

# Novel Approaches For Tuning Models' Hyper Parameters

Antoine Scardigli<sup>1</sup>, supervised by Diego Antognini<sup>2</sup>

<sup>1</sup>EPFL Bachelor Student in Informatic BA5

<sup>2</sup>EPFL PhD in LIA lab

## 1 INTRODUCTION

Hyperparameters are crucial in machine learning, as they determine if the model will be efficient and accurate. Usually, a tuning process is used to find good combinations of hyperparameters. This tuning process used to be very time-consuming as it historically consisted of training the model with all combinations of hyperparameters among the space of all possibilities for grid-search, and of a random set of combinations for random search. Then results were compared to choose the best combination. Now notice that the space of possibilities grows exponentially with the number of hyperparameters. Furthermore, for complex deep-learning problems a single training can take days.

Thus new approaches are needed to find good hyperparameters faster. In this project, we would like to choose a few deep-learning problems as they are usually long to train and compare different tuning approaches <sup>1</sup>. We would like to empirically quantify and qualify the speed for each method as well as the efficiency of the parameters found.

### Project Plan:

1. Choose and describe optimization algorithms that are going to be compared during the project.
2. Implement a benchmark tool to measure the results of various optimization algorithms.
3. Choose a batch of different machine learning problems (more precisely, deep-learning problems).
4. Analyze the results and determine if any approach performs better in average in terms of efficiency or in terms of speed than random search, which is the commonly used algorithm.

## 2 DESCRIBE TRADITIONAL AND NOVEL OPTIMIZATION ALGORITHMS FOR HYPERPARAMETER TUNING

We will in a first time describe current approaches, then main concepts and tools used behind more recent op-

<sup>1</sup>using the [Ray Tune](#) library

timization methods. Finally we will describe novel approaches proposed by RayTune.

### 2.1 Overview of traditional methods

The historical approach was Grid search. It consists of testing all hyperparameters combinations on the validation set to find the combination with the best metrics. Time budget then only depends on the granularity of the combinations tested. As an example, if there are  $d$  parameters, each with  $n$  possibilities, the space of different possibilities/combinations is  $n^d$ . This is exponential in function of the number of hyperparameters, so very inefficient for large  $d$ . Note that grid search is trivially parallel.

The most commonly used approach with grid search for hyperparameters optimization is Random search: It consists of trying randomly some combinations in the space of possibilities which should (should because random) find a decent combination in a reasonable amount of time. It is better than Grid search for the following reasons: If only a small number of hyperparameters  $a$  have a big influence on results out of  $d$  hyperparameters each having  $n$  possible different values, then, GridSearch tests with  $n$  different values each parameter. So there are only  $n^a$  useful trials (combinations where an important parameters changes) out of  $n^d$  different combinations. The proportion of useful tests out of all tests is:

$$\frac{n^a}{n^d} = n^{a-d} \quad (1)$$

That is exponentially inefficient in function of dimensions with low impact. Now random search is going to make all  $d$  parameters change at each trial (since random), so  $n^d$  useful tests since all a dimensions changes for every trial. The proportion of useful training is:

$$\frac{n^d}{n^d} = 100\% \quad (2)$$

A second interesting property about random search is that independently of the shape of the space, random

search is statistically assured to find a good solution if there is one, as long as there are enough trials. Imagine that a proportion of  $x\%$ ,  $x \in \mathbb{R} \setminus \{0 \leq x \leq 100\}$  of the combination space will provide very good results ( $x$  is often very small). We are interested in finding **at least one** solution in this subspace. Now let's define the  $p_x(n)$ , the **probability** that all  $n$  search are **not** in the subspace of interesting results (the subspace is  $\frac{100}{x}$  times smaller than the original space). Since all trials are i.i.d:

$$p_x(n) = \left(1 - \frac{x}{100}\right)^n \quad (3)$$

**Table 1** Table with example values of  $1 - p_x(n)$  with varying values of  $n$  and  $x$ .

Please remember that  $x$  is in percentage of the whole space size. The most distant value we rounded to 1 was 0,99995

$x \setminus n$	10	100	1000	10000
0,01	0	0,01	0,095	0,632
0,1	0,01	0,095	0,632	1
1	0,096	0,634	1	1
10	0,651	1	1	1
100	1	1	1	1

We clearly see on table 1 that  $1 - p_x(n)$  converges to 1 as  $n$  goes large. Thus we find a combination is in the chosen subspace with probability which tends to 1 as  $n$  goes large, this, independently of the subspace shape. Note that random search is also trivially parallel.

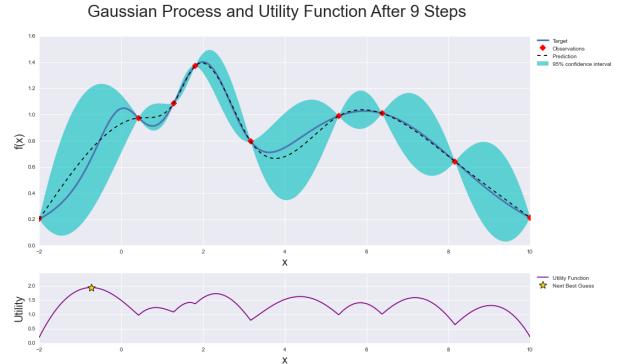
Please look at [Bergstra & Bengio \(2012\)](#) for more details.

## 2.2 Tools and concepts used by the novel tuning algorithms

New kinds of optimization methods have been proposed since a couple of years: The global idea is rather to try different parameters via grid search or random search, then to select a set of parameters in an **adaptive manner**: Once one knows the results of a first set, he shall choose next set smartly. These methods aim to identify good configurations more quickly. We will here explain and describe [all approaches](#) proposed by the Ray Tune python library.

But first, as they will be widely used in the described approaches, let's explain some of the main tools and concepts used in those different optimization methods.

- **SHA**: SuccessiveHalvingAlgorithm has a very intuitive concept: given  $n$  combinations, it trains all  $n$  models, select the best  $\frac{n}{2}$  (sometime  $\frac{n}{3}$ ), and then starts again with the  $\frac{n}{2}$  best combinations. The best (last) model is found in  $2 \times n$  trainings as  $\sum_{i=0}^{\infty} \frac{1}{2^i} = 2$ . Grid search is in only  $n$  trainings, but the



**Figure 1.** Here is a BO process, see [Gamboa \(2019\)](#).

strength of SHA is that the time allocated for a training can be much smaller since best models are going to get more and more training time. SHA's conceptual interest is to stop the training of model with low potential. This is one of the simplest adaptive algorithm. See [Li et al. \(2017\)](#).

- **ASHA**: Stands for AsynchronousSuccessive-HalvingAlgorithm . It allows parallelisation and asynchronisation of SHA, hence permitting a trivial parallelisation of HyperBand. See [Li et al. \(2018\)](#).

- **MSR**: Median Stopping Rule is very close conceptually to SHA. It simply stops a trial as its performance is below the median performance of all other trials after an equal training. It suffers the same problem as SHA: it needs to train all models at least once, which can be very long.

- **BO**: Bayesian Optimisation uses the principle of Bayesian inference: it exploits all observed trials results to infer the probability results of not yet observed trials. The probability representation of the objective function using previous observations is called the **surrogate**.

*It often is a Gaussian process* , as it is very useful to guess the values of unexplored trials: For a distribution of the result estimation  $N(\mu, \sigma)$ , if the  $\sigma$  is large, then exploration is interesting as an unknown maximum could probably be there, and if the  $\mu$  is big, then we can exploit: it could be near the maximum. The dilemma is resolved by taking the **maximum** of the potential function

$$f(x) = \mu(x) + k \times \sigma(x) \quad (4)$$

where  $k$  is the weight of exploration and  $x$  is a combination of hyperparameters.

We can observe in Figure 1 a Bayesian Optimisation in process, with the blue curve being the real

function, the black one the predicted function, and the blue area the incertitude about the predicted function. On the bottom in purple is the potential function.

Gaussian process is a logical surrogate function: it updates predictions after new observations, and chooses to observe next the combination with the best potential. It tries to **maximize**  $p(P|O)$ , the probability of having a good prediction given observations. See [Balandat et al. \(2019\)](#).

*Other surrogate functions exists such as the Tree-structured Parzen Estimator (TPE) :* using Bayes rule

$$p(P|O) = \frac{p(O|P) \times p(P)}{p(O)} \quad (5)$$

It is interested in  $p(O|P)$  and  $p(P)$  to find  $P(P|O)$  rather than directly in  $P(O|P)$ .

Here is the TPE algorithm explained: After a random initialization, models are separated in two groups, best performing ones, and the others, and let  $y$  be the threshold. Then let's have densities  $l$  and  $g$  from Parzen estimators's model such that

$$p(o|P) = \begin{cases} l(o) & o < y \\ g(o) & o \geq y \end{cases} \quad (6)$$

o an observation. The expected improvement if  $x$  is chosen as next observed model then is  $\frac{l(x)}{g(x)}$ , so we have to choose  $x$  that maximises  $\frac{g(x)}{l(x)}$ . See [Bergstra et al. \(2011\)](#) and [Falkner et al. \(2018\)](#).

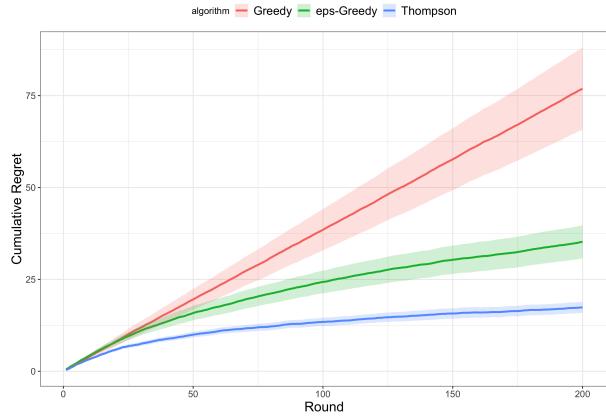
Note that the same mechanisms can be used to predict the expected remaining gains (if we continue the algorithm), hence Bayesian optimisation allows early termination. It also allows parallelism, but as a trade off for performances, as the model will have more information at any point in sequential mode than in parallel mode.

- **Multi-armed Bandit Problem:** This optimisation problem consists of optimizing gains when  $K$  cash machines are available, and it is only possible to use one cash machine per turn. The interest of this problem is to find the trade-off between exploration: trying an untested cash machine with the hope of discovering an even worthier one, and the exploitation: choosing the yet discovered best cash-machine. We define the regret as the difference between the reward sum in case of an optimal strategy and the sum of collected rewards. When we speak of optimizing hyperparameters, cash machines are models, and gains are efficiency results of the models. Here are some known algorithm which tries to find the best trade-off:

The simplest algorithm "greedy" consists of maximising exploitation: always choose the model with best performances.

Another simple strategy is the **epsilon first strategy** :  $\times N$  steps of random exploration followed by  $(1-\epsilon) \times N$  steps of exploitation of the best yet discovered models. It is very close to the epsilon strategy: probability of  $\epsilon$  of exploring new models and  $1-\epsilon$  of exploiting.

The **Thompson sampling algorithm** : The intuition is that you select a combination of hyperparameters with a probability proportional to that combination being the best. This adaptive algorithm is very similar to **SHA** or **MSR**, intuitively, we invest trainings in models with high potential, which is going to confirm or infirm quickly and with strong evidence that it indeed has strong potential. We can see in Figure 2 the strong advantage of finding a good optimization method. See [Dimmery et al. \(2019\)](#).



**Figure 2.** We can see that regret is clearly sub-linear for Thompson algorithm [5]

- **PBT**: Population-Based Training trains  $n$  models in parallel with different random combinations. Once all models are trained with budget  $B$ , they are tested and models with bad results copy the hyperparameters of most performing models plus some noise, intending to outperform best models by finding even better combinations. This is the only approach that modifies hyperparameters of models during training.

Summing up, it first starts by creating  $n$  models in parallel with random hyperparameters. Then it starts a **while Loop**. It trains all  $n$  with budget  $B$ , then only the best  $a$  models are kept, and the  $n-a$  other take hyperparameters close to the best models. Back to **Loop** until convergence. This algorithm is promising as it is similar to

random search with the difference it reallocates training budget smartly in zones with high potential.

- HyperBand. This method is using a bandit based optimisation: The problem of SHA is that training all  $n$  combinations can be much too long. If you only have a budget  $B$  to spent all trainings, Hyperband proposes to evaluate some random subsets  $m$  of combinations, and, using the SHA algorithm, to spent budget  $\frac{B}{\log(m)}$  for  $i$  from 0 to  $\log(m)$  randomly among all  $\frac{m}{2^i}$  models, as long as the budget is at least  $r$  and at least one model is trained with budget at least  $R$ ,  $R$  is a random input of the program ( $R > r$ ) , and  $r$  a function of  $R$  and  $i$ . The complexity then is  $B \times \log(R)$ .

Intuitively if a model requires a long time before having converging results, we would prefer a small  $n$ , to give enough budget to each model so it has relevant results. Equivalently, if the models are not noisy and train quickly, we want  $n$  to be as large as possible. HyperBand finds the solution to the latter Bandit problem by proposing random  $R$  and random sets of configuration  $m$  in a SHA with limited budget  $B$ . It balances very aggressive evaluations with many configurations with small budget, and very conservative evaluations with bigger budgets. It has infinite arm since the space of hyperparameters combinations needs to be infinite, which is usually the case (continuous). It also provides early stopping, meaning the algorithm stops as soon as it does not notice any significant improvement. Please see [Li et al. \(2017\)](#).

