Automatic Machine Learning (AutoML)

Jie Tang

Tsinghua University

May 22, 2020

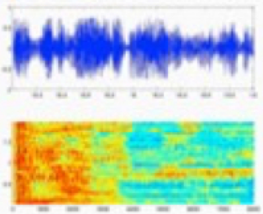
Overview

Successes of Deep Learning

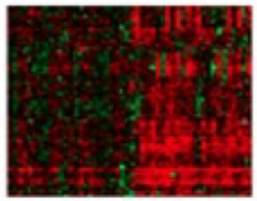
Images & Video

Text & Language

Speech & Audio

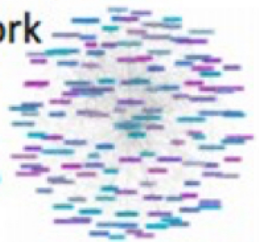


Gene Expression

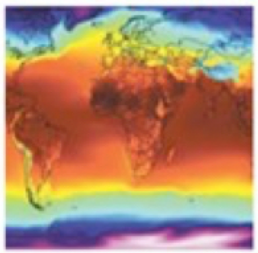


Product Recommendation

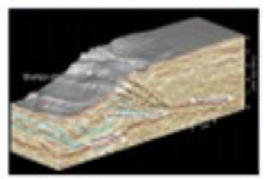
Relational Data/ Social Network



Climate Change

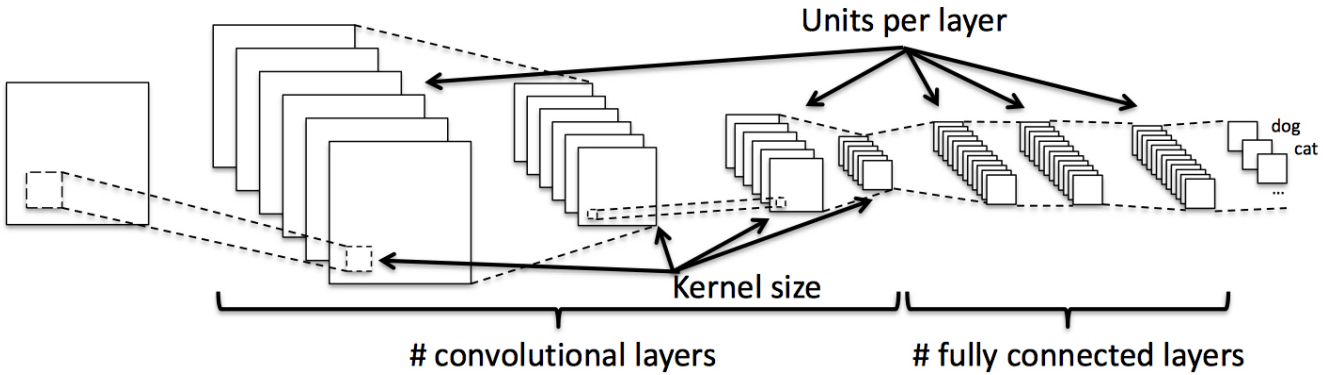


Geological Data



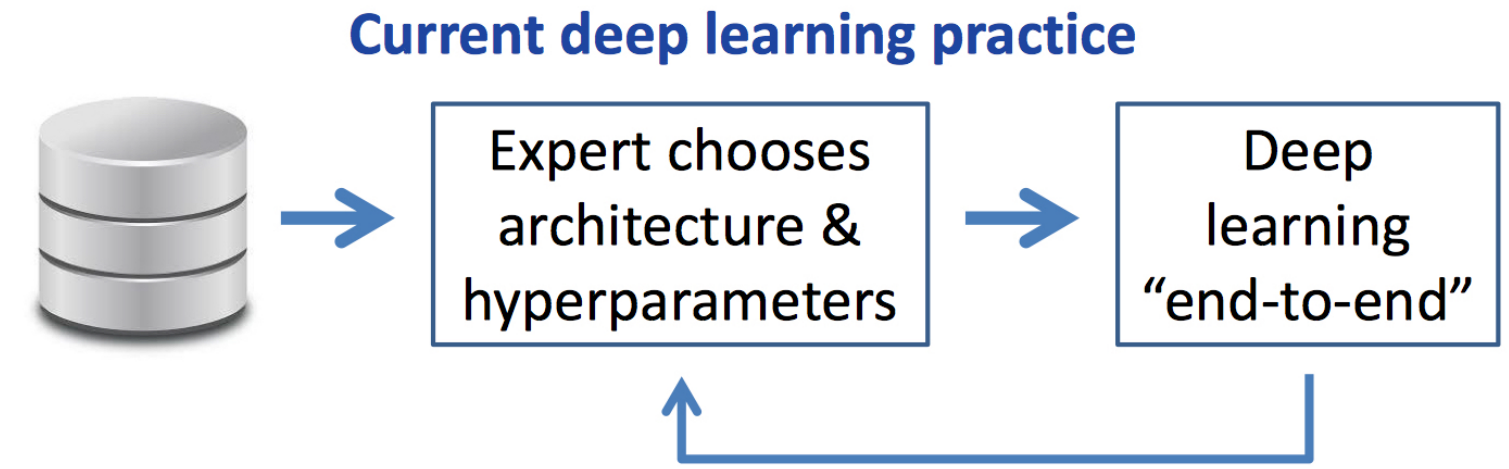
One Problem of Deep Learning

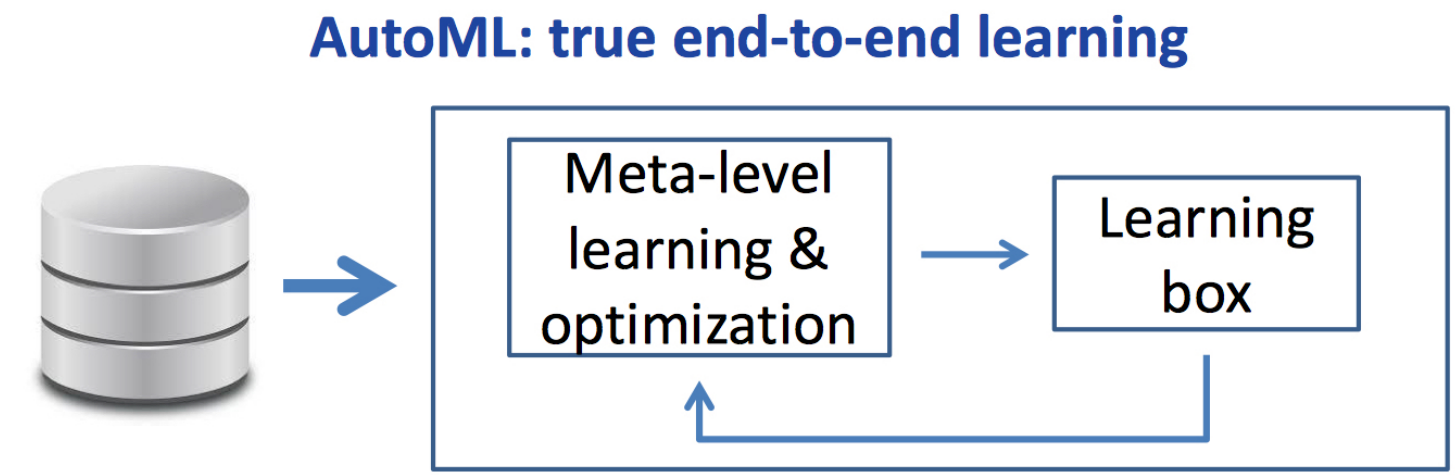
- Performance is very sensitive to many hyperparameters
  - Architectural hyperparameters



  - Optimization algorithm, learning rates, momentum, batch normalization, batch sizes, dropout rates, weight decay, data augmentation,...
  - Easily 20-50 design decisions
- A highly trained team of human experts is necessary: data scientists + domain experts

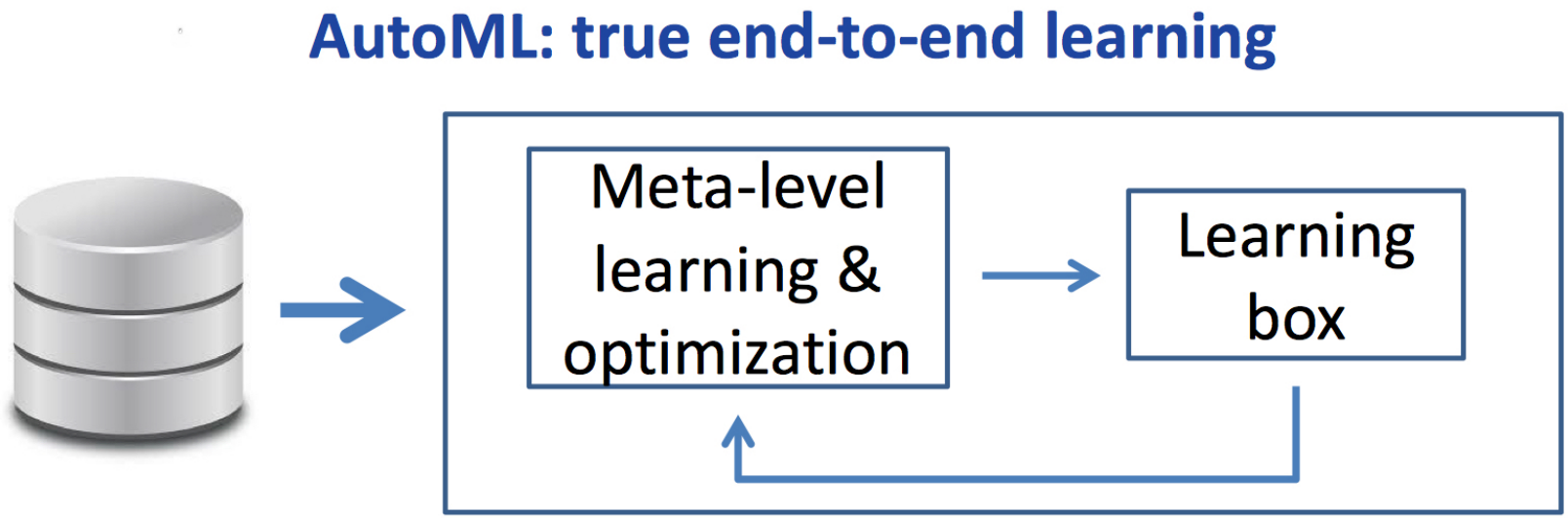
Deep Learning and AutoML



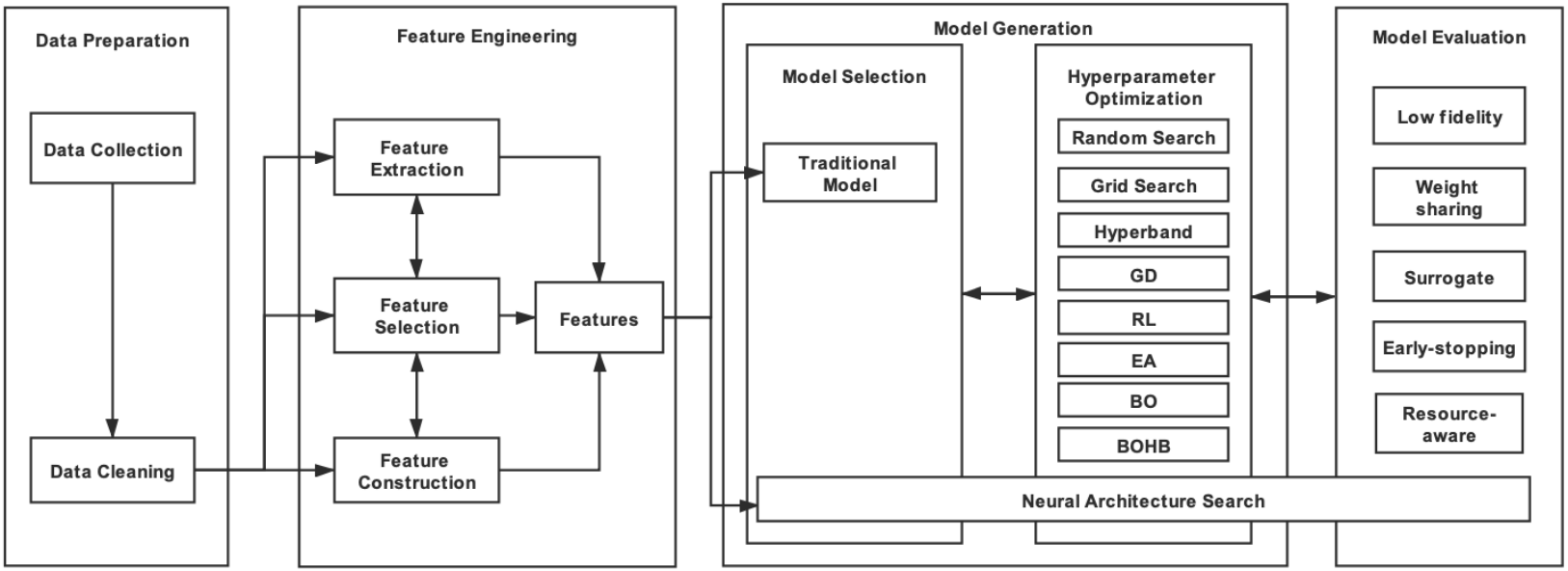


Learning box is not restricted to deep learning

- Traditional machine learning pipeline:
  - Clean & preprocess the data
  - Select / engineer better features
  - Select a model family
  - Set the hyperparameters
  - Construct ensembles of models
  - ...



