AutoML: Dynamic Configuration & Learning

Overview

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Black vs. Grey vs. White Box

- Often we treat AutoML as a black-box problem
  - Black box: We choose input to the black box and observe outcome

Black vs. Grey vs. White Box

- Often we treat AutoML as a black-box problem
  - Black box: We choose input to the black box and observe outcome
  - E.g., classical hyperparameter optimizer:
    Input: Hyperparameter configuration $\rightarrow$ Output: Accuracy

Black vs. Grey vs. White Box

- Often we treat AutoML as a black-box problem
  - Black box: We choose input to the black box and observe outcome
  - E.g., classical hyperparameter optimizer:
    Input: Hyperparameter configuration $\rightarrow$ Output: Accuracy
- We discussed how to extend AutoML to a more grey-box approach:
  - Grey Box: We still choose the input, but we can observe more than the outcome, e.g, intermediate results

Black vs. Grey vs. White Box

- Often we treat AutoML as a black-box problem
  - Black box: We choose input to the black box and observe outcome
  - E.g., classical hyperparameter optimizer:
    Input: Hyperparameter configuration $\rightarrow$ Output: Accuracy
- We discussed how to extend AutoML to a more grey-box approach:
  - Grey Box: We still choose the input, but we can observe more than the outcome, e.g, intermediate results
  - We might can control the "box" a bit, e.g., early termination

Black vs. Grey vs. White Box

- Often we treat AutoML as a black-box problem
  - Black box: We choose input to the black box and observe outcome
  - E.g., classical hyperparameter optimizer:
    Input: Hyperparameter configuration $\rightarrow$ Output: Accuracy
- We discussed how to extend AutoML to a more grey-box approach:
  - Grey Box: We still choose the input, but we can observe more than the outcome, e.g, intermediate results
  - We might can control the "box" a bit, e.g., early termination
  - E.g., learning curve predictions, multi-fidelity optimization, ...

Black vs. Grey vs. White Box

- Often we treat AutoML as a black-box problem
  - Black box: We choose input to the black box and observe outcome
  - E.g., classical hyperparameter optimizer:
    Input: Hyperparameter configuration $\rightarrow$ Output: Accuracy
- We discussed how to extend AutoML to a more grey-box approach:
  - Grey Box: We still choose the input, but we can observe more than the outcome, e.g, intermediate results
  - We might can control the "box" a bit, e.g., early termination
  - E.g., learning curve predictions, multi-fidelity optimization, ...

  $\rightsquigarrow$ often more efficient than black-box approaches (if done right)

Black vs. Grey vs. White Box

- Often we treat AutoML as a black-box problem
  - Black box: We choose input to the black box and observe outcome
  - E.g., classical hyperparameter optimizer:
    Input: Hyperparameter configuration $\rightarrow$ Output: Accuracy
- We discussed how to extend AutoML to a more grey-box approach:
  - Grey Box: We still choose the input, but we can observe more than the outcome, e.g, intermediate results
  - We might can control the “box” a bit, e.g., early termination
  - E.g., learning curve predictions, multi-fidelity optimization, ...

  $\rightsquigarrow$ often more efficient than black-box approaches (if done right)
- Ultimately, we would like to treat AutoML as a white-box problem
  - White-box: We can observe and control all details of an algorithm run

Black vs. Grey vs. White Box

- Often we treat AutoML as a black-box problem
  - Black box: We choose input to the black box and observe outcome
  - E.g., classical hyperparameter optimizer:
    Input: Hyperparameter configuration $\rightarrow$ Output: Accuracy
- We discussed how to extend AutoML to a more grey-box approach:
  - Grey Box: We still choose the input, but we can observe more than the outcome, e.g, intermediate results
  - We might can control the "box" a bit, e.g., early termination
  - E.g., learning curve predictions, multi-fidelity optimization, ...
  - $\rightsquigarrow$ often more efficient than black-box approaches (if done right)
- Ultimately, we would like to treat AutoML as a white-box problem
  - White-box: We can observe and control all details of an algorithm run

$\rightsquigarrow$ Goal: Replace algorithm components by learned policies

Iterative Optimization Heuristics

IOHs

Iterative Optimization Heuristics (IOHs) propose a set of solution candidates in each iteration based on previous evaluations.

Iterative Optimization Heuristics

IOHs

Iterative Optimization Heuristics (IOHs) propose a set of solution candidates in each iteration based on previous evaluations.

Important Observations:

- Many ML algorithms are iterative in nature, in particular for big data, e.g.:
  - SGD (for linear models or for deep neural networks)
  - Tree-based algorithms

Iterative Optimization Heuristics

IOHs

Iterative Optimization Heuristics (IOHs) propose a set of solution candidates in each iteration based on previous evaluations.

Important Observations:

- Many ML algorithms are iterative in nature, in particular for big data, e.g.:
  - SGD (for linear models or for deep neural networks)
  - Tree-based algorithms
- Often we have only a single solution candidate (e.g., weights of neural network)
  - If we use a evoluationary strategy as in neural evoluation, we have a population of solution candidates

Iterative Optimization Heuristics

IOHs

Iterative Optimization Heuristics (IOHs) propose a set of solution candidates in each iteration based on previous evaluations.

Important Observations:

- Many ML algorithms are iterative in nature, in particular for big data, e.g.:
  - SGD (for linear models or for deep neural networks)
  - Tree-based algorithms
- Often we have only a single solution candidate
  (e.g., weights of neural network)
  - If we use a evoluationary strategy as in neural evoluation,
    we have a population of solution candidates
- Hopefully, the quality of solution candiates improves in each iteration
  - Update of the weights of a neural network

Iterative Optimization Heuristics

IOHs

Iterative Optimization Heuristics (IOHs) propose a set of solution candidates in each iteration based on previous evaluations.

Important Observations:

- Many ML algorithms are iterative in nature, in particular for big data, e.g.:
  - SGD (for linear models or for deep neural networks)
  - Tree-based algorithms
- Often we have only a single solution candidate
  (e.g., weights of neural network)
  - If we use a evoluationary strategy as in neural evolution,
    we have a population of solution candidates
- Hopefully, the quality of solution candiates improves in each iteration
  - Update of the weights of a neural network
- Main component is the heuristic for proposal mechanism of new solution candidates

Learning for IOHs

Dynamic Adaptation of Hyperparameters

The goal is to dynamically adapt hyperparameters based on some feedback from the algorithm.

Learning for IOHs

Dynamic Adaptation of Hyperparameters

The goal is to dynamically adapt hyperparameters based on some feedback from the algorithm.

Dynamic Algorithm Configuration: DAC

The goal of DAC is to learn a policy from data
that adapts the hyperparameter settings of an IOH.

Learning for IOHs

Dynamic Adaptation of Hyperparameters

The goal is to dynamically adapt hyperparameters based on some feedback from the algorithm.

Dynamic Algorithm Configuration: DAC

The goal of DAC is to learn a policy from data
that adapts the hyperparameter settings of an IOH.

Learning to Learn: L2L

The goal of L2L is to learn a proposal mechanism from data.

AutoML: Dynamic Configuration & Learning

Dynamic Configuration

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Iterative Optimization Heuristics

- Many iterative heuristics in algorithms are dynamic and adaptive
  1. the algorithm's behavior changes over time
  2. the algorithm's behavior changes based on internal statistics

Iterative Optimization Heuristics

- Many iterative heuristics in algorithms are dynamic and adaptive
  1. the algorithm's behavior changes over time
  2. the algorithm's behavior changes based on internal statistics

- These heuristics might control other hyperparameters of the algorithms

Iterative Optimization Heuristics

- Many iterative heuristics in algorithms are dynamic and adaptive
  1. the algorithm's behavior changes over time
  2. the algorithm's behavior changes based on internal statistics

- These heuristics might control other hyperparameters of the algorithms
- Example: learning rate schedules for training DNNs
  1. exponential decaying learning rate: based on number of iterations, learning rate decreases

Iterative Optimization Heuristics

- Many iterative heuristics in algorithms are dynamic and adaptive
  1. the algorithm's behavior changes over time
  2. the algorithm's behavior changes based on internal statistics

- These heuristics might control other hyperparameters of the algorithms
- Example: learning rate schedules for training DNNs
  1. exponential decaying learning rate: based on number of iterations, learning rate decreases
  2. Reduce learning rate on plateaus: if the learning stagnates for some time, the learning rate is decreased by a factor

Iterative Optimization Heuristics

- Many iterative heuristics in algorithms are dynamic and adaptive
  1. the algorithm's behavior changes over time
  2. the algorithm's behavior changes based on internal statistics
- These heuristics might control other hyperparameters of the algorithms
- Example: learning rate schedules for training DNNs
  1. exponential decaying learning rate: based on number of iterations, learning rate decreases
  2. Reduce learning rate on plateaus: if the learning stagnates for some time, the learning rate is decreased by a factor
- other examples: restart probability of search, mutation rate of evolutionary algorithms, ...

