AutoML: Dynamic Configuration & Learning

Population-based Training

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

On-the-fly Adaption

- Dynamic algorithm configuration assumes that we have access to a representative learning environment in an offline learning phase

On-the-fly Adaption

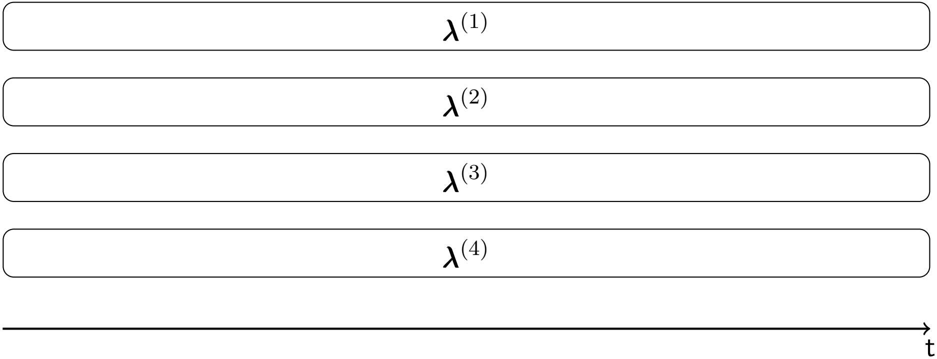
- Dynamic algorithm configuration assumes that we have access to a representative learning environment in an offline learning phase
- What if we don't access to such an env or don't have to time for offline learning?

On-the-fly Adaption

- Dynamic algorithm configuration assumes that we have access to a representative learning environment in an offline learning phase
- What if we don't access to such an env or don't have to time for offline learning?

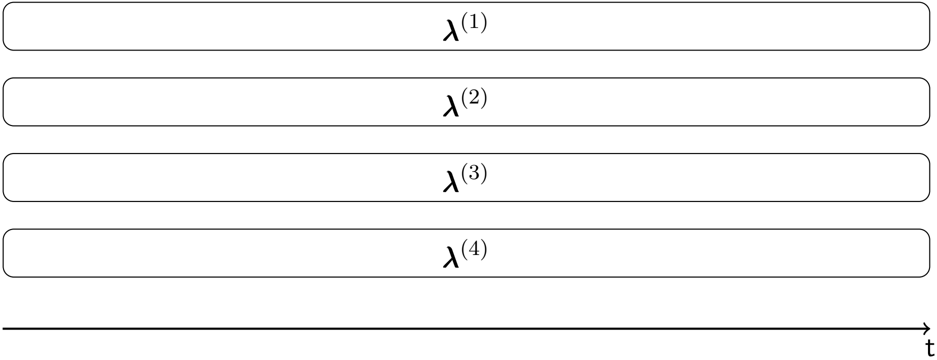
$\rightsquigarrow$ Try to figure out best hyperparameter settings on the fly

Massively parallelized Random Search



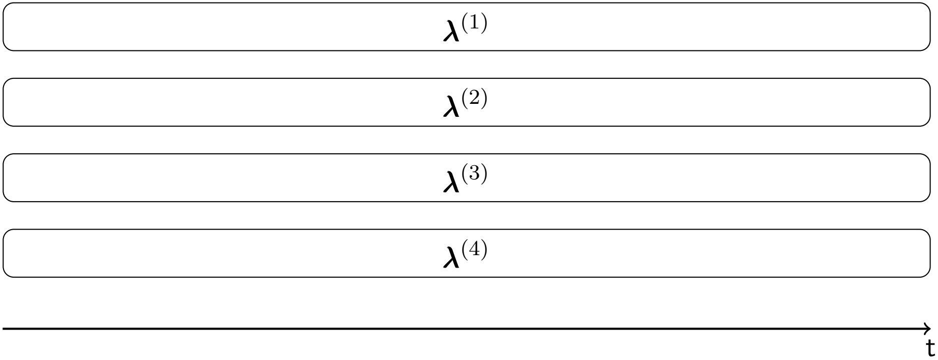
- Sample many hyperparameter configurations $\boldsymbol{\lambda}^{(i)}$ and evaluate them all in parallel

Massively parallelized Random Search



- Sample many hyperparameter configurations $\boldsymbol{\lambda}^{(i)}$ and evaluate them all in parallel
- Pure exploration on a large population of configurations

Massively parallelized Random Search

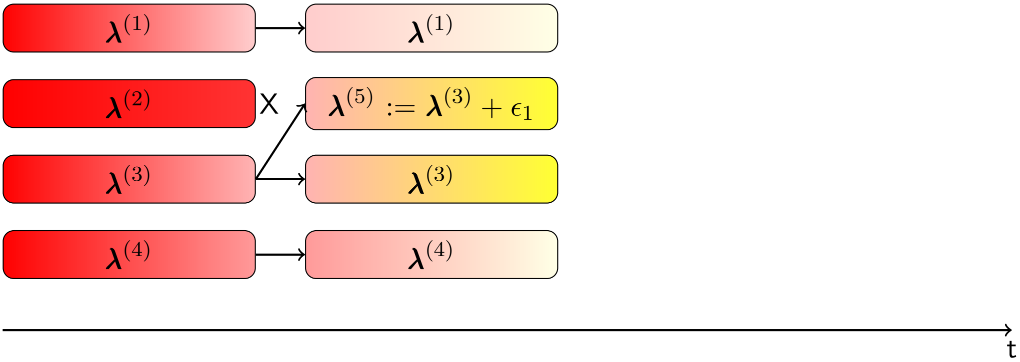


- Sample many hyperparameter configurations $\boldsymbol{\lambda}^{(i)}$ and evaluate them all in parallel
- Pure exploration on a large population of configurations
- No dynamic adaptation