An overview of AutoML pipeline



- Including Data Preparation, Feature Engineering, Model Generation/Evaluation ...
  - GD: Gradient Descent
  - RL: Reinforcement Learning
  - EA: Evolution-based Algorithms
  - BO: Bayesian Optimization
  - BOHB: Bayesian-Optimization-based Hyperband

Outline

Hyperparameter Optimization

Definition

Let

- $\lambda \in \Lambda$ be the hyperparameters of a ML algorithm A
- $\mathcal{L}\left(A_\lambda, D_{\text{train}}, D_{\text{valid}}\right)$ denotes the loss of A, using hyperparameters λ trained on D_{train} and evaluated on D_{valid}

The hyperparameter optimization (HPO) problem is to find a hyperparameter configuration λ^* that minimizes this loss:

$$\lambda^* \in \arg\min_{\lambda \in \Lambda} \mathcal{L}\left(A_\lambda, D_{\text{train}}, D_{\text{valid}}\right)$$

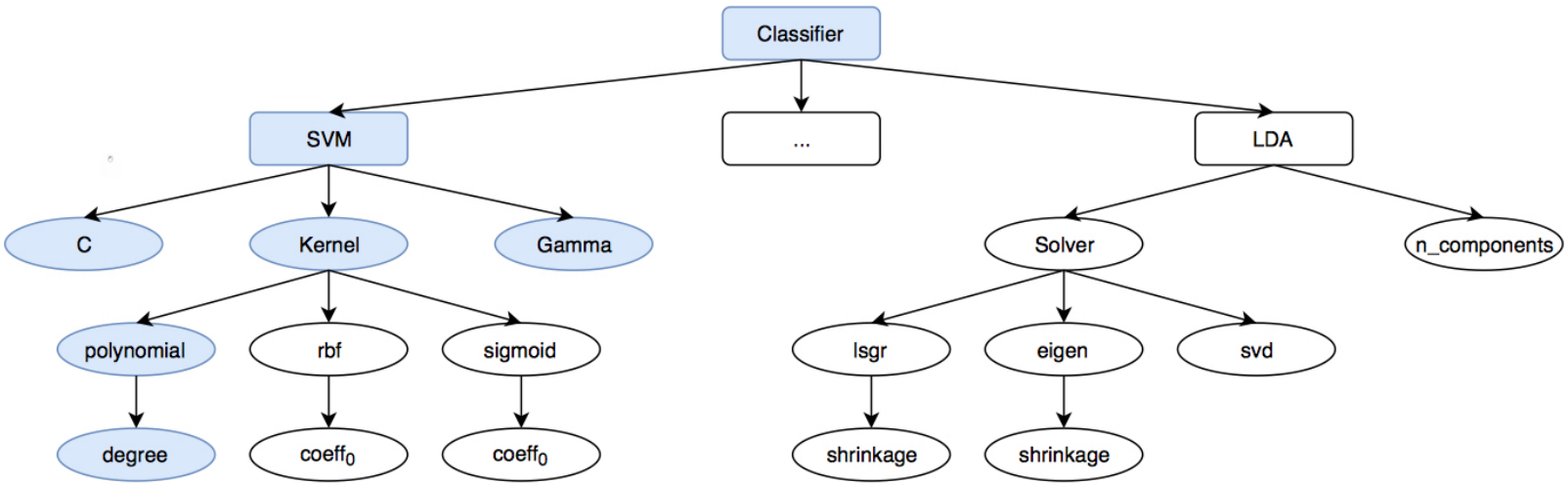
Types of Hyperparameters

- Continuous
  - Example: learning rate
- Integer
  - Example 1: #units in NN
  - Example 2: #neighbors in k-nearest neighbors
- Categorical
  - Finite domain, unordered
    - Example 1: algorithm $A \in \{\text{SVM, RF, NN}\}$
    - Example 2: activation function $\sigma \in \{\text{ReLU, sigmoid, tanh}\}$
    - Example 3: operator $\in \{\text{conv3}x\text{3}, \text{max pool}, \cdots\}$
    - Example 4: the splitting criterion used for decision trees
  - Special case: binary

Conditional hyperparameters

- Conditional hyperparameters B are only active if other hyperparameters A are set a certain way
  - Example 1:
    - A = choice of optimizer (Adam or SGD)
    - B = Adam's momentum hyperparameter (only active if A=Adam)
  - Example 2:
    - A= type of layer k (convolution, max pooling, fully connected, ...)
    - B = conv. kernel size of that layer (only active if A = convolution)
  - Example 3:
    - A = choice of classifier (RF or SVM)
    - B = SVM's kernel parameter (only active if A = SVM)

Conditional Hyperparameters Example



AutoML as Hyperparameter Optimization

Definition: Combined Algorithm Selection and Hyperparameter Optimization (CASH)

Let

- $\mathcal{A} = \{A^{(1)}, \ldots, A^{(n)}\}$ be a set of algorithms
- $\mathbf{\Lambda}^{(i)}$ denote the hyperparameter space of $A^{(i)}$, for $i = 1, \ldots, n$
- $\mathcal{L}(A^{(i)}_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$ denote the loss of $A^{(i)}$, using $\lambda \in \mathbf{\Lambda}^{(i)}$ trained on D_{train} and evaluated on D_{valid}.

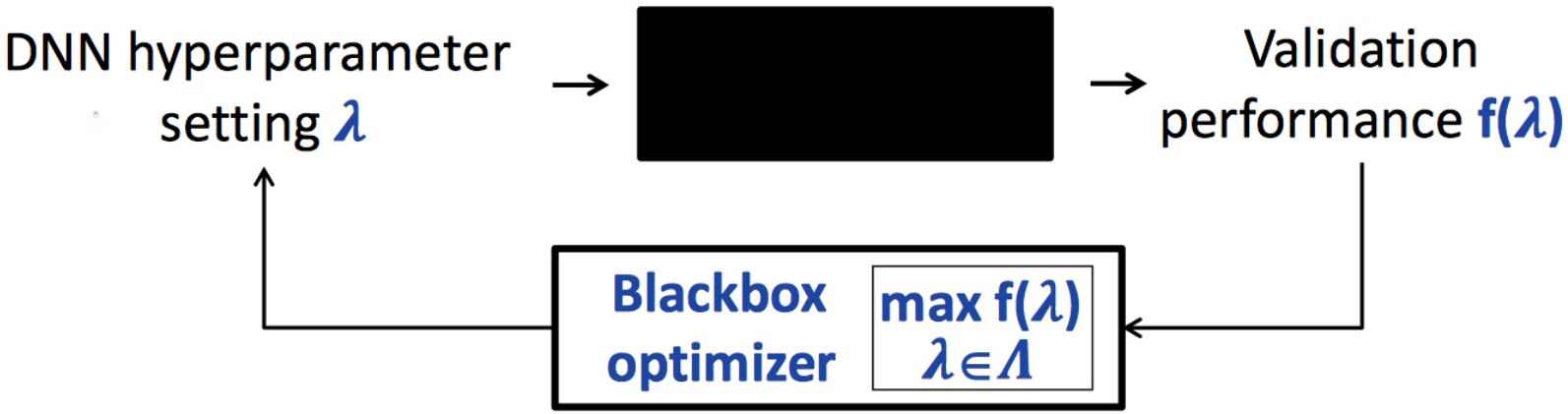
The Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem is to find a combination of algorithm $A^* = A^{(i)}$ and hyperparameter configuration $\boldsymbol{\lambda}^* \in \mathbf{\Lambda}^{(i)}$ that minimizes this loss:

$$A^*_{\boldsymbol{\lambda}^*} \in \underset{A^{(i)} \in \mathcal{A}, \boldsymbol{\lambda} \in \mathbf{\Lambda}^{(i)}}{\arg\min} \mathcal{L}(A^{(i)}_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$$

- CASH[1] = HPO + choice of algorithm

[1] Chris Thornton, et al. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In KDD 2013.

Blackbox Hyperparameter Optimization



- The blackbox function is expensive to evaluate
- sample efficiency is important

Grid Search

- Each continuous hyperparameter is discretized into k equidistant values
- For categorical hyperparameters each value is used
- Cartesian product of the discretized hyperparameters

$$\Lambda_{GS} = \lambda_{1:k_1}^{(1)} \times \lambda_{1:k_2}^{(2)} \times \cdots \times \lambda_{1:k_n}^{(n)}$$

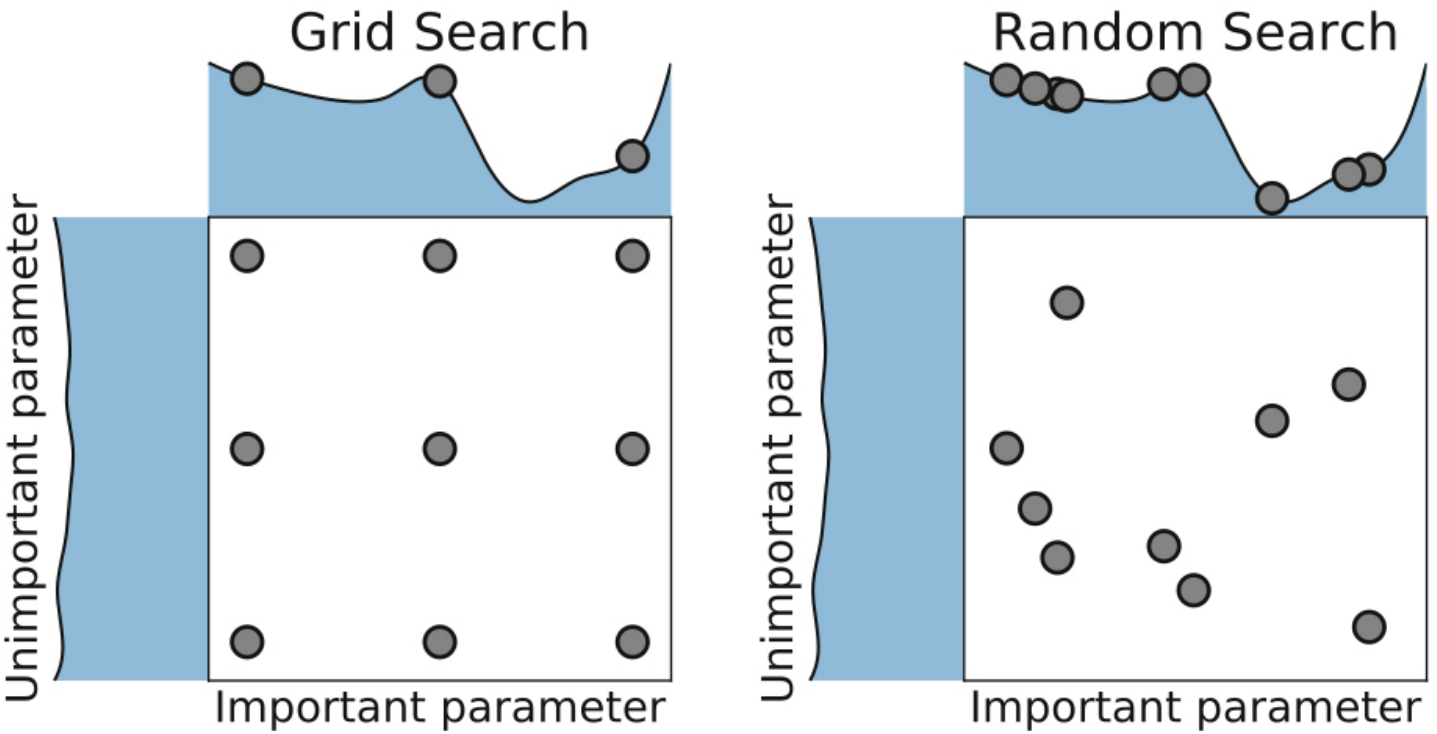
- Curse of dimensionality
- Does not exploit knowledge of well performing regions
  - Coarse grid + Finer grid

Random Search

- Converge faster than grid search
- Easier parallelization
- Flexible resource allocation
- Random search is a useful baseline
- Does not exploit knowledge of well performing regions
- Still very expensive

Grid Search and Random Search

- Random search works better than grid search when some hyperparameters are much more important than others



Bayesian Optimization

- An iterative algorithm
  - Fit a probabilistic model (e.g., Gaussian Process) to the function evaluations $\langle \lambda, f(\lambda) \rangle$
  - Acquisition function determines the utility of different candidate points, trading off exploration and exploitation
    - expected improvement (EI)

$$\mathbb{E}[\mathbb{I}(\boldsymbol{\lambda})] = \mathbb{E}\left[\max\left(f_{\min} - y, 0\right)\right]$$

