Syne Tune: Automated Hyperparameter and Neural Architecture Search at Scale.

Aaron Klein

October 7th, 2022

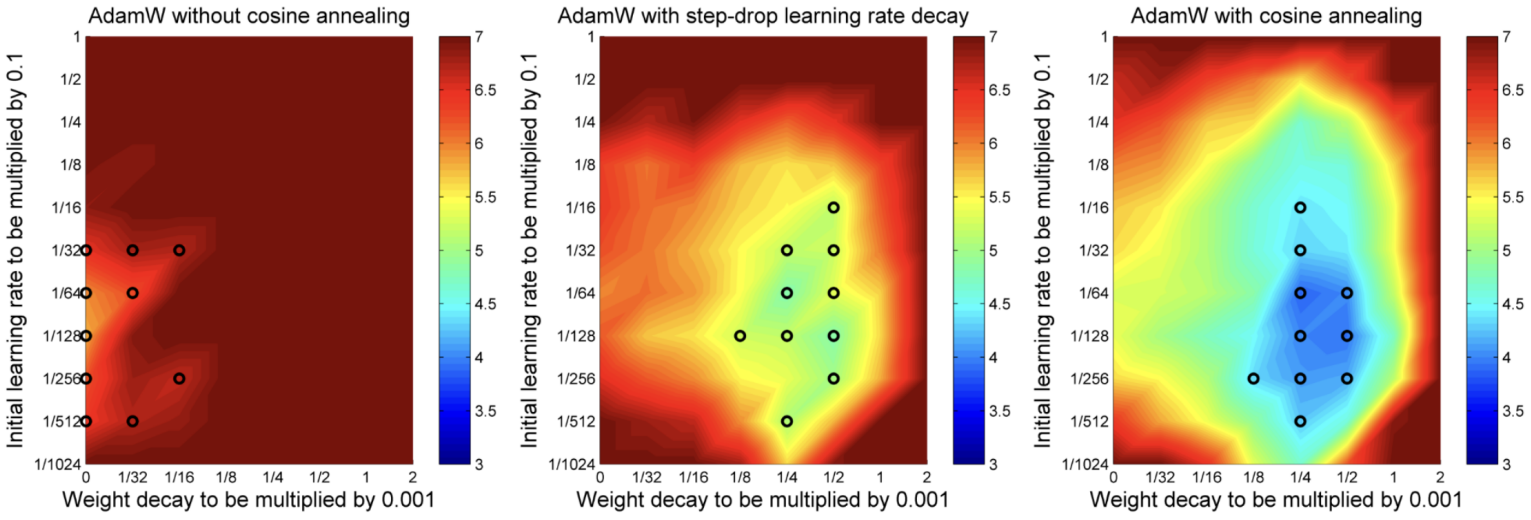
Syne Tune Team

- Aaron Klein, Applied Scientist
- Cedric Archambeau, Principal Applied Scientist
- David Salinas, Senior Applied Scientist
- Martin Wistuba, Applied Scientist
- Matthias Seeger, Principal Applied Scientist

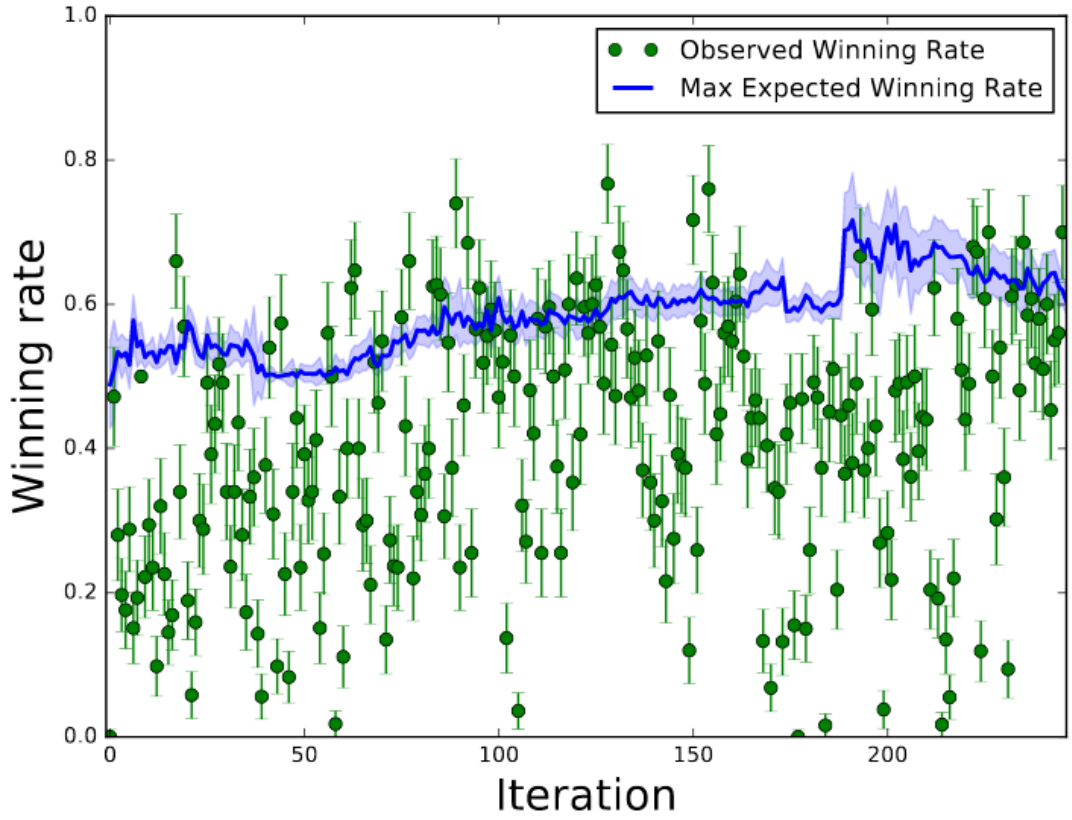
Outline

- Introduction hyperparameter and neural architecture search
- Overview Syne Tune
- Advanced hyperparameter and neural architecture search
- Use-cases

Hyperparameters of AdamW [LH19]



Tuning the hyperparameters of Monte Carlo tree search in AlphaGo [CHW$^+$18]



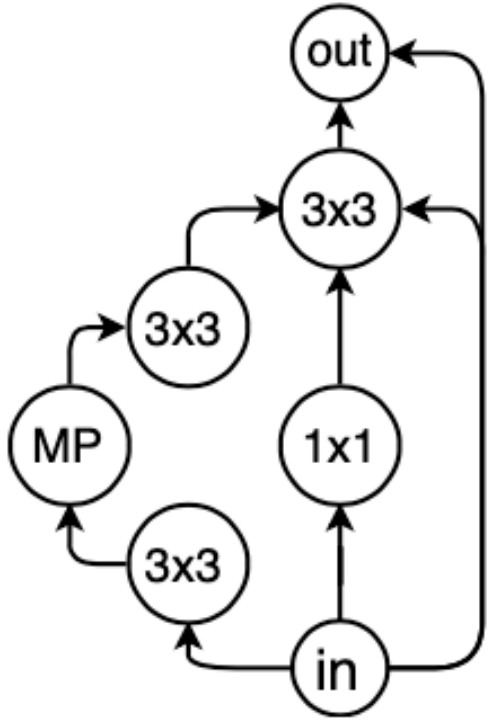
Hyperparameter Optimization

$$\boldsymbol{x}_\star \in \arg\min_{\boldsymbol{x} \in \mathbb{X}} f(\boldsymbol{x})$$

where

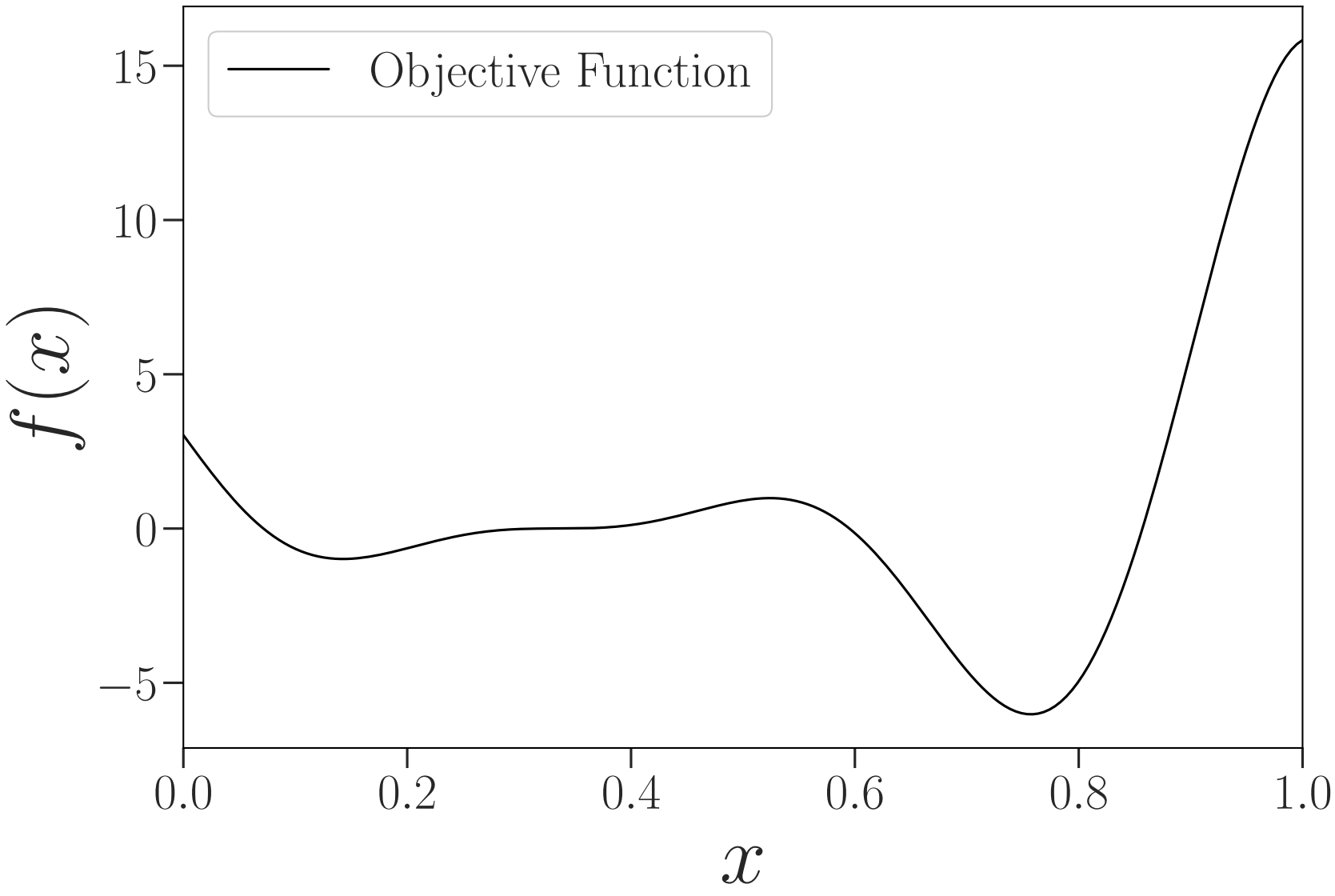
- $\mathbb{X}$ is called the configuration space
- $\boldsymbol{x} \in \mathbb{X}$ encodes all hyperparameters stacked to a vector
- $f : \mathbb{X} \to \mathbb{R}$ is the validation error after training with hyperparameters $\boldsymbol{x}$

