AE: A domain-agnostic platform for adaptive experimentation

Eytan Bakshy **Lili Dworkin** **Brian Karrer** **Konstantin Kashin**
Benjamin Letham **Ashwin Murthy** **Shaun Singh**
Facebook
{ebakshy, lilidworkin, briankarrer, kkashin,
bletham, amurthy, shaundsingh}@fb.com

Abstract

We describe AE, a machine learning platform for adaptive experimentation (e.g., Bayesian optimization, bandit optimization) that automates the process of sequential experimentation. Unlike existing solutions that are oriented primarily towards optimizing ML hyperparameters and simulations, AE is designed with online experimentation (A/B tests) in mind. Motivated by real-world examples from Facebook, we present a design for ML-assisted experimentation with multiple objectives, noisy, non-stationary measurements, and data from multiple experimentation modalities.

1 Introduction

Experimentation—the process of systematically exploring multiple variants—is a ubiquitous practice among engineers and researchers alike for improving the performance of machine learning models and routine product improvement. Increasing the throughput of experimentation can yield substantial returns [Azevedo et al., 2018], but is often challenging due to the complexity of real-world environments. Such concerns include multiple competing objectives, non-stationary measurements, multi-variate noisy observations, and heterogeneous, contextual effects [Letham et al., 2018]. In addition, an experimenter is called upon to make numerous decisions including what variations to try in what order, what to measure, how long to measure it, and what trade-offs between metrics are acceptable.

We introduce AE, a platform for managing sequential experimentation written in Python that confronts the complexity of using experiments for real-world optimization. On one hand, AE was designed to be easy-to-use and robust as a library for manual management of complex real-world experiments (for example through a Jupyter notebook environment). On the other hand, it provides an abstraction for specifying domain-specific experimentation recipes that encode routines for best-practices. These can be executed automatically, thereby democratizing efficient optimization through sequential experimentation. We motivate the system using real-world applications found in the Internet industry. It has successfully been applied to thousands of problems at Facebook, spanning A/B testing, hyperparameter optimization, infrastructure optimization, and hardware design.

AE provides first-class support for multiple experimentation modalities (e.g., online and offline experimentation). Furthermore, AE stores meta-data about the parameters being optimized and the experimenter's objectives. In turn, this data can support meta-analyses and meta-learning, whereby information from prior experiments can be used to improve model accuracy and optimization speed [Feurer et al., 2018]. From the perspective of the AE user, sequential experiments can be implemented using a high-level, object-oriented API. In total, AE manages the experimentation process and empowers experimenters to focus on learning about and optimizing their system.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

1.1 Related Work

In recent years, a number of tools – some open-source and others proprietary – have emerged to tackle the problem of sequential experimental design. Many of these tools have been developed with a focus on selecting optimal hyperparameters in machine learning models.

One of the main approaches to solving sequential experimental design problems is Bayesian optimization. Open-source packages such as Spearmint [Snoek et al., 2012] and BayesOpt [Martinez-Cantin, 2014] provide lightweight interfaces for setting up the optimization problem. MOE is a library that was designed to manage both Bayesian and bandit optimization [Clark et al., 2014]. However, none of these packages provide abstractions for dealing with non-stationary measurements, nor do they effectively manage and track human-in-the-loop optimization, which we have found to be an important feature for enabling widespread adoption. Among closed-source tools, SigOpt and Google's Vizier are the most well-known [Golovin et al., 2017] and are both implemented as hosted services. Apart from Spearmint, which allows for metric constraints, none of the aforementioned tools support dealing with competing objectives.

Beyond Bayesian optimization for continuous search spaces, libraries for optimizing complex search spaces have emerged to meet the needs of automating selection and training of models such as neural networks. HyperOpt, for example, provides semantics for composable, nested search spaces [Bergstra et al., 2015]. However, the library does not support the management of optimization needed for online experiments. Since exploration of search spaces when training models such as neural networks is resource intensive, systems are emerging that can interface with scheduling infrastructure and manage computational resources. This enables a new set of algorithms that are resource-aware and can manage parallel execution of evaluations. As one such example, Tune is a platform for model selection implemented using the Ray distributed computing framework [Liaw et al., 2018].