### 2.3 Description of Tuning algorithms proposed by Ray-Tune

- AxSearch: Ax can optimize continuous configurations (integer included) using [Bayesian optimization](#) improved using matern kernel to manage very noisy models. It optimizes discrete configurations (only several choices) as a [multi-armed bandit problem](#), solved using the Thompson sampling algorithm and empirical Bayes. We shall not use this latter algorithm as it is an optimisation algorithm for a small number of possibilities, opposed to deep-learning problems which could have a lot of combinations of hyperparameters. [Balandat et al. \(2019\)](#), [Dimmery et al. \(2019\)](#).
- DragonflySearch uses Bayesian optimisation adapted for expensive large scale problems with high dimensional spaces. It also provides a parallel approach. The main idea is the following: it

is known that [BO](#) is very successful for low dimensional models ( $d < 10$ ). And it is said to be exponentially difficult in function of dimensions. So the dragonfly algorithm is going to decompose the result function

$$g(x) = g_1(x_1) + g_2(x_2) + \dots + g_n(x_n) \quad x_i \in X_i \quad (7)$$

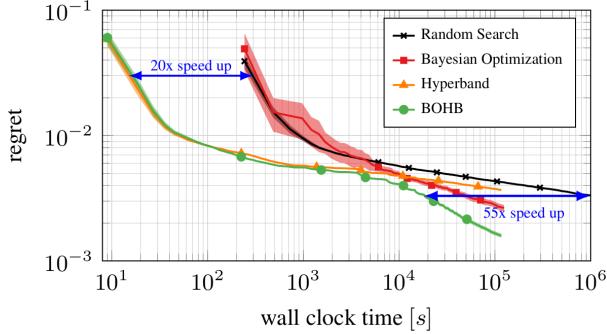
where  $X_i$  is a set of very low dimensionality randomly chosen such that every  $X_i$  is disjoint with any  $X_j$ , and all  $f_i$  is independent with any  $g_j$ , and  $g_i$  are considered as sampled from a Gaussian process. Then it is proved that the overall potential function described in our paragraph about [BO](#) is simply the sum of each potential function, resulting in a huge performance improvement. That is:

$$\begin{aligned} f(x) &= f_1(x_1) + f_2(x_2) + \dots + f_n(x_n) \\ \text{et } f_i(x_i) &= \mu_i(x_i) + k \times \sigma_i(x_i) \end{aligned} \quad (8)$$

One now simply has to observe the maximum of  $f$ . See [Kandasamy et al. \(2020\)](#).

- SkoptSearch is using regular BayesianOptimizationSearch. [Louppe & Kumar \(2016\)](#).
  - HyperOptSearch performs especially well in case of mixed search space, which may include both continuous, discrete, and conditional dimensions. It is using BO with either a Gaussian Process or the Tree-structured Parzen Estimators algorithm as surrogate function. See [Bergstra et al. \(2011\)](#).
  - BayesOptSearch is using regular BayesianOptimizationSearch as well.
  - TuneBOHB is named after Bayesian Optimisation concatenated with HyperBand. The interest is to take the best of the two worlds as [Hyperband](#) has strong performance and scalability, but samples configurations randomly (without learning from previous observations, although BO is very good at this). It uses Bayesian optimisation with TPE instead of Gaussian process as it would scale better in high dimensional space.
- [BOHB](#) uses HB algorithm to determine the number and the budget of each configuration, but replaces the random selection of combinations by a BO algorithm for most samples (with probability  $1 - \rho$ ) based on all observations so far. the combination is still selected with probability  $\rho$  at random. It can use the parallel properties of both TPE and HB to provide efficient parallelism. It is said by his authors to always beat or match both methods.
- We can observe on [Figure 3](#) how TuneBOHB is always better than both random search, BO, and

HyperBand. See [Falkner et al. \(2018\)](#).



**Figure 3.** Comparative plot of HB, BO, BOHB and random search for a deep learning problem according to [11]

- NevergradSearch: this Facebook open project proposes several algorithms. Here is a selection of three [evolutionary algorithm](#):

*TwoPointsDE* : Differential Evolution algorithm is the following:  
First  $n$  models are trained with random combinations.  
Then loop until convergence or end of time budget:  
(1) For every models  $x$ ,  
(2) choose 3 other distinct models  $a, b, c$ .  
(3) Then for any dimension of  $x$ , keep same value with high probability, and take a combination of  $a, b, c$ 's value otherwise (mutation), in a way that at least one dimension changes.  
(4) Then keep the best model between old  $x$  and new  $x$  (selection).

This is the historical algorithm. Facebook's algorithm only differs from the facts there are only 2 others models  $a$  and  $b$  instead of  $a, b, c$ . See [Storn & Price \(1997\)](#).

*CMA* : This algorithm uses two different concepts: First exploration: by updating the average of the distribution in a way that maximizes the likelihood of actual good models. The Covariance matrix is incremented to increase the likelihood of previous search steps.

The second concept is that it adaptively increases or decreases the variance depending on last steps. If the model is in an exploration phase, variance will increase until it finds the best combinations, but if we are close to best results (exploitation), the variance becomes very small, hence allowing fast convergence. See [Hansen \(2006\)](#).

*PSO* : This algorithm imitates birds or ants movement. First initialize  $n$  models with random combination of hyperparameters; let's call it position in the  $d$ -dimension space, and assign to each model an additional information: it's best position so far. Let  $g$  be the best position of all models so far. Then on each loop every model is going to have on every dimension a speed, functions both of its best position times a normal distribution sample and of the best position of the swarm times a normal distribution sample. Then update the position as last position + speed. See [Kennedy & Eberhart \(1995\)](#).

- ZOOptSearch: its particularity is that is focused on derivative-free optimization.  
The goal is to find the minimum of a function  $f$  without having any other informations than  $f(x)$  (performance of combination  $x$ ) given  $x$ .

*The classification* model will classify combinations as either good ( $k$  best) or bad, by using a parallel to axis hyper-rectangle, of same dimension as the combination space, that contains all good combinations. Then **loop** on the following: (1) Choose a combination either randomly from the  $d$  dimension rectangle with probability  $\lambda$ , or from the whole space with probability  $1-\lambda$ , (2) evaluates its efficiency and (3) replace the worst of the  $k$  best solution with this new solution, then (4) replace a random bad solution with the worst of the  $k$  solutions (the one just replaced).

Finally (5) update the classifier and go back to loop.

The asynchronous process is accomplished by parallelising the loop on different threads, but it needs to update the model after each evaluation. [Liu et al. \(2019\)](#).

- SigOptSearch: What they are doing is not clear at all for me, I have not been able to find any concept they were using.

**Table 2** Characteristic of all approaches.

Tuning Algorithm	Parallelism	Adaptive algorithm	Efficient despite ineffective parameters	Early termination	Main Concept
GridSearch	✓				Extensive search
RandomSearch	✓		✓		Random search
AxSearch		✓	✓	✓	BO or Bandit Based
DragonflySearch	✓	✓	✓	✓	BO
SkoptSearch		✓	✓	✓	BO
HyperOptSearch	✓	✓	✓	✓	BO with TPE
BayesOptSearch		✓	✓	✓	BO
TuneBOHB	✓	✓	✓	✓	BO and HyperBand
NevergradSearch		✓	✓		Population based
ZOOptSearch	✓	✓	✓		Zeroth Optimization (ASRACOS)
SigOptSearch	✓	✓	✓	✓	Unknown

## 2.4 Discussion

In a nutshell, we can regroup the algorithms by the concept they use: *AxSearch*, *DragonflySearch*, *SkoptSearch*, *HyperOptSearch* and *BayesOptSearch* uses Bayesian Optimisation, except *HyperOptSearch* uses TPE as a surrogate function, *dragonflySearch* is specialized for large scale problems (in case of high dimensional optimisation, it provides improved parallelism, and cheap approximations functions), and *AxSearch* also proposes a Bandit based solution in case of discrete spaces of very small size. *NevergradSearch* is the only population based optimizer which is promising. *ZooptSearch* proposes a derivative-free optimization. Finally *TuneBOHB* uses both concepts from Bayesian optimisation and Bandit-Based optimization, which is intuitively powerful as they complete each other's weakness. You can find in Table 2 a sum up of informations related to all algorithms

We decide to consider from now on only one optimization algorithm of each kind. That is: *DragonflySearch*, *AxSearch*, *SkOptSearch*, *NevergradSearch*, *ZooptSearch* and *TuneBOHB*.

These optimization algorithms will be compared between them and with *RandomSearch* and *GridSearch* by studying the metrics we will talk about in next section.

## 3 BENCHMARK TOOL

### 3.1 metrics

Before implementing the benchmark tool, we would like to think about the comparisons metrics we will use to compare all algorithms. It is mostly about comparing either test error or test accuracy as a function of time or trials:

- One of the most common approach is to compare the test error/accuracy of the best set of hyperparameters chosen by every optimization algorithm as a function of total number of trials/the time needed to find these hyperparameters.
- An enhancement of the protocol above would be to divide the number of trials into several occurrences, to have test accuracy of the best trials, but also the distribution of performance as a boxplot. with lower and upper quartiles, whiskers above and below to show the position of the most extreme data point, two black lines across the top of the figure to mark the upper and lower boundaries of a 95% confidence interval. [Bergstra & Bengio \(2012\)](#)
- Finally, an other approach is to use metrics from the validation set instead of the test set. The idea consists of finding the expected best validation accuracy (the average result from the best validation trial) in function of the number of assignments. The limitation of this method clearly is over fit, which is hopefully infrequent in NN. [Dodge et al. \(2019\)](#) also informs about the variance by adding an area  $\sigma$  wide around the validation accuracy.

We will process our data the following way: we define as best trial for trial number  $n$ , the trial with best validation accuracy for all trials from  $\theta$  to  $n$ .

This means the plot for validation value should always be increasing for classification accuracy (decreasing for Regression), but plot for test value could be decreasing (increasing for regression) because a new trial with

a better validation accuracy could have a worst test accuracy.

We are only interested in the yet best trial because what we want from a search algorithm is that it gives the best hyperparameter combination the fastest possible. So we are interested in the best test accuracy over all yet trained trials, model as a function of the number of trials done.

### 3.2 Toy Benchmark

We implement a first version of our benchmark tool. This version is very extensive as we allow various combinations for tuning optimization. To compensate with this extensiveness, we choose some very simple datasets, and do very few iterations (max 20 epochs). This toy benchmark has two goals: First to verify that every functionality works, that models are indeed learning... and second to understand which factors are important. We could then focus on the important factors with some harder datasets as a second batch. We conduct a large scale optimizer benchmark, with different datasets, and different labeling categories, as search algorithm's performances can highly depend on the test problem. Here is a short description of every labels:

**Model Inputs:** We wanted to have at least an input category of every kind to perform an extensive comparison, so we have both Image, Table, and Text input.

**Model Output:** Similarly, we seek extensiveness so we propose classification, regression and generation. Classification is the task of putting the input in one class, regression is the task of finding one or few real values from the input, and generation consists of creating credible output data looking like the input data, despite being new.

We don't do all combinations here by lack of time, but we do:

Image classification with two datasets: MNIST and Fashion MNIST. These are about finding back the digits and the cloth category respectively from a black and white image of low quality.

Image generation with the MNIST dataset, so here it is about creating back some images of digits.

Table Regression with both boston and diabete datasets. These are about finding the value of houses and the diabete progression measure respectively from a table of datas.

Text Regression is using the Amazon purchases comment reviews to predict a satisfaction score between 0 and 1.

Text Classification is using the IMDB dataset: a huge movies review dataset to predict if the writer either liked or didn't liked the movie.

**Models:** We have a lot of different models, still for extensivness. Unfortunately, we can not use every model

with every input category. As an example CNN, which pre-treats the input in smaller parts, is usefull for images and text, as the model has to extract the informations, but is useless in tables, where the information is already given.

Here we will just enumerate the particularities of every model.

CNN is the most complicated: On the one hand there is a generator that tries to create outputs looking like inputs, and there is a discriminator, that tries to make the difference between the real inputs and the outputs of the generator. The generator is trained by whether or not it succeeds to fool the discriminator, and the discriminator by whether of not he successfully discriminated the given input. Here we only used MNIST as it is a very simple dataset. We used pre trained data to have more interesting outputs. The metric is the Is-Score which quantifies how good is the generation.

CNN is used for visual and text datasets as they first pre-process the data to reduce its dimensionality through pooling, or by performing convolution, which consists of not taking into account a single value, subset of the whole input (for example a pixel for an image), but to also take into account near values.

RNN's particularity is that the connections of it's nodes form a directed graph. This allows it to have memory. It is hence very indicated for texts datasets as we would like to remember the influence of every word when going through the sentence.

We do the distinction between two RNN models:

LSTM stands for Long Short Term Memory and has three gates: "forget gate" adds new input to stored information and decide if some need to be forgotten, "input gate" pre-processes the data in function of it's importance to calculate the cell value, and then "output gate" decides which informations should be given to next node (memory).

GRU has only two gates: first a "reset gate" that decides which previous informations (important previous words in a sentence) should be kept, and then an "update gate" that decides if a new information should be kept in memory or not (if new word provides important information). MLP is a regular feedforward neural network with multiple layers and non linear activation functions.

Logistic Regression is a very simple model, the most likely to underfit as it is a single layer neural network with logistic function. We will use it for table regression.

**Optimizer:** Finally, the optimizer is the thing that makes the model train by updating the network weights. The most know is SGD: Stochastic Gradient Descent. It maintains a single and fixed learning rate for all weight updates. Adam proposes to keep the principle of gradient Descent, but allows a per-parameter learning rate, and also allows a learning rate evolution, as training in the begining is often to slow. In our benchtest, it will also manage:

The activation function, which defines output of a node given an input. We often want it to be continuous and differentiable.

The LR decay controls how the learning rates should be able to change.

The LR dropout will randomly set some LR to zero to avoid bad local minima.

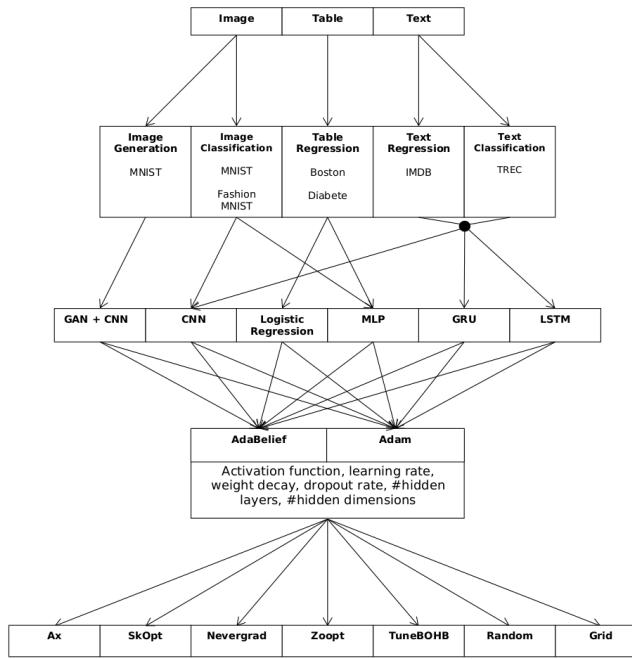
The number of layers is important to avoid overfit, underfit, and to have the best results.

And finally the dimensions of hidden layers.

We will compare this optimizer with **AdaBelief**, which is new and promising. It works like Adam, but improves one case: In the case of a large gradient and a small variance, Adam would use a small learning rate, although AdaBelief would keep a big learning rate by noticing that variance is small.

We use the same Trial scheduler for every combination: it is **ASHA**, as it is the scheduler recommended by ray-tune, being the most compatible. It stands for Asynchronous Successive HALving. So it is a parallelized version of **SHA**, which stops trials that are not goods enough. Finally we wish to present the different ways that exists to calculate the metrics. See Table 3.