Neural Architecture Search

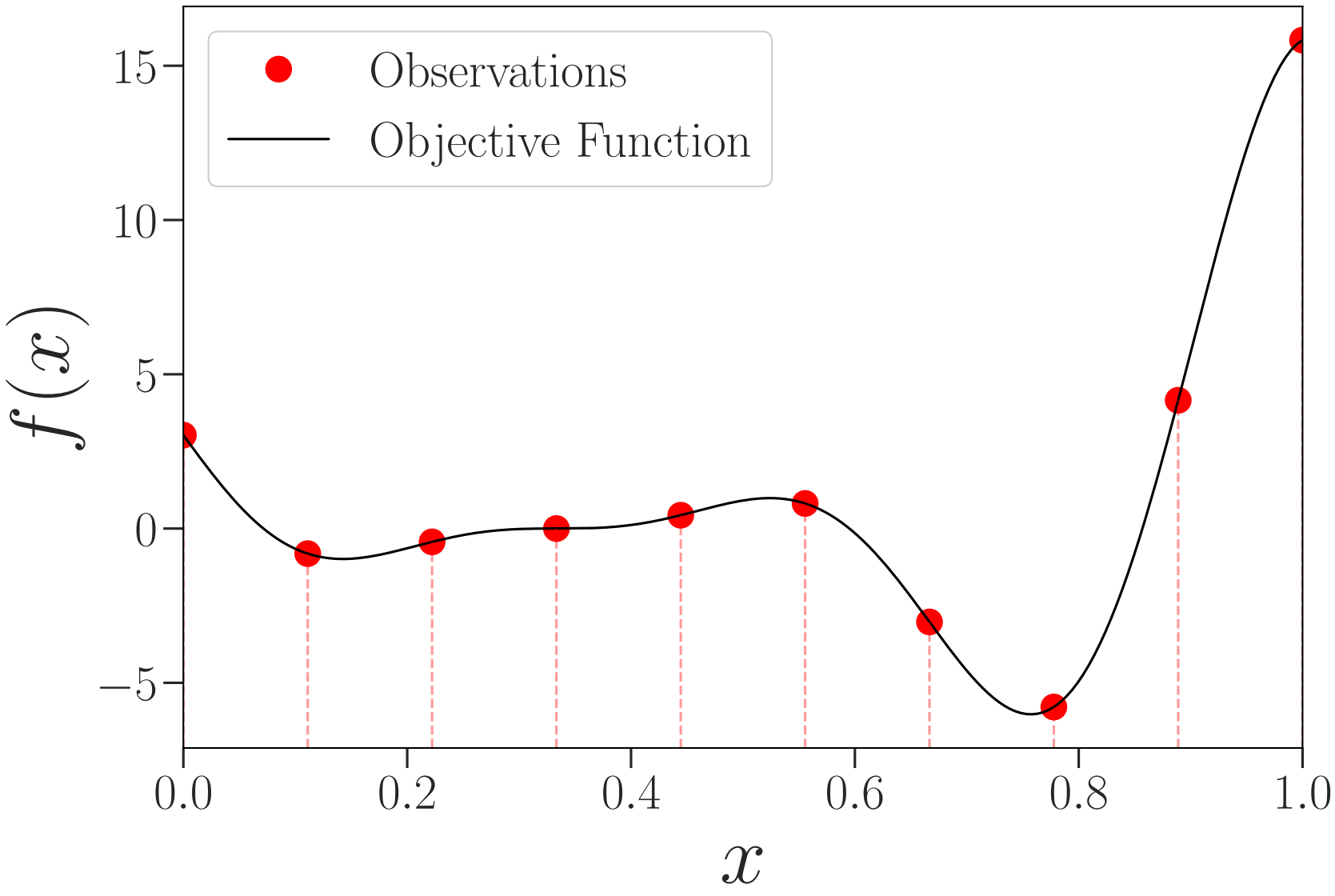


(Image credit: [YKR+19])

Objective



Grid Search


Observations
Objective Function
15
10
5
0
−5
$f(x)$
0.0
0.2
0.4
0.6
0.8
1.0
x

Grid Search

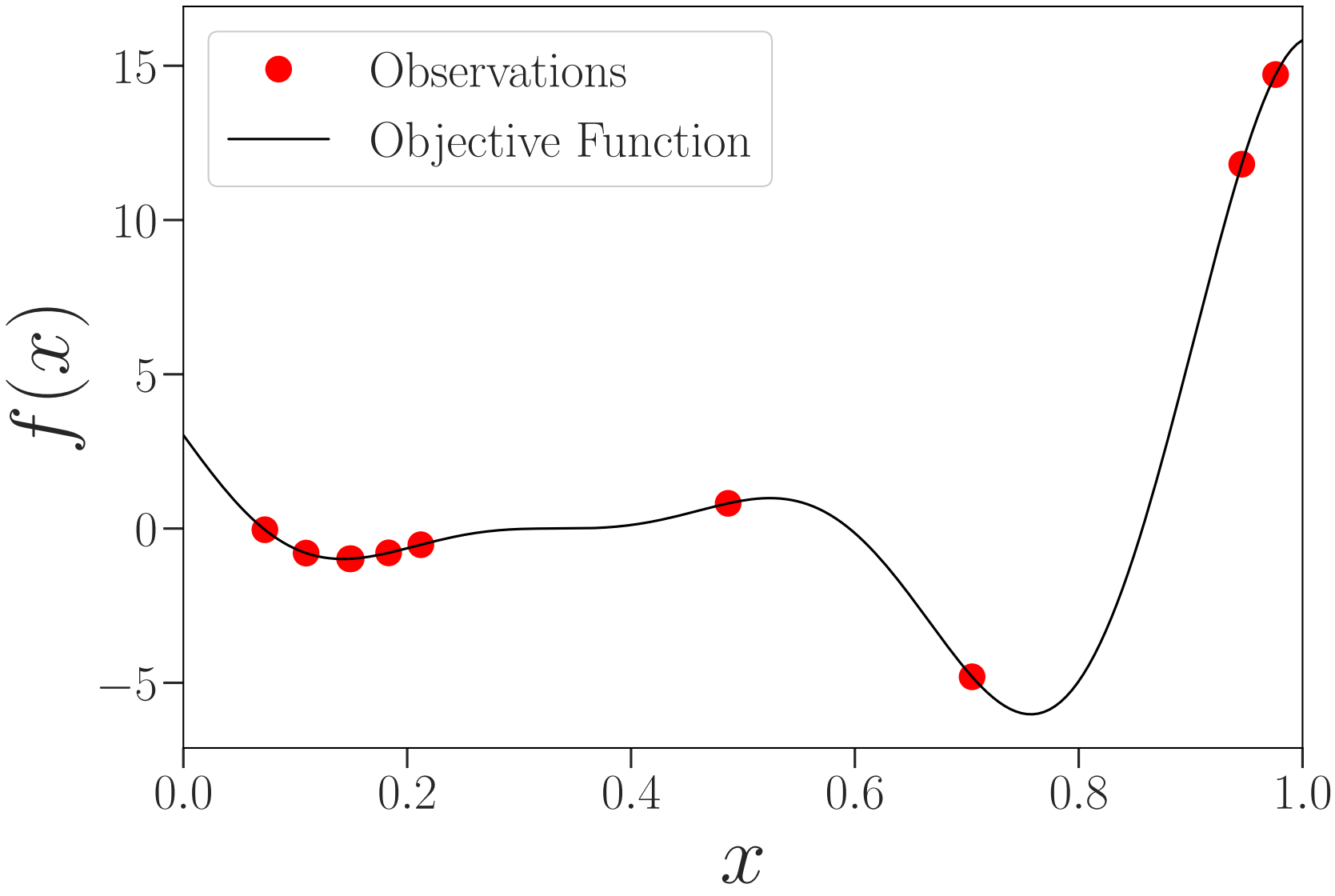
Pros:

- easy to implement
- easy to parallelize
- works for discrete and continuous configuration spaces

Cons:

- unlikely to find the optimum
- does not learn from past observations
- grid size grows exponentially with the number of hyperparameters

Random Search [BB12]



Random Search

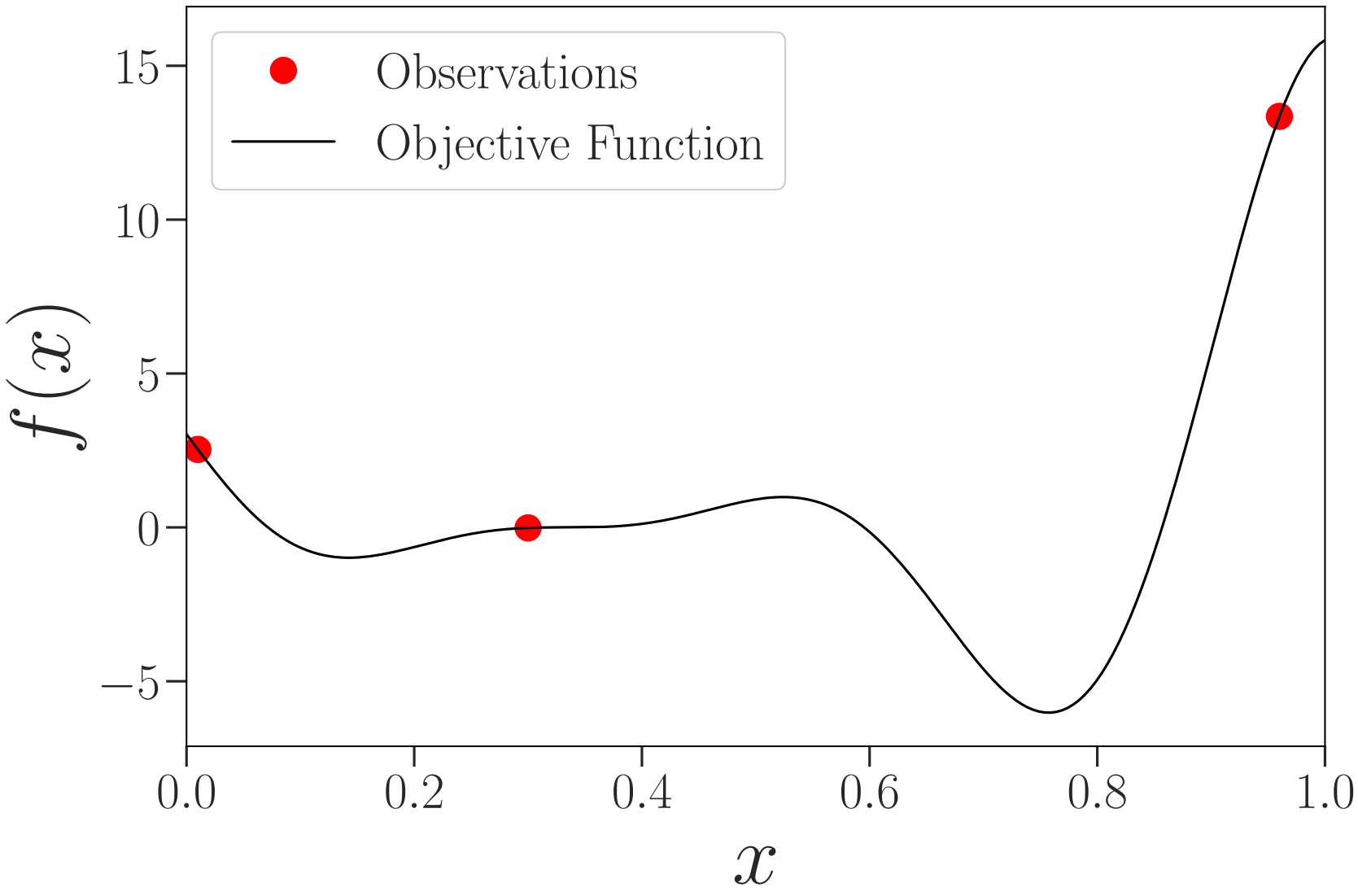
Pros:

- easy to implement
- easy to parallelize
- works for any configuration space
- will always converge to the optimum in the limit