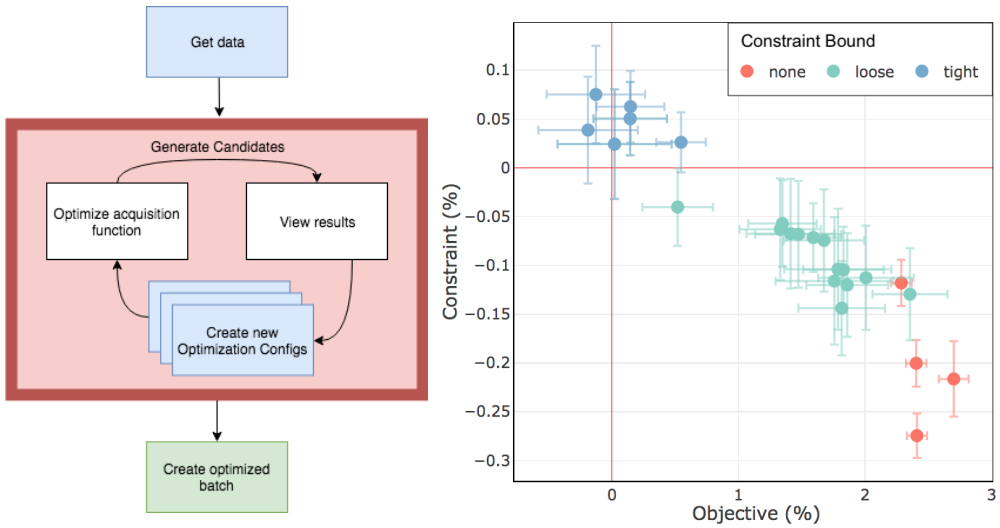
2 Motivation

Parameters are the controllable knobs that an experimenter can vary that may alter various metrics collected on experimental units. Examples of parameters include number of items to retrieve from a recommender system, cache sizes, or bit rates. A basic experimentation workflow consists of specifying experimental conditions, each defined as a set of parameter values. In the case of A/B tests, each condition is associated with a weight corresponding to the assignment probability. A collection of conditions and weights is called a *batch*. In the absence of past data, batches are generated randomly or quasi-randomly for exploration. When data does exist, one can apply explore-exploit algorithms such as Bayesian optimization or Thompson Sampling [Chapelle and Li, 2011] informed by ML models trained on past data.

Figure 1: The basic optimization flow for online experiments.

The batch is subsequently deployed to a population of experimental units (e.g., user IDs), with the fraction of the population receiving each condition proportional to the weight. An example of a system used for deploying conditions to a population is PlanOut [Bakshy et al., 2014], which randomly assigns units in a way that is orthogonal across experiments using a deterministic hashing function. Like many other industrial experimentation systems, PlanOut allows experimenters to specify *namespaces* to maintain mutual exclusion between highly related interventions (e.g., changes to the same ranking model). Multiple batches from the same experiment always take place within the same namespace. Following deployment, the experimenter collects data on a pre-specified objective and set of constraints for each condition. Based on the results, the experimenter might try a new set of conditions they deem promising as well as retrying old ones. This flow is often repeated several times in an optimization loop until the experimenter decides to stop. This optimization flow is presented in Figure 1.

A major challenge in online experiments is selecting which metrics to optimize [Deng and Shi, 2016], and how those metrics should be traded off. Part of the difficulty stems from metrics that are inherently high variance and hard to move (i.e., there is a low signal-to-noise ratio). Another consideration is that most real-world systems involve balancing various competing objectives. As a result, we have found that human involvement in the optimization loop is common and preferred among users of AE at Facebook. Experimenters may evaluate multiple sets of objectives and outcome constraints, and then introspect the candidates recommended by algorithms such as Bayesian optimization to understand the trade-offs inherent in the optimization in an interactive, human-in-the-loop process (see Figure 2a, which extends the candidate generation step in the general workflow in Figure 1 to reflect human intervention). This introspection of candidates occurs through visualizations such as the one in Figure 2b, which presents a typical trade-off an experimenter may encounter. Ultimately, we have found that users rely on rich domain-specific knowledge to interpret and make decisions using Pareto-front optimization and visualizations. As a result, we expect that human-in-the-loop optimization will remain a routine and beneficial practice, at least in the medium-term.