    - Upper confidence bound (UCB)

$$a_{\mathrm{UCB}}(\lambda; \beta) = \mu(\lambda) - \beta\sigma(\lambda)$$

    - ...
- Popular since Mockus[1974]
  - Sample-efficient
  - Works when objective is nonconvex, noisy, has unknown derivatives, etc
  - Recent results [Srinivas et al, 2010; Bull 2011; de Freitas et al, 2016; Kawaguchi et al, 2016]

Illustration of Bayesian optimization

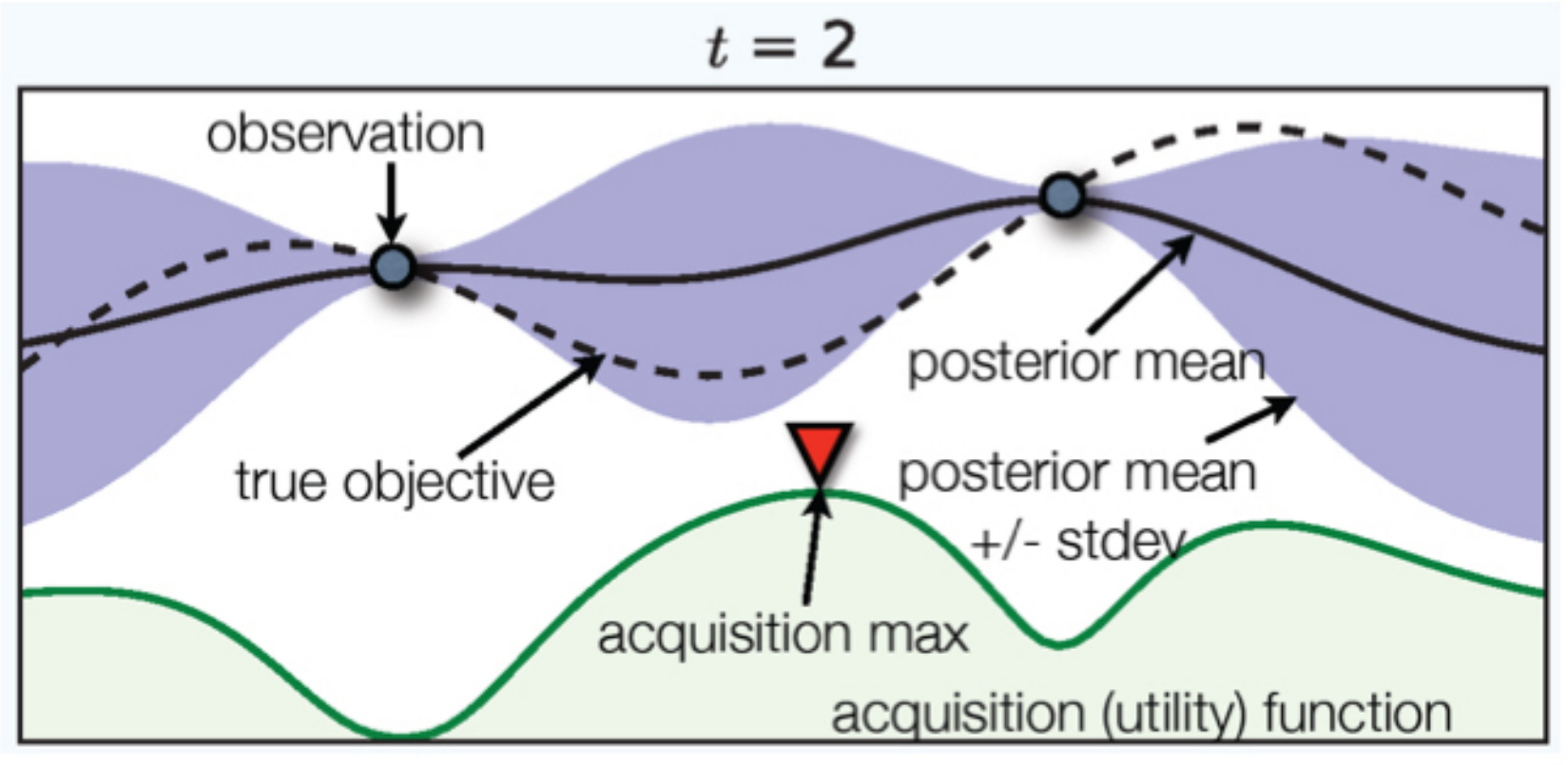


Illustration of Bayesian optimization

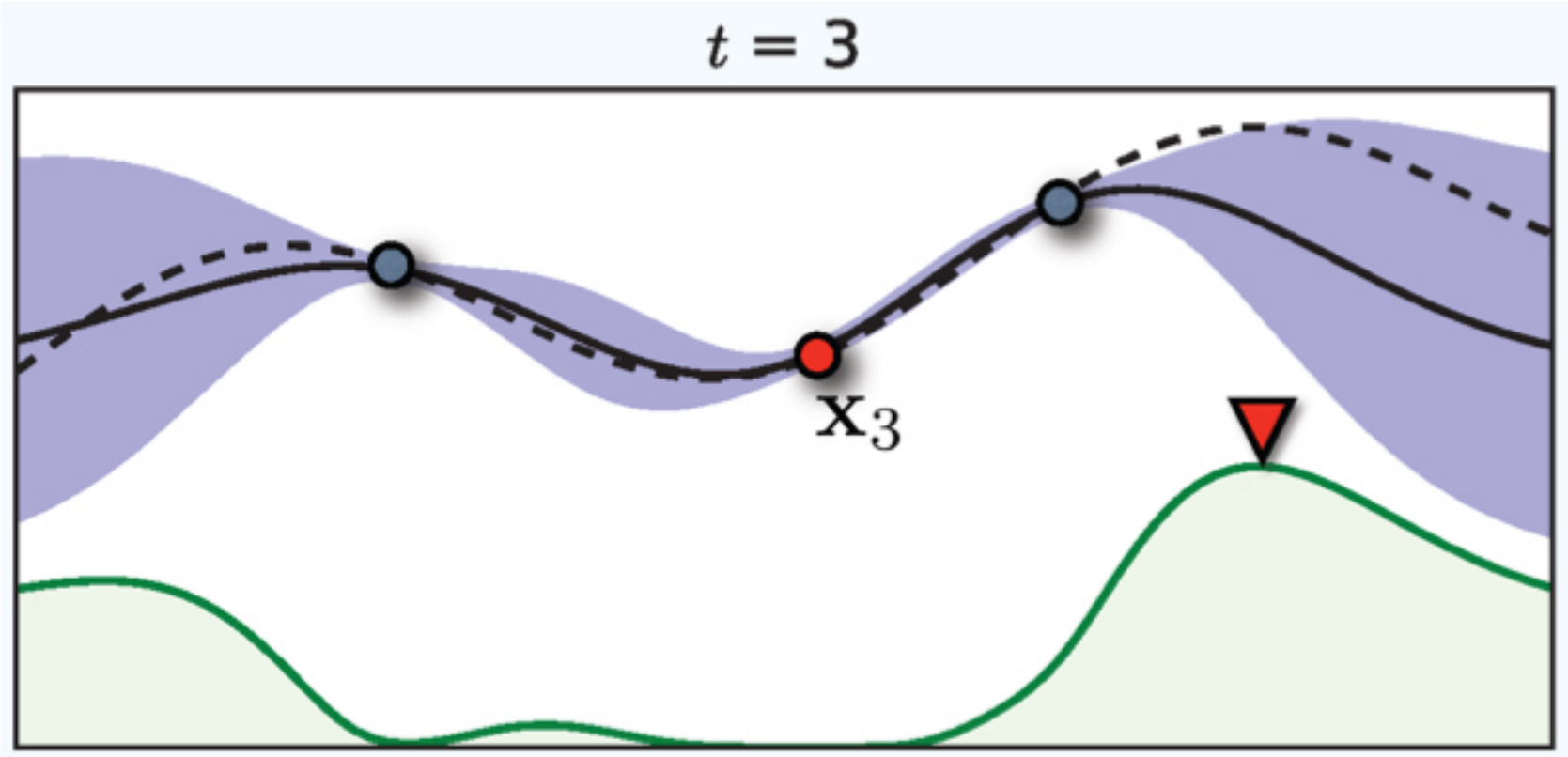
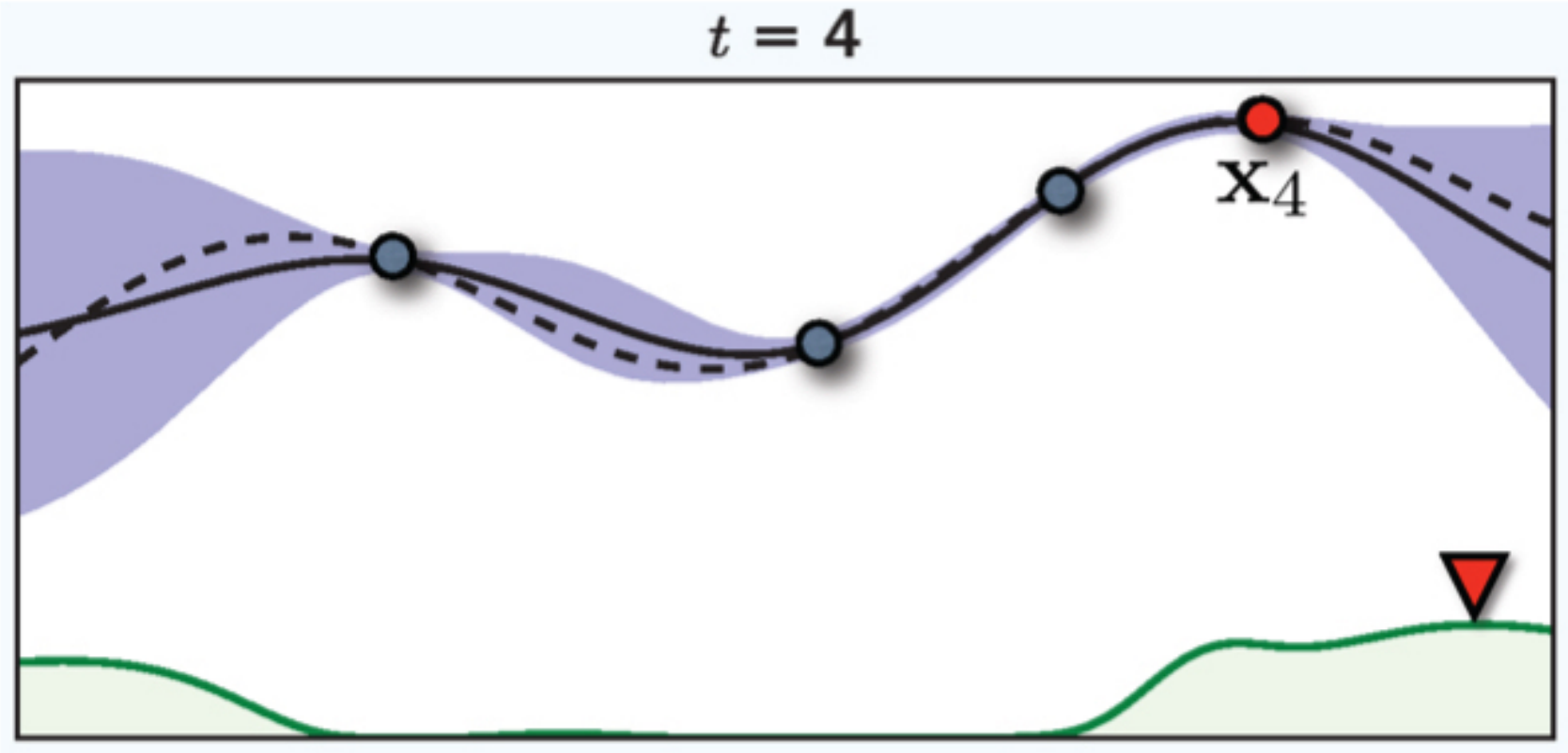


Illustration of Bayesian optimization



Example: Bayesian Optimization in AlphaGo

- "During the development of AlphaGo, its many hyperparameters were tuned with Bayesian optimization multiple times."
- "This automatic tuning process resulted in substantial improvements in playing strength. For example, prior to the match with Lee Sedol, we tuned the latest AlphaGo agent and this improved its win-rate from 50% to 66.5% in self-play games. This tuned version was deployed in the final match."
- "Of course, since we tuned AlphaGo many times during its development cycle, the compounded contribution was even higher than this percentage."