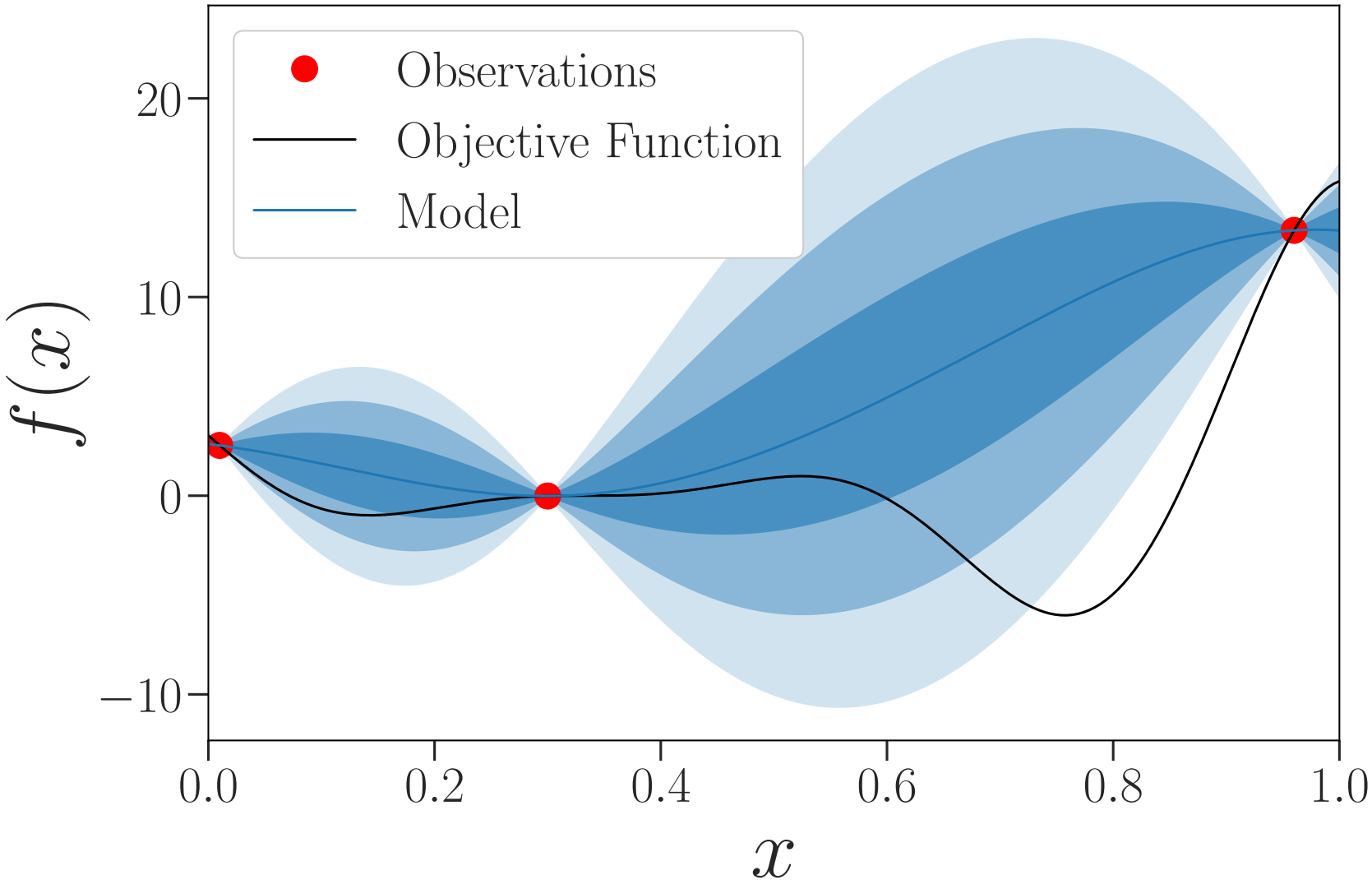
Cons:

- does not learn from past observations

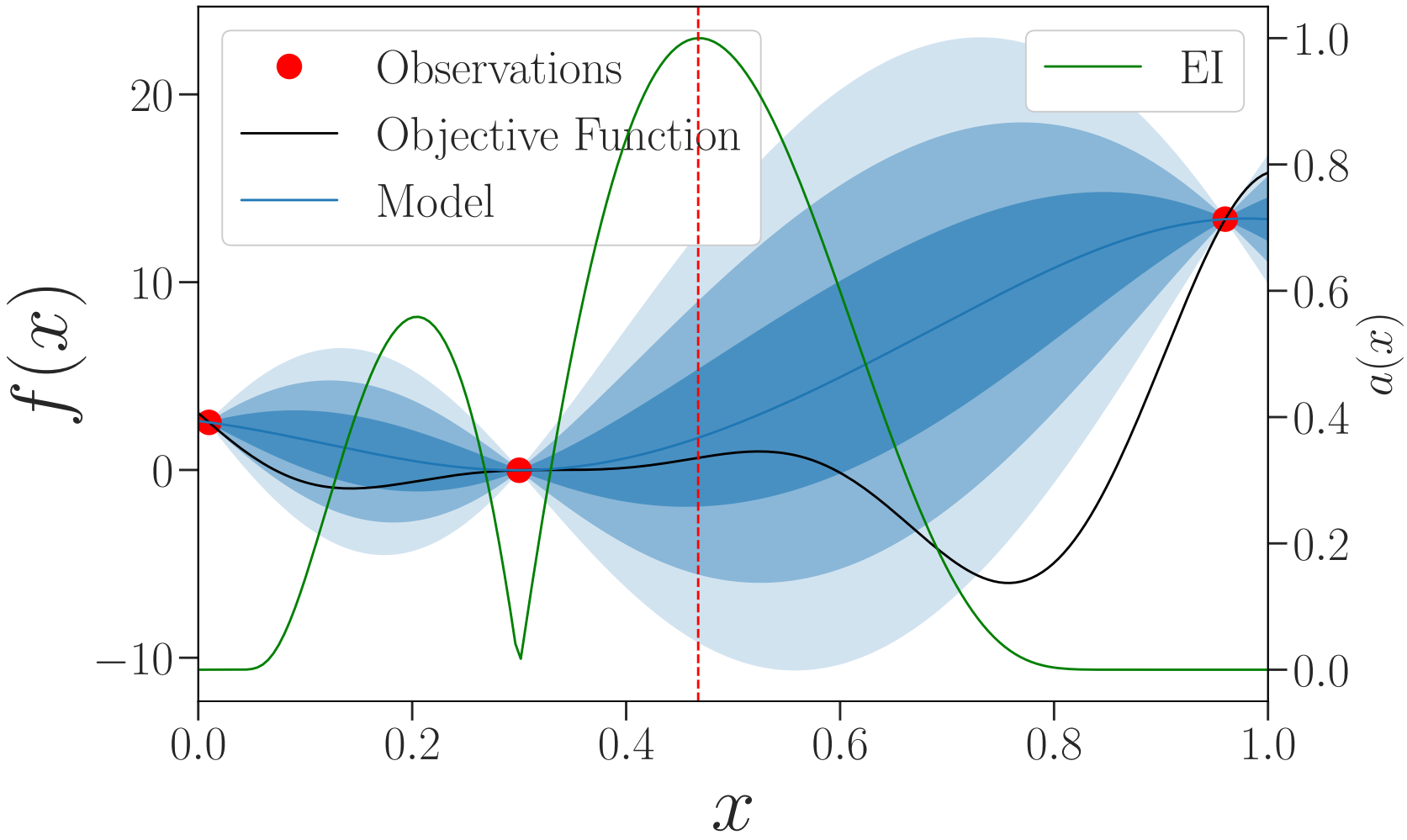
Bayesian Optimization



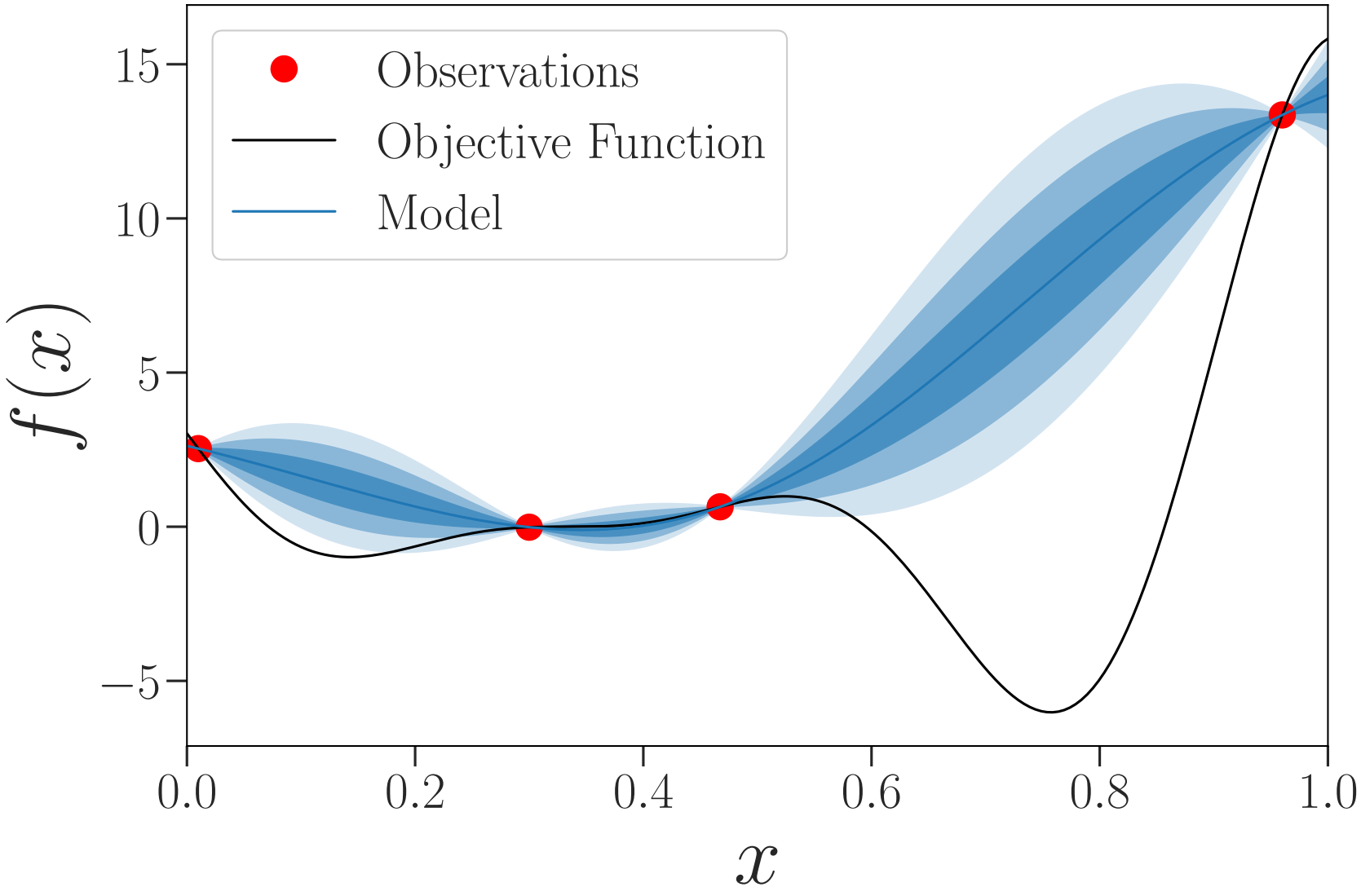
Bayesian Optimization



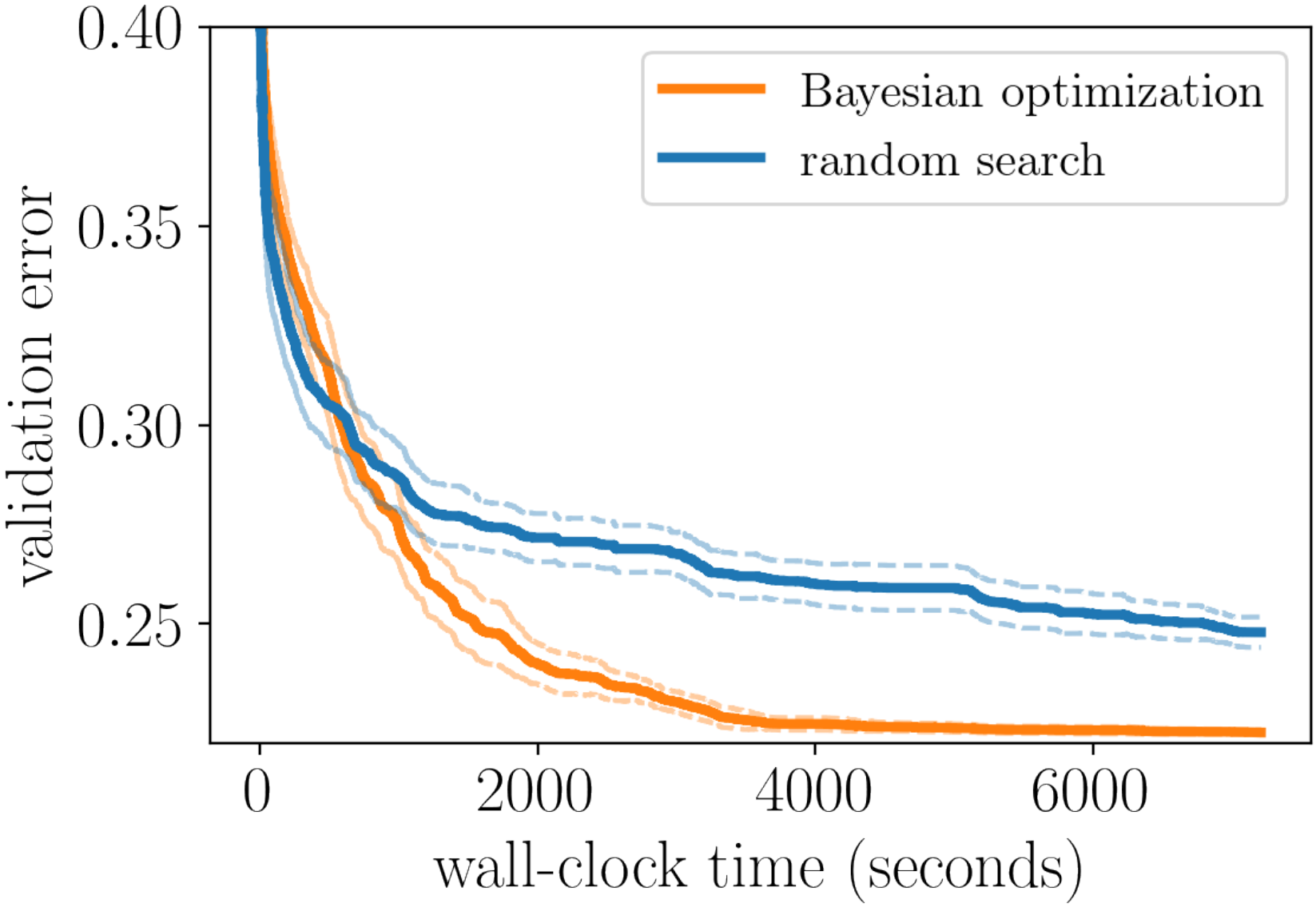
Bayesian Optimization



Bayesian Optimization



Comparison


0.40
0.35
0.30
0.25
validation error
0
2000
4000
6000
wall-clock time (seconds)
Bayesian optimization
random search

Bayesian Optimization

Pros:

- more sample efficient
- converges to the optimum under some assumptions

Cons:

- harder to parallelize

Syne Tune

Syne Tune

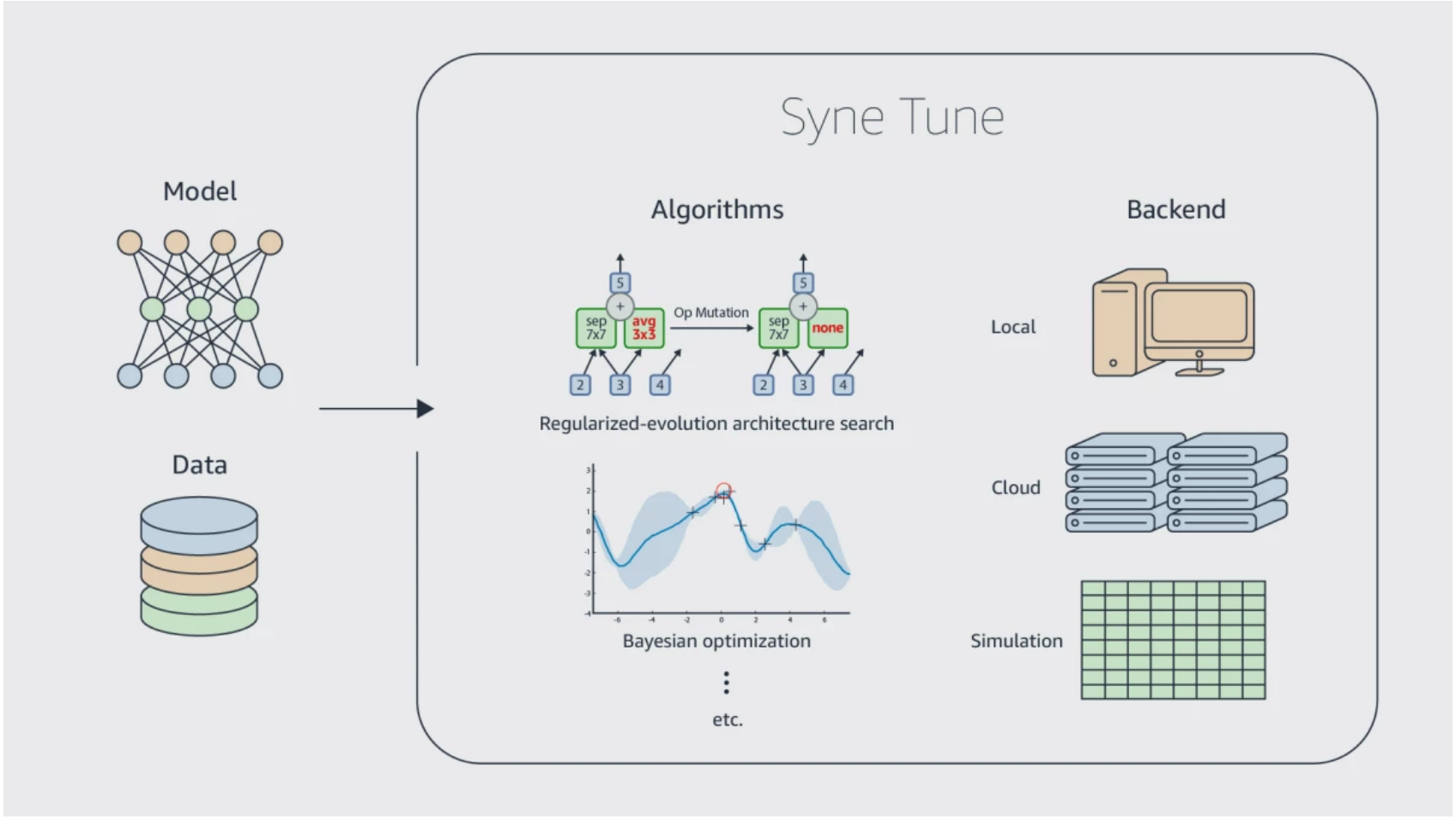
release 0.2 | 3.7 | 3.8 | 3.9 | License Apache 2.0 | downloads/month 2k

This package provides state-of-the-art distributed hyperparameter optimizers (HPO) where trials can be evaluated with several trial backend options (local backend to evaluate trials locally; SageMaker to evaluate trials as separate SageMaker training jobs; a simulation backend to quickly benchmark parallel asynchronous schedulers).

Installing

To install Syne Tune from pip, you can simply do:

```
pip install 'syne-tune'
```


Syne Tune
Model
Data
Algorithms
Op Mutation
sep 7x7
avg 3x3
sep 7x7
none
Regularized-evolution architecture search
Bayesian optimization
etc.
Backend
Local
Cloud
Simulation

Terminology

- Scheduler: the HPO algorithm that samples a new hyperparameter configuration
- Backend: manages the workers to parallelize the HPO process
- Trial: the evaluation of an hyperparameter configuration, which can be paused / resumed

High-level description of HPO loop and terminology

```
def tuner_loop(scheduler, trial_backend):
    while not_done():
        if worker_is_free():
            config = scheduler.suggest()
            trial_backend.start_trial(config)
        for result in trial_backend.fetch_new_results():
            decision = scheduler.on_trial_result(result)
            if decision == "stop":
                trial_backend.stop_trial(result.trial)
```

High-level description of HPO loop and terminology

```
def tuner_loop(scheduler, trial_backend):
    while not_done():
        if worker_is_free():
            config = scheduler.suggest()
            trial_backend.start_trial(config)
        for result in trial_backend.fetch_new_results():
            decision = scheduler.on_trial_result(result)
            if decision == "stop":
                trial_backend.stop_trial(result.trial)
```

- The tuner loops above searches the best configuration until stopping criterion is met

High-level description of HPO loop and terminology

```
def tuner_loop(scheduler, trial_backend):
    while not_done():
        if worker_is_free():
            config = scheduler.suggest()
            trial_backend.start_trial(config)
        for result in trial_backend.fetch_new_results():
            decision = scheduler.on_trial_result(result)
            if decision == "stop":
                trial_backend.stop_trial(result.trial)
```

- The tuner loops above searches the best configuration until stopping criterion is met
- Scheduler: An hpo algorithm that implements suggest(), on_trial_result() that returns the next candidate to evaluate and whether a trial should stop given its new result

High-level description of HPO loop and terminology

```
def tuner_loop(scheduler, trial_backend):
    while not_done():
        if worker_is_free():
            config = scheduler.suggest()
            trial_backend.start_trial(config)
        for result in trial_backend.fetch_new_results():
            decision = scheduler.on_trial_result(result)
            if decision == "stop":
                trial_backend.stop_trial(result.trial)
```

- The tuner loops above searches the best configuration until stopping criterion is met
- Scheduler: An hpo algorithm that implements suggest(), on_trial_result() that returns the next candidate to evaluate and whether a trial should stop given its new result
- Trial backend: how the blackbox is evaluated, responsible for starting/stopping jobs also doing the bookkeeping of results

How to use Syne Tune?