(a) Human-in-the-loop

(b) Trade-off plot

Figure 2: (a) Schematic diagram of a human-in-the-loop experimentation workflow and (b) trade-offs obtained under different constraints.

2.1 Design Requirements

AE was designed with real-world, online experimentation in mind, which has the following unique considerations when compared to vanilla blackbox optimization:

1. **Noisy function evaluations** - Outcomes generated by A/B tests often have high levels of noise. This noise is generally controllable (e.g., scales downwards with the square root of the number of samples allocated to each condition).

2. **Domain-specific logic for candidate generation and deployment** - Different types of experiments may require different modeling strategies and iteration procedures depending on the structure of the problem, budget, and amount of time it takes to produce a measurement. The system should provide a way to specify an analysis plan that defines the iteration procedure for a given experiment type.

3. **Multiple experimentation modalities and multi-fidelity optimization** - The system should support applications where experimental results of variable fidelity can be obtained from different sources (e.g., high-fidelity data from online experimentation and low-fidelity proxies from offline simulators).

4. **Non-stationary outcomes in time** - Outcomes measured early on in the experiment may suffer from some initial upward or downward bias before approaching a stable state. Underlying reasons include population shift, novel effects, and learning.

5. **Multi-objective optimization** - Experiments typically aim to optimize multiple objective functions, which cannot be easily combined into a single objective. The system should support a human-in-the-loop model and be able to track candidates generated using different optimization procedures, with the goal of ensuring reproducibility and facilitating meta-analysis.

3 System Architecture

In this section, we present what happens at each stage of the optimization flow in Figure 1, introduce AE's abstractions, and discuss how the design addresses the aforementioned requirements.

3.1 Initializing an Experiment

An AE *Experiment* is the canonical unit for managing an optimization sequence. The core user-defined components needed to configure an Experiment, shown in Figure 3, are:

- *SearchSpace* - Defines *Parameters* to search over (parameter type and whether the search is over a range or a discrete set of choices). Additionally, AE supports *ParameterConstraints*, represented as linear constraints on a set of parameters (i.e., $w \cdot x \leq b$, where x is a vector of parameter values and w is a weight vector). Such constraints are common to many online experiments (e.g., restricting the total number of items to be retrieved from different recommendation engines or restricting sizes of buffers to not exceed some total memory limit).

- *OptimizationConfig* - Specifies both an *Objective* to optimize as well as a set of *OutcomeConstraints* on other outcome functions. We have found that approaching multi-objective optimization as a constrained optimization problem was often much more intuitive and easier to work with for users of AE compared with other means of combining objectives, such as asking users to select linear combinations of objectives. The problem now reduces to maximizing the objective metric while satisfying constraints on the other metrics. As illustrated in Figure 3, the Objective and OutcomeConstraints are defined in terms of *Metrics*, which provide the logic for how AE interfaces with external data fetching APIs to compute outcomes. This is described in more detail below in Section 3.4.