As one picture says more than a thousands words, you can simply look at figure 4 to understand how the toy benchmark works. As a sum up we have several layers which are respectively: model input, model output, NN category, problem category, optimizer, Search algorithm. We also gave the same trial scheduler to all problems but BOHB: it is the ASHA scheduler.



**Figure 4.** plan of the toy benchmark and all the combinations it allows

#### Notes on the benchmark:

**Table 3** Here  $y$  is a label data, and  $\bar{y}$  is the output of the model.

We also define  $T_p$  the true positive,  $T_n$  the true negative,  $F_p$  the false positive and  $F_n$  the false negative.  $D_{KL}$  is the Kullback–Leibler divergence.

name	formula	best	worst
<b>classification</b>			
accuracy	$\frac{T_p + T_n}{n}$	1	0
precision	$\frac{T_p}{T_p + F_p}$	1	0
recall	$\frac{T_p}{T_p + F_n}$	1	0
f1	$\frac{2 * precision * recall}{precision + recall}$	1	0
log_loss	$-(y * \log(\bar{y}) + (1 - y) \log(1 - \bar{y}))$	0	$+\infty$
<b>regression</b>			
MAE	$\frac{\sum_i  y_i - \bar{y}_i }{n}$	0	$+\infty$
MSE	$\frac{\sum_i (y_i - \bar{y}_i)^2}{n}$	0	$+\infty$
RMSE	$\sqrt{MSE}$	0	$+\infty$
R2 score	$1 - \frac{MSE}{\sigma^2}$	1	$-\infty$
<b>Generation</b>			
Inception Score	$e^{E[D_{KL}(p(y \bar{y}) * p(\bar{y}))]}$	#Class	1
Also all classification metrics from generated output			

After testing this benchmark, we noticed we were not able to make Dragonfly work, which is regrettable. The goal here is to compare the difference of results between each search algorithm, and especially to see if learning search algorithms can provide significantly better results than random search in significantly shorter time. So we are interested in the relative difference between results of every search algorithms. It is normal that we have results below the "good" results in the literature because we don't train on a lot of epoch.

We end up with an 8 dimension parameter space:

- Learning rate: from 10e-4 to 0.1.
- Weight decay: from 10e-4 to 0.1.

- Sigmoid function: either relu, tanh, or sigmoid.
- Number of hidden dimensions: from 32 to 256.
- Number of hidden layers: from 1 to 3.
- Dropout probability: from 0 to 0.5.
- Optimizer: either Adam or AdaBelief.
- Model: which neuronal model will learn.

We will do a detailed analysis on one dataset of each category in this section, followed by an global analysis over all datasets. It is still possible to see in Appendix A an analysis per dataset for datasets in redundant categories.

We also noticed that several search algorithm could not manage log distribution nor discrete distribution. To be fair, we then decided to have parameters distribution as uniform and continuous (this was managed by all search algorithms), this should not be an issue as it is the job of the search algorithms to find good values despite bad distribution.

Although we have 3 population based algorithm, they are all using NeverGrad, so as this is only the first benchmark we will only use one of them to reduce information overload (CMA ), but for second benchmark we will compare all 3.

### 3.3 Detailed analysis on boston dataset

For the detailed Analysis on boston dataset, we give 256 trials to every search algorithm (so 256 different combinations of hyperparameters). Each of these trials can learn on up to 20 epochs, but it can also be stopped before by the scheduler (all search algorithms have the same scheduler but BOHB). The scheduler stops a trial if it stopped learning, or if it gives very bad results in the first epochs.

We repeated this process 4 times so we reduce influence of noise, and also so we can quantify variance.

Our analysis will rely on the metrics we presented in section 3.1.

To sum up, these metrics were:

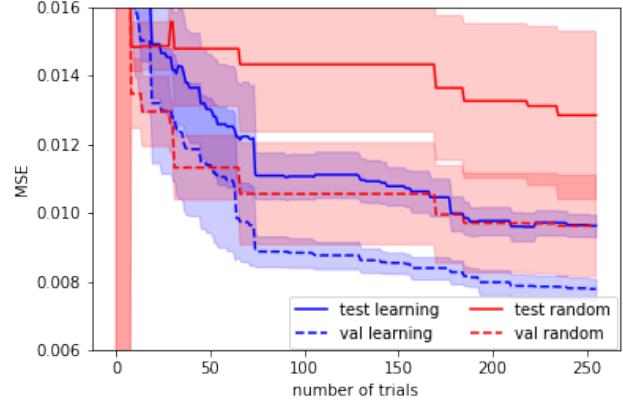
1. Plot of the best test MSE, ie from best hyperparameter combination found so far, as a function of number of trials / time.
2. Plot of the best test MSE from best hyperparameter found so far, as a function of number of trials / time, including boxplot to have interesting informations about variance, quartiles, and outbound values.
3. Plot of the best expected validation accuracy from best hyperparameter found so far, as a function of number of trials, including variance information.

We can plot all of these informations on a single figure per algorithm. Note we decided to use a variance area for 2. instead of boxplots for obvious visibility reasons.

#### 3.3.1 General Informations

---

##### Random Search versus all other algorithms on Boston Dataset



**Figure 5.** Here we plot the averaged test and val MSE of Random Search, and we compare it with the mean of all other algorithms (learning algorithms). Color areas are  $\sigma$  wide

As an introduction figure, we show in figure 5 the comparison between the random search test and validation MSE, with the mean of all other algorithms's test and validation MSE.

Note that test MSE is the value we are interested in in real life, but here we want to compare the search algorithms, so it is interesting to compare validation accuracy as well as it is the metric that optimizer, scheduler, and especially search algorithm use to learn and decide new hyperparameter combinations.

We were afraid that plotting boxplots would cause information overload, so we will add color areas that are  $\sigma$  wide instead.

As we predicted, validation MSE is always decreasing, but test MSE is sometimes increases (a little).

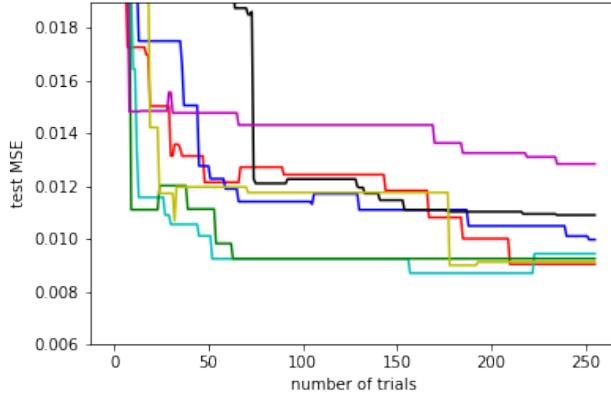
Also we notice that validation MSE is always below test MSE, this is because the model is selected on his validation MSE, so there is a positive bias on it.

#### 3.3.2 Test and validation MSE

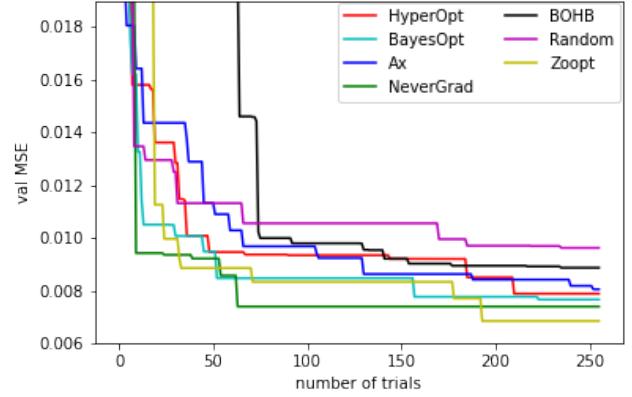
We see in figure 6 that random-search has a test MSE that is nearly two times worst than all other learning algorithms.

Also we learn that most of the improvement happens in the 64 first trials. We also see in figure 9 that random search has as bad results for validation MSE as well. These two plots permit an overall comparison. NeverGrad, Zootp, and Bayes seems to be the best, and Random is the worst by far.

#### 3.3.3 Variance per Search Algorithm

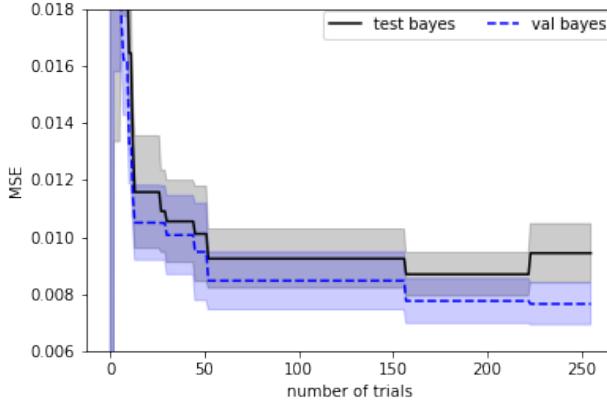
Test MSE

**Figure 6.** Here we plot the average test loss of the best trial obtained so far per search algorithm as a function of the number of trial. This is the plot presented as 1.

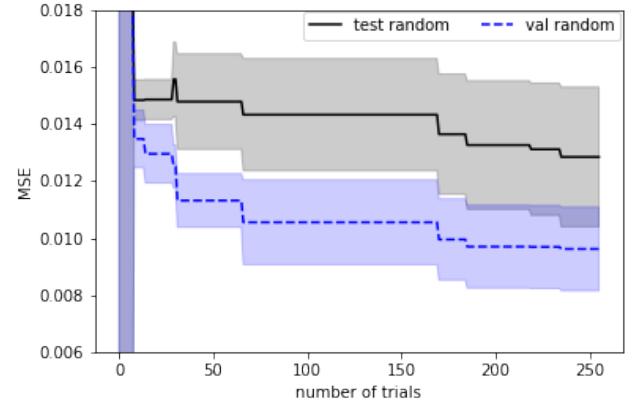
Validation MSE

**Figure 9.** Here we plot the average validation loss of the best trial obtained so far per search algorithm as a function of number of trial.

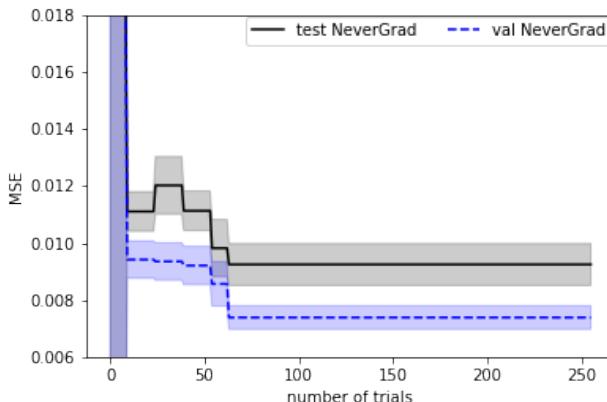
#### Tests and Validation MSE per algorithms with $\sigma$ narrow area on Boston Dataset



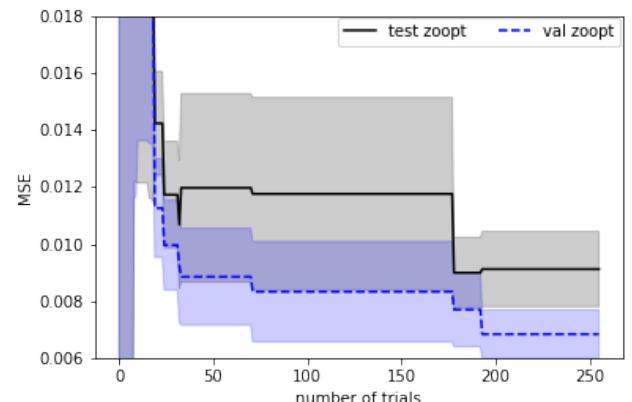
**Figure 7.** Validation, Test, Variance MSE with Bayes



**Figure 10.** Validation, Test, Variance MSE with Random



**Figure 8.** Validation, Test, Variance MSE with NeverGrad



**Figure 11.** Validation, Test, Variance MSE with Zootp

To reduce information overload, we will plot the recommended metrics only for the top 3 performing algorithms and random search. All axis are the same. See in figure 7 the plot for BayesOpt, in figure 8 for NeverGrad, in figure 10 for Random, and in figure 11 for Zoot.

We also understand from this data that validation loss has similar signification than test loss as the difference between the two is close for all algorithms.

We see that random and zoot have the biggest variance. and that variance is otherwise pretty low. There is a huge variance during the first 5 trials, but NeverGrad finds very quickly a very good solution (after 64 trials). Random doesn't converge and keep improving slowly up to the 256 trials).

Variance for random search is huge, thus is expected as it is random by definition so the best trial is subject to a maximized variance. This is not desired as it means we could be unlucky and have very bad results despite searching for a lot of trials. This is another reason why NeverGrad is preferable as it has a very small variance.

### 3.4 Detailed analysis on TREC dataset

For the detailed Analysis on TREC dataset, we give 128 trials to every search algorithm. Each of these trials can learn on up to 20 epochs.

We repeated this process 5 times so we reduce influence of noise, and also so we can quantify variance.

#### 3.4.1 General Informations

As an introduction figure, we show in figure 12 the comparison between the random search test and validation accuracy, with the mean of all other algorithms's test and validation accuracy.

We observe that validation accuracy is way above test accuracy for all search algorithms. Also, Random actually gives similar results than the other search algorithms.

#### 3.4.2 Test and validation accuracy

We see in figure 13

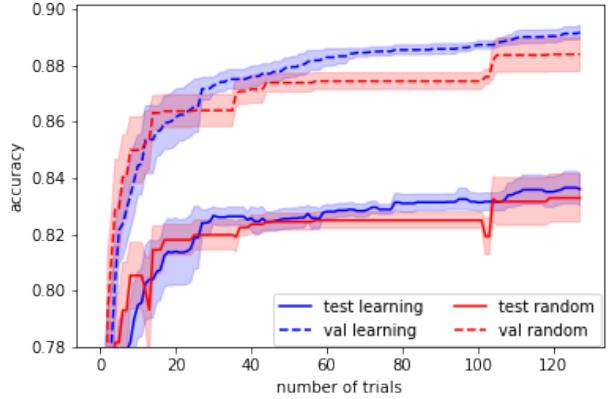
We also see in figure 16

These two plots permit an overall comparison. NeverGrad, Zoot, and Bayes seems to be the best, and Random is the worst by far.

#### 3.4.3 Variance per Search Algorithm

To reduce information overload, we will plot the recommended metrics only for the top 4 performing algorithms. All axis are the same.

