219, 2017.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. URL `http://arxiv.org/abs/1412.6980`.

Kingma, Diederik P. and Welling, Max. Auto-encoding variational bayes. In Bengio, Yoshua and LeCun, Yann (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL `http://arxiv.org/abs/1312.6114`.

Klein, A., Falkner, S., Mansur, N., and Hutter, F. Robo: A flexible and robust bayesian optimization framework in python. In *NIPS 2017 Bayesian Optimization Workshop*, December 2017.

Klein, Aaron and Hutter, Frank. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*, 2019.

Klein, Aaron, Falkner, Stefan, Bartels, Simon, Hennig, Philipp, and Hutter, Frank. Fast bayesian optimization of machine learning hyperparameters on large datasets. *CoRR*, abs/1605.07079, 2016.

Klein, Richard A, Vianello, Michelangelo, Hasselman, Fred, Adams, Byron G, Adams Jr, Reginald B, Alper, Sinan, Aveyard, Mark, Axt, Jordan R, Babalola, Mayowa T, Bahník, Štěpán, et al. Many labs 2: Investigating variation in replicability across samples and settings. *Advances in Methods and Practices in Psychological Science*, 1(4):443–490, 2018.

Kluyver, Thomas, Ragan-Kelley, Benjamin, Pérez, Fernando, Granger, Brian, Bussonnier, Matthias, Frederic, Jonathan, Kelley, Kyle, Hamrick, Jessica, Grout, Jason, Corlay, Sylvain, Ivanov, Paul, Avila, Damián, Abdalla, Safia, and Willing, Carol. Jupyter notebooks – a publishing format for reproducible computational workflows. In Loizides, F. and Schmidt, B. (eds.), *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pp. 87 – 90. IOS Press, 2016.

Krizhevsky, Alex, Hinton, Geoffrey, et al. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25: 1097–1105, 2012.

Kuhn, Thomas S. *The Structure of Scientific Revolutions*. University of Chicago press, 1962.

Kuprieiev, Ruslan. Data version control: Git extension for data scientists – manage your code and data together, May 2018. URL `https://dataversioncontrol.com`.

Kurtzer, Gregory M., Sochat, Vanessa, and Bauer, Michael W. Singularity: Scientific containers for mobility of compute. *PLOS ONE*, 12(5):1–20, 05 2017. doi: 10.1371/journal.pone.0177459.

Lakatos, Imre. Falsification and the methodology of scientific research programmes. In *Can theories be refuted?*, pp. 205–259. Springer, 1976.

Lakatos, Imre and Musgrave, Alan. *Criticism and the growth of knowledge: Volume 4: Proceedings of the International Colloquium in the Philosophy of Science, London, 1965*. Cambridge University Press, 1970.

Langley, Pat. The changing science of machine learning, 2011.

LeCun, Yann, Boser, Bernhard, Denker, John S, Henderson, Donnie, Howard, Richard E, Hubbard, Wayne, and Jackel, Lawrence D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Leek, Jeff. A few things that would reduce stress around reproducibility/replicability in science, 2017. URL `https://simplystatistics.org/2017/11/21/rr-sress/`.

Lehmann, Erich Leo. *Elements of large-sample theory*. Springer Science & Business Media, 2004.

Li, Liam and Talwalkar, Ameet. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pp. 367–377. PMLR, 2020.

Li, Liam, Jamieson, Kevin, Rostamizadeh, Afshin, Gonina, Ekaterina, Ben-tzur, Jonathan, Hardt, Moritz, Recht, Benjamin, and Talwalkar, Ameet. A system for massively parallel hyperparameter tuning. In Dhillon, I., Papailiopoulos, D., and Sze, V. (eds.), *Proceedings of Machine Learning and Systems*, volume 2, pp. 230–246, 2020. URL `https://proceedings.mlsys.org/paper/2020/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf`.

Li, Lisha, Jamieson, Kevin, DeSalvo, Giulia, Rostamizadeh, Afshin, and Talwalkar, Ameet. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018a. URL `http://jmlr.org/papers/v18/16-558.html`.

Li, Zhao Lucis, Liang, Chieh-Jan Mike, He, Wenjia, Zhu, Lianjie, Dai, Wenjun, Jiang, Jin, and Sun, Guangzhong. Metis: Robustly tuning tail latencies of cloud systems. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pp. 981–992, 2018b.

Liaw, Richard, Liang, Eric, Nishihara, Robert, Moritz, Philipp, Gonzalez, Joseph E, and Stoica, Ion. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Marben, J., Müller, P., and Hutter, F. Boah: A tool suite for multi-fidelity bayesian optimization & analysis of hyperparameters. *arXiv:1908.06756 [cs.LG]*.

Lindauer, Marius and Hutter, Frank. Best practices for scientific research on neural architecture search. *Journal of Machine Learning Research*, 21(243):1–18, 2020.

Lindauer, Marius, Eggensperger, Katharina, Feurer, Matthias, Falkner, Stefan, Biedenkapp, André, and Hutter, Frank. Smac v3: Algorithm configuration in python. `https://github.com/automl/SMAC3`, 2017.

Lindsay, Robert K, Buchanan, Bruce G, Feigenbaum, Edward A, and Lederberg, Joshua. Dendral: a case study of the first expert system for scientific hypothesis formation. *Artificial intelligence*, 61(2):209–261, 1993.

Liu, Chenxi, Zoph, Barret, Neumann, Maxim, Shlens, Jonathon, Hua, Wei, Li, Li-Jia, Fei-Fei, Li, Yuille, Alan, Huang, Jonathan, and Murphy, Kevin. Progressive neural architecture search. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Long, Jonathan, Shelhamer, Evan, and Darrell, Trevor. Fully convolutional networks for semantic segmentation, 2014.

Lowe, David G. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pp. 1150–1157. Ieee, 1999.

Lucic, Mario, Kurach, Karol, Michalski, Marcin, Gelly, Sylvain, and Bousquet, Olivier. Are gans created equal? a large-scale study. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 700–709. Curran Associates, Inc., 2018.

Maddison, Chris J., Mnih, Andriy, and Teh, Yee Whye. The concrete distribution: A continuous relaxation of discrete random variables. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=S1jE5L5gl`.

Mahajan, Dhruv, Girshick, Ross, Ramanathan, Vignesh, He, Kaiming, Paluri, Manohar, Li, Yixuan, Bharambe, Ashwin, and van der Maaten, Laurens. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 181–196, 2018.

Mania, Horia, Miller, John, Schmidt, Ludwig, Hardt, Moritz, and Recht, Benjamin. Model similarity mitigates test set overuse. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/48237d9f2dea8c74c2a72126cf63d933-Paper.pdf`.

Mann, Henry B and Whitney, Donald R. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pp. 50–60, 1947.

Mantovani, Rafael Gomes, Horváth, Tomáš, Cerri, Ricardo, Junior, Sylvio Barbon, Vanschoren, Joaquin, and de Carvalho, André Carlos Ponce de Leon Ferreira. An empirical study on hyperparameter tuning of decision trees. *arXiv preprint arXiv:1812.02207*, 2018.

Matthias Feurer, Aaron Klein, Katharina Eggensperger Jost Springenberg Manuel Blum and Hutter, Frank. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28 (2015)*, pp. 2962–2970, 2015.

Matthias Feurer, Katharina Eggensperger, Stefan Falkner Marius Lindauer and Hutter, Frank. Auto-sklearn 2.0: The next generation. In *arXiv:2007.04074 [cs.LG]*, 2020.

Mayo, Deborah G. *Statistical inference as severe testing*. Cambridge: Cambridge University Press, 2018.

McShane, Blakeley B, Gal, David, Gelman, Andrew, Robert, Christian, and Tackett, Jennifer L. Abandon statistical significance. *The American Statistician*, 73(sup1):235–245, 2019.

Melis, Gábor, Dyer, Chris, and Blunsom, Phil. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*, 2018.

Mendels, Gideon and Lahav, Nimrod. Comet.ml - supercharging machine learning, May 2018. URL `https://www.comet.ml`.

Merkel, Dirk. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014. ISSN 1075-3583.

Minsky, Marvin and Papert, Seymour A. *Perceptrons: An introduction to computational geometry*. MIT press, 1969.

Mitchell, Margaret, Wu, Simone, Zaldivar, Andrew, Barnes, Parker, Vasserman, Lucy, Hutchinson, Ben, Spitzer, Elena, Raji, Inioluwa Deborah, and Gebru, Timnit. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pp. 220–229, 2019.

Močkus, Jonas. On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pp. 400–404. Springer, 1975.

Murphy, Kevin P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Musgrave, Kevin, Belongie, Serge, and Lim, Ser-Nam. A Metric Learning Reality Check. *arXiv*, 2020. URL `http://arxiv.org/abs/2003.08505`.

Nadeau, Claude and Bengio, Yoshua. Inference for the generalization error. In Solla, S. A., Leen, T. K., and Müller, K. (eds.), *Advances in Neural Information Processing Systems 12*, pp. 307–313. MIT Press, 2000.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

Narang, Sharan, Chung, Hyung Won, Tay, Yi, Fedus, William, Fevry, Thibault, Matena, Michael, Malkan, Karishma, Fiedel, Noah, Shazeer, Noam, Lan, Zhenzhong, et al. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*, 2021.

Navarro, Danielle J. Between the devil and the deep blue sea: Tensions between scientific judgement and statistical model selection. *Computational Brain & Behavior*, 2(1):28–34,

2019.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 5, 2011.

Newell, A. and Simon, H. The logic theory machine–a complex information processing system. *IRE Transactions on Information Theory*, 2(3):61–79, 1956. doi: 10.1109/TIT.1956.1056797.

Neyman, Jerzy and Pearson, Egon S. On the use and interpretation of certain test criteria for purposes of statistical inference: Part i. *Biometrika*, pp. 175–240, 1928.

Neyman, Jerzy and Pearson, Egon Sharpe. Ix. on the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 231(694-706):289–337, 1933.

Nielsen, Morten, Lundegaard, Claus, Blicher, Thomas, Lamberth, Kasper, Harndahl, Mikkel, Justesen, Sune, Røder, Gustav, Peters, Bjoern, Sette, Alessandro, Lund, Ole, et al. Netmhcpan, a method for quantitative predictions of peptide binding to any hla-a and-b locus protein of known sequence. *PloS one*, 2(8), 2007.

Noether, Gottfried E. Sample size determination for some common nonparametric tests. *Journal of the American Statistical Association*, 82(398):645–647, 1987.

Nosek, Brian A, Ebersole, Charles R, DeHaven, Alexander C, and Mellor, David T. The preregistration revolution. *Proceedings of the National Academy of Sciences*, 115(11): 2600–2606, 2018.

O'Donnell, Timothy J, Rubinsteyn, Alex, Bonsack, Maria, Riemer, Angelika B, Laserson, Uri, and Hammerbacher, Jeff. Mhcflurry: open-source class i mhc binding affinity prediction. *Cell systems*, 7(1):129–132, 2018.

Olson, Randal S., Bartley, Nathan, Urbanowicz, Ryan J., and Moore, Jason H. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pp. 485–492, New York, NY, USA, 2016a. ACM. ISBN 978-1-4503-4206-3. doi: 10.1145/2908812.2908918. URL `http://doi.acm.org/10.1145/2908812.2908918`.

Olson, Randal S., Urbanowicz, Ryan J., Andrews, Peter C., Lavender, Nicole A., Kidd, La Creis, and Moore, Jason H. *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I*, chapter Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pp. 123–137. Springer International Publishing, 2016b. ISBN 978-3-319-31204-0. doi: 10.1007/978-3-319-31204-0_9. URL `http://dx.doi.org/10.1007/978-3-319-31204-0_9`.

Parzen, Emanuel. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318. PMLR, 2013.

Paszke, Adam, Gross, Sam, Chintala, Soumith, Chanan, Gregory, Yang, Edward, DeVito, Zachary, Lin, Zeming, Desmaison, Alban, Antiga, Luca, and Lerer, Adam. Automatic differentiation in pytorch. 2017.

Pearl, Judea and Mackenzie, Dana. *The book of why: the new science of cause and effect*. Basic books, 2018.

Pearson, Hillary, Daouda, Tariq, Granados, Diana Paola, Durette, Chantal, Bonneil, Eric, Courcelles, Mathieu, Rodenbrock, Anja, Laverdure, Jean-Philippe, Côté, Caroline, Mader, Sylvie, et al. Mhc class i–associated peptides derive from selective regions of the human genome. *The Journal of clinical investigation*, 126(12):4690–4701, 2016.

Perezgonzalez, Jose D. Fisher, neyman-pearson or nhst? a tutorial for teaching data testing. *Frontiers in Psychology*, 6:223, 2015.

Perme, Maja Pohar and Manevski, Damjan. Confidence intervals for the mann–whitney test. *Statistical methods in medical research*, 28(12):3755–3768, 2019.

Perrone, Valerio, Jenatton, Rodolphe, Seeger, Matthias, and Archambeau, Cédric. Scalable hyperparameter transfer learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 6846–6856, 2018.

Pfisterer, Florian, van Rijn, Jan N, Probst, Philipp, Müller, Andreas, and Bischl, Bernd. Learning multiple defaults for machine learning algorithms. *arXiv preprint arXiv:1811.09409*, 2018.

Pineau, Joelle, Vincent-Lamarre, Philippe, Sinha, Koustuv, Larivière, Vincent, Beygelzimer, Alina, d'Alché Buc, Florence, Fox, Emily, and Larochelle, Hugo. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *arXiv preprint arXiv:2003.12206*, 2020.

Plesser, Hans E. Reproducibility vs. replicability: a brief history of a confused terminology. *Frontiers in neuroinformatics*, 11:76, 2018.

Polyak, Boris T. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

Popper, Karl. Logik der forschung. *Philosophy and Rhetoric of Science*, pp. 51, 1934.

Popper, Karl. *The logic of scientific discovery*. Routledge, 2005.

