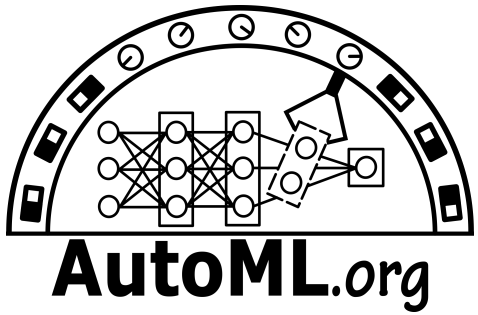
Automated Machine Learning
(AutoML)

M. Lindauer F. Hutter

University of Freiburg



Lecture 8:
Neural Architecture Search (Part 2) and Multi-fidelity Optimization

Where are we? The big picture

- Introduction
- Background
  - Design spaces in ML
  - Evaluation and visualization
- Hyperparameter optimization (HPO)
  - Bayesian optimization
  - Other black-box techniques
  - More details on Gaussian processes
- Pentecost (Holiday) – no lecture

→ Architecture search I + II & Multi-fidelity Optimization

- Meta-Learning
- Learning to learn & optimize
- Beyond AutoML: algorithm configuration and control
- Project announcement and closing

Reminder: Efficient NAS by faster performance estimation

Speed-up method	How are speed-ups achieved?	References
Lower fidelity estimates	Training time reduced by training for fewer epochs, on subset of data, downscaled models, downscaled data, ...	Li et al. (2017), Zoph et al. (2018), Zela et al. (2018), Falkner et al. (2018), Real et al. (2019), Runge et al. (2019)
Learning Curve Extrapolation	Training time reduced as performance can be extrapolated after just a few epochs of training.	Swersky et al. (2014), Domhan et al. (2015), Klein et al. (2017a), Baker et al. (2017b)
Weight Inheritance/ Network Morphisms	Instead of training models from scratch, they are warm-started by inheriting weights of, e.g., a parent model.	Real et al. (2017), Elsken et al. (2017), Elsken et al. (2019), Cai et al. (2018a.b)
One-Shot Models/ Weight Sharing	Only the one-shot model needs to be trained; its weights are then shared across different architectures that are just subgraphs of the one-shot model.	Saxena and Verbeek (2016), Pham et al. (2018), Bender et al. (2018), Liu et al. (2019b), Cai et al. (2019), Xie et al. (2019)

Learning Goals

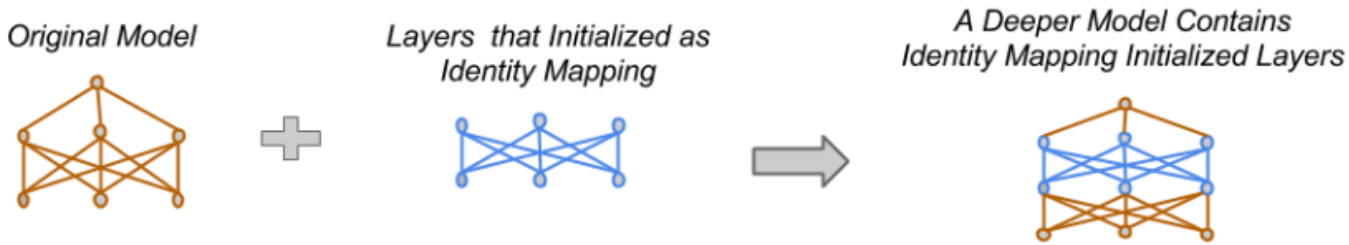
After this lecture, you will be able to . . .

- describe several ways of speeding up over blackbox NAS
  - define network morphisms & explain how to use them to speed up NAS
  - explain various methods for extrapolating learning curves
  - explain various multi-fidelity Bayesian optimization methods
- discuss when and how to use NAS and HPO in practice
  - describe failure modes of DARTS
  - describe Auto-DispNet
- describe NAS-Bench-101, a benchmark for NAS

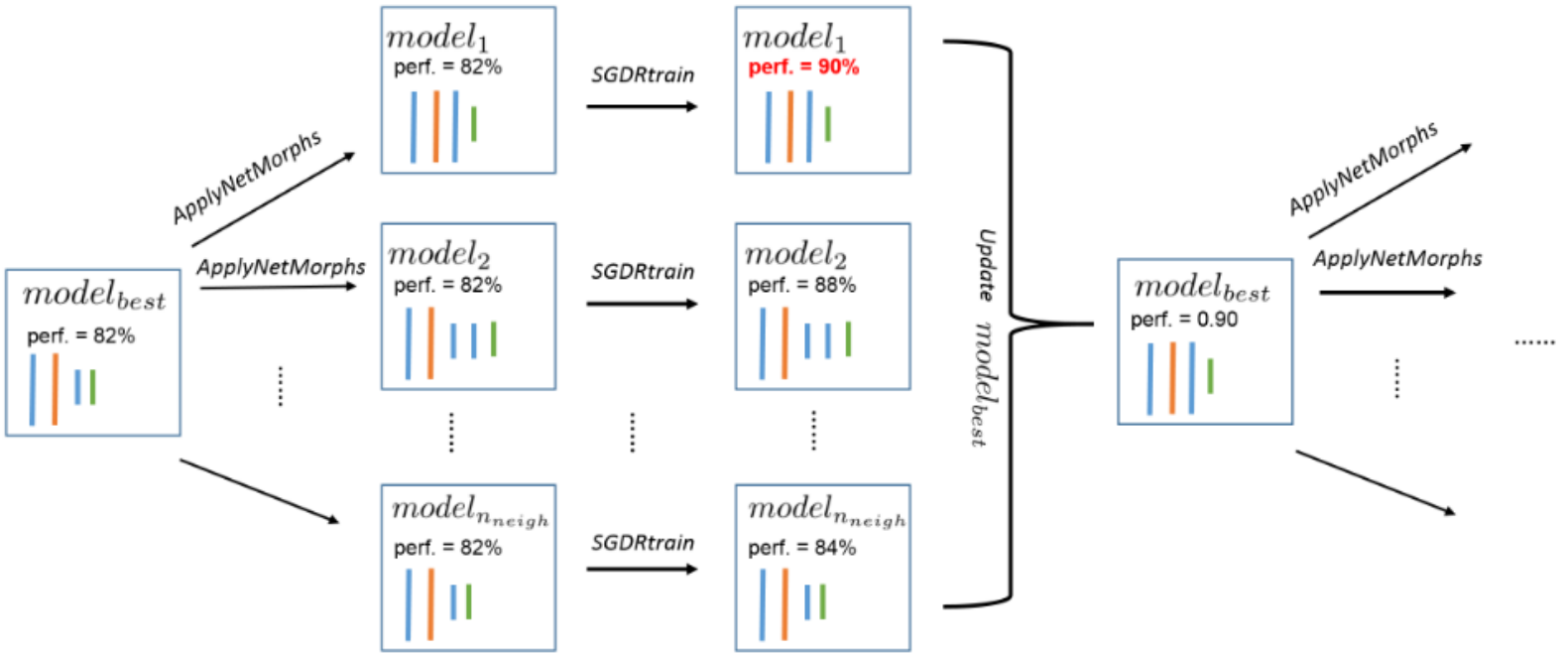
Lecture Overview

Network Morphisms

- Network Morphisms [Chen et al. '16; Wei et al. '16; Cai et al. '17]
  - Change the network structure, but not the modelled function
  - i.e., for every input the network yields the same output as before applying the network morphisms
  - Example operations: "Net2DeeperNet", "Net2WiderNet", etc.



Network Morphisms Allow Efficient Moves in Architecture Space

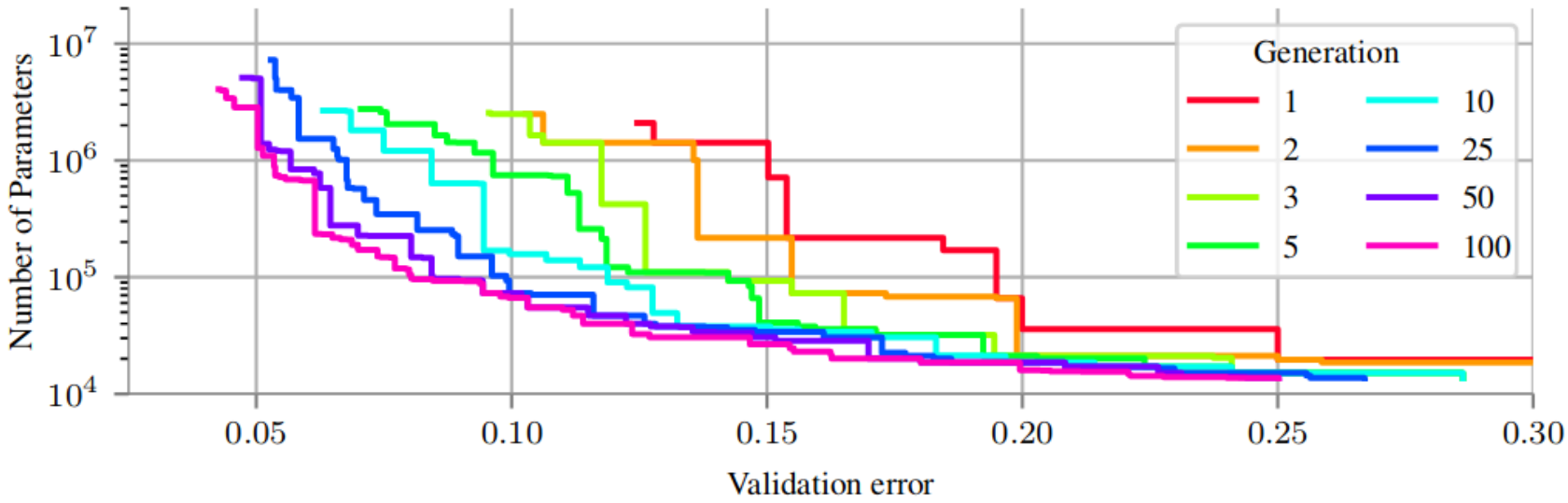


[Real et al. '17; Cai et al, '17; Elsken et al, '17; Cai et al. '18; Elsken et al. '19]

Efficient Multi-objective NAS [Elsken et al. '19]

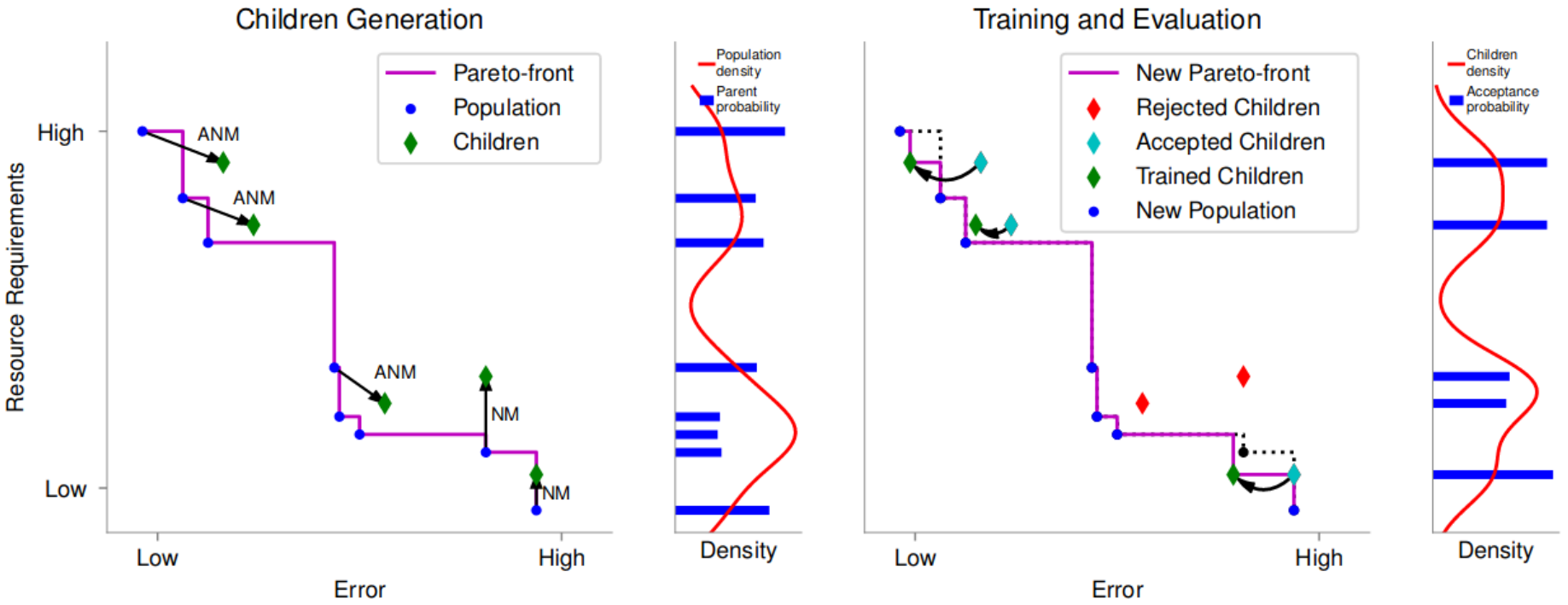
LEMONADE: Lamarckian Evolution for Multi-Objective Neural Arch. Design

- Maintain a Pareto front of the 2 or more objectives
  - Evolve a population of Pareto-optimal architectures over time
- Use weight inheritance
  - Inherit already trained weights from parent architectures to child ones
  - Allow approx. network morphisms (function not preserved perfectly)
  - Still cheap through weight inheritance (e.g., 1 week on 8 GPUs)