Random Search versus all other algorithms  
on TREC dataset

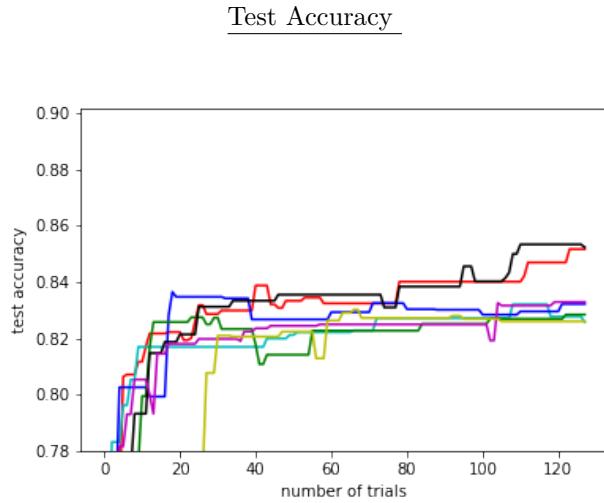


**Figure 12.** Here we plot the averaged test and val accuracy of Random Search, and we compare it with the mean of all other algorithms (learning algorithms). Color areas are  $\sigma$  wide

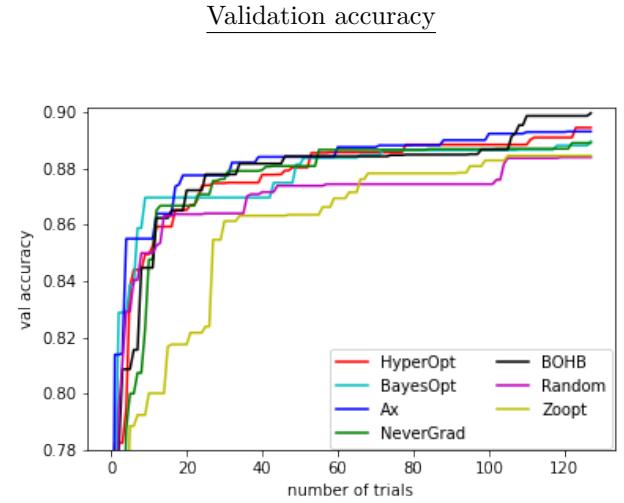
See in figure 14 the plot for HyperOpt, in figure 15 for Ax, in figure 17 for BOHB, and in figure 18 for Random.

The reason why random is so good is maybe because the dataset is so simple. Then even mediocre configurations would not have so big accuracy difference with very good configurations.

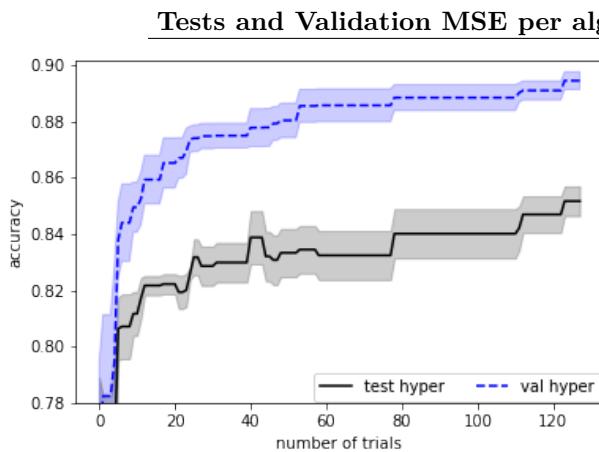
Also note that the difference between test and validation accuracy is very high here, so difference of results of accuracy among search algorithms seems relatively smaller. What we mean here is that there is still a 3% test accuracy difference between the best algorithm (BOHB) and the average of the other algorithms, but this improvement seems small compared to the 5% gap between test accuracy and validation accuracy.



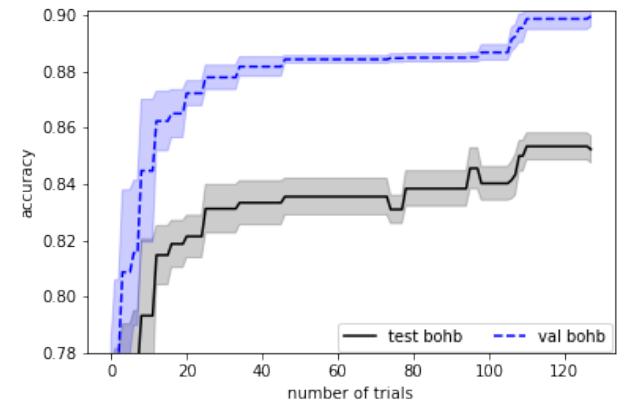
**Figure 13.** Here we plot the average test accuracy of the best trial obtained so far per search algorithm as a function of the number of trial.



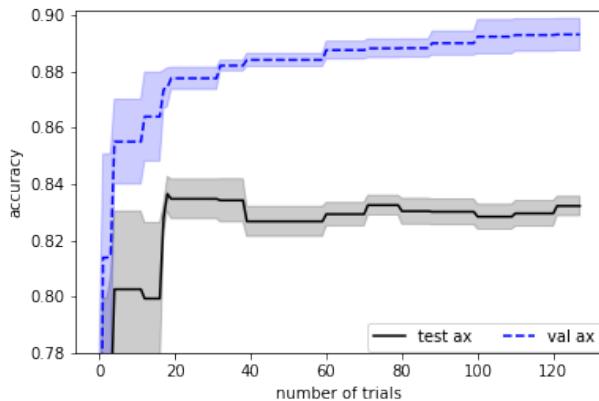
**Figure 16.** Here we plot the average validation accuracy of the best trial obtained so far per search algorithm as a function of number of trial.



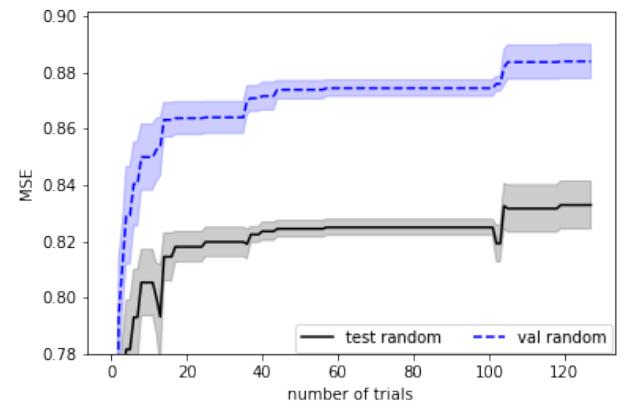
**Figure 14.** Validation, Test, Variance accuracy with Hyper



**Figure 17.** Validation, Test, Variance accuracy with BOHB



**Figure 15.** Validation, Test, Variance accuracy with Ax



**Figure 18.** Validation, Test, Variance accuracy with Random

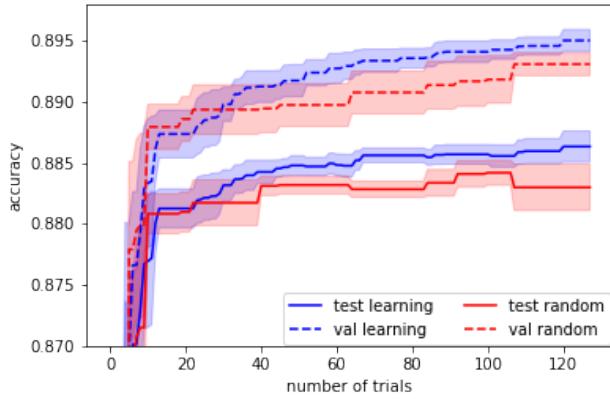
### 3.5 Detailed analysis on FMNIST dataset

For the detailed Analysis on FMNIST dataset, we give 128 trials to every search algorithm. Each of these trials can learn on up to 20 epochs.

We repeated this process 4 times so we reduce influence of noise, and also so we can quantify variance.

#### 3.5.1 General Informations

##### Random Search versus all other algorithms on FMNIST Dataset



**Figure 19.** Here we plot the averaged test and val accuracy of Random Search, and we compare it with the mean of all other algorithms (learning algorithms). Color areas are  $\sigma$  wide

As an introduction figure, we show in figure 19 the comparison between the random search test and validation accuracy, with the mean of all other algorithms's test and validation accuracy.

We observe that validation accuracy is way above test accuracy for all search algorithms. Also, Random actually gives similar results than the other search algorithms.

#### 3.5.2 Test and validation accuracy

We see in figure 34

We also see in figure 37

These two plots permit an overall comparison. HyperOpt seems to be the best by far, followed closely by BayesOpt and BOHB. Random is clearly the worst.

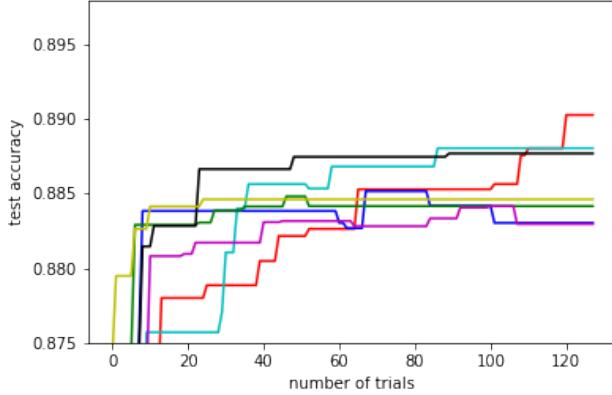
#### 3.5.3 Variance per Search Algorithm

To reduce information overload, we will plot the recommended metrics only for the top 4 performing algorithms. All axis are the same.

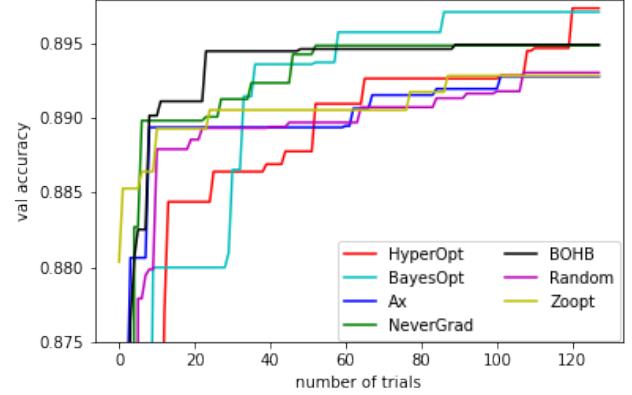
See in figure 35 the plot for HyperOpt, in figure 36 for Bayes, in figure 38 for BOHB, and in figure 39 for Random.

The reason why random is so good is maybe because the dataset is so simple. Then even mediocre configurations would not have so big accuracy difference with very good configurations.

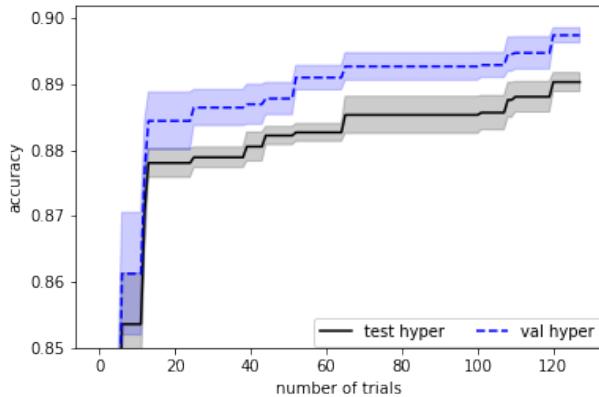
Also note that the difference between test and validation accuracy is very high here, so difference of results of accuracy among search algorithms seems relatively smaller. What we mean here is that there is still a 3% test accuracy difference between the best algorithm (BOHB) and the average of the other algorithms, but this improvement seems small compared to the 5% gap between test accuracy and validation accuracy.

Test Accuracy

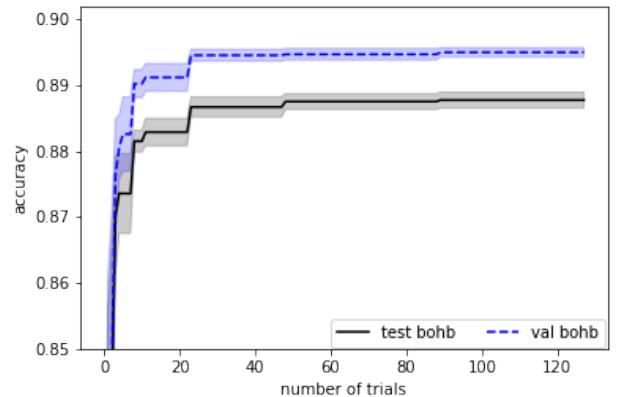
**Figure 20.** Here we plot the average test accuracy of the best trial obtained so far per search algorithm as a function of the number of trial.

Validation accuracy

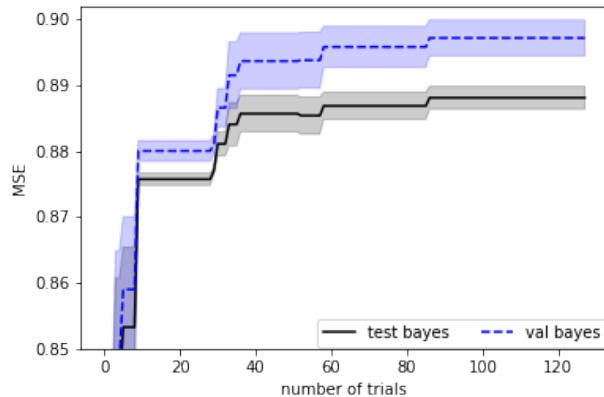
**Figure 23.** Here we plot the average validation accuracy of the best trial obtained so far per search algorithm as a function of number of trial.

Tests and Validation MSE per algorithms with  $\sigma$  narrow area on FMNIST Dataset

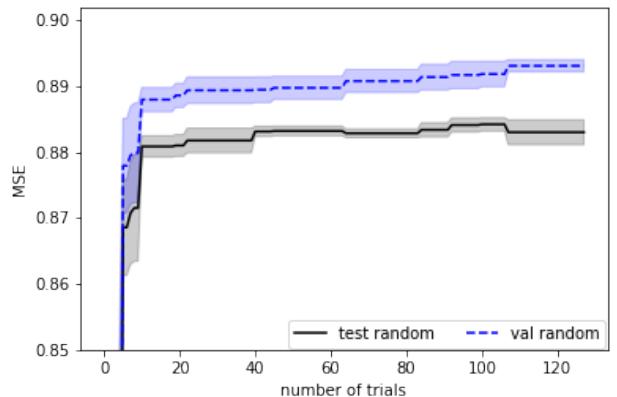
**Figure 21.** Validation, Test, Variance accuracy with Hyper



**Figure 24.** Validation, Test, Variance accuracy with BOHB



**Figure 22.** Validation, Test, Variance accuracy with Ax



**Figure 25.** Validation, Test, Variance accuracy with Random

### 3.6 Detailed analysis on GAN with MNIST dataset

For the detailed Analysis on IMDB dataset, we give 128 trials to every search algorithm. Each of these trials can learn on up to 4 epochs.