AutoML Challenges for Bayesian Optimization

- Problems for standard Gaussian Process (GP) approach:
  - scale cubically in the number of data points
  - poor scalability to high dimensions
  - Mixed continuous/discrete hyperparameters
  - Conditional hyperparameters
- Simple solution used in SMAC framework[2]: random forests

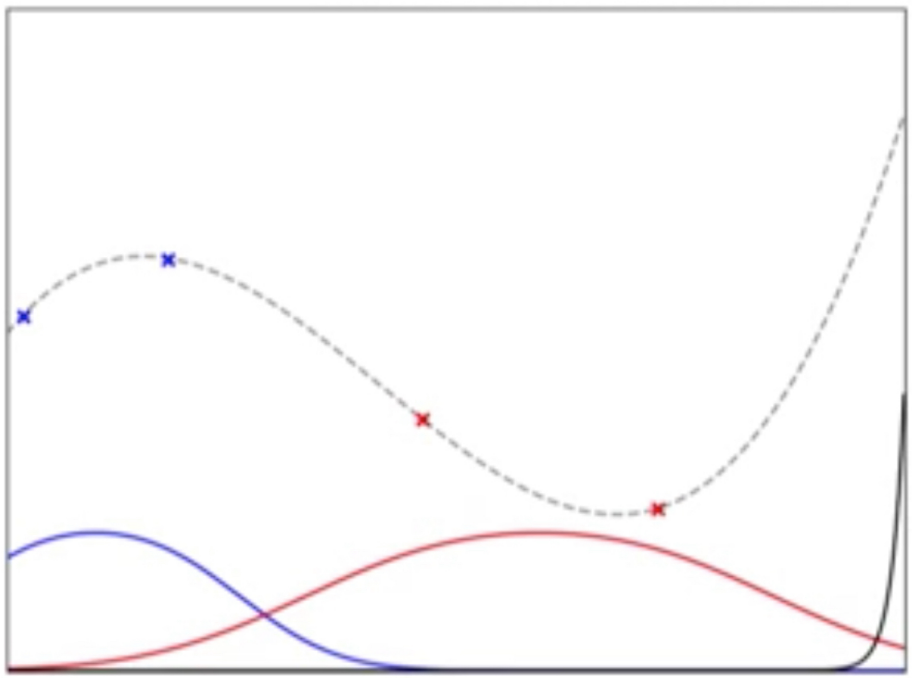
[2] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In: Coello C.A.C. (eds) Learning and Intelligent Optimization. LION 2011. Lecture Notes in Computer Science, vol 6683. Springer, Berlin, Heidelberg

Bayesian Optimization with Neural Networks

- The simplest way: NN as a feature extractor to preprocess inputs and then use the outputs of the final hidden layer as basis functions for Bayesian linear regression.[Snoek et al, ICML 2015]
- Fully Bayesian neural network trained with stochastic gradient Hamiltonian Monte Carlo.[Springenberg et al, NIPS 2016]
- A variational auto-encoder can be used to embed complex inputs into a real-valued vector such that a regular Gaussian process can handle it.[Xiaoyu Lu et al, ICML 2018]
- ...

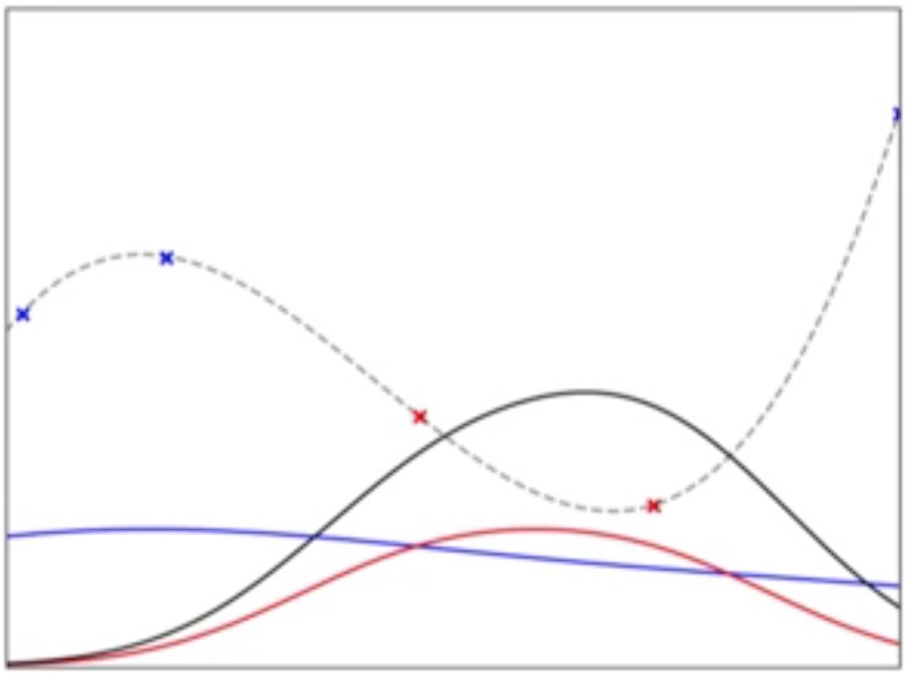
Tree of Parzen Estimators (TPE)

- Non-parametric KDEs for $p(\lambda \text{ is good})$ and $p(\lambda \text{ is bad})$, rather than $p(y|\lambda)$
- Acquisition function
  - **$p(\lambda \text{ is good})/p(\lambda \text{ is bad})$**
  - Equivalent to expected improvement
- Pros:
  - Efficient: $\mathrm{O}\left(\mathrm{N}^{*}\mathrm{d}\right)$
  - Parallelizable
  - Robust
- Cons:
  - Less sample-efficient than GPs



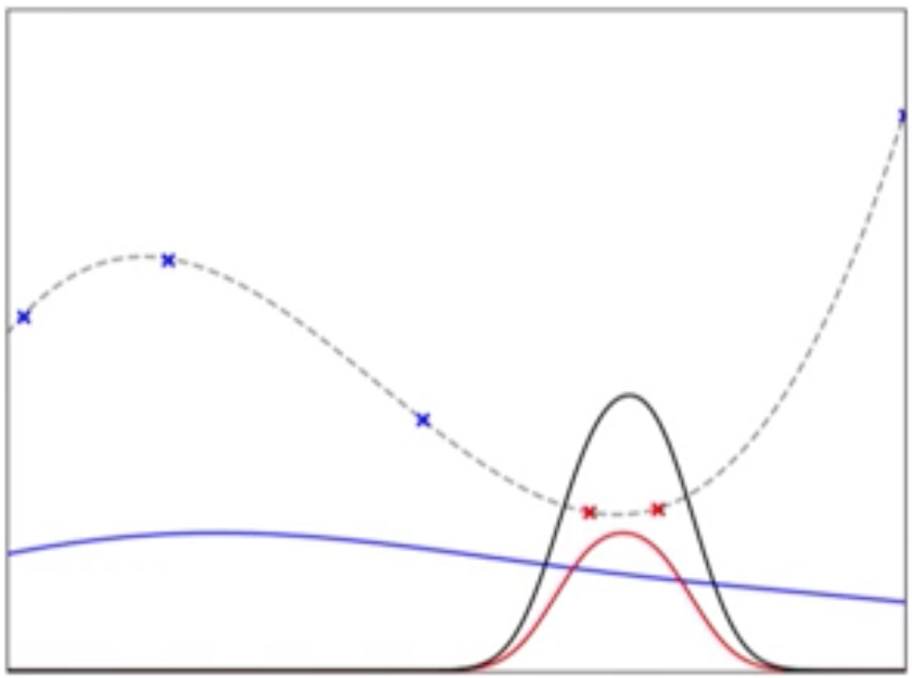
Tree of Parzen Estimators (TPE)

- Non-parametric KDEs for $p(\lambda \text{ is good})$ and $p(\lambda \text{ is bad})$, rather than $p(y|\lambda)$
- Acquisition function
  - **$p(\lambda \text{ is good})/p(\lambda \text{ is bad})$**
  - Equivalent to expected improvement
- Pros:
  - Efficient: $\mathrm{O}\left(\mathrm{N}^{*}\mathrm{d}\right)$
  - Parallelizable
  - Robust
- Cons:
  - Less sample-efficient than GPs



Tree of Parzen Estimators (TPE)

- Non-parametric KDEs for $p(\lambda \text{ is good})$ and $p(\lambda \text{ is bad})$, rather than $p(y|\lambda)$
- Acquisition function
  - $\boldsymbol{p(\lambda \text{ is good})/p(\lambda \text{ is bad})}$
  - Equivalent to expected improvement
- Pros:
  - Efficient: $\mathrm{O}\left(\mathrm{N}^{*}\mathrm{d}\right)$
  - Parallelizable
  - Robust
- Cons:
  - Less sample-efficient than GPs

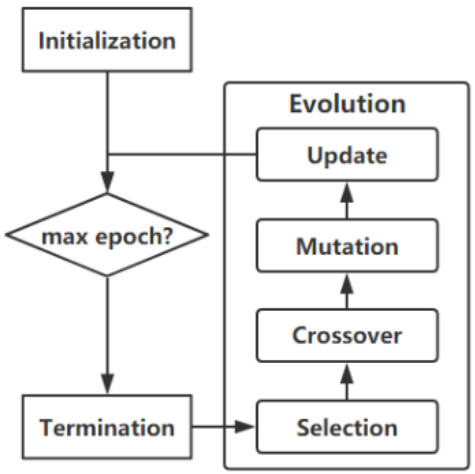


Population-based methods

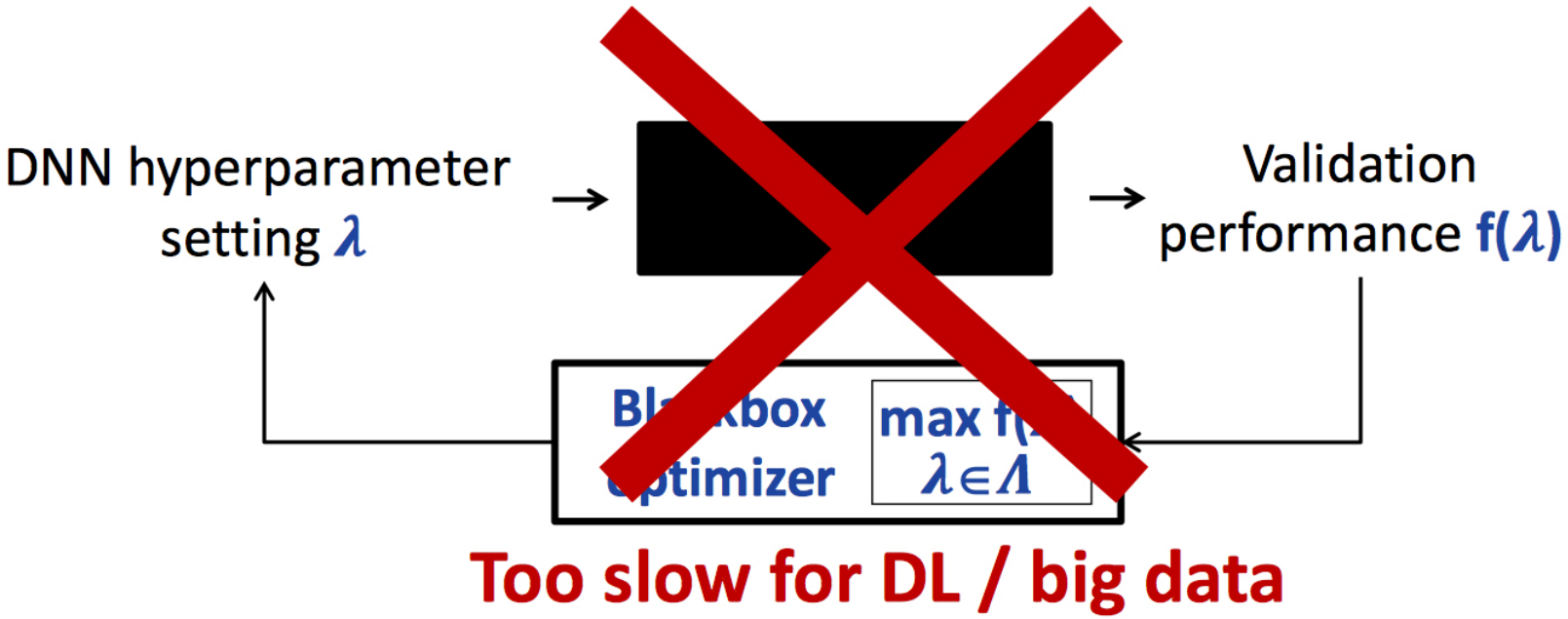
- population-based methods
  - maintain a population, i.e., a set of configurations
  - local perturbations (so-called mutations) and combinations of different members (so-called crossover) to obtain a new generation of better configurations
- genetic algorithms, evolutionary algorithms, particle swarm optimization...
- covariance matrix adaption evolutionary strategy (CMA-ES)
  - samples configurations from a multivariate Gaussian whose mean and covariance are updated in each generation based on the success of the population's individuals.
  - dominating the Black-Box Optimization Benchmarking (BBOB) challenge

Evolution-based Algorithm (EA)

- Selection
  - select a subset of candidates from population
- Crossover
  - combine selected candidates to generate a new offspring candidate
- Mutation
  - e.g. enable/disable a connection between layers, alter learning rate or filter size ...
- Update
  - discard some of the candidates.
  - e.g. bad-performing ones/oldest ones