First Step: Annotating a training script

```
# train_network.py
import time
from argparse import ArgumentParser

if __name__ == '__main__':
    parser = ArgumentParser()
    parser.add_argument('--epochs', type=int)
    parser.add_argument('--num_layers', type=int)
    parser.add_argument('--learning_rate', type=float)
    args, _ = parser.parse_known_args()
    for epoch in range(args.epochs):
        mean_loss = train_epoch()
```

First Step: Annotating a training script

```
# train_neural_network.py
import time
from syne_tune import Reporter
from argparse import ArgumentParser

if __name__ == '__main__':
    parser = ArgumentParser()
    parser.add_argument('--epochs', type=int)
    parser.add_argument('--num_layers', type=int)
    parser.add_argument('--learning_rate', type=float)
    args, _ = parser.parse_known_args()
    report = Reporter()
    for epoch in range(args.epochs):
        mean_loss = train_epoch()
        # Feed the score back to Syne Tune.
        report(mean_loss=mean_loss, epoch=epoch + 1)
```

- The score is reported by calling `report(...)` at the last line
- It also automatically logs other metrics (runtime, dollar-cost when running on cloud machine) that can be optimized as well

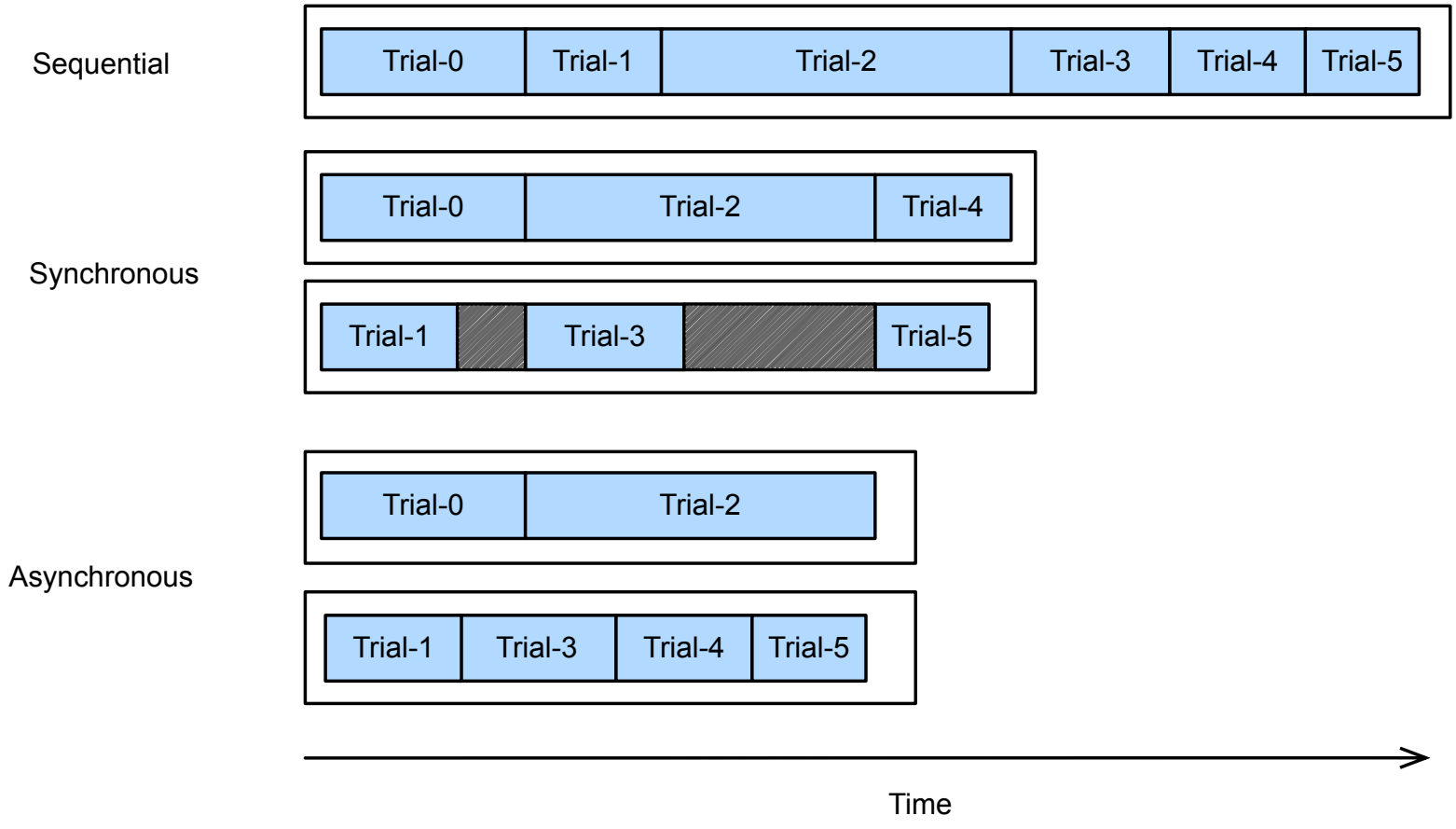
Second Step: Launch Syne Tune

```
from syne_tune import Tuner, StoppingCriterion
from syne_tune.backend import LocalBackend
from syne_tune.config_space import loguniform, randint
from syne_tune.optimizer.baselines import BayesianOptimization

# hyperparameter search space to consider
config_space = {
    'epochs': 100,
    'num_layers': randint(1, 20),
    'learning_rate': loguniform(1e-6, 1e-4)
}
tuner = Tuner(
    trial_backend=LocalBackend(entry_point='train_network.py'),
    scheduler=BayesianOptimization(
    config_space, metric='val_loss')
    ),
    stop_criterion=StoppingCriterion(max_wallclock_time=30),
    n_workers=4,  # how many trials are evaluated in parallel
)
tuner.run()
```

Distributed Hyperparameter Optimization

Synchronous vs Asynchronous Parallelization



The Backend

Execute trials locally:

```
backend = LocalBackend(entry_point="train_network.py")
tuner = Tuner(trial_backend=backend, n_workers=4, ...)
tuner.run()
```

The Backend

Execute trials locally:

```
backend = LocalBackend(entry_point="train_network.py")
tuner = Tuner(trial_backend=backend, n_workers=4, ...)
tuner.run()
```

Execute trials on the cloud, each as its own Sagemaker training job:

```
# define a Sagemaker estimator
trial_backend = SageMakerBackend(
    sm_estimator=PyTorch(
        entry_point="train_network.py",
        instance_type="ml.p2.xlarge",
        instance_count=1,
        framework_version='1.7.1',
    ),
)
```

The Backend

Execute trials locally:

```
backend = LocalBackend(entry_point="train_network.py")
tuner = Tuner(trial_backend=backend, n_workers=4, ...)
tuner.run()
```

Execute trials on the cloud, each as its own Sagemaker training job:

```
# define a Sagemaker estimator
trial_backend = SageMakerBackend(
    sm_estimator=PyTorch(
        entry_point="train_network.py",
        instance_type="ml.p2.xlarge",
        instance_count=1,
        framework_version='1.7.1',
    ),
)
```

Execute trials with simulations from lookup tables:

```
trial_backend = BlackboxRepositoryBackend(
    blackbox_name="nasbench201",
    dataset="cifar100",
)
```

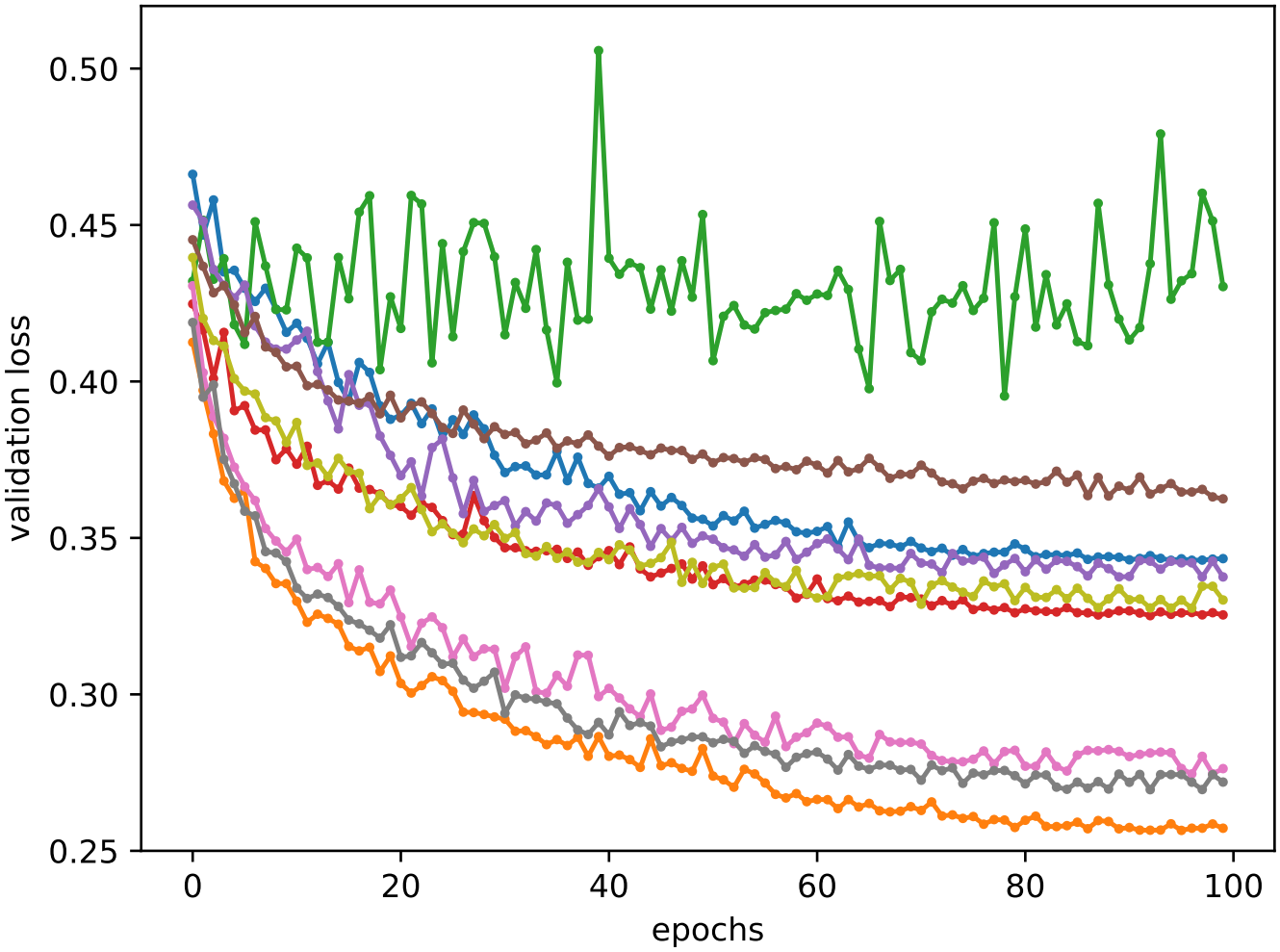
Multi-fidelity Hyperparameter Optimization

HPO is expensive

- Depending on the size of your configuration space, HPO might require ten to hundreds of trials to converge
- Parallelization reduces waiting time but not the compute

How can we speed up HPO?