Prabhu, Vinay Uday and Birhane, Abeba. Large image datasets: A pyrrhic win for computer vision? *arXiv preprint arXiv:2006.16923*, 2020.

Probst, Philipp, Boulesteix, Anne-Laure, and Bischl, Bernd. Tunability: Importance of hyperparameters of machine learning algorithms. *J. Mach. Learn. Res.*, 20(53):1–32, 2019.

Raff, Edward. A Step Toward Quantifying Independently Reproducible Machine Learning Research. In *NeurIPS*, 2019. URL http://arxiv.org/abs/1909.06674.

Raff, Edward. Research Reproducibility as a Survival Analysis. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2021. URL http://arxiv.org/abs/2012.09932.

Rapin, J. and Teytaud, O. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad, 2018.

Rasmussen, Carl Edward and Williams, Christopher K. I. *Gaussian Processes for Machine Learning*. The MIT Press, 2005. ISBN 026218253X.

Real, Esteban, Aggarwal, Alok, Huang, Yanping, and Le, Quoc V. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Recht, Benjamin, Roelofs, Rebecca, Schmidt, Ludwig, and Shankar, Vaishaal. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018.

Recht, Benjamin, Roelofs, Rebecca, Schmidt, Ludwig, and Shankar, Vaishaal. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pp. 5389–5400. PMLR, 2019.

Reddi, Vijay Janapa, Cheng, Christine, Kanter, David, Mattson, Peter, Schmuelling, Guenther, Wu, Carole-Jean, Anderson, Brian, Breughe, Maximilien, Charlebois, Mark, Chou, William, et al. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 446–459. IEEE, 2020.

Reimers, Nils and Gurevych, Iryna. 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*, pp. 338–348, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1035. URL https://www.aclweb.org/anthology/D17-1035.

Reimers, Nils and Gurevych, Iryna. Why comparing single performance scores does not allow to draw conclusions about machine learning approaches. *arXiv preprint arXiv:1803.09578*, 2018.

Riezler, Stefan and Maxwell, John T. On some pitfalls in automatic evaluation and significance testing for mt. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pp. 57–64, 2005.

Roelofs, Rebecca, Shankar, Vaishaal, Recht, Benjamin, Fridovich-Keil, Sara, Hardt, Moritz, Miller, John, and Schmidt, Ludwig. A meta-analysis of overfitting in machine learning. *Advances in Neural Information Processing Systems*, 32:9179–9189, 2019.

Rosenblatt, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Rosenblatt, M. Remarks on some nonparametric estimates of a density function. annals of mathematical statistics. 1956.

Rosenthal, Robert. The file drawer problem and tolerance for null results. *Psychological bulletin*, 86(3):638, 1979.

Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Rutishauser, Heinz. Theory of gradient methods. In *Refined iterative methods for computation of the solution and the eigenvalues of self-adjoint boundary value problems*, pp. 24–49. Springer, 1959.

Salimans, Tim and Kingma, Durk P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 901–909, 2016.

Salimans, Tim, Goodfellow, Ian, Zaremba, Wojciech, Cheung, Vicki, Radford, Alec, and Chen, Xi. Improved techniques for training gans. *Advances in neural information processing systems*, 29:2234–2242, 2016.

Sampat, Anand and Shabaz, Patel. datmo: The easiest way to manage your models for data science, May 2018. URL `https://datmo.com`.

Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959. doi: 10.1147/rd.33.0210.

Sandler, Mark, Howard, Andrew, Zhu, Menglong, Zhmoginov, Andrey, and Chen, Liang-Chieh. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Santurkar, Shibani, Tsipras, Dimitris, Ilyas, Andrew, and Madry, Aleksander. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pp. 2483–2493, 2018.

Schmidt, Robin M, Schneider, Frank, and Hennig, Philipp. Descending through a crowded valley–benchmarking deep learning optimizers. *arXiv preprint arXiv:2007.01547*, 2020.

Schneider, Frank, Balles, Lukas, and Hennig, Philipp. Deepobs: A deep learning optimizer benchmark suite. *arXiv preprint arXiv:1903.05499*, 2019.

Sculley, D., Snoek, Jasper, Wiltschko, Alex, and Rahimi, Ali. Winner's curse? on pace, progress, and empirical rigor, 2018. URL `https://openreview.net/forum?id=rJWF0Fywf`.

Shahriari, Bobak, Swersky, Kevin, Wang, Ziyu, Adams, Ryan P, and De Freitas, Nando. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

Shapin, Steven and Schaffer, Simon. *Leviathan and the air-pump*. Princeton University Press, 1985.

Silver, David, Schrittwieser, Julian, Simonyan, Karen, Antonoglou, Ioannis, Huang, Aja, Guez, Arthur, Hubert, Thomas, Baker, Lucas, Lai, Matthew, Bolton, Adrian, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Sivaprasad, Prabhu Teja, Mai, Florian, Vogels, Thijs, Jaggi, Martin, and Fleuret, François. Optimizer benchmarking needs to account for hyperparameter tuning. In *International Conference on Machine Learning*, pp. 9036–9045. PMLR, 2020.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

Snoek, Jasper, Rippel, Oren, Swersky, Kevin, Kiros, Ryan, Satish, Nadathur, Sundaram, Narayanan, Patwary, Mostofa, Prabhat, Mr, and Adams, Ryan. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pp. 2171–2180. PMLR, 2015.

So, David, Le, Quoc, and Liang, Chen. The evolved transformer. In *International Conference on Machine Learning*, pp. 5877–5886. PMLR, 2019.

Socher, Richard, Perelygin, Alex, Wu, Jean, Chuang, Jason, Manning, Christopher D, Ng, Andrew, and Potts, Christopher. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, pp. 1631–1642, 2013.

Springenberg, Jost Tobias, Klein, Aaron, Falkner, Stefan, and Hutter, Frank. Bayesian optimization with robust bayesian neural networks. *Advances in neural information processing systems*, 29:4134–4142, 2016.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Stone, Mervyn. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):111–133, 1974.

Such, Felipe Petroski, Rawal, Aditya, Lehman, Joel, Stanley, Kenneth, and Clune, Jeffrey. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. In *International Conference on Machine Learning*, pp. 9206–9216. PMLR, 2020.

Sutskever, Ilya, Martens, James, Dahl, George, and Hinton, Geoffrey. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147. PMLR, 2013.

Taylor Berg-Kirkpatrick, David Burkett and Klein, Dan. An empirical investigation of statistical significance in nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 995–1005. Association for Computational Linguistics, 2012.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

Torralba, Antonio, Fergus, Rob, and Freeman, William T. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

Tripuraneni, Nilesh, Jordan, Michael, and Jin, Chi. On the theory of transfer learning: The importance of task diversity. *Advances in Neural Information Processing Systems*, 33, 2020.

Van Rijn, Jan N and Hutter, Frank. Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2367–2376, 2018.

Vapnik, Vladimir. *The nature of statistical learning theory*. Springer science & business media, 1998.

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Ł ukasz, and Polosukhin, Illia. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Vaux, David L, Fidler, Fiona, and Cumming, Geoff. Replicates and repeats—what is the difference and is it significant?: A brief discussion of statistics and experimental design. *EMBO reports*, 13(4):291–296, 2012.

Vita, Randi, Mahajan, Swapnil, Overton, James A, Dhanda, Sandeep Kumar, Martini, Sheridan, Cantrell, Jason R, Wheeler, Daniel K, Sette, Alessandro, and Peters, Bjoern. The immune epitope database (iedb): 2018 update. *Nucleic acids research*, 47(D1):D339–D343, 2019.

Wang, Alex, Singh, Amanpreet, Michael, Julian, Hill, Felix, Levy, Omer, and Bowman, Samuel R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *EMNLP 2018*, pp. 353, 2018.

Wang, Alex, Pruksachatkun, Yada, Nangia, Nikita, Singh, Amanpreet, Michael, Julian, Hill, Felix, Levy, Omer, and Bowman, Samuel R. Superglue: a stickier benchmark for general-purpose language understanding systems. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 3266–3280, 2019.

Wei, Tao, Wang, Changhu, Rui, Yong, and Chen, Chang Wen. Network morphism. In *International Conference on Machine Learning*, pp. 564–572. PMLR, 2016.

Wolf, Thomas, Debut, Lysandre, Sanh, Victor, Chaumond, Julien, Delangue, Clement, Moi, Anthony, Cistac, Pierric, Rault, Tim, Louf, R'emi, Funtowicz, Morgan, and Brew, Jamie. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*,

abs/1910.03771, 2019.

Wolpert, David H, Macready, William G, et al. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Xie, Qizhe, Hovy, Eduard, Luong, Minh-Thang, and Le, Quoc V. Self-training with noisy student improves imagenet classification. *arXiv preprint arXiv:1911.04252*, 2019.

Ying, Chris, Klein, Aaron, Christiansen, Eric, Real, Esteban, Murphy, Kevin, and Hutter, Frank. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pp. 7105–7114. PMLR, 2019.

Yogatama, Dani and Mann, Gideon. Efficient transfer learning method for automatic hyperparameter tuning. In *Artificial Intelligence and Statistics*, pp. 1077–1085, 2014.

Zela, Arber, Klein, Aaron, Falkner, Stefan, and Hutter, Frank. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv preprint arXiv:1807.06906*, 2018.

Zhang, Chiyuan, Bengio, Samy, Hardt, Moritz, Recht, Benjamin, and Vinyals, Oriol. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Zoph, Barret, Vasudevan, Vijay, Shlens, Jonathon, and Le, Quoc V. Learning transferable architectures for scalable image recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8697–8710, 2018. doi: 10.1109/CVPR.2018.00907.

Appendix A

Survey of machine-learning experimental methods at NeurIPS2019 and ICLR2020

A.1. Introduction

Experiments play an increasing role in machine learning research. The prototypical experimental paradigm is to measure the generalization error of a model on a benchmark dataset, and, often, to compare it to baseline models. Yet, trustworthy experimentation is difficult: thorough benchmarking has shown that classic baselines outperform later work initially reported as improvement for knowledge base completion (Kadlec et al., 2017), neural language models (Melis et al., 2018), or generative adversarial networks (Lucic et al., 2018). Empirical results are affected by arbitrary factors such as the random seeds of neural-network fitting procedures (Bouthillier et al., 2019) or non-deterministic benchmarking environments in reinforcement learning (Henderson et al., 2018). Computational budget matters for a fair comparison (Lucic et al., 2018; Melis et al., 2018), in particular for selection of model hyperparameter (Dodge et al., 2019).

The growing recognition of these challenges in machine learning research has motivated research in better procedures for experimentation and benchmarking: better hyperparameter tuning (Bergstra & Bengio, 2012; Hansen, 2006; Hutter et al., 2019; Li et al., 2018a; Snoek et al., 2012), controlled statistical testing of model performance (Bouckaert & Frank, 2004; Demšar, 2006; Dietterich, 1998; Hothorn et al., 2005). To reach best the community, ideally these developments should fit as well as possible the current practices.

Here, we present a survey of experimental procedures currently used by practitioners at two of the leading conferences, NeurIPS2019 and ICLR2020. The survey was conducted by asking simple anonymous questions to the corresponding authors of the papers published at these venues. We detail the results of the survey and provide a light analysis[1].

[1]Anonymous data is available at github.com/bouthilx/ml-survey-2020

Highlights. A vast majority of empirical works optimize model hyper-parameters, thought almost half of these use manual tuning and most of the automatic hyper-parameter optimization is done with grid search. The typical number of hyper-parameter set is in interval 3-5, and less than 50 model fits are used to explore the search space. In addition, most works also optimized their baselines (typically, around 4 baselines). Finally, studies typically reported 4 results per model per task to provide a measure of variance.

A.2. Survey methodology

The objective of the survey is to provide a portrait of the current common practices for experimental design in the machine learning community. We selected papers at the peer-reviewed conferences NeurIPS 2019 and ICLR 2020 as a representative sample of practices considered valid by the reviewers.

We modeled our questions to capture benchmarking methods, the predominant experimental procedure to measure the performance of a new algorithm. The survey is limited to 10 short and simple questions, all multiple choices. This was important to favor high response rates from the researchers, and should provide valuable information nevertheless considering the lack of documentation on the topic. The survey was also anonymous to favor high response rates.

First question serves as a filter of empirical and theoretical papers, second one as a filter for questions on hyperparameter optimization. The third, fourth and fifth questions measure the popularity of methods as well as the search space in terms of dimensionality and exploration. Sixth question measures comparability of models in benchmarks, while seventh and eight ones measure the number of comparisons and variety of benchmarks. Finally ninth question quantifies sample size upon which conclusions are made and tenth question measures reproducibility of the papers with sample size 1.

For NeurIPS, all author names, paper title and PDF were collected from the pre-proceeding webpage[2]. The emails were then automatically collected from the PDFs, with manual intervention when required. For ICLR, all author names, paper title and emails were collected using the official OpenReview library[3].

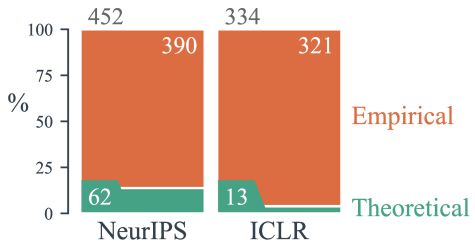
The survey for NeurIPS was sent on Thursday, 12th December, during the conference. A second email to remind authors about the survey was sent on Wednesday, 18th December. The response rate promptly raised from 30% to 34% after the reminder. The survey for ICLR was sent on Monday, 23th December, 4 days after the paper decision notifications. Considering the high response rate, no second email was sent to remind authors of ICLR.