Parametrization of Learning Rate Schedules

- How can we parameterize learning rate schedules?
  1. exponential decaying learning rate:
     - initial learning rate
     - minimal learning rate
     - multiplicative factor

Parametrization of Learning Rate Schedules

- How can we parameterize learning rate schedules?
  1. exponential decaying learning rate:
     - ⋆ initial learning rate
     - ⋆ minimal learning rate
     - ⋆ multiplicative factor
  2. Reduce learning rate on plateaus:
     - ⋆ patience (in number of epochs)
     - ⋆ patience threshold
     - ⋆ decreasing factor
     - ⋆ cool-down break (in number of epochs)

Parametrization of Learning Rate Schedules

- How can we parameterize learning rate schedules?
  1. exponential decaying learning rate:
     - ⋆ initial learning rate
     - ⋆ minimal learning rate
     - ⋆ multiplicative factor
  2. Reduce learning rate on plateaus:
     - ⋆ patience (in number of epochs)
     - ⋆ patience threshold
     - ⋆ decreasing factor
     - ⋆ cool-down break (in number of epochs)

$\rightsquigarrow$ Many hyperparameters only to control a single hyperparameter

Parametrization of Learning Rate Schedules

- How can we parameterize learning rate schedules?
  1. exponential decaying learning rate:
     - initial learning rate
     - minimal learning rate
     - multiplicative factor
  2. Reduce learning rate on plateaus:
     - patience (in number of epochs)
     - patience threshold
     - decreasing factor
     - cool-down break (in number of epochs)

$\rightsquigarrow$ Many hyperparameters only to control a single hyperparameter

- Still not guaranteed that optimal setting of e.g. learning rate schedules will lead to optimal learning behavior
  - Learning rate schedules are only heuristics

Dynamic Algorithm Configuration

- So far, we assumed that an algorithm runs with static settings
- However, settings, such as learning rate, have to be adapted over time

Definition

Let

- $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ be a hyperparameter configuration of an algorithm $\mathcal{A}$,

Dynamic Algorithm Configuration

- So far, we assumed that an algorithm runs with static settings
- However, settings, such as learning rate, have to be adapted over time

Definition

Let

- $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ be a hyperparameter configuration of an algorithm $\mathcal{A}$,
- $p(\mathcal{D})$ be a probability distribution over meta datasets $\mathcal{D} \in \mathbf{D}$,

Dynamic Algorithm Configuration

- So far, we assumed that an algorithm runs with static settings
- However, settings, such as learning rate, have to be adapted over time

Definition

Let

- $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ be a hyperparameter configuration of an algorithm $\mathcal{A}$,
- $p(\mathcal{D})$ be a probability distribution over meta datasets $\mathcal{D} \in \mathbf{D}$,
- $s^{(t)}$ be a state description of $\mathcal{A}$ solving $\mathcal{D}$ at time point t,

Dynamic Algorithm Configuration

- So far, we assumed that an algorithm runs with static settings
- However, settings, such as learning rate, have to be adapted over time

Definition

Let

- $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ be a hyperparameter configuration of an algorithm $\mathcal{A}$,
- $p(\mathcal{D})$ be a probability distribution over meta datasets $\mathcal{D} \in \mathbf{D}$,
- $s^{(t)}$ be a state description of $\mathcal{A}$ solving $\mathcal{D}$ at time point t,
- $c : \mathbf{\Pi} \times \mathbf{D} \to \mathbb{R}$ be a cost metric assessing the cost of a control policy $\pi \in \Pi$ on $\mathcal{D} \in \mathbf{D}$

Dynamic Algorithm Configuration

- So far, we assumed that an algorithm runs with static settings
- However, settings, such as learning rate, have to be adapted over time

Definition

Let

- $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ be a hyperparameter configuration of an algorithm $\mathcal{A}$,
- $p(\mathcal{D})$ be a probability distribution over meta datasets $\mathcal{D} \in \mathbf{D}$,
- $s^{(t)}$ be a state description of $\mathcal{A}$ solving $\mathcal{D}$ at time point t,
- $c : \mathbf{\Pi} \times \mathbf{D} \to \mathbb{R}$ be a cost metric assessing the cost of a control policy $\pi \in \Pi$ on $\mathcal{D} \in \mathbf{D}$

the *dynamic algorithm configuration problem* is to obtain a policy $\pi^* : s_t \times \mathcal{D} \mapsto \boldsymbol{\lambda}$ by optimizing its cost across a distribution of datasets:

$$\pi^* \in \arg\min_{\pi \in \mathbf{\Pi}} \int_{\mathbf{D}} p(\mathcal{D}) c(\pi, \mathcal{D}) \, \mathrm{d}\mathcal{D}$$

Dynamic Algorithm Configuration as Contextual MDP [Biedenkapp et al. 2020]

State $s^{(t)}$ are described by statistics gathered in the algorithm run

Dynamic Algorithm Configuration as Contextual MDP [Biedenkapp et al. 2020]

State $s^{(t)}$ are described by statistics gathered in the algorithm run

Action $a^{(t)}$ change hyperparameters according to some control policy π

Dynamic Algorithm Configuration as Contextual MDP [Biedenkapp et al. 2020]

State $s^{(t)}$ are described by statistics gathered in the algorithm run

Action $a^{(t)}$ change hyperparameters according to some control policy π

Transition run the algorithm from state $s^{(t)}$ to $s^{(t+1)}$ for a "short" moment by using the hyperparameters defined by $a^{(t)}$

Dynamic Algorithm Configuration as Contextual MDP [Biedenkapp et al. 2020]

State $s^{(t)}$ are described by statistics gathered in the algorithm run

Action $a^{(t)}$ change hyperparameters according to some control policy π

Transition run the algorithm from state $s^{(t)}$ to $s^{(t+1)}$ for a "short" moment by using the hyperparameters defined by $a^{(t)}$

Reward $r^{(t)}$ Return your current solution quality (or an approximation)

Dynamic Algorithm Configuration as Contextual MDP [Biedenkapp et al. 2020]

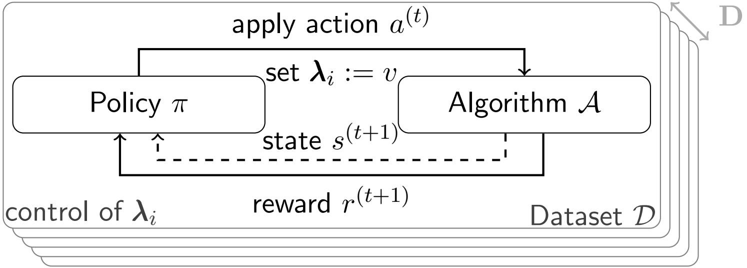
State $s^{(t)}$ are described by statistics gathered in the algorithm run

Action $a^{(t)}$ change hyperparameters according to some control policy π

Transition run the algorithm from state $s^{(t)}$ to $s^{(t+1)}$ for a "short" moment by using the hyperparameters defined by $a^{(t)}$

Reward $r^{(t)}$ Return your current solution quality (or an approximation)

Context $\mathcal{D}$ A given dataset (or task)



Solving Dynamic Algorithm Configuration

Solve unknown MDP by using reinforcement learning (RL):

$$\mathcal{V}_{\mathcal{D}}^{\pi}(s^{(t)}) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{\mathcal{D}}^{(t+k+1)} | s^{(t)} = s\right]$$

Solving Dynamic Algorithm Configuration

Solve unknown MDP by using reinforcement learning (RL):

$$\mathcal{V}^{\pi}_{\mathcal{D}}(s^{(t)}) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{\mathcal{D}}^{(t+k+1)} \middle| s^{(t)} = s\right]$$

$$= \mathbb{E}\left[r_{\mathcal{D}}^{(t+1)} + \gamma \mathcal{V}^{\pi}_{\mathcal{D}}(s^{(t+1)}) \middle| s^{(t+1)} \sim \mathcal{T}_{\mathcal{D}}(s^{(t)}, \pi(s^{(t)}))\right]$$

