Amazon SageMaker Automatic Model Tuning: Scalable Gradient-Free Optimization

Valerio Perrone[1], Huibin Shen, Aida Zolic, Iaroslav Shcherbatyi, Amr Ahmed
Tanya Bansal, Michele Donini, Fela Winkelmolen*, Rodolphe Jenatton*
Jean Baptiste Faddoul, Barbara Pogorzelska, Miroslav Miladinovic
Krishnaram Kenthapadi, Matthias Seeger, Cédric Archambeau
Amazon Web Services

ABSTRACT

Tuning complex machine learning systems is challenging. Machine learning typically requires to set hyperparameters, be it regularization, architecture, or optimization parameters, whose tuning is critical to achieve good predictive performance. To democratize access to machine learning systems, it is essential to automate the tuning. This paper presents Amazon SageMaker Automatic Model Tuning (AMT), a fully managed system for gradient-free optimization at scale. AMT finds the best version of a trained machine learning model by repeatedly evaluating it with different hyperparameter configurations. It leverages either random search or Bayesian optimization to choose the hyperparameter values resulting in the best model, as measured by the metric chosen by the user. AMT can be used with built-in algorithms, custom algorithms, and Amazon SageMaker pre-built containers for machine learning frameworks. We discuss the core functionality, system architecture, our design principles, and lessons learned. We also describe more advanced features of AMT, such as automated early stopping and warm-starting, showing in experiments their benefits to users.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Computer systems organization** → **Dependable and fault-tolerant systems and networks**.

KEYWORDS

AutoML, hyperparameter tuning, scalable systems

ACM Reference Format:
Valerio Perrone[1], Huibin Shen, Aida Zolic, Iaroslav Shcherbatyi, Amr Ahmed, Tanya Bansal, Michele Donini, Fela Winkelmolen*, Rodolphe Jenatton*, Jean Baptiste Faddoul, Barbara Pogorzelska, Miroslav Miladinovic, Krishnaram Kenthapadi, Matthias Seeger, Cédric Archambeau. 2021. Amazon SageMaker Automatic Model Tuning: Scalable Gradient-Free Optimization. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3447548.3467098

[1]Correspondence to: Valerio Perrone, <vperrone@amazon.com>.
*Work done while at Amazon Web Services.

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08.
https://doi.org/10.1145/3447548.3467098

1 INTRODUCTION

In modern machine learning, complex statistical models with many free parameters are fit to data by way of automated and highly scalable algorithms. For example, weights of a deep neural network are learned by stochastic gradient descent (SGD), minimizing a loss function over the training data. Unfortunately, some remaining *hyperparameters (HPs)* cannot be adjusted this way, and their values can significantly affect the prediction quality of the final model. In a neural network, we need to choose the learning rate of the stochastic optimizer, regularization constants, the type of activation functions, and architecture parameters such as the number or the width of the different layers. In Bayesian models, priors need to be specified, while for random forests or gradient boosted decision trees, the number and maximum depth of trees are important HPs.

The problem of hyperparameter tuning can be formulated as the minimization of an objective function $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ denotes the space of valid HP configurations, while the value $f(\mathbf{x})$ corresponds to the metric we wish to optimize. We assume that $\mathbf{x} = [x_1, \ldots, x_d]$, where $x_j \in \mathcal{X}_j$ is one of the d hyperparameters, such that $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_d$. For example, given some $\mathbf{x} \in \mathcal{X}$, $f(\mathbf{x})$ may correspond to the held-out error rate of a machine learning model when trained and evaluated using the HPs $\mathbf{x}$. In practice, hyperparameter optimization requires addressing a number of challenges. First, we do not know the analytical form of the function f (*i.e.*, it can only be observed through evaluations) and is thus difficult to optimize as we cannot compute its gradients. Second, evaluations of $f(\mathbf{x})$ are often expensive in terms of time and compute (e.g., training a deep neural network on a large dataset), so it is important to identify a good hyperparameter configuration $\mathbf{x}_*$ with the least number of queries of f. Third, for complex models, the HP configuration space $\mathcal{X}$ can have diverse types of attributes, some of which may be integer or categorical. For numerical attributes, search ranges need to be determined. Some attributes in $\mathcal{X}$ can even be conditional (e.g., the width of the l-th layer of a neural network is only relevant if the model has at least l layers). Finally, even if $f(\mathbf{x})$ varies smoothly in $\mathbf{x}$, evaluations of $f(\mathbf{x})$ are typically noisy.

We present Amazon SageMaker Automatic Model Tuning (AMT), a fully managed system for gradient-free function optimization at scale.[1] The key contributions of our work are as follows:

- Design, architecture and implementation of hyperparameter optimization as a distributed, fault-tolerant, scalable, secure and fully managed service, integrated with Amazon SageMaker (§3).

[1]`Amazon SageMaker` is a service that allows easy training and hosting of machine learning models. For details, see https://aws.amazon.com/sagemaker and [32].

- Description of the Bayesian Optimization algorithm powering AMT, including efficient hyperparameter representation, surrogate Gaussian process model, acquisition functions, and parallel and asynchronous evaluations (§4).
- Overview of advanced features such as log scaling, automated early stopping and warm start (§5).
- Discussion of deployment results as well as challenges encountered and lessons learned (§6).

2 PRELIMINARIES

Traditionally, HPs are, either hand-tuned by experts in what amounts to a laborious process, or they are selected using brute force schemes such as grid search or random search. In response to increasing model complexity, a range of more sample-efficient HP optimization techniques have emerged. A comprehensive review of modern hyperparameter optimization is provided in [10]. Here, we will focus on work relevant in the context of AMT.

2.1 Model-Free Hyperparameter Optimization

Any HPO method is proposing evaluation points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$, such that

$$\min_{t=1,\ldots,T} f(\mathbf{x}_t) \approx \min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}).$$

For the simplest methods, the choice of $\mathbf{x}_t$ does not depend on earlier observations. In *grid search*, we fix K values for every HP x_j, then evaluate f on the Cartesian product, so that $T = K^d$. In *random search*, we draw $\mathbf{x}_t$ independently and uniformly at random. More specifically, each $x_{t,j}$ is drawn uniformly from $\mathcal{X}_j$. For numerical HPs, the distribution may also be uniform in a transformed domain (e.g., log domain). Random search is more effective than grid search when some of the HPs x_j are irrelevant as it considers a larger number of values of the relevant ones [4]. Both methods are easily parallelizable. Random search should always be considered as a baseline, and is frequently used to initialize more sophisticated HPO methods. Another alternative is picking a set of pseudo-random points known as Sobol sequences [53]. The advantage is that they provide a better coverage of the search space, but are deterministic