[2]https://papers.nips.cc/book/advances-in-neural-information-processing-systems-32-2019
[3]https://github.com/openreview/openreview-py

A.3. Results

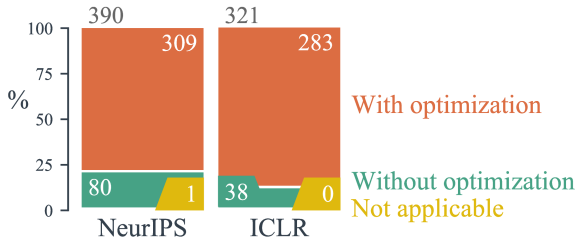
The questionnaire was sent to all first authors or corresponding authors of the accepted papers at NeurIPS2019 and ICLR2020. The response rates were 35.6% $\left(\frac{452}{1269}\right)$ and 48.6% $\left(\frac{334}{687}\right)$ for NeurIPS2019 and ICLR2020 respectively. On each figure below, the total number of answers is given at the top, while the number of replies for a given answer is given in the bars. The confidence intervals for all results are below 5% for a confidence level of 95%.

Question 1)
Did you have any experiments in your paper? If no, you are already done with the survey, thank you and have a good day. :)



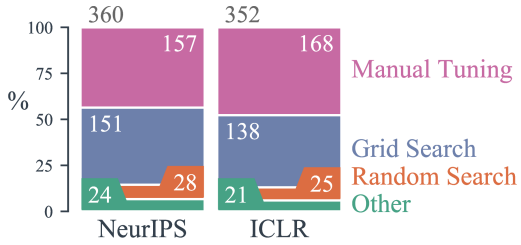
Question 2)
Did you optimize your hyperparameters?

Results are for empirical papers only.



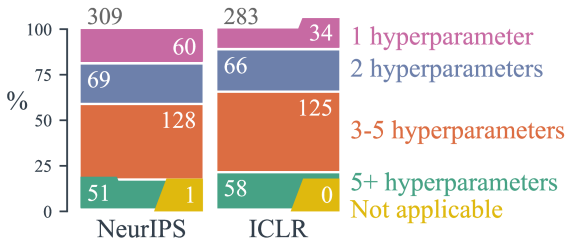
Question 3)
If yes, how did you tune them?

Results are for empirical papers with optimization only. Papers may use more than one method, hence it sums to more than the number of empirical papers.



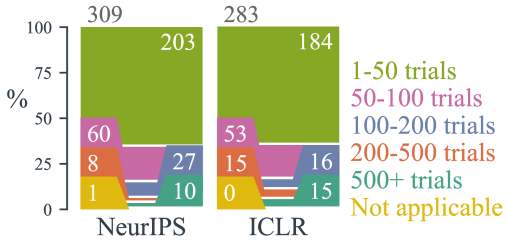
Question 4)
How many hyperparameters did you optimize?

Results are for empirical papers with optimization only.



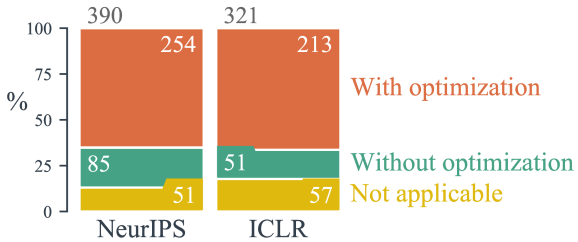
Question 5)
How many trials/experiments in total during the optimization? (How many different set of hyperparameters were evaluated)

Results are for empirical papers with optimization only.



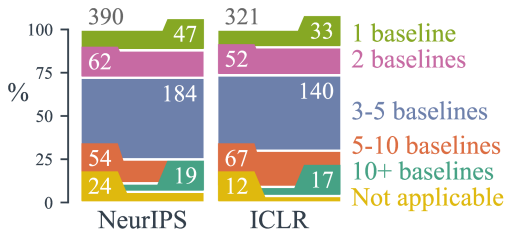
Question 6)
Did you optimize the hyperparameters of your baselines? (The other models or algorithms you compared with)

Results are for empirical papers only.



Question 7)
How many baselines (models, algos) did you compare with? (If different across datasets, please report maximum number, not total)

Results are for empirical papers only.

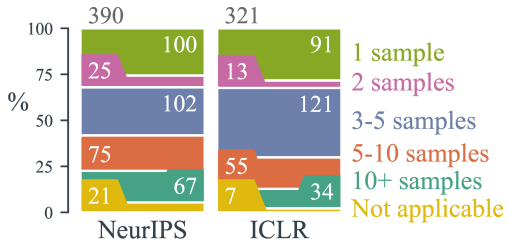


Question 8)
How many datasets or tasks did you compare on?

Results are for empirical papers only.

Question 9)
How many results did you report for each model (ex: for different seeds)

Results are for empirical papers only.



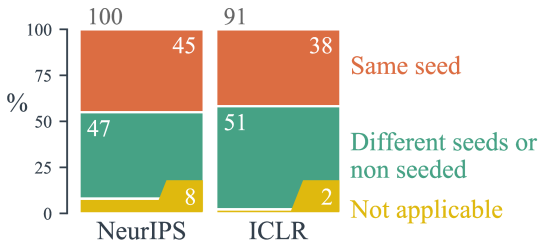
A.4. Discussion

A.4.1. Analysis

We first note the results are similar for NeurIPS and ICLR across all questions. One of the few significant differences being the larger proportion of theoretical papers at NeurIPS with approximately 10% more.

Question 10)
If you answered 1 to previous question, did you use the same seed in all your experiments? (If you did not seed your experiments, you should answer 'no' to this question.)

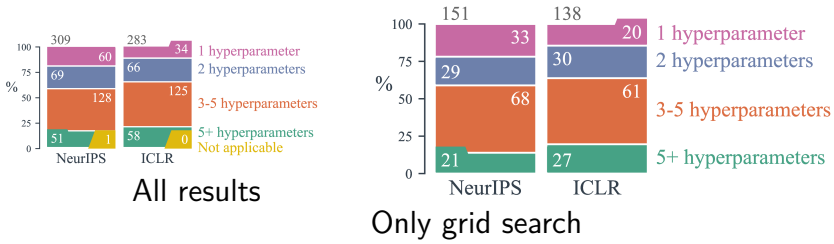
Results are for empirical papers with 1 result per model only.



A.4.1.1. Hyper-parameter tuning

The most popular tuning approach is manually, but followed closely by grid search (a 1.6% difference between the two approaches at NeurIPS and 8.5% at ICLR). Together they account for more than 85% hyperparameter procedures in both conferences. The number of hyperparameters is mostly in the interval 3 to 5. The proportions are preserved if we look specifically at the answers of papers using grid search.

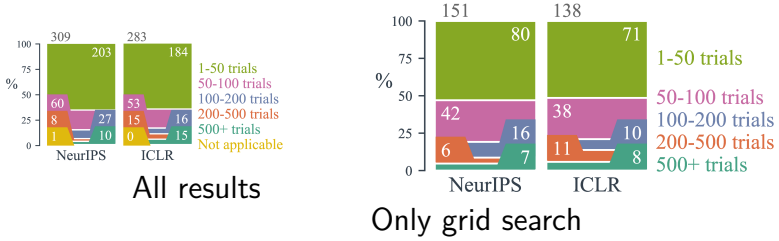
Fig. A.1. Results of question 4), **number of hyperparameters optimized**. All answers on the left, grid search only on the right.



In more than 50% of the papers, the hyperparameter search was executed on less than 50 trials. Again, the proportions are preserved if we look specifically at the answers of papers using grid search.

We now focus on grid search results because it is possible to infer the number of different values for each hyperparameters which were evaluated in the papers. Using the lower bound of the number of hyperparameters (n_h) and upper bound of number of trials (n_t) reported in the survey, we can estimate an upper bound on the number of different values (n_v) evaluated

Fig. A.2. Results of question 5), **number of trials in total during hyper-parameter optimization**. All answers on the left, grid search only on the right.



in the corresponding papers with $\exp\left(\frac{\log n_t}{n_h}\right)$, given that using grid search produces the number of trials $n_t = n_v^{n_h}$. We illustrate these results in Figure A.3. For more than 50% of the papers in which grid search was used, 6 or less different values were evaluated. It is important to understand that this is an upper bound, computed using n_h = 1,2,3,6 for answers 1, 2, 3–5 and 5+ to question 4) and n_t = 50,100,200,500,1000 for answers to question 4). The true average number of values used must therefore be lower. As an example, changing only n_h from 3 to 4 for answer 3–5 reduces the red bar of 50% to 4 values or less.

This low number of points reveals the importance of the selection of the grid. As shown by Bergstra & Bengio (2012), grid search's performance is ensured only if the region of optimal hyperparameters is large with respect to the grid's granularity. As such, in order to ensure performance of a coarse grid search of 6 values, it would require either that the hyperparameters are optimal on a wide region, or that the prior knowledge of experimenter made it possible to design a coarse grid precisely positioned on region of optimality. In either case, hyperparameter optimization could be considered rather pointless. Considering the accumulating evidence suggesting the opposite (Lucic et al., 2018; Melis et al., 2018; Dodge et al., 2019), we instead speculate that the present statistics suggest under-performing hyperparameter optimization procedures. This represents more than 20% of the empirical papers for NeurIPS and 19% for ICLR.

One shortcoming of the survey is the lack of information on the type of model used. Consequently, we cannot know if these numbers applies similarly to models that are computationally cheap or expensive. We would expect however that large neural networks are still more popular at ICLR than NeurIPS based on the history of these conferences. We thus believe the relative similarity of the results between the conferences to be an indication that the proportions reported here would apply to both papers on cheap and expensive algorithms.

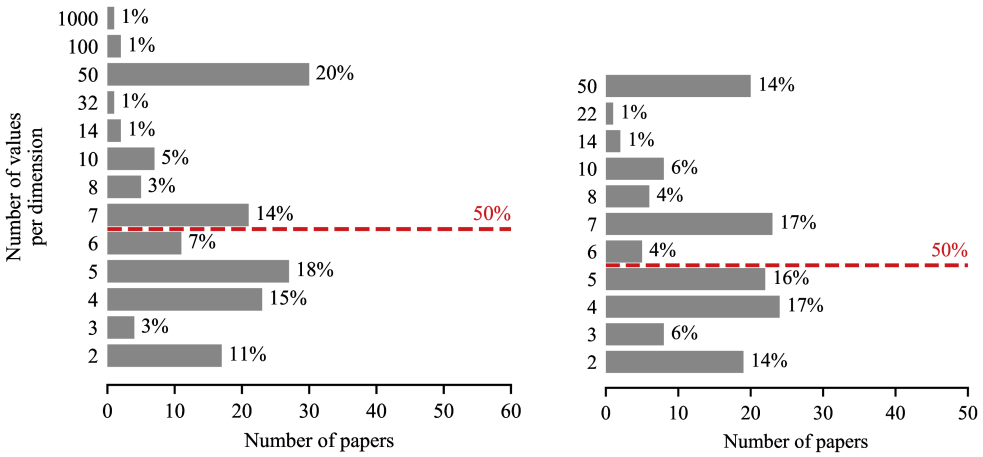


Fig. A.3. Number of different values for each hyperparameter

A.4.1.2. Baselines

The majority of authors of empirical papers reported optimizing their baselines' hyperparameters. This strikes us as against our personal knowledge of recent machine learning literature. Our hypothesis is that many authors answered 'Yes' for reporting results of other papers in which the hyperparameters were optimized. This would be technically true and the question should have been formulated differently to avoid this possible confusion. The goal of this question was to measure to which extend baselines are comparable, that is, using the same experiment design as the contributed algorithm. Both the number of baselines and datasets most importantly falls in the 3 to 5 interval. We expected these numbers to be lower and suspect that it may a be bias caused by the reviewing process of the conferences we selected.

A.4.1.3. Statistical power

The most common number of reported results per model are between 1 and 5 inclusively, 26% and 37% of which falls in the interval (3, 5) for NeurIPS and ICLR respectively while 26% and 25% reported 1 result. About half of the papers with only one result per model were seeded. Statistical tests are rarely used in recent machine learning literature, nevertheless we believe it is insightful to consider what statistical power or what size effect could be detected with the sample sizes reported in this survey.

Using Neyman-Person approach for statistical testing and controlling for both error type I and error type II at 0.05, we can compute the minimal difference of performance that can be detected between two algorithms. For simplicity, lets consider a classification task in which a model have 90% accuracy on a test set of 10k samples. We can estimate the variance of the performance measure due the sampling of the test set by assuming the measure follows a

binomial distribution[4]. For a sample size of 1, the minimal difference is about 1% accuracy, while it is 0.45% for sample size 5, and 0.31% for sample size 10. As 50% of sample sizes reported in the survey are below 6, the community should be concerned of benchmarks where many algorithms compared have similar performances, and make sure a sufficient sample size is used.

A.4.1.4. Opening questions on hyperparameter optimization

Most studies use grid search or random search for hyperparameter optimization. This lack of diversity raises the question of why more sophisticated hyperparameter optimization methods are not used. Are researchers lacking trust in these more sophisticated methods? Are they lacking computational power and therefore favor manual procedures? Are the available codebases considered to complicated to use?

A.4.2. Shortcomings of the survey

A.4.2.1. Sampling bias

All samples have an inherent bias. For this survey, the bias comes both from the population targeted –papers selected at two of the leading conferences, NeurIPS2019 and ICLR2020–, has well as self-selection bias in the responses: respectively 35.6% and 48.6% of researcher surveyed replied. This bias may result in sampling more researchers sensitive to the question of experimental design. As such, the bias could be considered towards better practices, implying that actual practices could be slightly less systematic then our results imply. The survey was anonymous which made it impossible to limit number of answers per paper. Consequently, unmeasured duplicate answers could also have contributed to a bias. Comparison of our results with the statistics of the reproducibility checklist at NeurIPS could serve as an estimate of these biases.

A.4.2.2. Choice of questions

Results of the survey are aggregated over all empirical papers. The implications of methodologies vary significantly however for different type of tasks or different kind of models. We failed to include a question which would allow such dichotomy in the analysis. We will discuss this further in next section.

As we were collecting the answers of the survey, researchers reported some confusion on several questions.

Question 2): Many papers were not using benchmarks but rather ablation studies as their experimental paradigm. We intended questions 2-5 as measures of the exploration of the hyperparameter space and its effect beyond simple optimization. Nevertheless,

[4]Which is approximately valid unless performance is very close to 100%

some authors reported answering *not applicable* when using ablation studies. An optional textual answer for *not applicable* would have been very informative.