Solving Dynamic Algorithm Configuration

Solve unknown MDP by using reinforcement learning (RL):

$$\mathcal{V}^{\pi}_{\mathcal{D}}(s^{(t)}) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{\mathcal{D}}^{(t+k+1)} | s^{(t)} = s\right]$$

$$= \mathbb{E}\left[r_{\mathcal{D}}^{(t+1)} + \gamma \mathcal{V}^{\pi}_{\mathcal{D}}(s^{(t+1)}) | s^{(t+1)} \sim \mathcal{T}_{\mathcal{D}}(s^{(t)}, \pi(s^{(t)}))\right]$$

$$\pi^* \in \arg\max_{\pi \in \mathbf{\Pi}} \int_{\mathbf{D}} p(\mathcal{D}) \int_{\mathcal{S}^{(0)}} \Pr(s^{(0)}) \cdot \mathcal{V}^{\pi}_{\mathcal{D}}(s^{(0)}) \, \mathrm{d}s^{(0)} \, \mathrm{d}\mathcal{D}$$

Solving Dynamic Algorithm Configuration

Solve unknown MDP by using reinforcement learning (RL):

$$\mathcal{V}^{\pi}_{\mathcal{D}}(s^{(t)}) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{\mathcal{D}}^{(t+k+1)} | s^{(t)} = s\right]$$

$$= \mathbb{E}\left[r_{\mathcal{D}}^{(t+1)} + \gamma \mathcal{V}^{\pi}_{\mathcal{D}}(s^{(t+1)}) | s^{(t+1)} \sim \mathcal{T}_{\mathcal{D}}(s^{(t)}, \pi(s^{(t)}))\right]$$

$$\pi^* \in \operatorname*{arg\,max}_{\pi \in \mathbf{\Pi}} \int_{\mathbf{D}} p(\mathcal{D}) \int_{\mathcal{S}^{(0)}} \Pr(s^{(0)}) \cdot \mathcal{V}^{\pi}_{\mathcal{D}}(s^{(0)}) \, \mathrm{d}s^{(0)} \, \mathrm{d}\mathcal{D}$$

$\rightsquigarrow$ equivalent to Dynamic Algorithm Configuration definition

Dynamic Algorithm Configuration across Datasets [Biedenkapp et al. 2020]

- Challenge: Evaluating a policies on all datasets is often not feasible

Dynamic Algorithm Configuration across Datasets [Biedenkapp et al. 2020]

- Challenge: Evaluating a policies on all datasets is often not feasible
- Curriculum learning [Bengio et al. 2009] showed that we should have a curriculum of tasks we tackle

Dynamic Algorithm Configuration across Datasets [Biedenkapp et al. 2020]

- Challenge: Evaluating a policies on all datasets is often not feasible
- Curriculum learning [Bengio et al. 2009] showed that we should have a curriculum of tasks we tackle
- Self-paced learning [Kumar et al. 2010] tries to automatically find such as a curriculum
  - Focus on "easy" tasks where the agent can improve most:

Dynamic Algorithm Configuration across Datasets [Biedenkapp et al. 2020]

- Challenge: Evaluating a policies on all datasets is often not feasible
- Curriculum learning [Bengio et al. 2009] showed that we should have a curriculum of tasks we tackle
- Self-paced learning [Kumar et al. 2010] tries to automatically find such as a curriculum
  - Focus on "easy" tasks where the agent can improve most:

$$\max_{\pi, \mathbf{v}} \mathcal{C}(\pi, \mathbf{v}, K) = \sum_{i=1}^{|\mathbf{D}|} \mathbf{v}_i \mathcal{R}_i(\pi) - \frac{1}{K} \sum_{i=1}^{|\mathbf{D}|} \mathbf{v}_i$$

with θ being the agent's policy parameters and
$\mathbf{v}$ being a masking vector for choosing the tasks at hand.

AutoML: Dynamic Configuration & Learning

Learning to Adjust Learning Rates

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Learning Problem [Daniel et al. 2016]

- Optimization of a function:

$$\theta \in \arg\min F(\mathbf{X}; \theta)$$

where $\mathbf{X}$ is an input matrix and f is parameterized by θ.

Learning Problem [Daniel et al. 2016]

- Optimization of a function:

$$\theta \in \arg\min F(\mathbf{X}; \theta)$$

where $\mathbf{X}$ is an input matrix and f is parameterized by θ.

$$F(\mathbf{X}; \theta) = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}^{(i)}; \theta)$$

Learning Step Size Policies [Daniel et al. 2016]

- Idea: Learn the hyperparameters of the weight update (short notation)

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla F(\theta^{(t)})$$

$$\nabla F(\theta^{(t)}) = \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(\theta^{(t)})$$

Learning Step Size Policies [Daniel et al. 2016]

- Idea: Learn the hyperparameters of the weight update (short notation)

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla F(\theta^{(t)})$$
$$\nabla F(\theta^{(t)}) = \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(\theta^{(t)})$$

- For SGD, this would be for example the learning rate α

Learning Step Size Policies [Daniel et al. 2016]

- Idea: Learn the hyperparameters of the weight update (short notation)

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla F(\theta^{(t)})$$
$$\nabla F(\theta^{(t)}) = \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(\theta^{(t)})$$

- For SGD, this would be for example the learning rate α
- Note (i): α have to be adapted in the course of the training
  - similar to learning rate schedules (e.g., cosine annealing)

Learning Step Size Policies [Daniel et al. 2016]

- Idea: Learn the hyperparameters of the weight update (short notation)

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla F(\theta^{(t)})$$
$$\nabla F(\theta^{(t)}) = \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(\theta^{(t)})$$

- For SGD, this would be for example the learning rate α
- Note (i): α have to be adapted in the course of the training
  - similar to learning rate schedules (e.g., cosine annealing)
- Note(ii): later we denote the learnt hyperparameters as λ

Learning Step Size Policies [Daniel et al. 2016]

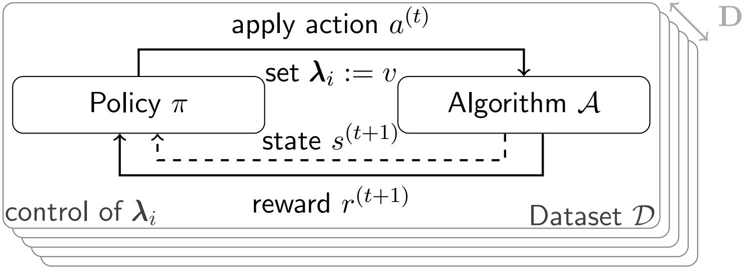
- Idea: Learn the hyperparameters of the weight update (short notation)

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla F(\theta^{(t)})$$
$$\nabla F(\theta^{(t)}) = \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(\theta^{(t)})$$

- For SGD, this would be for example the learning rate α
- Note (i): α have to be adapted in the course of the training
  - similar to learning rate schedules (e.g., cosine annealing)
- Note(ii): later we denote the learnt hyperparameters as λ

- Idea: Use reinforcement learning to learn a policy $\pi : s \mapsto a$ to control the learning rate (or other adaptive hyperparameters)

Recap: Reinforcement Learning for Dynamic Algorithm Configuration



To apply that, we need to define:

1. State description
2. Action space
3. Reward function

RL for Step Size Policies: State [Daniel et al. 2016]

Predictive change in function value:

$$s_1 = \log\left(\mathsf{Var}(\Delta\tilde{f}_i)\right)$$

$$\Delta\tilde{f}_i = \tilde{f}(\mathbf{x}^{(i)};\theta+\delta\theta) - f(\mathbf{x}^{(i)};\theta)$$

where $\tilde{f}(\mathbf{x}^{(i)};\theta+\delta\theta)$ is done by a first order Taylor expansion

RL for Step Size Policies: State [Daniel et al. 2016]

Predictive change in function value:

$$s_1 = \log\left(\mathsf{Var}(\Delta \tilde{f}_i)\right)$$

$$\Delta \tilde{f}_i = \tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta) - f(\mathbf{x}^{(i)}; \theta)$$

where $\tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta)$ is done by a first order Taylor expansion

Disagreement of function values:

$$s_2 = \log\left(\mathsf{Var}(f(\mathbf{x}^{(i)}; \theta))\right)$$

RL for Step Size Policies: State [Daniel et al. 2016]