These simple baselines can be improved upon by making use of earlier observations $\{(\mathbf{x}_{t_1}, f(\mathbf{x}_{t_1})) \mid t_1 < t\}$ in order to plan subsequent evaluations $\mathbf{x}_{t_2}$, $t_2 \geq t$. Population-based methods, such as evolutionary or genetic algorithms, are attractive if parallel computing resources are available [16, 48]. In each generation, all configurations from a fixed-size population are evaluated. Then, a new population is created by randomly mutating a certain percentile of the top performing configurations, as well as sampling from a background distribution, and the process repeats with the next generation. Evolutionary algorithms (EAs) are powerful and flexible search strategies, which can work even in search spaces of complex structure. However, EAs can be difficult to configure to the problem at hand, and since fine-grained knowledge about f can be encoded only in a large population, they tend to require a substantial amount of parallel compute resources. In some EAs, low-fidelity approximations of f are employed during early generations [22] in order to reduce computational cost. Multi-fidelity strategies are discussed in more generality below.

2.2 Surrogate Models. Bayesian Optimization

A key idea to improve data efficiency of sampling is to maintain a *surrogate model* of the function f. At decision step t, this model is fit to previous data $\mathcal{D}_{<t} = \{(\mathbf{x}_{t_1}, f(\mathbf{x}_{t_1})) \mid t_1 < t\}$. In *Bayesian optimization (BO)*, we employ probabilistic models, not only predicting best estimates (posterior means), but also uncertainties (posterior variances) for each $\mathbf{x} \in \mathcal{X}$: the value of the next $\mathbf{x}_t$ could come from exploration (sampling where f is most uncertain) or from exploitation (minimizing our best estimate), and a calibrated probabilistic model can be used to resolve the trade-off between these desiderata optimally [56]. More concretely, we choose $\mathbf{x}_t$ as the best point according to an acquisition function $\mathcal{A}(\mathbf{x}|\mathcal{D}_{<t})$, which is a utility function averaged over the posterior predictive distribution $p(f(\mathbf{x})|\mathcal{D}_{<t})$. The most common surrogate model for BO is based on *Gaussian processes (GPs)* [47], which not only have simple closed-form expressions for posterior and predictive distributions, but also come with strong theoretical guarantees. Other BO surrogate models include random forests [20] and Bayesian neural networks [54], which can suffer from uncalibrated uncertainty estimates. We give a detailed account of sequential BO with a GP surrogate model in Section 4. Tutorials on BO are provided in [5, 50].

2.3 Early Stopping. Multi-Fidelity Optimization

Modern deep learning architectures can come with hundreds of millions of parameters, and even a single training run can take many hours or even days. In such settings, it is common practice to cheaply probe configurations $\mathbf{x}$ by training for few epochs or on a subset of the data. While this gives rise to a low-fidelity approximation of $f(\mathbf{x})$, such data can be sufficient to filter out poor configurations early on, so that full training is run only for the most promising ones. To this end, we consider functions $f(\mathbf{x}, r)$, r being a resource attribute, where $f(\mathbf{x}, r_{\max}) = f(\mathbf{x})$ is the expensive metric of interest, while $f(\mathbf{x}, r)$, $r < r_{\max}$ are cheaper-to-evaluate approximations. Here, r could be the number of training epochs for a deep neural network (DNN), the number of trees for gradient boosting, or the dataset subsampling ratio. When training DNNs, we can evaluate $f(\mathbf{x}, r)$, $r = 1, 2, \ldots$ by computing the validation metric after every epoch. For gradient boosting implementations supporting incremental training, we can obtain intermediate values $f(\mathbf{x}, r)$ as well. In *early stopping* HPO, the evaluation of $\mathbf{x}$ is terminated at a level r if the probability of it scoring worse at $r_{\max}$ than some earlier $\mathbf{x}'$ is predicted high enough. The median rule is a simple instance of this idea [14], while other techniques aim to extrapolate learning curves beyond the current r [8, 26]. Early stopping is particularly well suited to asynchronous parallel execution: whenever a job is stopped, an evaluation can be started for the next configuration $\mathbf{x}$ proposed by HPO. An alternative to stopping configurations is to pause and (potentially) resume them later [58].

Besides early stopping HPO, successive halving (SH) [23, 25] and Hyperband [30] are two other basic instances of *multi-fidelity* techniques. In each round, $f(\mathbf{x}, r_{\min})$ is evaluated for a number n of configurations $\mathbf{x}$ sampled at random. Next, $f(\mathbf{x}, 2r_{\min})$ is run for the top $n/2$ of configurations, while the bottom half are discarded. This filtering step is repeated until $r_{\max}$ is reached. One drawback of SH and Hyperband is their synchronous nature, which is remedied by ASHA [31]. All these methods make use of SH scheduling of

evaluations, yet new configurations are chosen at random. BOHB [9] combines synchronous Hyperband with model-based HPO, and a more recent combination of ASHA with Bayesian optimization is MOBSTER [27]. This method tends to be far more efficient than synchronous counterparts, and can often save up to half of resources compared to ASHA.

3 THE SYSTEM

In this section we give an overview of the system underpinning SageMaker AMT. We lay out design principles, describe the system architecture, and highlight a number of challenges related to running HPO in the cloud.

3.1 Design Principles

We present the key requirements underlying the design of SageMaker AMT.

- *Easy to use and fully managed*: To ensure broad adoption by data scientists and ML developers with varying levels of ML knowledge, AMT should be easy to use, with minimal effort needed for a new user to setup and execute. Further, we would like AMT to be offered as a fully managed service, with stable API and default configuration settings so that the implementation complexity is abstracted away from the customer. AMT spares users the pain to provision hardware, install the right software, and download the data. It takes care of uploading the models to the users' accounts and providing them with training performance metrics.
- *Tightly integrated with other SageMaker components*: Considering that model tuning (HPO) is typically performed as a part of the ML pipeline involving several other components, AMT should seamlessly operate with other SageMaker components and APIs.
- *Scalable*: AMT should scale with respect to different data sizes, ML training algorithms, number of HPs, metrics, HPO methods and hardware configurations. Scalability includes a failure-resistant workflow with built-in retry mechanisms to guarantee robustness.
- *Cost-effective*: AMT should be cost-effective to the customer, in terms of both compute and human costs. We would like to enable the customer to specify a budget and support cost reduction techniques such as early stopping and warm start.
- *Available*: A pillar of AMT's design is to ensure that it is highly available. This includes success of the synchronous API's as well as minimizing the failures of the asynchronous workflows underlying the hyperparameter tuning jobs.
- *Secure*: Security is a top priority for AMT. In the context of HPO, this means not putting customer's data at risk. As we will elaborate below, our system does not store any customer data. In addition, we provide extra security features such as the ability to run workloads in a network-isolated mode (i.e., containers cannot make network calls) and/or in a VPC, addressing security requirements for AMT's users.