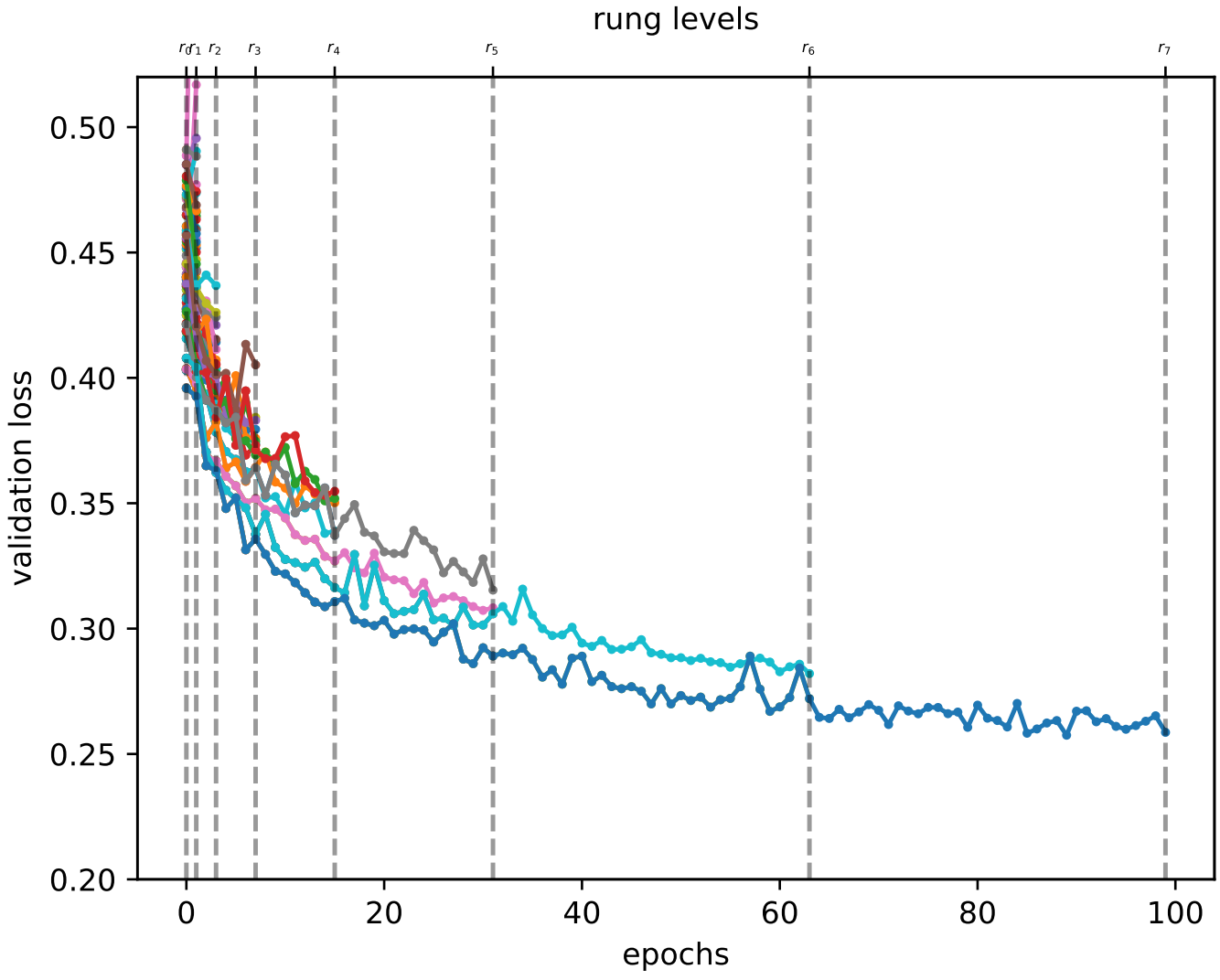
Learning Curves



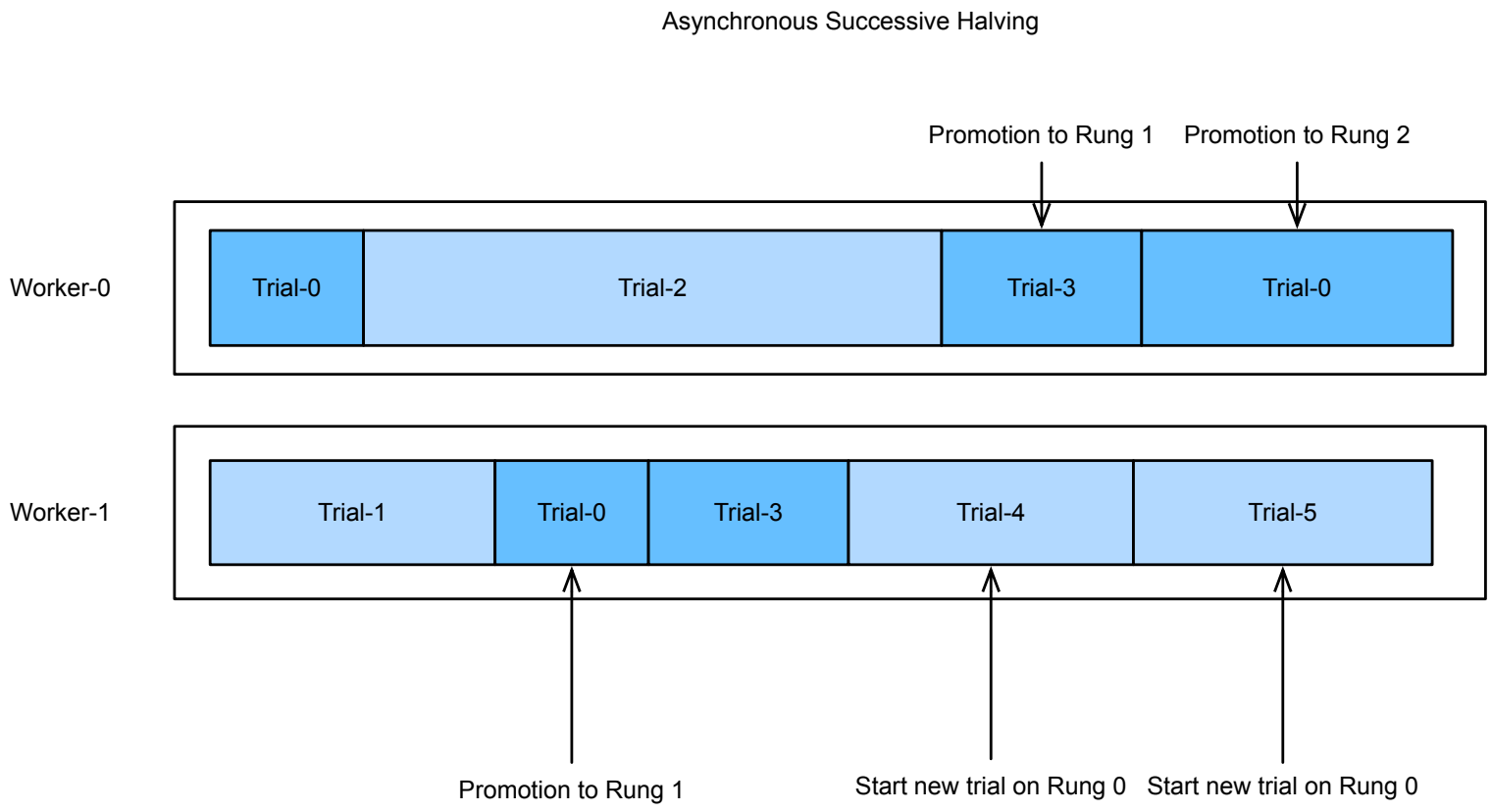
Multi-fidelity Hyperparameter Optimization

- We often have access to cheap-to-evaluate approximations $\tilde{f}(\boldsymbol{x}, b)$ of the true objective function $f(\boldsymbol{x})$, so called fidelities.
- Each fidelity is parameterized by a so-called budget $b \in [b_{min}, b_{max}]$.
  - if $b = b_{max}$: then $\tilde{f}(\boldsymbol{x}, b_{max}) = f(\boldsymbol{x})$
  - if $b < b_{max}$: then $\tilde{f}(\boldsymbol{x}, b)$ is only an approximation of $f(\boldsymbol{x})$ whose quality typically increases with b.
- Additionally we also often model the cost $c(\boldsymbol{x}, b)$, e.g runtime, it takes to evaluate $\boldsymbol{x}$ on budget b.

Successive Halving [JT16]



Asynchronous Successive Halving (ASHA) [LJR+18]



Successive Halving

Pros:

- easy to implement
- reduces the overall computation

Cons:

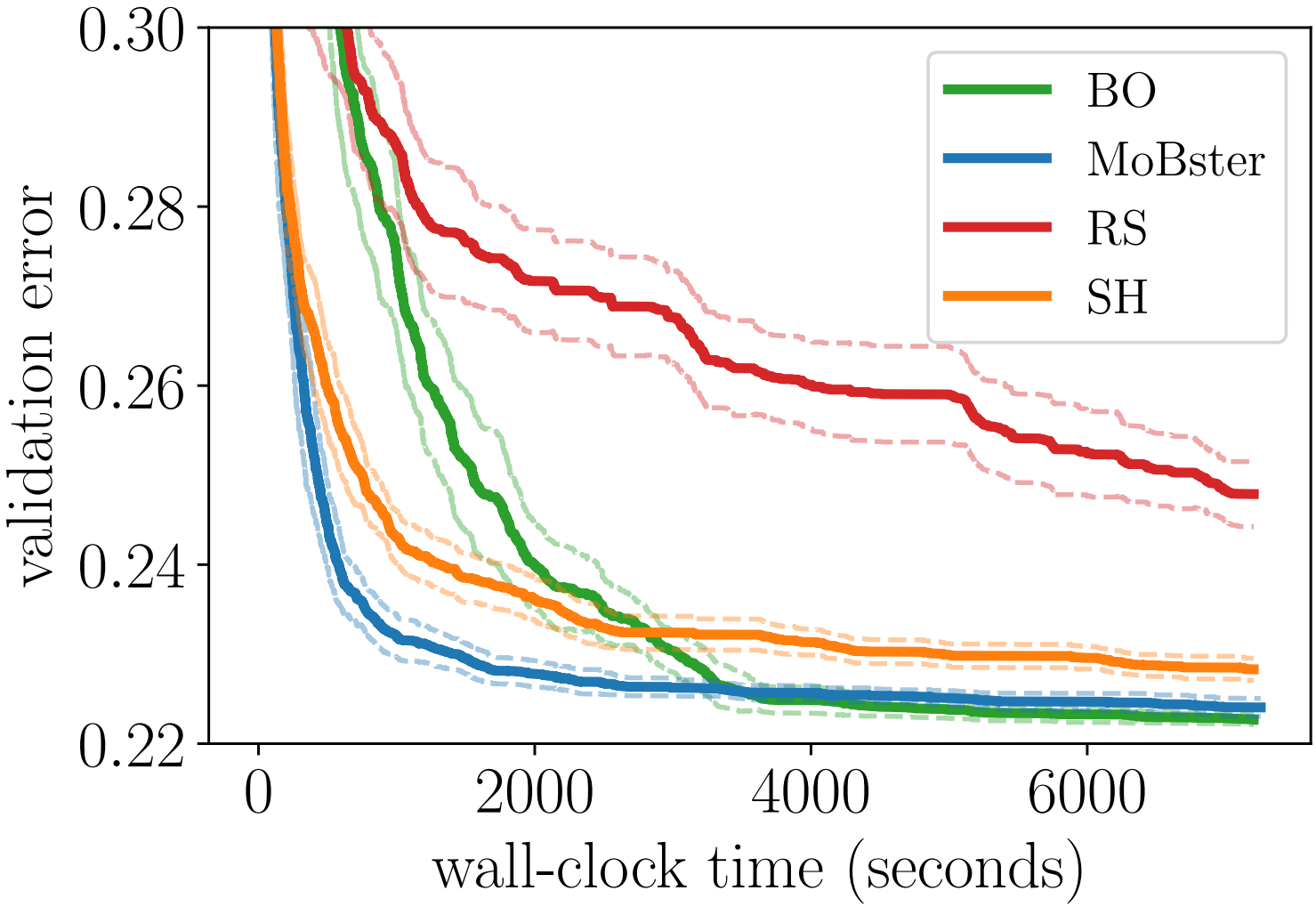
- not easy to parallelize in the asynchronous setting
- still based on random search

MoBster [KTL+20]

MOdel Based aSynchronous mulTi fidelity optimizER (MoBster):

- combines asynchronous successive halving with Bayesian optimization
- instead of modelling $f(\boldsymbol{x})$ we model $\tilde{f}(\boldsymbol{x}, b)$

MoBster


0.30
0.28
0.26
0.24
0.22
validation error
0
2000
4000
6000
wall-clock time (seconds)
BO
MoBster
RS
SH

MoBster

Pros:

- combines the benefits of successive halving and Bayesian optimization

Cons:

- not easy to implement

How to use it in Syne Tune?

```
from syne_tune import Tuner, StoppingCriterion
from syne_tune.backend import LocalBackend
from syne_tune.config_space import loguniform, randint
from syne_tune.optimizer.baselines import BayesianOptimization

# hyperparameter search space to consider
config_space = {
    'epochs': 100,
    'num_layers': randint(1, 20),
    'learning_rate': loguniform(1e-6, 1e-4)
}
tuner = Tuner(
    trial_backend=LocalBackend(entry_point='train_network.py'),
    scheduler=BayesianOptimization(config_space, metric='val_loss'),
    stop_criterion=StoppingCriterion(max_wallclock_time=30),
    n_workers=4,  # how many trials are evaluated in parallel
)
tuner.run()
```

How to use it in Syne Tune?

```
from syne_tune import Tuner, StoppingCriterion
from syne_tune.backend import LocalBackend
from syne_tune.config_space import loguniform, randint
from syne_tune.optimizer.baselines import MOBSTER

# hyperparameter search space to consider
config_space = {
    'epochs': 100,
    'num_layers': randint(1, 20),
    'learning_rate': loguniform(1e-6, 1e-4)
}
tuner = Tuner(
    trial_backend=LocalBackend(entry_point='train_network.py'),
    scheduler=MOBSTER(config_space, metric='val_loss', resource_attr='epochs'),
    stop_criterion=StoppingCriterion(max_wallclock_time=30),
    n_workers=4,  # how many trials are evaluated in parallel
)
tuner.run()
```

Multi-Objective Hyperparameter Optimization

More than just one objective.