Predictive change in function value:

$$s_1 = \log\left(\text{Var}(\Delta\tilde{f}_i)\right)$$

$$\Delta\tilde{f}_i = \tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta) - f(\mathbf{x}^{(i)}; \theta)$$

where $\tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta)$ is done by a first order Taylor expansion

Disagreement of function values:

$$s_2 = \log\left(\text{Var}(f(\mathbf{x}^{(i)}; \theta))\right)$$

Discounted Average (smoothing noise from mini-batches):

$$\hat{s}_i \leftarrow \gamma\hat{s}_i + (1-\gamma)s_i$$

RL for Step Size Policies: State [Daniel et al. 2016]

Predictive change in function value:

$$s_1 = \log\left(\mathsf{Var}(\Delta\tilde{f}_i)\right)$$

$$\Delta\tilde{f}_i = \tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta) - f(\mathbf{x}^{(i)}; \theta)$$

where $\tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta)$ is done by a first order Taylor expansion

Disagreement of function values:

$$s_2 = \log\left(\mathsf{Var}(f(\mathbf{x}^{(i)}; \theta))\right)$$

Discounted Average (smoothing noise from mini-batches):

$$\hat{s}_i \leftarrow \gamma\hat{s}_i + (1-\gamma)s_i$$

Uncertainty Estimate (noise level):

$$s_{K+i} \leftarrow \gamma s_{K+i} + (1-\gamma)(s_i - \hat{s}_i)^2$$

RL for Step Size Policies: Learning [Daniel et al. 2016]

Reward (average loss improvement over time):

$$r = \frac{1}{T-1} \sum_{t=2}^{T} \left(\log(L^{(t-1)}) - \log(L^{(t)}) \right)$$

RL for Step Size Policies: Learning [Daniel et al. 2016]

Reward (average loss improvement over time):

$$r = \frac{1}{T-1} \sum_{t=2}^{T} \left(\log(L^{(t-1)}) - \log(L^{(t)}) \right)$$

Optimal Policy:

$$\pi^*(\lambda \mid s) \in \arg\max_{\pi} \int \int p(s) \pi(\boldsymbol{\lambda} \mid s) r(\boldsymbol{\lambda}, s) \, \mathrm{d}s \, \mathrm{d}\boldsymbol{\lambda}$$

RL for Step Size Policies: Learning [Daniel et al. 2016]

Reward (average loss improvement over time):

$$r = \frac{1}{T-1} \sum_{t=2}^{T} \left(\log(L^{(t-1)}) - \log(L^{(t)}) \right)$$

Optimal Policy:

$$\pi^*(\lambda \mid s) \in \arg\max_{\pi} \int \int p(s)\pi(\boldsymbol{\lambda} \mid s) r(\boldsymbol{\lambda}, s) \, \mathrm{d}s \, \mathrm{d}\boldsymbol{\lambda}$$

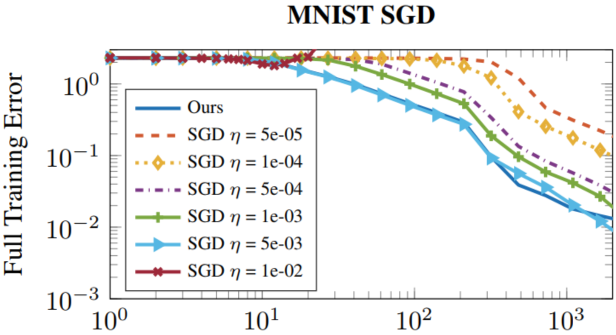
- can be learnt for example via Relative Entropy Policy Search (REPS) [Peter et al. 2010]

RL for Step Size Policies: Training [Daniel et al. 2016]

- Goal: obtain robust policies,
  i.e., good performance for many different DNN architectures
  - $\rightsquigarrow$ Sample architectures e.g., with different numbers of filters and layers
  - $\rightsquigarrow$ (Sub-)Sample dataset
  - $\rightsquigarrow$ Sample number of optimization steps

RL for Step Size Policies: Training [Daniel et al. 2016]

- Goal: obtain robust policies,
  i.e., good performance for many different DNN architectures
  - ⇝ Sample architectures e.g., with different numbers of filters and layers
  - ⇝ (Sub-)Sample dataset
  - ⇝ Sample number of optimization steps



"Ours" refers to the approach by [Daniel et al. 2016] and η is the learning rate

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

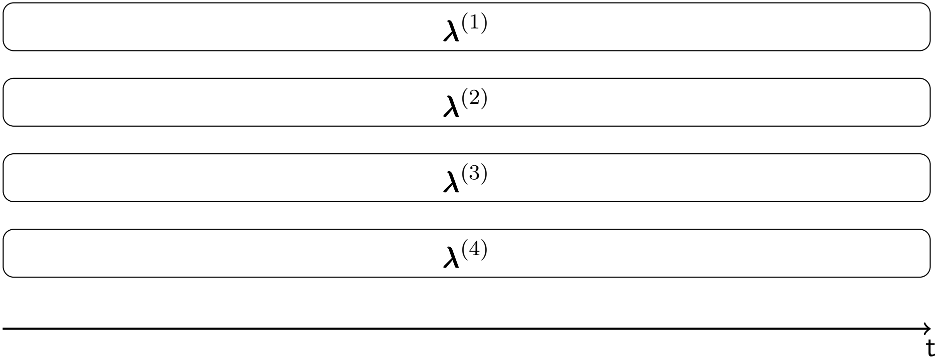
- Dynamic algorithm configuration assumes that we have access to a representative learning environment in an offline learning phase
- What if we don't access to such an env or don't have to time for offline learning?

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?

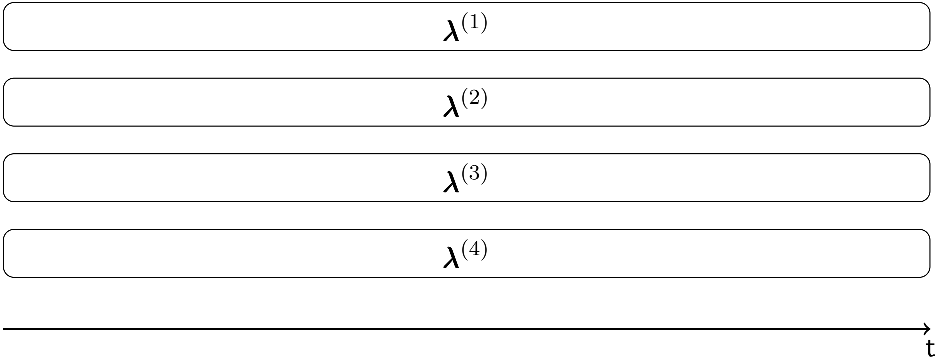
$\rightsquigarrow$ Try to figure out best hyperparameter settings on the fly

Massively parallelized Random Search



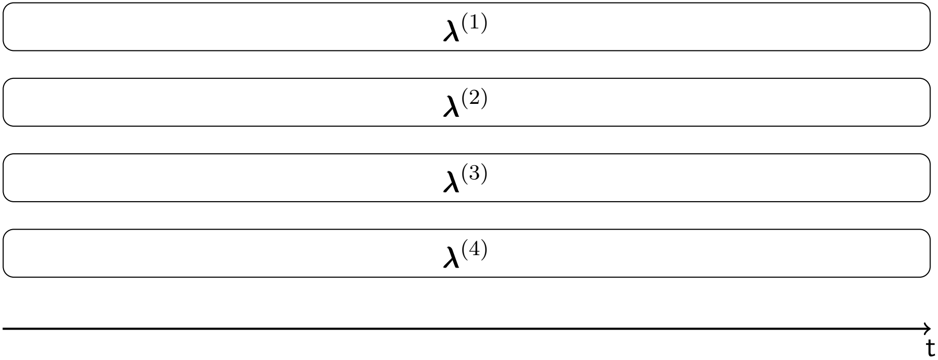
- Sample many hyperparameter configurations $\boldsymbol{\lambda}^{(i)}$ and evaluate them all in parallel