3.2 System Architecture

Sagemaker AMT provides a distributed, fault-tolerant, scalable, secure and fully-managed service for HPO. One of the key building blocks of the service is the AWS Sagemaker Training platform, which executes the training jobs and obtains the value of the objective metric for any candidate hyperparameter chosen by the Hyperparameter Selection Service. In this way, each candidate set of hyperparameters tried is associated to a corresponding Sagemaker Training Job in the user's AWS account. Using the SageMaker training platform allows AMT to scale well. Several training jobs can be run in parallel, use distributed clusters of EC2 instances, and scale HPO to large volumes of data. This includes a failure-resistant workflow with built-in retry mechanisms to guarantee robustness.

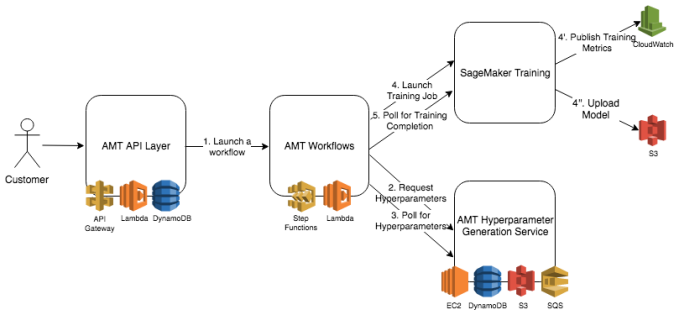

Figure 1: System architecture of SageMaker AMT. The diagram shows the different components of AMT and how they interact. A number of AWS building blocks are combined to deliver a scalable, robust, secure and fully managed service.

Sagemaker AMT's backend is built using a fully server-less architecture by means of a number of AWS building blocks. It uses AWS API Gateway, AWS Lambda, AWS DynamoDB, AWS Step Functions, AWS Cloudwatch Events in its back-end workflow. AWS API Gateway and AWS Lambda is used to power the API Layer which customers use to call various Sagemaker AMT APIs, such as Create/List/Describe/StopHyperparamaterTuningJobs. AWS DynamoDB is used as the persistent store to keep all the metadata associated with the job and also track the current state of the job. The overall system's architecture is depicted in Figure 1. Sagemaker AMT deals only with the metadata for the jobs and all customer data is handled by the Sagemaker Training platform, ensuring that no customer data is stored into the DynamoDB Table. AWS Cloudwatch Events, AWS Step Functions and AWS Lambda are used in the AMT workflows engine, which is responsible for kicking off the evaluation of hyperparameter configurations from the Hyperparameter Selection Service, starting training jobs, tracking their progress and repeating the process until the stopping criterion is met.

3.3 Challenges for HPO in the cloud

While running at scale poses a number of challenges, AMT is a highly distributed and fault tolerant system. System resiliency was one of the guiding principles when building AMT. Example failure scenarios are when the BO engine suggests hyperparameters that can run out of memory or when individual training jobs fail due to dependency issues. The AMT workflow engine is designed to be resilient against failures, and has a built-in retry mechanism to guarantee robustness.

AMT runs every evaluation as a separate training job on the SageMaker Training platform. Each training job provides customers with a usable model, logs and metrics persisted in CloudWatch. A training job involves setting up a new cluster of EC2 instances, waiting for the setup to complete, and downloading algorithm images. This introduced an overhead that was pronounced for smaller datasets. To address this, AMT puts in place compute provisioning optimizations to reduce the time of setting up clusters and getting them ready to run the training.

4 THE ALGORITHM

Next, we describe the main components and modelling choices behind the BO algorithm powering SageMaker AMT. We start with the representation of the hyperparameters, followed by the key decisions we made around the choice of the surrogate model and acquisition function. We also describe the way AMT tackles the common use case of exploiting parallel compute resources.

4.1 Input configuration

To tune the HPs of a machine learning model, AMT needs an input description of the space over which the optimization is performed. We denote by $\mathcal{H}$ the set of HPs we are working with, such as $\mathcal{H} = \{\texttt{learning rate}, \texttt{loss function}\}$. For each $h \in \mathcal{H}$, we also define $\mathcal{B}_h$ the domain of values that h can possibly take, leading to the global domain of HPs $\mathcal{B}_\mathcal{H} \triangleq \mathcal{B}_{h_1} \times \cdots \times \mathcal{B}_{h_{|\mathcal{H}|}}$. Each $h \in \mathcal{H}$ has data type continuous (real-valued), integer, or categorical, where the numerical types come with lower and upper bounds. Integer HPs are handled by working in the continuous space and rounding to the nearest integer, while categorical HPs are one-hot encoded.

It is common among ML practitioners to exploit some prior knowledge to adequately transform the space of some HP, such as applying a log-transform for a regularisation parameter. This point is important as the final performance hinges on this preprocessing. There has been a large body work dedicated to automatically finding such input transformations (e.g., see [1] and references therein). In AMT this is tackled through input warping and log scaling, which are described in the following sections.

4.2 Gaussian process modelling

Once the input HPs are encoded, AMT builds a model mapping hyperparameter configurations to their predicted performance. We follow a form of GP-based global optimisation [5, 24, 33, 39], meaning that the function f to minimize is assumed to be drawn from a GP with a given mean and covariance function. The GP is a particularly suitable model for an online learning system such as AMT, where past hyperparameter evaluations are iteratively used to pick the next ones. Since observations y collected from f are normalized to mean zero, we can consider a zero-mean function without loss of generality. The choice of the covariance function $\mathcal{K}_{\boldsymbol{\theta}}$, which depends on some Gaussian process hyperparameters (GPHPs) $\boldsymbol{\theta}$, will be discussed in detail in the next paragraph.

More formally, and given an encoded configuration $\mathbf{x}$, our probabilistic model reads $f(\mathbf{x}) \sim \mathcal{GP}(0, \mathcal{K}_{\boldsymbol{\theta}})$ and $y|f(\mathbf{x}) \sim \mathcal{N}(f(\mathbf{x}), \sigma_0^2)$, where the observation y is modelled as a Gaussian random variable with mean $f(\mathbf{x})$ and variance σ_0^2: a standard GP regression setup [47, Chapter 2]. Many choices for the covariance function (or kernel) $\mathcal{K}_{\boldsymbol{\theta}} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ are possible. The Matérn-5/2 kernel with automatic relevance determination (ARD) parametrisation [47, Chapter 4] is advocated in [51, 52, 57], where it is shown that ARD does not lead to overfitting, provided that the GPHP are properly handled. We follow this choice, which has become a de-facto standard in most BO packages.