We repeated this process 4 times so we reduce influence of noise, and also so we can quantify variance.

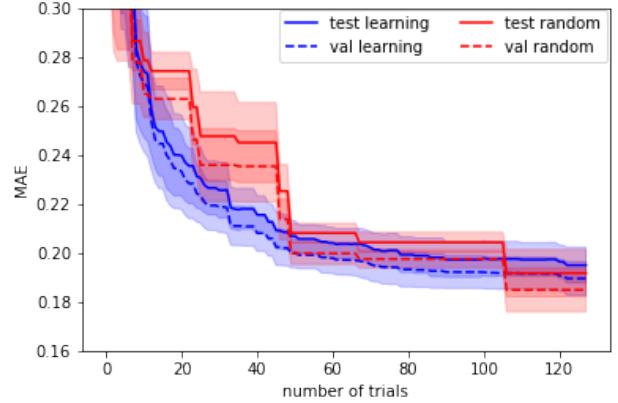
Gan are very different to regression or classification tasks as we did in the rest of this benchmark. We think it is important to have also a very different problem to see how the search algorithms will perform in a completely different context: Indeed we changed the search space: it is now about optimizing the learning rate, the first beta parameter, the weight decay, and the optimizer for two different models: the generator and the discriminator. Following the same philosophy, we used a different scheduler: we used a population based scheduler, this means bad trials will take hyperparameter values near from well performing trials. For short, population based scheduler will do intensive exploitation as a lot of trials are going to take hyperparameter combinations near known good combinations. So we expect the search algorithm to still have added value on exploration. Note we couldn't test BOHB as it is the only search algorithm to use its own scheduler, but we used TwoPointsDE that we already presented here. As GAN don't use the concept validation or test dataset, we will provide slightly different plots, as well as quantitative results from best and worst trials. We also cannot plot an Inception score for a given trial, as we would not know what value give to a trial. Should it be the Inception Score when it has the hyperparameters given by the search algorithm? Or once the PB scheduler chose between giving him a better known combination and letting the trial keep his original one? Or should it be the score once it has finished to train? We thought the best way to handle this specific particularity was to plot as x-axis not only **score of every trial once they have finished training**, but rather **score of every trial every time they train once** (and can potentially be scheduled). So if  $n$  is the number of time a trial train, our plot should contain  $n$  times more values on the x-axis than before for the same number of trial search by a search algorithm.

#### 3.6.1 General Informations

As an introduction figure, we show in figure 33 the comparison between the random search test and validation accuracy, with the mean of all other algorithms's test and validation accuracy.

We observe that validation accuracy is way above test accuracy for all search algorithms. Also, Random actually gives similar results than the other search

#### Random Search versus all other algorithms on IMDB Dataset



**Figure 26.** Here we plot the averaged test and val accuracy of Random Search, and we compare it with the mean of all other algorithms (learning algorithms). Color areas are  $\sigma$  wide

algorithms.

#### 3.6.2 Test and validation accuracy

We see in figure 34

We also see in figure 37

These two plots permit an overall comparison. HyperOpt seems to be the best by far, followed closely by BayesOpt and BOHB. Random is clearly the worst.

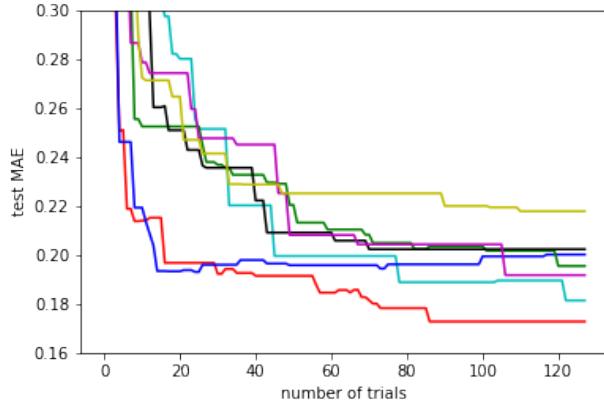
#### 3.6.3 Variance per Search Algorithm

To reduce information overload, we will plot the recommended metrics only for the top 4 performing algorithms. All axis are the same.

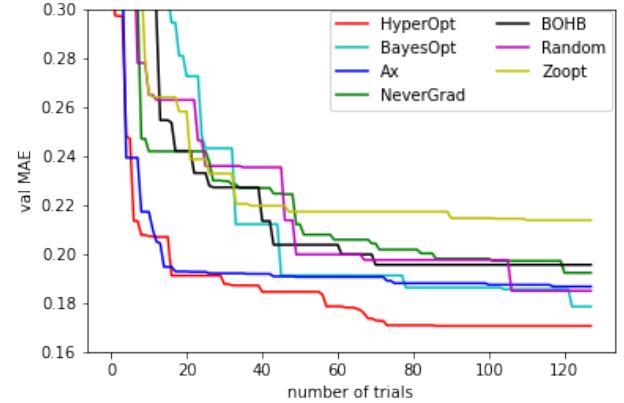
See in figure 35 the plot for HyperOpt, in figure 36 for Bayes, in figure 38 for Ax, and in figure 39 for Random.

The reason why random is so good is maybe because the dataset is so simple. Then even mediocre configurations would not have so big accuracy difference with very good configurations.

Also note that the difference between test and validation accuracy is very high here, so difference of results of accuracy among search algorithms seems relatively smaller. What we mean here is that there is still a 3% test accuracy difference between the best algorithm (BOHB) and the average of the other algorithms, but this improvement seems small compared to the 5% gap between test accuracy and validation accuracy.

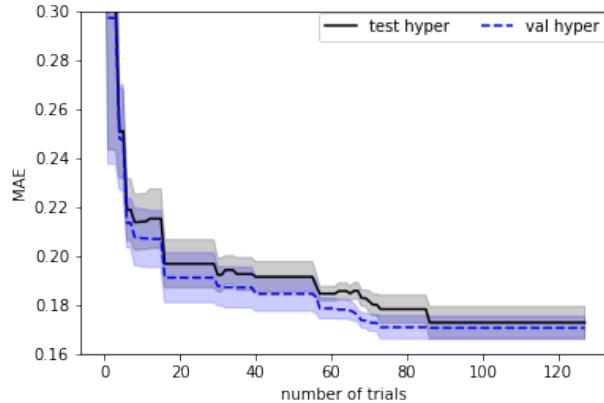
Test Accuracy

**Figure 27.** Here we plot the average test accuracy of the best trial obtained so far per search algorithm as a function of the number of trial.

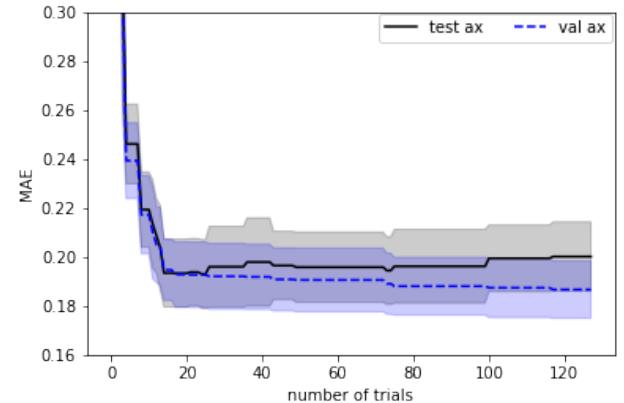
Validation accuracy

**Figure 30.** Here we plot the average validation accuracy of the best trial obtained so far per search algorithm as a function of number of trial.

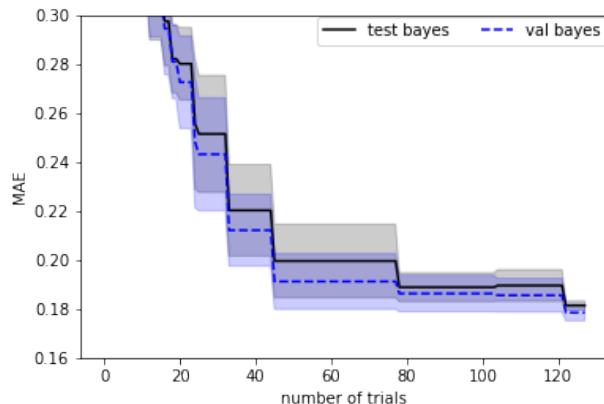
#### Tests and Validation MSE per algorithms with $\sigma$ narrow area on FMNIST Dataset



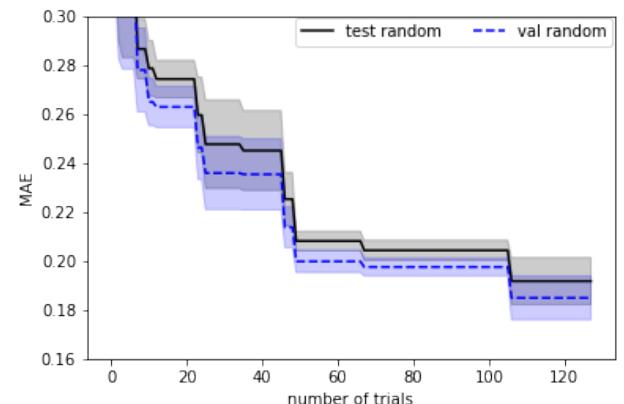
**Figure 28.** Validation, Test, Variance accuracy with Hyper



**Figure 31.** Validation, Test, Variance accuracy with BOHB



**Figure 29.** Validation, Test, Variance accuracy with Ax



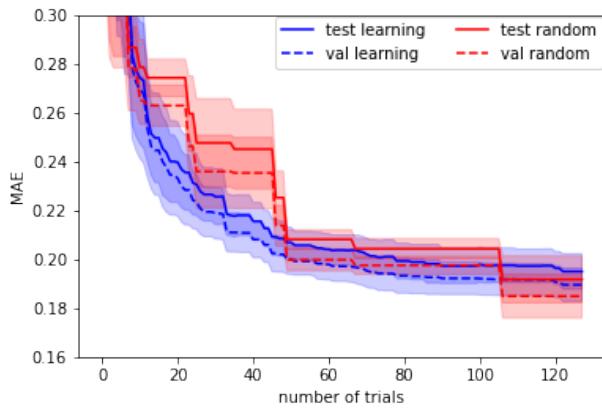
**Figure 32.** Validation, Test, Variance accuracy with Random

Detailed analysis on IMDB dataset For the detailed Analysis on IMDB dataset, we give 128 trials to every search algorithm. Each of these trials can learn on up to 4 epochs.

We repeated this process 4 times so we reduce influence of noise, and also so we can quantify variance.

### 3.6.4 General Informations

#### Random Search versus all other algorithms on IMDB Dataset



**Figure 33.** Here we plot the averaged test and val accuracy of Random Search, and we compare it with the mean of all other algorithms (learning algorithms). Color areas are  $\sigma$  wide

As an introduction figure, we show in figure 33 the comparison between the random search test and validation accuracy, with the mean of all other algorithms's test and validation accuracy.

We observe that validation accuracy is way above test accuracy for all search algorithms. Also, Random actually gives similar results than the other search algorithms.

### 3.6.5 Test and validation accuracy

We see in figure 34

We also see in figure 37

These two plots permit an overall comparison. HyperOpt seems to be the best by far, followed closely by BayesOpt and BOHB. Random is clearly the worst.

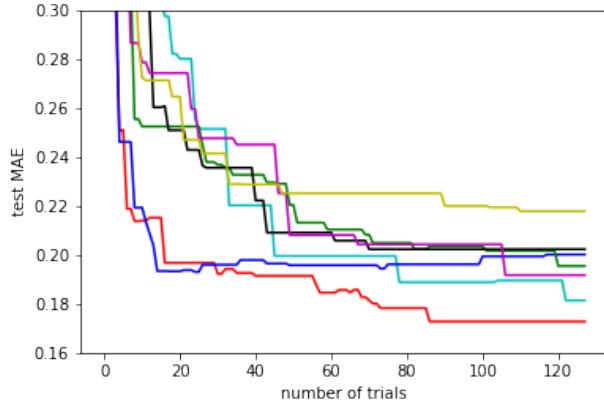
### 3.6.6 Variance per Search Algorithm

To reduce information overload, we will plot the recommended metrics only for the top 4 performing algorithms. All axis are the same.

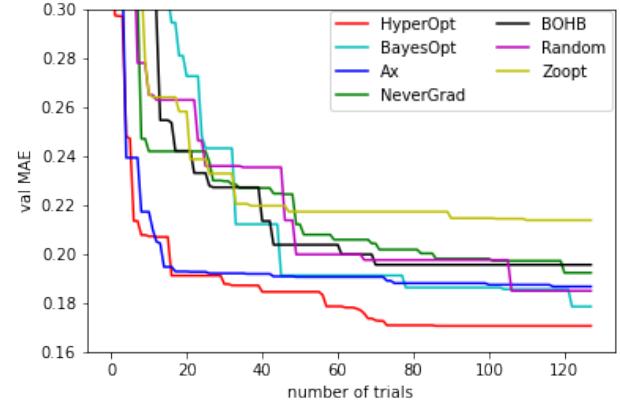
See in figure 35 the plot for HyperOpt, in figure 36 for Bayes, in figure 38 for Ax, and in figure 39 for Random.

The reason why random is so good is maybe because the dataset is so simple. Then even mediocre configurations would not have so big accuracy difference with very good configurations.

Also note that the difference between test and validation accuracy is very high here, so difference of results of accuracy among search algorithms seems relatively smaller. What we mean here is that there is still a 3% test accuracy difference between the best algorithm (BOHB) and the average of the other algorithms, but this improvement seems small compared to the 5% gap between test accuracy and validation accuracy.

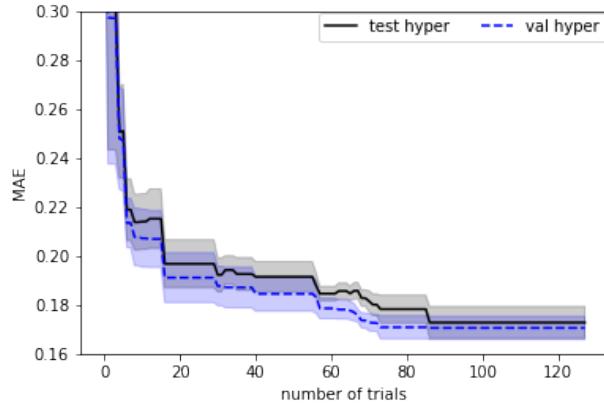
Test Accuracy

**Figure 34.** Here we plot the average test accuracy of the best trial obtained so far per search algorithm as a function of the number of trial.