Beyond Blackbox Hyperparameter Optimization



Hyperparameter Gradient Descent

- Formulation as bilevel optimization problem

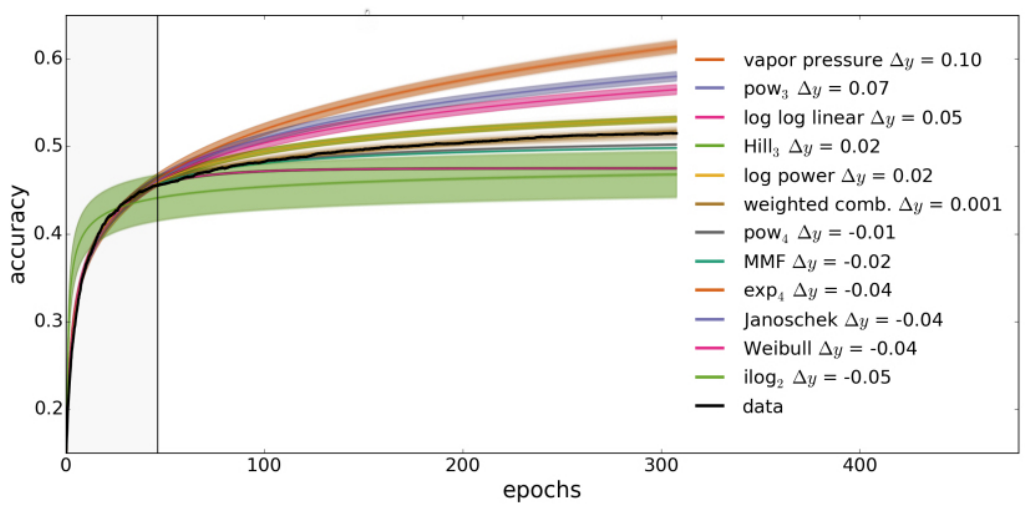
$$\begin{aligned} &\min_{\lambda} \mathcal{L}_{val}\left(w^*(\lambda), \lambda\right) \\ \text{s.t.} \quad &w^*(\lambda) = \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w, \lambda) \end{aligned}$$

- Derive through the entire optimization process [MacLaurin et al, ICML 2015]
- Interleave optimization steps [Luketina et al, ICML 2016]

Hyperparameter gradient step w.r.t. $\nabla_{\lambda}\mathcal{L}_{val}$
Parameter gradient step w.r.t. $\nabla_{w}\mathcal{L}_{train}$

Probabilistic Extrapolation of Learning Curves

- Humans have one advantage: when they evaluate a poor hyperparameter setting they can quickly detect (after a few steps of SGD) and terminate the corresponding evaluation to save time
- Mimic the early termination of bad runs using a probabilistic model that extrapolates the performance from the first part of a learning curve
- Speed up automatic hyperparameter optimization
- Parametric learning curve models [Domhan et al, IJCAI 2015]



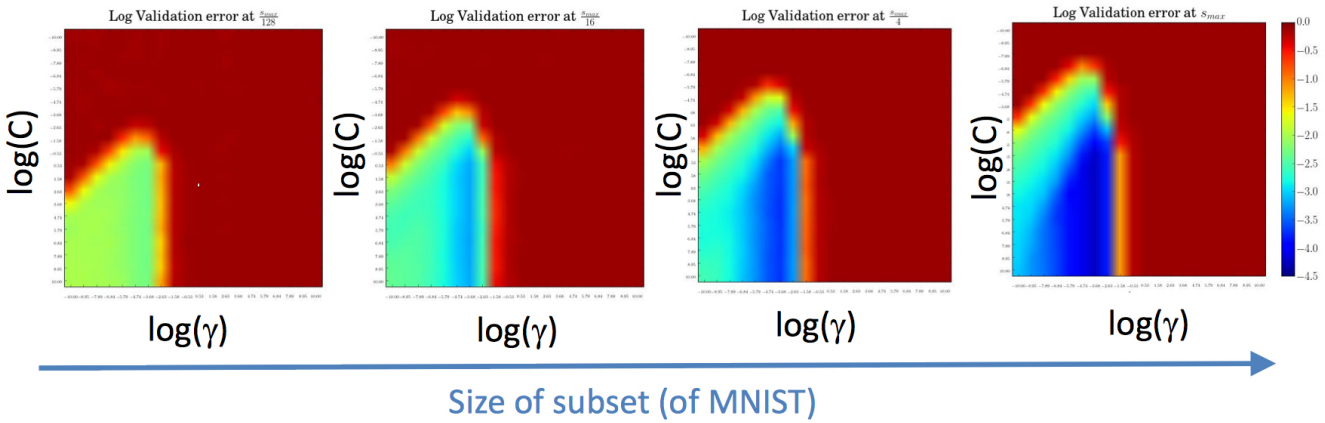
Reference name	Formula
vapor pressure	$\exp(a + \frac{b}{x} + c\log(x))$
pow$_3$	$c - ax^{-\alpha}$
log log linear	$\log(a\log(x) + b)$
Hill$_3$	$\frac{y_{\max} x^\eta}{\kappa^\eta + x^\eta}$
log power	$\frac{a}{1+\left(\frac{x}{e^b}\right)^c}$
pow$_4$	$c - (ax + b)^{-\alpha}$
MMF	$\alpha - \frac{\alpha-\beta}{1+(\kappa x)^\delta}$
exp$_4$	$c - e^{-ax^\alpha + b}$
Janoschek	$\alpha - (\alpha - \beta)e^{-\kappa x^\delta}$
Weibull	$\alpha - (\alpha - \beta)e^{-(\kappa x)^\delta}$
ilog$_2$	$c - \frac{a}{\log x}$

Multi-Fidelity Optimization

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

Multi-fidelity Optimization

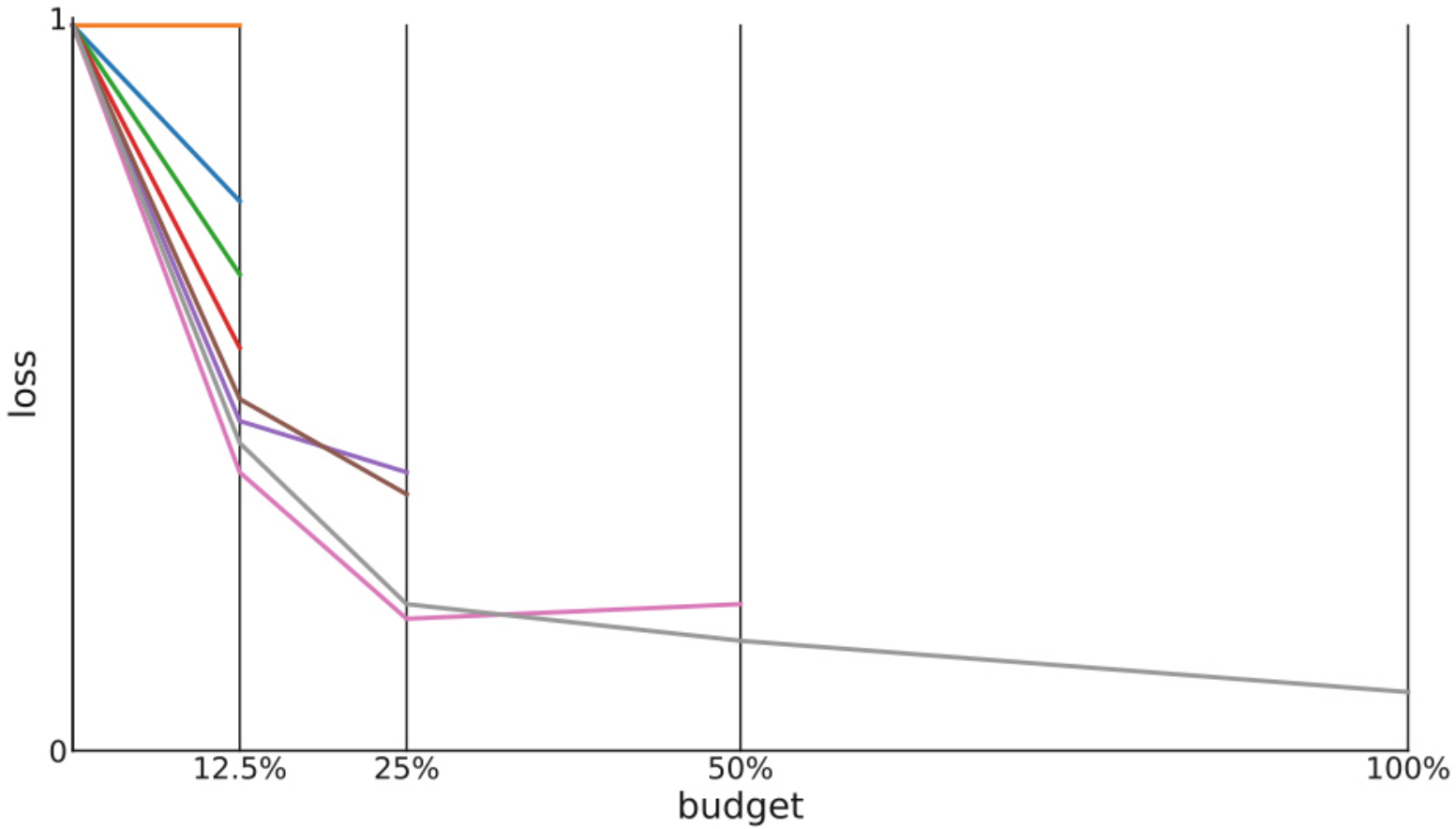
- Make use of cheap low-fidelity evaluations
  - E.g., subsets of the data (here: SVM on MNIST)



  - Many cheap evaluations on small subsets
  - Few expensive evaluations on the full data
  - Up to 1000x speedups [Klein et al, AISTATS 2017]

Successive Halving (SH)

- For a given initial budget, query all algorithms for that budget; then, remove the half that performed worst, double the budget and successively repeat until only a single algorithm is left.



Hyperband

- SH suffers from budget-vs-number of configurations trade off
  - try many configurations and only assign a small budget to each
    - may prematurely terminate good configurations
  - try only a few and assign them a larger budget.
    - may run poor configurations too long and thereby wasting resources

Algorithm 1: HYPERBAND algorithm for hyperparameter optimization.

input : R, η (default $\eta = 3$)
initialization : $s_{\max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\max} + 1)R$
for $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ **do**
  $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$, $r = R\eta^{-s}$
  // begin SUCCESSIVEHALVING with (n, r) inner loop
  $T =$ `get_hyperparameter_configuration`(n)
  for $i \in \{0, \ldots, s\}$ **do**
    $n_i = \lfloor n\eta^{-i} \rfloor$
    $r_i = r\eta^i$
    $L = \{$`run_then_return_val_loss`$(t, r_i) : t \in T\}$
    $T =$ `top_k`$(T, L, \lfloor n_i/\eta \rfloor)$
  end
end
return *Configuration with the smallest intermediate loss seen so far.*

Hyperband

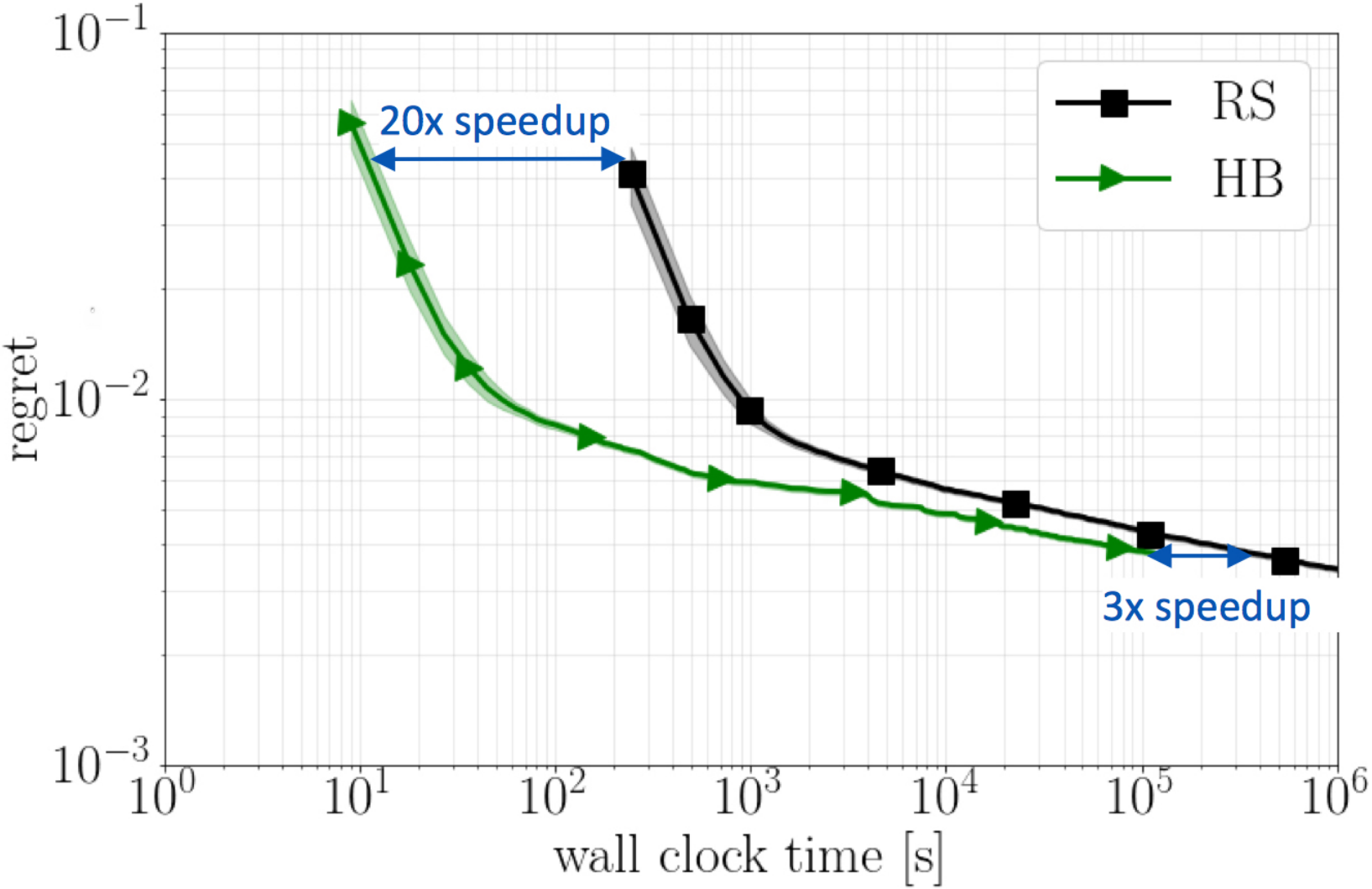
- Hyperhand
  - the outer loop iterates over different values of n and r (lines 1-2)
  - the inner loop invokes Successive Halving for fixed values of n and r (lines 3-9)