GP hyperparameters. Our probabilistic model comes with some GPHPs $\boldsymbol{\theta}$. We highlight two possible options to treat these parameters, both of which are implemented in AMT. A traditional way of determining the GPHPs consists in finding $\boldsymbol{\theta}$ that maximises the log marginal likelihood of our probabilistic model [47, Section 5.4]. While this approach, known as empirical Bayes, is efficient and often leads to good results, [51, and follow-up work] rather advocate the full Bayesian treatment of integrating out $\boldsymbol{\theta}$ by Markov chain Monte Carlo (MCMC). In our experiments we found the latter approach to be less likely to overfit in the few-observation regime (i.e., early in the BO procedure), but is also more costly, since GP computations have to be done for every MCMC sample. In AMT, we implement slice sampling, one of the most widely used techniques for GPHPs [34, 37]. In our implementation we use one chain of 300 samples, with 250 samples as burn-in and thinning every 5 samples, resulting in an effective sample size of 10. We fix upper and lower bounds on the GPHPs for numerical stability, and use a random (normalised) direction, as opposed to a coordinate-wise strategy, to go from our multivariate problem ($\boldsymbol{\theta} \in \mathbb{R}^k$) to the standard univariate formulation of slice sampling. We observed that slice sampling is a better approach to learn the GPHPs compared to empirical Bayes, especially at the beginning of HPO where the latter is more prone to overfitting due to the small number of observations.

Input warping. It is common practice among ML practitioners to exploit some prior knowledge to adequately transform the space of some HP, such as applying a log-transform for a regularization parameter. We leverage the ideas developed in [52], where the configuration $\mathbf{x}$ is transformed entry-wise by applying, for each dimension $j \in \{1, \ldots, d\}$, $\omega(\mathbf{x}_j) \triangleq \text{BetaCDF}(\mathbf{x}_j, \alpha_j, \beta_j)$, where $\{\alpha_j, \beta_j\}_{j \in \{1,\ldots,d\}}$ are GPHPs that govern the shape of the transformation. We refer to $\boldsymbol{\omega}(\mathbf{x}) \in \mathbb{R}^d$ as the vector resulting from all entry-wise transformations. An alternative, which is the default choice in AMT, is to consider the CDF of the Kumaraswamy's distribution, which is more tractable than the CDF of the Beta distribution. A convenient way of handling these additional GHPHs is to overload our definition of the covariance function so that for, any two $\mathbf{x}, \mathbf{x}'$, $K_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}) \triangleq K_{\boldsymbol{\theta}}(\boldsymbol{\omega}(\mathbf{x}), \boldsymbol{\omega}(\mathbf{x}'))$, where $\{\alpha_j, \beta_j\}_{j \in \{1,\ldots,d\}}$ are merged within the global vector $\boldsymbol{\theta}$ of GPHPs.

4.3 Acquisition functions

Denote evaluations done up to now by $\mathcal{D} = \{(\mathbf{x}^c, y^c) \mid c \in C\}$. In GP-based Bayesian optimization, an *acquisition function* is optimized in order to determine the hyperparameter configuration at which f should be evaluated next. Most common acquisition functions $\mathcal{A}(\mathbf{x})$ depend on the GP posterior $P(f \mid \mathcal{D})$ only via its marginal mean $\mu(\mathbf{x})$ and variance $\sigma^2(\mathbf{x})$. There is an extensive literature dedicated to the design of acquisition functions. The Expected improvement (EI) was introduced by [36] and is

probably the most popular acquisition function (notably popularised by the EGO algorithm [24]). EI is the default choice for toolboxes like SMAC [20], Spearmint [51], and AMT. It is defined as $\mathcal{A}(\mathbf{x}|\mathcal{D}, \theta) = \mathbb{E}[\max\{0, y^* - y(\mathbf{x})\}]$, where the expectation is taken with respect to the posterior distribution $y(\mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x}))$. Here, $y^* = \min\{y^c \mid c \in C\}$ is the best target value observed so far. In the case of EI, the expectation appearing in $\mathcal{A}(\mathbf{x})$ is taken with respect to a Gaussian marginal consistent with the Gaussian process posterior and, as such, has a simple closed-form expression. Another basic acquisition function is provided by Thompson sampling, which is optimized by drawing a realization of the GP posterior and searching for its minimum point $\mathbf{x}$ [19, 59]. Exact Thompson sampling requires a sample from the *joint* posterior process, which is intractable. An approximation to this scheme is used in AMT, whereby *marginal* variables are sampled from $\mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x}))$ at $\mathbf{x}$ locations from a dense set. The set is obtained through a Sobol sequence generator [53] populating the search space as densely as possible. Rather than using the sampled variables directly, the resulting pseudo-random grid is used as a set of anchor points to initialize the local optimization of the EI. This scales linearly in the number of locations and works well in practice.

Other interesting families of acquisition functions include those built on upper-confidence bound ideas [55] or information-theoretic criteria [17, 18, 61, 62]. While more sample-efficient, many of these typically come with higher computational cost than the EI, which adds overhead when the tuned model is fast to train (e.g., on a small dataset). For this reason, we chose the EI as a robust acquisition function striking a balance between performance, simplicity and computational time. Alternative acquisition functions to make the EI cost-aware and steer the hyperparameter search towards cheaper configurations are described in [15, 29]. These variants mitigate the cost associated to expensive hyperparameter configurations, such as very large architectures when tuning neural network models.

4.4 Parallelism

While the BO procedure presented so far is purely sequential, many users are interested in exploiting parallel resources to speed up their tuning jobs. To address this, AMT lets them specify whether the target function should be evaluated in parallel for different candidates, and with how many parallel evaluations at a time. There are different ways of taking advantage of a distributed computational environment. If we suppose we have access to L threads/machines, we can simply return the top-L candidates as ranked by the acquisition function $\mathcal{A}$. We proceed to the next step only when the L candidate's evaluations in parallel are completed. This strategy works well as long as the candidates are diverse enough and their evaluation time is not too heterogenous, which is unfortunately rarely the case. For these reasons, AMT optimizes the acquisition function so as to induce diversity and adopts an asynchronous strategy. As soon as one of the L evaluations is done, we update the GP with this new configuration and pick the next candidate to fill in the available computation slot (making sure, of course, not to select one of the $L - 1$ pending candidates). One disadvantage is that this does not take into account the information coming from the fact that we picked the $L - 1$ pending candidates. To tackle this, asynchronous processing could be based on fantasizing [27, 51].