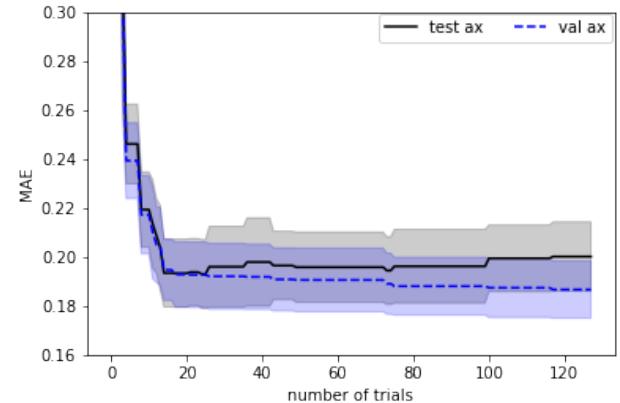
Validation accuracy

**Figure 37.** Here we plot the average validation accuracy of the best trial obtained so far per search algorithm as a function of number of trial.

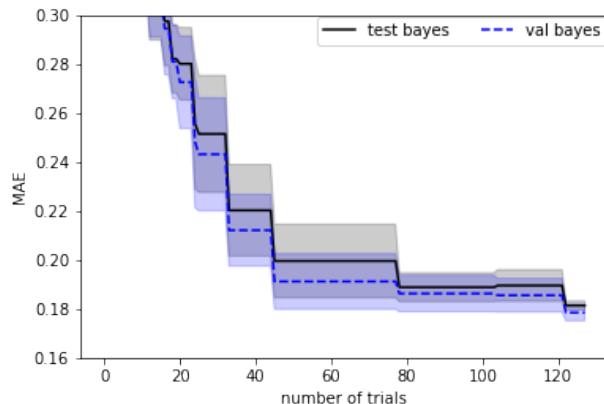
#### Tests and Validation MSE per algorithms with $\sigma$ narrow area on FMNIST Dataset



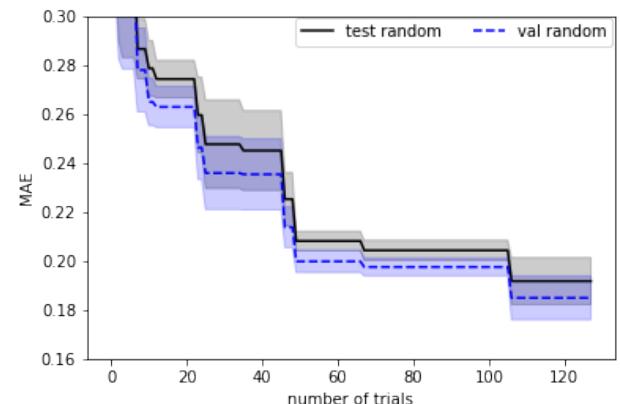
**Figure 35.** Validation, Test, Variance accuracy with Hyper



**Figure 38.** Validation, Test, Variance accuracy with BOHB



**Figure 36.** Validation, Test, Variance accuracy with Ax



**Figure 39.** Validation, Test, Variance accuracy with Random

### 3.7 OVERALL Analysis

#### 3.7.1 Time Analysis

##### Parallel case:

We here want to study the time it takes for a search algorithm to do search the best hyperparameter combination for one dataset. So to make 256 or 128 models learn up to 20 epochs. Remember that all Search Algorithms have schedulers that schedules their trials, and for example make a model learn up to 20 epochs only if it has a chance to beat the current best model. We are going to compare the time that takes a search algorithm compared to the time Random Search took. Without surprise, we see that this value is similar among datasets. On average, if random search takes time  $T$ :

HyperOpt would take  $0.97T$ .

BayesOpt would take  $1.04T$ .

Ax would take  $2.34T$ .

NeverGrad would take  $1.01T$ .

BOHB would take time  $2.4T$ .

ZooptSearch would take time  $1.16T$ .

Search algorithms can have a big influence in the time it takes as they can limit parallelism, because they could need to learn from trials in a sequential way.

Here we can see that most algorithms take a time very close from the reference of Random Search ( which parallelises perfectly). Two algorithms take really more time: there is AxSearch and BOHB. We can explain these difference: BOHB is the only search algorithms using its own scheduler so it is not surprising that it takes different time. But it is more surprising that despite the time he is granted he was not able to match the other search algorithms.

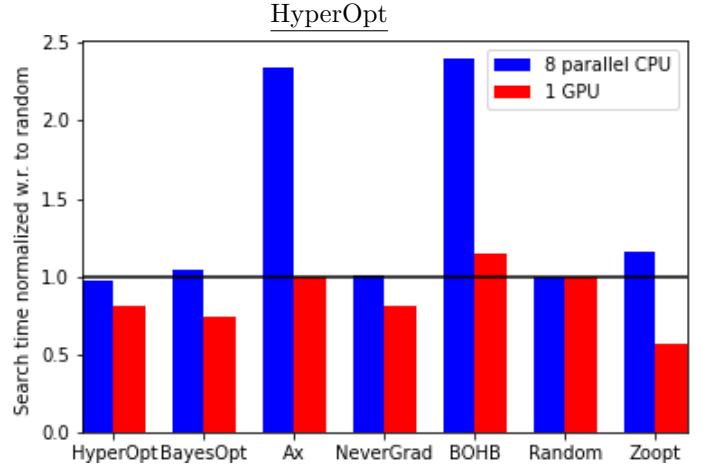
And AxSearch is the only Search algorithms that explicitly limits parallelism: To enforce sequential treatment of the information, it explicitly used no more than 3 of the 8 thread. As all other algorithms used all 8 threads, it is logical that Ax is about  $8/3$  slower.

We provide in page ?? plots of trial accuracy as a function of training time, for each search algorithm, note that these plots mostly depend on the scheduler, but since the scheduler makes his decisions based on the results obtained by the search algorithm, there are noticeable differences.

Intuitively, we would like to have a tendency curve with a very high accuracy at the smallest execution time. Ax is the best considering this criteria.

sequential case: For IMDB we used Google Collab Free GPU, which are fast and perform sequential trial search. We see less disparities in the distribution, which is probably still very noisy as google allocation is dynamic.

We find that algorithms faster than random are doing so because they exploit more often the CNN model,

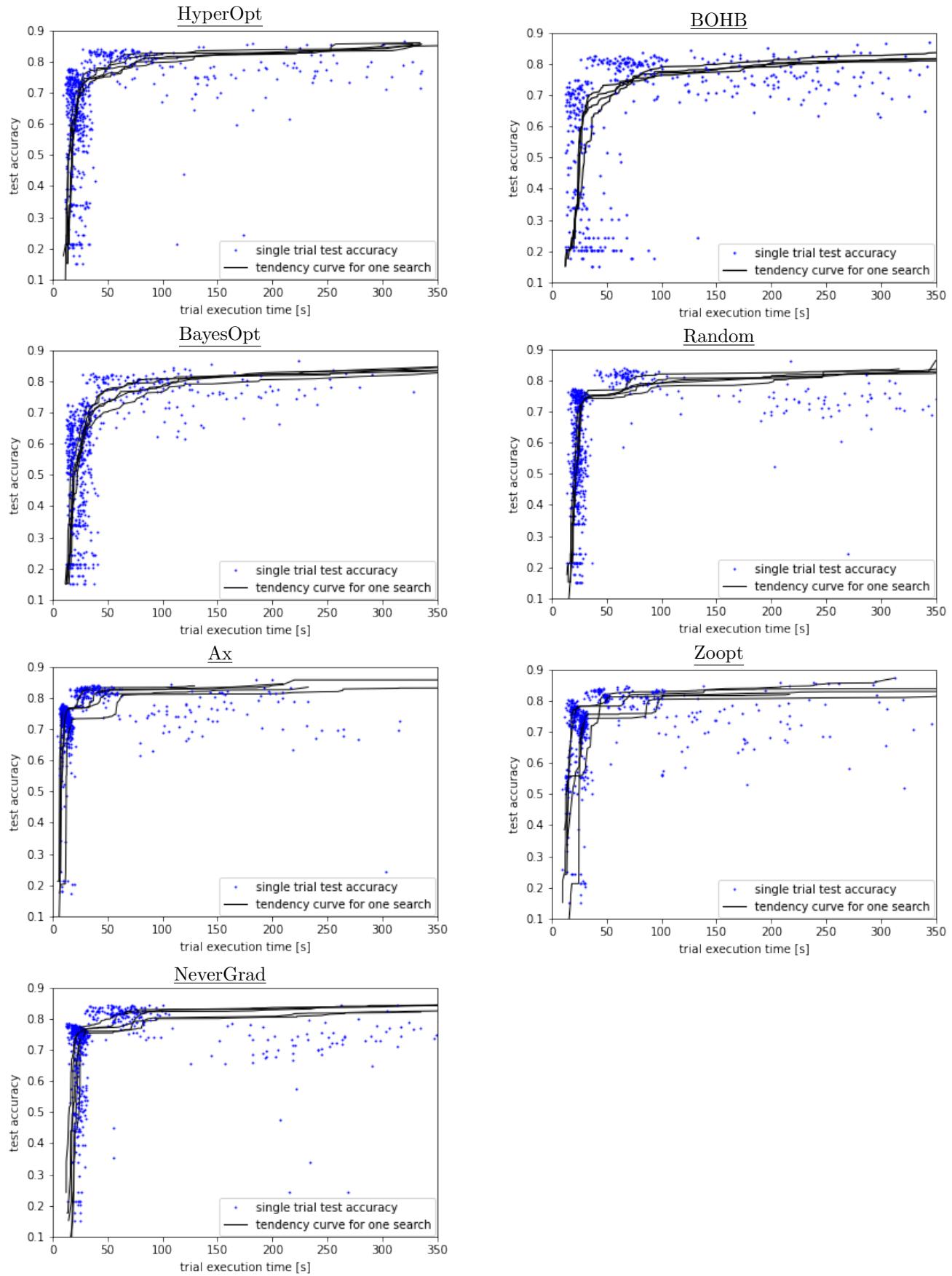


which is faster to train than CNN.

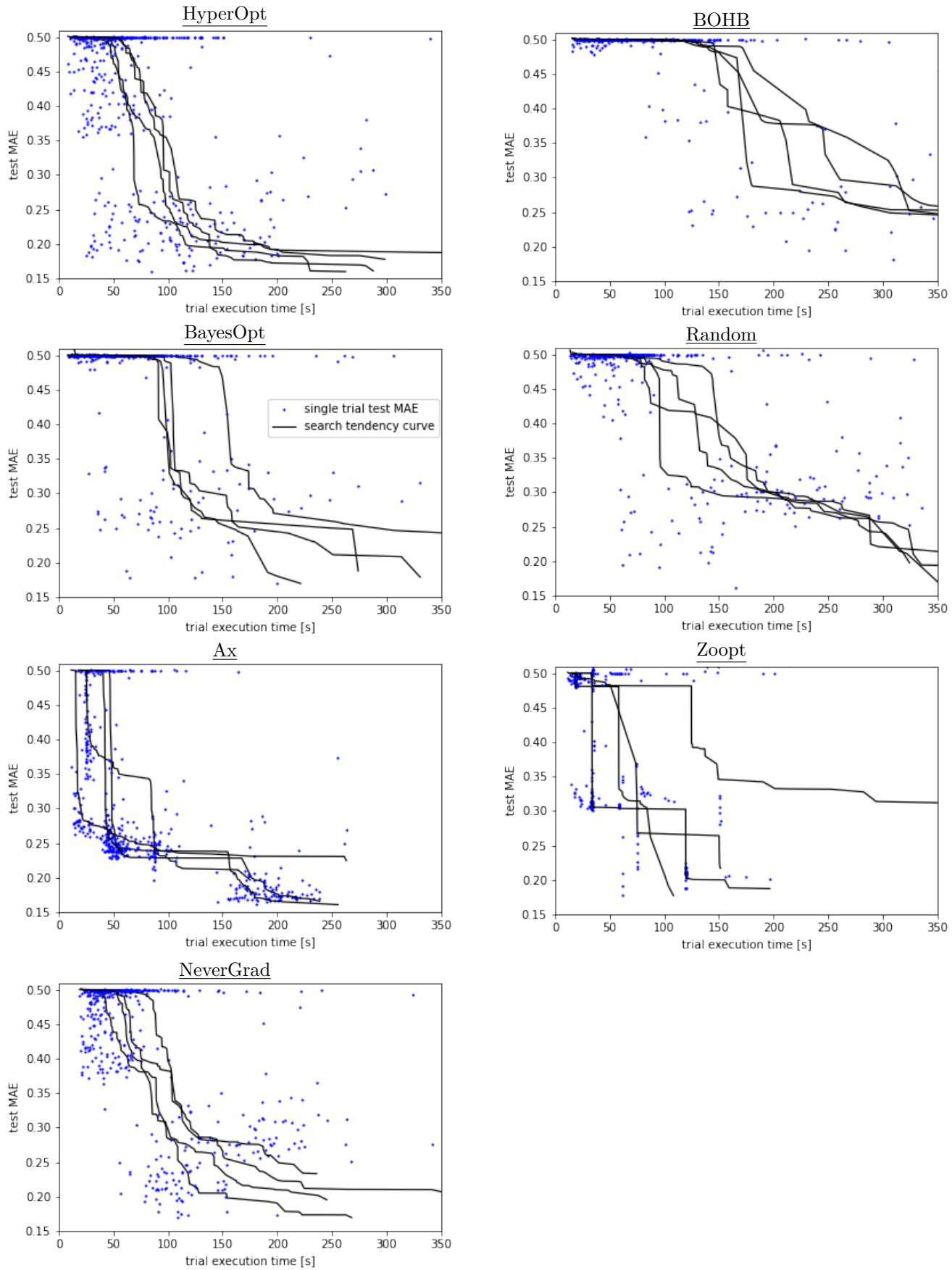
##### sequential case:

As we can observe, Random really has the smoothest curve, which is bad, opposed better algorithms such as HyperOpt,Ax, Nevergrad, who reach for small trial time some good configurations.