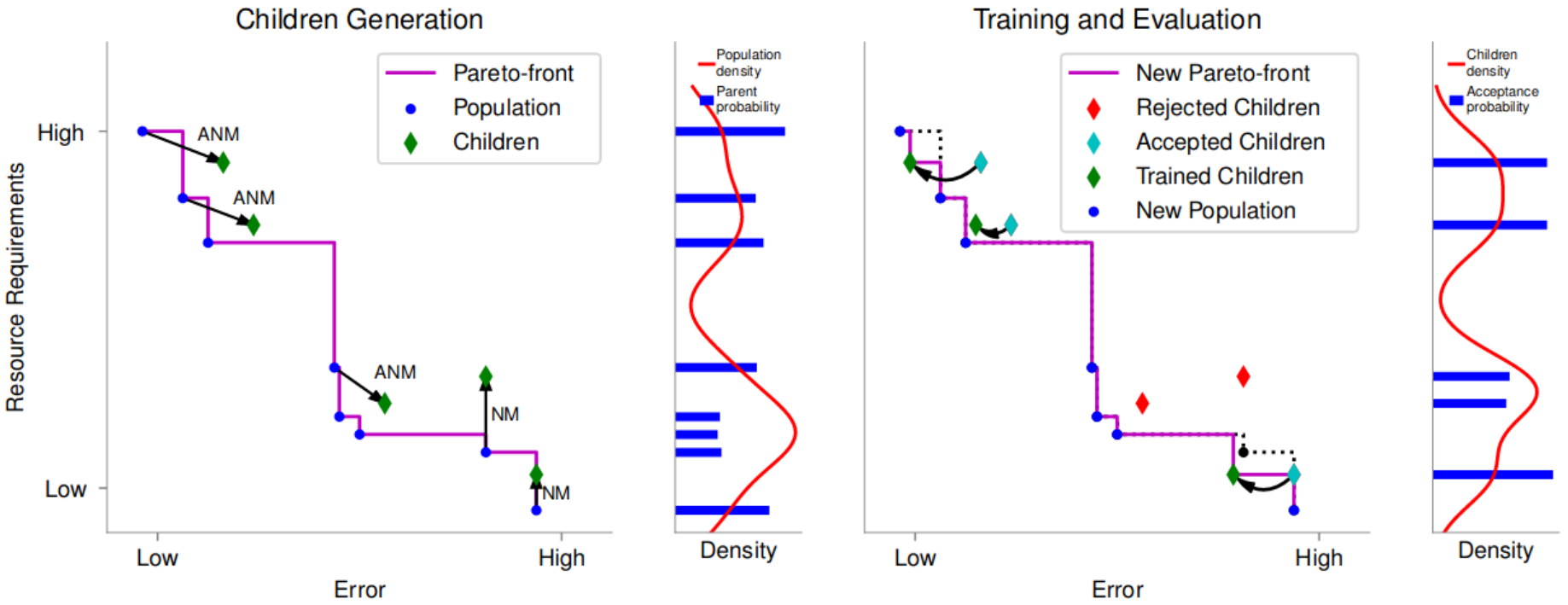
Efficient Multi-objective NAS [Elsken et al. '19]

- Children generation and training/evaluation of architectures



Efficient Multi-objective NAS [Elsken et al. '19]

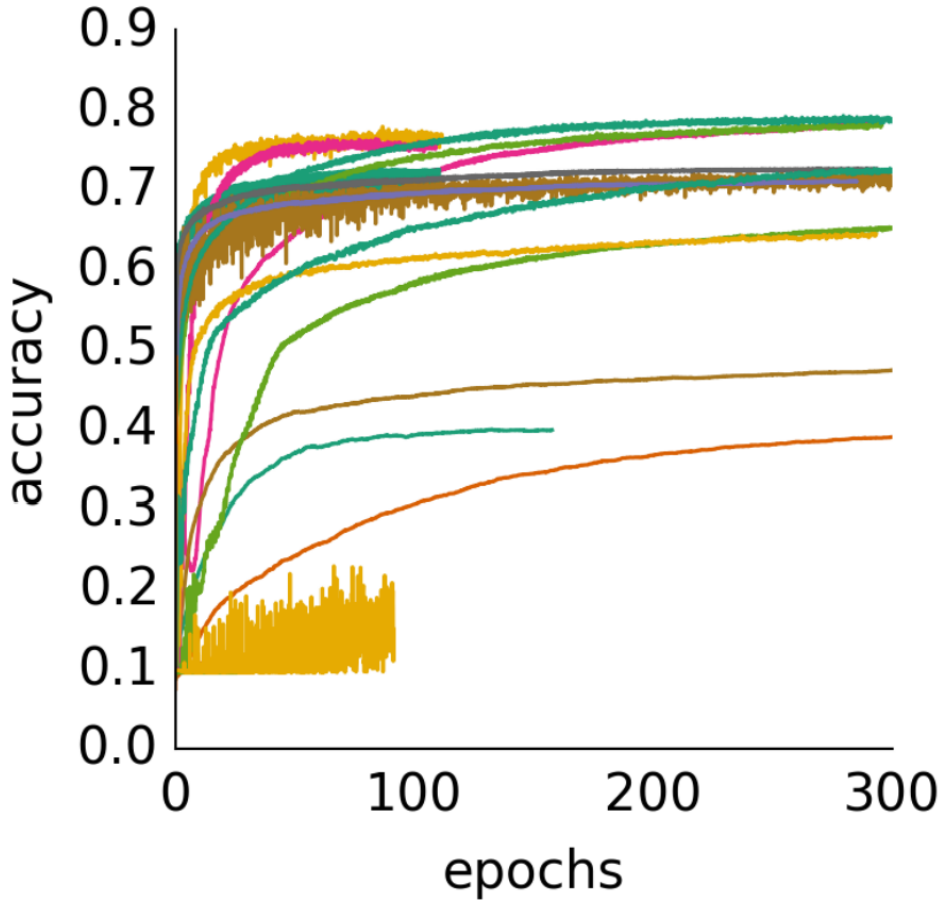
- Children generation and training/evaluation of architectures



- Possible project: apply LEMONADE to SeizureNet
- Possible project: apply LEMONADE to algorithmic fairness
- Possible project: apply LEMONADE to adversarial robustness

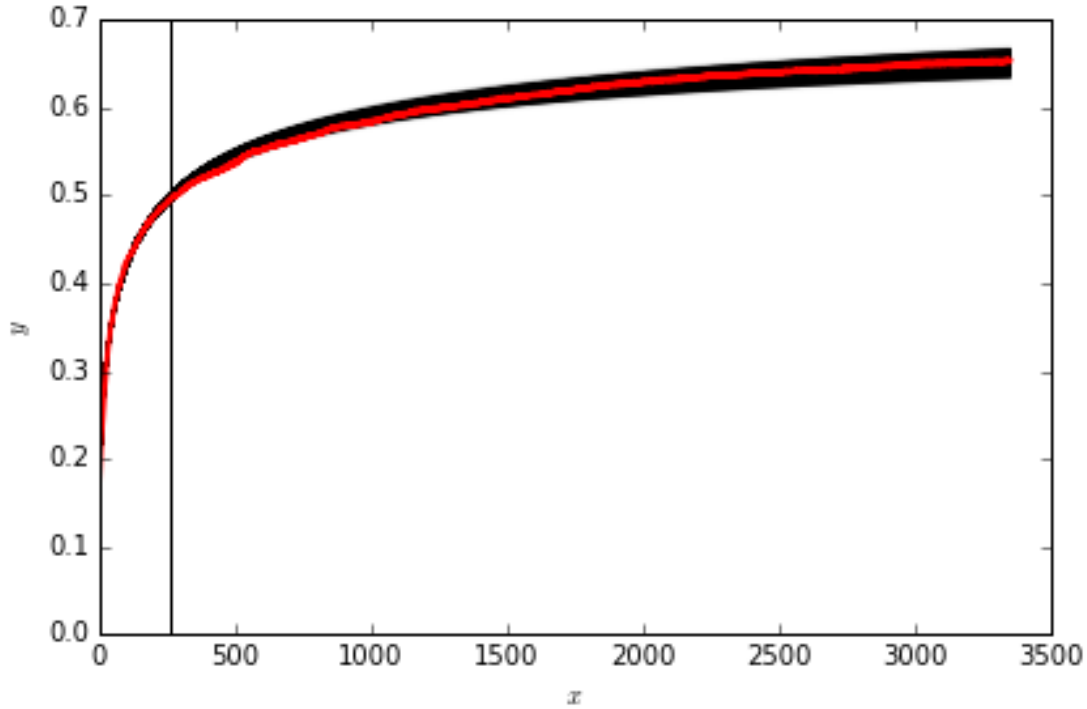
Lecture Overview

Learning Curves



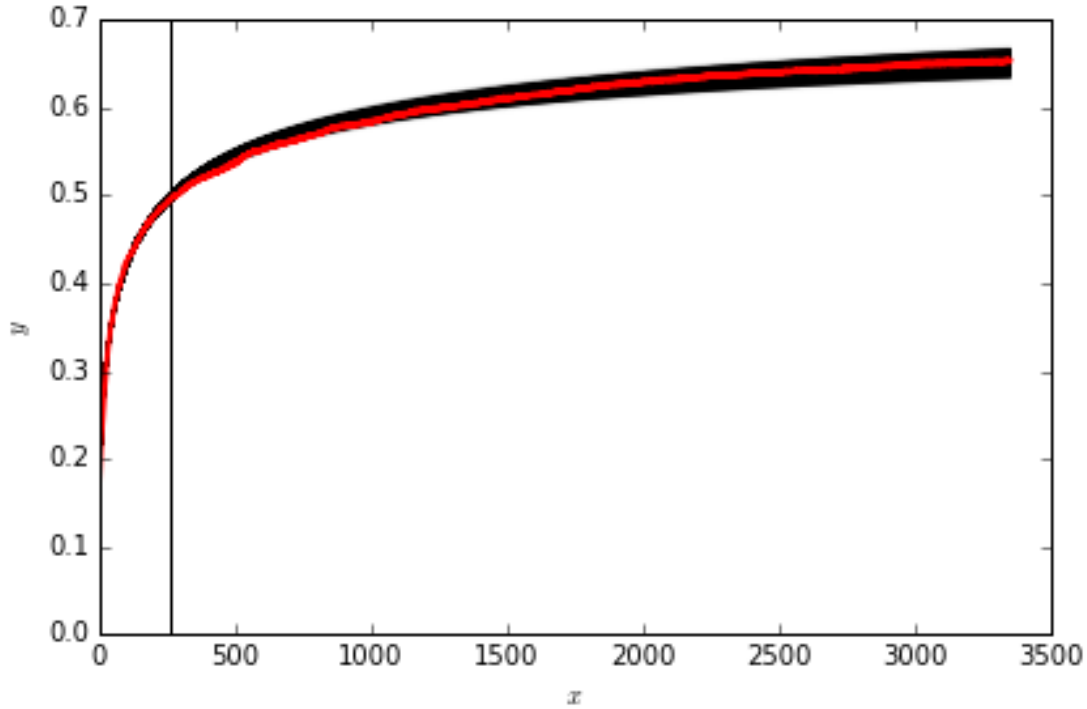
Exemplary learning curves of training deep neural networks
Many ML algorithms iteratively optimize a (loss) function

Learning Curve Predictions



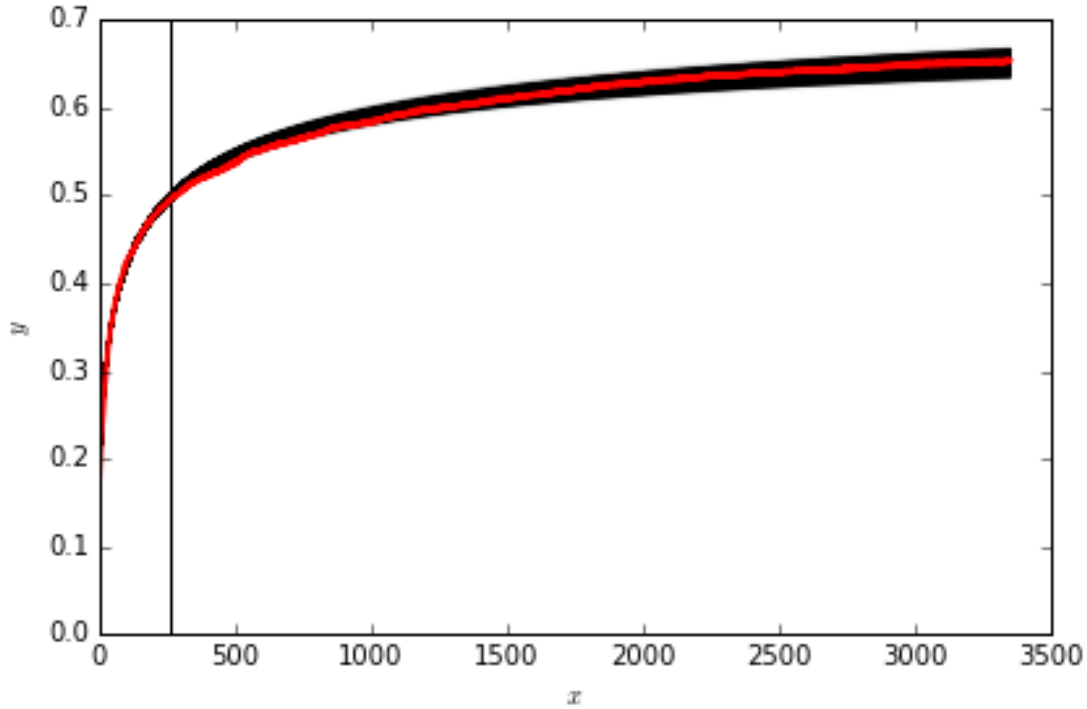
1. Observe learning curve for the first n steps (here $n = 250$)