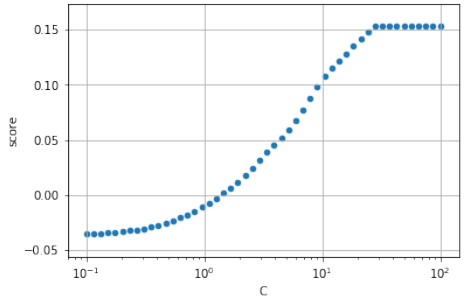

Figure 2: Change of validation set score with different values of the capacity parameter of a Support Vector Machine. Note the logarithmic scale of the capacity parameter C.

5 ADVANCED FEATURES

Beyond standard BO, AMT comes with a number of extra features to speed up tuning jobs, saving computational time and potentially finding better hyperparameter configurations. We start by describing log scaling, followed by early stopping and warm-starting.

5.1 Log Scaling

A common property of learning problems is that a linear change in validation performance requires an exponential increase in the learning capacity of the estimator (as defined by the VC dimension in [60]). To illustrate this, Figure 2 shows the relationship between the capacity parameter of SVM and validation accuracy. As a result, a wide search range is often chosen for hyperparameters controlling model capacity. For instance, a typical choice for the capacity parameter C of support vector machine is $\{10^{-9} \dots 10^{9}\}$. Note that 99% of the volume of this example search space corresponds to values of hyperparameter $C \in \{10^{7} \dots 10^{9}\}$. As a result, smaller values of C might be under-explored by BO if applied directly to such range, as it attempts to evenly explore the search space. To avoid under-exploring smaller values, a log transformation is applied to such model capacity related variables. Such transformation is generally referred to as "log scaling". Search within transformed search spaces can be done automatically by AMT, provided that the user indicates that such transformation is appropriate for a given hyperparameter via the API. For all the algorithms provided in SageMaker, such as XGBoost and Linear Learner, recommendations are given for which hyperparameters log scaling is appropriate and accelerates the tuning.

Compared to the GP input warping, which automatically detects the right scaling as the tuning job progresses, log scaling is applied from the start and comes without extra parameters to estimate. The internal warping can learn a larger family of transformations compared to the ones supported by hyperparameter scaling, and is thus always active by default. However, log scaling is useful when the appropriate input transformation is known in advance. Unlike input warping, it can be used not only with BO but also with random search.

5.2 Early Stopping

Tuning involves evaluating several hyperparameter configurations, which can be costly. Generally, a new hyperparameter configuration $\mathbf{x}_t$ may not improve over the previous ones, i.e., $f(\mathbf{x}_t) \geq f(\mathbf{x}_{t'})$ for

one or more $t' < t$. With early stopping, AMT uses the information of the previously-evaluated configurations to predict whether a specific candidate $\mathbf{x}_t$ is promising. If not, it stops the evaluation, thus reducing the overall time. This works whenever we can obtain intermediate values $f(\mathbf{x}_t^1), f(\mathbf{x}_t^2), ..., f(\mathbf{x}_t^n)$, with $f(\mathbf{x}_t^r)$ representing the value of the objective for the configuration $\mathbf{x}_t$ at training iteration r.

To implement early stopping, AMT employs the simple but effective median rule [14] to determine which HP configurations to stop early. If $f(\mathbf{x}_t^r)$ is worse than the median of the previously evaluated configurations at the same iteration r, we stop the training. An alternative is to predict future performance via a model and stop poor configurations. In our experiments, the simple median rule performed at least as well as, and often better, than predictions from linear and random forest models. A concern with the median rule is that lower-fidelities are not necessarily representative of the final values: as the training proceeds, a seemingly poor HP configuration can eventually improve enough to become the best one. To improve resilience, we only make stopping decisions after a given number of training iterations. As the total number of iterations can vary for different algorithms and use-cases, this threshold is determined dynamically based on the duration of the fully completed hyperparameter evaluations. We also considered the additional safeguard of always completing 10 hyperparameter evaluations before activating the median rule. However, this produced only slight improvements in the final objective value at the price of reduced time savings, and was therefore discarded.

5.3 Warm start

A typical ask from users running several related tuning jobs is to be able to build on previous experiments. For example, one may want to gradually increase the number of iterations, change hyperparameter ranges, change which hyperparameters to tune, or even tune the same model on a new dataset. In all these cases, it is desirable to re-use information from previous tuning jobs rather than starting from scratch. Since related tuning tasks are intuitively expected to be similar, this setting lends itself to some form of *transfer learning*. With warm start, AMT uses the results of previous tuning jobs to inform which combinations of hyperparameters to search over in the new tuning job. This can make the search for the best combination of hyperparameters more efficient.

Speeding up HP tuning with transfer learning is an active line of research. Most of this activity has focused on transferring HP knowledge across different datasets [3, 11, 14, 43, 44, 46, 49]. However, most of this previous work assumes the availability of some meta-data, describing in which sense datasets differ from each other [11]. When designing warm start, we found this assumption prohibitively restrictive in practice. Computing meta-data in real-world predictive systems is challenging due to the computational overhead or privacy reasons. We thus opted for a light-weight solution, purely based on past hyperparameter evaluations and requiring no access to meta-data. It should be noted that simple warm-starting typically works well when data is stationary, and can otherwise bias the surrogate model towards unpromising configurations. Future improvements could make the solution more robust to "negative transfer" arising from tasks that are not positively correlated.

5.4 AMT for AutoML

AMT also serves as key component for SageMaker Autopilot [7], the AutoML service of Amazon SageMaker. In Autopilot, users can upload a tabular dataset and the service will explore a complex search space, consisting of feature preprocessing, different ML algorithms and their hyperparameter spaces. For more details on how AMT is used in Autopilot we refer the readers to [7]. The application to Autopilot highlights how finding a good single ML model through hyperparameter optimization is still key for real-world applications, including in the context of AutoML. Many techniques, such as ensembling [6], can boost the accuracy by combining different models. However, when the number of models in an ensemble is large, inference time may become unacceptable, hindering practical usage. The application of AMT to Autopilot demonstrates the relevance of HPO in real-world applications, which require simple, robust and deployable models. Further, HPO is not incompatible with ensembling, and can be applied to build better ensembles.

6 USE CASES AND DEPLOYMENT RESULTS

We now consider a number of case studies highlighting the benefits of AMT, with respect to both its core algorithm and advanced functionalities. We first demonstrate the appeal of the BO strategy implemented in AMT over random search, and then turn to the empirical advantages of the advanced features we described in the previous section.