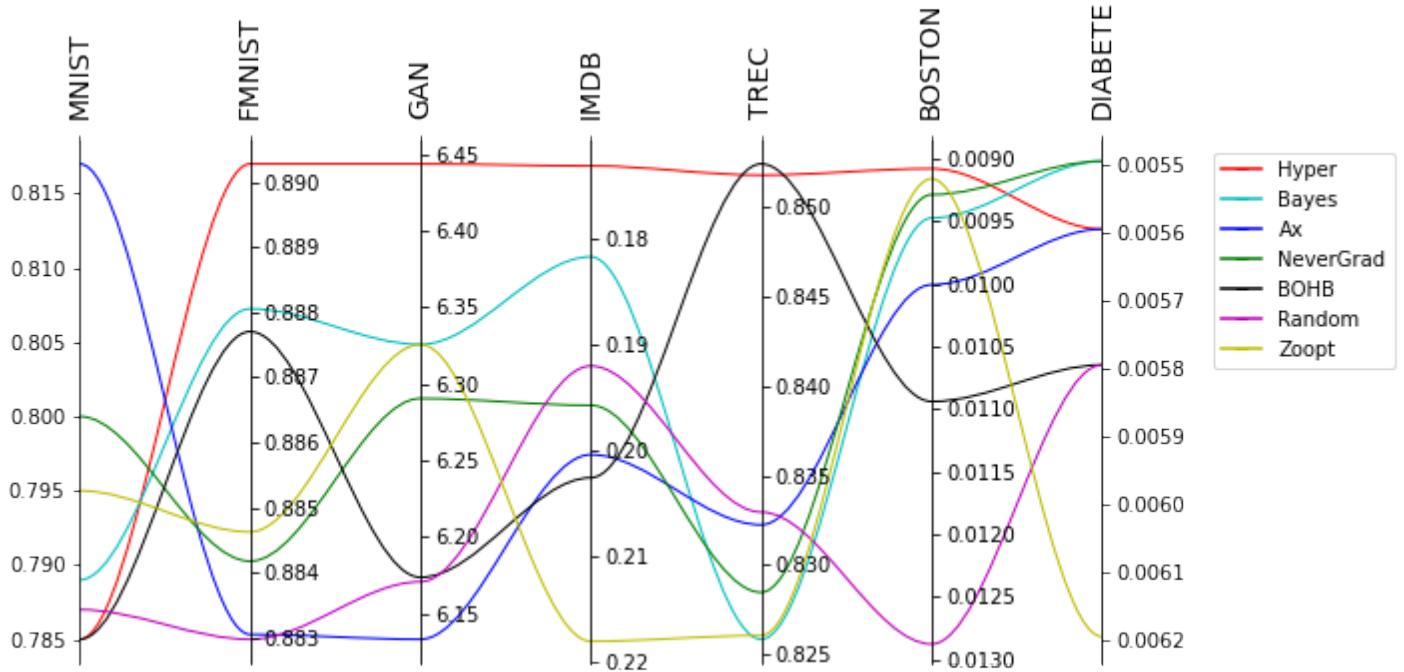
**Test accuracy per single trial as a function of execution time per algorithms with tendency curves**



**Test accuracy per single trial as a function of execution time per algorithms with tendency curves**



### Overall dataset performance comparison for all algorithms



#### 3.7.2 Dataset overall performance

```
// et la je fais la comparaison des résultats de chaque
search algo pour chaque probleme comme le graphe du
dernier article envoyé par diégo. // et la je fais a moyenne
de tous les noises scaled pour réduire le bruit
```

## 4 BATCH OF DIFFERENT MACHINE LEARNING PROBLEMS

## 5 ANALYZE

## 6 CONCLUSION

## A APPENDIX A: TOY BENCHMARK FURTHER ANALYSIS

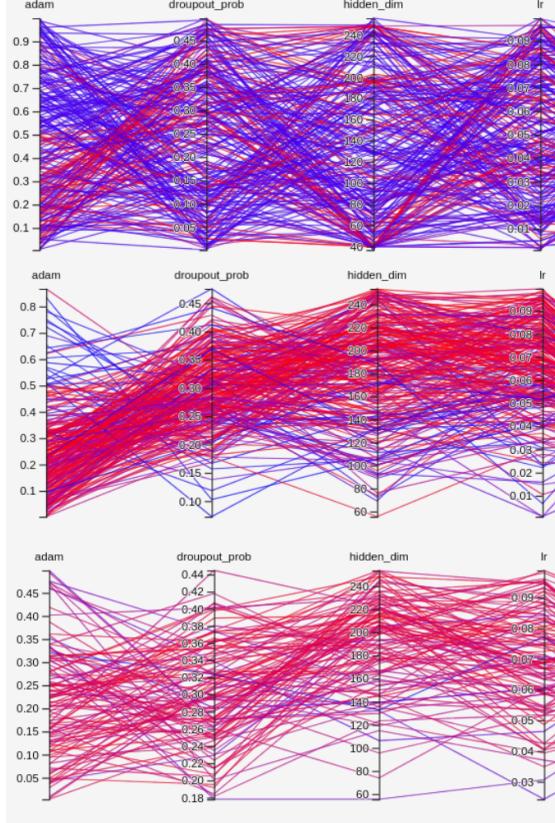
### A.1 Fashion MNIST Analysis

For reason of computational time, human time, and place on this report, we will do a detailed analysis for one of the problems, and then understand what are the relevant things to analyse for the 6 other datasets. We choose fashion MNIST dataset as it is simple, but not to much.

#### 1. Visual validation and global informations

First thing we do is to collect all datas from every search algorithm and to compare how the hyperparameters of every trial from each search algorithms is sampled. We expect random search to give uniform distribution, and all the others to have very specific distribution, with a lot of trials having the same (good) hyperparameters. We can use tensorflow for this purpose.

As explained on the figure, we understand some informations: to get good validation accuracy in fashion mnist in our benchmark, learning rate is good to be high (in [0.05-0.1]), as well as weight decay (in [0.05-0.09]). The best sigmoid function is Tanh, the number of dimension must be high (most values in [200-240]), the number of hidden layers low (1 hidden layer: this is probably because we have so few iterations. Thus we tend to underfit). Most interestingly, Adam and MLP seems to



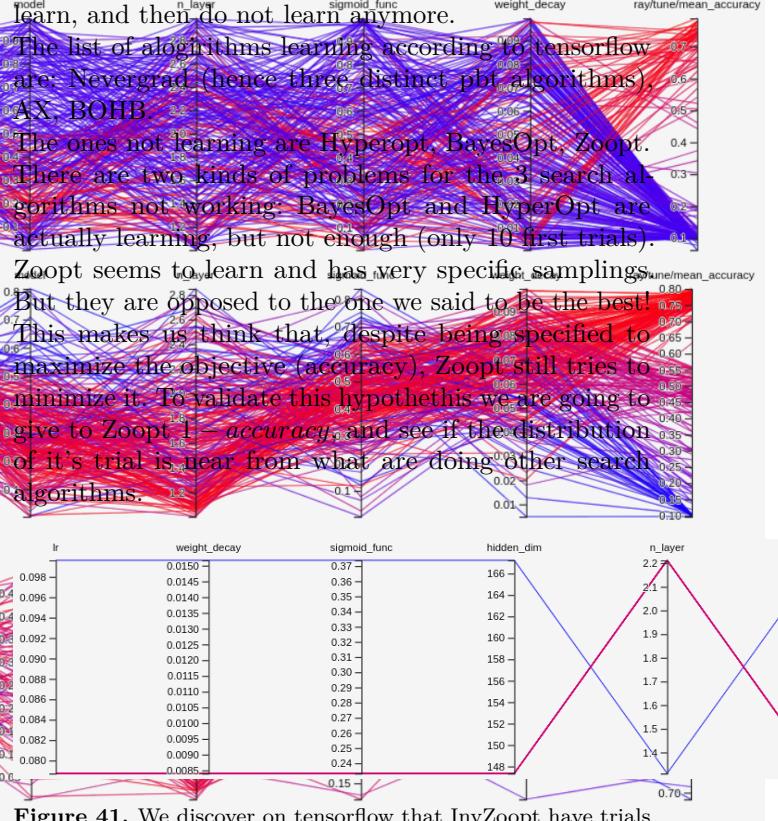
**Figure 40.** We here see the the distribution of hyperparameters per search algorithms. Every line is a trial, red or blue indicates whether it provides good or bad accuracy. The former is from random search. It corresponds very much to what we expected: we have a very wide and uniform looking distribution. The middle one is from AxSearch, and is similar to Nevergrad and BOHB: we have basically all trials going in the same good position, with still some wide to search for new combinations. Finally the last one is still from AxSearch once we keep only "good" trials (the one with more than 70 % accuracy). This allows us to understand which are the good values for this problem. Take care as the scale is different for the last one.

be not efficient compared to AdaBelief and CNN, and this seems to be the major factor for obtaining good results: in the last image form figure 5, ALL trials with accuracy above 0.7 used Adabelied and CNN.

Dropout probability shall be in [0.2-0.4].

This figure is a good news at it proves our benchmark works at it should.

Nevertheless, not all search algorithm work like on the second image. This is surprising as they are implemented exactly in the same way in our benchmark, and the space search is the same for all. So we can think that either this dataset did not suit with these algorithms, or that the Hyperparameter Optimization we are doing more generally in this benchmark does not suit these algorithms. We can easily check this question by testing on another dataset. After further testing, it seems that HyperOpt and BayesOpt only use the first 10 trials to



**Figure 41.** We discover on tensorflow that InvZoopt have trials with similar values as for the working algorithms, but that there is basically no exploration with Zoot: As we can see below, all trials go almost through the same path.

As we can see on the table 4, Zoot trying to minimize  $1 - \text{accuracy}$  is working (InvZoopt).

**Table 4** Here is a little sum up of the informations. Mean of 10 bests goal is to reduce the variance of the noisy information that validation accuracy is.

name	mean	mean of 10 bests	best
BayesOpt	0.251	0.774	0.789
Random	0.25	0.772	0.787
Ax	0.597	0.798	0.817
NeverGrad	0.539	0.793	0.8
BOHB	0.492	0.778	0.785
Zoot	0.101	0.144	0.51
HyperOpt	0.23	0.768	0.785
InvZoopt	0.142	0.773	0.795

The most interesting thing is to compare Random-Search with all other algorithms.

We can see that random search is very good as it's best trial is only 3% far from the best trial from the best search algorithms. It is however an important difference. Note that despite, these informations are very noisy, so it is a good thing to do the same comparison with all other datasets to have a more global view.

### 2. validation accuracy repartition over time

We first thought of comparing mean of validation accuracy, or envisaged comparing boxplots of n best results for every search algorithm, but we remember that the thing we want from a search algorithm is that it gives the best hyperparameters the fastest possible. So we are simply interested in the max test accuracy over all yet trained trials model as a function of iterations. We will perform test accuracy in next paragraph to validate our idea, and see if validation accuracy is reliable, but here we can start by using validation accuracy as it is the value we have since it is used by the optimizer, scheduler, and search algorithm.

### 3. test accuracy

First we notice that Hyperopt's model seems to overfit badly. As it is the only one doing so, we cannot know if it is a statistical mistake or a frequent thing, as we select models on 4 iterations, but here we measure test accuracy with up to 20 iterations.

Also we find that there is a small correlation between validation results and test results as we can see with "Correlation".

We make the hypothesis that test accuracy is noisy.

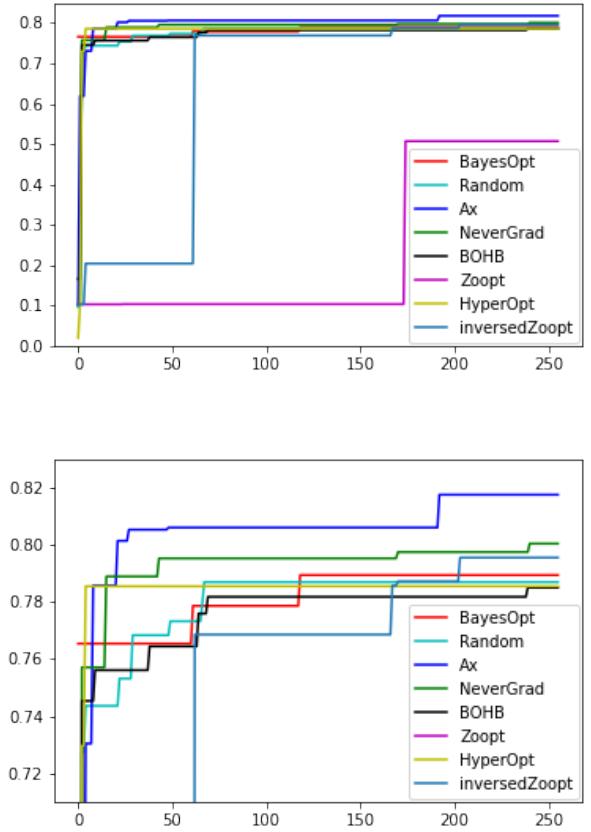
We calculate the mean and standard deviation of 20 test accuracy obtained from the same model, see "20 Random". We see that the test accuracy is not so noisy as the std is small.

If test is not so noisy, then it is validation accuracy that is noisy. We think that validation and test are actually highly correlated, but that the validation here is noisy. This is logical as validation is the metric for optimizer, scheduler, and search algorithm:

If a trial shows better validation accuracy then it really have (due to variance), it is more likely to be chosen, and if it shows lower validation accuracy, it is likely to be stopped/rejected. So we think we can define  $y$  the validation accuracy,  $z$  the test accuracy, and  $x$  the measured validation accuracy, and say:  $y = x + N(\mu, \sigma^2)$ , where  $\mu$  would typically be negative for the same number of iterations.

then we would have  $y$  and  $z$  highly correlated (to be verified thanks to next datasets).

Note that learning search algorithms can not exploit this noise to get fake better results than random search as this noise is random. So it is by definition as likely to give better or worse results. We compute  $\mu$  and  $\sigma$  of this normal distribution in "difference" and as expected  $\mu$  is



**Figure 42.** Here we plot the max validation accuracy already obtained (so maximum validation accuracy between this trial and all previous trials) as a function of trials number.

The objective being to be as much on the upper left as possible. The lower image is the same plot with a focus on the accuracy in [0.71;0.83]. We see that Ax and NeverGrad are really both better and faster than random for this dataset.

negative even for one more iteration (5 test iterations against 4 validation iterations).

The fact variance of noise is small is a good news, as it shows that the hypothesis is realistic. Furthermore, the noise is here the same for all search algorithms, this means we can, on average, rely on validation accuracy. In real life, one would likely launch the search algorithm, and just give as hyperparameters to his model the best configuration proposed by the search algorithm, that is with the best validation test. So on average, it is relevant to compare the search algorithms on their validation accuracy.

Looking at Table 5 makes us notice that BayesOpt and HyperOpt are not so bad. Maybe they stop learning after 10 iterations because they have learned all that could be learned, and they spent all other trials by doing random exploration?

