BOHB: Robust and Efficient Hyperparameter Optimization at Scale

Stefan Falkner, Aaron Klein, Frank Hutter

Presented by: Muhammed Ugur

Hyperparameter Overview

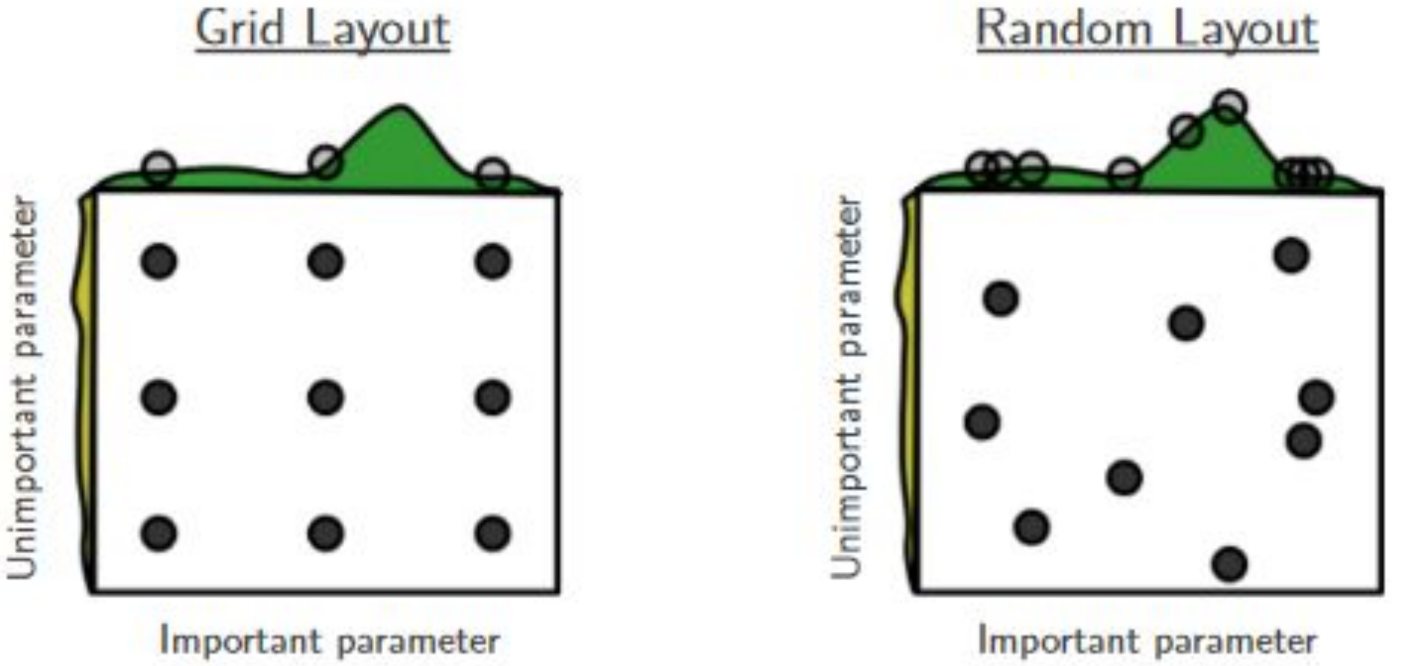
- Hyperparameters = configurations
  - Continuous, discrete, categorical
- For ML
  - Architectural
    - Number and width of layers
  - Optimization
    - Learning rate, batch size, epochs, activation function
  - Regularization

Hyperparameter Overview

- Hyperparameter optimization (HPO) problem
  - Given a set of hyperparameters and an objective function
  - Select optimal assignment that minimizes objective function
  - Treat this as a search space

- For ML
  - Models are highly sensitive to internal configurations
  - Hyperparameter optimization is critical
  - Objective function is to minimize validation error

Example



Grid Layout
Unimportant parameter
Important parameter
Random Layout
Unimportant parameter
Important parameter

[1]

Motivation

Ideal HPO method

- Output best configuration
  - If constrained, output best configuration at any given time
- Scales efficiently
  - Exponential search space
  - Evaluating a configuration requires training and validation
- Utilizes resources efficiently
  - Parallelization
- Robustness
  - Different deep learning methods
  - Different types of hyperparameters

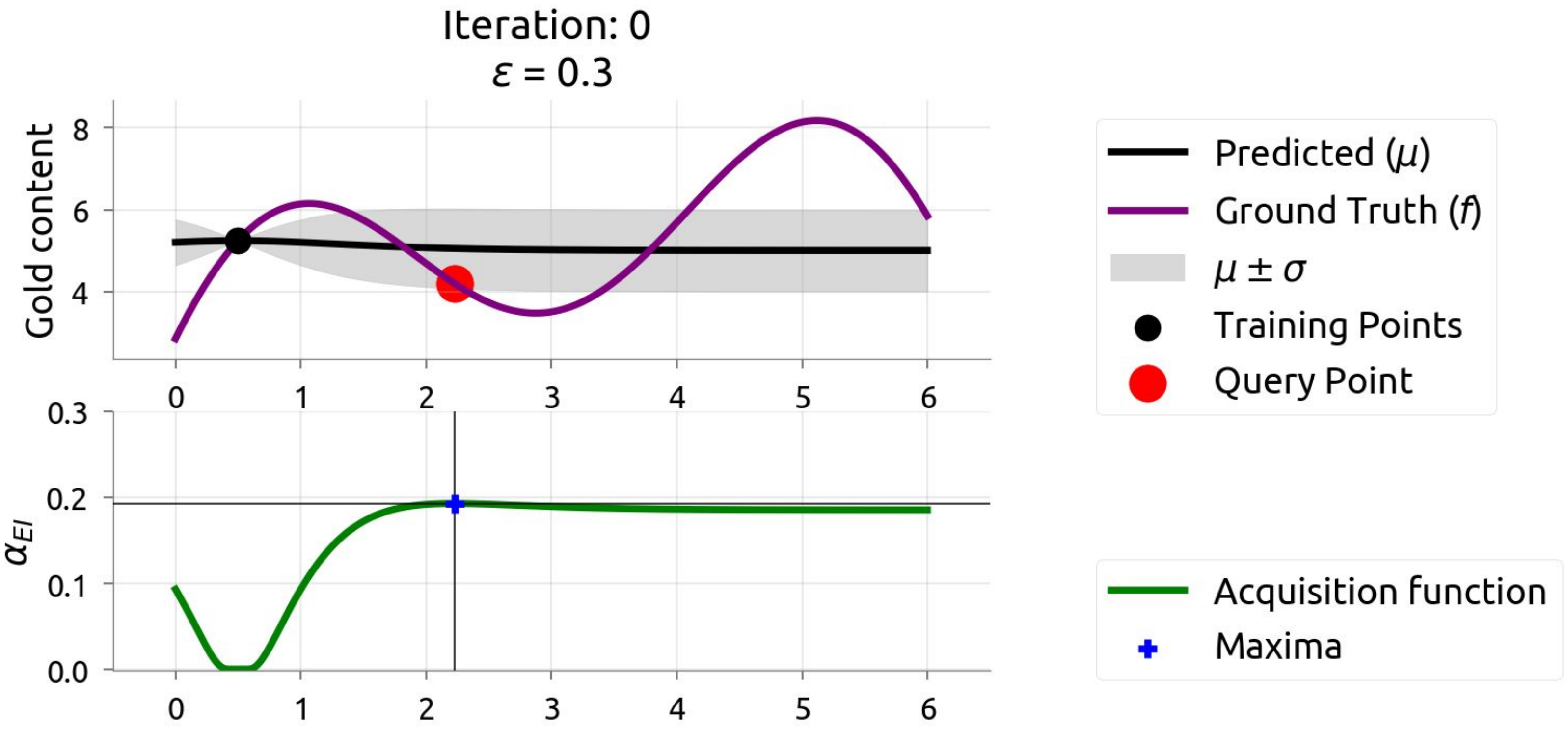
BOHB: High-level Idea

- Main HPO methods
  - Random search
  - Bayesian optimization (model-based)
  - Hyperband (bandit algorithms)
- BOHB combines Bayesian optimization and Hyperband
  - Attempts to satisfy all ideal goals

Bayesian Optimization

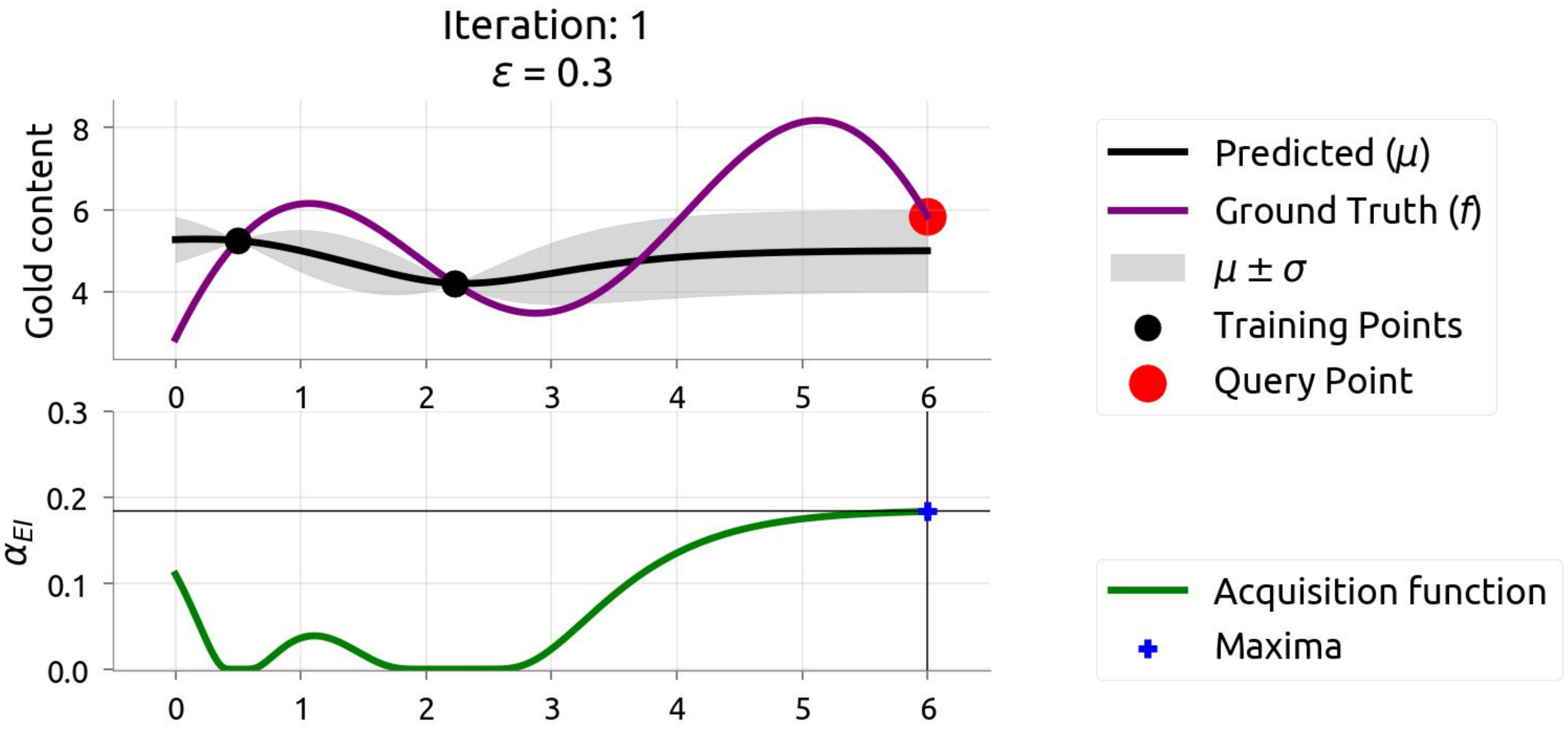
- **High-level idea:** use prior information to guide tuning
  - Balance out evaluating unknown regions (exploration) with known regions (exploitation)
  - Models underlying objective function distribution
- At each step
  - Determine the next best configuration to evaluate based on prior info
    - Uses acquisition function (tells us how desirable it is to evaluate a configuration)
    - e.g. maximize expected improvement (how much can we improve)
  - Evaluate
  - Update probabilistic model
  - Continue (until convergence or resource constraint)