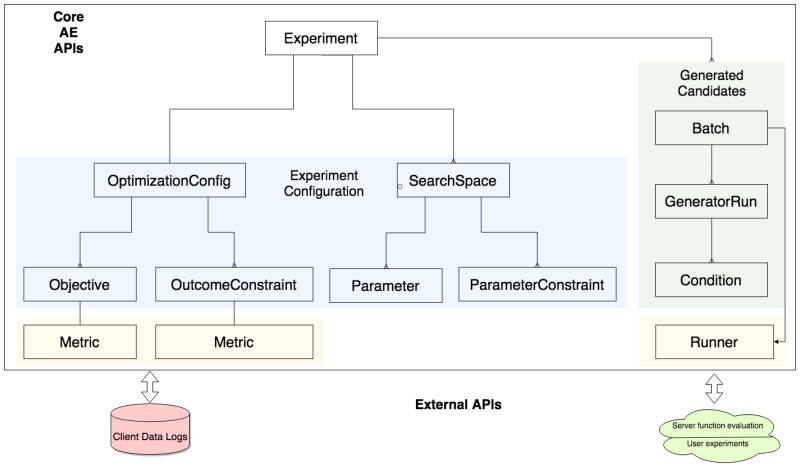


Figure 3: Core AE Abstractions

3.2 Creating an Exploratory Batch

The optimization procedure typically starts with *Conditions* (sets of parameter values) generated using a space-filling design, such as a Sobol sequence [Owen, 1998]. These Conditions (and associated weights) are grouped and run concurrently in a *Batch*. In contrast to other Bayesian optimization procedures that run one configuration at a time, online experiments necessitate running multiple points simultaneously due to the non-stationarity of the outcomes. This eliminates confounding factors arising from configurations being run at different times [Bakshy et al., 2014]. AE interfaces with external systems via two main APIs: deploying a batch and getting back data.

3.3 Deploying a Batch

The Batch is deployed using a *Runner*, which maps AE's generic condition representation to that of any external evaluation system. A default Runner is defined on the Experiment, but can be overridden for any given Batch, as shown in Figure 3. The Runner can be fully customized for each application, ranging from interfaces to A/B testing frameworks, such as PlanOut [Bakshy et al., 2014], to ML training infrastructure. The flexibility of defining different Runners within a given experiment facilitates advanced applications such as online / offline optimization discussed in Section 4.

3.4 Getting Data

Multi-objective optimization naturally requires multiple metrics to be measured. Similar to AE's support of multiple Runners within an experiment, the system supports different kinds of Metrics (and thus different procedures for querying data) within an experiment, again useful for mixed-mode experimentation. Querying for data will automatically group and dispatch queries to the corresponding data stores. Upon receiving the query results, AE will construct data in a standardized format that is easily used within our modeling and analysis stack.

3.5 Generating Candidates

The candidate generation process may use any sequential experimentation algorithm, including bandit optimization or Bayesian optimization. Bayesian optimization is a sample-efficient approach to solving blackbox optimization problems, most commonly involving multiple (e.g., 2-24) continuous parameters being optimized. Bayesian optimization relies on a probabilistic surrogate model, such as a Gaussian process (henceforth GP), and an acquisition function, which identifies new candidates by navigating some explore/exploit trade-offs. In our experience, GPs are surprisingly accurate models of many real-world parameter tuning and policy optimization problems. The specific models and acquisition functions vary across use cases – we routinely use a noisy variant of expected improvement for most online experiments [Letham et al., 2018], but use the knowledge gradient for hyperparameter optimization with dataset downsampling [Wu and Frazier, 2017].

AE simplifies the development of new models and acquisition functions by providing an interface between AE objects and PyTorch types, referred to as a *Generator*. Implementing a new model requires only dealing with PyTorch structures and there is a clean separation between AE and the model internals. AE interfaces with botorch, a core set of models and acquisition functions which are implemented in PyTorch/GPyTorch [Paszke et al., 2017, Gardner et al., 2018]. Botorch supports GPU-acceleration, fast and accurate approximate uncertainty estimation for scaling to large datasets, deep kernel learning for handling many features, and implementations of parallel acquisition functions for efficient generation of batches, such as those found in Wilson et al. [2018].

In candidate generation, the PyTorch model returns a Torch tensor of generated points, which the modeling stack transforms into Condition objects with corresponding weights, possibly indicating the desired sample size. The modeling and candidate generation stack ultimately returns a *GeneratorRun* object, which contains these Conditions and weights, as well as the OptimizationConfig, SearchSpace, and other inputs into the Generator. The GeneratorRun is then stored on the Batch, as shown in Figure 3, which allows AE to track the provenance of the Conditions within a Batch, thus ensuring reproducibility. This is important given that experimenters frequently create amalgam Batches with multiple GeneratorRun objects (each with potentially different OptimizationConfigs and/or SearchSpaces) to address the concerns they face when choosing objectives raised in Section 2.