Question 4): There was an overlap between the intervals (1, 50), (50, 100), (100, 200) and (200-500).

Question 6): As reported by one of the respondent, the question did not ask specifically about using the same hyperparameter optimization procedure. This likely confused many authors leading them to answer *yes* if their baselines where optimized results from other papers.

Question 7): Some authors told us they answered *not applicable* because they were the first to tackle a new problem. There is generally a straw man solution that can be used as a baseline in such situation and failing to compare with it should have been considered as 0 baseline instead of *not applicable*. There was however no option '0' in the multiple choices.

Question 8): There was an overlap between intervals (3, 5) and (5, 10).

Question 9): There was an overlap between intervals (3, 5) and (5, 10).

Question 9-10): Based on the feedback received and textual answers, questions 9) and 10) were often mis-understood as seeding of model initialization in particular. The question was rather about seeding of any possible source of variation in the experiments. As a result, several authors reported to us answering *not applicable* because model initialisation was not a source of variation in their experiments. Additionally, the question did not ask specifically *per model per task*. This could have led some authors to answer number of results reported over all tasks, therefore introducing a bias upwards.

Despite these limitations, we believe current trends can be identified as we highlighted on the experimentation methodology of researchers in machine learning. Some differences are indeed large enough to be considered significant.

A.4.3. Categorization of answers for question 10

Some authors gave textual answers to question 10. We received more textual answers than the ones reported here, but we only considered those of empirical papers for which authors answered '1' to question 9). Hence, we ignored the others. There is only 7 textual answers out of the 100 and 91 answers for question 10). The goal of question 10) was to measure to which extend papers with single results per model were reproducible. If an answer does not fit the typical *Same seed* but satisfies reproducibility, we categorize it as *Same seed*. Otherwise we categorize it as *Different seeds or non seeded*. The rationale for each decisions are described below.

> *"I reported average score for each model, averaged over 10 runs. Each of the 10 runs used the same seed across all models. "*

Category:: Same seed

Rationale:: The average estimation was seeded and thus reproducible.

> *"Same seed, but estimated the variance in previous works to be small. "*

Category:: Same seed

Rationale:: Used the same seed.

> *"not applicable. there's no seed because we use pre-trained models "*

Category:: Same seed

Rationale:: Reusing the same pre-trained model has the same effect as using the same seed to train a model.

> *"Seeded the data examples, did not seed the simulations (where however enough Monte Carlo replicates were conducted to make noise negligible) "*

Category:: Same seed

Rationale:: Assuming the noise due to simulation was indeed negligible enough, we assume seeding the sampling of data samples is enough to make the experiments reproducible.

> *"Our paper did not involve training deep networks, and there were not any mysterious hyperparameters.... "*

Category:: Same seed

Rationale:: Because the algorithm was apparently fully deterministic in a deterministic environment.

> *"Checked that results were consistent with different seeds but no proper study of variance "*

Category:: Different seeds or non seeded

Rationale:: Because different seeds were evaluated but results for only one of them was reported.

> *"Choice of seed has no effect on training outcome "*

Category:: Same seed

Rationale:: Assuming the authors are right and seeding has negligible effect in their experiments.

A.5. Conclusions

For reproducibility and AutoML, there is active research in benchmarking and hyperparameter procedures in machine learning. We hope that the survey results presented here can help inform this research. As this document is merely a research report, we purposely limited interpretation of the results and drawing recommendations. However, trends that stand out

to our eyes are, *1)* the simplicity of hyper-parameter tuning strategies (mostly manual search and grid search), *2)* the small number of model fits explored during this tuning (often 50 or less), which biases the results and *3)* the small number of performances reported, which limits statistical power. These practices are most likely due to the high computational cost of fitting modern machine-learning models.

Acknowledgments

We are deeply grateful to the participants of the survey who took time to answer the questions.

Appendix B

Accounting for Variance in Machine Learning Benchmarks

B.1. Notes on reproducibility

Ensuring full reproducibility is often a tedious work. We provide here notes and remarks on the issues we encountered while working towards fully reproducible experiments.
The testing procedure. To ensure proper study of the sources of variation it was necessary to control them close to perfection. For all tasks, we ran a pipeline of tests to ensure perfect reproducibility at execution and also at resumption. During the tests, each source of variation was varied with 5 different seeds, each executed 5 times. This ensured that the pipeline was reproducible for different seeds. Additionally, for each source of variation and for each seed, another training was executed but automatically interrupted after each epoch. The worker would then start the training of the next seed and iterate through the trainings for all seeds before resuming the first one. All these tests uncovered many bugs and typical reproducibility issues in machine learning. We report here some notes.
Computer architecture & drivers. Although we did not measure the variance induced by different GPU architectures, we did observe that different GPU models would lead to different results. The CPU model had less impact on the Deep Learning tasks but the MLP-MHC task was sensitive to it. We therefore limited all tasks to specific computer architectures. We also observed issues when CUDA drivers were updated during preliminary experiments. We ensured all experiments were run using CUDA 10.2.
Software & seeds. PyTorch versions lead to different results as well. We ran every Deep Learning experiments with PyTorch 1.2.0.

We implemented our data pipeline so that we could seed the iterators, the data augmentation objects and the splitting of the datasets. We had less control at the level of the models however. For PyTorch 1.2.0, the random number generator (RNG) must be seeded globally which makes it difficult to seed different parts separately. We seeded PyTorch's global

RNG for weight initialization at the beginning of the training process and then seeded PyTorch's RNG for the dropout. Afterwards we checkpoint the RNG state so that we can restore the RNG states at resumption. We found that models with convolutionnal layers would not yield reproducible results unless we enabled `cudnn.deterministic` and disabled `cudnn.benchmark`.

We used the library RoBO (Klein et al., 2017) for our Bayesian Optimizer. There was no support for seeding, we therefore resorted to seeding the global seed of python and numpy random number generators. We needed again to keep track of the RNG states and checkpoint them so that we can resume the Bayesian Optimizer without harming the reproducibility.

For one of our case study, image segmentation, we have been unable to make the learning pipeline perfectly reproducible. This is problematic because it prevents us from studying each source of variation in isolation. We thus trained our model with every seeds fixed across all 200 trainings and measured the variance we could not control. This is represented as the numerical noise in Figures 6.1 and B.3.

B.2. Our bootstrap procedure

Cross-validation with different k impacts the number of samples, it is not the case with not bootstrap. That means flexible sample sizes for statistical tests is hardly possible with cross-validation within affecting the training dataset sizes. Hothorn et al. (2005) focuses on the dataset sampling as the most important source of variation and marginalize out all other sources by taking the average performance over multiple runs for a given dataset. This increases even more the computational cost of the statistical tests.

We probe the effect of data sampling with bootstrap, specifically by bootstrapping to generate training sets and measuring the out-of-bootstrap error, as introduced by Breiman (1996b) in the context of bagging and generalized by Hothorn et al. (2005). For completeness, we formalize this use of the bootstrap to create training and test sets and how it can estimate the variance of performance measure due to data sampling on a finite dataset.

We assume we are seeking to generate sets of i.i.d. samples from true distribution $\mathcal{D}$. Ideally we would have access to $\mathcal{D}$ and could sample our finite datasets independently from it.

$$S_b^t = \{(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)\} \sim \mathcal{D} \tag{B.2.1}$$

Instead we have one dataset $S \sim \mathcal{D}^n$ of finite size n and need to sample independent datasets from it. A popular method in machine learning to estimate performance on a small dataset is cross-validation (Bouckaert & Frank, 2004; Dietterich, 1998). This method however underestimates variance because of correlations induced by the process. We instead favor bootstrapping (Efron, 1979) as used by Hothorn et al. (2005) to simulate independent data

sampling from the true distribution.

$$S_b^t = \{(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)\} \sim S \tag{B.2.2}$$

Where $S_b^t \sim S$ represents sampling the b-th training set with replacement from the set S. We then turn to out-of-bootstrapping to generate the held-out set. We use all remaining samples in $S \setminus S_b^t$ to sample S_b^o.

$$S_b^o = \{(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)\} \sim S \setminus S_b^t \tag{B.2.3}$$

This procedure is represented as $(S^{tv}, S^o) \sim sp_{n,n'}(S)$ in the empirical average risk $\hat{R}_{\mathcal{P}}(S, n, n')$, end of Section 6.2.1.

B.3. Statistical testing

We are interested in asserting whether a learning algorithm A better performs than another learning algorithm B. Measuring the performance of these learning algorithms is not a deterministic process however and we may be deceived if noise is not accounted for. Because of the noise, we cannot know for sure whether a conclusion we draw is true, but using a statistical test, we can at least ensure a bounded rate of false positives (drawing $A > B$ while truth is $A \leq B$) and false negatives (drawing $A \leq B$ while truth is $A > B$). The capacity of a statistical test to identify true differences, that is, of correctly inferring $A > B$ when this is true, is called the statistical power of a test. The procedure we describe here seeks to avoid deception from false positives while providing a strong statistical power.

We will describe the entire procedure, from the generation of the performance measures (Sections B.3.1 & B.3.2), the estimation of sample size (Section B.3.3), computation of $\mathbb{P}(A > B)$ (Section B.3.4), computation of the confidence interval (Section B.3.5) to the inference based on the statistical test (Section B.3.6)

B.3.1. Randomizing sources of variance

As shown in Section 6.3, randomizing as many sources of variance as possible in the learning pipelines help reduce the correlation and thus improve the reliability of the performance estimation. The simplest way to randomize as many as possible is to simply avoid seeding the random number generators. We list here sources of variations we faced in our case studies, but there exists many other sources of variations in diverse learning algorithms and tasks.

Data splits: The data being used should ideally always be different samples from the true distribution of interest. In practice we only have access to a finite dataset and therefore the best we can do is random splits with cross-validation or out-of-bootstrap as described in Appendix B.2.

Data order: The ordering of the data can have a surprisingly important impact as can be observed in Figure 6.1.

Data augmentation: Stochastic data augmentation should not be seeded, so that it follows a different sequence at each run.

Model initialization: Model initialization, e.g. weights initialization in neural networks, should be randomized across all trainings.

Model stochasticity: Learning algorithms sometimes include stochastic computations such as dropout in neural networks (Srivastava et al., 2014), or samplings methods (Kingma & Welling, 2014; Maddison et al., 2017).

Hyperparameter optimization: The optimization of the hyperparameters generally include stochasticity which should ideally be randomized. Running multiple hyperparameter optimizations may often be practically unaffordable. Tests may still be carried out while fixing the hyperparameters after a single hyperparameter optimization, but keep in mind the incurred degradation of the reliability of the conclusion as shown in Section 6.4.

B.3.2. Pairing

Pairing is optional but is highly recommended to increase statistical power. Avoiding seeding is the simplest solution for the randomization, but it is not the best solution. If possible, meticulously seeding all sources of variation with different random seeds at each run makes it possible to pair trainings of the algorithms so that we can conduct paired comparisons.

Pairing is a simple but powerful way of increasing the power of statistical tests, that is, enabling the reliable detection of difference with smaller sample sizes. Let σ_A and σ_B be the standard deviation of the performance metric of learning algorithms A and B respectively. If measures of $\hat{R}_e^A$ and $\hat{R}_e^A$ are not paired, the standard deviation of $\hat{R}_e^A - \hat{R}_e^B$ is then $\sigma_A + \sigma_B$. If we pair them, then we marginalize out sources of variance which results in a smaller variance $\sigma_{A-B} \leq \sigma_A + \sigma_B$. This reduction of variance makes it possible to reliably detect smaller differences without increasing the sample size.

To pair the learning algorithms, sources of variation should be randomized similarly for all of them. For instance, the random split of the dataset obtained from out-of-bootstrap should be used for both A and B when making a comparison. Suppose we plan to execute 10 runs of A and B, then we should generate 10 different splits $\{(S_1^{tv}, S_1^o), (S_2^{tv}, S_2^o), \cdots, (S_{10}^{tv}, S_{10}^o)\}$ and train A and B on each. The performances $(\hat{R}_{ei}^A, \hat{R}_{ei}^B)$ would then be compared only on the corresponding splits (S_i^{tv}, S_i^o). The same would apply to all other sources of variations. In practical terms, pairing A and B requires sampling seeds for each pairs, re-using the same seed for A and B in each pairs.

For some sources of variation it may not make sense to pair. This is the case for instance with weights initialization if A and B involve different neural network architectures. We can still pair. This would not help much, but would not hurt as well. In doubt, it is better to pair.

B.3.3. Sample size

As explained in Section 6.3, the more runs we have from $\hat{R}_e^A$ and $\hat{R}_e^B$, the more reliable the estimate of $\mathbb{P}(A > B)$ is. Lets note this number of runs as the sample size N, not to be confused with dataset size n. There exist a way of computing the minimal sample size required to ensure a minimal rate of false negatives based on power analysis.

We must first set the threshold γ for our test. Based on our experiments in Section 6.4, we recommend a value of 0.75. We then set the desired rates of false positives and false negatives with α and β respectively. Usual value for α is 0.05 while β ranges from 0.05 to 0.2. We recommend $\beta = 0.05$ for a strong statistical power.

The estimation of $P(A > B)$ is equivalent to a Mann–Whitney test (Perme & Manevski, 2019), thus we can use Noether's sample size determination method for this type of test (Noether, 1987).

$$N \geq \left(\frac{\Phi^{-1}(1-\alpha) - \Phi^{-1}(\beta)}{\sqrt{6}(\frac{1}{2} - \gamma)} \right)^2$$

Where Φ^{-1} is the inverse cumulative function of the normal distribution.

Figure B.1 shows how the minimal sample size evolves with γ. Detecting $P(A > B)$ below $\gamma = 0.6$ is unpractical, requiring more that 700 trainings below 0.55 for instance. For a threshold that is representative of the published improvements as presented in Figure 6.3, $\gamma = 0.75$, the minimal sample size required to ensure a rate of 5% false negatives (as defined by $\beta = 0.05$) is reasonably small; 29 trainings.