Bayesian Optimization



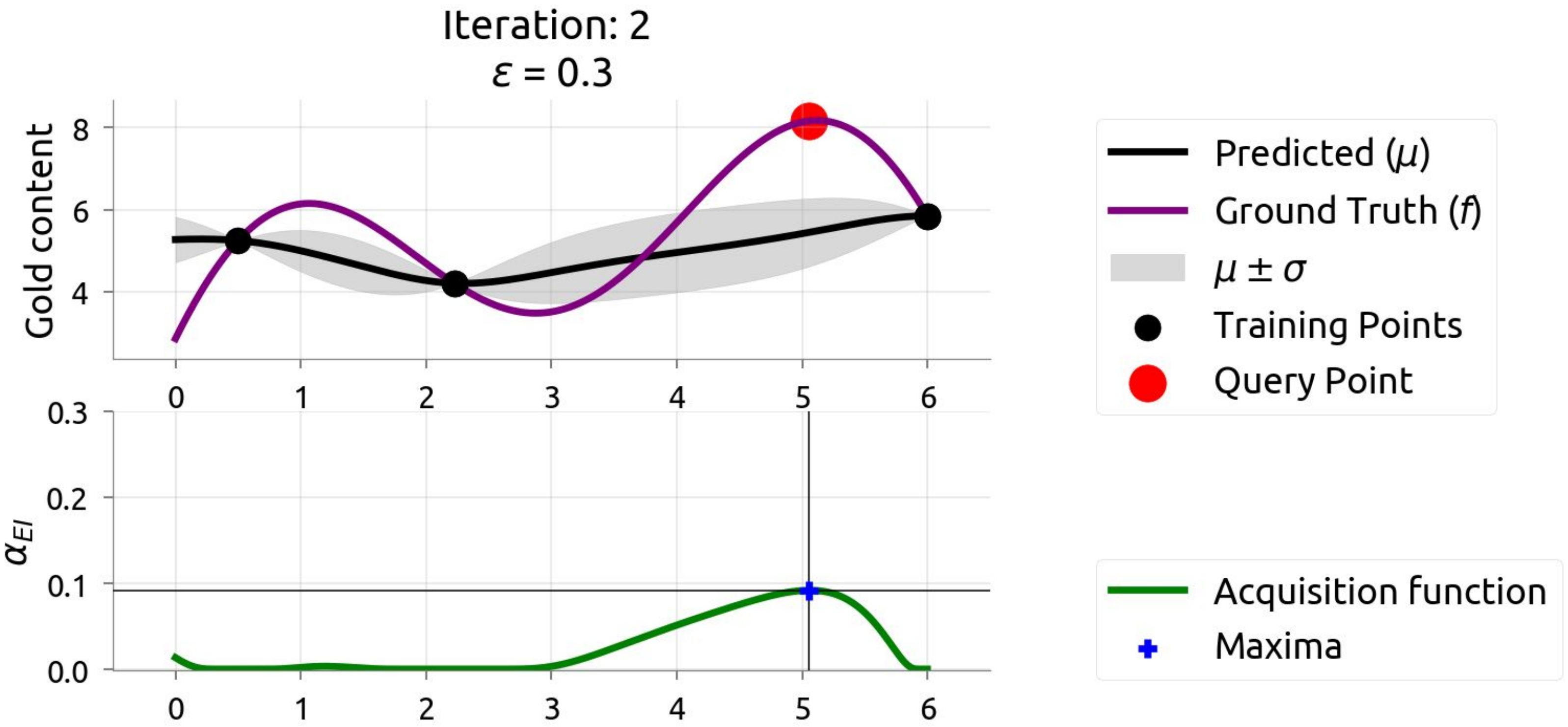
[6]

Bayesian Optimization



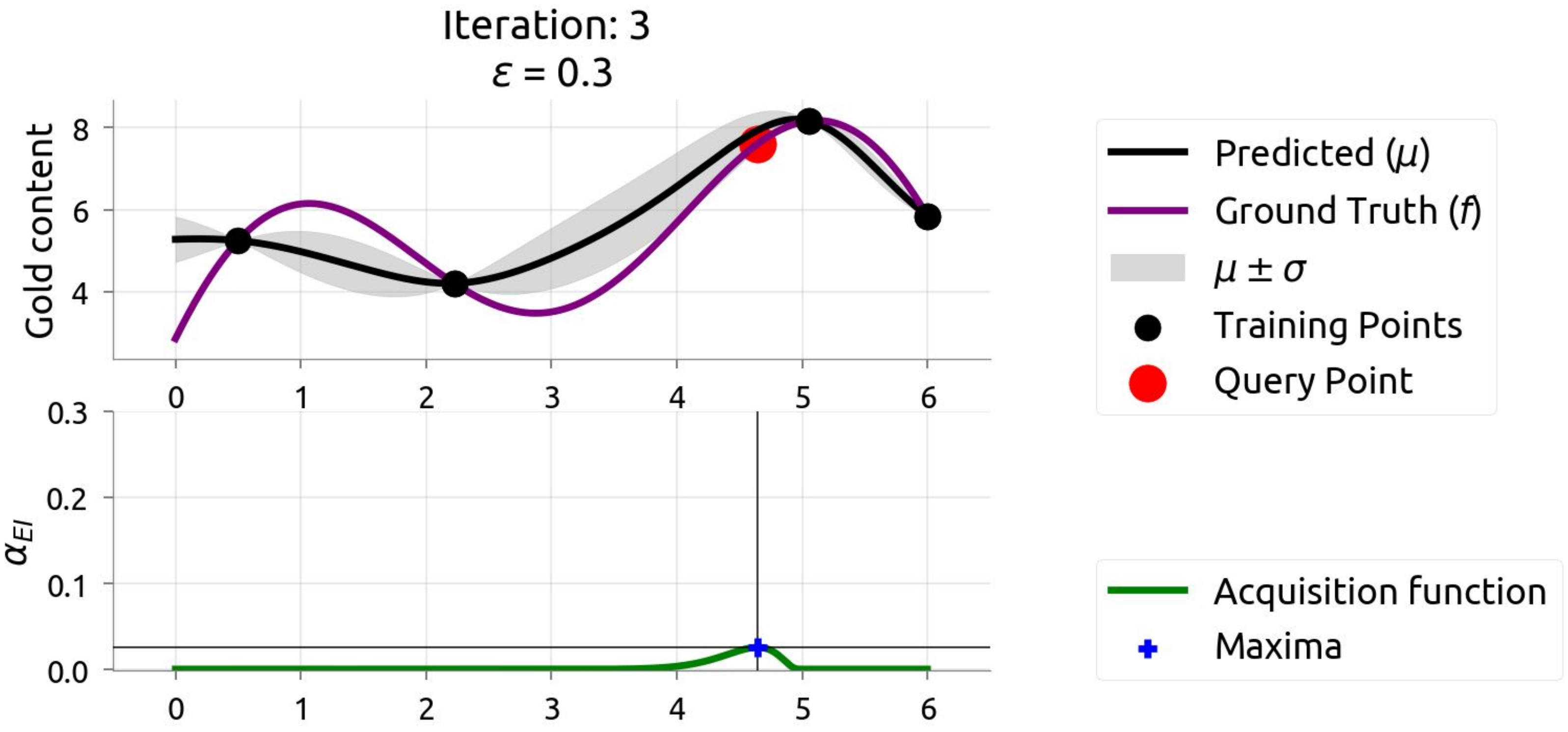
[6]

Bayesian Optimization



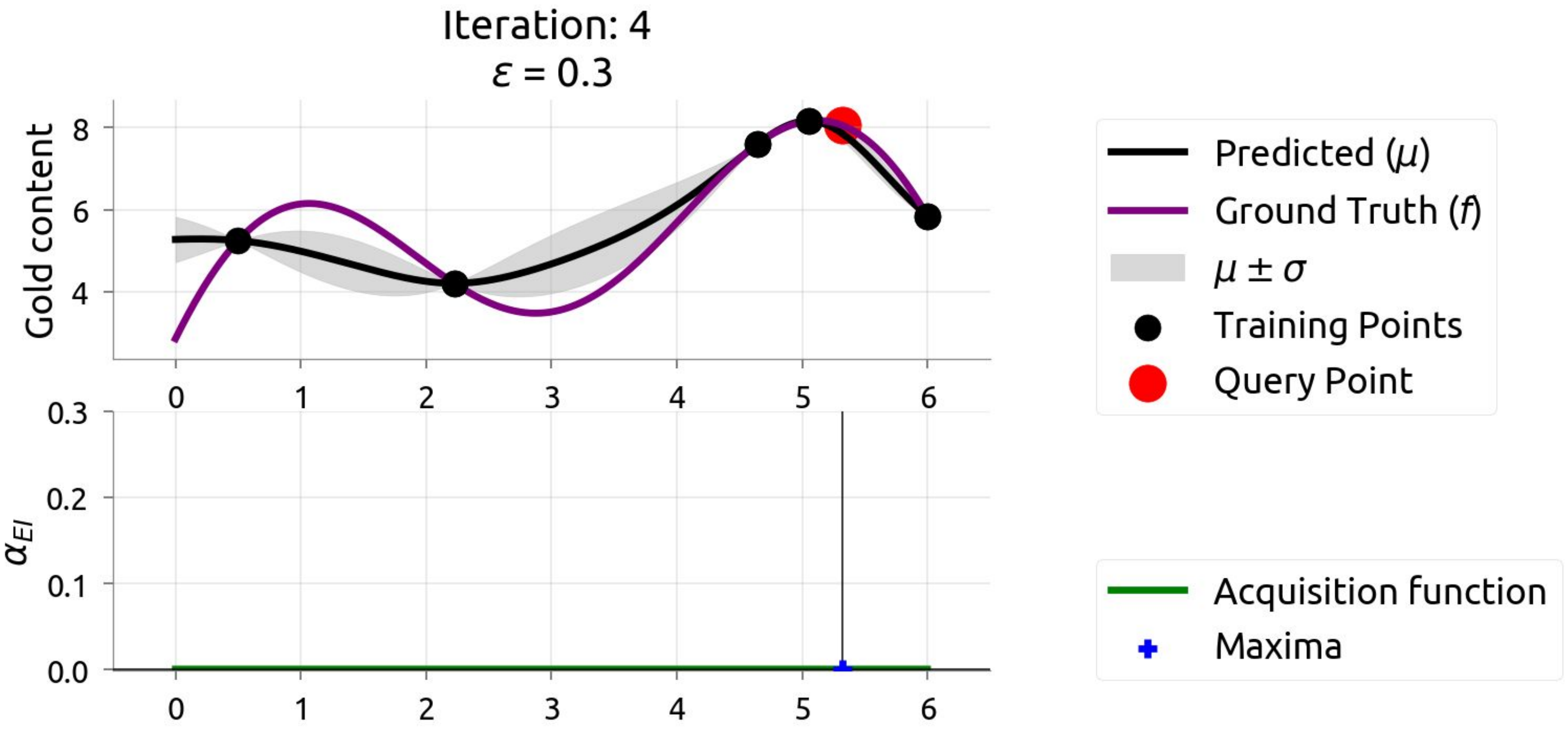
[6]