6.1 BO vs random search

A very common use case is tuning regularization hyperparameters. We demonstrate this by tuning the regularization terms `alpha` and `lambda` of the XGBoost algorithm on the `direct marketing` binary classification dataset from UCI to minimize the AUC. Figure 3 compares the performance of random search and BO, as implemented in AMT. Each experiment was replicated with 50 different random seeds, and we reported average and standard deviation. Specifically, from the left and middle plots in Figure 3, we can see that BO suggested better performing hyperparameters then random search. The right plot of Figure 3 shows that BO consistently outperforms random search across all number of hyperparameter evaluations.[2] Although BO is more competitive in the sequential setting, AMT also offers random search. It is important to note that users find random search particularly useful in the distributed setting where a large number of configurations are evaluated in parallel, reducing wall-clock time.

6.2 Log scaling

Figure 3 depicts a run of AMT with log scaling. While BO focuses on a small region of the search space, it still attempts to explore the rest of the search space. This suggests that well performing training jobs are found with low values of `alpha`. With log scaling, BO is steered towards these values and focuses on the most relevant region of the space. Models with larger learning capacity require more compute resources to perform the training phase (e.g., larger depth of trees in XGBoost leads to exponentially larger training time).

[2] A notebook to run this example on AWS SageMaker: https://github.com/awslabs/amazon-sagemaker-examples/tree/master/hyperparameter_tuning/xgboost_random_log.

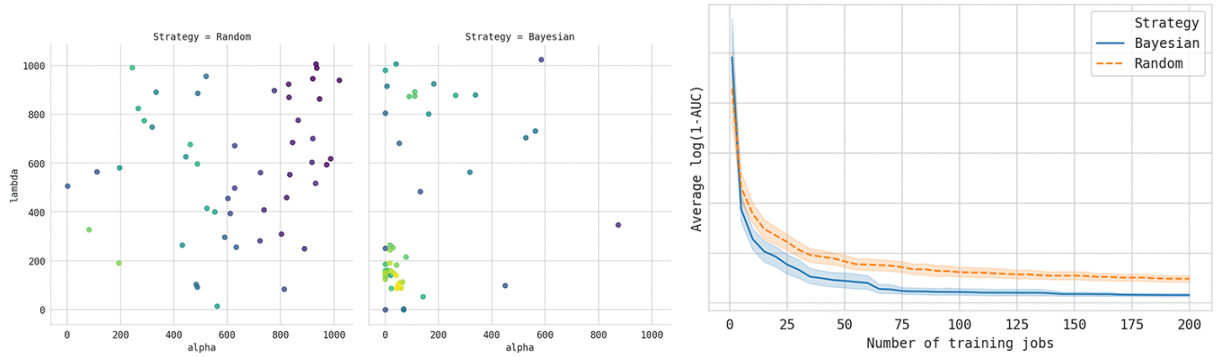

Figure 3: `Left`: Hyperparameters suggested by random search. The x-axis and y-axis are the hyperparameter values for `alpha` and `lambda` and the color of the points represent the quality of the hyperparameters (yellow: better AUC; violet: worse AUC). `Middle`: Hyperparameters are suggested by AMT through BO. `Right`: Best model score obtained so far (y-axis, where lower is better) as more hyperparameter evaluations are performed (x-axis).

Log scaling accelerates the search of good hyperparameter configurations, and also reduces the exploration of costly configurations – which usually correspond to large hyperparameter values. As the choice of scaling requires in-depth knowledge of the internals of the learning algorithms, recommendations for common algorithms on SageMaker were made available. A lesson learned from the development of log scaling is to carefully consider all possible edge-case inputs to the service. We uncovered a non-trivial failure of AMT where a user would first run a tuning job with a linear scaling of a hyperparameter (e.g., in $[0.0, 1.0]$), and use it to warm start a new tuning job with log scaling enabled. While the value 0 could have been explored in the parent job, this becomes invalid in the child job. These non-trivial issues can only be uncovered through careful examination of all possible edge-cases associated to a new functionality.

6.3 Early stopping

One of the main concerns with early stopping is that it could negatively impact the final objective values. Furthermore, the effect of early stopping is most noticeable on longer training jobs. We consider Amazon SageMaker's built-in linear learner algorithm on the Gdelt (Global Database of Events, Language and Tone) dataset in both single instance and distributed training mode.[3] Specifically, we used the Gdelt dataset from multiple years in distributed mode and from a single year in single instance mode. Figure 4 compares a hyperparameter tuning job with and without early stopping. Each experiment was replicated 10 times, and the median of the best model score so far (in terms of absolute loss, lower is better) is shown on the y-axis, while the x-axis represents time. Each tuning job was launched with a budget of 100 hyperparameter configurations to explore. AMT with early stopping not only explores the same number of HP configurations in less time, but yields hyperparameter configurations with similar performance.

6.4 Warm start

It is common to update models regularly, typically with a different choice of the hyperparameter space, iteration count or dataset. This

[3]http://www.gdeltproject.org/.

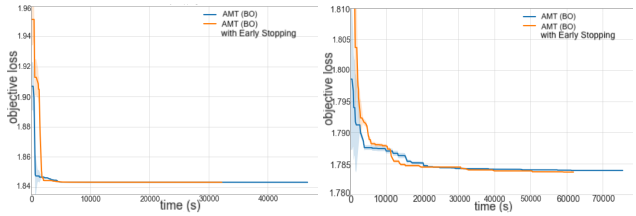

Figure 4: Absolute loss over time with and without early stopping when tuning linear learner on the Gdelt data, for single instance mode (left) and distributed mode (right). In both cases, early stopping achieves a similar loss in less time.

requires re-tuning the model hyperparameters. AMT's warm start offers a simple approach to learn from previous tuning tasks. We demonstrate this on the problem of building an image classifier and iteratively tuning it by running multiple hyperparameter tuning jobs. We focus on two simple use cases: running two sequential hyperparameter tuning jobs on the same algorithm and dataset, and launching a new tuning job on the same algorithm on an augmented dataset. We train `Amazon SageMaker`'s built-in image classification algorithm on the Caltech-256 dataset, and tune its hyperparameters to maximize validation accuracy. Figure 5 shows the impact of warm starting from previous tuning jobs. Initially, there is a single tuning job. Once complete, we launch a new tuning job by selecting the previous job as the parent. The plot shows that the new tuning job (red dots) quickly detects good hyperparameter configurations thanks to the knowledge from the parent job (black dots). As the optimization progresses, the validation accuracy reaches 0.47, thus improving over 0.33, the best previous metric found by running the tuning job from scratch. Warm start can also be applied *across different datasets*. We apply a range of data augmentations, including crop, color, and random transformations (i.e., image rotation, shear, and aspect ratio variations). To launch the last tuning job, we warm start from both previous jobs and run BO for 10 more iterations. The accuracy for the new tuning job (blue dots) improved again compared to the parent jobs (red and black dots), reaching 0.52.