Learning Curve Predictions



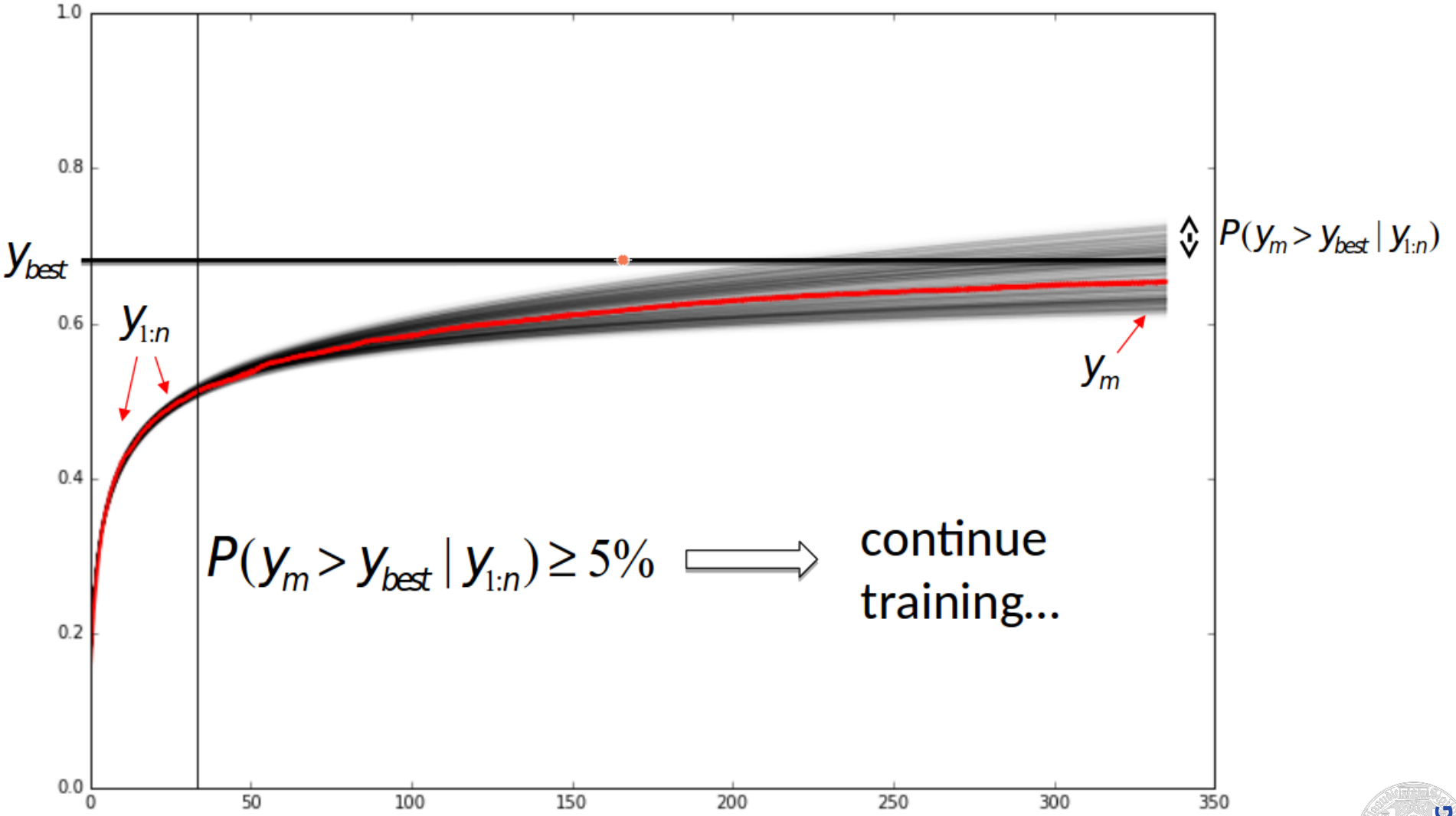
1. Observe learning curve for the first n steps (here $n = 250$)
2. Extrapolation: fit parametric model on partial learning curve to predict remaining learning curve

Learning Curve Predictions



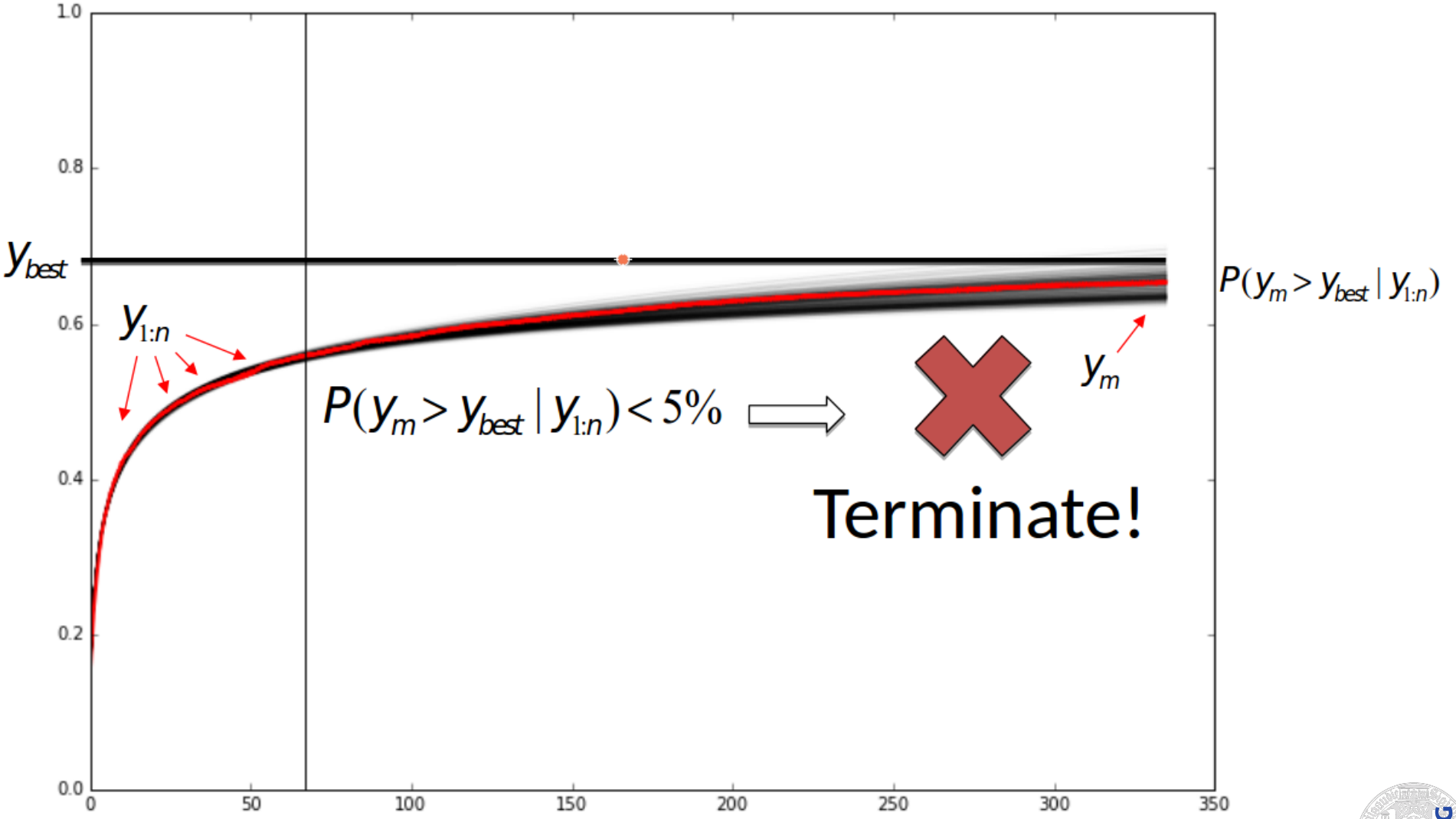
1. Observe learning curve for the first n steps (here $n = 250$)
2. Extrapolation: fit parametric model on partial learning curve to predict remaining learning curve
3. Various models can be used (see following slides)

Learning Curves: Early Termination



$\rightarrow$ need for probabilistic predictions / quantification of uncertainty

Learning Curves: Early Termination



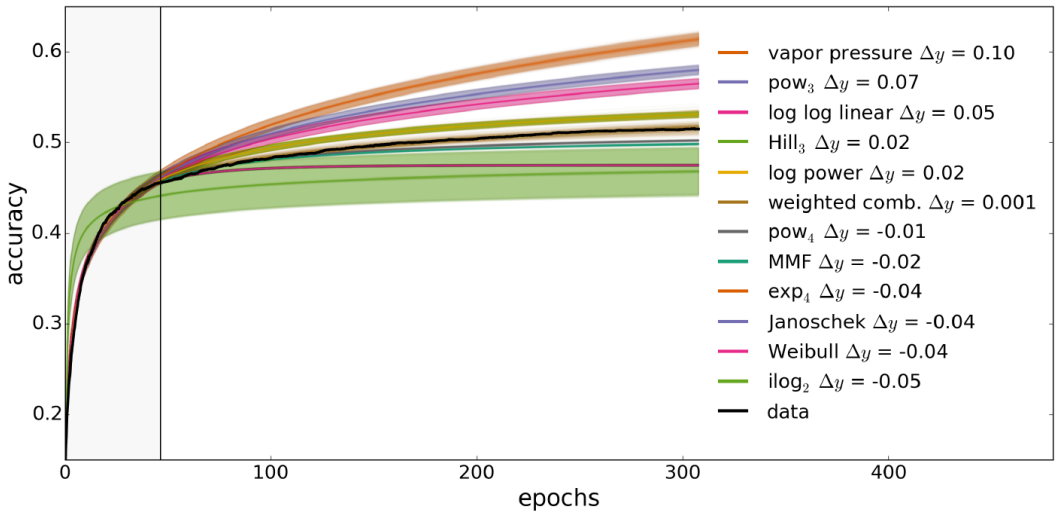
$\rightarrow$ need for probabilistic predictions / quantification of uncertainty

Domhan et al, 2015: Parametric Learning Curves

- Use a parametric model f_k with parameters $\boldsymbol{\theta}$ to model performance at step t as: $y_t = f_k(t|\boldsymbol{\theta}) + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Domhan et al, 2015: Parametric Learning Curves

- Use a parametric model f_k with parameters $\boldsymbol{\theta}$ to model performance at step t as: $y_t = f_k(t|\boldsymbol{\theta}) + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.
- Linear combination of $K = 11$ parametric types of models: $f_{comb}(t|\boldsymbol{\xi}) = \sum_{k=1}^{K} w_k f_k(t|\boldsymbol{\theta}_k)$, where $\boldsymbol{\xi} = (w_1, \ldots, w_K, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K, \sigma^2)$

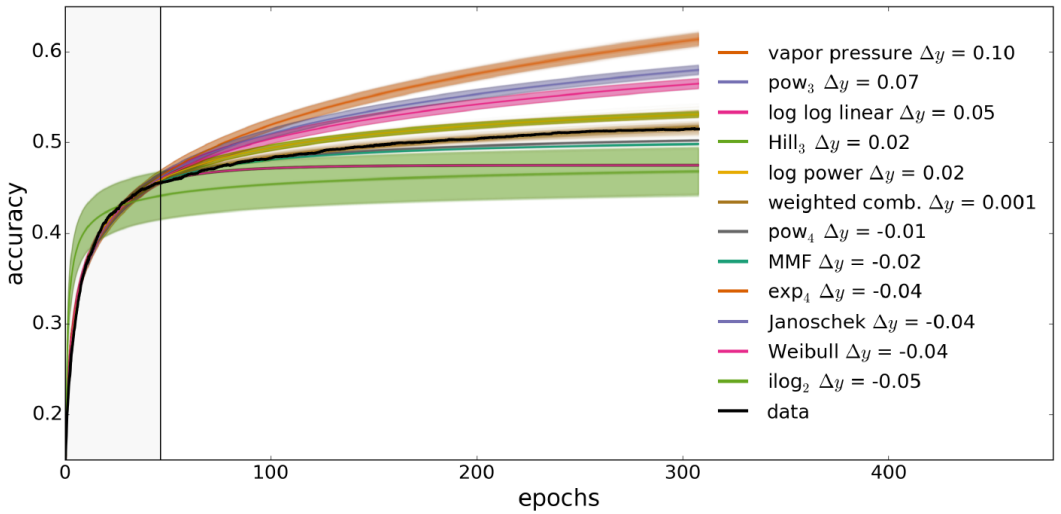


Reference name	Formula
vapor pressure	$\exp(a + \frac{b}{x} + c\log(x))$
pow_3	$c - ax^{-\alpha}$
log log linear	$\log(a\log(x) + b)$
Hill_3	$\frac{y_{\max} x^\eta}{\kappa^\eta + x^\eta}$
log power	$\frac{a}{1+\left(\frac{x}{e^b}\right)^c}$
pow_4	$c - (ax + b)^{-\alpha}$
MMF	$\alpha - \frac{\alpha-\beta}{1+(\kappa x)^\delta}$
exp_4	$c - e^{-ax^\alpha + b}$
Janoschek	$\alpha - (\alpha - \beta)e^{-\kappa x^\delta}$
Weibull	$\alpha - (\alpha - \beta)e^{-(\kappa x)^\delta}$
ilog_2	$c - \frac{a}{\log x}$

$K = 11$ parametric families for modelling learning curves

Domhan et al, 2015: Parametric Learning Curves

- Use a parametric model f_k with parameters $\boldsymbol{\theta}$ to model performance at step t as: $y_t = f_k(t|\boldsymbol{\theta}) + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.
- Linear combination of $K = 11$ parametric types of models: $f_{comb}(t|\boldsymbol{\xi}) = \sum_{k=1}^{K} w_k f_k(t|\boldsymbol{\theta}_k)$, where $\boldsymbol{\xi} = (w_1, \ldots, w_K, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K, \sigma^2)$

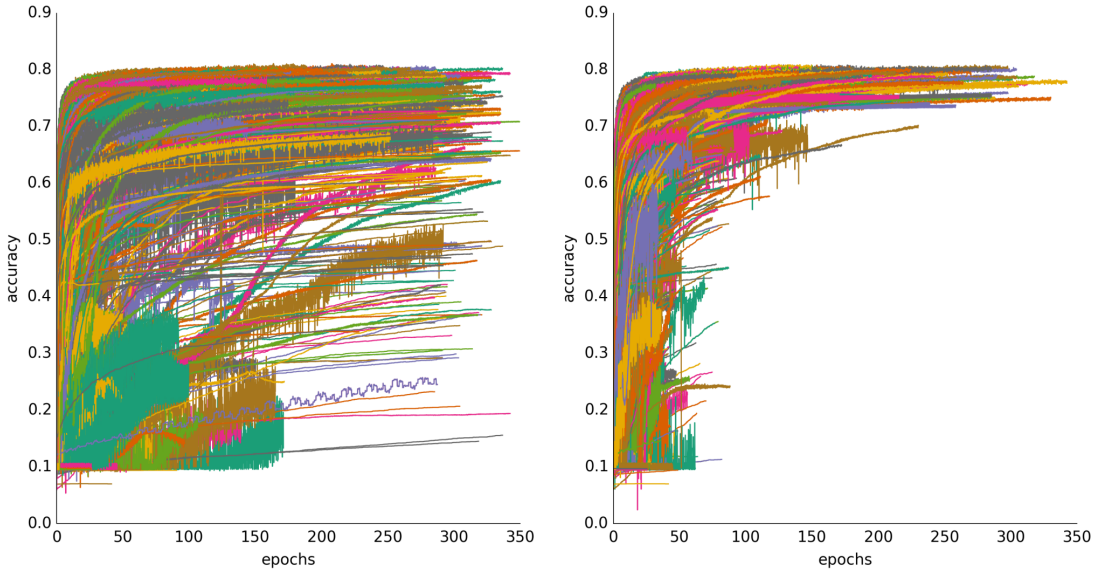


Reference name	Formula
vapor pressure	$\exp(a + \frac{b}{x} + c\log(x))$
pow_3	$c - ax^{-\alpha}$
log log linear	$\log(a\log(x) + b)$
Hill_3	$\frac{y_{\max} x^\eta}{\kappa^\eta + x^\eta}$
log power	$\frac{a}{1+\left(\frac{x}{e^b}\right)^c}$
pow_4	$c - (ax + b)^{-\alpha}$
MMF	$\alpha - \frac{\alpha-\beta}{1+(\kappa x)^\delta}$
exp_4	$c - e^{-ax^\alpha + b}$
Janoschek	$\alpha - (\alpha - \beta)e^{-\kappa x^\delta}$
Weibull	$\alpha - (\alpha - \beta)e^{-(\kappa x)^\delta}$
ilog_2	$c - \frac{a}{\log x}$