Bayesian Optimization



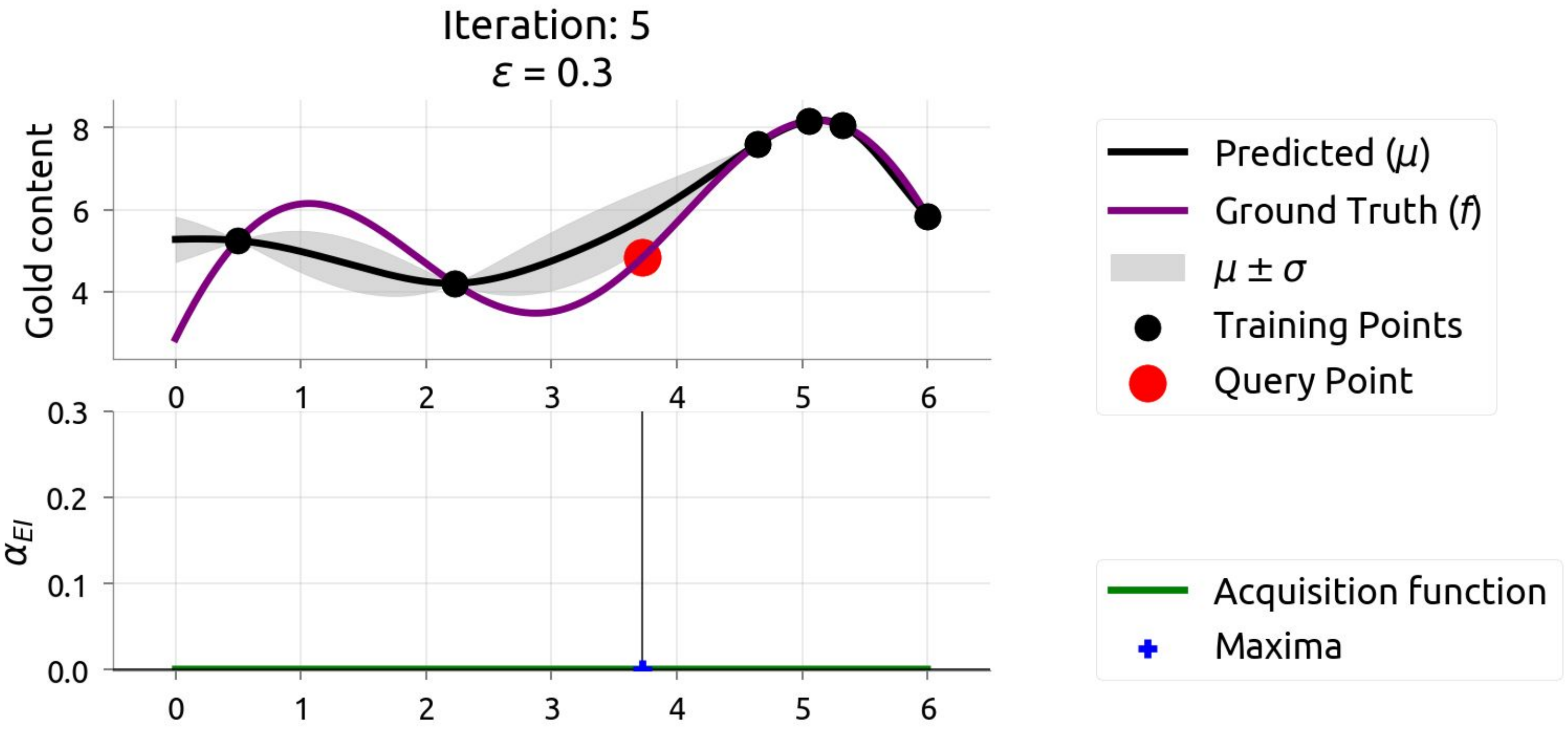
[6]

Bayesian Optimization



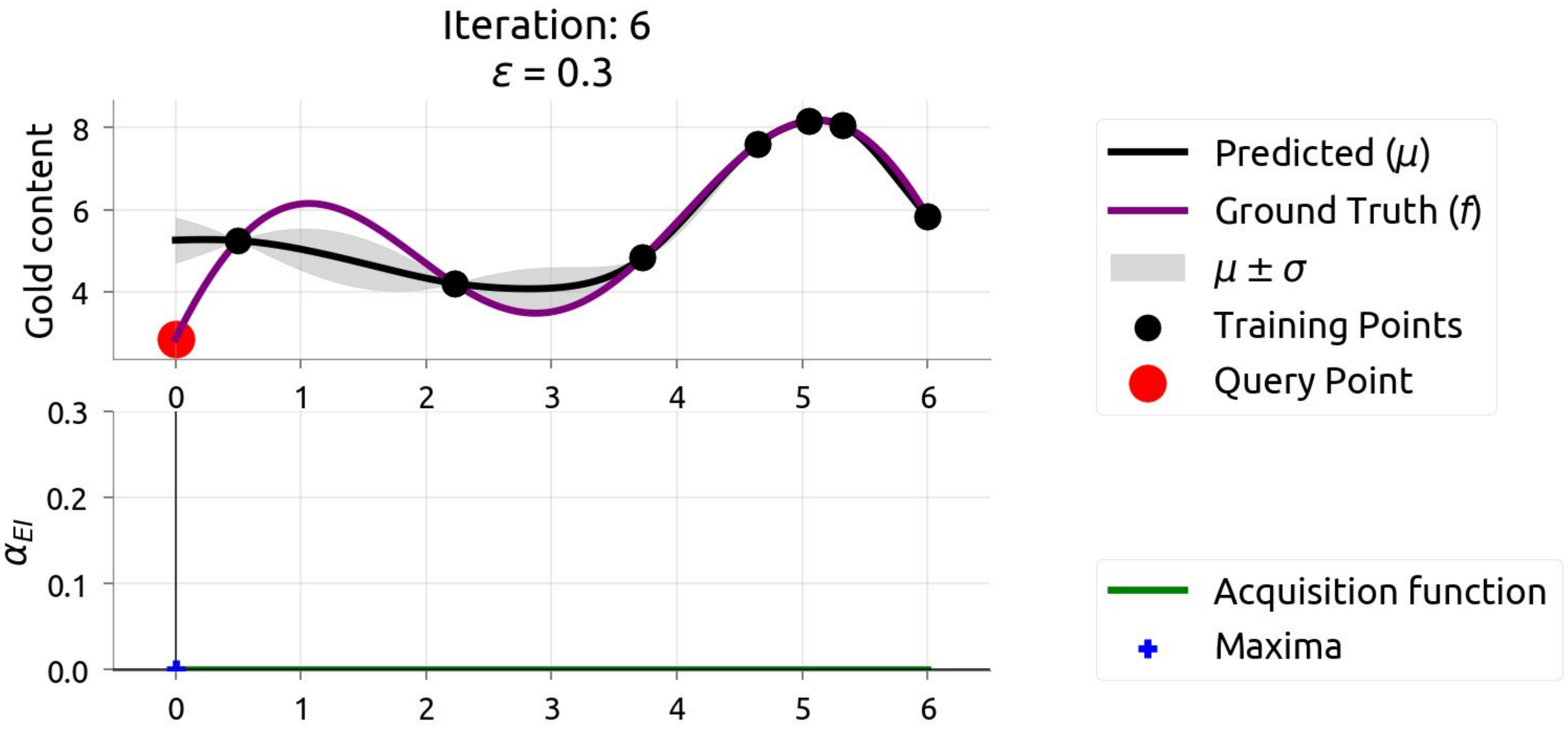
[6]

Bayesian Optimization



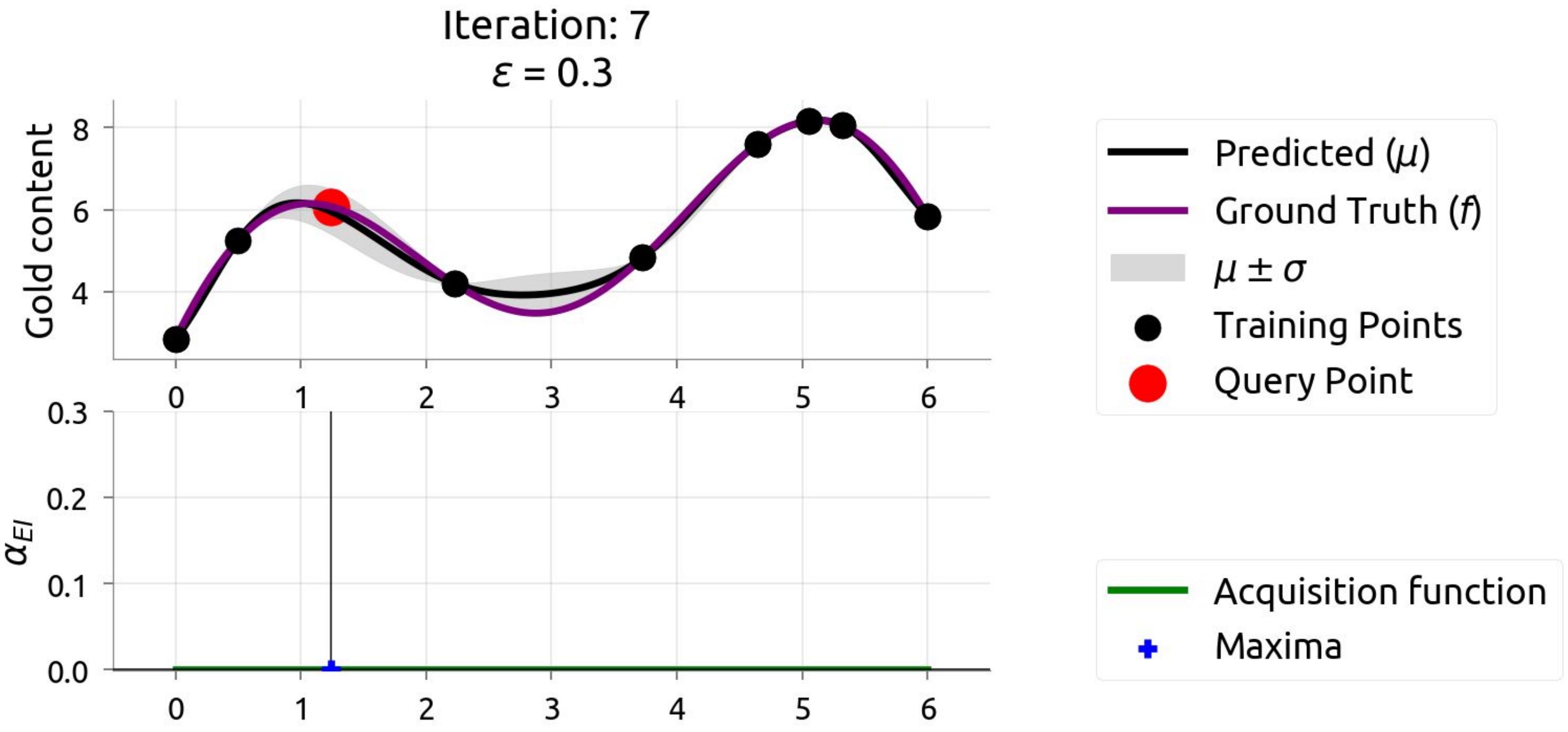
[6]