B.3.4. Compute $\mathbb{P}(A > B)$

For all paired performances $(\hat{R}_{ei}^A, \hat{R}_{ei}^B)$, we compute $I_{\{\hat{R}_{ei}^A, \hat{R}_{ei}^B\}}$, where I is the indicator function. If trainings were not paired as described in Section B.3.2, the pairs are randomly selected. We can then compute $\mathbb{P}(A > B)$ following Equation 6.4.1.

B.3.5. Confidence interval of $\mathbb{P}(A > B)$ with percentile bootstrap

For the estimation of $\mathbb{P}(A > B)$ with values below 0.95, we recommend the use of the the percentile bootstrap[1] (Efron & Tibshirani, 1994).

[1]Percentile bootstrap is not always reliable depending on the underlying distribution and resampling methods but should generally be good for distributions of $\mathbb{P}(A > B)$ of learning algorithms below 0.95. See Canty et al. (2006) for a discussion on the topic.

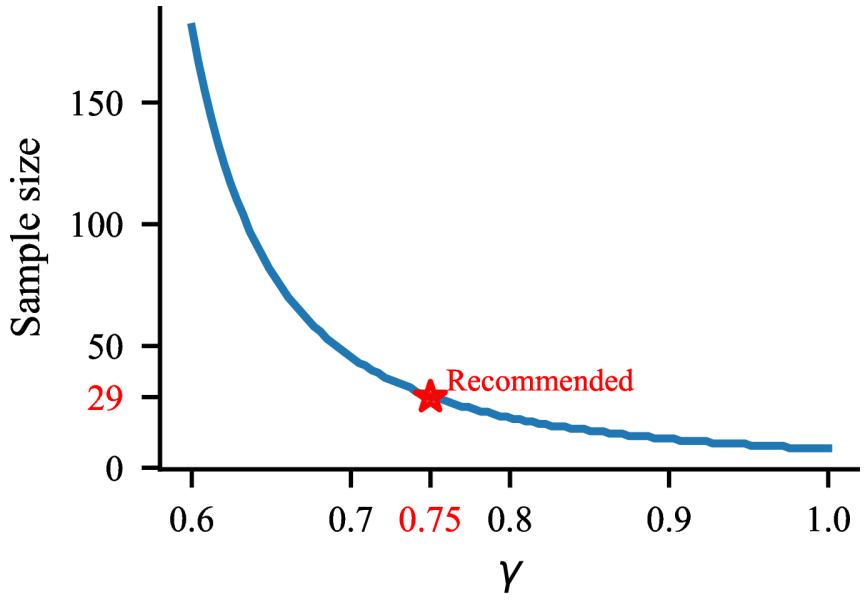


Fig. B.1. Minimum sample size to detect $P(A > B) > \gamma$ reliably. x-axis is the threshold γ and y-axis is the minimum sample size to reliably detect $P(A > B) > \gamma$. The red star shows the recommended threshold γ based on our results in Section 6.4 and the corresponding minimal sample size. We see that detecting reliably $P(A > B) < 0.6$ is unpractical with minimal sample sizes quickly moving above 500. The recommended threshold on the other hand leads to a reasonable sample size of 29.

Suppose we have N pairs $(\hat{R}_{ei}^A, \hat{R}_{ei}^B)$. To compute the percentile bootstrap, we first generate K groups of N pairs. To do so, we sample with replacement N pairs, and do so independently K times. For each of the K groups, we compute $\mathbb{P}(A > B)$. We sort the K estimations of $\mathbb{P}(A > B)$ and pick the $\alpha/2$-percentile and $(1 - \alpha/2)$- percentile as the lower and upper bounds. The confidence interval is defined as these lower and upper bounds computed with percentile bootstrap.

B.3.6. Statistical test with $\mathbb{P}(A > B)$

Let CI_{min} and CI_{max} be the lower and upper bounds of the confidence interval. We draw a conclusion based on the three following scenarios.

$\text{CI}_{\text{min}} \leq 0.5$: : Not statistically significant. No conclusion should be drawn as the result could be explained by noise alone.

$\text{CI}_{\text{max}} \leq \gamma$: : Not statistically meaningful. Perhaps $\text{CI}_{\text{min}} > 0.5$ but it is irrelevant since $\mathbb{P}(A > B)$ is too small to be meaningful.

$\text{CI}_{\text{min}} > 0.5 \wedge \text{CI}_{\text{max}} > \gamma$: : Statistically significant and meaningful. We can conclude that learning algorithm A is better performing than B in the conditions defined by the experiments.

Table 1. Computational infrastructure for CIFAR10-VGG11 experiments.

Hardware/Software	Type/Version
CPU	Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
GPU model	Tesla V100-SXM2-16GB
GPU driver	440.33.01
OS	CentOS 7.7.1908 Core
Python	3.6.3
PyTorch	1.2.0
CUDA	10.2

B.4. Case studies

B.4.1. CIFAR10 Image classification with VGG11

Task. CIFAR10 (Krizhevsky et al., 2009) is a dataset of 60,000 32x32 color images selected from 80 million tiny images dataset (Torralba et al., 2008), divided in 10 balanced classes. The original split contains 50,000 images for training and 10,000 images for testing. We applied random cropping and random horizontal flipping data augmentations.

Bootstrapping. The aggregation of all original training and testing samples are used for the bootstrap. To preserve the balance of the classes, we applied stratified bootstrap. For each class separately, we sampled with replacement 4,000 training samples, 1,000 for validation and 1,000 for testing. As for all tasks, we use out-of-bootstrap to ensure samples cannot be contained in more than one set.

Model. We used VGG11 (Simonyan & Zisserman, 2014) with batch-normalization and no dropout. The weights are initialized with Glorot method based on a uniform distribution (Glorot & Bengio, 2010).

Search space for hyperparameters. We focused on learning rate, weight decay, momentum and learning rate schedule. Batch-size was omitted to simplify the multi-model training on GPUs, so that memory usage was consistent and predictable across all hyperparameter settings. To ease the definition of the search space for the learning rate schedule, we used exponential decay instead of multi-step decay despite the wide use of the latter with similar tasks and models (Simonyan & Zisserman, 2014; Xie et al., 2019; Mahajan et al., 2018; **?**; He et al., 2016b; **?**). The former only require tuning of γ while the later requires additionally selecting number of steps. Search space for all experiments and default values used for the variance experiments are presented in Table 2.

Table 2. Search space and default values for the hyperparameters in CIFAR10-VGG11 experiments.

Hyperparameters	Default	Space
learning rate	0.03	log(0.001, 0.3)
weight decay	0.002	log(10^{-6}, 10^{-2})
momentum	0.9	lin(0.5, 0.99)
γ of lr schedule	0.97	lin(0.96, 0.999)
batch-size	128	-

Table 3. Search space and default values for the hyperparameters in SST-2/RTE-BERT experiments.

Hyperparameters	Default	Space
learning rate	$2 * 10^{-5}$	log(10^{-5}, 10^{-4})
weight decay	0.0	log(10^{-4}, $2 * 10^{-3}$)
std for weights init.	0.2	log(0.01, 0.5)
β_1	0.9	-
β_2	0.999	-
dropout rate	0.1	-
batch size	32	-

B.4.2. Glue-SST2 sentiment prediction with BERT

Task. SST2 (Stanford Sentiment Treebank) (Socher et al., 2013) is a binary classification task included in GLUE (Wang et al., 2018). In this task, the input is a sentence from a collection of movie reviews, and the target is the associated sentiment (either positive or negative). The publicly available data contains around 68k entries.

Bootstrapping. We maintained the same size ratio between train/validation (i.e., 0.013) when performing the bootstrapping analysis. We performed standard out-of-bootstrap without conserving class balance since the original dataset is not balanced and ratios between classes vary from training and validation set in the original splits. The variable ratios of classes across bootstrap samples generate additional variance in our results, but is representative of the effect of generating a dataset that is not perfectly balanced.

Model. We used the BERT (Devlin et al., 2018) implementation provided by the Hugging Face (Wolf et al., 2019) repository. BERT is a Transformer (Vaswani et al., 2017) encoder pre-trained on the self-supervised Masked Language Model task (Devlin et al., 2018). We chose BERT given its importance and influence in the NLP literature. It is worthy to note that the pre-training phase of BERT is also affected by sources of variations. Nevertheless, we didn't investigate this phase given the amount of time (and resources) required to perform it. Instead, we always start from the (same) pre-trained model image provided by the Hugging Face (Wolf et al., 2019) repository. Indeed, the weight initialization was only applied to

the final classifier. The initialization method used is standard Gaussian with 0.0 mean and standard deviation that depends on the related hyperparameter.
Search space of hyperparameters. We ran a small-scale hyperparameter space exploration in order to select the hyperparameter search space to use in our experiments. As such, we decided to include the learning rate, weight decay and the standard deviation for the model parameter initialization (see Table 3). We fixed the dropout probability to the value of 0.1 as in the original BERT architecture. For the same reason, we fixed $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Default values used for the variance experiments are also reported in Table 3. The model has been fine-tuned on SST2 for 3 epochs, with a batch size of 32. Training has been performed with mixed precision. Note that for weight decay we used the default value from the Hugging Face repository (i.e., 0.0) even if this is outside of the hyperparameter search space. We confirmed that this makes no difference by looking at the results of the small-scale hyperparameter space exploration.

B.4.3. Glue-RTE entailment prediction with BERT

Task. RTE (Recognizing Textual Entailment) (Bentivogli et al., 2009) is a also a binary classification task included in GLUE (Wang et al., 2018). The task is a collection of text fragment pairs, and the target is to predict if the first text fragment entails the second one. RTE dataset only contains around 2.5k entries.
Bootstrapping. In our bootstrapping analysis we maintained the train/validation ratio of 0.1. As for Glue-SST2, we used standard out-of-bootstrap and did not preserve original class ratios.
Model & search space of hyperparameters. We used the BERT (Devlin et al., 2018) model for RTE as well, trained in the same way specified in the SST-2 section. In particular, we used the same hyperparameters (see Table 3), same batch size, and we trained in the same mixed-precision environment. The model has been fine-tuned on RTE for 3 epochs.

B.4.4. PascalVOC image segmentation with ResNet Backbone

Task. The PascalVOC segmentation task (Everingham et al., 2012) entails generating pixel-level segmentations to classify each pixel in an image as one of 20 classes or background. This publicly available dataset contains 2913 images and associated ground truth segmentation labels. The original splits contains 2184 images for training and 729 for validation. Images were normalized and zero-padded to a final size of 512x512.
Bootstrapping. We used a train/validation ratio of 0.25 for our bootstrap analysis, generating training sets of 2184 images, validation and test sets of 729 images each. Since multiple classes can appear in a single image, the original dataset was not balanced, we thus used standard out-of-bootstrap for our experiments.

Table 4. Computational infrastructure for PASCAL VOC experiments.

Hardware/Software	Type/Version
CPU	Intel(R) Xeon(R) Silver 4216 CPU @ 2.1GHz
GPU model	Tesla V100 Volta 32G
GPU driver	440.33.01
OS	CentOS 7.7.1908 Core
Python	3.6.3
PyTorch	1.2.0
CUDA	10.2

Table 5. Search spaces for PASCAL VOC image segmentation.

Hyperparameters	Default	Space
learning rate	0.002	log(10^{-5}, 10^{-2})
momentum	0.9	lin(0.50, 0.99)
weight decay	0.000001	log(10^{-8}, 10^{-1})
batch-size	16	-

Table 6. Search spaces for the different hyperparameters for the MLP-MHC task

Hyperparameters	Default	Space
hidden layer size		lin(20, 400)
L2-weight decay		log(0, 1)

Model. We used an FCN-16s (Long et al., 2014) with a ResNet18 backbone (He et al., 2016a) pretrained on ImageNet (Deng et al., 2009). After exploring several possible backbones, ResNet18 was selected since it could be trained relatively quickly. We use weighted cross entropy, with only predictions within the original image boundary contributing to the loss. The model is optimized using SGD with momentum.

Metric. The metric used is the mean Intersection over Union (mIoU) of the twenty classes and the background class. The complement of the mIoU, the mean Jaccard Distance, is the metric minimized in all HPO experiments.

Search space of hyperparameters. Certain hyperparmeters, such as the number of kernals, or the total number of layers, are part of the definition of the ResNet18 architecture. As a result, we explored key optimization hyperparameters including: learning rate, momentum, and weight decay. The hyperparameter ranges selected, as well as the default hyperparameters used in the variance experiments, can be found in table 5 and in table **??**, respectively. A batch size of 16 was used for all experiments.

# HPs	Hyperparameters	Default Value
1	hidden layer size	150
2	L2-weight decay	0.001

Table 7. Defaults for MLP-MHC task.

Table 8. Comparison of performance on datasets

Model name	Dataset	AUC	PCC
NetMHCpan4	HPV	0.53	0.39
MHCflurry	HPV	0.58	0.41
MLP-MHC	HPV	0.63	0.31
NetMHCpan4	NetMHC-CVsplits	0.854	0.620
MHCflurry	NetMHC-CVsplits	0.964*	0.671*
MLP-MHC	NetMHC-CVsplits	0.861	0.660

Table 9. Comparison of models for the MLP-MHC task

Model name	Inputs	Model design	Dataset	Sequenc
NetMHCpan4	allele+peptide	shallow MLP	custom CV split(Vita et al., 2019)	BLC
MHCflurry	peptide	ensemble of shallow MLPs	(O'Donnell et al., 2018)	BLC
MLP-MHC	allele+peptide	shallow MLP	same as (O'Donnell et al., 2018)	S

Table 10. Computational infrastructure for MLP-MHC experiments.