3.6 Automated Analyses

AE was designed in part to be easy-to-use for experimenters who want to manually set up, run, and analyze their experiments. It was also designed with the goal of automating sequential, machine-learning assisted experimentation for non-experts. When an experimenter deploys a Batch, AE can automatically schedule an analysis that will poll for data, determine when an experiment is ready to analyze, and generate candidates for review.

The analysis can be customized by the experimenter using the *AnalysisPlan* abstraction. For example, the user can define the specific criteria for determining when results are ready, how many candidates to generate, and how to borrow information across batches (e.g., via a multi-task model with an ICM kernel, which accounts for non-stationarity in the data) [Letham and Bakshy, 2018]. Modeling strategies and candidate generation procedures can be explicitly defined on the AnalysisPlan, or left to AE to determine automatically using a set of reasonable defaults given the structure of the problem. The AnalysisPlan thus defines a playbook for how to analyze and iterate on experiments that can be tailored for specific domains. Examples of AnalysisPlans include tools for automated content optimization of marketing campaigns, logic for controlling hyperparameter optimization, and services for automatically tuning the HHVM JIT [Letham et al., 2018].

Importantly, these analyses are designed to be reproducible, since every run of the analysis is logged via an *AnalysisRun* that tracks the SearchSpace, OptimizationConfig, and other inputs into the analysis. Reproducibility of the optimization allows us to re-run analyses after more data has arrived, and to track which optimization procedures led to which final results. The resulting central repository of analyses paves the way for meta-learning.

3.7 Storage

AE has modular, extensible support for saving and loading experiments in different formats to meet the needs of diverse users, ranging from researchers who prefer lightweight, transportable storage to production applications requiring a centralized, high-performance database. AE both supports serializing experiments to JSON and storing experiments in a MySQL database via SQLAlchemy. The MySQL-based storage facilitates meta-learning through easy querying of past experiments run by the same team or experiments targeting similar parameters or metrics.

4 Applications and Extensions

Here we describe examples of applications that utilize the AE platform as a way of illustrating the platform's flexibility.

Combined online/offline experimentation For some applications, such as tuning recommender systems, offline simulators are available that provide fast, but biased, estimates of the outcomes of online experiments. We use AE in this setting to manage both the online and offline modes of experimentation, and to use offline experiments to improve candidate generation for online experiments [Letham and Bakshy, 2018]. Viewed broadly, this example demonstrates AE's support for multi-fidelity optimization.

We use a multi-task GP with a kernel that models covariance across the offline and online sets of experiments [Swersky et al., 2013]. AE is then used to generate new batches, which can be configured to either deploy to the offline simulator or online system. This is easily supported in AE by specifying the appropriate Runner configuration on the new batch. Such experiments can run in a fully automated mode, continuously deploying candidates to both the online and offline systems according to some regular schedule.

Contextual Tuning In a standard parameter tuning setup, AE aims to identify a single optimal set of parameter values. AE can also identify optimal parameter values depending on input features (*Contexts*). As an illustrative example, AE has been used extensively to optimize video uploads and playback according to real-time contextual information. For uploads, upload quality is tuned according to information such as available bandwidth, video duration, and initial quality. For playback, bitrate values are chosen by AE, according to connection type, the number of stalls, number of seconds played so far, etc. Tuning separate parameters per context can be seen as providing a

solution to the contextual bandit problems [Krause and Ong, 2011, Bottou et al., 2013, Li et al., 2015].

Hyper-parameter Optimization AE is used for thousands of hyperparameter experiments each month at Facebook using a Bayesian optimization algorithm similar to the method described in Snoek et al. [2012]. These use-cases are generally implemented as simplifications or constrained forms of the general AE structure. Non-stationarity is generally not considered in these applications, reducing Batch as synchronization primitive to a singleton Condition. Multi-objective optimization is also less of a concern given the presence of a well-defined objective (e.g., improving model prediction accuracy). However, at Facebook, models are trained offline and then experimented with online. In the future, this two-step experimentation process, beyond the scope of libraries aimed only at hyperparameter tuning, could be confronted using AE in a straightforward manner.