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

BOHB: Bayesian Optimization & Hyperband

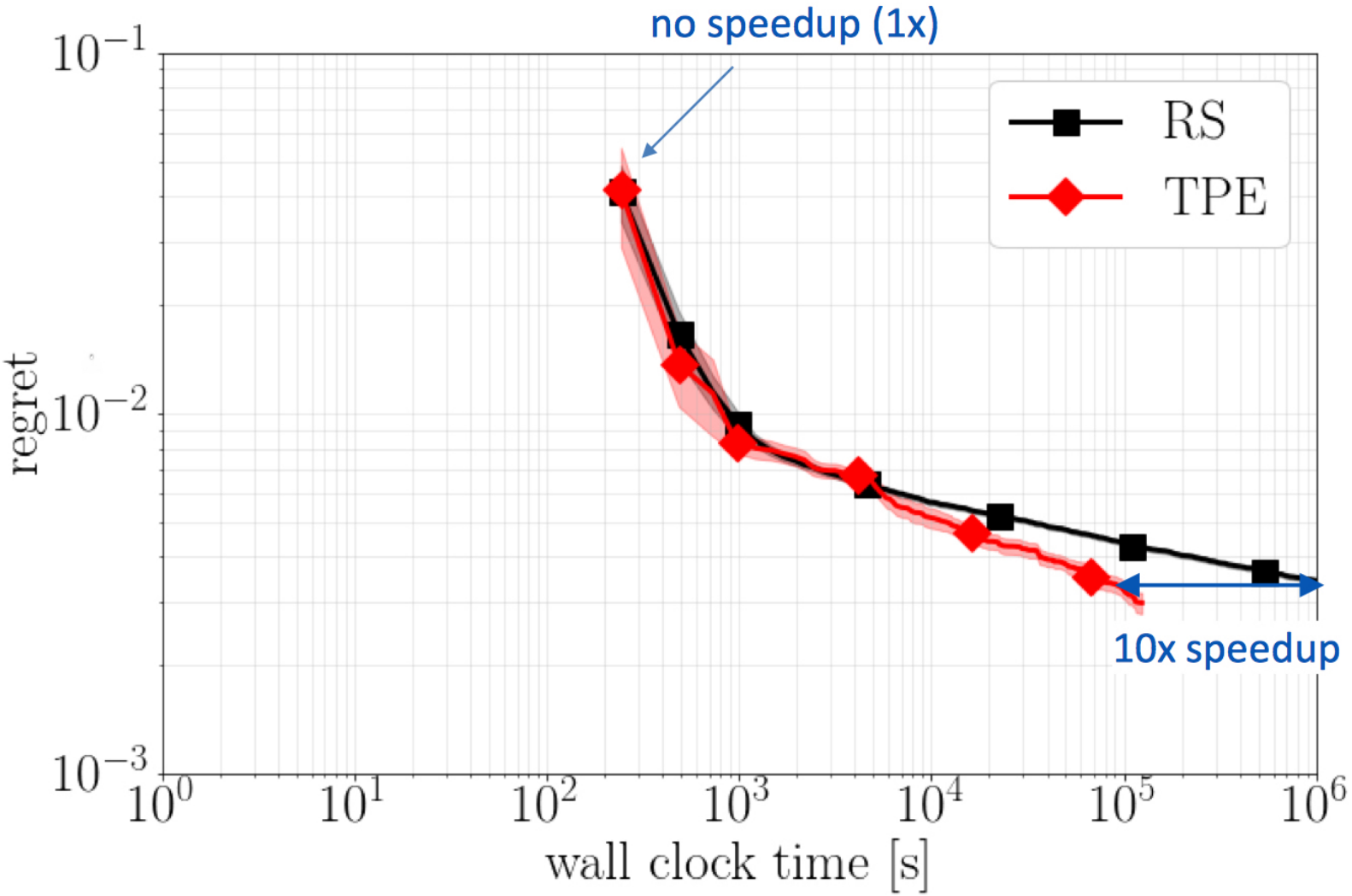
- Combining the best of both worlds in BOHB
  - Bayesian optimization
    - for choosing the configuration to evaluate
    - strong final performance (good performance in the long run by replacing HyperBand's random search by Bayesian optimization)
  - Hyperband
    - for deciding how to allocate budgets
    - strong anytime performance (quick improvements in the beginning by using low fidelities in HyperBand)

Hyperband vs. Random Search



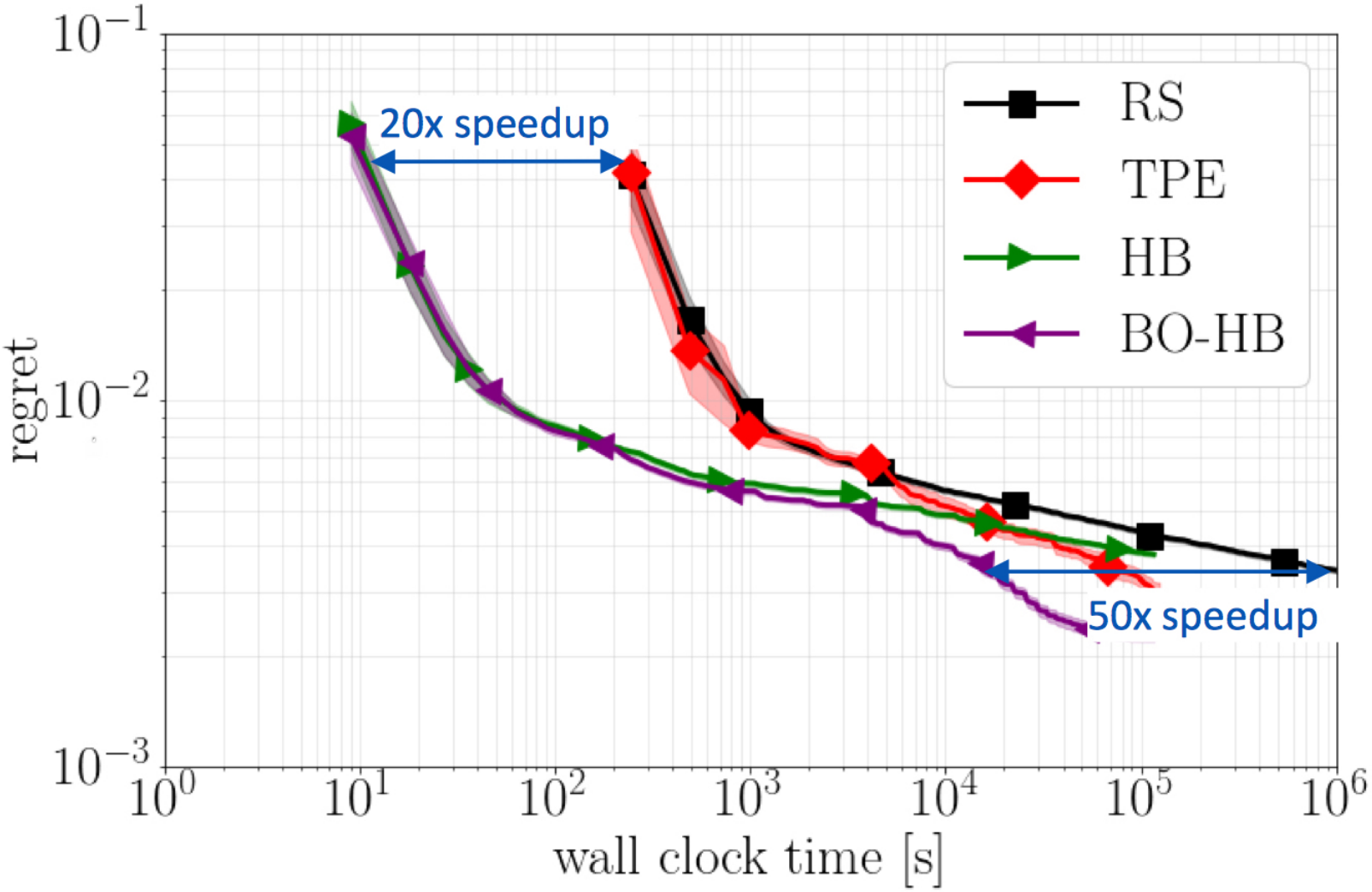
Biggest advantage: much improved anytime performance

Bayesian Optimization vs. Random Search



Biggest advantage: much improved final performance

Combining Bayesian Optimization & Hyperband



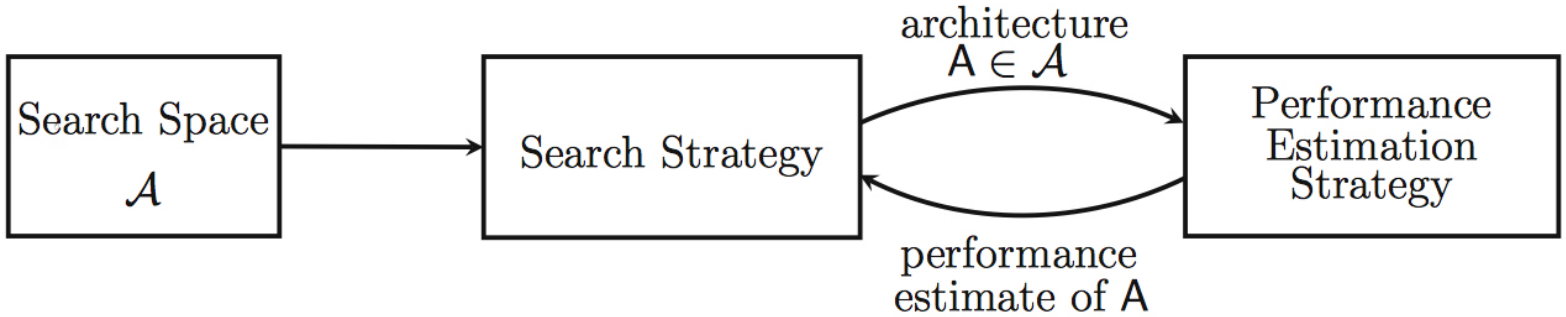
Best of both worlds: strong anytime and final performance

HPO Tools

- If you have access to multiple fidelities
  - BOHB
  - Combines the advantages of TPE and Hyperband
- If you do not have access to multiple fidelities
  - Low-dim, continuous: Gaussian Process-based BO (e.g., Spearmint)
  - High-dim, categorical, conditional: SMAC or TPE
  - CMA-ES
- Open-source AutoML tools based on HPO: Auto-WEKA, Hyperopt-sklearn, Auto-sklearn, TPOT, H2O AutoML...