Hardware/Software	Type/Version
CPU	Intel(R) Xeon(R) CPU E5-2640 v4
	320 CPU @ 2.40GHz
OS	CentOS 7.7.1908 Core
Python	3.6.8
sklearn	0.22.2.post1
BLAS	3.4.2

B.4.5. Major histocompatibility class I-associated peptide binding prediction with shallow MLP

Task. The MLP-MHC is a regression task with the goal of predicting the relative binding affinity for a given peptide and major histocompatibility complex class I (MHC) allele pair. The major histocompatibility complex (MHC) class I proteins are present on the surface of most nucleated cells in jawed vertebrates (Pearson et al., 2016). These proteins bind short peptides that arise from the degradation of intra-cellular proteins (Pearson et al., 2016). The complex of peptide-MHC molecule is used by immune cells to recognize healthy cells and eliminate cancerous or infected cells, a mechanism studied in the development of immunotherapy and vaccines (O'Donnell et al., 2018). The peptide binding prediction task

is therefore at the base of the search for good vaccine and immunotherapy targets (O'Donnell et al., 2018; Jurtz et al., 2017).

The input data is the concatenated pairs of sequences: the MHC allele and the peptide sequence. For the MHC alleles, we restricted the sequences to the binding pocket of the peptide, as seen in Jurtz et al. (2017). The prediction target is a normalized binding affinity score, as described in Jurtz et al. (2017) and O'Donnell et al. (2018).

Datasets and sequence encoding. While both *MHCflurry* and *NetMHCpan4* models use a BLOSUM62 encoding (Henikoff & Henikoff, 1992) for the amino acids, in we chose to instead encode the amino acids as one-hot as described in Nielsen et al. (2007).

The *NetMHCpan4* model is trained on a manually filtered dataset from the immune epitope database (Vita et al., 2019; Jurtz et al., 2017) that has been split into five folds used for cross-validation, available on the author's website (Jurtz et al., 2017).

In contrast, the *MHCflurry* model is trained on a custom multi-source dataset (available from Mendeley data and the O'Donnell et al. (2018) publication cite) and validated/tested on two external datasets from Pearson et al. (2016) and an HPV peptide dataset available at the same website as above.

Bootstrapping. We have three different sets for training, validating and testing. We thus performed bootstrapping separately on each set for every training and evaluation.

Model. The model is a shallow MLP with one hidden layer from *sklearn*. We used the default setting for the non-linearity *relu* and weight initialization strategy (Glorot & Bengio, 2010). The following table (Table 9) offers some comparison points between our model and the *NetMHCpan4* (Jurtz et al., 2017) and *MHCflurry* (O'Donnell et al., 2018) models.

While the *MHCflurry* model (O'Donnell et al., 2018) train only no the peptide sequences and uses ensembling to perform its predictions, training multiple models for each MHC allele, the *NetMHCpan4* model (Jurtz et al., 2017) uses the allele sequence as input and trains one single model.

We chose to retain the strategy proposed by the *NetMHCpan4* model, where a single model is trained for all alleles (Jurtz et al., 2017). As a reference, MHCflurry uses ensembling to perform predictions; indeed, the authors report that for each MHC allele, an ensemble of 8-16 are selected from the 320 that were trained (O'Donnell et al., 2018).

Search space of hyperparameters. For the hyperparameter search, we selected hidden layer sizes between 20 and 400 (Table 6), to engulph a range slightly larger than the ones described by both Jurtz et al. (2017) and O'Donnell et al. (2018). The second hyperparameter that was explored was the L2 regularisation parameter, for which a log-uniform range between 0 and 1 was explored.

Comparison of performance. We would like to state the goal of the present study was not to establish new state of the art (SOTA) on the MHC-peptide binding prediction task. However, we still report that when comparing the performance of our model to those of *NetMHCpan4*

and *MHCflurry* we found the performance of our model comparable. Briefly, for the results in Table 9, we used the existing pre-trained *NetMHCpan4* and *MHCflurry* tools to predict the binding affinity of both datasets: the previously described HPV external test data (HPV) from O'Donnell et al. (2018) and the cross-validation test datasets from Jurtz et al. (2017) (NetMHC-CVsplits).

We would like to point out that since the *MHCflurry* model was published later than the *NetMHCpan4* one, there is a high chance that the dataset from the cross-validation splits (NetMHC-CVsplits) may be contained in the dataset used to train the existing *MHCflurry* tool. The proper way to compare performances would be to re-train the *MHCflurry* model on each fold and test susequently its performance; however, since our goal is not to reach new SOTA on this task, we leave this experiment to be performed at a later time.

This would result in a likely overestimation of the performance of *MHCflurry* on this dataset, which we noted with the $*$ sign in Table 9.

A more in-depth study is necessary to compare in a more through way this performance with respect to the differences in model design, dataset encoding and other factors.

B.5. Hyperparameter optimization algorithms

B.5.1. Grid Search

Let a_i, b_i and n be the hyperparameters of the grid search, where a_i and b_i are vectors of minima and maxima for each dimension of the search space, and n is the number of values per dimension. We define Δ_i as the interval between each value on dimension i. A point on the grid is defined by $p_{ij} = a_i + \Delta_i(j-1)$. Grid search is simply the evaluation of $r(\lambda)$ from Equation 6.2.2 on all possible combinations of values p_{ij}.

B.5.2. Noisy Grid Search

Grid search is a fully deterministic algorithm. Yet, it is highly sensitive to the design of the grid. To provide a variance estimate of similar choices of the grid and to be able to distinguish lucky grid, we consider a noisy version of grid search.

For the noisy grid search, we replace a_i by $\tilde{a}_i \sim U(a_i - \frac{\Delta_i}{2}, a_i + \frac{\Delta_i}{2})$ and similarly for b_i. $\tilde{\Delta}_i$ and $\tilde{p_{ij}}$ then follows from $\tilde{a}_i$ and $\tilde{b}_i$. In expectation, noisy grid search will cover the same grid as grid search, as proven below.

$$
\begin{aligned}
\mathbb{E}[\tilde{p}_{ij}] &= \mathbb{E}\left[\tilde{a}_i + \tilde{\Delta}_i(j-1)\right] \\
&= \mathbb{E}\left[\tilde{a}_i + \frac{\tilde{b}-\tilde{a}}{n-1}(j-1)\right] \\
&= \mathbb{E}\left[\tilde{a}_i\right] + \mathbb{E}\left[\frac{\tilde{b}}{n-1}\right](j-1) - \mathbb{E}\left[\frac{\tilde{a}}{n-1}\right](j-1) \\
&= a_i + \frac{b}{n-1}(j-1) - \frac{a}{n-1}(j-1) \\
&= a_i + \frac{b-a}{n-1}(j-1) \\
&= a_i + \Delta_i(j-1) = p_{ij}
\end{aligned}
$$

This provides us a variance estimate of grid search that we can compare against non-deterministic hyperparameter optimization algorithms.

B.5.3. Random Search

The search space of random search will be increased by $\pm\frac{\Delta_i}{2}$ as defined for the noisy grid search to ensure that they both cover the same search space. For all hyperparameters, the values are sampled from a uniform $p_i \sim U(a_i - \frac{\Delta_i}{2}, b_i + \frac{\Delta_i}{2})$. For learning rate and weight decay, values are sampled uniformly in the logarithmic space.

B.6. Hyperparameter optimization results

Figure B.2 presents the optimization curves of the hyperparameter optimization executions in Section 6.2.2.

B.7. Normality of performance distributions in the case studies

Figure B.3 presents the Shapiro-Wilk test of normality on all our results on sources of variations.

B.8. Randomizing more sources of variance increase the quality of the estimator

Figure 6.5 only presented the Glue-RTE and CIFAR10 tasks. We provide here a complete picture of the standard deviation of the different estimators in Figure B.4. We further present a decomposition of the mean-squared-error in Figure B.5 to help understand why accounting for more sources of variations improves the mean-squared-error of the biased estimators.

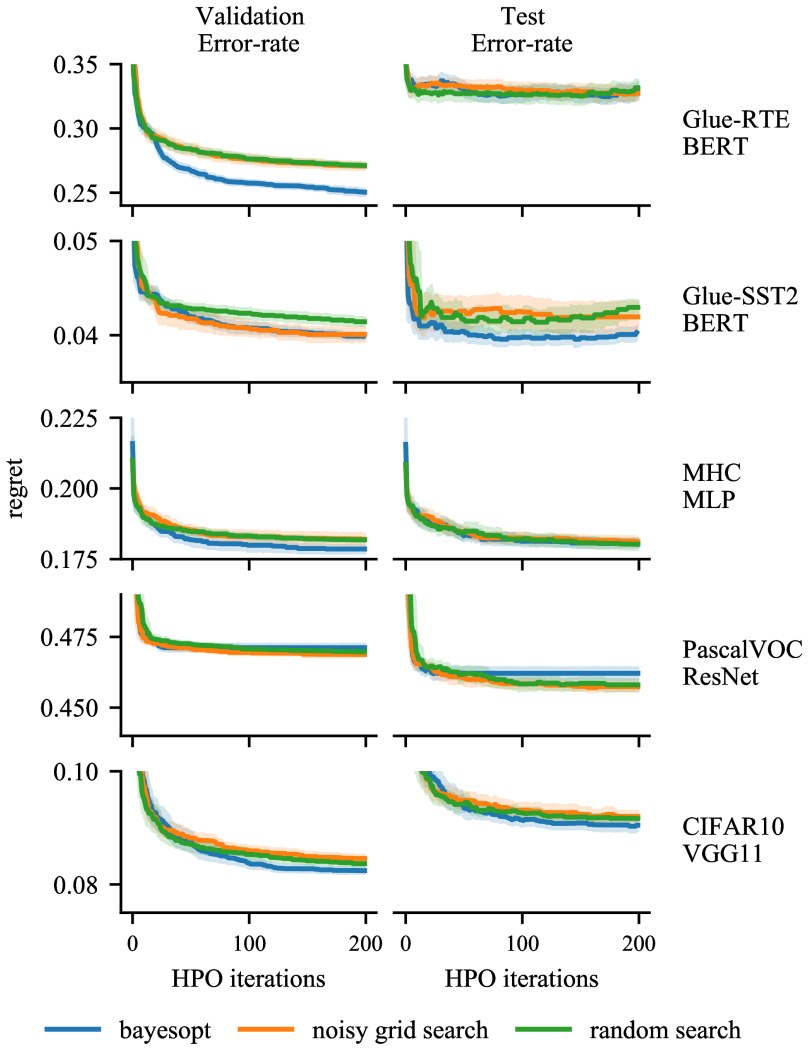


Fig. B.2. Optimization curves of hyperparameter optimization executions. Each row presents the result for a different task. Left column are results on validation set, the one hyperparameters were optimized on. Right column are results on the test sets. Hyperparameter optimization methods are Bayesian Optimization, Noisy Grid Search (See Section B.5.2), and Random Search. The y-axis are the best objectives found until an iteration i, on a different scale for each task. Left and right plots share the same scale on y-axis, so that we can easily observe whether validation error-rate corresponds to test error-rate. The bold lines are averages and the size of lighter colored areas represents the standard deviations. They are computed based on 20 independent executions for each algorithms, during which only the seed of the hyperparameter optimization is randomized. For more details on the experiments see Section 6.2.2.3. Two striking results emerge from these graphs. 1) The typical search spaces are well optimized by all algorithms, and in some cases there is even signs of slight over-fitting (on BERT tasks). 2) The standard deviation stabilizes early, before 50 iterations in most cases. These results suggests that larger budgets for hyperparameter optimization would not reduce the variability of the results in similar search spaces. This is likely not the case however for more complex search spaces such as those observed in the neural architecture search literature.

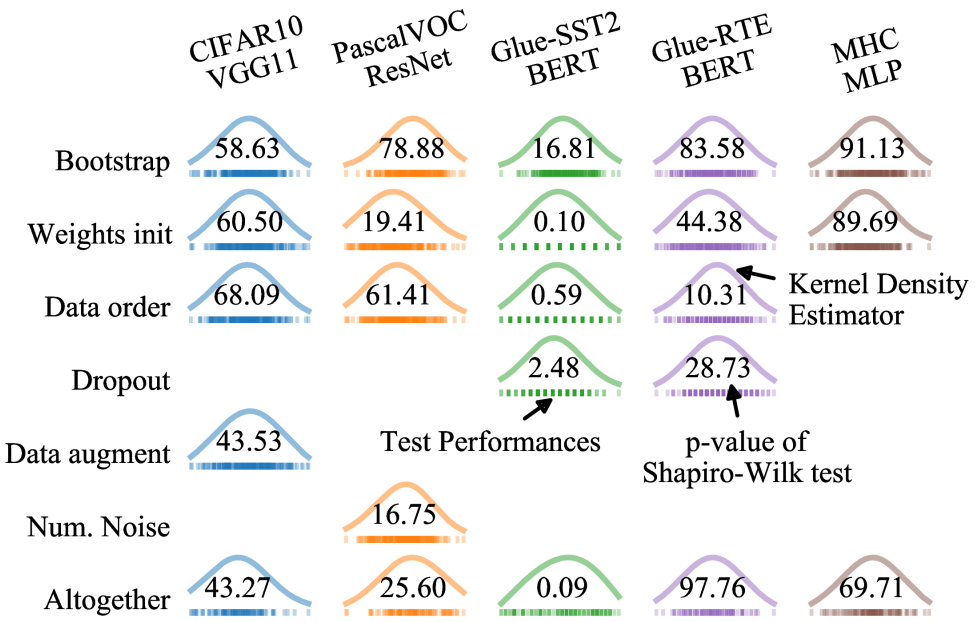


Fig. B.3. Performance distributions conditional to different sources of variations. Each row is a different source of variation. For each source, all other sources are kept fix when training and evaluating models. The last row presents the distributions when all the sources of variation are randomized altogether. Each column is the results for the different tasks. We can see that except for Glue-SST2 BERT, all case studies have distributions of performances very close to normal. In the case of Glue-SST2 BERT, we note that the size of the test set is so small that it discretizes the possible performances. The distribution is nevertheless roughly symmetrical and thus amenable to many statistical tests.