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

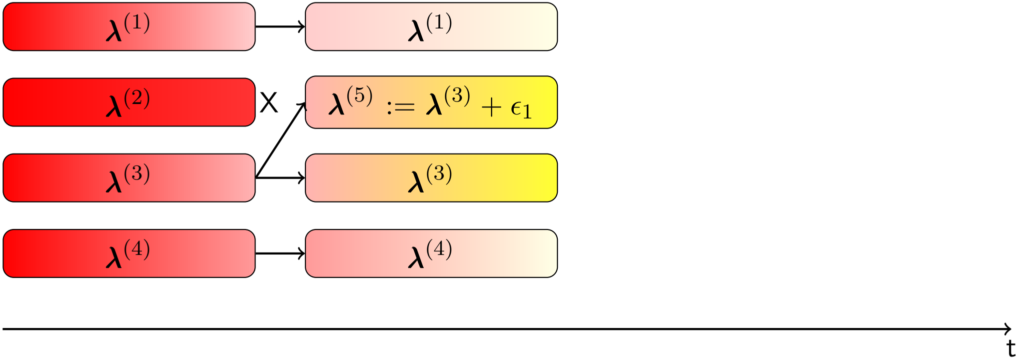


- Sample many hyperparameter configurations $\boldsymbol{\lambda}^{(i)}$ and evaluate them all in parallel
- Pure exploration on a large population of configurations
- No dynamic adaptation

Population-based Training [Jaderberg et al. 2017]

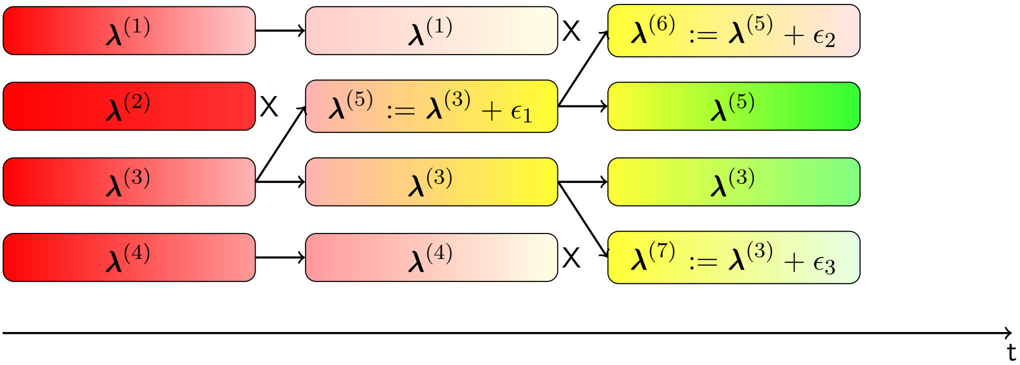
- The color indicates the performance over time

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
  - $\leadsto$ 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

⇝ 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:

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

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

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:

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

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

AutoML: Dynamic Configuration & Learning

Learning to Learn: Supervised

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Learning to Learn

Idea

- Learn algorithms directly, e.g., how to search in the weight space
- First idea: learn weight updates of a neural network

Learning to Learn

Idea

- Learn algorithms directly, e.g., how to search in the weight space
- First idea: learn weight updates of a neural network

Learning to learn by gradient descent by gradient descent

[Andrychowicz et al. 2016]

Weight updates (note: θ denote DNN weights):

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla f(\theta^{(t)})$$

Learning to Learn

Idea

- Learn algorithms directly, e.g., how to search in the weight space
- First idea: learn weight updates of a neural network

Learning to learn by gradient descent by gradient descent

[Andrychowicz et al. 2016]

Weight updates (note: θ denote DNN weights):

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla f(\theta^{(t)})$$

Even more general:

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}(\nabla f(\theta^{(t)}), \phi)$$

where g is the optimizer and ϕ are the parameters of the optimizer g.

Learning to Learn

Idea

- Learn algorithms directly, e.g., how to search in the weight space
- First idea: learn weight updates of a neural network

Learning to learn by gradient descent by gradient descent

[Andrychowicz et al. 2016]

Weight updates (note: θ denote DNN weights):

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla f(\theta^{(t)})$$

Even more general:

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}(\nabla f(\theta^{(t)}), \phi)$$

where g is the optimizer and ϕ are the parameters of the optimizer g.
$\rightsquigarrow$ Goal: Optimize f wrt θ by learning g (resp. ϕ)

Learning to Learn: Objective [Andrychowicz et al. 2016]

$$L(\phi) = \mathbb{E}\left[f(\theta^*(f, \phi))\right]$$

where L is a loss function and $\theta^*(f, \phi)$ are the optimized weights θ^* by using the optimizer parameterized with ϕ on function f.

Learning to Learn: Objective [Andrychowicz et al. 2016]

$$L(\phi) = \mathbb{E}\left[f(\theta^*(f, \phi))\right]$$

where L is a loss function and $\theta^*(f, \phi)$ are the optimized weights θ^* by using the optimizer parameterized with ϕ on function f.

$$L(\phi) = \mathbb{E}\left[\sum_{t=1}^{T} w^{(t)} f(\theta^{(t)})\right]$$

Learning to Learn: Objective [Andrychowicz et al. 2016]

$$L(\phi) = \mathbb{E}\left[f(\theta^*(f,\phi))\right]$$

where L is a loss function and $\theta^*(f,\phi)$ are the optimized weights θ^* by using the optimizer parameterized with ϕ on function f.

$$L(\phi) = \mathbb{E}\left[\sum_{t=1}^{T} w^{(t)} f(\theta^{(t)})\right]$$

where w_t are arbitrary weights associated with each time step and

Learning to Learn: Objective [Andrychowicz et al. 2016]

$$L(\phi) = \mathbb{E}\left[f(\theta^*(f, \phi))\right]$$

where L is a loss function and $\theta^*(f, \phi)$ are the optimized weights θ^* by using the optimizer parameterized with ϕ on function f.

$$L(\phi) = \mathbb{E}\left[\sum_{t=1}^{T} w^{(t)} f(\theta^{(t)})\right]$$

where w_t are arbitrary weights associated with each time step and

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}$$

$$\begin{pmatrix} g^{(t)} \\ h^{(t+1)} \end{pmatrix} = m(\nabla_\theta f(\theta^{(t)}), h^{(t)}, \phi)$$

Learning to Learn: Objective [Andrychowicz et al. 2016]

$$L(\phi) = \mathbb{E}\left[f(\theta^*(f, \phi))\right]$$

where L is a loss function and $\theta^*(f, \phi)$ are the optimized weights θ^* by using the optimizer parameterized with ϕ on function f.

$$L(\phi) = \mathbb{E}\left[\sum_{t=1}^{T} w^{(t)} f(\theta^{(t)})\right]$$

where w_t are arbitrary weights associated with each time step and

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}$$

$$\begin{pmatrix} g^{(t)} \\ h^{(t+1)} \end{pmatrix} = m(\nabla_\theta f(\theta^{(t)}), h^{(t)}, \phi)$$

$\leadsto$ Goal: Learn m via ϕ by using gradient descent by optimizing L

Learning to Learn: Objective [Andrychowicz et al. 2016]

$$L(\phi) = \mathbb{E}\left[f(\theta^*(f,\phi))\right]$$

where L is a loss function and $\theta^*(f,\phi)$ are the optimized weights θ^* by using the optimizer parameterized with ϕ on function f.

$$L(\phi) = \mathbb{E}\left[\sum_{t=1}^{T} w^{(t)} f(\theta^{(t)})\right]$$

where w_t are arbitrary weights associated with each time step and

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}$$

$$\begin{pmatrix} g^{(t)} \\ h^{(t+1)} \end{pmatrix} = m(\nabla_\theta f(\theta^{(t)}), h^{(t)}, \phi)$$

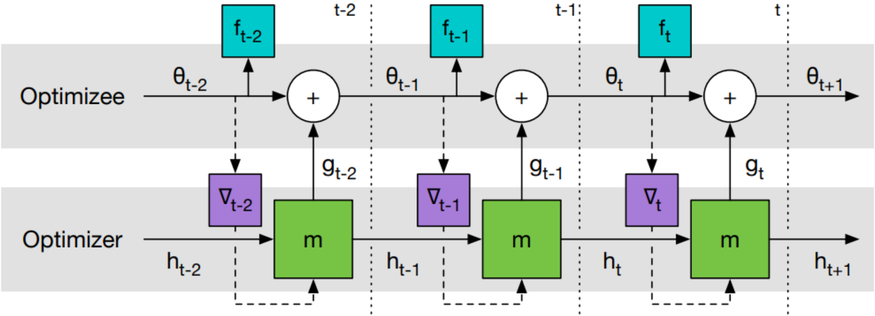
$\rightsquigarrow$ Goal: Learn m via ϕ by using gradient descent by optimizing L