Population-based Training [Jaderberg et al. 2017]

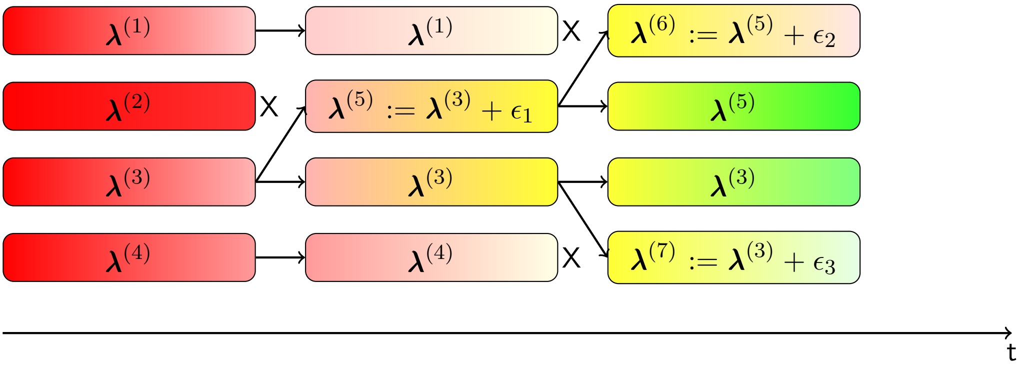
- The color indicates the performance over time

Population-based Training [Jaderberg et al. 2017]



- The color indicates the performance over time

Population-based Training [Jaderberg et al. 2017]



- The color indicates the performance over time

Population-based Training [Jaderberg et al. 2017; Liang et al. 2020]

General workflow of PBT:

1. Sample initial population
   - Each population member is a combination of hyperparameter setting $\boldsymbol{\lambda}$ and (partially trained) model

Population-based Training [Jaderberg et al. 2017; Liang et al. 2020]

General workflow of PBT:

1. Sample initial population
   - Each population member is a combination of hyperparameter setting $\boldsymbol{\lambda}$ and (partially trained) model
2. Train population for a bit

Population-based Training [Jaderberg et al. 2017; Liang et al. 2020]

General workflow of PBT:

1. Sample initial population
   - Each population member is a combination of hyperparameter setting $\boldsymbol{\lambda}$ and (partially trained) model
2. Train population for a bit
3. Tournament selection to drop poorly performing population members

Population-based Training [Jaderberg et al. 2017; Liang et al. 2020]

General workflow of PBT:

1. Sample initial population
   - Each population member is a combination of hyperparameter setting $\boldsymbol{\lambda}$ and (partially trained) model
2. Train population for a bit
3. Tournament selection to drop poorly performing population members
4. Use mutation (and cross-over) to generate off-springs
   - Change the hyperparameter settings, but inherits the partially trained model (+ pertubation)

Population-based Training [Jaderberg et al. 2017; Liang et al. 2020]

General workflow of PBT:

1. Sample initial population
   - Each population member is a combination of hyperparameter setting $\boldsymbol{\lambda}$ and (partially trained) model
2. Train population for a bit
3. Tournament selection to drop poorly performing population members
4. Use mutation (and cross-over) to generate off-springs
   - Change the hyperparameter settings, but inherits the partially trained model (+ pertubation)

$\rightsquigarrow$ New population consists of so-far best performing ones and new off-springs

Population-based Training [Jaderberg et al. 2017; Liang et al. 2020]

General workflow of PBT:

1. Sample initial population
   - Each population member is a combination of hyperparameter setting $\boldsymbol{\lambda}$ and (partially trained) model
2. Train population for a bit
3. Tournament selection to drop poorly performing population members
4. Use mutation (and cross-over) to generate off-springs
   - Change the hyperparameter settings, but inherits the partially trained model (+ pertubation)

$\rightsquigarrow$ New population consists of so-far best performing ones and new off-springs

5. Go to 2.

Properties of Population-based Training (PBT)

- PBT returns an already trained model (e.g., DNN or RL policy)

Properties of Population-based Training (PBT)

- PBT returns an already trained model (e.g., DNN or RL policy)
- PBT uses evolutionary computing to determine well-performing hyperparameter settings

Properties of Population-based Training (PBT)

- PBT returns an already trained model (e.g., DNN or RL policy)
- PBT uses evolutionary computing to determine well-performing hyperparameter settings
- Since hyperparameter settings changes while training the models,
  PBT relates to dynamic algorithm configuration