B.9. Analysis of robustness of comparison methods

In addition to simulations described in Section 6.4.2, we executed experiments in which we varied the sample size and the threshold γ. To select the threshold of the average, we converted γ into the equivalent performance difference ($\delta = \Phi^{-1}(\gamma)\sigma$). Results are presented in Figure B.6

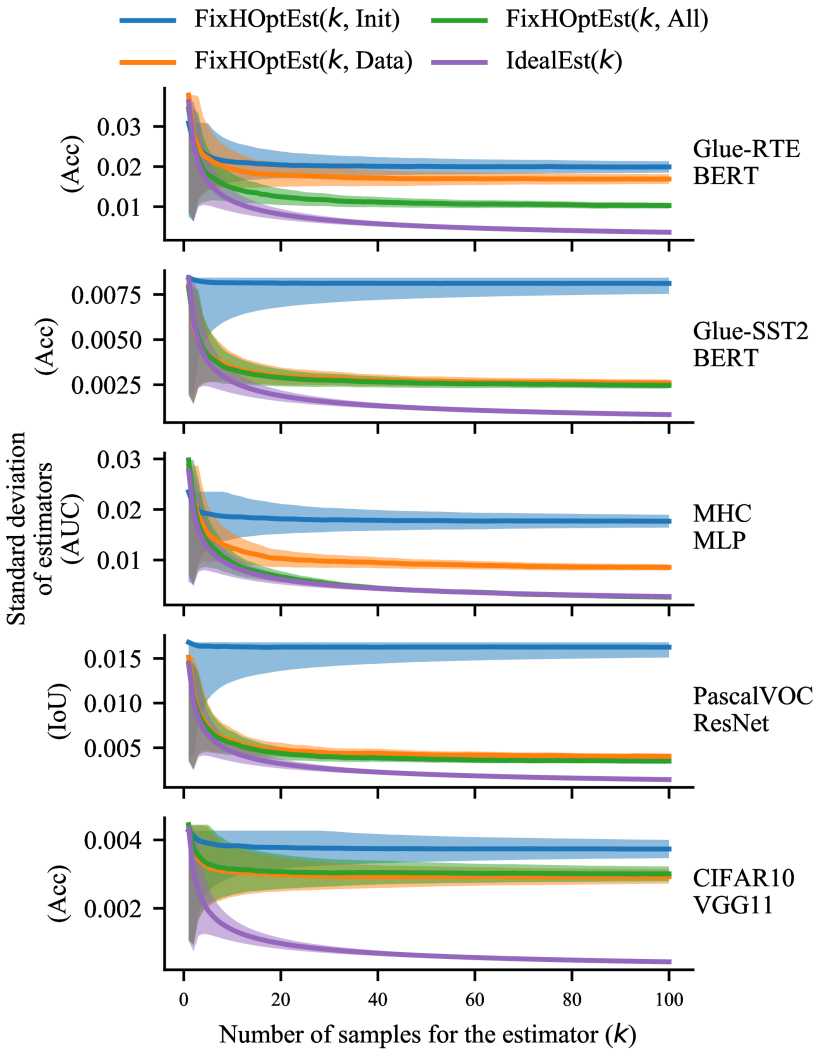


Fig. B.4. Standard error of biased and ideal estimators with k samples. Each plot represents the standard error of the different tasks described in Section 6.2.2. On x axis, the number of samples used by the estimators to compute the average performance. On y axis, the standard deviation of the estimations, in terms of task objective; Classification accuracy (Acc), Intersection over Union (IoU), Area Under the Curve (AUC). Uncertainty represented in light color is computed analytically as the approximate standard deviation of the standard deviation of a normal distribution computed on k samples. For all case studies, **accounting for more sources of variation reduces or keeps constant the standard error of $\hat{\mu}_{(k)}$.** In all case studies, only accounting for weights initialization, `FixHOptEst(k, Init)`, is by far the worst estimator. Comparatively, `FixHOptEst(k, All)` provides a systematic improvement towards `IdealEst(k)` for no additional computational cost compared to `FixHOptEst(k, Init)`. **Ignoring variance from** HOpt **is harmful for a good estimation of $\hat{R}_{\mathcal{P}}$.** The MHC task with MLP is the only one for which `FixHOptEst(k, All)` matches `IdealEST(k, All)`. We suspect this may be explained by the relatively small standard deviation due to hyperparameter optimization observed in Figure 6.1. `FixHOptEst(k, All)` would have thus captured most of the variability in the learning pipeline.

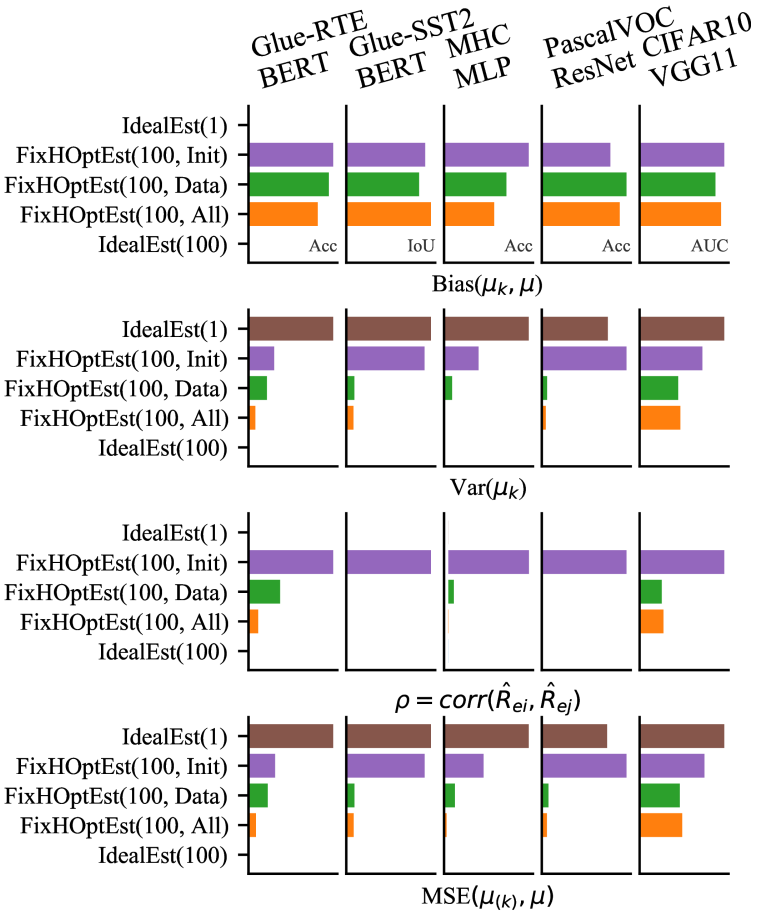


Fig. B.5. Decomposition of the Mean-Squared-Error for different estimators of $\hat{R}_{\mathcal{P}}$. On each sub-figure from top to bottom, 1) bias between the estimator and the expected empirical risk Bias$(\mu_{(k)}, \mu)$, 2) variance of the estimator Var$(\mu_{(k)})$, 3) correlation between performances measures $\hat{R}_e$ as presented in Equation 6.3.2 and 4) the mean-squared-error of the estimator MSE$(\mu_{(k)}, \mu)$. For each sub-figure, each row is a different estimators, with `IdealEst(k=1)` as a comparison point. The experimental procedure to compute these statistics are described in section subsection 6.3.3. Without any surprise the `IdealEst(100, All)` minimizes the mean-squared-error so well that it looks close to 0 on the figure compared to the other estimators. Among the other estimators, the mean-squared-error is reduced most significantly by `FixedHOptEst(100, All)` on all tasks. If we look at the decomposition of the mean-squared-error, i.e., the bias and the variance, we see on first sub-figure that the bias is stable across all biased estimators on all tasks, while on second sub-figure the variance varies widely. It is thus the reduced variance of the biased estimators that leads to improved mean-squared-error. This is a counter-intuitive result because the estimator with lowest variance are these accounting for more sources of variations. The intuition is thus that they should have more variance, not less. We derived the variance of the biased estimators in Equation 6.3.2 which highlighted that the correlation among performances $\hat{R}_e$ can increase the variance of the biased estimators. We can see in the third sub-figure that this correlation drastically drops when accounting for more sources of variances. The mean-squared-error, in other words the quality of the estimators, is thus significantly improved by decorrelating the performance measures.

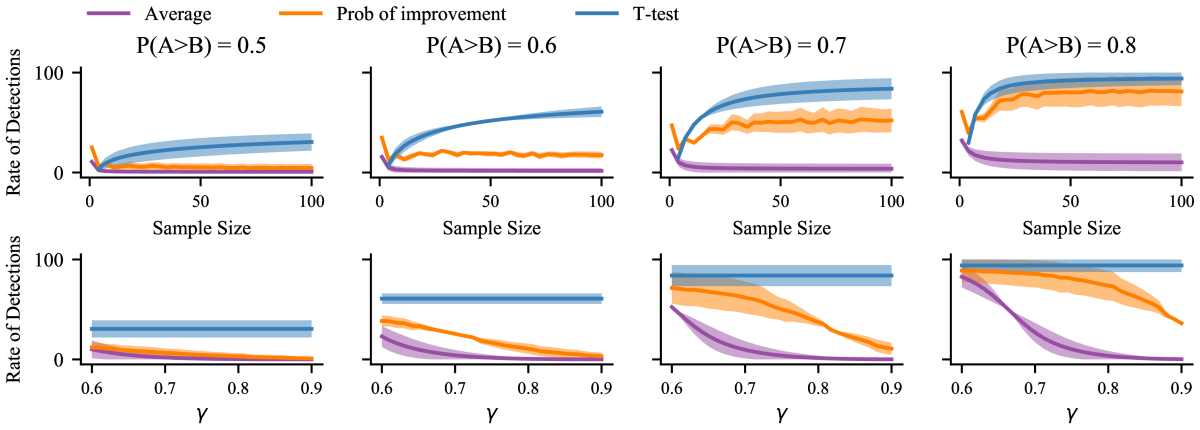


Fig. B.6. Analysis of the robustness of comparison methods. On the first row, rate of detections of comparison methods in function of the sample size. On the second row, rate of detections of comparison methods in function of the threshold γ. Each column are simulations with different true simulated probability of of a learning algorithm A to outperform another algorithm B across random fluctuations (ex: random data splits).

Appendix C

Oríon: Comparison Table

C.1. Criteria for comparison of frameworks

C.1.1. Usability

These criteria attests for the ease of use of the libraries. How quickly the user can get up to speed, how easily it can debug and inspect processes.

C.1.1.1. Languages supported

The programming languages supported are languages used by the user to execute the black-box process. It may be different than the language used to implement the library itself.

C.1.1.2. Disruptive

The library is considered disruptive for the researcher's workflow if it requires substantial modifications. Libraries that offer flexible interfaces are less disruptive while libraries that requires very specific implementations are highly disruptive.

Low: The code requires only 1-5 lines of modifications and does not affect the control flow of the user's program.

Mild: The code may require more than 5 lines of code or affect the control flow of the user's program.

High: The code requires more than 5 lines of code and affect the control flow of the user's program.

C.1.1.3. Scheduling

Some libraries are limited to specific scheduling systems (ex: through Kubernetes). This criteria evaluates whether a library supports only a few schedulers or if they are compatible with any.

C.1.1.4. Auto-scalability

Libraries integrated with scheduling systems may support auto-scaling, that is, the capacity of dynamically allocating computational resources and spawning workers according to the needs of the hyperparameter optimization process.

C.1.1.5. Fault tolerance

Hyperparameter optimization at scale may fail on several fronts. Fault tolerance is an important feature to streamline the optimization. Faults considered are errors during the black box optimization, crash of the optimization algorithm process or of the worker.

C.1.1.6. Visualizations

Visualisation is crucial tool to allow inspection of the optimization process and interpretation of the results.
No: No vizualisations.
Min: Minimal plots: Regret plot
Good: Coverage of a good set of vizualizations: Regret, Parallel Coordinate plots
High: Coverage of a large set of vizualizations: Regret, Parallel Coordinate plots, Optimization space, and more.

C.1.2. Algorithms

These high level criteria attest for the variety of algorithms supported by the libraries.

C.1.2.1. Bayesian Optimization

We consider any type of Bayesian optimization, from classical Gaussian process based Bayesian Optimization to Tree Structured Parzen Estimator (TPE) (Bergstra et al., 2011).

C.1.2.2. Bandit

Arm-based optimization algorithms fall in the *Bandit* category. These include Hyperband (Li et al., 2018a) and ASHA (Li et al., 2020).

C.1.2.3. Evolutionary

We include in the *evolutionary* category any algorithm using mutations. This, among others, includes Population Based Training (PBT) (Jaderberg et al., 2017) and Covariance matrix adaptation evolution strategy (CMA-ES) (Hansen & Ostermeier, 2001).

C.1.3. Strategies for efficiency

Hyperparameter optimization is computationally expensive. We include in this comparison three features that improves its efficiency.

C.1.3.1. Conditional

Conditional search space helps reducing the explorable space and thus can speed up hyperparameter optimization dramatically. A library may support one or many of the points below. We report *Yes* in Table 1 if all points are supported, *Part.* otherwise.

- Hierarchical choices. Ex: Randomly selecting one of different group of choices and then randomly selecting a choice within this group.
- Conditional choices. Ex: The group of choices to sample from is conditional to the value of another hyperparameters.
- Conditionally constrained space. Ex: The possible values is constrained based on values of other hyperparameters. This may be applied to any type of prior, not only choices.

C.1.3.2. Multi-fidelity

Multi-fidelity speeds up hyperparameter optimization by looking at low fidelity first (ex: objective value after few epochs of training) to guide selection of hyperparameters to try at higher fidelity. For instance, libraries that includes Hyperband (Li et al., 2018a) or ASHA (Li et al., 2020) supports multi-fidelity.

C.1.3.3. Warm-starting

Algorithms like Bayesian optimization can converge faster if provided with prior data, a procedure known as warm-starting.
No: No warm-start possible.
Part.: Requires the user to manually feed the algorithm with data.
Yes: Supports warm-starting through automated transfert of data from one experiment to another.