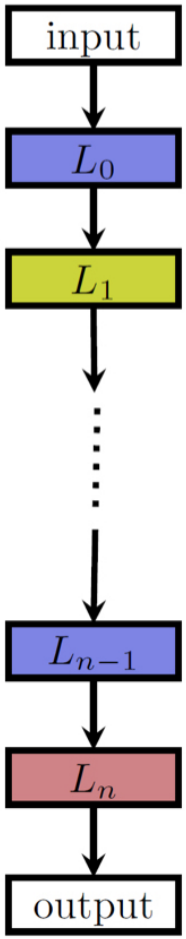
Outline

Neural Architecture Search

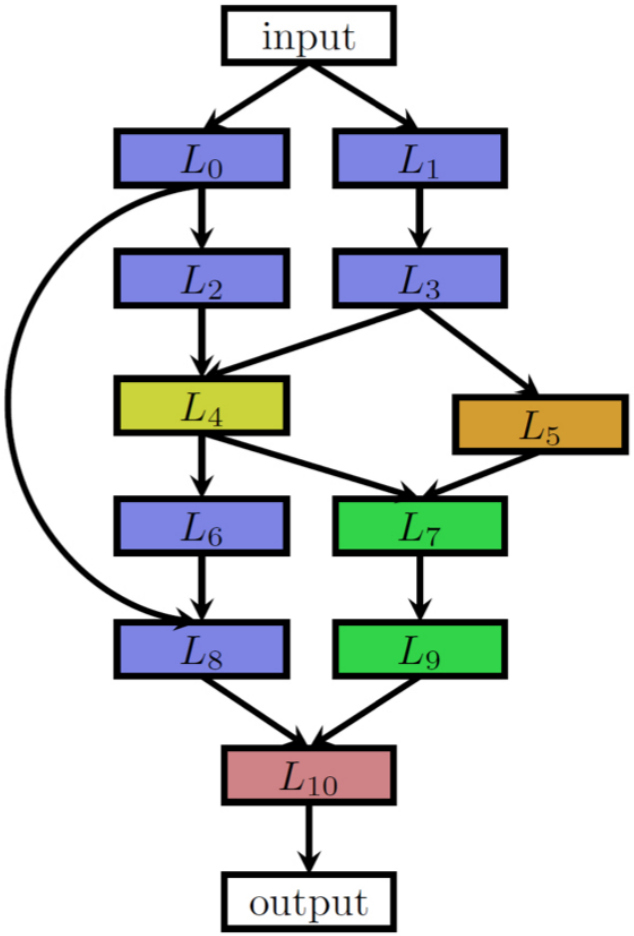


- A search strategy selects an architecture A from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

Basic Neural Architecture Search Spaces

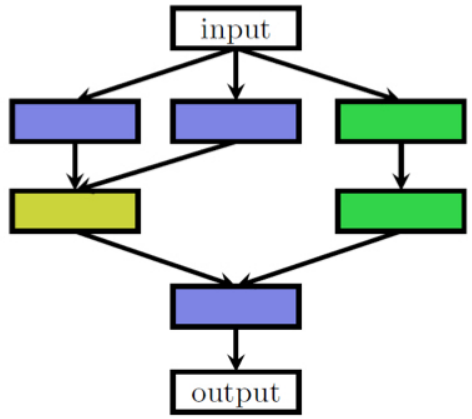


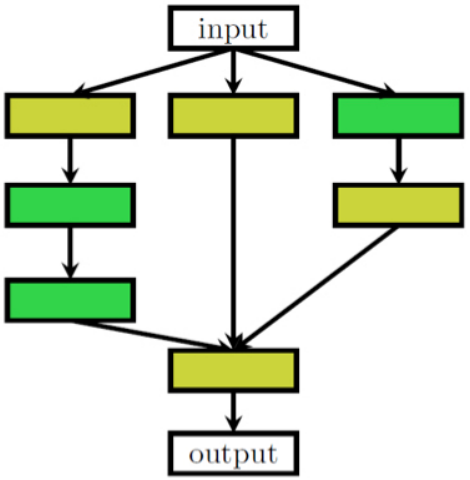
Chain-structured space
(different colours:
different layer types)



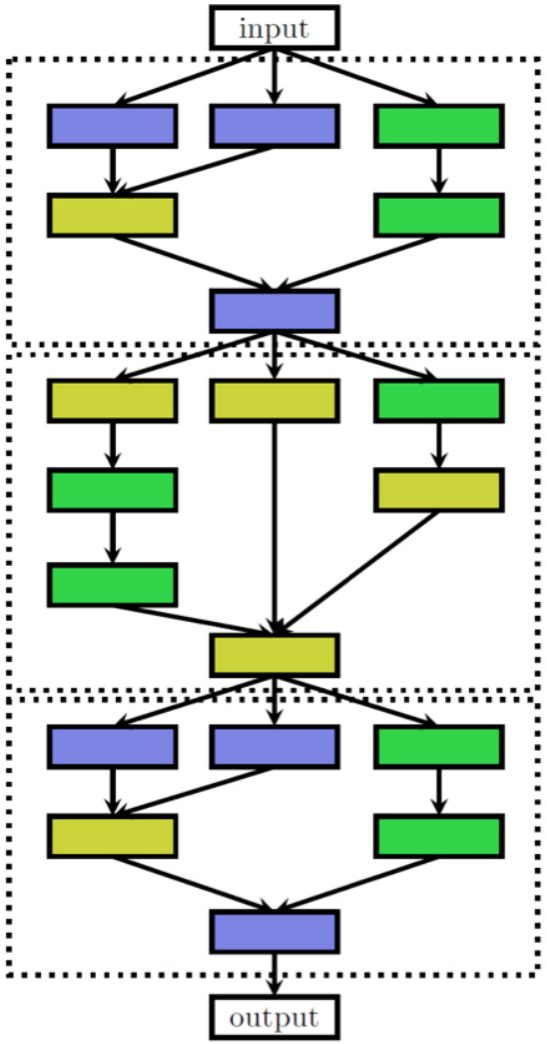
More complex space
with multiple branches
and skip connections

Cell Search Spaces



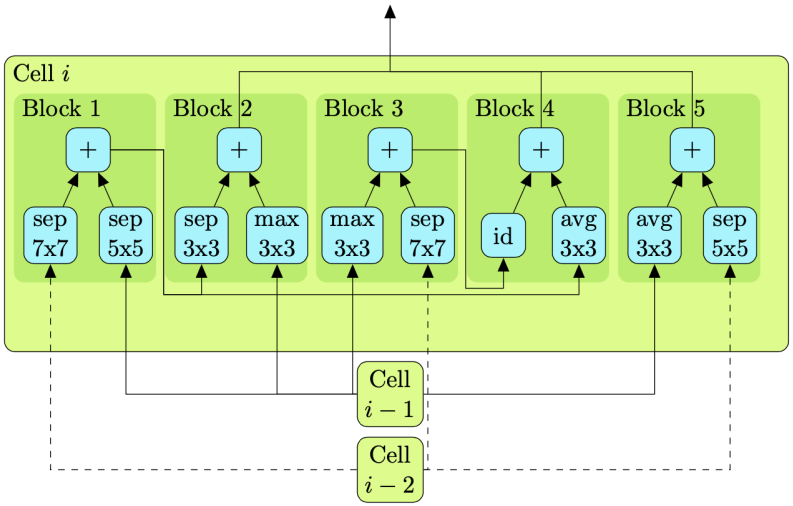


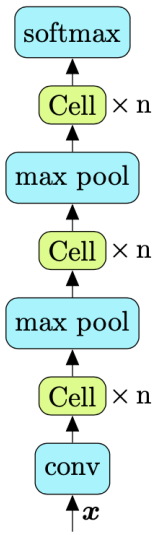
Two possible cells

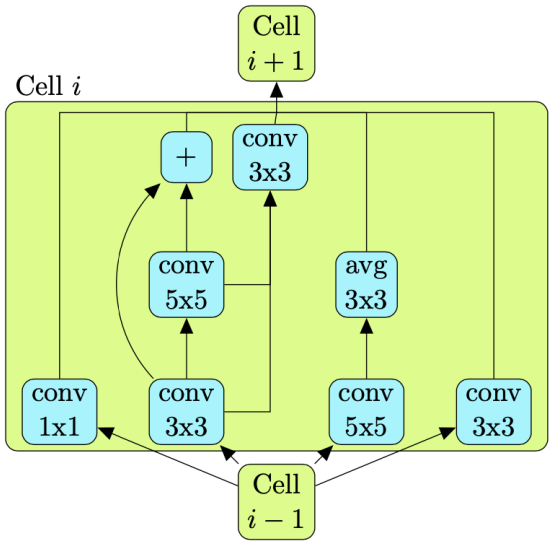


Architecture composed of stacking together individual cells

Cell Search Spaces







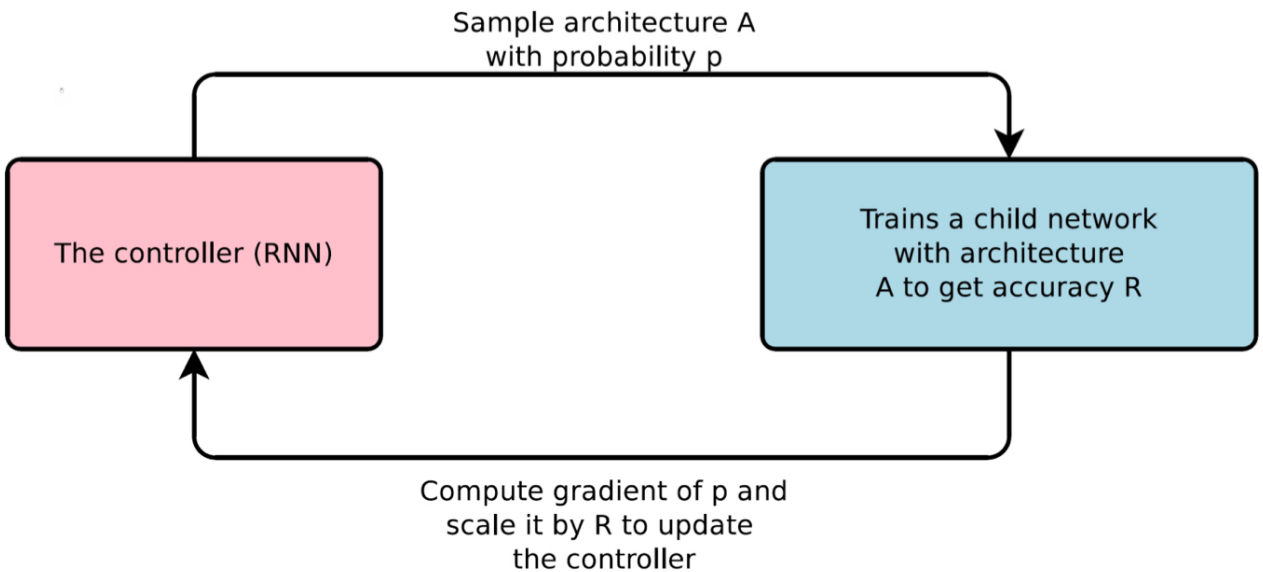
Cell of NASNet [Zoph, 2018]

BlockQNN cell [Zhong, 2018]

- Architectures are easily transferred across datasets.
- Argues:
  - [Tan, 2018]: Conventional cell spaces can't provide much diversity.
  - [Hu, 2018]: Comparable performance to cell-searched architectures can be easily arrived at, with an appropriate initial architecture.

Reinforcement Learning

- NAS became a mainstream research topic in the machine learning community after NAS with Reinforcement Learning [Zoph& Le, ICLR 2017]
  - State-of-the-art results for CIFAR-10, Penn Treebank
  - Large computational demands
    - 800 GPUs for 28 days, 12,800 architectures evaluated



- Different RL approaches differ in how they represent the agent's policy and how they optimize it

Neuroevolution

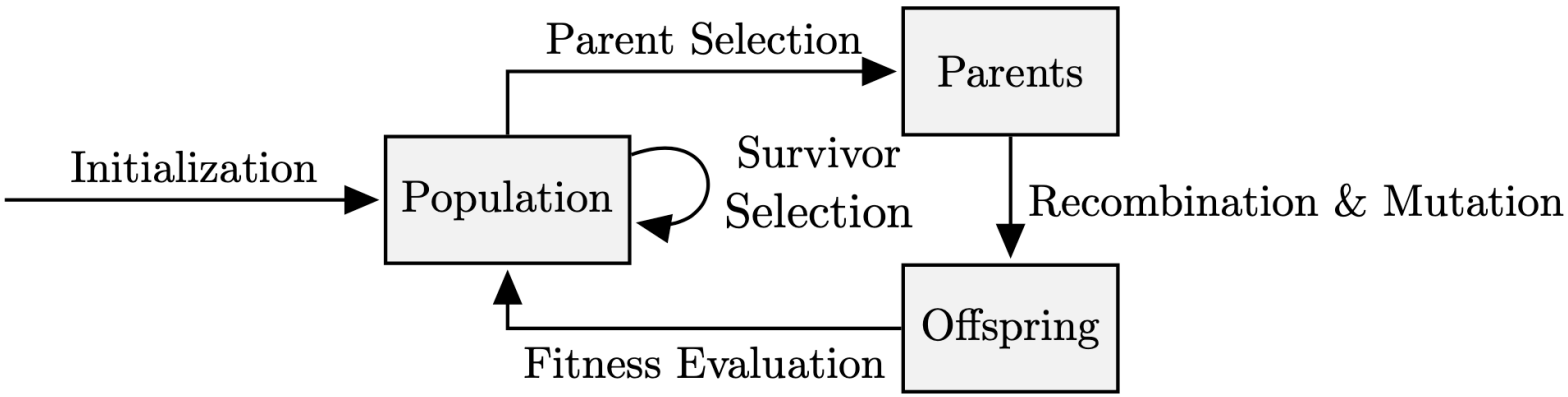
- Neuroevolution: use evolutionary algorithms for optimizing the neural architecture (already since the 1989 [3])
  - Optimize both architecture and weights with evolutionary methods
  - Use gradient-based methods for optimizing weights and solely use evolutionary algorithms for optimizing the neural architecture
    - scale to neural architectures with millions of weights for supervised learning tasks