5 Conclusion

The AE system is built to handle a wide variety of sequential optimization applications. It addresses many of the complexities of sequential field experiments such as complex user preferences, noisy, non-stationary measurements, and multi-fidelity data. The components of the system are modular, which allows user customization of different processes like deployment, data fetching, analysis, and storage. Native support for modeling over multiple types of data (i.e., fidelities) is vital for field experimentation with supporting data (e.g., offline simulations).

A limitation of the current implementation is the lack of sophisticated scheduling of new batches and automated stopping of existing batches. AE currently supports basic scheduling sufficient for an optimization loop. Incorporating a more advanced scheduler, for example containing validation checks on new candidate batches, into AE is a direction for future work. Similarly, automated stopping of specific conditions within a batch is a common feature of hyperparameter tuning libraries, which likely requires modification to be suitable for online experimentation.

References

Eduardo M Azevedo, Alex Deng, José Luis Montiel Olea, Justin Rao, and E Glen Weyl. A/b testing. *Working paper*, 2018.

Eytan Bakshy, Dean Eckles, and Michael S. Bernstein. Designing and deploying online field experiments. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 283–292, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2744-2. doi: 10.1145/2566486.2567967.

James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science Discovery*, 8(1):014008, 2015. URL `http://stacks.iop.org/1749-4699/8/i=1/a=014008`.

Léon Bottou, Jonas Peters, Joaquin Quiñonero-Candela, Denis X Charles, D Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. Counterfactual reasoning and learning systems: the example of computational advertising. *The Journal of Machine Learning Research*, 14(1):3207–3260, 2013.

Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2249–2257. Curran Associates, Inc., 2011. URL `http://papers.nips.cc/paper/4321-an-empirical-evaluation-of-thompson-sampling.pdf`.

Scott Clark, Eric Liu, Peter Frazier, JiaLei Wang, Deniz Oktay, and Norases Vesdapunt. Moe: A global, black box optimization engine for real world metric optimization, 2014. URL `https://github.com/Yelp/MOE`.

Alex Deng and Xiaolin Shi. Data-driven metric development for online controlled experiments: Seven lessons learned. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD, pages 77–86, 2016.

Matthias Feurer, Benjamin Letham, and Eytan Bakshy. Scalable meta-learning for bayesian optimization. *arXiv preprint arXiv:1802.02219*, 2018.

Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems 31*, NIPS, 2018.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John 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*, KDD '17, pages 1487–1495, New York, NY, USA, 2017. ACM. doi: 10.1145/3097983.3098043. URL http://doi.acm.org/10.1145/3097983.3098043.

Andreas Krause and Cheng S. Ong. Contextual Gaussian process bandit optimization. In *Advances in Neural Information Processing Systems 24*. 2011.

Benjamin Letham and Eytan Bakshy. Bayesian optimization for policy search via mixed online-offline experimentation. *Working paper*, 2018.

Benjamin Letham, Brian Karrer, Guilherme Ottoni, Eytan Bakshy, et al. Constrained bayesian optimization with noisy experiments. *Bayesian Analysis*, 2018.

Lihong Li, Jin Young Kim, and Imed Zitouni. Toward predicting the outcome of an A/B experiment for search relevance. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining*, WSDM, pages 37–46, 2015.

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

Ruben Martinez-Cantin. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *J. Mach. Learn. Res.*, 15(1):3735–3739, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2750364.

Art B. Owen. Scrambling Sobol' and Niederreiter-Xing points. *Journal of Complexity*, 14:466–489, 1998.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, NIPS, 2012.

Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems 26*, pages 2004–2012. 2013.

James Wilson, Frank Hutter, and Marc Deisenroth. Maximizing acquisition functions for bayesian optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9905–9916. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8194-maximizing-acquisition-functions-for-bayesian-optimization.pdf.

Jian Wu and Peter I. Frazier. Continuous-fidelity bayesian optimization with knowledge gradient. *NIPS Workshop on Bayesian Optimization*, 2017.