**Table 5** Here we compare the validation accuracy of the best trial from each search algorithm to the test accuracy of a model with same hyperparameters.

name	validation	5 steps	10 steps	20 steps
BayesOpt	0.789	0.806	0.848	0.851
Random	0.787	0.781	0.823	0.835
Ax	0.817	0.75	0.81	0.852
NeverGrad	0.8	0.8	0.825	0.839
BOHB	0.785	0.781	0.815	0.83
HyperOpt	0.785	0.774	0.765	0.611 !
InvZoopt	0.795	0.8157	0.837	0.84
Correlation		-0.405	0.107	0.396
$\mu$ of 20 Random		0.789	0.828	0.844
$\sigma$ of 20 Random		0.012	0.014	0.01
$\mu$ of difference		-0.007	0.024	0.014
$\sigma$ of difference		0.027	0.026	0.077

//Détails: TODO: references and hypertexts, remove metrics from image

## A.2 MNIST Analysis

Here we study the MNIST dataset, that is very easy, and is similar to fashion MNIST as it has exactly the same search space.

### 1. Visual validation and global informations

As explained on the figure, we understand some informations using tensorflow (in bold informations different then for fashion mnist): to get good validation accuracy in mnist in our benchmark, **learning rate is good to be low** (in [0.0001-0.02]), as well as **weight decay** (in [0.0001-0.07]). The best sigmoid function is Tanh, the number of dimension and hidden layers doesn't seem to be that important. Adam and MLP are still not efficient compared to AdaBelief and CNN.

**Table 6** Validation accuracy for MNIST.

name	mean	mean of 10 bests	best
BayesOpt	0.383	0.945	0.963
Random	0.354	0.932	0.944
Ax	0.888	0.959	0.969
NeverGrad	0.765	0.961	0.972
BOHB	0.348	0.944	0.956
HyperOpt	0.675	0.911	0.919
InvZoopt	0.697	0.938	0.963

Dropout probability shall be in [0.2-0.3].

As we can see on table 6, this time random search is really less good then almost all others algorithms: between 2 and 3% less good although we are getting very close to 100%, so one would have thought that the gap would be reduced.

### 2. validation accuracy repartition over time

### 3. test accuracy

A lot of models tends to overfit. This is because this dataset is so simple. Next time we would increase the number of iterations to have a good validation accuracy for more iterations even though overfitting is not the main issue here.

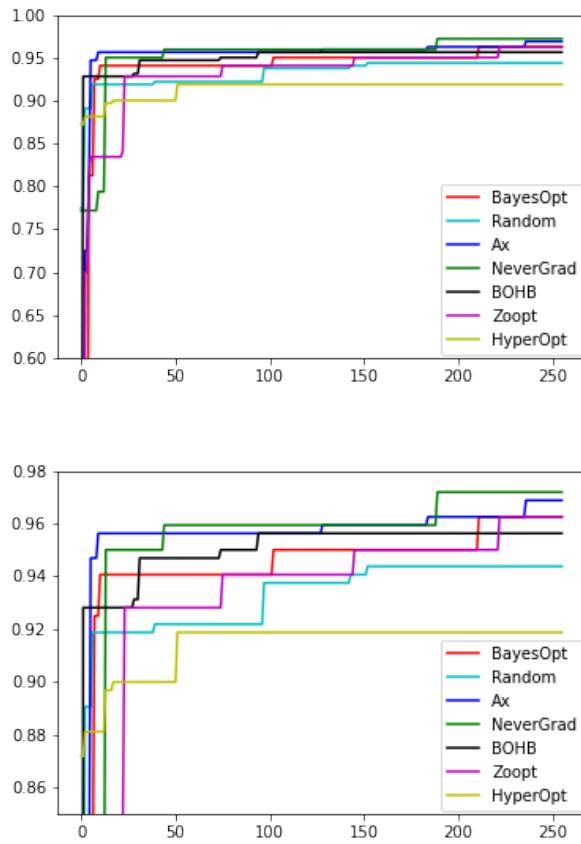
Main issue is that all models have a validation accuracy way above test accuracy.

It is the role of the search algorithm to handle noisy inputs, this is why all search algorithms have a similar drop of result from validation to test (see "difference"). Once again the best opponent to Random Search is AxSearch on test data, but that could be noisy interpretation.

## A.3 Diabetes Analysis

Here we study the Diabetes regression dataset, that is known for being very easy. We choose MSE as loss function. Note that we normalised the data with mean 0 and std of 1. So it is normal to have very small loss as labels are smalls. **Since all loss are below 0.01, we will present loss in this section in %**

### 1. Visual validation and global informations



**Figure 43.** Here we plot the max validation accuracy already obtained as a function of trials number.

The objective being to be as much on the upper left as possible. We see that All learning algorithms but HyperOpt are really both better and faster than random for MNIST.

We understand some informations using tensorflow to get good validation loss in diabetes in our benchmark, learning rate is good to be high (in [0.05-1]), the weight decay around 0.03. The best sigmoid function is not relevant here as unused., the number of dimension shall be arround 140 and the best number of layers is 2. Dropout probability shall be low: in [0.1-0.2] Best model is MLP (opposed to LogReg) and Adam is preferred over Adabelief! .

As we can see on table 10, we had a problem of outlayer values for some of the 7 algorithms, in this case we would replace by 100000 (means are still not significant). Random search is a little less good then others algorithms, but it does not seem significant for this easy dataset. This can be because, as it is easy, it is easy to find a very good configuration (even randomly), but it contradicts results from mnist which was easy as well. Maybe this one is order of magnitudes easier?

## 2. validation loss repartition over time

**Table 7** Here we compare the validation accuracy of the best trial from each search algorithm to the test accuracy of a model with same hyperparameters.

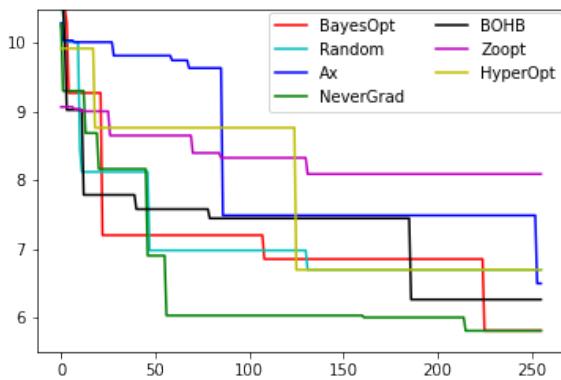
name	validation	5 steps	10 steps	20 steps
BayesOpt	0.963	0.916	0.931	0.906
Random	0.944	0.85	0.891	0.891
Ax	0.969	0.95	0.925	0.931
NeverGrad	0.972	0.869	0.875	0.881
BOHB	0.956	0.853	0.791	0.906
HyperOpt	0.919	0.775	0.825	0.763
InvZooth	0.963	0.73	0.825	0.888
$\mu$ of difference		-0.106	-0.089	-0.074
$\sigma$ of difference		0.064	0.046	0.037

**Table 8** Validation loss for Diabetes, in % from standard normal distribution.

name	mean	mean of 10 bests	best
BayesOpt	10.5	7.6	5.8
Random	91.2	8	6.7
Ax	10.5	8.1	6.5
NeverGrad	9.8	6.1	5.8
BOHB	3147.8	7.6	6.3
HyperOpt	400.3	8.5	6.7
Zooth	10.6	8.5	8.1

## 3. test accuracy

It doesn't seem, but there is a 6% difference between test accuracy of Nevergrad, and Random. We are surprised as model do not overfit as for mnist. This time all algorithms are better than random except for BOHB and Zooth.



**Figure 44.** Here we plot the min loss accuracy already obtained as a function of trials number.

The objective being to be as much on the lower left as possible. Random is near to all other search algorithms but NeverGrad, that clearly dominates here.

**Table 9** Here we compare the validation accuracy of the best trial from each search algorithm to the test accuracy of a model with same hyperparameters.

	name	validation	5 steps	10 steps	20 steps
BayesOpt	BayesOpt	5.8	7.2	5.5	5.5
Random	Random	6.7	6.1	5.9	5.8
Ax	Ax	6.5	8	6.3	5.6
NeverGrad	NeverGrad	5.8	8.8	6.6	5.5
BOHB	BOHB	6.3	7.7	5.6	5.8
HyperOpt	HyperOpt	6.7	7.5	5.6	5.6
Zoot	Zoot	8.1	8	6.9	6.2
$\mu$ of difference			1.1	-0.5	-0.8
$\sigma$ of difference			1.1	0.6	0.5

#### A.4 Boston Analysis

Here we study the Boston regression dataset, a very known and yet complex regression dataset. We choose MSE as loss function. Note that we normalised the data with mean 0 and std of 1. **Here loss is not in %** anymore.

#### 1. Visual validation and global informations

We understand some informations using tensorflow. To get good validation loss in boston in our benchmark, learning rate is good to be moderate (in [0.02-0.04]), the weight decay around 0.04. The best sigmoid function is not relevant here as unused., the number of dimension shall be high: around 200 and the best number of layers is 1 or 2. Dropout probability shall be around 0.35. Most trials use LogReg as it provides low loss, but lowest loss are performed with MLP.

**Table 10** Validation loss for Boston with standard normal distribution.  
extreme values would replaced by 100000 (means are still not very significant ).

name	mean	mean of 10 bests	best
BayesOpt	17365	0.147	0.016
Random	25942	0.395	0.179
Ax	7910	0.372	0.022
NeverGrad	4264	0.241	0.122
BOHB	13429	0.118	0.034
HyperOpt	10581	0.124	0.021
Zoot	8751	1.65	0.154

Random search is notably less good than all other search algorithms but Zoot and NeverGrad.

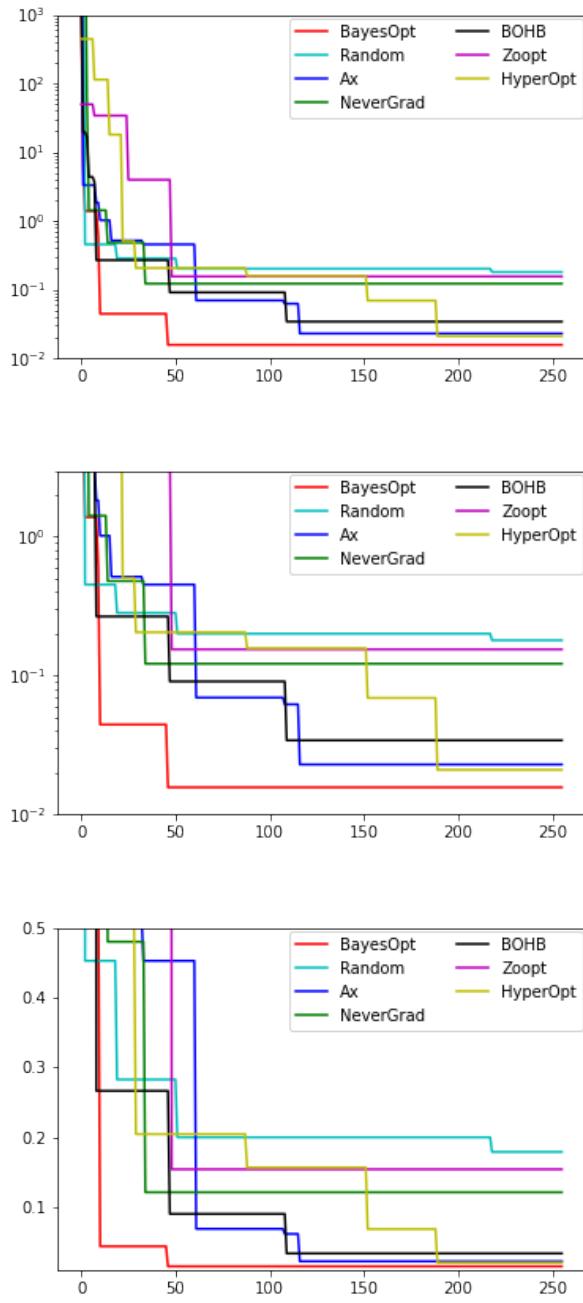
#### 2. validation loss repartition over time

A problem is that we have so wide interval for loss. Next time we will use RMSE.

#### 3. test accuracy

Here we made an experiment: we reduced a dimension. This can be because of the loss that is squared, but bad algorithms from validation loss are far worst (random and nevergrad), and good algorithms are better. Loss of random search is 8 times worst than BayesOpt's loss. Random and NeverGrad are so bad because they both chose LogReg.

We have no overfitting at all.



**Figure 45.** Here we plot the min loss accuracy already obtained as a function of trials number.

The objective being to be as much on the lower left as possible. Be careful as it is a log scale on the y-axis for the upper images.

## A.5 GAN Analysis

## A.6 TREC Analysis

## A.7 IMDB Analysis

**Table 11** Here we compare the validation accuracy of the best trial from each search algorithm to the test accuracy of a model with same hyperparameters.

name	validation	5 steps	10 steps	20 steps
BayesOpt	0.016	0.403	0.035	0.014
Random	0.179	6.036	0.855	0.124
Ax	0.022	0.169	0.069	0.034
NeverGrad	0.122	2.352	0.833	0.154
BOHB	0.034	0.045	0.025	0.021
HyperOpt	0.021	0.022	0.02	0.016
Zoot	0.154	1.51	0.221	0.024

## REFERENCES

- Balandat M., Karrer B., Jiang D. R., Daulton S., Letham B., Wilson A. G., Bakshy E., 2019, arXiv preprint arXiv:1910.06403
- Bergstra J., Bengio Y., 2012, The Journal of Machine Learning Research, 13, 281
- Bergstra J. S., Bardenet R., Bengio Y., Kégl B., 2011, in Advances in neural information processing systems. pp 2546–2554
- Dimmery D., Bakshy E., Sekhon J., 2019, in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp 2914–2922
- Dodge J., Gururangan S., Card D., Schwartz R., Smith N. A., 2019, arXiv preprint arXiv:1909.03004
- Falkner S., Klein A., Hutter F., 2018, arXiv preprint arXiv:1807.01774
- Gamboa M., 2019, Bayesian optimization from medium Hansen N., 2006, in , Towards a new evolutionary computation. Springer, pp 75–102
- Kandasamy K., Vysyraju K. R., Neiswanger W., Paria B., Collins C. R., Schneider J., Poczos B., Xing E. P., 2020, Journal of Machine Learning Research, 21, 1
- Kennedy J., Eberhart R., 1995, in Proceedings of ICNN'95-International Conference on Neural Networks. pp 1942–1948
- Li L., Jamieson K., DeSalvo G., Rostamizadeh A., Talwalkar A., 2017, The Journal of Machine Learning Research, 18, 6765
- Li L., Jamieson K., Rostamizadeh A., Gonina E., Hardt M., Recht B., Talwalkar A., 2018, arXiv preprint arXiv:1810.05934

Liu Y.-R., Hu Y.-Q., Qian H., Yu Y., 2019, in Proceedings of the First International Conference on Distributed Artificial Intelligence. pp 1–8

Louppe G., Kumar M., 2016, Bayesian optimization with skopt

Storn R., Price K., 1997, Journal of global optimization, 11, 341