$\rightsquigarrow$ "Learning to learn gradient descent by gradient descent"

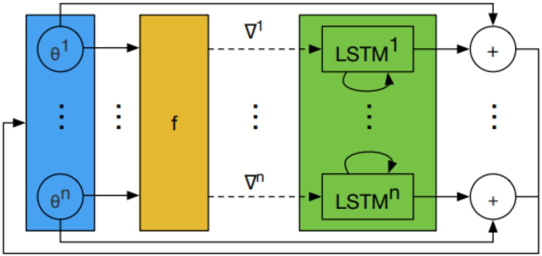
Learning to Learn: LSTM approach [Andrychowicz et al. 2016]

Optimizee Target network to be trained

Optimizer LSTM with hidden state h_t that predicts weight updates g_t

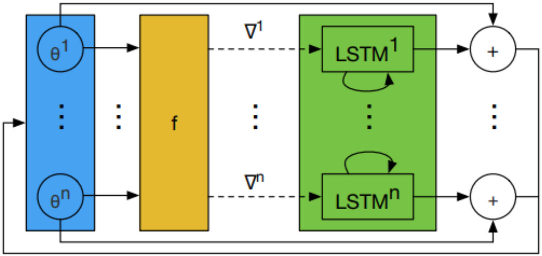


Learning to Learn: Coordinatewise LSTM optimizer [Andrychowicz et al. 2016]



- One LSTM for each coordinate (i.e., weight)
- All LSTMs have shared parameters ϕ
- Each coordinate has its own separate hidden state

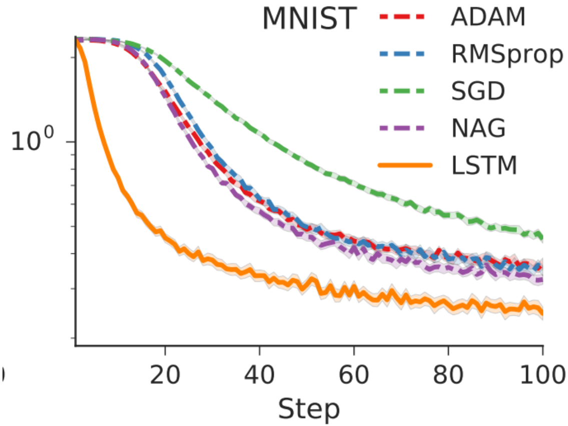
Learning to Learn: Coordinatewise LSTM optimizer [Andrychowicz et al. 2016]



- One LSTM for each coordinate (i.e., weight)
- All LSTMs have shared parameters ϕ
- Each coordinate has its own separate hidden state

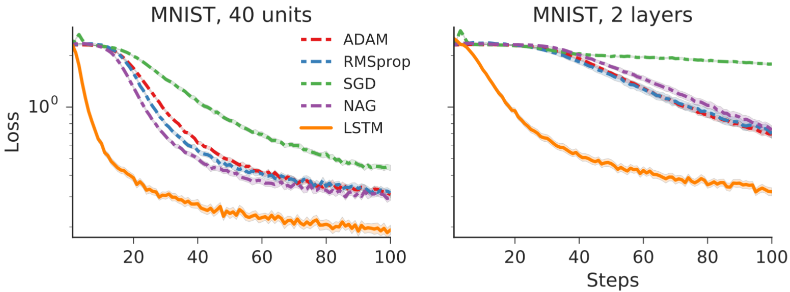
$\leadsto$ We can train the LSTM on k weights and apply it larger DNNs with k' weights, where $k \leq k'$

Learning to Learn with LSTM: Results [Andrychowicz et al. 2016]



Learning to Learn with LSTM: Results [Andrychowicz et al. 2016]

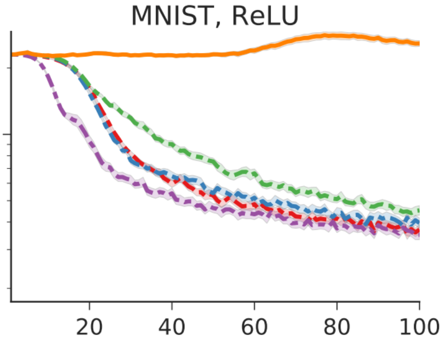
Changing the original architecture of the DNN:



⇝ learnt optimizer is robust against some architectural changes

Learning to Learn with LSTM: Results [Andrychowicz et al. 2016]

Changing the activation function to ReLU:



$\rightsquigarrow$ fails on other activation functions

Learning Black-box Optimization [Chen et al. 2017]

Black Box Optimization Setting

$$\mathbf{x}^* \in \arg\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$$

1. Given the current state of knowledge $h^{(t)}$ propose a query point $\mathbf{x}^{(t)}$
2. Observe the response $y^{(t)}$
3. Update any internal statistics to produce $h^{(t+1)}$

Learning Black-box Optimization [Chen et al. 2017]

Learning Black Box Optimization

Essentially, a similar idea as before:

$$\begin{aligned} h^{(t)}, \mathbf{x}^{(t)} &= \mathsf{RNN}_\phi(h^{(t-1)}, \mathbf{x}^{(t-1)}, y^{(t)}) \\ y^{(t)} &\sim p(y|\mathbf{x}^{(t)}) \end{aligned}$$

- Using recurrent neural network (RNN) to predict next x_t.
- $h^{(t)}$ is the internal hidden state

Learning Black-box Optimization: Loss Functions [Chen et al. 2017]

- Sum loss: Provides more information than final loss

$$L_{\text{sum}}(\phi) = \mathbb{E}_{f, y^{(1:T-1)}} \left[\sum_{t=1}^{T} f(\mathbf{x}^{(t)}) \right]$$

Learning Black-box Optimization: Loss Functions [Chen et al. 2017]

- Sum loss: Provides more information than final loss

$$L_{\text{sum}}(\phi) = \mathbb{E}_{f,y^{(1:T-1)}}\left[\sum_{t=1}^{T} f(\mathbf{x}^{(t)})\right]$$

- EI loss: Try to learn behavior of Bayesian optimizer based on expected improvement (EI)
  - requires model (e.g., GP)

$$L_{\text{EI}}(\phi) = -\mathbb{E}_{f,y^{(1:T-1)}}\left[\sum_{t=1}^{T} \text{EI}(\mathbf{x}^{(t)}|y^{(1:t-1)})\right]$$

Learning Black-box Optimization: Loss Functions [Chen et al. 2017]

- Sum loss: Provides more information than final loss

$$L_{\mathsf{sum}}(\phi) = \mathbb{E}_{f,y^{(1:T-1)}}\left[\sum_{t=1}^{T} f(\mathbf{x}^{(t)})\right]$$

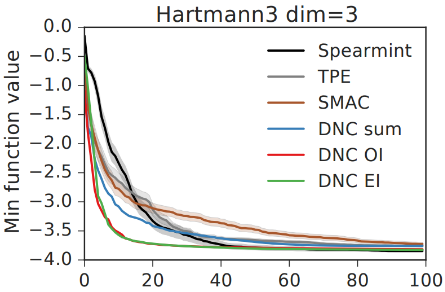
- EI loss: Try to learn behavior of Bayesian optimizer based on expected improvement (EI)
  - requires model (e.g., GP)

$$L_{\mathsf{EI}}(\phi) = -\mathbb{E}_{f,y^{(1:T-1)}}\left[\sum_{t=1}^{T} \mathsf{EI}(\mathbf{x}^{(t)}|y^{(1:t-1)})\right]$$

- Observed Improvement Loss:

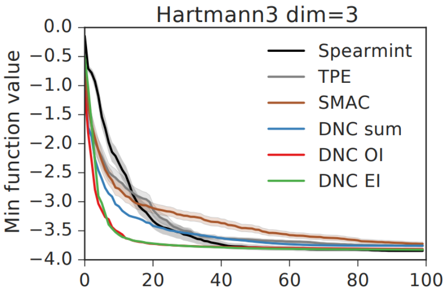
$$L_{\mathsf{OI}}(\phi) = \mathbb{E}_{f,y^{(1:T-1)}}\left[\sum_{t=1}^{T} \min\left\{f(\mathbf{x}^{(t)}) - \min_{i<t}(f(\mathbf{x}^{(i)})), 0\right\}\right]$$

Learning Black-box Optimization: Results [Chen et al. 2017]



- Hartmann3 is an artificial function with 3 dimensions

Learning Black-box Optimization: Results [Chen et al. 2017]