$K = 11$ parametric families for modelling learning curves

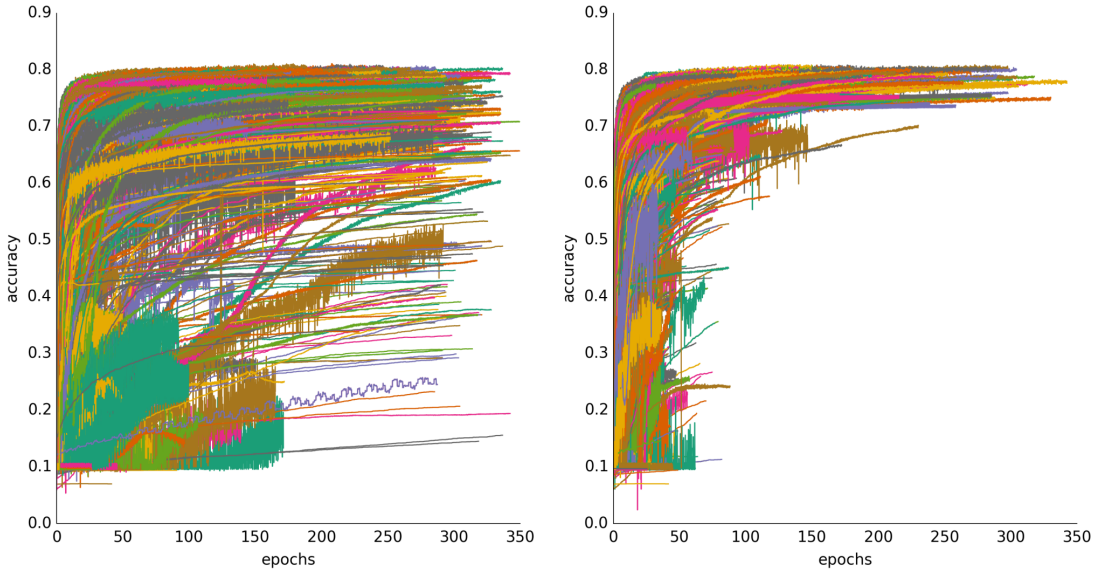
- Use Markov Chain Monte Carlo sampling of $\boldsymbol{\xi}$ to obtain uncertainties

Predictive Termination by Domhan et al, 2015



All learning curves vs. learning curves with early termination

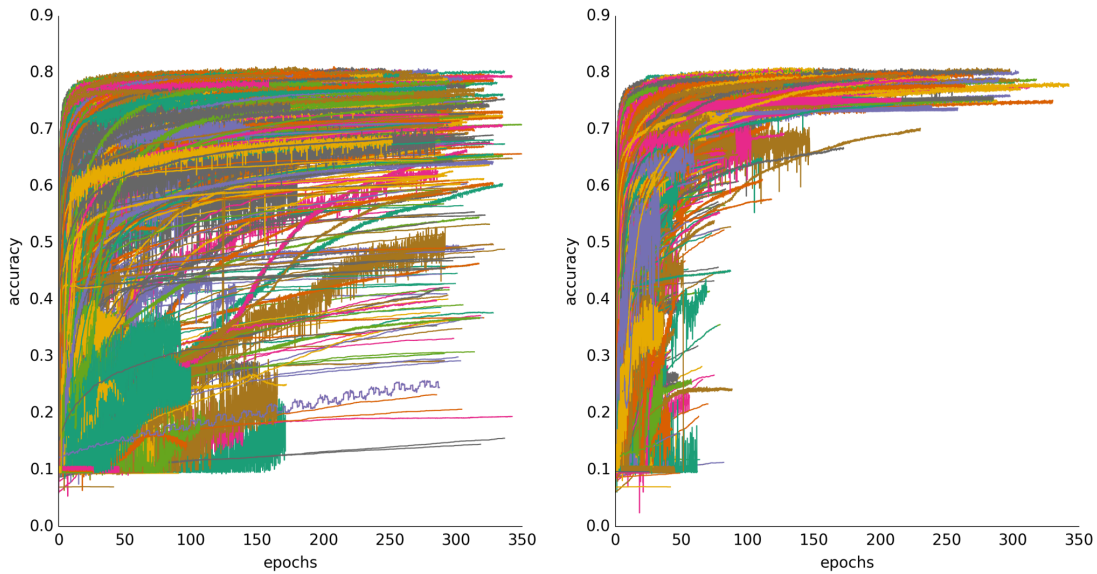
Predictive Termination by Domhan et al, 2015



All learning curves vs. learning curves with early termination

- Disadvantages of this model?

Predictive Termination by Domhan et al, 2015



All learning curves vs. learning curves with early termination

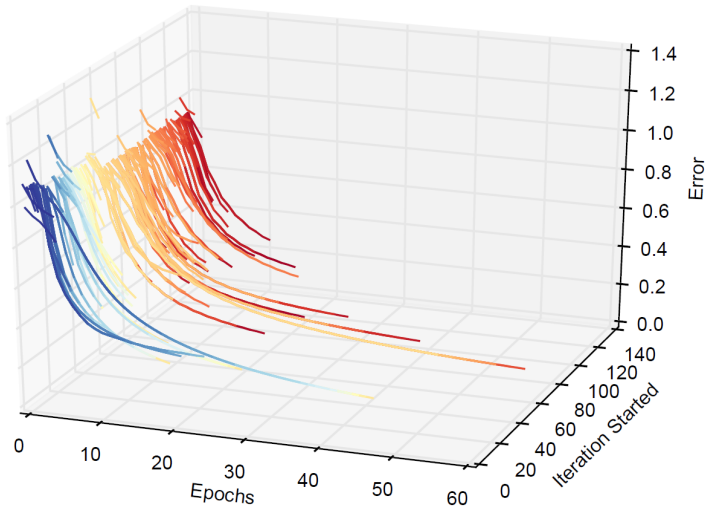
- Disadvantages of this model?
  - Relies on manually-selected parametric families of curves
  - Does not take into account hyperparameters used
    - $\rightarrow$ can't learn across hyperparameters
  - Does not even learn across curves; simply extrapolates one at a time

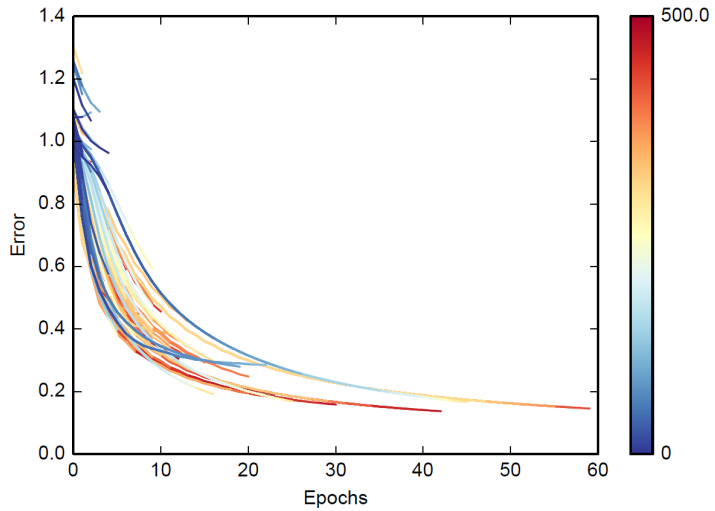
Freeze-Thaw Bayesian Optimization (Swersky et al, 2014)

- Use a Gaussian process with inputs λ and t; special kernel for t
- For N configurations and T epochs each: $O(N^3t^3) \rightarrow$ approximation
- Iteratively: either extend existing configuration or try new one

Freeze-Thaw Bayesian Optimization (Swersky et al, 2014)

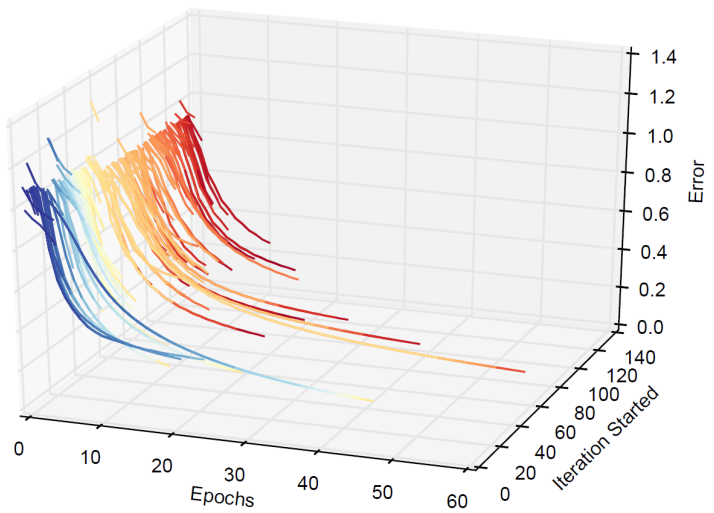
- Use a Gaussian process with inputs λ and t; special kernel for t
- For N configurations and T epochs each: $O(N^3t^3) \rightarrow$ approximation
- Iteratively: either extend existing configuration or try new one
- Result for probabilistic matrix factorization:

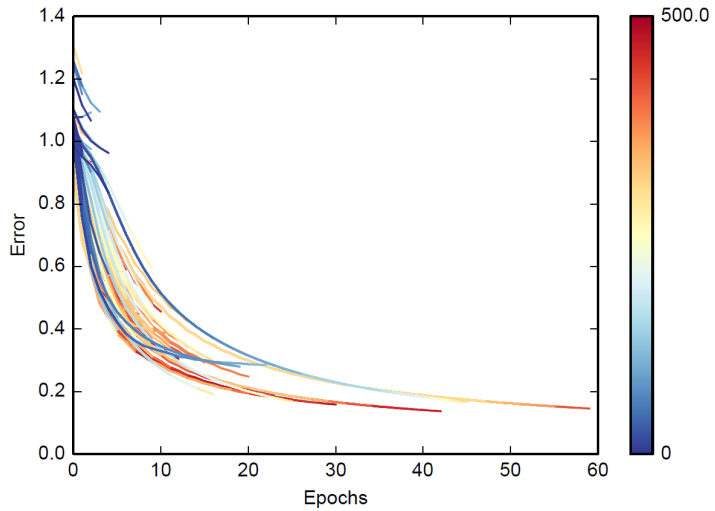




Freeze-Thaw Bayesian Optimization (Swersky et al, 2014)

- Use a Gaussian process with inputs λ and t; special kernel for t
- For N configurations and T epochs each: $O(N^3t^3) \rightarrow$ approximation
- Iteratively: either extend existing configuration or try new one
- Result for probabilistic matrix factorization:

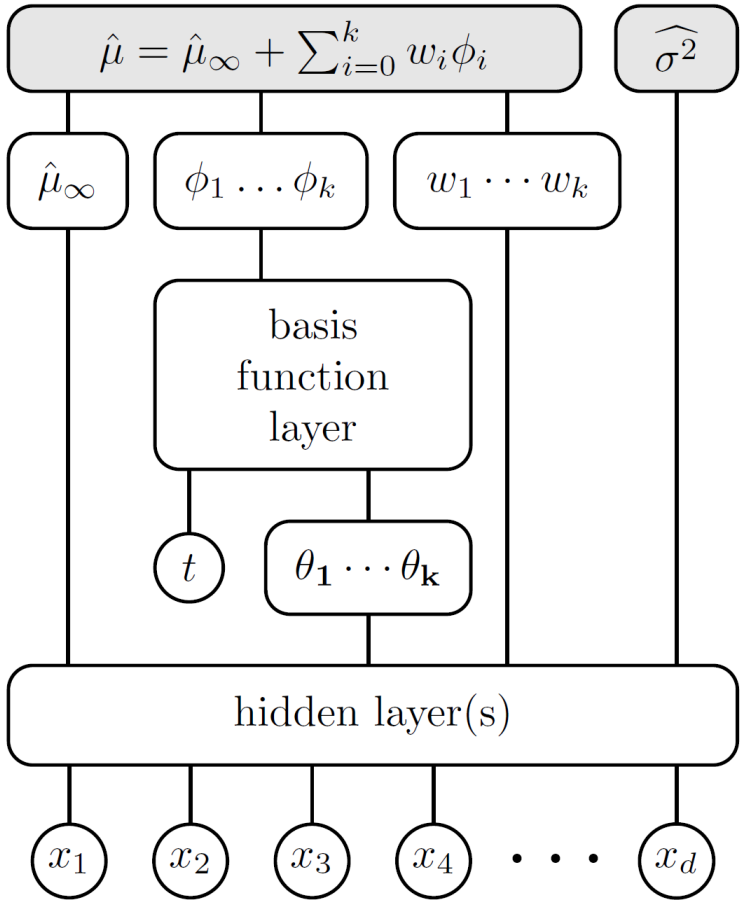




- Unfortunately, no results for DNNs; no code available

Klein et al, 2017: LC-Net

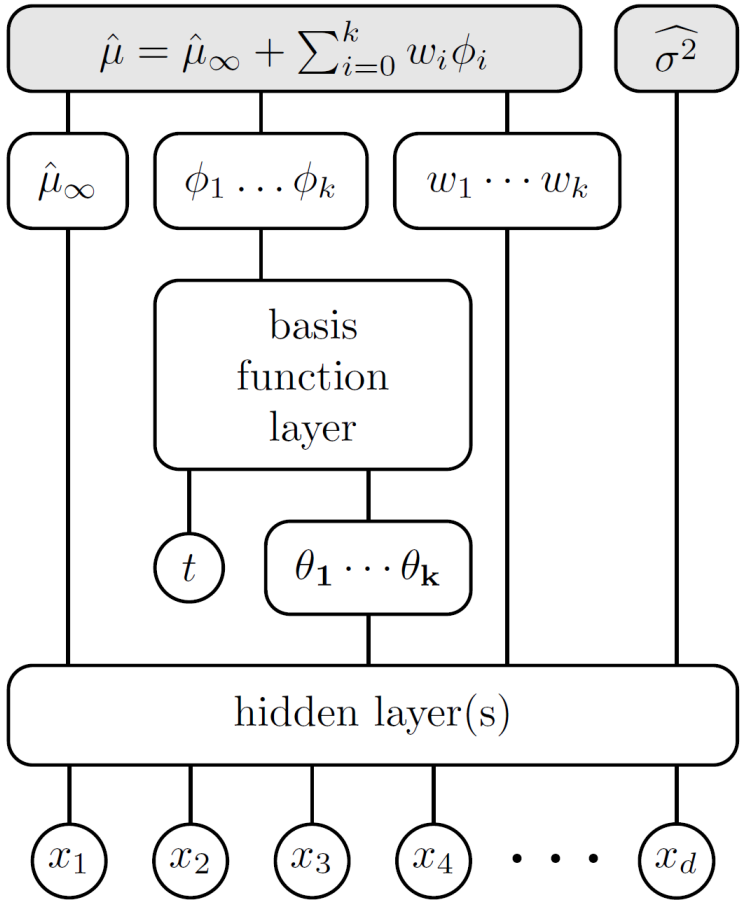
- Make a layer out of the parametric learning curves by Domhan et al.
- Also support hyperparameters as inputs (in the figure denoted by $x_1, \ldots, x_d$)