As expected, this feature particularly helped users who run their tuning jobs iteratively. Interestingly, we also found this feature to be a practical enabler for users who would like to tune their

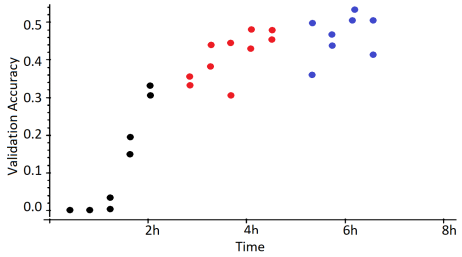

Figure 5: Validation accuracy (y-axis) over time (x-axis) for the tuned image classifier on the Caltech-256 dataset. The black dots show some iterations of AMT when starting from scratch. Warm start allows you to keep tuning the same algorithm on the same data (red dots) and on a transformed version of the data obtained via data augmentations (blue dots), with the validation accuracy continuously improving.

models for a very large number of evaluations. While this would be prohibitively expensive due to the cubical scaling of GPs, it can be achieved through running tuning jobs in a sequence, each time warm-starting from the previous one (e.g., with 500 HPO evaluations per tuning task).

6.5 Post launch performance

Many customers have adopted AMT as part of their ML workflows since the product launch in December 2017. In particular, four use cases by large enterprises for AMT are currently publicly presented as example use cases.[4] Applications include automated bidding to consume or supply electrical energy, click through rate and conversion rate predictions, malicious actor detection for DNS security, as well as general prototyping of novel ML applications. Note that all the listed applications involve business critical data for customers of AMT, which highlights the importance of building a tuning service on top of a secure training platform to guarantee secure storage and processing of data. The generality of our container-based tuning service allows us to accommodate a wide variety of learning algorithms, such as custom neural-network-based solutions implemented in TensorFlow, or gradient boosting with decision trees.

An abundance of empirical data is continuously collected to monitor AMT's performance. For instance, API communication was available at more than service-level agreement for the 99.99% of time in 2020. A number of requested tuning jobs also showcased the scalability of service. An example includes requests with 5 training jobs in parallel, each running on a cluster of 100 CPU machines of 4 cores and 16 GB of RAM. Other examples are individual training jobs involving clusters of 128 GPU accelerators. The overall service is able to process spikes of many hundreds of tuning jobs with no operational issues.

7 RELATED WORK

Before concluding, we briefly review open source solutions for gradient-free optimization in this section. Rather than providing an exhaustive list, we aim to give an overview of the tools available publicly. One of the earliest packages for Bayesian Optimization using GP as a surrogate model is `Spearmint` [51], where several important extensions including multi-task BO [57], input-warping [52] and handling of unknown constraints [13] have been introduced. The same strategy has also been implemented in other open source packages such as `BayesianOptimization` [38], `scikit-optimize` (easy to use when training scikit-learn algorithms) and `Emukit` [40].[5] Unlike using a GP as surrogate model, `SMAC` [21] uses random forest, which makes it appealing for high dimensional and discrete problems. With the growing popularity of deep learning frameworks, BO has also been implemented for all the major deep learning frameworks. `BoTorch` [2] is the BO implementation built on top of `PyTorch` [41] and `GPyTorch` [12], with an emphasis on modular interface, support for scalable GPs as well as multi-objective BO. In `TensorFlow`, there is `GPflowOpt` [28], which is the BO implementation dependent on `GPflow` [35].[6] Finally, `AutoGluon HNAS` [27] provides asynchronous BO, with and without multi-fidelity optimization, as part of `AutoGluon`.

8 CONCLUSION

We presented SageMaker AMT, a fully-managed service to optimize gradient-free functions in the cloud. Powered by BO, Sagemaker AMT brings together the state-of-the-art for hyperparameter optimization and offers it as a highly scalable solution. We gave insights into its design principles, showed how it integrates with other SageMaker's components, and shared practice for designing a fault-tolerant system in the cloud. Through a set of real-world use cases, we showed that AMT is an effective tool for HPO. It also offers a set of advanced features, such as automatic early stopping and warm-starting from previous tuning jobs, which demonstrably speed up the search of good hyperparameter configurations. Beyond performance metrics, several ML applications involve optimizing multiple constraints and alternative metrics at the same time, such as maximum memory usage, inference latency or fairness [15, 29, 42, 45]. In the future, AMT could be extended to optimize multiple objectives simultaneously, automatically suggesting hyperparameter configurations that are optimal along several criteria and search for the Pareto frontier of the multiple objectives.

ACKNOWLEDGMENTS

AMT has contributions from many members of the SageMaker team, notably Ralf Herbrich, Leo Dirac, Michael Brueckner, Viktor Kubinec, Anne Milbert, Choucri Bechir, Enrico Sartoriello, Thibaut Lienart, Furkan Bozkurt, Ilyes Khamlichi, Yifan Su, Adnan Kukuljac, Ugur Adiguzel, Mahdi Heidari.

REFERENCES

[1] J.-A. M. Assael, Z. Wang, and N. de Freitas. Heteroscedastic treed Bayesian optimisation. Technical report, preprint arXiv:1410.7172, 2014.
[2] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. BoTorch: Programmable Bayesian Optimization in PyTorch. *arXiv:1910.06403*, Oct. 2019.
[3] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag. Collaborative hyperparameter tuning. In *ICML*, 2013.
[4] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research (JMLR)*, 13, 2012.

[4] https://aws.amazon.com/sagemaker/customers

[5] https://github.com/scikit-optimize

[6] https://www.tensorflow.org/

[5] E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Technical report, preprint arXiv:1012.2599, 2010.

[6] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. *ICML*, 2004.

[7] P. Das, V. Perrone, N. Ivkin, T. Bansal, Z. Karnin, H. Shen, I. Shcherbatyi, Y. Elor, W. Wu, A. Zolic, T. Lienart, A. Tang, A. Ahmed, J. B. Faddoul, R. Jenatton, F. Winkelmolen, P. Gautier, L. Dirac, A. Perunicic, M. Miladinovic, G. Zappella, C. Archambeau, M. Seeger, B. Dutt, and L. Rouesnel. Amazon sagemaker autopilot: a white box automl solution at scale. *arXiv:2012.08483*, 2020.

[8] T. Domhan, T. Springenberg, and F. Hutter. Extrapolating learning curves of deep neural networks. In *ICML AutoML Workshop*, 2014.

[9] S. Falkner, A. Klein, and F. Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *ICML*, pages 1436–1445, 2018.