In practice we often care about more things than just validation performance:

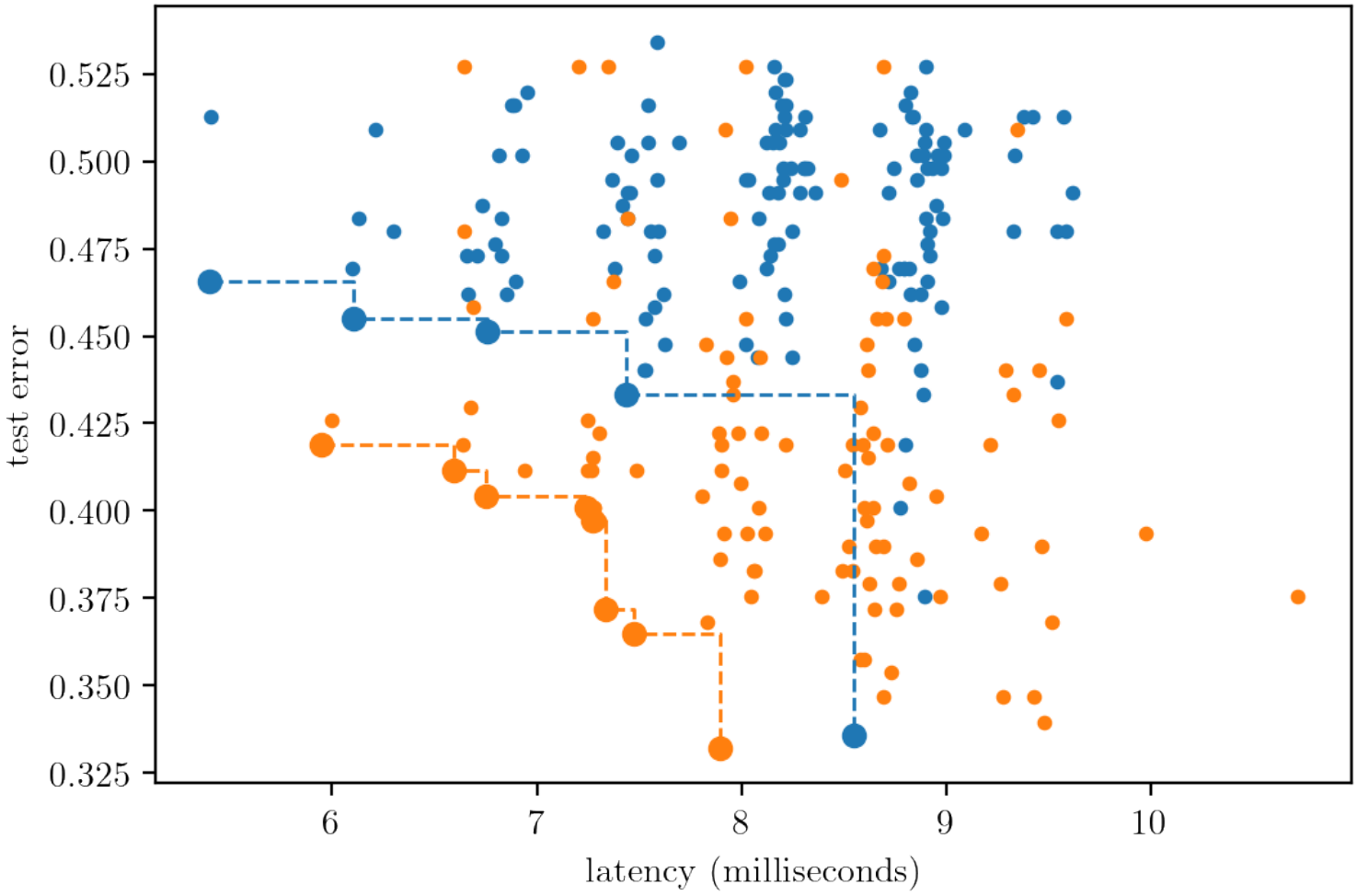
- memory consumption
- latency during inference
- training time
- ...

Multi-Objective Hyperparameter Optimization

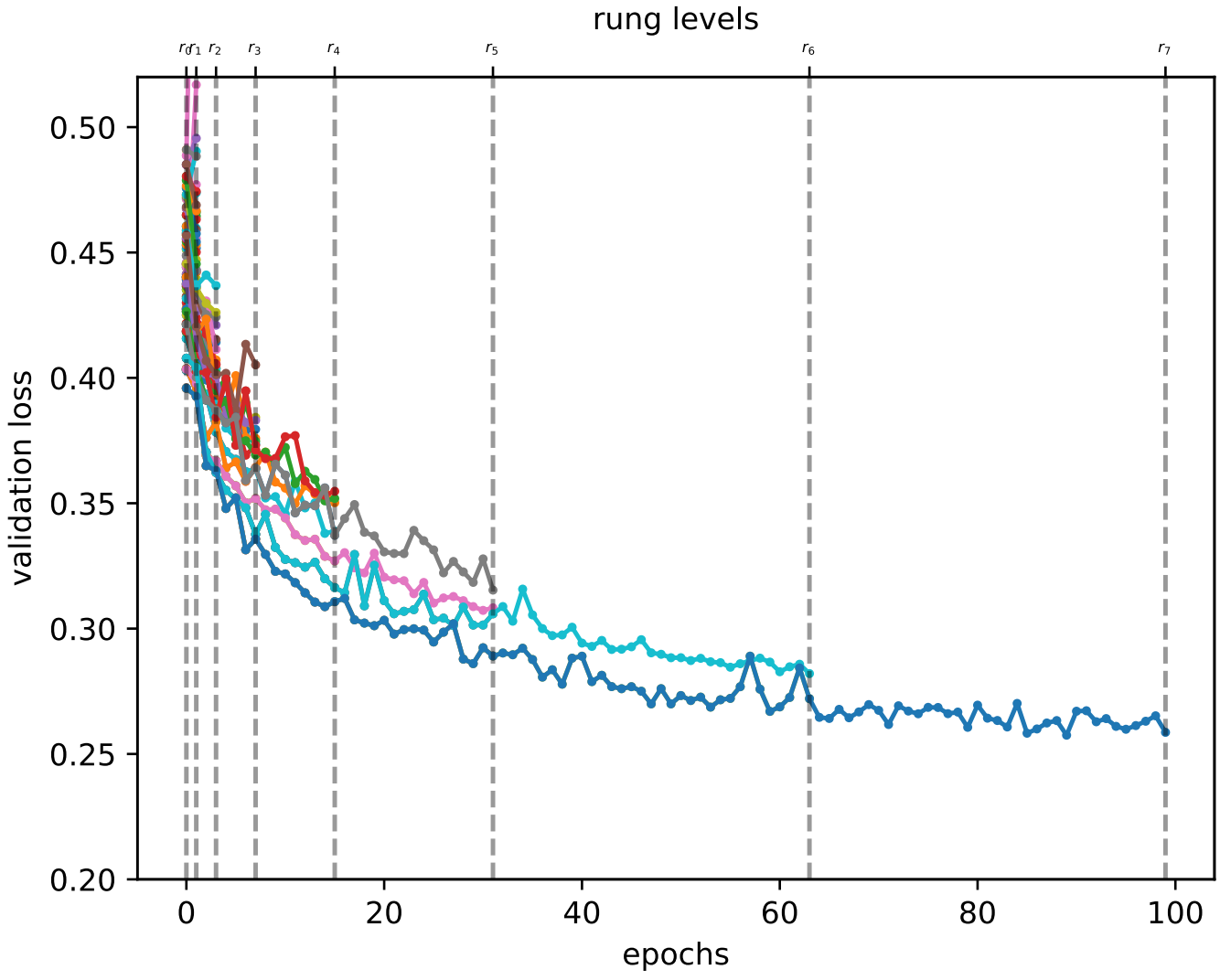
$$\min(f_0(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x}))$$

- usually there is no single $\boldsymbol{x}_\star \in \mathbb{X}$ that simultaneously optimizes $f_0, \ldots f_k$
- define $\boldsymbol{x}_0 \succ \boldsymbol{x}_1$ iff $f_i(\boldsymbol{x}_0) \leq f_i(\boldsymbol{x}_1), \forall i \in [k]$ and $\exists i \in [k]$ s.t $f_i(\boldsymbol{x}_0) < f_i(\boldsymbol{x}_1)$
- we aim to find the Pareto Front: $P_f = \{\boldsymbol{x} \in \mathbb{X} | \exists! \boldsymbol{x}' \in \mathbb{X} : \boldsymbol{x}' \succ \boldsymbol{x}\}$

Multi-Objective Hyperparameter Optimization

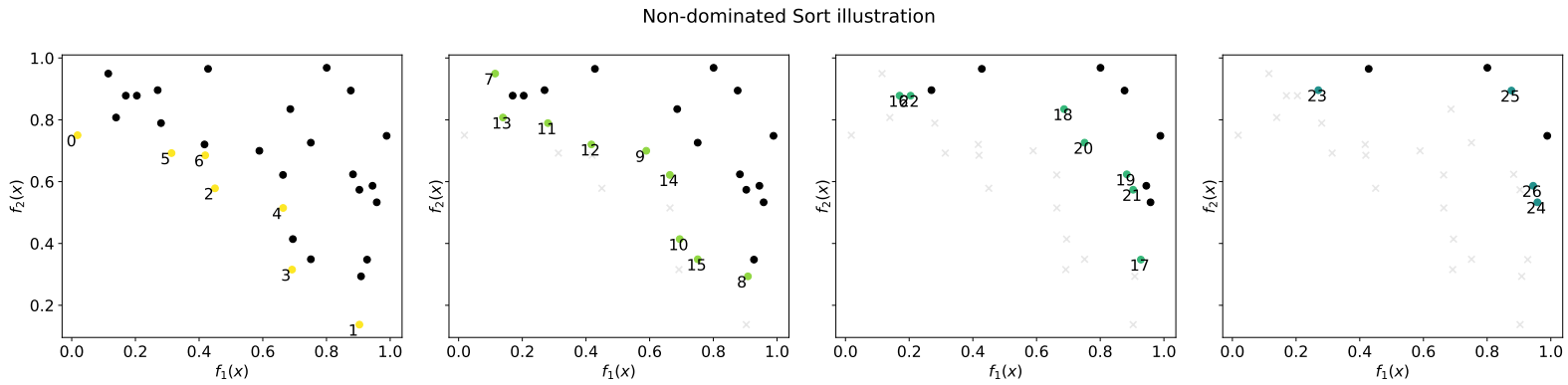


Successive Halving



Multi-Objective ASHA [SDZ+21]

To extend Sucessive Halving to the multi-objective scenario, we use non-dominated sorting to sort configurations at each rung level



How to use it in Syne Tune?