Klein et al, 2017: LC-Net

- Make a layer out of the parametric learning curves by Domhan et al.
- Also support hyperparameters as inputs (in the figure denoted by $x_1, \ldots, x_d$)

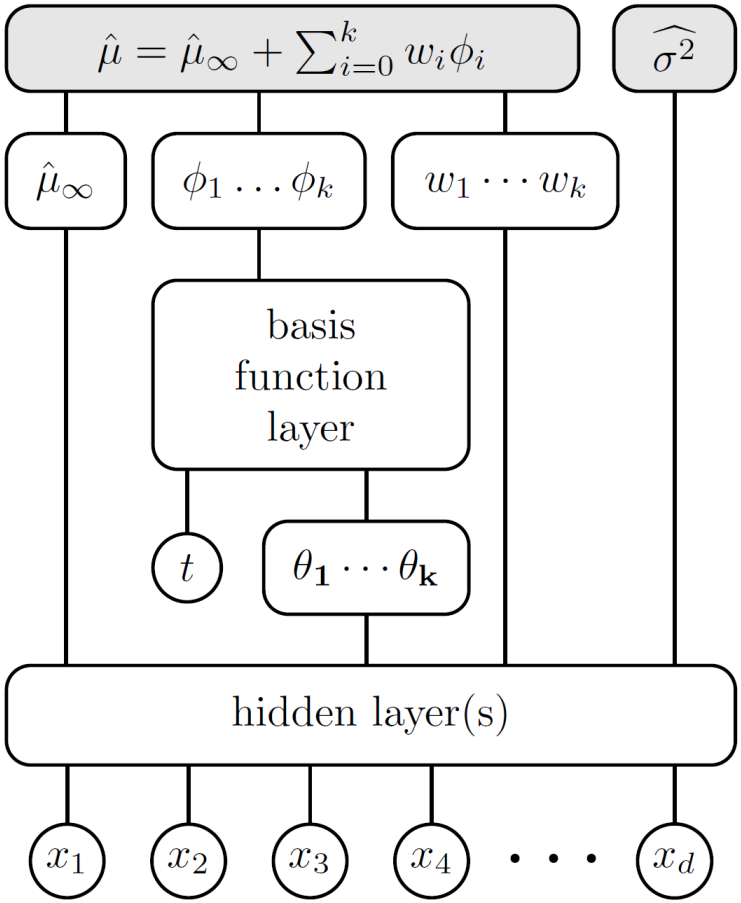
- Disadvantages of this model?



Klein et al, 2017: LC-Net

- Make a layer out of the parametric learning curves by Domhan et al.
- Also support hyperparameters as inputs (in the figure denoted by $x_1, \ldots, x_d$)

- Disadvantages of this model?
  - Relies on manually-selected parametric families of curves
  - Cannot quickly integrate new information from extending the current curve (or from new runs)

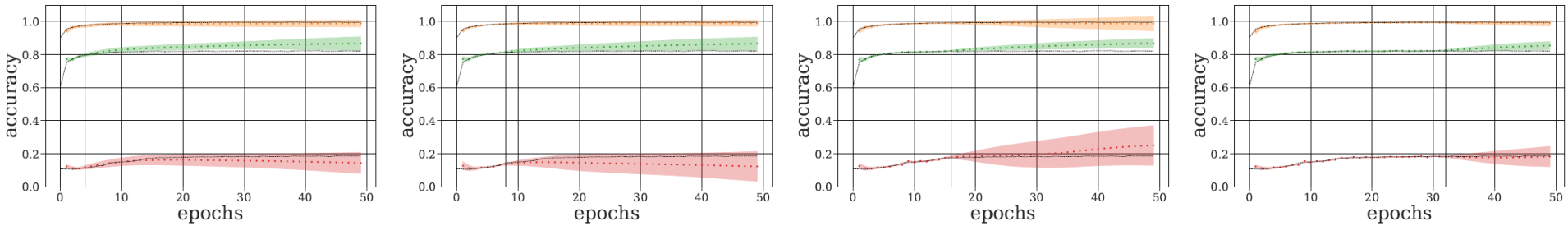


Gargiani et al, 2019: Sequence Models (e.g., Bayesian RNN)

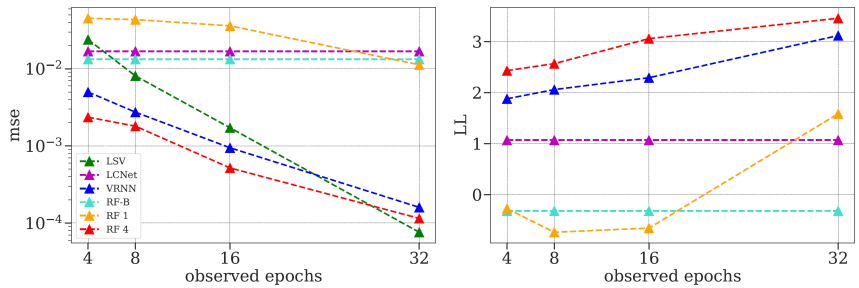
- Learning curves are sequences
  - Previous models don't treat them like this
  - We can use an RNN (in particular, an LSTM) to predict the next value from a given sequence
  - We can use variational dropout to obtain uncertainty estimates:

Gargiani et al, 2019: Sequence Models (e.g., Bayesian RNN)

- Learning curves are sequences
  - Previous models don't treat them like this
  - We can use an RNN (in particular, an LSTM) to predict the next value from a given sequence
  - We can use variational dropout to obtain uncertainty estimates:



- Note: we can also use a simpler model
  - E.g., a random forest to map from a fixed-size window to the next value



Compare: Baker et al, 2017

- Idea: Map from configurations (including architectural hyperparameters) and partial learning curves to the final performance
- Advantages
  - Much simpler idea than all the approaches just discussed: no need to model the entire learning curve
  - Much easier to implement
- Disadvantage?

Compare: Baker et al, 2017

- Idea: Map from configurations (including architectural hyperparameters) and partial learning curves to the final performance
- Advantages
  - Much simpler idea than all the approaches just discussed: no need to model the entire learning curve
  - Much easier to implement
- Disadvantage? $\rightarrow$ requires many (e.g., 100) fully-evaluated learning curves as training data
  - After 100 full function evaluations we want to be pretty much converged in practice
  - But definitely helpful for speeding up RL
  - Can be combined with Hyperband and be used whenever we have enough runs for a budget

Possible extension of LC models waiting to be done

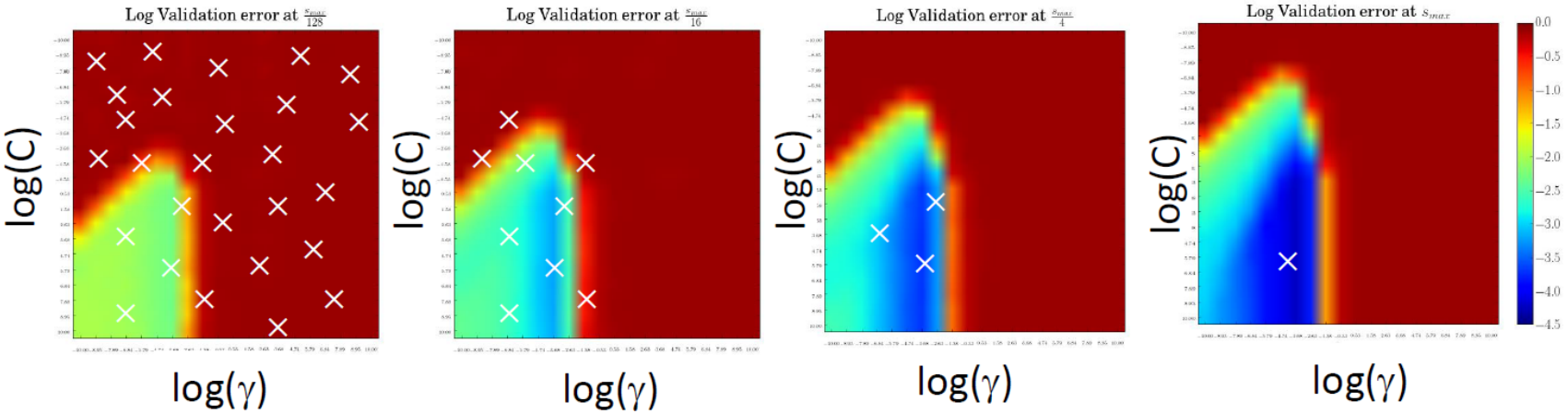
- We could keep track of additional information to feed to our model for better predictions
  - E.g., training & validation cross-entropy loss & accuracy
    - Instead of only validation accuracy
  - E.g., split cross-entry into data-dependent & weight dependent parts
  - E.g., keep track of gradient norms, activation statistics, ...
- Information about learning rate (& weight decay) at each step

$\rightarrow$ possible student project

Lecture Overview

Motivating Example

- Performance of an SVM on MNIST and subsets of it:



- Evaluations on the smallest subset (128 data points) cost 10 000 less than on the full data set

Multifidelity Optimization

- Use cheap approximations of the expensive blackbox, performance on which correlates with the blackbox, e.g.
  - Subsets of the data
  - Fewer epochs of iterative training algorithms (e.g., SGD; DARTS)
  - Shorter MCMC chains in Bayesian deep learning
  - Fewer trials in deep reinforcement learning
  - Downsampled images in object recognition
  - Shallower networks

Multifidelity Optimization

- Use cheap approximations of the expensive blackbox, performance on which correlates with the blackbox, e.g.
  - Subsets of the data
  - Fewer epochs of iterative training algorithms (e.g., SGD; DARTS)
  - Shorter MCMC chains in Bayesian deep learning
  - Fewer trials in deep reinforcement learning
  - Downsampled images in object recognition
  - Shallower networks
- Also applicable in different domains, e.g., fluid simulations:
  - Less particles
  - Shorter simulations

Multifidelity Optimization

- Use cheap approximations of the expensive blackbox, performance on which correlates with the blackbox, e.g.
  - Subsets of the data
  - Fewer epochs of iterative training algorithms (e.g., SGD; DARTS)
  - Shorter MCMC chains in Bayesian deep learning
  - Fewer trials in deep reinforcement learning
  - Downsampled images in object recognition
  - Shallower networks
- Also applicable in different domains, e.g., fluid simulations:
  - Less particles
  - Shorter simulations
- Possible project (Arber has already been looking into this): using multiple fidelities to find best setting of DARTS

One approach for handling multiple fidelities

- Multi-fidelity Bayesian optimization for fidelity with values $b \in B$
- Standard Bayesian optimization uses a model $f(\lambda) \approx y$ to select the next λ
- Multi-fidelity Bayesian optimization uses a model $f(\lambda, b) \approx y$ to select the next (λ, b)

One approach for handling multiple fidelities

- Multi-fidelity Bayesian optimization for fidelity with values $b \in B$
- Standard Bayesian optimization uses a model $f(\lambda) \approx y$ to select the next λ
- Multi-fidelity Bayesian optimization uses a model $f(\lambda, b) \approx y$ to select the next (λ, b)
  - Model f needs to be good at extrapolating from small to large b
  - E.g., a learning curve model
    - When the budgets are different numbers of epochs for SGD
  - E.g., a Gaussian process for extrapolating from small to large datasets
    - 1 000-fold speedups for SVM on MNIST example [Klein et al, AISTATS 2017]
    - This was based on Entropy Search

Refresher: Entropy Search

- Define the p_{min} distribution given data $\mathcal{D}$:

$$p_{\text{min}}(x' \mid \mathcal{D}) := p(x' \in \arg\min_{x \in X} f(x) \mid \mathcal{D})$$

- Entropy search aims to minimize the entropy $H[p_{\text{min}}]$
- In a nutshell:
  1. Estimate $p(f \mid \mathcal{D})$, e.g., using a GP (as before)
  2. Approximate p_{min} by representer points and Monte-Carlo simulations
  3. Select x that minimizes the following acquisition function:

$$a_{\text{ES}}(x) := \mathbb{E}_{p(y|x,\mathcal{D})}\left[H[p_{\text{min}}(\cdot \mid \mathcal{D} \cup \{(x,y)\})]\right]$$

Multi-fidelity Bayesian Optimization with Entropy Search

- We care about the p_{min} distribution for the maximal budget B:

$$p_{\text{min}}(x' \mid \mathcal{D}) := p(x' \in \arg\min_{x \in X} f(x, B) \mid \mathcal{D})$$

- We still want to minimize the entropy $H[p_{\text{min}}]$

Multi-fidelity Bayesian Optimization with Entropy Search

- We care about the p_{min} distribution for the maximal budget B:

$$p_{\text{min}}(x' \mid \mathcal{D}) := p(x' \in \arg\min_{x \in X} f(x, B) \mid \mathcal{D})$$

- We still want to minimize the entropy $H[p_{\text{min}}]$
- Now we aim for the biggest reduction in entropy per time spent
  - Next to f, we now also model the cost $c(x, b)$
  - We choose the next (x, b) by maximizing:

$$a_{\text{ES}}(x, b) := \mathbb{E}_{p(y|(x,b),\mathcal{D})} \left[\frac{H[p_{\text{min}}(\cdot \mid \mathcal{D})] - H[p_{\text{min}}(\cdot \mid \mathcal{D} \cup \{((x, b), y)\})]}{c(x, b)} \right]$$