[10] M. Feurer and F. Hutter. Hyperparameter optimization. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *AutoML: Methods, Sytems, Challenges*, chapter 1. Springer, 2019.

[11] M. Feurer, T. Springenberg, and F. Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[12] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *NeurIPS*, 2018.

[13] M. A. Gelbart, J. Snoek, and R. P. Adams. Bayesian optimization with unknown constraints. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, page 250–259, 2014.

[14] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google Vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495, 2017.

[15] G. Guinet, V. Perrone, and C. Archambeau. Pareto-efficient Acquisition Functions for Cost-Aware Bayesian Optimization. *NeurIPS Meta Learning Workshop*, 2020.

[16] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2), 2001.

[17] P. Hennig and C. J. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research (JMLR)*, 2012.

[18] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. Technical report, preprint arXiv:1406.2541, 2014.

[19] M. Hoffman, B. Shahriari, and N. de Freitas. On correlation and budget constraints in model-based bandit optimization with application to automatic machine learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 365–374, 2014.

[20] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION-5*, pages 507–523, 2011.

[21] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*. Springer, 2011.

[22] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, et al. Population based training of neural networks. Technical report, preprint arXiv:1711.09846, 2017.

[23] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. Technical report, preprint arXiv:1502.07943, 2015.

[24] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 1998.

[25] Z. Karnin, T. Koren, and O. Somekh. Almost optimal exploration in multi-armed bandits. In *ICML*, 2013.

[26] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter. Learning curve prediction with Bayesian neural networks. In *International Conference on Learning Representations (ICLR)*, volume 17, 2017.

[27] A. Klein, L. Tiao, T. Lienart, C. Archambeau, and M. Seeger. Model-based asynchronous hyperparameter and neural architecture search. *arXiv preprint arXiv:2003.10865*, 2020.

[28] N. Knudde, J. van der Herten, T. Dhaene, and I. Couckuyt. GPflowOpt: A Bayesian Optimization Library using TensorFlow. *arXiv preprint – arXiv:1711.03845*, 2017.

[29] E. H. Lee, V. Perrone, C. Archambeau, and M. Seeger. Cost-aware Bayesian optimization. In *ICML AutoML Workshop*, 2020.

[30] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. Technical report, preprint arXiv:1603.06560, 2016.

[31] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, M. Hardt, B. Recht, and A. Talwalkar. Massively parallel hyperparameter tuning. Technical Report 1810.05934v4 [cs.LG], ArXiv, 2019.

[32] E. Liberty, Z. Karnin, B. Xiang, L. Rouesnel, B. Coskun, R. Nallapati, J. Delgado, A. Sadoughi, Y. Astashonok, P. Das, C. Balioglu, S. Chakravarty, M. Jha, P. Gautier, D. Arpin, T. Januschowski, V. Flunkert, Y. Wang, J. Gasthaus, L. Stella, S. Rangapuram, D. Salinas, S. Schelter, and A. Smola. Elastic machine learning algorithms in Amazon SageMaker. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020.

[33] D. J. Lizotte. *Practical Bayesian optimization*. PhD thesis, University of Alberta, 2008.

[34] D. J. C. Mackay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[35] A. G. d. G. Matthews, M. van der Wilk, T. Nickson, K. Fujii, A. Boukouvalas, P. León-Villagrá, Z. Ghahramani, and J. Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40), apr 2017.

[36] J. Mockus, V. Tiesis, and A. Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 1978.

[37] I. Murray and R. P. Adams. Slice sampling covariance hyperparameters of latent Gaussian models. In *NeurIPS*, pages 1732–1740, 2010.

[38] F. Nogueira. Bayesian optimization: Open source constrained global optimization tool for Python, 2014.

[39] M. A. Osborne, R. Garnett, and S. J. Roberts. Gaussian processes for global optimization. In *Proceedings of the 3rd Learning and Intelligent OptimizatioN Conference (LION 3)*, 2009.

[40] A. Paleyes, M. Pullin, M. Mahsereci, N. Lawrence, and J. González. Emulation of physical processes with Emukit. In *Second Workshop on Machine Learning and the Physical Sciences, NeurIPS*, 2019.

[41] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

[42] V. Perrone, M. Donini, K. Kenthapadi, and C. Archambeau. Fair Bayesian optimization. In *ICML AutoML Workshop*, 2020.

[43] V. Perrone, R. Jenatton, M. Seeger, and C. Archambeau. Multiple adaptive Bayesian linear regression for scalable Bayesian optimization with warm start. *NeurIPS Meta Learning Workshop*, 2017.

[44] V. Perrone, R. Jenatton, M. W. Seeger, and C. Archambeau. Scalable hyperparameter transfer learning. *NeurIPS*, 2018.

[45] V. Perrone, I. Shcherbatyi, R. Jenatton, C. Archambeau, and M. Seeger. Constrained Bayesian optimization with max-value entropy search. In *NeurIPS Meta Learning Workshop*, 2019.

[46] V. Perrone, H. Shen, M. W. Seeger, C. Archambeau, and R. Jenatton. Learning search spaces for Bayesian optimization: Another view of hyperparameter transfer learning. *NeurIPS*, 32, 2019.

[47] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[48] E. Real, C. Liang, D. So, and Q. Le. Evolving machine learning algorithms from scratch. In *ICML*, 2020.

[49] D. Salinas, H. Shen, and V. Perrone. A quantile-based approach for hyperparameter transfer learning. *ICML*, 2020.

[50] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *IEEE*, 2016.

[51] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *NeurIPS*, pages 2960–2968, 2012.

[52] J. Snoek, K. Swersky, R. S. Zemel, and R. P. Adams. Input warping for Bayesian optimization of non-stationary functions. Technical report, preprint arXiv:1402.0929, 2014.

[53] I. M. Sobol. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4), 1967.

[54] J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter. Bayesian optimization with robust Bayesian neural networks. In *NeurIPS*, pages 4134–4142, 2016.

[55] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *ICML*, page 1015–1022, 2010.

[56] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Information-theoretic regret bounds for Gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58, 2012.

[57] K. Swersky, J. Snoek, and R. P. Adams. Multi-task Bayesian optimization. In *NeurIPS*, pages 2004–2012, 2013.

[58] K. Swersky, J. Snoek, and R. P. Adams. Freeze-thaw Bayesian optimization. Technical report, preprint arXiv:1406.3896, 2014.

[59] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, pages 285–294, 1933.

[60] V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[61] J. Villemonteix, E. Vazquez, and E. Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4):509–534, 2009.

[62] Z. Wang and S. Jegelka. Max-value entropy search for efficient Bayesian optimization. *ICML*, 70, 2017.