Bayesian Optimization



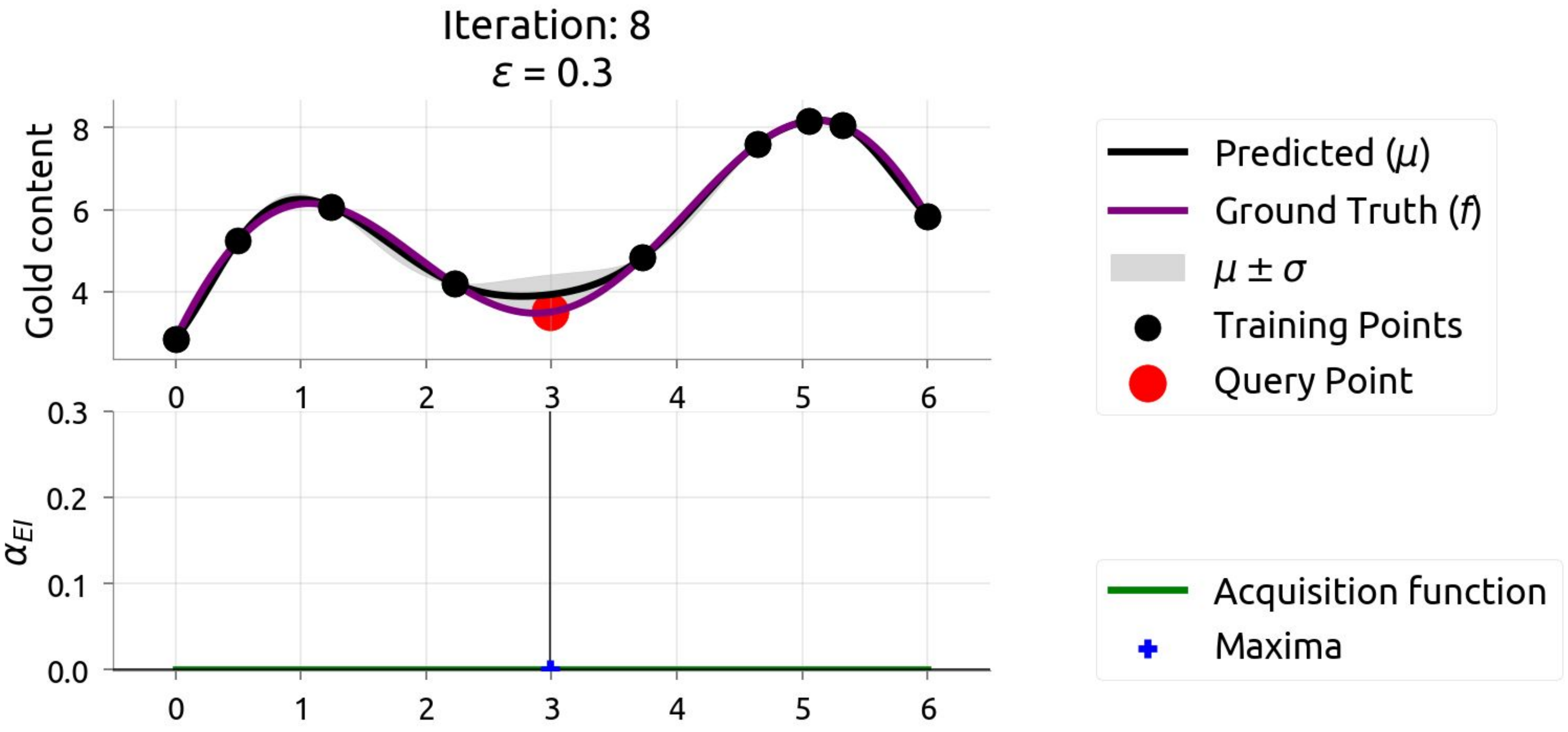
[6]

Bayesian Optimization



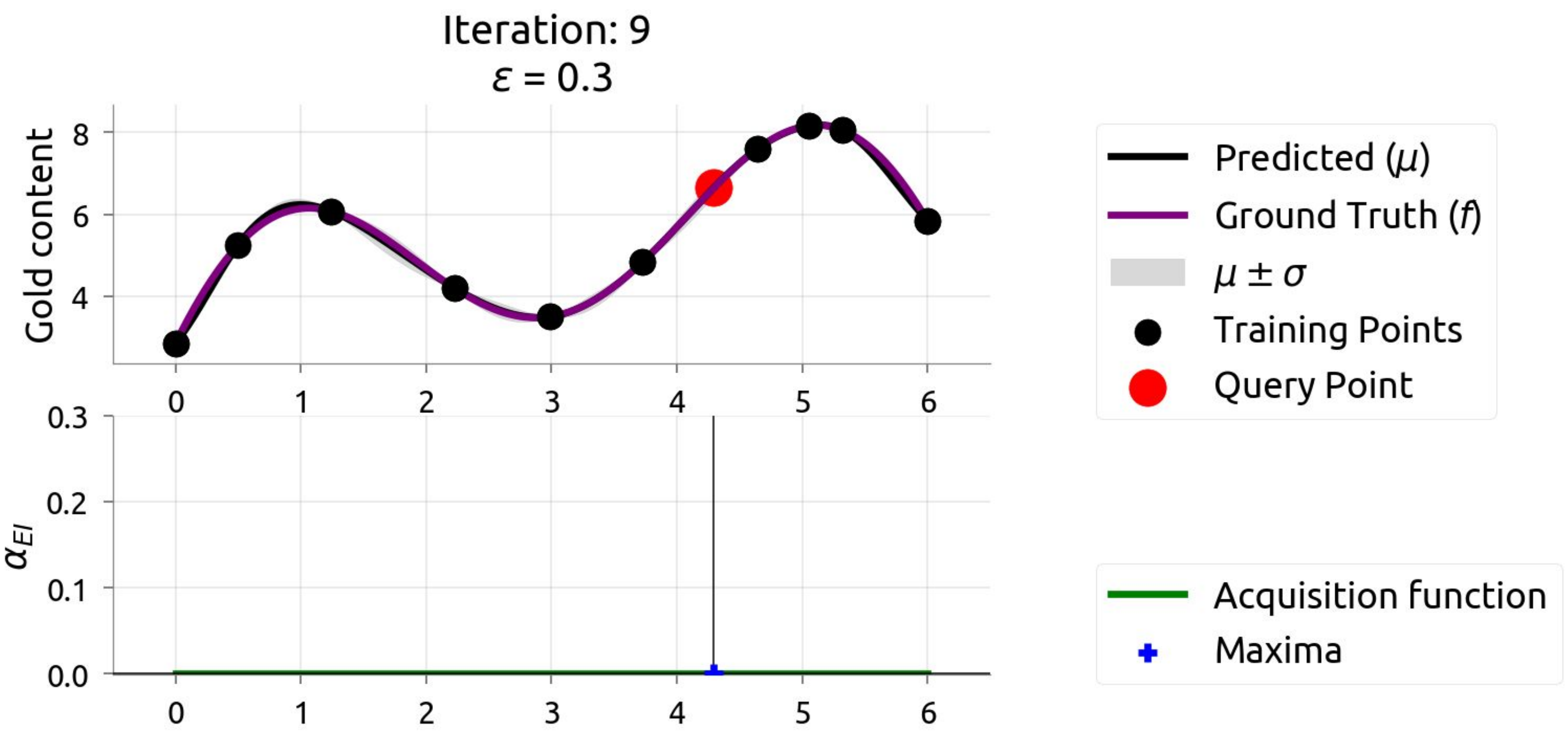
[6]

Bayesian Optimization



[6]

Bayesian Optimization



[6]

Bayesian Optimization

- Converges to the best configurations
- However, does not scale well due to Gaussian processes
  - Cubic relative to configurations tried so far
- Tree Parzen Estimator
  - Also BO method, but uses a kernel density estimator (KDE)
    - Constructs two distributions for “good” and “bad” points
    - Maximize ratio of these two distributions to choose next configuration
  - Equivalent to maximizing expected improvement
  - Linear relative to configurations tried so far

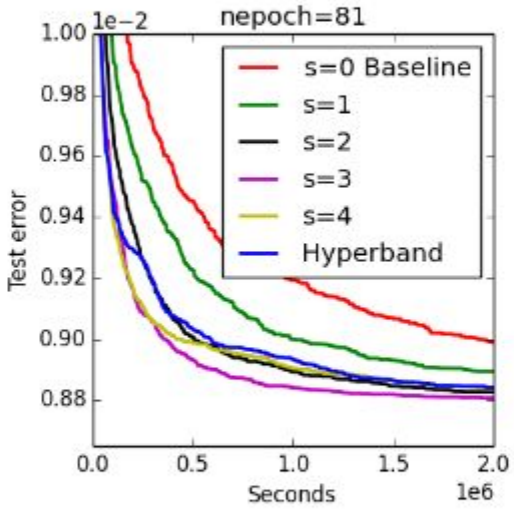
Hyperband

- **High-level idea:** compare relative performance of configurations and prematurely stop the bad performers and continue the good performers
- Allocate a *budget* to each configuration
  - e.g. number of epochs or data points used for training
- Repeatedly call Successive Halving (SH) on different budgets
  - Evaluate n randomly sampled configurations and keep the top 1/x
  - Increase the budget per configuration by a factor of x (allocate more resources)
  - Repeat until the maximum per-configuration budget is reached

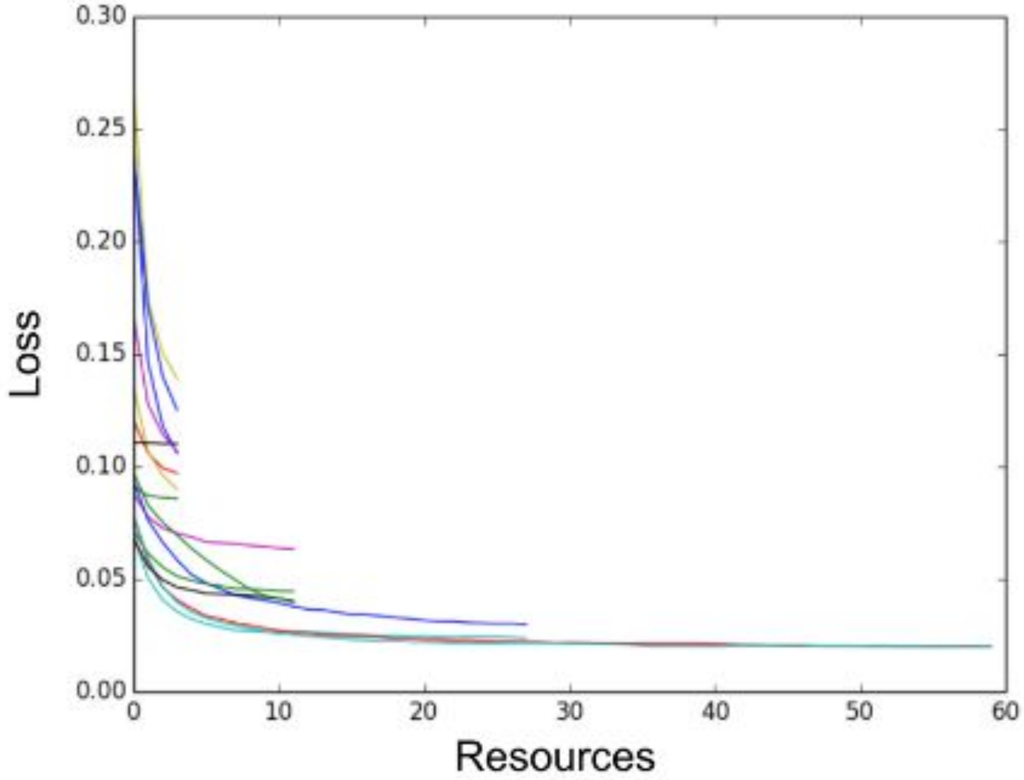
Hyperband

Example of early stopping (SH)