C.2. Evaluation of frameworks

C.2.1. Ax (Bakshy et al., 2018)

C.2.1.1. Usability

Languages supported. Only supports python.

Disruptive. (Low)

The loop API allows defining an optimization loop with a single function call, but is limited to sequential optimization. The service API requires few additional lines to define the client, retrieve parameters and send results. The service API can easily fit in most research workflows.

Scheduling. There is no scheduling system in Ax. It must be integrated with the service of the developer API.

Auto-scalability. There is no scheduling system and thus no auto-scalability capability.

Fault tolerance. No automatic handling of failing trials is provided in the different APIs.

Visualizations. (High)

Ax offers a large variety of plots. Among many, there are contour plots of the search space, feature importance plots, marginal effects (equivalent to partial dependencies), pareto plots for multi-objective optimization, slice plots to represent 1-d views of the search space, regret plot with support for averages across runs and bar plots of optimization times.

C.2.1.2. Algorithms

Bayesian Optimization. Wrapping on top of BoTorch, Ax provides flexible and efficient solutions for Bayesian Optimizations.

Bandit. Ax allows a factorial design of the experiments which enables bandit optimization. The use of the factorial design requires however significantly more work from the researchers than out-of-the-box implementations of algorithms such as Hyperband (Li et al., 2018a).

Evolutionary. There are no Evolutionary algorithms in Ax.

C.2.1.3. Strategies for efficiency

Conditional. (No)

There is no support for conditional search spaces.

Multi-fidelity. (Yes)

Multi-fidelity can be achieved with the multi-factorial design of the experiments.

Warm-starting. (Part.)

The Bayesian optimization algorithms of BoTorch can be manually fitted on prior data by the user. There is no system that automates warm-starting.

C.2.2. HyperOpt (Bergstra et al., 2013)

C.2.2.1. Usability

Languages supported. Only supports python.

Disruptive. (Mild)

The fmin function executes sequential optimization with a single function call. It can be turned into a master process if using the MongoDB trials. In this case workers must be spawned separately. The arguments passed to the function are not unpacked and requires additional tinkering from the user to support calls from fmin.

Scheduling. (Any)

There is no scheduling system in HyperOpt, but it can be interfaced with a Spark cluster using the Spark Trials.

Auto-scalability. (Part.)

Only possible through Spark's dynamic resource allocation when using spark trials.

Fault tolerance. (Part.)

Only when using spark trials.

Visualizations. (Min)

The repository contains 4 undocumented functions to plot time-series of objectives, histogram of objectives, per-dimension scatter plots of dimension values and objectives, and a line plot of attachment data saved in trials.

C.2.2.2. Algorithms

Bayesian Optimization. Hyperopt is the reference for the original implementation of the Tree Structured Parzen Estimator (TPE) (Bergstra et al., 2011).

Bandit. None

Evolutionary. None

C.2.2.3. Strategies for efficiency

Conditional. (Part.)

Hyperopt supports hierarchical choices.

Multi-fidelity. None

Warm-starting. (Part.)

The user may warm-start the TPE by manually inserting data before starting the optimization.

C.2.3. Katib (George et al., 2020)

C.2.3.1. Usability

Languages supported. Any languages are supported.

Disruptive. (High)

On the user's code side, Katib is fairly non-intrusive supporting hyperparameter definition through commandline arguments or environment variables. The setup required to run it with kubeflow is however very disruptive for any researcher not already using it.

Scheduling. Natively supported through kubeflow.
Auto-scalability. Natively supported through kubeflow.
Fault tolerance. Supported through kubeflow.
Visualizations. (Min)

Katib provides a dashboard including a parallel coordinates plot and a table of trial results.

C.2.3.2. Algorithms

Bayesian Optimization. Katib interfaces with hyperopt, which contains original python implementation of TPE (Bergstra et al., 2011), GOptuna, which contains native re-implementation of TPE in Go language, skopt (bayesian opt), chocolate which contains a native implementation of bayesian optimization based on scikit-learn's gaussian processes.
Bandit. Katib includes a native implementation of Hyperband (Li et al., 2018a).
Evolutionary. Katib includes CMA-ES.

C.2.3.3. Strategies for efficiency

Conditional. We have found no support for any form of conditional search spaces in Katib.
Multi-fidelity. Multi-fidelity is supported through Hyperband (Li et al., 2018a).
Warm-starting. We have found no support for warm-starting in Katib.

C.2.4. Nevergrad (Rapin & Teytaud, 2018)

C.2.4.1. Usability

Languages supported. Python and R
Disruptive. (Low)

Setting up experiment with Nevergrad requires only a few lines of code. It is also possible to adapt to most user workflows using the ask/tell interface.
Scheduling. There is no scheduling system in Nevergrad. It must be integrated with the ask/tell interface.
Auto-scalability. There is no scheduling system, but a `batch mode` is available to help parallelize workers locally using `concurrent.futures.ThreadPoolExecutor`.
Fault tolerance. We have found no mechanisms for fault tolerance.
Visualizations. (Min)

Nevergrad supports plotting of the regret curve along with a ranking frequency matrix called *fight plots* in the library.

C.2.4.2. Algorithms

Bayesian Optimization. An optimizer wrapping the library BayesianOptimization[1] provides support for Bayesian Optimization.
Bandit. The only algorithm supported is NoisyBandit.
Evolutionary. Among the libraries considered, Nevergrad is by far the one containing the largest number of evolutionary algorithm implementations. It is also by far the one containing the most gradient-free optimization methods.

C.2.4.3. Strategies for efficiency

Conditional. We have found no support for conditional dimensions in Nevergrad.
Multi-fidelity. We have found no support for multi-fidelity in Nevergrad.
Warm-starting. Nevergrad supports a form of warm-starting through *chaining*. The *chaining* of algorithms consists in defining a sequence of algorithms that will be used one after the other when given budgets of trials are reached. The trials observed by a given algorithm is passed to the next one.

C.2.5. NNI

C.2.5.1. Usability

Languages supported. Python
Disruptive. (Low)

NNI can easily be integrated in most code using `nni.get_next_parameter()` and `nni.report_final_result()`.

Scheduling. NNI provides support for many schedulers. It supports multiple services based on Kubernetes (OpenPAI, Kubeflow, AKS) as well as Azure Machine Learning and ssh-based scheduling.
Auto-scalability. Auto-scalability can be achieved with most schedulers supported by NNI.
Fault tolerance. We could not find fault tolerance options in NNI itself. The schedulers can offer some form of fault tolerance however.
Visualizations. The dashboard in NNI provides a regret curve and parallel coordinates plots.

C.2.5.2. Algorithms

Bayesian Optimization. NNI provides wrappers for BOHB (Falkner et al., 2018), TPE (Bergstra et al., 2011), SMAC, a native Metis Tuner (Li et al., 2018b) and a Bayesian Optimizer based on sklearn.
Bandit. NNI provides Hyperband (Li et al., 2018a).

[1] github.com/fmfn/BayesianOptimizer

Evolutionary. NNI provides the Population Based Training algorithm (Jaderberg et al., 2017).

C.2.5.3. Strategies for efficiency

Conditional. NNI supports hierarchical choices with the `choice` parameter type.
Multi-fidelity. Multi-fidelity is supported through Hyperband (Li et al., 2018a). It is implemented as a combination of `Tuner` and `Assessor` which makes it only support time based fidelities.
Warm-starting. We have found no ways to warm-start algorithms with NNI.

C.2.6. Optuna (Akiba et al., 2019)

C.2.6.1. Usability

Languages supported. Python
Disruptive. (High)

The *pythonic search space* of Optuna allows for more flexibility but makes it more disruptive to adapt existing code.
Scheduling. Optuna is compatible with any scheduling system.
Auto-scalability. Optuna does not support auto-scalability natively but its integration with Dask and Ray makes it possible to scale executions of Optuna.
Fault tolerance. Failing trials will cause the optimizer to end unless the user defines special types of exceptions that must be ignored.
Visualizations. (High)

Optuna offers a rich set of visualizations: Regret plot, parallel coordinate plots, hyperparameter importance, partial dependency plot (called contour plots in Optuna), pareto front for multi-objective experiments, and more.

C.2.6.2. Algorithms

Bayesian Optimization. Optuna provides a native implementation of TPE (Bergstra et al., 2011) including support for multi-dimensional optimization (through `sample_relative()`).
Bandit. Optuna provides an implementation of Hyperband (Li et al., 2018a) as a 'pruner'.
Evolutionary. Optuna provides a wrapper for the library `cmaes`[2].

C.2.6.3. Strategies for efficiency

Conditional. Optuna has one of the best support for conditional distributions thanks to its *pythonic search space*.

[2] github.com/CyberAgent/cmaes

Multi-fidelity. Optuna provides an implementation of Hyperband (Li et al., 2018a) as a `Pruner`. It is more constraining than a general implementation for which the fidelity is not assumed to be related to training time, but can still be considered as multi-fidelity.
Warm-starting. Optuna supports fixing hyperparameters to warm-start algorithms in smaller search space using the `PartialFixedSampler`.

C.2.7. Ray-Tune (Liaw et al., 2018)

C.2.7.1. Usability

Languages supported. Python
Disruptive. (Mild)

Most features of Ray-Tune can be used with few lines of codes. The user is required however to define a function that will be optimized, thus constraining the supported workflows.
Scheduling. Scheduling is done through Ray, which supports Kubernetes for the cloud and Slurm for HPC.
Auto-scalability. Ray allows auto-scalability through the cluster managers and ray serve.
Fault tolerance. Ray-Tune supports rescheduling of crashing trials.
Visualizations. Ray-Tune natively generates TensorBoard files which can be used to visualize parallel coordinates plots.

C.2.7.2. Algorithms

Bayesian Optimization. Through extensive wrappers, Ray-Tune supports multiple variants of Bayesian Optimization: `AxSearch`, `DragonflySearch`, `SkoptSearch`, `BayesOptSearch`, `TuneBOHB`, `OptunaSearch` and `NevergradSearch`.
Bandit. Ray-Tune have native implementations of both Hyperband (Li et al., 2018a) and ASHA (Li et al., 2020). It also supports BOHB (Falkner et al., 2018) through a wrapper.
Evolutionary. Ray-Tune have a native implementation of Population Based Training (PBT) (Jaderberg et al., 2017). It also supports other evolutionary algorithms through the wrappers `NevergradSearch` and `OptunaSearch`.

C.2.7.3. Strategies for efficiency

Conditional. Ray-Tune supports very flexible conditional search spaces with the method `tune.sample_from()` that can access other hyperparameter values and adjust accordingly.
Multi-fidelity. Ray-Tune supports multi-fidelity with algorithms Hyperband, ASHA and BOHB. All these algorithms are implemented in the form of schedulers however which limits them to time-based fidelities.
Warm-starting. Users can manually pass a list of trials to try initially if no objectives given, or warm-start if objectives are given using argument `points_to_evaluate`.

C.2.8. SMAC3 (Lindauer et al., 2017)

C.2.8.1. Usability

Languages supported. Python
Disruptive. (High)

SMAC3 only supports optimizing functions that complies to SMAC3 interfaces, which constraints the possible workflows for users. The search space used by SMAC3, `ConfigSpace`, is one of the most powerful, but it is more verbose, more complex and requires more code than most other libraries.

There exists an simpler interface `fmin_smac` for the optimization of functions with simple search spaces.
Scheduling. SMAC3 relies on file-system to share information across workers by default. It can be used with any scheduler. SMAC3 also provides a bridge to Dask.
Auto-scalability. SMAC3 can support auto-scalability when used with Dask.
Fault tolerance. We found no mechanisms for fault tolerance.
Visualizations. SMAC3 provides no visualization tools natively but is compatible with the tool CAVE (Biedenkapp et al., 2018; Lindauer et al.) developed by the same research group.

C.2.8.2. Algorithms

Bayesian Optimization. SMAC3 is a reference for high quality Bayesian Optimization algorithms.
Bandit. SMAC3 supports Hyperband (Li et al., 2018a) and BOHB (Falkner et al., 2018).
Evolutionary. None.

C.2.8.3. Strategies for efficiency

Conditional. The library `ConfigSpace` used by SMAC3 fully supports conditional search spaces.
Multi-fidelity. SMAC3 supports Hyperband (Li et al., 2018a) and BOHB (Falkner et al., 2018).
Warm-starting. The user can manually pass previous runs to the optimizer with `runhistory` to warm-start it.

C.2.9. Oríon

C.2.9.1. Usability

Languages supported. Any.

Disruptive. Assuming the user script supports hyperparameter definitions using arguments, Oríon only requires adding one line of code. Otherwise, the python API with `suggest/observe` can support most workflows.
Scheduling. Any.
Auto-scalability. (Part.)

Oríon can support auto-scalability when used with Dask.

Fault tolerance. (Part.)

The user can define a number of broken trials that may be tolerated, but broken trials are not re-executed automatically. If a worker crash it is not automatically respawned.
Visualizations. Oríon provides regret, local parameter importance, parallel coordinate plots and partial dependency plots.

C.2.9.2. Algorithms

Bayesian Optimization. A native implementation of Tree-Structure Parzen Estimator (Bergstra et al., 2011) is included in Oríon. Oríon provide wrappers for scikit-optimize and RoBO (Klein et al., 2017). This gives access to Bayesian Optimization with Gaussian Processes and Random Forests, BOHAMIANN (Springenberg et al., 2016), DNGO (Snoek et al., 2015) and ABLR (Perrone et al., 2018).
Bandit. Oríon includes native implementations of Hyperband (Li et al., 2018a) and ASHA (Li et al., 2020).
Evolutionary. The EvolutionaryES algorithm (So et al., 2019) is implemented in Oríon.

C.2.9.3. Strategies for efficiency

Conditional. None.
Multi-fidelity. Hyperband and ASHA implementations in Oríon supports any type of fidelity.
Warm-starting. The Experiment Version Control provides seamless warm-starting under the hood.