Multi-fidelity Bayesian Optimization with Entropy Search

- The entire algorithm iterates the following 2 steps until time is up:
  1. Select (x, b) by maximizing:

$$a_{\text{ES}}(x, b) := \mathbb{E}_{p(y|(x,b),\mathcal{D})} \left[\frac{H[p_{\text{min}}(\cdot \mid \mathcal{D})] - H[p_{\text{min}}(\cdot \mid \mathcal{D} \cup \{((x, b), y)\}]}{c(x, b)} \right]$$

  2. Observe performance and cost, and update models f and c

Multi-fidelity Bayesian Optimization with Entropy Search

- The entire algorithm iterates the following 2 steps until time is up:
  1. Select (x, b) by maximizing:

$$a_{\text{ES}}(x, b) := \mathbb{E}_{p(y|(x,b),\mathcal{D})}\left[\frac{H[p_{\min}(\cdot \mid \mathcal{D})] - H[p_{\min}(\cdot \mid \mathcal{D} \cup \{((x, b), y)\}]}{c(x, b)}\right]$$

  2. Observe performance and cost, and update models f and c

- Pros:
  - Conceptually beautiful
  - 1 000-fold speedups for optimizing SVMs on MNIST
- Cons:
  - Scalability of GPs is a big problem (limits size of initial design)
  - Limited applicability of Gaussian processes

Related works

- Similar ways to attack the same problem
  - First choose x based on UCB, then choose b afterwards [Kandasamy et al, 2016; Kandsamy et al, 2017]
  - Use the knowledge gradient acquisition function instead of entropy search [Wu et al, 2019]
  - Use the efficient max-value entropy search instead of standard entropy search [Takeno et al, 2019]

Related works

- Similar ways to attack the same problem
  - First choose x based on UCB, then choose b afterwards [Kandasamy et al, 2016; Kandsamy et al, 2017]
  - Use the knowledge gradient acquisition function instead of entropy search [Wu et al, 2019]
  - Use the efficient max-value entropy search instead of standard entropy search [Takeno et al, 2019]
  - Possible project: compare these empirically

Related works

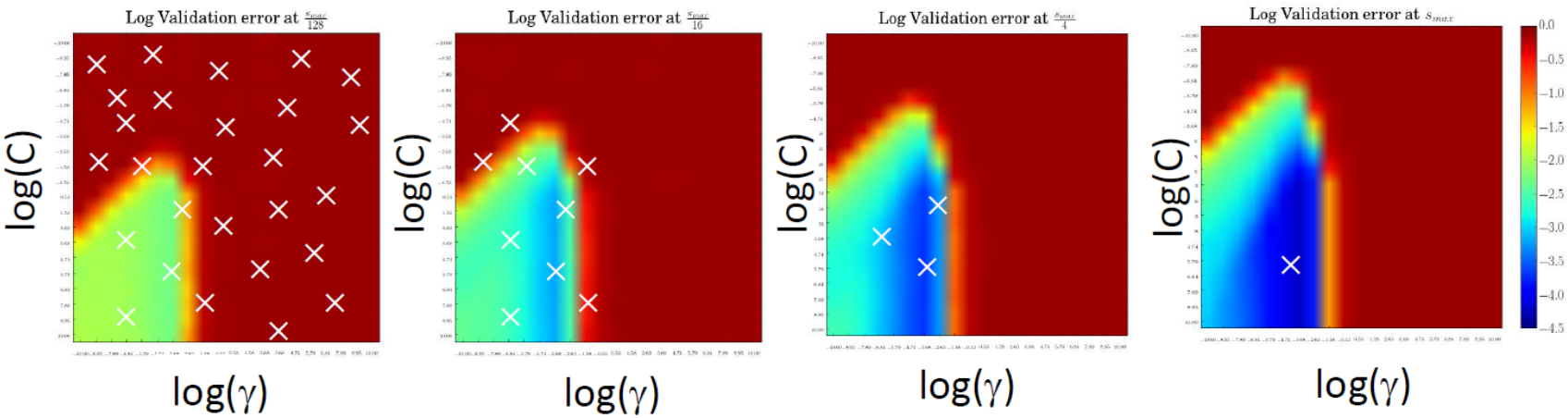
- Similar ways to attack the same problem
  - First choose x based on UCB, then choose b afterwards [Kandasamy et al, 2016; Kandsamy et al, 2017]
  - Use the knowledge gradient acquisition function instead of entropy search [Wu et al, 2019]
  - Use the efficient max-value entropy search instead of standard entropy search [Takeno et al, 2019]
  - Possible project: compare these empirically
- More scalable models
  - Bayesian optimization with Bayesian neural networks [Springenberg et al, 2016]; but not yet evaluated for the multi-fidelity case

Related works

- Similar ways to attack the same problem
  - First choose x based on UCB, then choose b afterwards [Kandasamy et al, 2016; Kandsamy et al, 2017]
  - Use the knowledge gradient acquisition function instead of entropy search [Wu et al, 2019]
  - Use the efficient max-value entropy search instead of standard entropy search [Takeno et al, 2019]
  - Possible project: compare these empirically
- More scalable models
  - Bayesian optimization with Bayesian neural networks [Springenberg et al, 2016]; but not yet evaluated for the multi-fidelity case
- Possible project: scalable model for the multi-multi-fidelity case (or also some model-free approach)

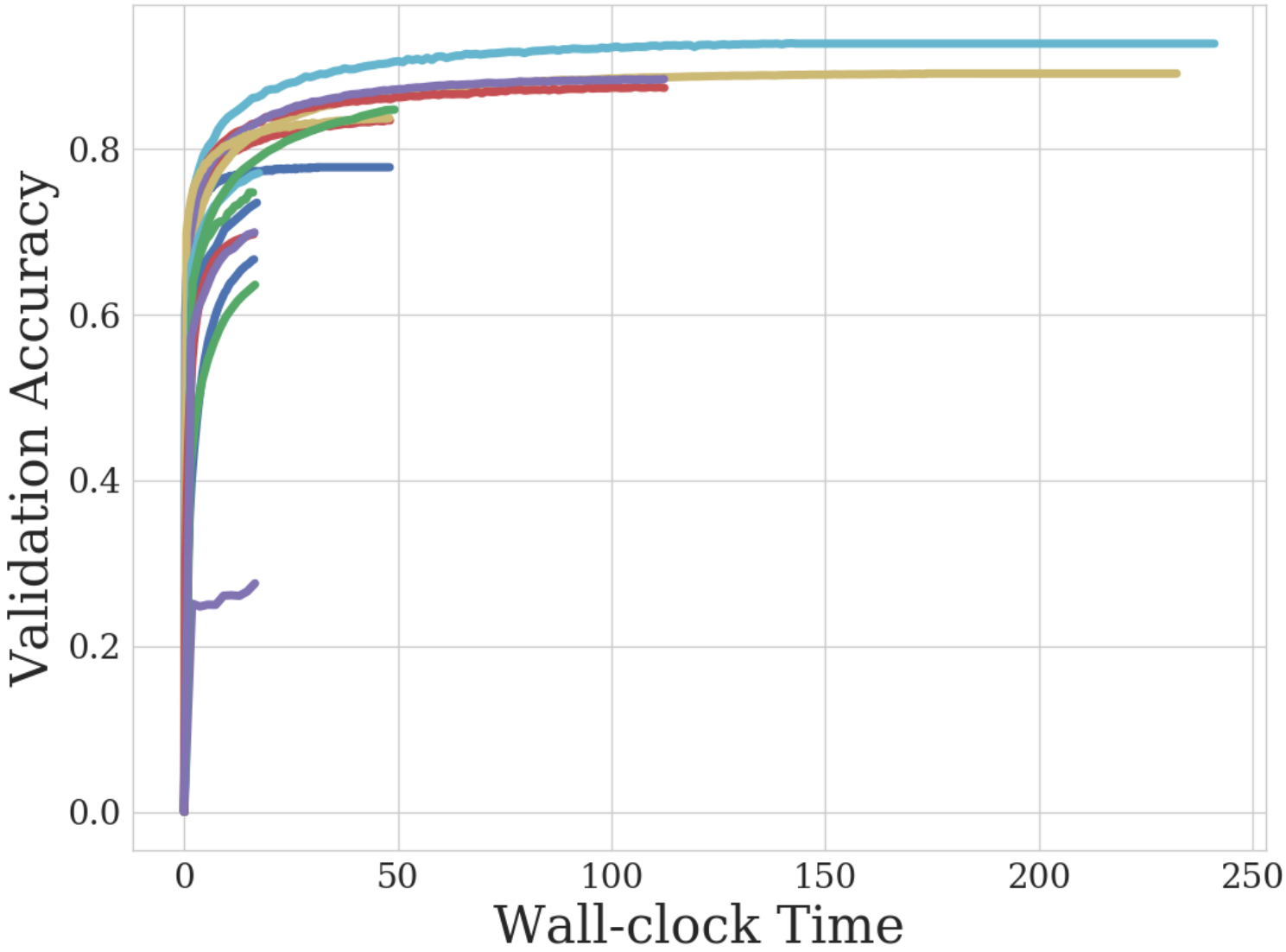
Can't we solve this problem more easily?

- Performance of an SVM on MNIST and subsets of it:

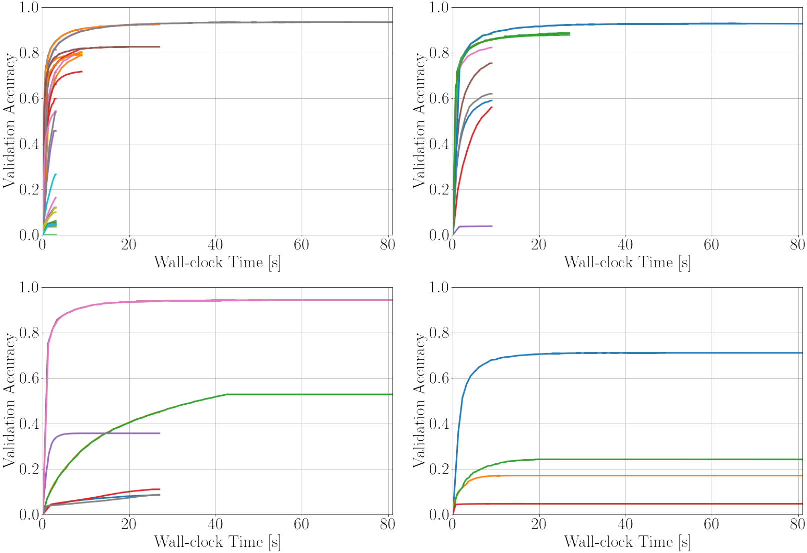


- Successive Halving [Jamieson & Talwalkar, 2015]
  - Try a lot of random configurations on lowest budget
  - Let best ones survive to next budget; iterate
- This is surprisingly effective!

Successive Halving with a Wall Clock Time Budget



Hyperband with a Wall Clock Time Budget: 4 iterations



[Li et al, 2017]

Hyperband

Algorithm 1: Pseudocode for Hyperband

input : budgets b_{min} and b_{max}, η

$s_{max} = \lfloor \log_\eta \frac{b_{max}}{b_{min}} \rfloor + 1$ // number of different iterations

for $s_i \in \{ s_{max} - 1, s_{max} - 2, \ldots, 1, 0 \}$ **do**

  $n = \lceil \frac{s_{max}}{s_i + 1} \cdot \eta^{s_i} \rceil$ // number of initial configurations

  $b = \eta^{-s_i} \cdot b_{max}$ // initial budget

  $\mathsf{C} = \text{sample\_random\_configurations}(n)$

  do

    $\mathsf{L} = \text{map}(\text{evaluate}, C, b)$ // run with budget b

    $n_i = \lfloor n_i / \eta \rfloor$ // reduce number of configurations

    $b_i = b_i \cdot \eta$ // increase their budget

    $C = C[\text{argsort}(L)[: n_i]]$ // keep best η^{-1} of configs

  while $b_i \leq b_{max}$

  return configuration or model with lowest loss

BOHB: Robust and Efficient Hyperparameter Optimization at Scale

- Simple Combination of Bayesian Optimization and HyperBand
  [Falkner et al, ICML 2018]
  - Bayesian optimization for selecting configurations (a TPE-like variant)
  - Hyperband for selecting the budgets for them

BOHB: Robust and Efficient Hyperparameter Optimization at Scale

- Simple Combination of Bayesian Optimization and HyperBand
  [Falkner et al, ICML 2018]
  - Bayesian optimization for selecting configurations (a TPE-like variant)
  - Hyperband for selecting the budgets for them
- Advantages
  - Robust and efficient off-the-shelf tool
  - Strong anytime performance
  - Strong performance with larger budgets
  - Scalable to high dimensions, parallel workers, different parameter types (categorical, integer, continuous)
  - On Github: `https://github.com/automl/HpBandSter`

BOHB: Robust and Efficient Hyperparameter Optimization at Scale

- Simple Combination of Bayesian Optimization and HyperBand
  [Falkner et al, ICML 2018]
  - Bayesian optimization for selecting configurations (a TPE-like variant)
  - Hyperband for selecting the budgets for them
- Advantages
  - Robust and efficient off-the-shelf tool
  - Strong anytime performance
  - Strong performance with larger budgets
  - Scalable to high dimensions, parallel workers, different parameter types (categorical, integer, continuous)
  - On Github: `https://github.com/automl/HpBandSter`
- To the best of my knowledge, still the best available tool
- Many possible projects around BOHB – better models, multi-multi-fidelity, etc