- Hartmann3 is an artificial function with 3 dimensions

⇝ L_{OI} and L_{EI} perform best

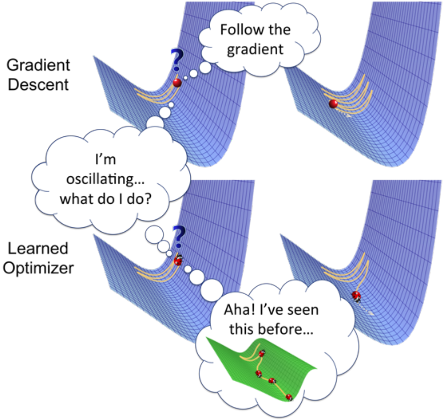
⇝ L_{OI} easier to compute than L_{EI}
because we need a predictive model to compute EI

AutoML: Dynamic Configuration & Learning

Learning to Learn: Reinforcement Learning

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Learning to Optimize via Reinforcement Learning [Li and Malik. 2017]



Source: `https://bair.berkeley.edu/blog/2017/09/12/learning-to-optimize-with-rl/`

Learning to Optimize via Reinforcement Learning [Li and Malik. 2017]

Reinforcement Learning for Learning to Optimize

State current location, objective values and gradients evaluated at the current and past locations

Learning to Optimize via Reinforcement Learning [Li and Malik. 2017]

Reinforcement Learning for Learning to Optimize

State current location, objective values and gradients evaluated at the current and past locations

Action Step update $\Delta \mathbf{x}$

Learning to Optimize via Reinforcement Learning [Li and Malik. 2017]

Reinforcement Learning for Learning to Optimize

State current location, objective values and gradients evaluated at the current and past locations

Action Step update $\Delta\mathbf{x}$

Transition $\mathbf{x}^{(t)} \leftarrow \mathbf{x}^{(t-1)} + \Delta\mathbf{x}$

Learning to Optimize via Reinforcement Learning [Li and Malik. 2017]

Reinforcement Learning for Learning to Optimize

State current location, objective values and gradients evaluated at the current and past locations

Action Step update $\Delta \mathbf{x}$

Transition $\mathbf{x}^{(t)} \leftarrow \mathbf{x}^{(t-1)} + \Delta \mathbf{x}$

Cost/Reward Objective value at the current location

- Since the RL agent will optimize the cumulative cost, this is equivalent to L_{sum} [Chen et al. 2017] $(\gamma = 0)$
- encourages the policy to reach the minimum of the objective function as quickly as possible

Learning to Optimize via Reinforcement Learning [Li and Malik. 2017]

Reinforcement Learning for Learning to Optimize

State current location, objective values and gradients evaluated at the current and past locations

Action Step update $\Delta\mathbf{x}$

Transition $\mathbf{x}^{(t)} \leftarrow \mathbf{x}^{(t-1)} + \Delta\mathbf{x}$

Cost/Reward Objective value at the current location

- Since the RL agent will optimize the cumulative cost, this is equivalent to L_{sum} [Chen et al. 2017] $(\gamma = 0)$
- encourages the policy to reach the minimum of the objective function as quickly as possible

Policy DNN predicting μ_d of Gaussian (with constant variance σ^2) for dimension d; sample $\Delta\mathbf{x}_d \sim \mathcal{N}(\mu_d, \sigma^2)$

Learning to Optimize via Reinforcement Learning [Li and Malik. 2017]

Reinforcement Learning for Learning to Optimize

State current location, objective values and gradients evaluated at the current and past locations

Action Step update $\Delta\mathbf{x}$

Transition $\mathbf{x}^{(t)} \leftarrow \mathbf{x}^{(t-1)} + \Delta\mathbf{x}$

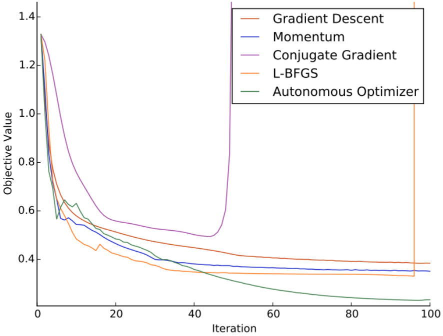
Cost/Reward Objective value at the current location

- Since the RL agent will optimize the cumulative cost, this is equivalent to L_{sum} [Chen et al. 2017] $(\gamma = 0)$
- encourages the policy to reach the minimum of the objective function as quickly as possible

Policy DNN predicting μ_d of Gaussian (with constant variance σ^2) for dimension d; sample $\Delta\mathbf{x}_d \sim \mathcal{N}(\mu_d, \sigma^2)$

Training Set randomly generated objective functions

Learning to Optimize via Reinforcement Learning Results [Li and Malik. 2017]



- 2-layer DNN with ReLUs
- Training datasets for training RL agent:
  four multivariate Gaussians and sampling 25 points from each
  - $\rightsquigarrow$ hard toy problem

Learning Acquisition Functions [Volpp et al. 2019]

- Instead of learning everything, it might be sufficient to learn hand-design heuristics

Learning Acquisition Functions [Volpp et al. 2019]

- Instead of learning everything, it might be sufficient to learn hand-design heuristics
- In Bayesian Optimization (BO), the most critical hand-design heuristic is the acquisition function
  - trade-off between exploitation and exploration
  - Depending on the problem at hand, you might need a different acquisition function

Learning Acquisition Functions [Volpp et al. 2019]

- Instead of learning everything, it might be sufficient to learn hand-design heuristics
- In Bayesian Optimization (BO), the most critical hand-design heuristic is the acquisition function
  - trade-off between exploitation and exploration
  - Depending on the problem at hand, you might need a different acquisition function
  - Choices:
    - probability of improvement (PI)
    - expected improvement (EI)
    - upper confidence bounds (UCB)
    - entropy search (ES) – quite expensive!
    - knowledge gradient (KG)
    - ...

Learning Acquisition Functions [Volpp et al. 2019]

- Instead of learning everything, it might be sufficient to learn hand-design heuristics
- In Bayesian Optimization (BO), the most critical hand-design heuristic is the acquisition function
  - trade-off between exploitation and exploration
  - Depending on the problem at hand, you might need a different acquisition function
  - Choices:
    - probability of improvement (PI)
    - expected improvement (EI)
    - upper confidence bounds (UCB)
    - entropy search (ES) – quite expensive!
    - knowledge gradient (KG)
    - ...
- Idea: Learn a *neural acquisition function* from data

$\rightsquigarrow$ Replace acquisition function

Bayesian Optimization: Algorithm

Algorithm 1 Bayesian Optimization (BO)

Input : Search Space $\mathcal{X}$, black box function f, acquisition function α, maximal number of function evaluations T

$\mathcal{D}^{(0)} \leftarrow$ initial_design($\mathcal{X}$);
for $t = 1, 2, \ldots T - |D_0|$ **do**
    $\hat{c} : \mathbf{x} \mapsto c(\mathbf{x}) \leftarrow$ fit predictive model on $\mathcal{D}^{(t-1)}$;
    select $\mathbf{x}^{(t)}$ by optimizing $\mathbf{x}^{(t)} \in \arg\max_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x}; \mathcal{D}^{(t-1)}, \hat{c})$;
    Query $y^{(t)} := f(\mathbf{x}^{(t)})$;
    Add observation to data $D^{(t)} := D^{(t-1)} \cup \{\langle \mathbf{x}^{(t)}, y^{(t)} \rangle\}$;
return *Best* $\mathbf{x}$ *according to* D *or* $\hat{c}$

Neural Acquisition Function [Volpp et al. 2019]

Although the acquisition function α depends on the history $\mathcal{D}^{(t-1)}$ and the predictive model $\hat{c}$, α mainly makes use of the predictive mean μ and variance σ^2.

Neural Acquisition Function [Volpp et al. 2019]

Although the acquisition function α depends on the history $\mathcal{D}^{(t-1)}$ and the predictive model $\hat{c}$, α mainly makes use of the predictive mean μ and variance σ^2.

Neural acquisition function (AF):

$$\alpha_\theta(\mathbf{x}) = \alpha_\theta(\mu^{(t)}(\mathbf{x}), \sigma^{(t)}(\mathbf{x}), \mathbf{x}, t, T)$$

where θ are the parameters of a neural network, and μ, σ, $\mathbf{x}$, t, T are its inputs.