Properties of Population-based Training (PBT)

- PBT returns an already trained model (e.g., DNN or RL policy)
- PBT uses evolutionary computing to determine well-performing hyperparameter settings
- Since hyperparameter settings changes while training the models,
  PBT relates to dynamic algorithm configuration
- Since each population member (i.e., model) can be trained independently,
  PBT can be efficiently parallelized
  - $\rightsquigarrow$ Drawback: requires substantial parallel compute resources

Combining Population-based Training and Bayesian Optimization

- Bayesian Optimization (BO) is well known for its sample efficiency

Combining Population-based Training and Bayesian Optimization

- Bayesian Optimization (BO) is well known for its sample efficiency
- Idea: Can we use BO to guide PBT?

Combining Population-based Training and Bayesian Optimization

- Bayesian Optimization (BO) is well known for its sample efficiency
- Idea: Can we use BO to guide PBT?

$\rightsquigarrow$ Less parallel compute resources are required(?)

Combining Population-based Training and Bayesian Optimization

- Bayesian Optimization (BO) is well known for its sample efficiency
- Idea: Can we use BO to guide PBT?

$\rightsquigarrow$ Less parallel compute resources are required(?)

$\rightsquigarrow$ Scales better to higher dimensional spaces(?)

PBT + BO: Outline

1. Sample initial population
   - Each population member is a combination of hyperparameter setting $\boldsymbol{\lambda}$ and (partially trained) model
2. Train population for a bit
3. Tournament selection to drop poorly performing population members
4. Use Bayesian optimization to select new hyperparameter settings
   - Change the hyperparameter settings, but inherits the partially trained model (+ pertubation)

$\rightsquigarrow$ New population consists of so-far best performing ones and new off-springs

5. Go to 2.

PBT + BO: Parallel Evaluation

- Challenge I: PBT runs in parallel asynchronously

PBT + BO: Parallel Evaluation

- Challenge I: PBT runs in parallel asynchronously

$\rightsquigarrow$ BO has to take into account that other hyperparameter settings are being evaluated already

PBT + BO: Parallel Evaluation

- Challenge I: PBT runs in parallel asynchronously

⇝ BO has to take into account that other hyperparameter settings are being evaluated already

- Several ideas on how to parallelize BO

PBT + BO: Parallel Evaluation

- Challenge I: PBT runs in parallel asynchronously

$\rightsquigarrow$ BO has to take into account that other hyperparameter settings are being evaluated already

- Several ideas on how to parallelize BO
  - Randomize the model training or optimization of the acquisition function

PBT + BO: Parallel Evaluation

- Challenge I: PBT runs in parallel asynchronously

$\rightsquigarrow$ BO has to take into account that other hyperparameter settings are being evaluated already

- Several ideas on how to parallelize BO
  - Randomize the model training or optimization of the acquisition function
  - Thompson sampling to use only a single explanation of the data (in proportion to its likelihood)

PBT + BO: Parallel Evaluation

- Challenge I: PBT runs in parallel asynchronously

$\rightsquigarrow$ BO has to take into account that other hyperparameter settings are being evaluated already

- Several ideas on how to parallelize BO
  - Randomize the model training or optimization of the acquisition function
  - Thompson sampling to use only a single explanation of the data (in proportion to its likelihood)
  - Hallucinate performance of other hyperparameter settings in optimistically, pessimistically or in expectation of the current surrogate model

PBT + BO: Parallel Evaluation [Parker-Holder et al. 2020]

- Challenge II: The cost depends on the previous $\boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)}, \ldots, \boldsymbol{\lambda}^{(t-1)}$

PBT + BO: Parallel Evaluation [Parker-Holder et al. 2020]

- Challenge II: The cost depends on the previous $\boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)}, \dots, \boldsymbol{\lambda}^{(t-1)}$
- BO-Surrogate model predicts the cost improvement over time:

$$c_{\mathsf{PBT}}^{(t)}(\boldsymbol{\lambda}) = \frac{c^{(t)}(\boldsymbol{\lambda}) - c^{(t-1)}(\boldsymbol{\lambda})}{\Delta t}$$

where $c^{(t)}(\boldsymbol{\lambda})$ is the cost for a given hyperparameter setting at time step t.

PBT + BO: Parallel Evaluation [Parker-Holder et al. 2020]

- Challenge II: The cost depends on the previous $\boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)}, \ldots, \boldsymbol{\lambda}^{(t-1)}$
- BO-Surrogate model predicts the cost improvement over time:

$$c^{(t)}_{\text{PBT}}(\boldsymbol{\lambda}) = \frac{c^{(t)}(\boldsymbol{\lambda}) - c^{(t-1)}(\boldsymbol{\lambda})}{\Delta t}$$

where $c^{(t)}(\boldsymbol{\lambda})$ is the cost for a given hyperparameter setting at time step t.

- Remark: Also add $c^{(t-1)}$ as an input to the BO-surrogate model to ease the task of predicting the improvement