How BOHB chooses the next configuration

Algorithm 2: BOHB: choose configuration

input : observations D, fraction of random runs ρ, percentile q, number of samples N_s, minimum number of points N_{min} to build a model, and bandwidth factor b_w

output : next configuration to evaluate

if *rand()* $< \rho$ **then return** random configuration

$b = \arg\max \{D_b : |D_b| \geq N_{min} + 2\}$

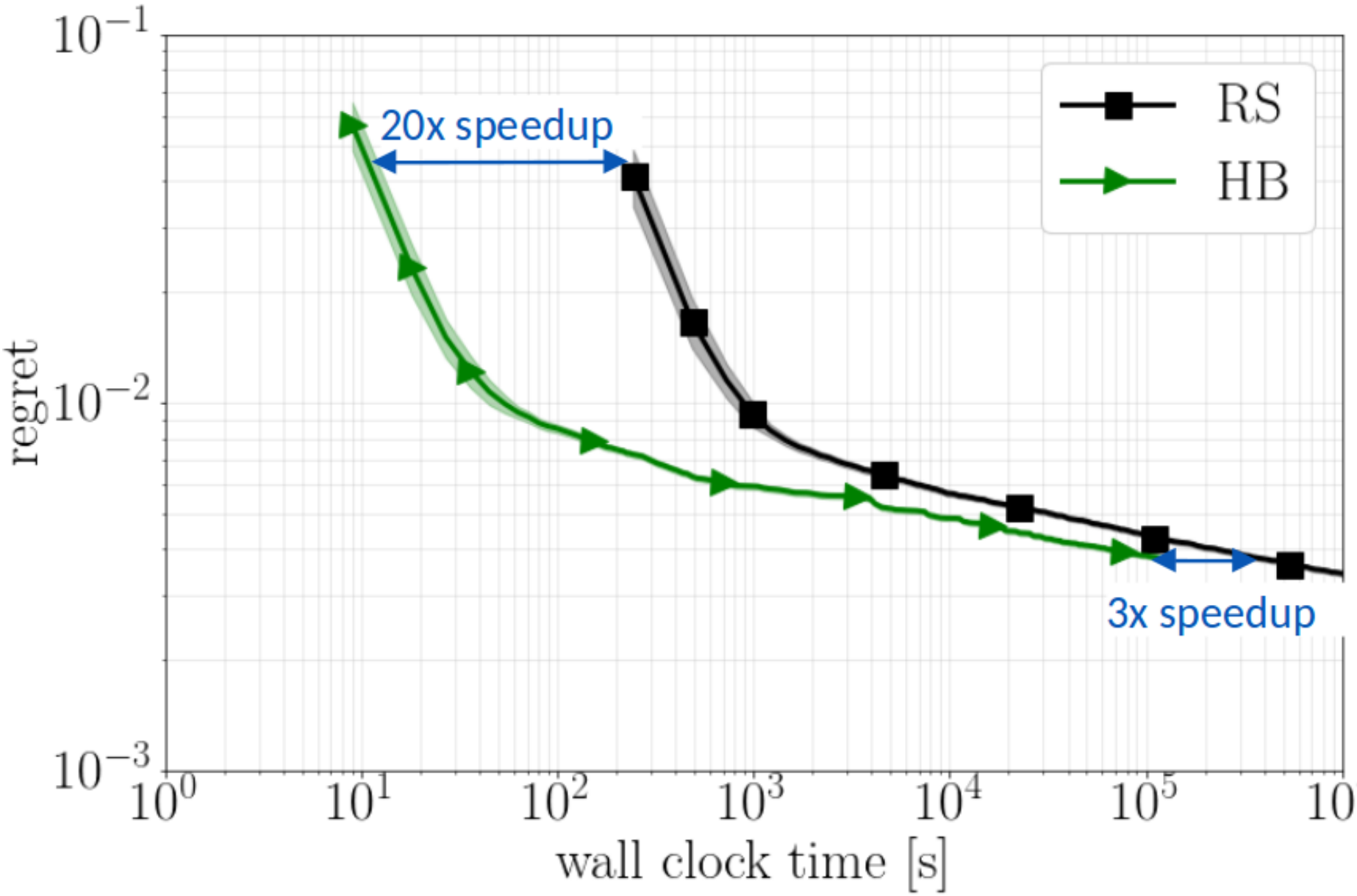
if $b = \emptyset$ **then return** random configuration

fit KDEs $l(\cdot)$ and $g(\cdot)$ to good and bad points, respectively

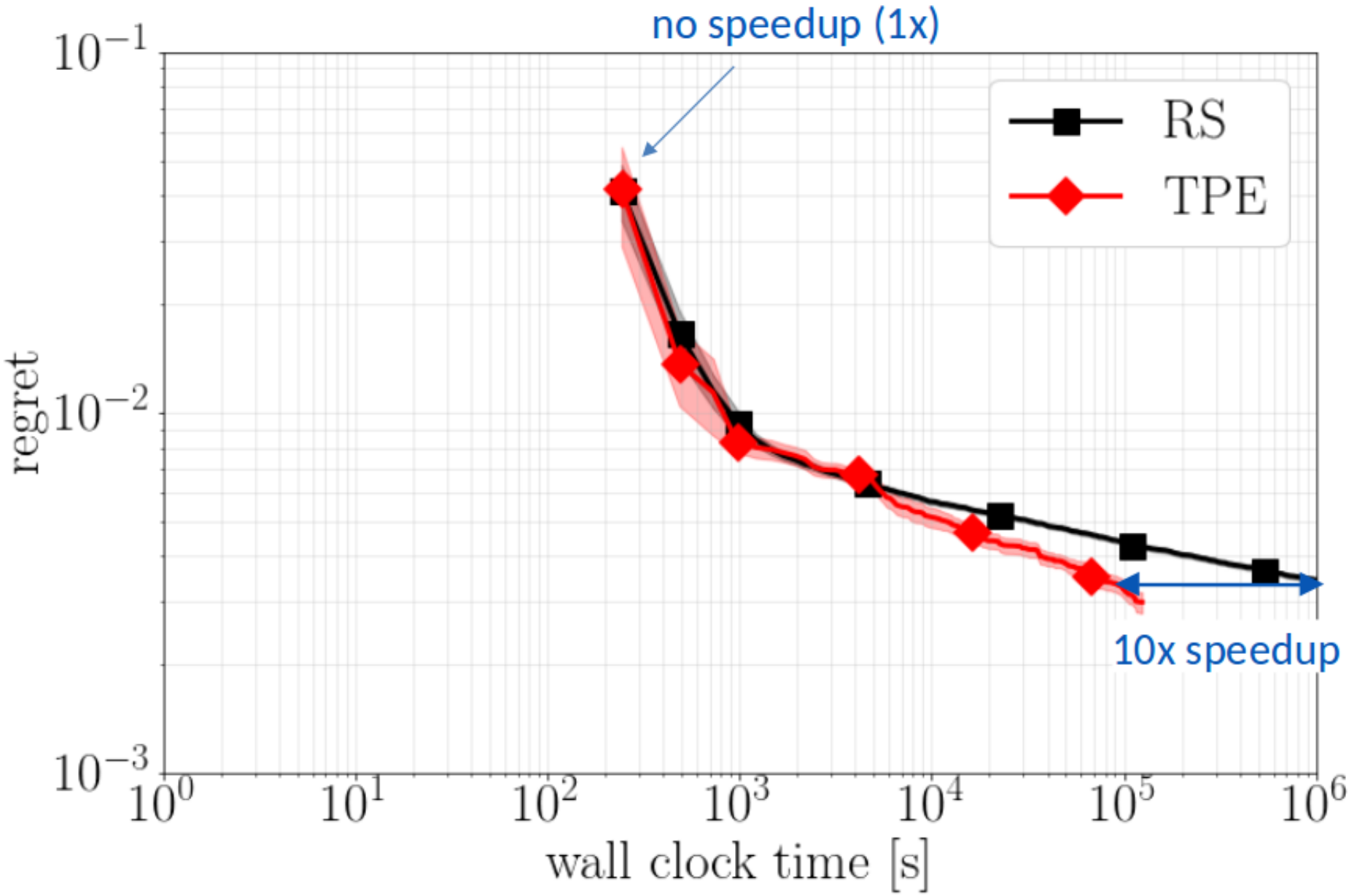
draw N_s samples according to $l'(x)$ (similar to $l(x)$)