- Hyperband calls SH with different budgets (different starting number of configurations)
- Returns the configuration with the smallest objective function overall



[2]



[4]

Hyperband

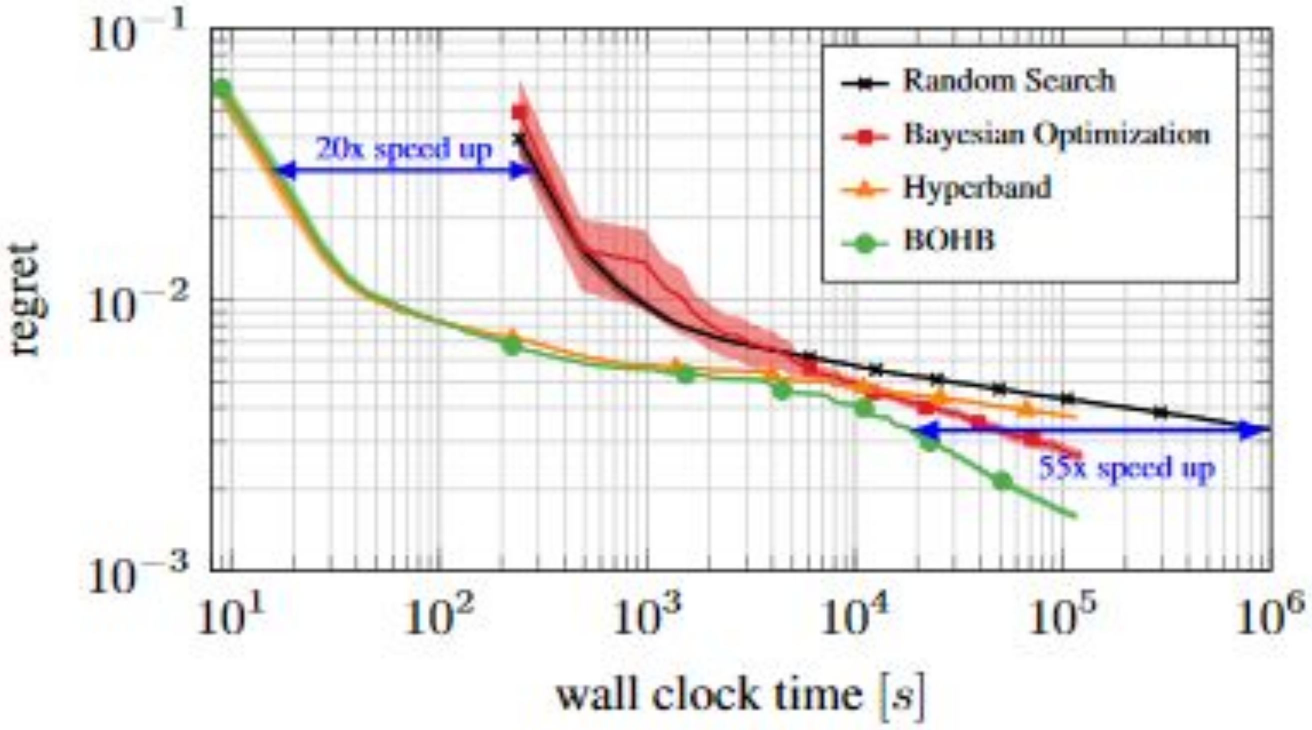
- Pros
  - Has theoretical guarantees
  - Parallelizable
  - Simpler than model-based methods
- Cons
  - Convergence
    - Randomly samples a set of configurations
    - Can eliminate late learners early
    - Requires many evaluations
  - Stragglers

BOHB

- Combines BO (specifically Tree Parzen Estimator) and HB
- Replaces random sampling in HB with BO component
  - Keep track of all configuration and budget pairs
  - Build up TPE model for each budget
    - Requires sufficient amount of evaluations
    - Use TPE model for largest budget with enough data (large budget => better validation)
  - Use this to better select configurations during HB

BOHB

- Initial progress from HB
- Strong final performance from BO



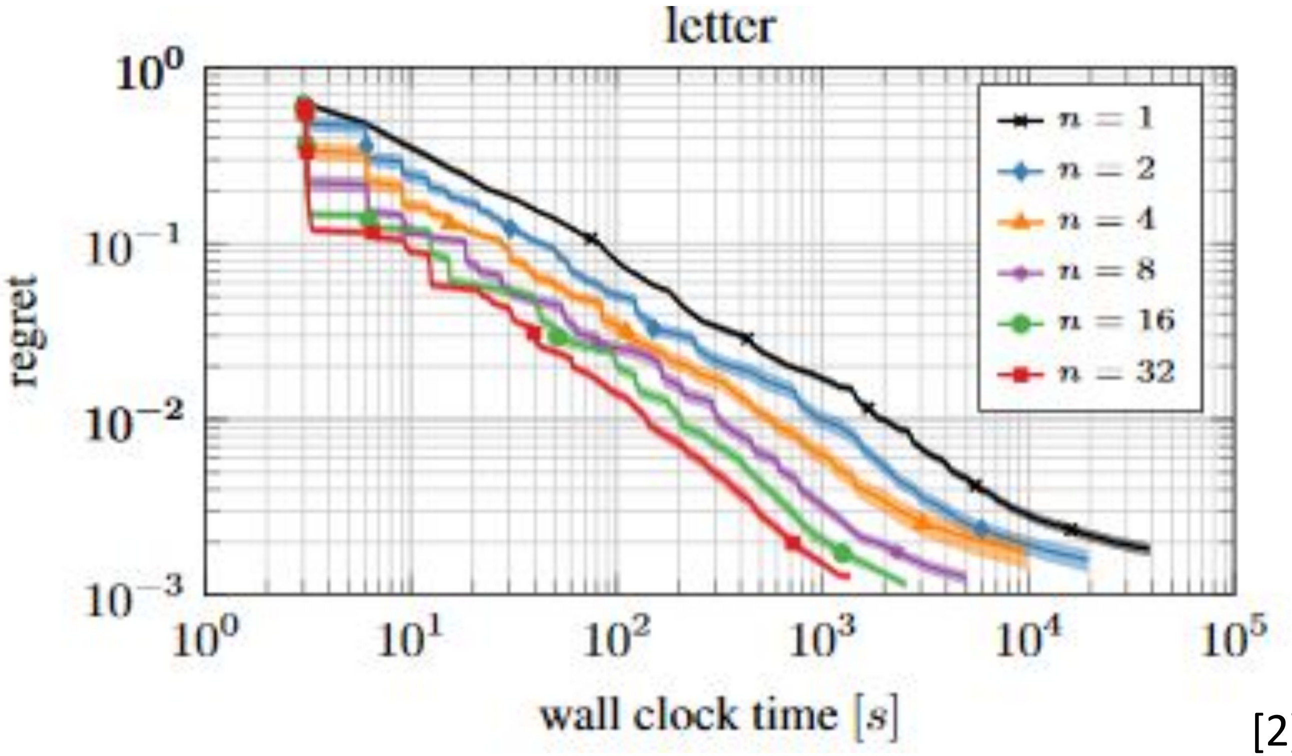
[2]

BOHB

- Parallelize aspects of HB (and TPE)
  - Parallel budget iterations
  - Evaluate many configurations at once
- Master node
  - Keeps track of model to make configuration selections
- Worker nodes
  - Evaluation configurations based on budget

Evaluation

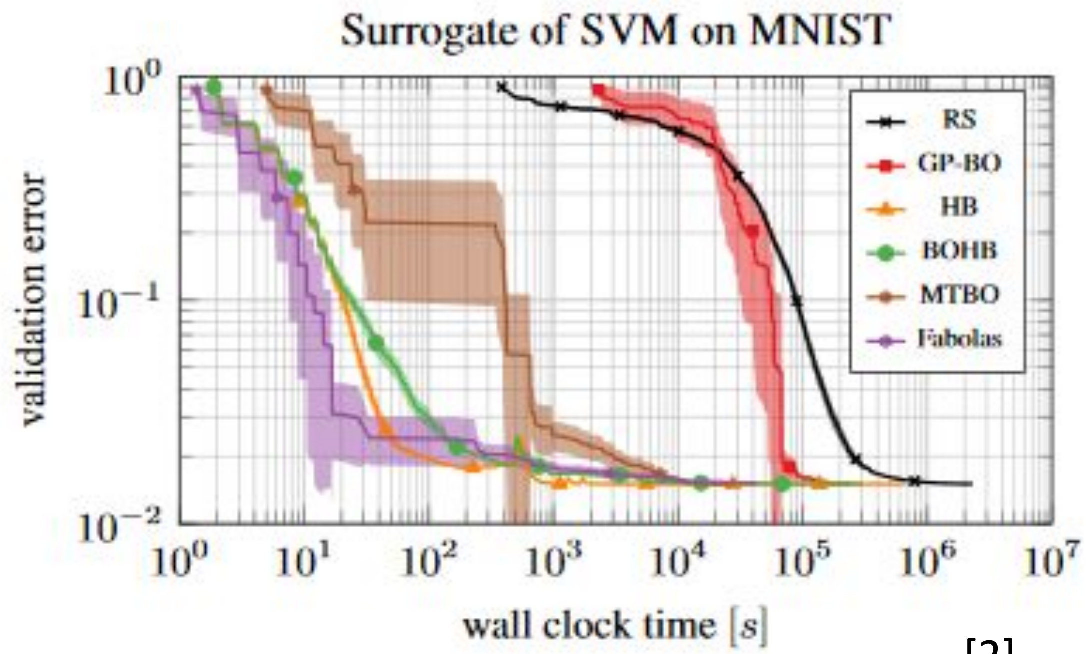
- Effect of parallelization
    - n = number of workers



[2]

Evaluation

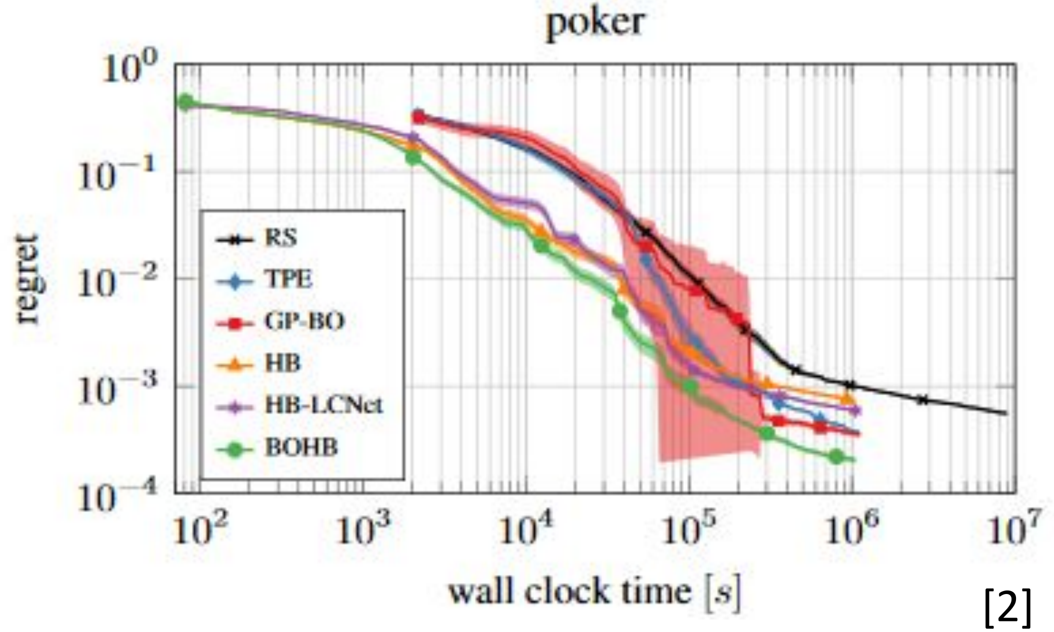
- SVM on MNIST
  - Regularization and kernel hyperparameters
  - Budget is number of training points