RL to train Neural AF [Volpp et al. 2019]

Policy π_θ: Neural acquisition function α_θ

RL to train Neural AF [Volpp et al. 2019]

Policy π_θ: Neural acquisition function α_θ

Episode: run of π on $f \in \mathcal{F}'$

- $\mathcal{F}$ is a set of functions we can sample functions from

RL to train Neural AF [Volpp et al. 2019]

Policy π_θ: Neural acquisition function α_θ

Episode: run of π on $f \in \mathcal{F}'$

- $\mathcal{F}$ is a set of functions we can sample functions from

State $s^{(t)}$: $\mu^{(t)}$ and $\sigma^{(t)}$ on a set of points $\xi^{(t)}$, and t and T

RL to train Neural AF [Volpp et al. 2019]

Policy π_θ: Neural acquisition function α_θ

Episode: run of π on $f \in \mathcal{F}'$

- $\mathcal{F}$ is a set of functions we can sample functions from

State $s^{(t)}$: $\mu^{(t)}$ and $\sigma^{(t)}$ on a set of points $\xi^{(t)}$, and t and T

Action $a^{(t)}$: Sampled point $\mathbf{x}^{(t)} \in \xi^{(t)}$

RL to train Neural AF [Volpp et al. 2019]

Policy π_θ: Neural acquisition function α_θ

Episode: run of π on $f \in \mathcal{F}'$

- $\mathcal{F}$ is a set of functions we can sample functions from

State $s^{(t)}$: $\mu^{(t)}$ and $\sigma^{(t)}$ on a set of points $\xi^{(t)}$, and t and T

Action $a^{(t)}$: Sampled point $\mathbf{x}^{(t)} \in \xi^{(t)}$

Reward $r^{(t)}$: negative simple regret: $r^{(t)} = f(\mathbf{x}^*) - f(\hat{\mathbf{x}})$

- assumes that we can estimate the optimal $\mathbf{x}^*$ for *training* functions

RL to train Neural AF [Volpp et al. 2019]

Policy π_θ: Neural acquisition function α_θ

Episode: run of π on $f \in \mathcal{F}'$

- $\mathcal{F}$ is a set of functions we can sample functions from

State $s^{(t)}$: $\mu^{(t)}$ and $\sigma^{(t)}$ on a set of points $\xi^{(t)}$, and t and T

Action $a^{(t)}$: Sampled point $\mathbf{x}^{(t)} \in \xi^{(t)}$

Reward $r^{(t)}$: negative simple regret: $r^{(t)} = f(\mathbf{x}^*) - f(\hat{\mathbf{x}})$

- assumes that we can estimate the optimal $\mathbf{x}^*$ for *training* functions

Transition probability : Noisy evaluation of f and the predictive model update

Policy [Volpp et al. 2019]

- The state is described by a discrete set of points $\xi^{(t)} = \{\xi_n\}_{n=1}^{N}$

Policy [Volpp et al. 2019]

- The state is described by a discrete set of points $\xi^{(t)} = \{\xi_n\}_{n=1}^{N}$
- We feed these points through the predictive model and the neural AF to obtain $\alpha_\theta(\xi_n) = \alpha_\theta(\mu^{(t)}(\xi_n), \sigma^{(t)}(\xi_n), \xi_n, t, T)$,

Policy [Volpp et al. 2019]

- The state is described by a discrete set of points $\xi^{(t)} = \{\xi_n\}_{n=1}^N$
- We feed these points through the predictive model and the neural AF to obtain $\alpha_\theta(\xi_n) = \alpha_\theta(\mu^{(t)}(\xi_n), \sigma^{(t)}(\xi_n), \xi_n, t, T)$,
- $\alpha_\theta(\xi_i)$ are interpreted as the logits of categorical distribution, s.t.

$$\pi_\alpha(\cdot \mid s^{(t)}) = \mathsf{Cat}\left[\alpha_\theta(\xi_1), \ldots, \alpha_\theta(\xi_N)\right]$$

Policy [Volpp et al. 2019]

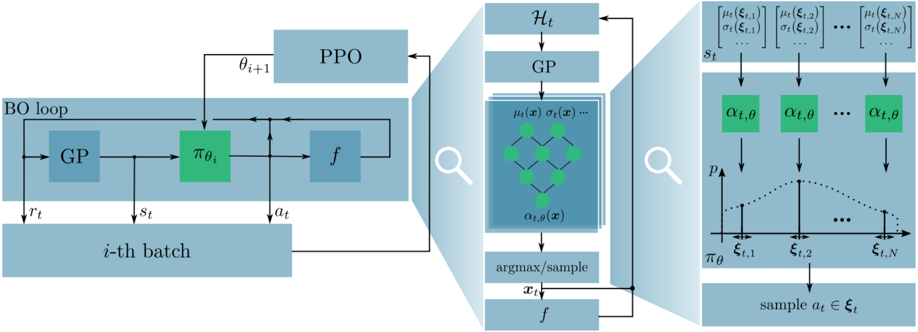
- The state is described by a discrete set of points $\xi^{(t)} = \{\xi_n\}_{n=1}^{N}$
- We feed these points through the predictive model and the neural AF to obtain $\alpha_\theta(\xi_n) = \alpha_\theta(\mu^{(t)}(\xi_n), \sigma^{(t)}(\xi_n), \xi_n, t, T)$,
- $\alpha_\theta(\xi_i)$ are interpreted as the logits of categorical distribution, s.t.

$$\pi_\alpha(\cdot \mid s^{(t)}) = \text{Cat}\left[\alpha_\theta(\xi_1), \ldots, \alpha_\theta(\xi_N)\right]$$

- Due to curse of dimensionality, we need a two step approach for $\xi^{(t)}$
  1. sample ξ_{global} using a coarse Sobol grid
  2. sample ξ_{local} using local optimization starting from the best samples in ξ_{global}

$\rightsquigarrow$ $\xi^{(t)} = \xi_{\text{global}} \cup \xi_{\text{local}}$

Learning Acquisition Functions: Overview [Volpp et al. 2019]

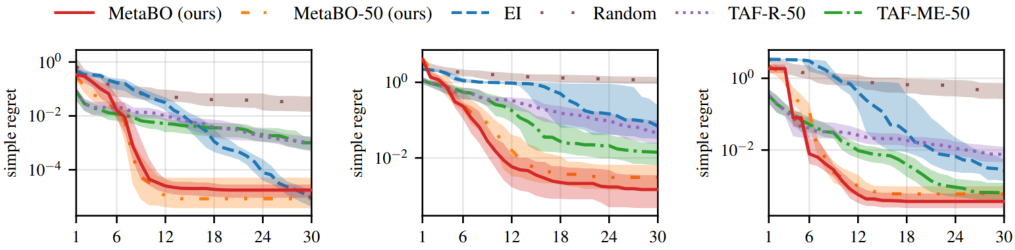


MetaBO Training Loop

Neural AF in the BO loop

Policy architecture

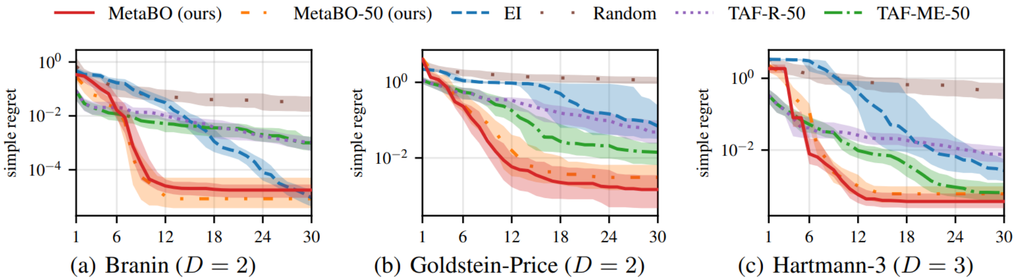
Results on Artificial Functions [Volpp et al. 2019]



(a) Branin ($D = 2$) (b) Goldstein-Price ($D = 2$) (c) Hartmann-3 ($D = 3$)

- Approach by [Volpp et al. 2019] called MetaBO
- MetaBO performs better than other acquisition functions (EI, GP-UCB, PI) and other baselines (Random, TAF)

Results on Artificial Functions [Volpp et al. 2019]



- Approach by [Volpp et al. 2019] called MetaBO
- MetaBO performs better than other acquisition functions (EI, GP-UCB, PI) and other baselines (Random, TAF)

Assumption: You have a family of functions at hand that resembles your target function.