return sample with highest ratio $l(x)/g(x)$

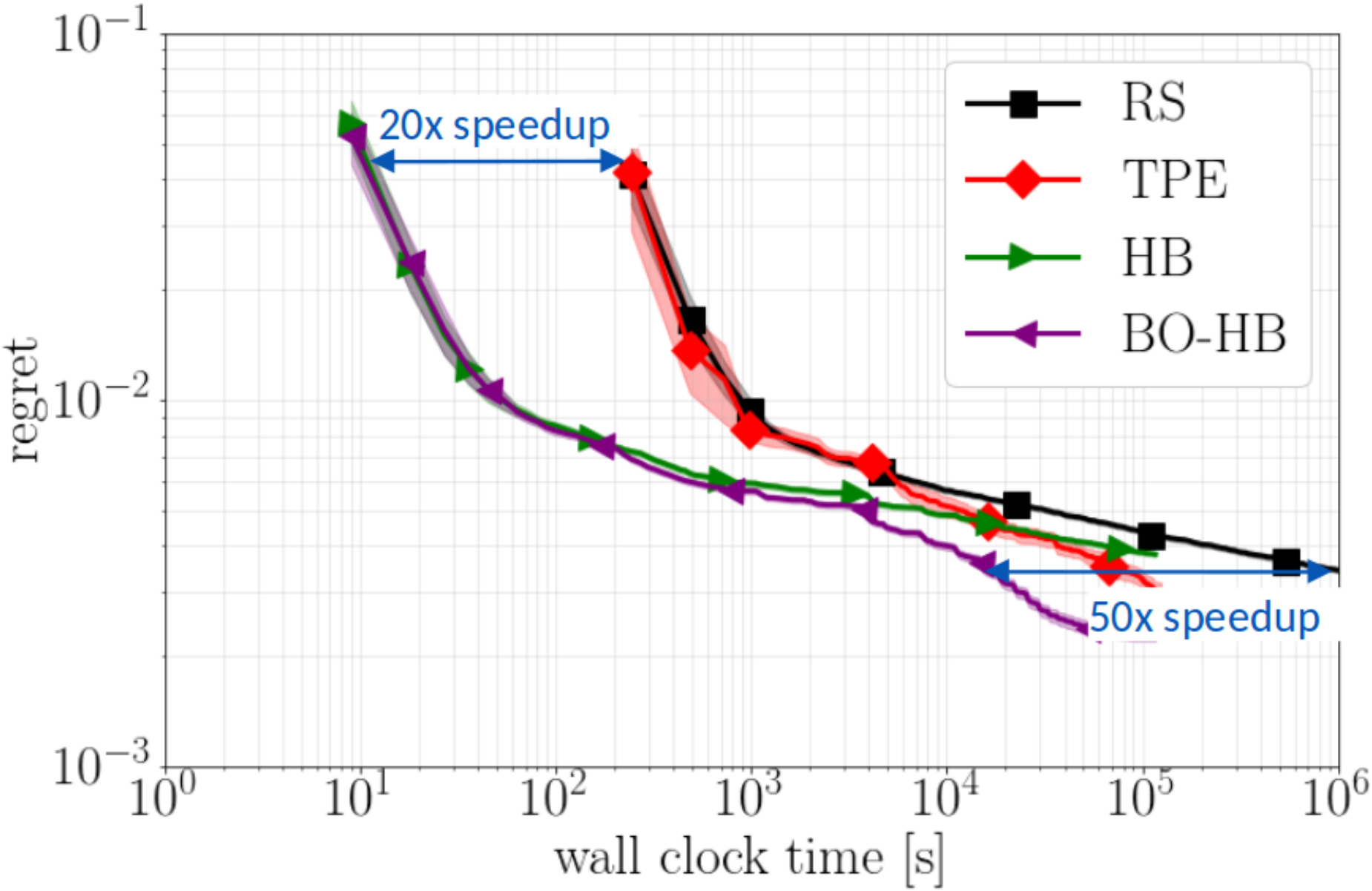
Random search vs. Hyperband



Random search vs. Bayesian optimization



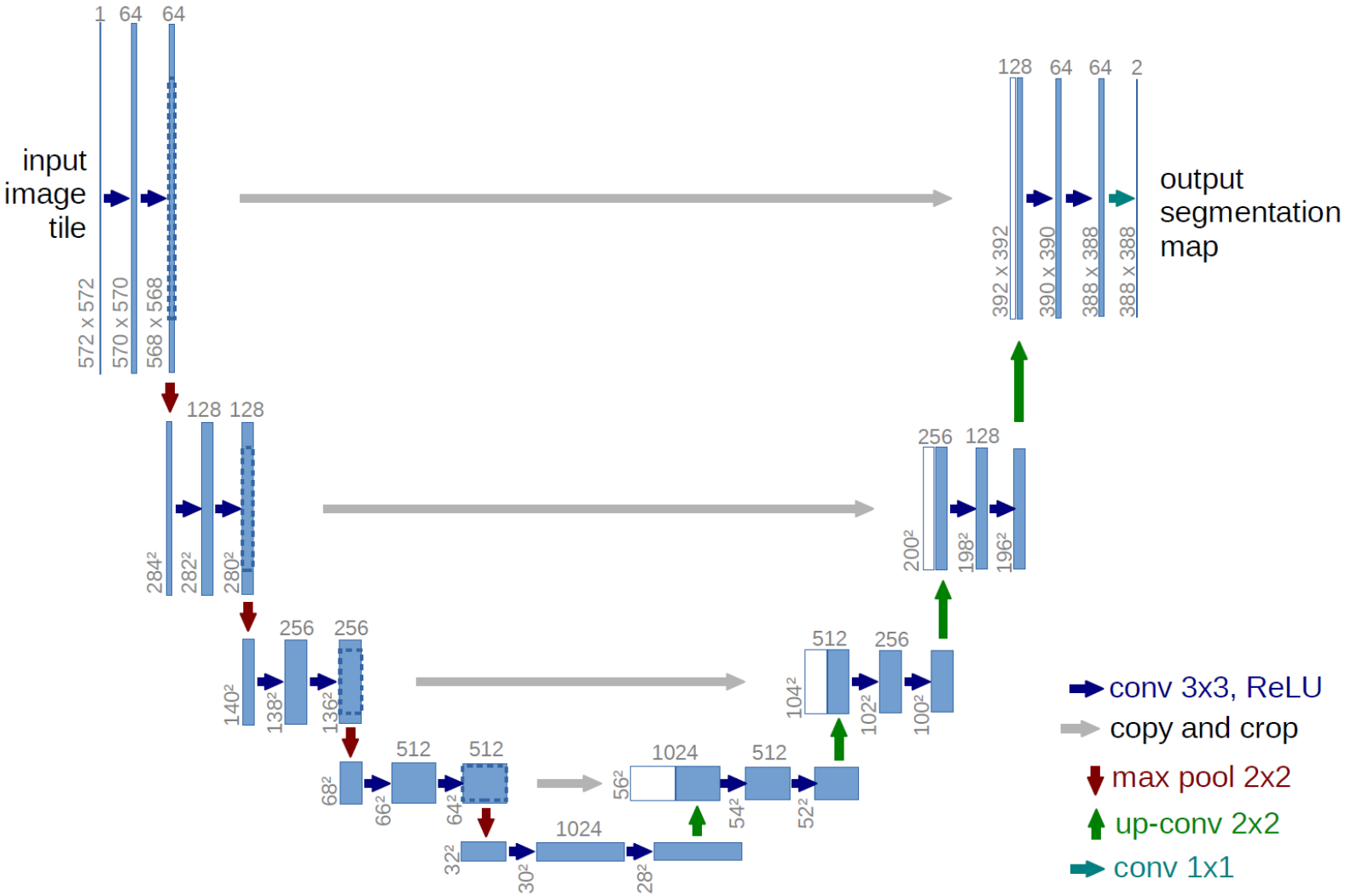
BOHB achieves the best of both worlds



Lecture Overview

The Problem of Disparity Estimation

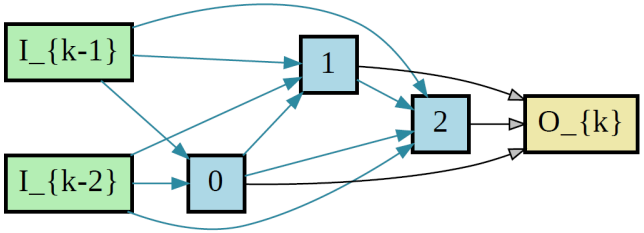
Background: U-Net



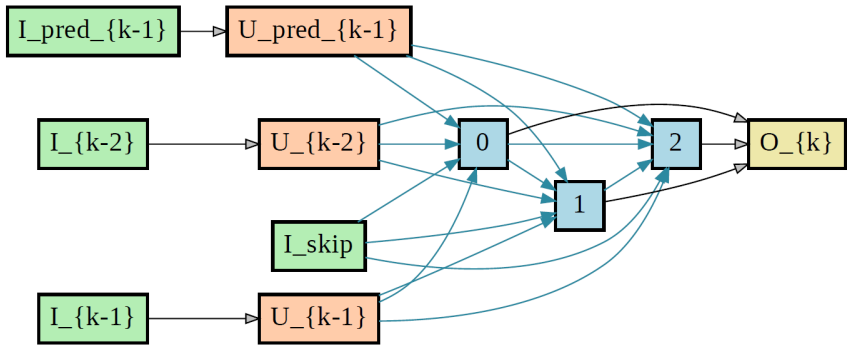
Skip connections from similar spatial resolution to avoid loosing information

Search space

- Standard cells like in DARTS for
  - Keeping spatial resolution
  - Downsampling
- New upsampling cell that supports U-Net like skip connections



(a) Normal/Reduction cell



(b) Upsampling cell

Neural Architecture Search by DARTS, HPO by BOHB

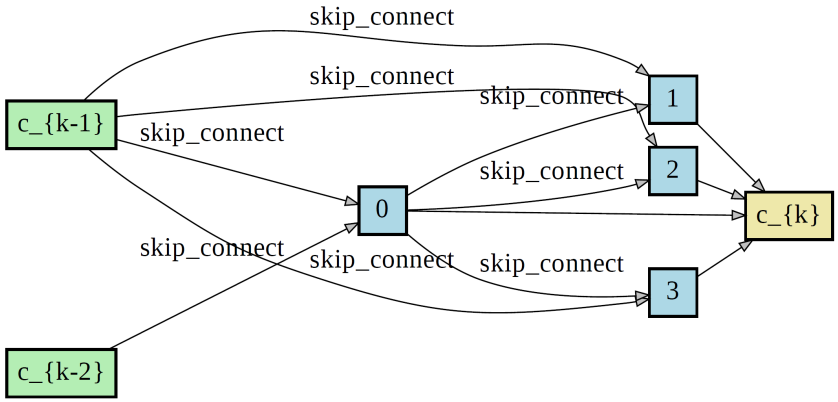
- NAS: optimize neural architecture with DARTS
  - Faster than BOHB
- HPO: then optimize hyperparameters with BOHB
  - DARTS does not apply
  - The weight sharing idea is restricted to the architecture space
- Result [Saikat et al, 2019]
  - Both NAS and HPO yielded substantial improvements
  - E.g., EPE on Sintel: $2.36 \rightarrow 2.14 \rightarrow 1.94$

Details for Neural Architecture Search by DARTS

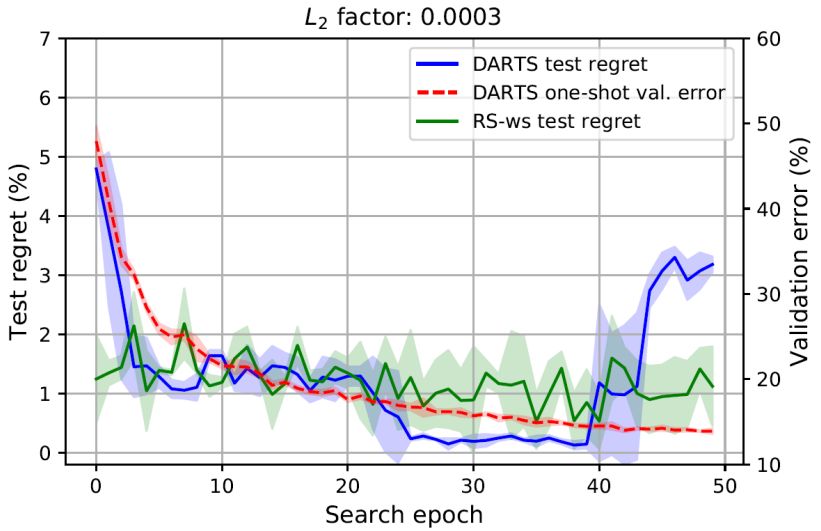
- Performance improved substantially
- Very important: warmstarting of the network weights
  - First, keep one-shot architecture weights fixed to the uniform distribution
  - Only afterwards, alternative updates of weights and architectural parameters
- Without warmstarting:
  DARTS found cells with only parameterless operations

Failure Modes of DARTS

- Idenfified six small search spaces where DARTS fails badly
  [Zela et al, 2019]
- In all cases, it only selects skip connections



- Overfitting of validation loss
- Can be fixed by more regularization:
  L_2 (for the inner objective), droppath, or early stopping



Recommendations for NAS and HPO

- So far, NAS often does not readily work out of the box
- BOHB typically does work out of the box
- Option 1: NAS, followed by HPO
- Option 2: Use HPO wrapped around NAS to make it more robust
  - Not evaluated so far, but likely good

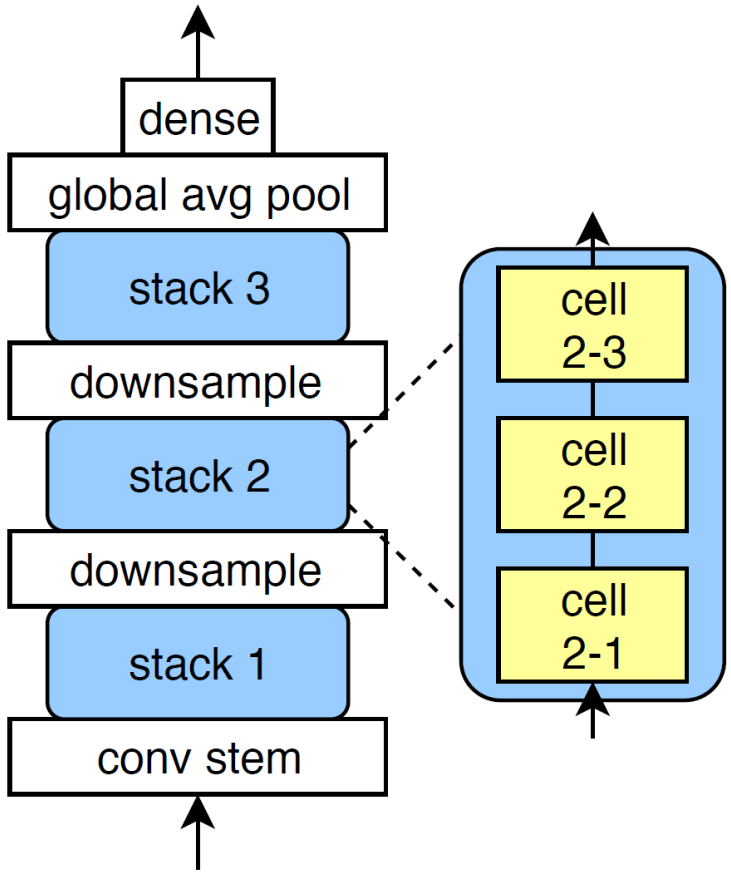
Lecture Overview

NAS-Bench-101

- Poor standards of research due to computational expense
- E.g., Zoph & Le (2017) did not compare to other methods
  - Spoiler: it turns out that Bayesian optimization with random forests (SMAC) is much better than RL
- To fix this, we created a tabular surrogate benchmark
  - Small cell search space with 423k unique architectures for CIFAR-10
  - Evaluated each of them & stored result in a table
  - Result: you can now quickly evaluate NAS algorithms on your laptop
  - You can also run multi-fidelity algorithms
    - We evaluated for each of four different epoch budgets
    - We also performed three repeats of each experiment
- Just introduced, but already heavily taken up by the community
- Cannot evaluate weight sharing or weight inheritance algorithms

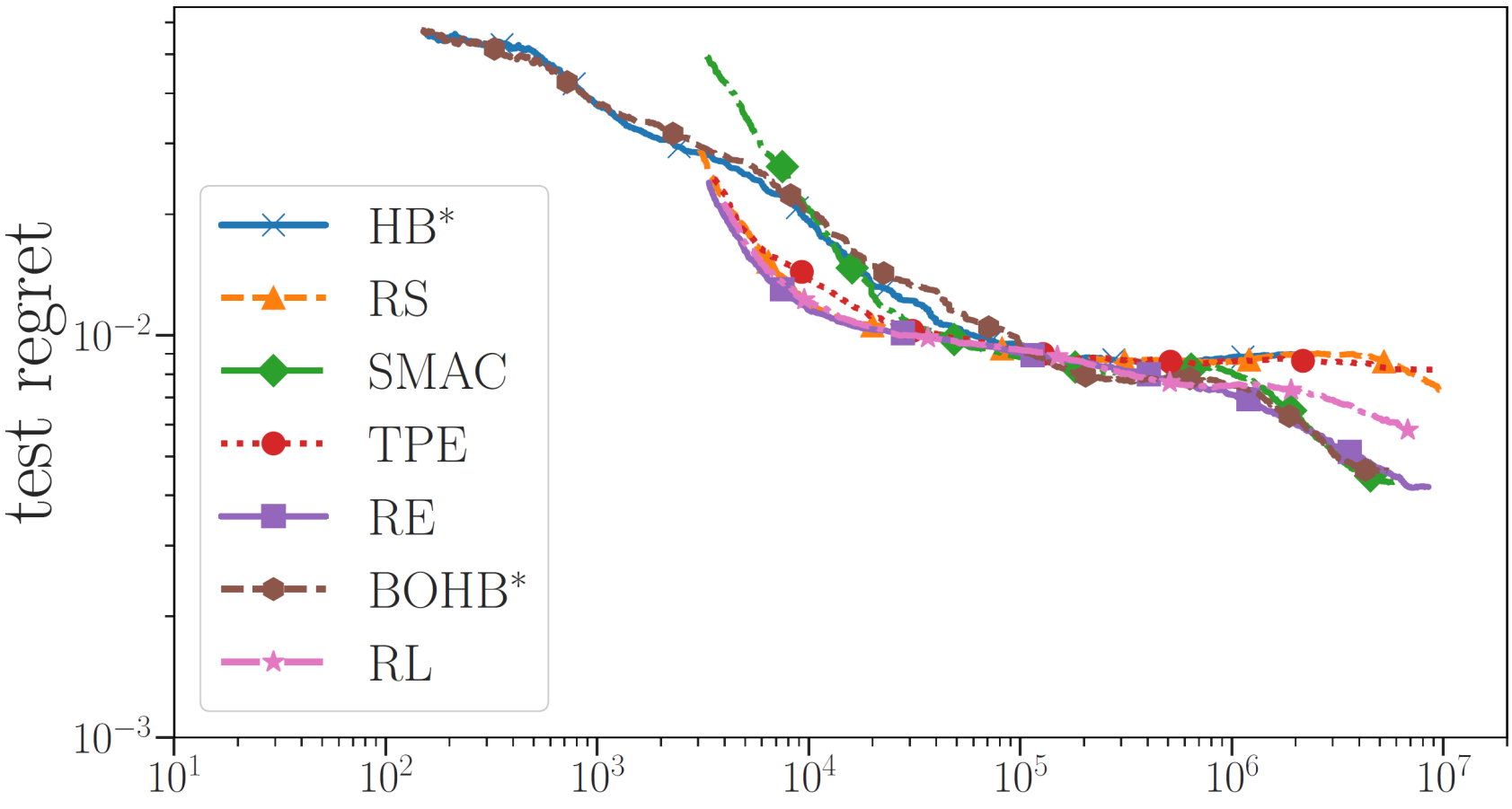
NAS-Bench-101: Search Space Details

- Cells are directed acyclic graphs with 3 operations: 3x3 convolution, 1x1 convolution, 3x3 max-pool
- Limited number of vertices & edges to limit to 423k models
  - Maximum of 7 vertices, and 9 edges (out of 21)
  - Architectures with > 9 active edges are invalid



NAS-Bench-101: Results

- Evaluated RL, regularized evolution (RE), random search (RS), Hyperband (HB), TPE, SMAC, BOHB
- Test errors: RE, SMAC, BOHB $<$ RL $<$ RS & HB



Planned additional benchmarks

- Nas-Bench-201
  - Standard DARTS space, without any limitations
    - Drop restriction to exhastive evaluations
    - Only tiny fraction evaluated
    - But 1M evaluations in, e.g., 50 dimensions, should make performance predictable
  - Include hyperparameters in the space $\rightarrow$ their choice is very important
  - Pytorch instead of Tensorflow (thus probably not with Google)
- HPOlib 2.0
  - Multi-fidelity benchmarks
  - Multi-task benchmarks (several datasets)

Learning Goals

After this lecture, you will be able to . . .

- describe several ways of speeding up over blackbox NAS
  - define network morphisms & explain how to use them to speed up NAS
  - explain various methods for extrapolating learning curves
  - explain various multi-fidelity Bayesian optimization methods
- discuss when and how to use NAS and HPO in practice
  - describe failure modes of DARTS
  - describe Auto-DispNet
- describe NAS-Bench-101, a benchmark for NAS