[2]

Evaluation

- Feed-Forward NNs on OpenML Datasets
  - Hyperparameters: initial learning rate, batch size, number of layers, etc.
  - 6 datasets, shown here is only one (Poker)



[2]

Questions

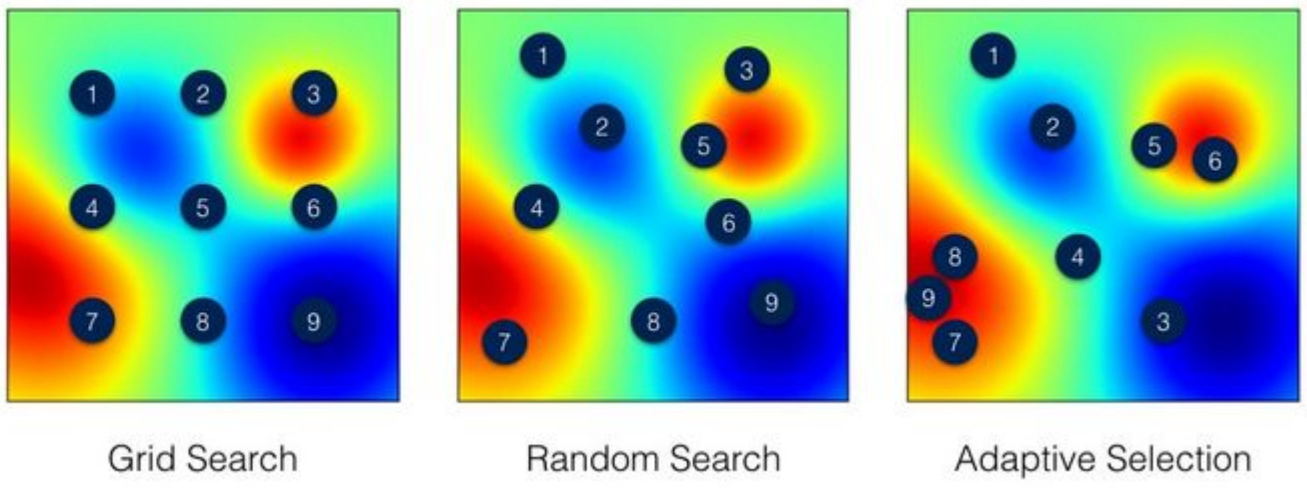
Implementation of BOHB:
https://github.com/automl/HpBandSter

A System for Massively Parallel Hyperparameter Tuning

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, Ameet Talwalkar

Motivation

[5]

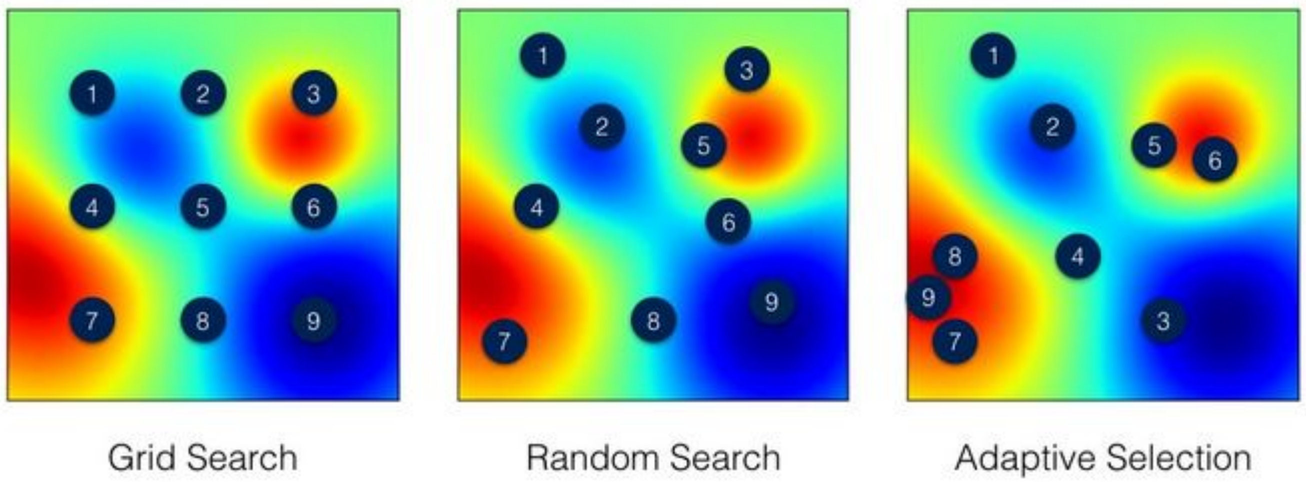


Motivation

- High-dimensional search spaces
- Increasing training times
  - Sequential methods are not necessarily feasible
  - Select configuration in the wall clock time it takes to train the model
- Rise of parallel computing
  - Exploit this to find good configurations in reasonable times
- Productionization of ML
  - Distributed HPO (efficient use of resources)

Motivation

[5]



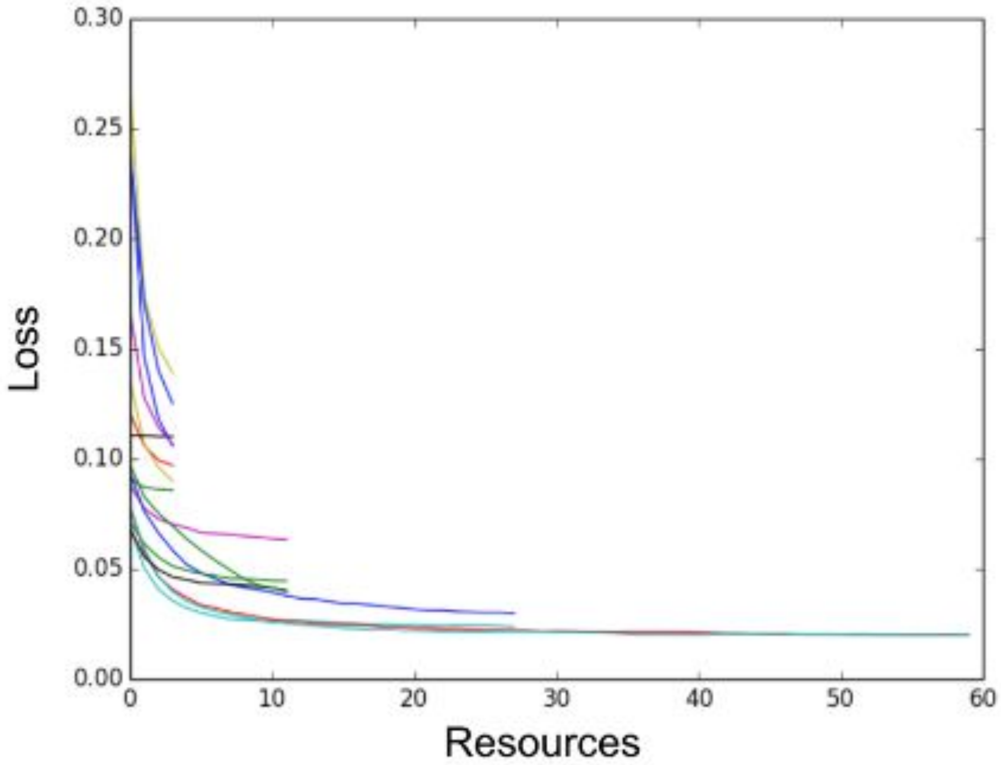
Easy to parallelize, but does not scale well with hyperparameters

Hard to parallelize, but scales well with hyperparameters

ASHA: High-level Idea

- Asynchronous Successive Halving Algorithm (ASHA)
  - Instead of starting with many configurations and narrowing down
  - Promote as soon as possible and keep adding new configurations
- Similar results to SHA, but more scalable
- Designed for
  - # configurations > # workers
  - Finishing within a small multiple of training time
- ASHA is "production-ready"

SHA



[4]

SHA

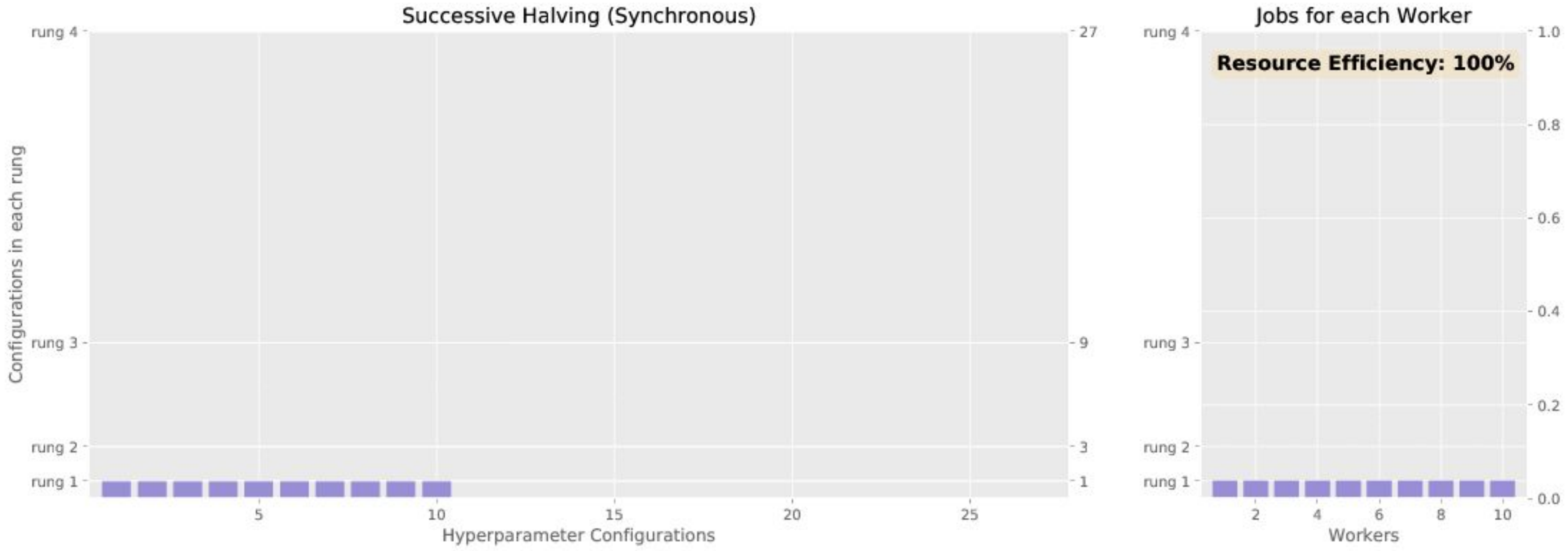
1/η = rate of elimination at each rung (halving stage)

	Configurations Remaining	Epochs per Configuration
Rung 1	27	1
Rung 2	9	3
Rung 3	3	9
Rung 4	1	27

Table 1: SHA with η=3 starting with 27 configurations, each allocated a resource of 1 epoch in the first rung.