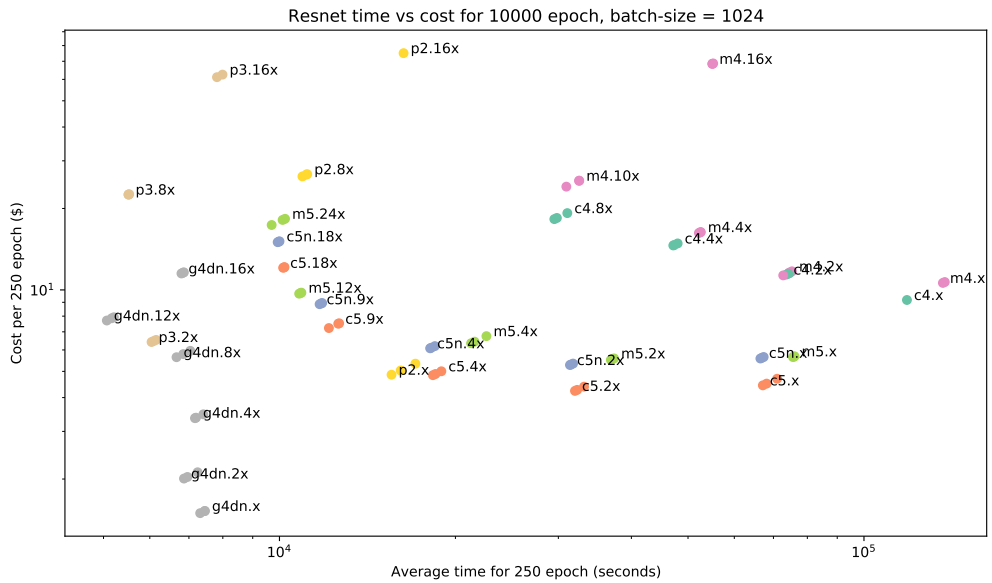
```
from syne_tune import Tuner, StoppingCriterion
from syne_tune.backend import LocalBackend
from syne_tune.config_space import loguniform, randint
from syne_tune.optimizer.baselines import MOASHA

# hyperparameter search space to consider
config_space = {
    'epochs': 100,
    'num_layers': randint(1, 20),
    'learning_rate': loguniform(1e-6, 1e-4)
}
tuner = Tuner(
    trial_backend=LocalBackend(entry_point='train_network.py'),
    scheduler = MOASHA(time_attr='epochs', metrics=['val_loss', 'latency'],
    mode=['min', 'min'], config_space=config_space),
    stop_criterion=StoppingCriterion(max_wallclock_time=30),
    n_workers=4,  # how many trials are evaluated in parallel
)
tuner.run()
```

Use-cases

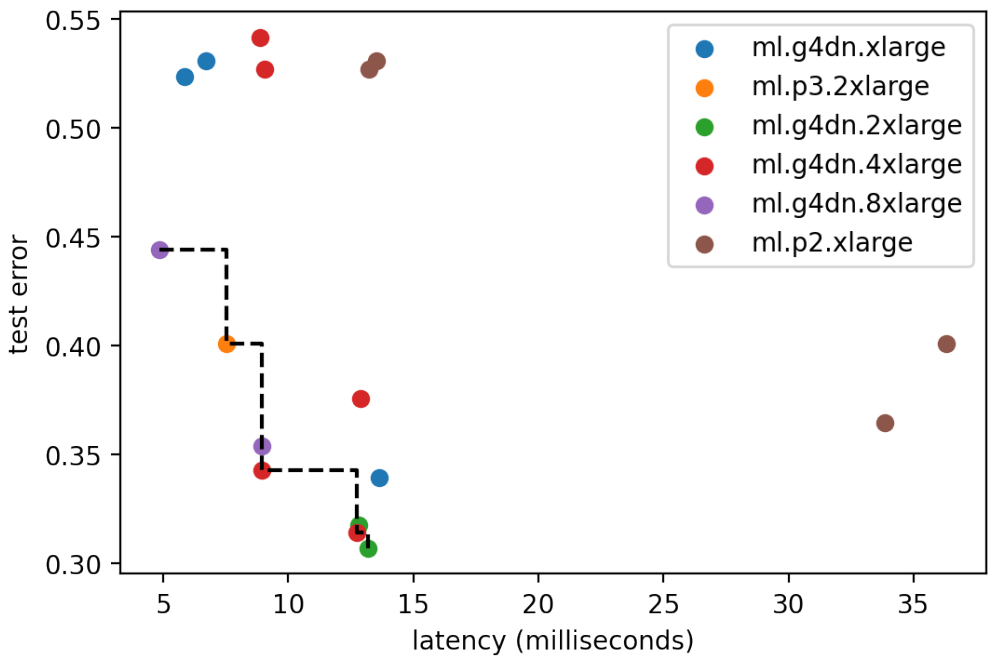
Use-case 1: Tune hardware with multi-objective optimization

- Hardware or machine type is often let constant...
- But it has a massive effect on cost/runtime and requires expertise to be set correctly



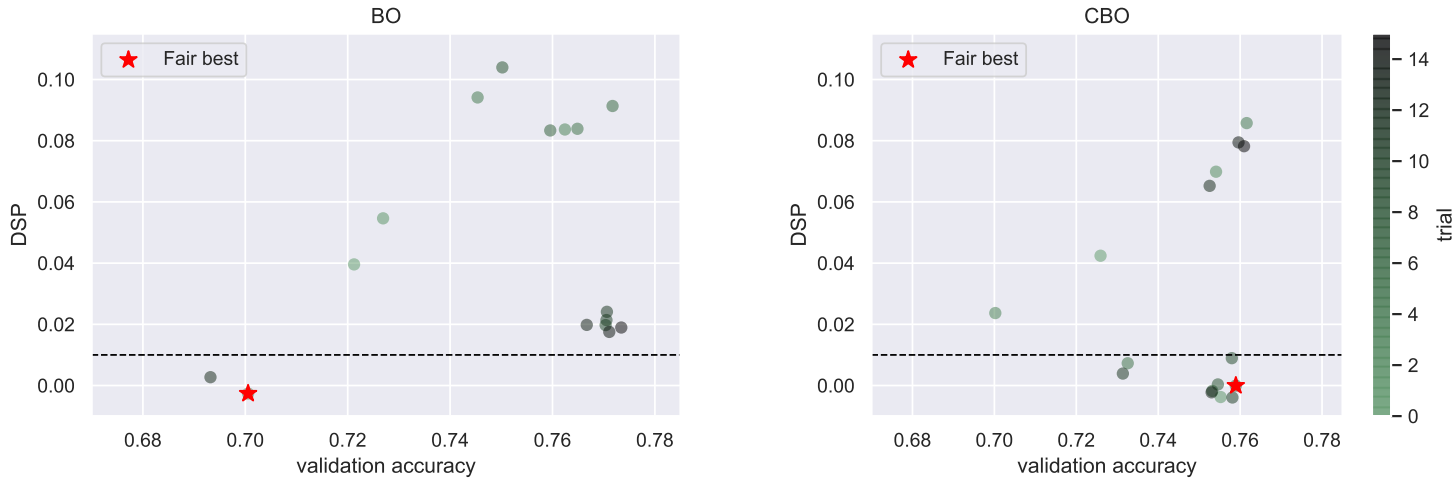
Use-case 1: Tune hardware with multi-objective optimization

- Blackbox: fine-tune BERT on RTE from GLUE benchmark
- Objectives: latency and test error
- Hyperparameters: learning-rate, batch-size, warm-up ratio of learning-rate schedule, model type and instance-type



Use-case 2: Fair HPO [PDZ+21]

Objective: Find optimal hyperparameters that meet a certain statistical fairness criteria: $\boldsymbol{x}_\star \in \arg\min_{\boldsymbol{x}\in\mathbb{X}} f(\boldsymbol{x})\text{s.t.}c(\boldsymbol{x}) \geq \delta$



Available Methods in Syne Tune

name	reference	(a)sync	multi-fidelity	search
RS	[BB12]	async	no	random
GP	[SLA12]	async	no	BO
TPE	[BBBK11]	async	no	density-ratio estimation
REA	[RAHL19]	async	no	evolution
BORE	[TKS+21]	async	no	density-ratio estimation
ASHA	[LJR+18]	async	stopping/promotion	random
MOBSTER	[KTL+20]	async	stopping/promotion	BO
BOHB	[FKH18]	async	stopping/promotion	TPE
MSR	[GSM+17]	async	stopping	random
SyncHyperband	[LJD+17]	sync	promotion	random
SyncBOHB	[FKH18]	sync	promotion	TPE
PBT	[JDO+17]	sync	promotion	evolution
RUSH	[ZSA21]	async	stopping/promotion	random
ASHABB	[PSS+19]	async	stopping/promotion	random
ASHACTS	[SSP20]	async	stopping/promotion	parametric surrogate
ZS	[WSS15]	async	no	greedy-selection
MOASHA	[SDZ+21]	async	stopping	random
CBO	[GKX+14]	async	no	BO (CEI)

Conclusions

- Hyperparameter optimization systematically finds the right hyperparameters and architectural design choices.
- Bayesian optimization is usually more sample efficient than grid search or random search
- Syne Tune is an open-source library for HPO that is leveraged in Automated Model Tuning and Autopilot
- We can speed up the optimization process by using multi-fidelity optimization
- Multi-objective optimization for simultaneously optimizing multiple objectives

Resources

- GitHub: https://github.com/awslabs/syne-tune
- Paper: https://openreview.net/forum?id=BVeGJ-THIg9
- Slack channel: *#syne-tune-interest*
- Blog Posts:
  - Run distributed hyperparameter and neural architecture tuning jobs with Syne Tune
  - Hyperparameter optimization for fine-tuning pre-trained transformer models from Hugging Face
  - A hyperparameter optimization library for reproducible research
  - Learn Amazon Simple Storage Service transfer configuration with Syne Tune
- Twitter: @kleiaaro

References

[BB12] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.

[BBBK11] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NIPS'11)*, 2011.

[CHW+18] Y. Chen, A. Huang, Z. Wang, I. Antonoglou, J. Schrittwieser, D. Silver, and N. de Freitas. Bayesian optimization in alphago. *arXiv:1812.06855 [cs.LG]*, 2018.

[FKH18] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

[GKX+14] J. Gardner, M. Kusner, Z. Xu, K. Weinberger, and J. Cunningham. Bayesian optimization with inequality constraints. In *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*, 2014.

[GSM+17] 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*, 2017.

[JDO+17] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population based training of neural networks. *arXiv:1711.09846 [cs.LG]*, 2017.

[JT16] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS'16)*, 2016.

[KTL+20] A. Klein, L. C. Tiao, T. Lienart, C. Archambeau, and M. Seeger. Model-based asynchronous hyperparameter optimization. *arXiv:2003.10865 [cs.LG]*, 2020.

[LH19] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR'19)*, 2019.

[LJD+17] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *International Conference on Learning Representations (ICLR'17)*, 2017.

[LJR+18] L. Li, K. Jamieson, A. Rostamizadeh, K. Gonina, M. Hardt, B. Recht, and A. Talwalkar. Massively parallel hyperparameter tuning. *arXiv:1810.05934 [cs.LG]*, 2018.

[PDZ+21] V. Perrone, M. Donini, M. B. Zafar, R. Schmucker, K. Kenthapadi, and C. Archambeau. Fair Bayesian optimization. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, 2021.

[PSS+19] V. Perrone, H. Shen, M. Seeger, C. Archambeau, and R. Jenatton. Learning search spaces for Bayesian optimization: Another view of hyperparameter transfer learning. In *Proceedings of the 32th International Conference on Advances in Neural Information Processing Systems (NIPS'19)*, 2019.

[RAHL19] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized Evolution for Image Classifier Architecture Search. In *Proceedings of the Conference on Artificial Intelligence (AAAI'19)*, 2019.

[SDZ+21] R. Schmucker, M. Donini, M. B. Zafar, D. Salinas, and C. Archambeau. Multi-objective asynchronous successive halving. *arXiv:2106.12639 [stat.ML]*, 2021.

[SLA12] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NIPS'12)*, 2012.

[SSP20] D. Salinas, H. Shen, and V. Perrone. A quantile-based approach for hyperparameter transfer learning. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*, 2020.

[TKS+21] L. C. Tiao, A. Klein, M. W. Seeger, E. V. Bonilla, C. Archambeau, and F. Ramos. Bore: Bayesian optimization by density-ratio estimation. In *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*, 2021.

[WSS15] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Sequential model-free hyperparameter tuning. In *IEEE International Conference on Data Mining, ICDM 2015*, pages 1033–1038, 2015.

[YKR+19] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, 2019.

[ZSA21] G. Zappella, D. Salinas, and C. Archambeau. A resource-efficient method for repeated HPO and NAS problems. *arXiv:2103.16111 [cs.LG]*, 2021.