[5]

SHA


Successive Halving (Synchronous)
rung 4
rung 3
rung 2
rung 1
Configurations in each rung
27
9
3
1
5
10
15
20
25
Hyperparameter Configurations
Jobs for each Worker
Resource Efficiency: 100%
1.0
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
Workers

[5]

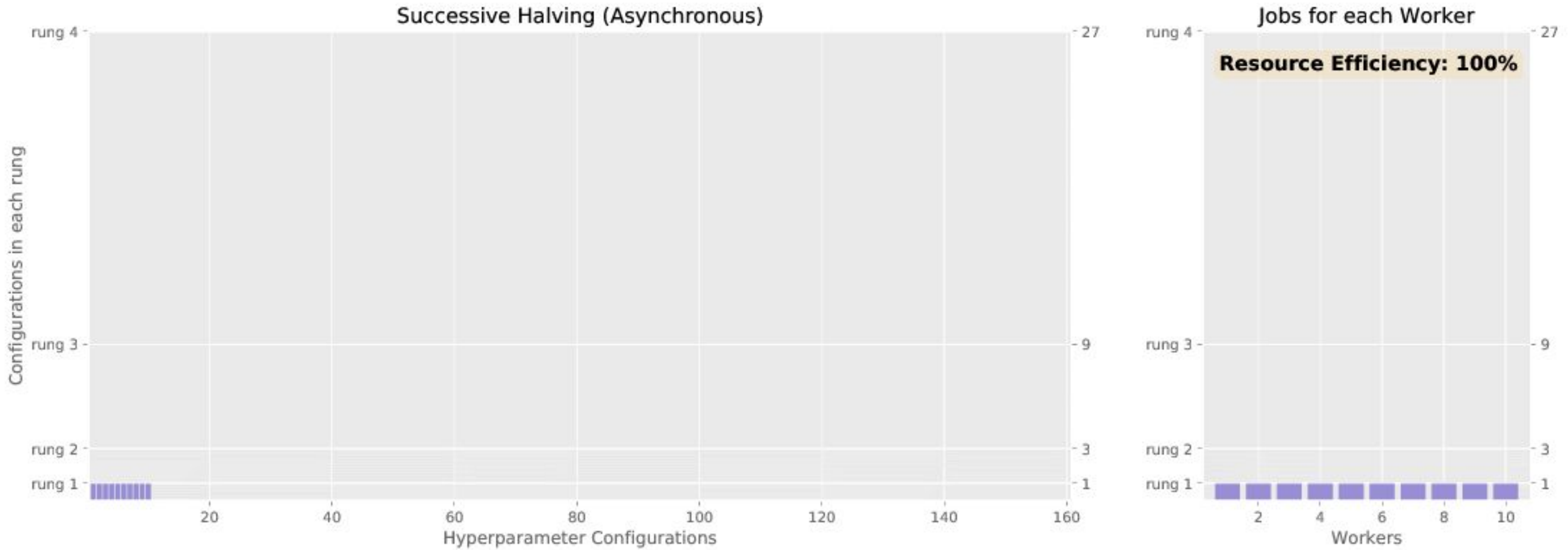
SHA

- Issues with SHA
  - Sequential
  - Wait for each configuration to finish in each rung before moving on (bottleneck)
  - Difficult to parallelize efficiently

ASHA

- Starts out by adding configurations to lowest rung
  - Based on worker availability
- Once a worker finishes, ASHA looks for promotions in all rungs
  - If not, add a configuration to lowest rung
  - This will eventually lead to more promotions

ASHA



Successive Halving (Asynchronous)
Jobs for each Worker
Resource Efficiency: 100%
rung 4
rung 3
rung 2
rung 1
27
9
3
1
Configurations in each rung
20
40
60
80
100
120
140
160
Hyperparameter Configurations
2
4
6
8
10
Workers

[5]

ASHA

- Better utilizes workers compared to SHA
- Returns configuration in at most twice the training time

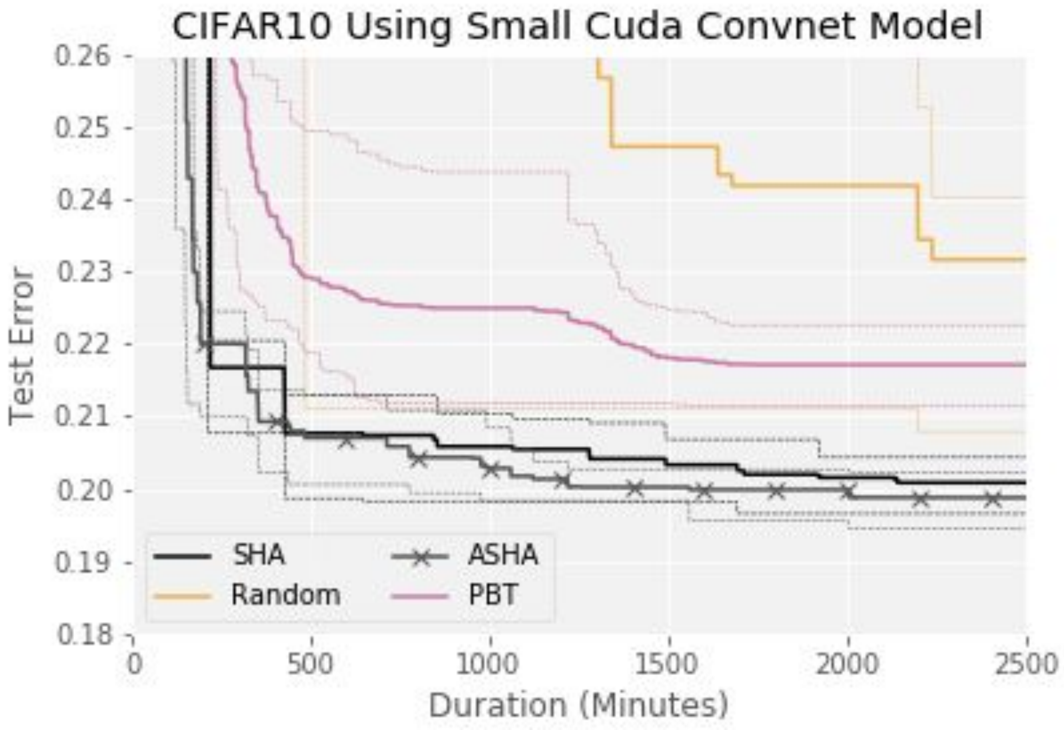
$$\left(\sum_{i=s}^{\log_\eta(R)} \eta^{i-\log_\eta(R)}\right) \times time(R) \leq 2\, time(R).$$

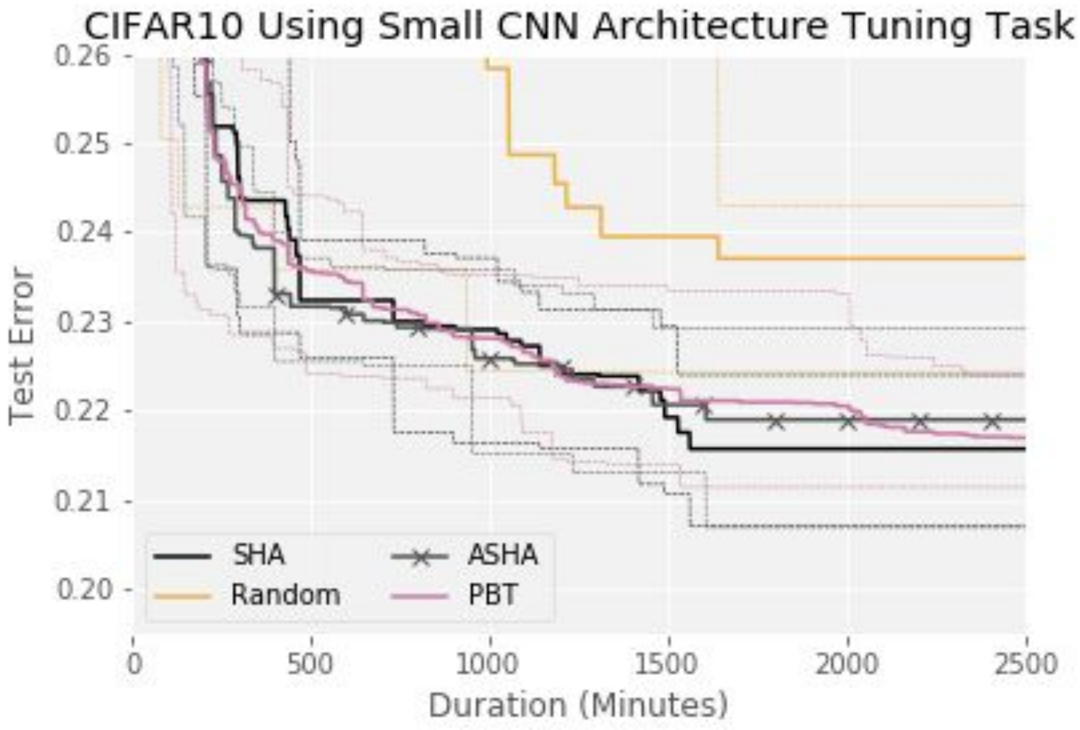
Related Systems

- Hyperband, BOHB
- Population-based training (PBT)
  - Evaluate population of hyperparameters
  - Remove worst performers
  - Mutate best performers and add them in place of removed ones
  - No theoretical guarantees
- Vizier
  - Google's black-box optimization service
  - Has various early-stopping configurations
- RayTune, CHOPT, Optuna

Evaluation

- Compare against SHA, PBT, and Random on a single machine
- Tune CIFAR-10 (CNN)

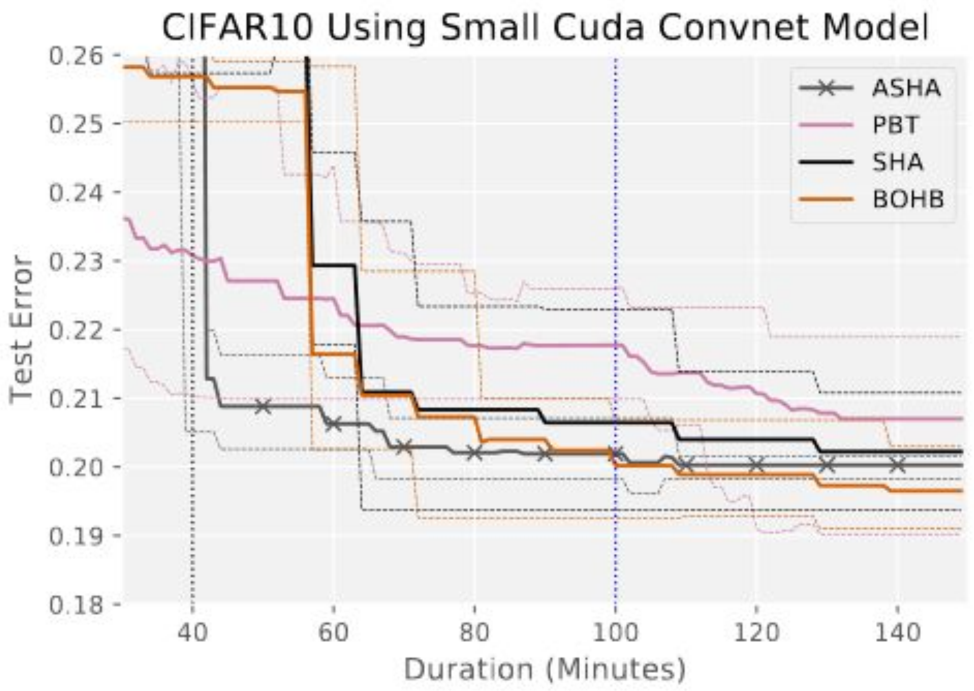


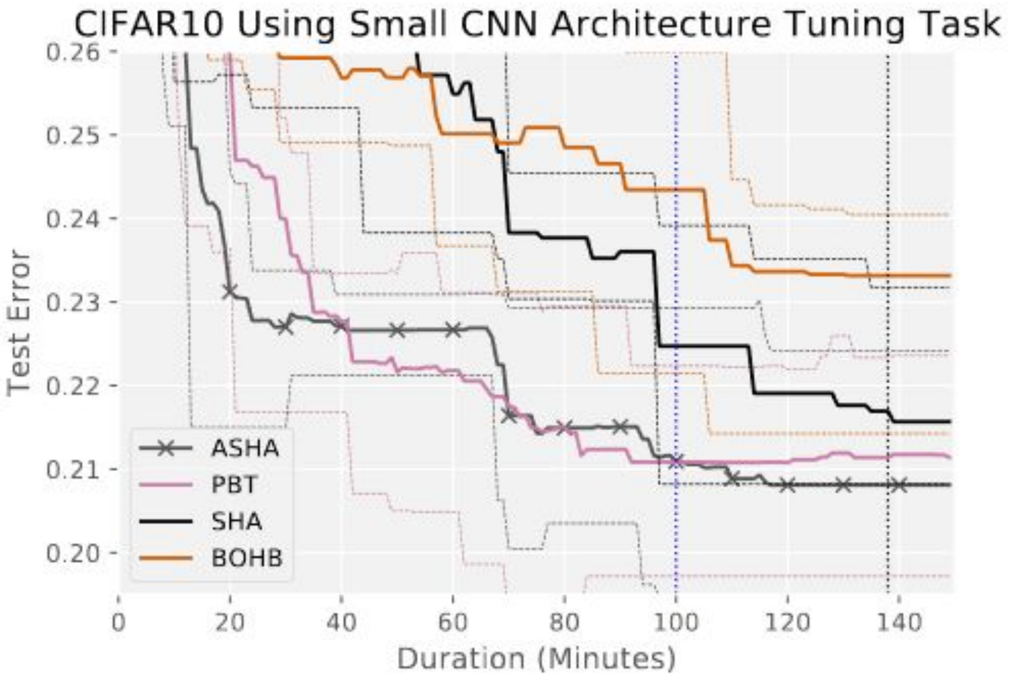


[5]

Evaluation

- Compare against SHA, BOHB, and PBT with 25 workers for 150 min (5 trials)
- Tune CIFAR-10 (CNN)

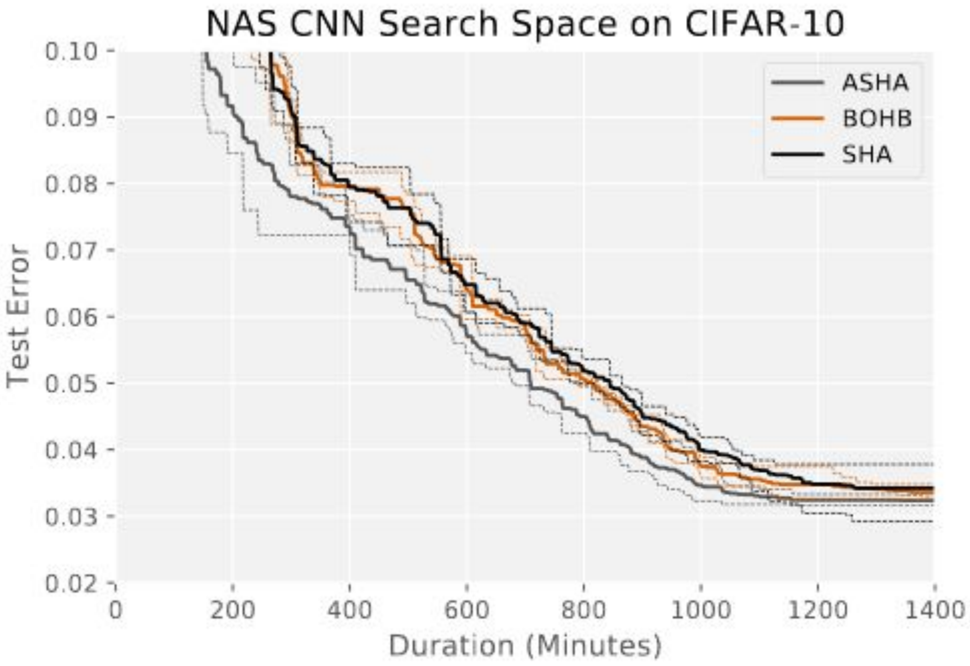


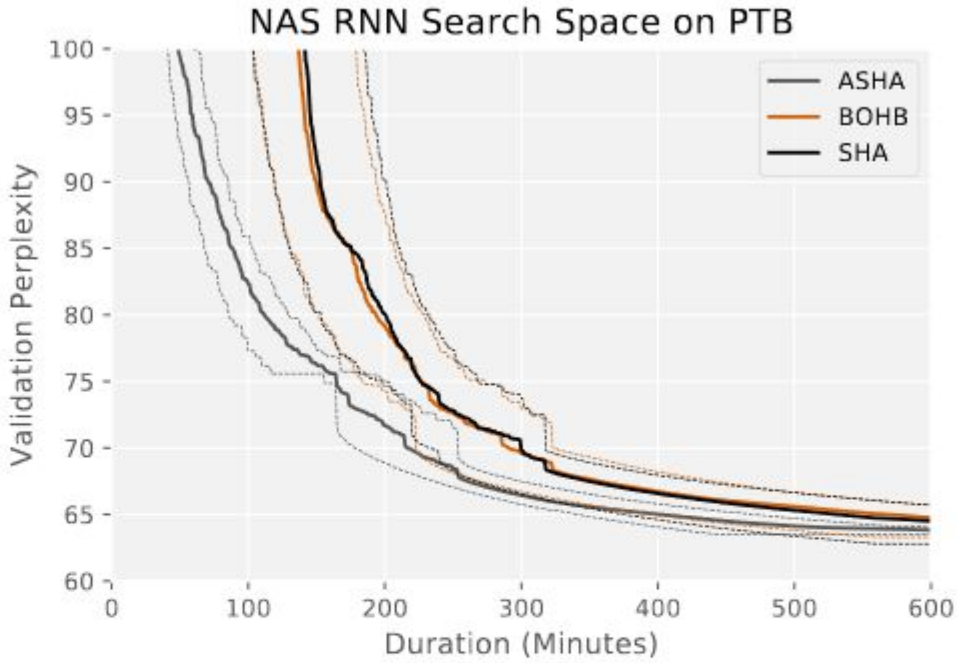


[3]

Evaluation

- Compare against SHA and BOHB on 16 workers
- Neural Architecture Search as a specialized HPO
    - Design CNNs (CIFAR-10)
    - Design RNNs (Penn Treebank)

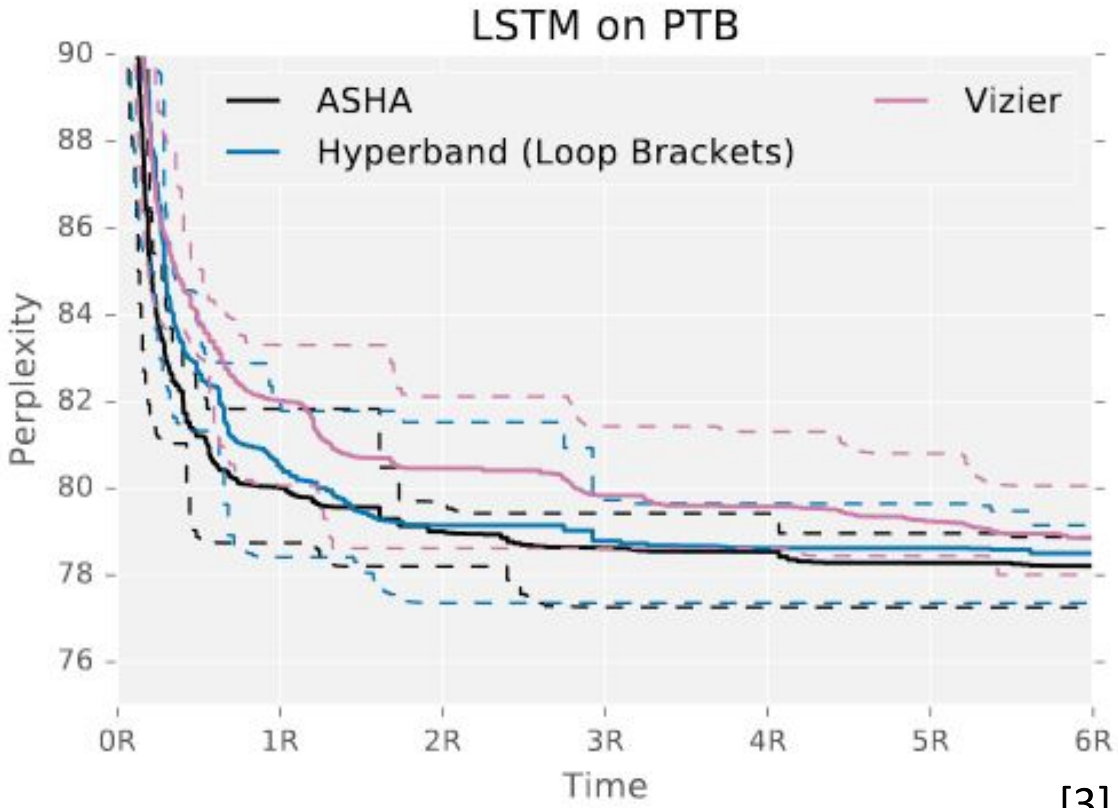




[3]

Evaluation

- Compare against (parallel) Hyperband and Vizier with 500 workers
- LSTM (requires weeks to run)



[3]

Productionization

- Usability
  - Do not expose user to internal configurations of ASHA
  - Set default values (elimination rate, max early-stopping rate, brackets to run)
  - Set the number of configurations evaluated as the explicit stopping time
- Automatic Scaling of Parallel Training
  - Speed up training in higher rungs with more GPUs
- Resource Allocation
  - Adaptively allocate resources for better algorithm and cluster utilization
- Reproducibility
  - Pausing and restarting evaluations (checkpointing)
  - Track promotions for each bracket
  - Provides some fault-tolerance

Questions

References

1. James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 13, null (3/1/2012), 281–305.
2. *Falkner, Stefan et al. "BOHB: Robust and Efficient Hyperparameter Optimization at Scale." ICML (2018).*
3. Li, Liam, et al. "Massively parallel hyperparameter tuning." (2018).
4. Li, Lisha, et al. "Hyperband: A novel bandit-based approach to hyperparameter optimization." *The Journal of Machine Learning Research* 18.1 (2017): 6765-6816.
5. https://blog.ml.cmu.edu/2018/12/12/massively-parallel-hyperparameter-optimization/
6. https://distill.pub/2020/bayesian-optimization/?utm_campaign=Data_Elixir&utm_source=Data_Elixir_285