[3] Miller, G., Todd, P., Hedge, S.: Designing neural networks using genetic algorithms. In: 3rd International Conference on Genetic Algorithms (ICGA'89) (1989)

Neuroevolution Algorithm

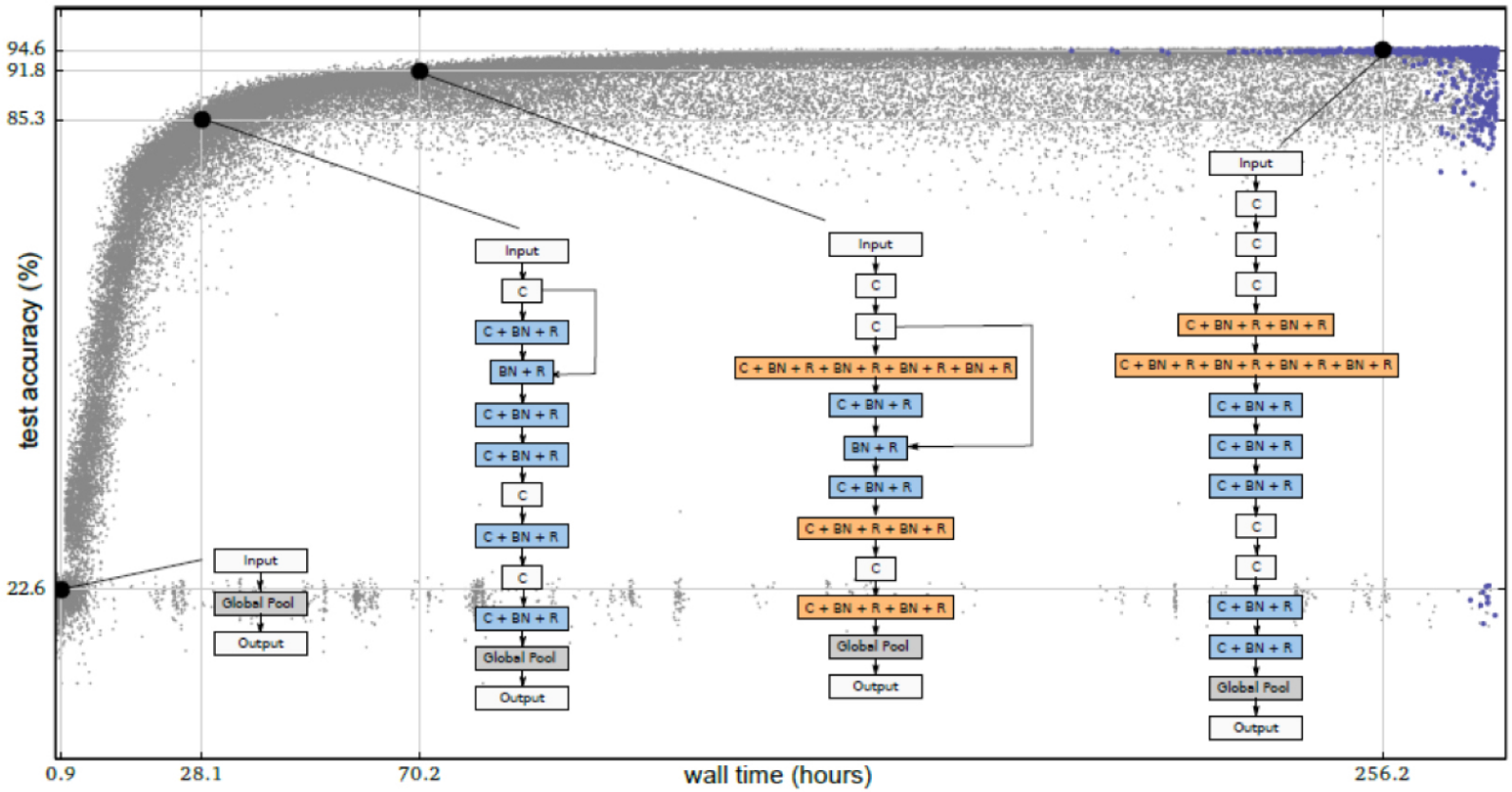
- a population of models, i.e., a set of (possibly trained) networks
- in every evolution step, at least one model from the population is sampled and serves as a parent to generate offsprings by applying mutations to it.
- mutation: local operation: adding or removing a layer, altering the hyperparameters of a layer, adding skip connections, altering training hyperparameters...
- After training the offsprings, their fitness (e.g., performance on a validation set) is evaluated and they are added to the population

Neuroevolution Algorithm



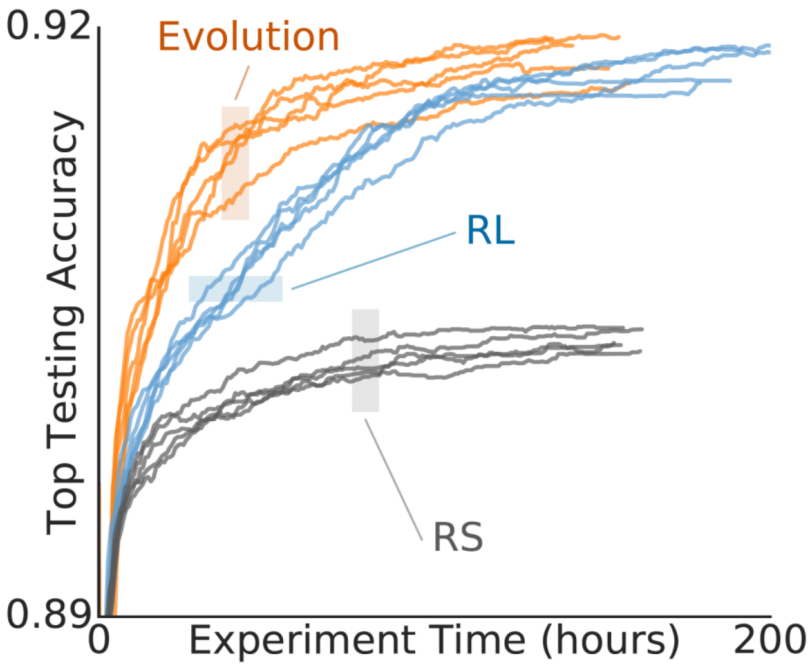
- Neuro-evolutionary methods differ in how they sample parents, update populations, and generate offsprings.

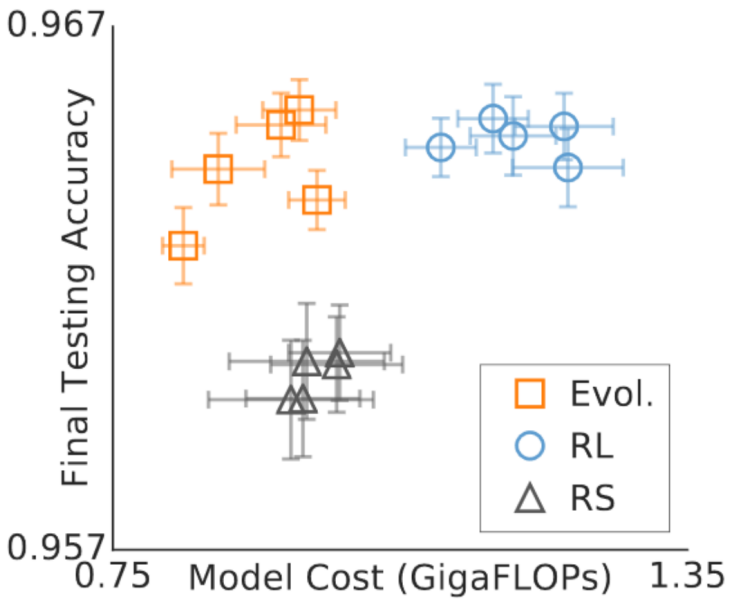
Neuroevolution Performance



Comparison of Evolution, RL and Random Search

- comparing RL, Evolution, and Random Search (RS)
  - RL and evolution perform equally well in terms of final test accuracy
  - Evolution has better anytime performance and finds smaller models





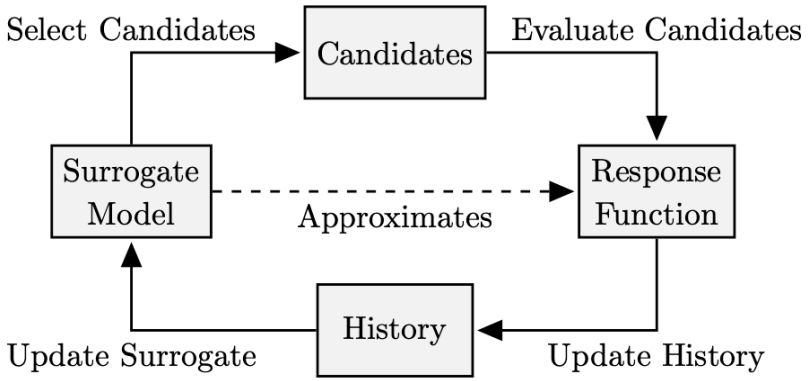
Surrogate Model-Based Optimization

- a general deep learning algorithm $\Lambda : D * A \rightarrow M$
  - D is the space of all datasets, A is the architecture (topology &all required properties) search space, and M is the space of all DL models.
  - Suppose α is an architecture from A, NAS is the task of finding the architecture α^* maximizing an objective function $\mathcal{O}$ on the validation set d_{valid}

$$\boldsymbol{\alpha}^* = \arg\max_{\boldsymbol{\alpha} \in A} \mathcal{O}\left(\Lambda\left(\boldsymbol{\alpha}, d_{\text{train}}\right), d_{\text{valid}}\right) = \arg\max_{\boldsymbol{\alpha} \in A} f\left(\boldsymbol{\alpha}\right)$$

    where f is called response function.
- SMBO: Use a surrogate model to predict/approximate the response function (architecture performance) of a given architecture.
- Surrogate model is trained on previous evaluated architectures, predicts on the architecture search space, to identify promising candidate architectures.

Surrogate Model-Based Optimization



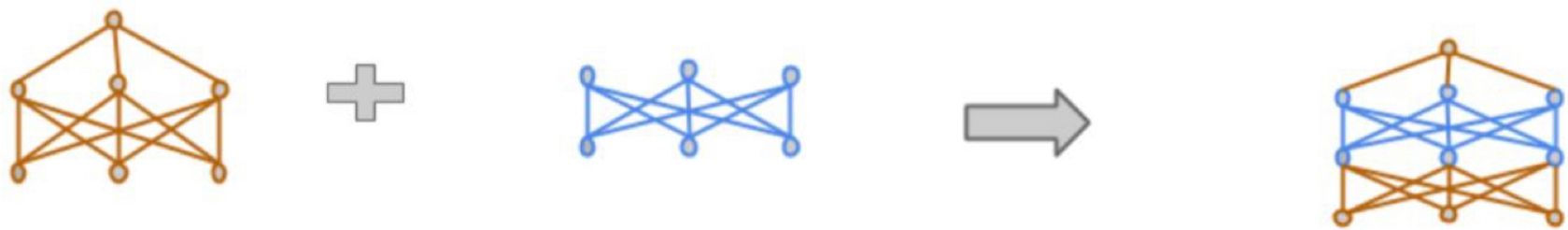
- Approaches for surrogate model-based optimization
  - Bayesian Optimization: predict response and uncertainty by a probabilistic surrogate model, obtain candidates by an acquisition function.[Kandasamy et al, NIPS 2018]
  - Architecture Encoding: train a surrogate model given by the encodings of architectures
    - Update cells and train an RNN/MLP surrogate model given by the encoding of the cell, to predict the validation accuracy of the architecture.[Liu, 2018a]
    - Jointly train a surrogate model with an autoencoder for the continuous architecture representation.[Luo, 2018]

Bayesian Optimization

- Joint optimization of a vision architecture with 238 hyperparameters with TPE [Bergstra et al, ICML 2013]
- Auto-Net
  - Joint architecture and hyperparameter search with SMAC
  - First Auto-DL system to win a competition dataset against human experts [Mendoza et al, AutoML 2016]
- Kernels for GP-based NAS
  - Arc kernel [Swersky et al, BayesOpt2013]
  - NASBOT [Kandasamy et al, NIPS 2018]
- Sequential model-based optimization
  - PNAS [Liu et al, ECCV 2018]

Weight Inheritance & Network morphisms

- Network morphisms
  - Change the network structure, but not the modelled function
  - for every input the network yields the same output as before applying the network morphism
- Used in NAS algorithms as operations to generate new networks
- Allow efficient moves in architecture space, avoid costly training from scratch
- Deeper, wider



Network morphisms

Definition

Network morphism Type I. Let $f_i^{w_i}(x)$ be some part of a NN $f^w(x)$, e.g., a layer or a subnetwork. We replace $f_i^{w_i}$ by

$$\tilde{f}_i^{\tilde{w}_i}(x) = A f_i^{w_i}(x) + b$$

The network morphism equation obviously holds for $A = \mathbf{1}, b = \mathbf{0}$.

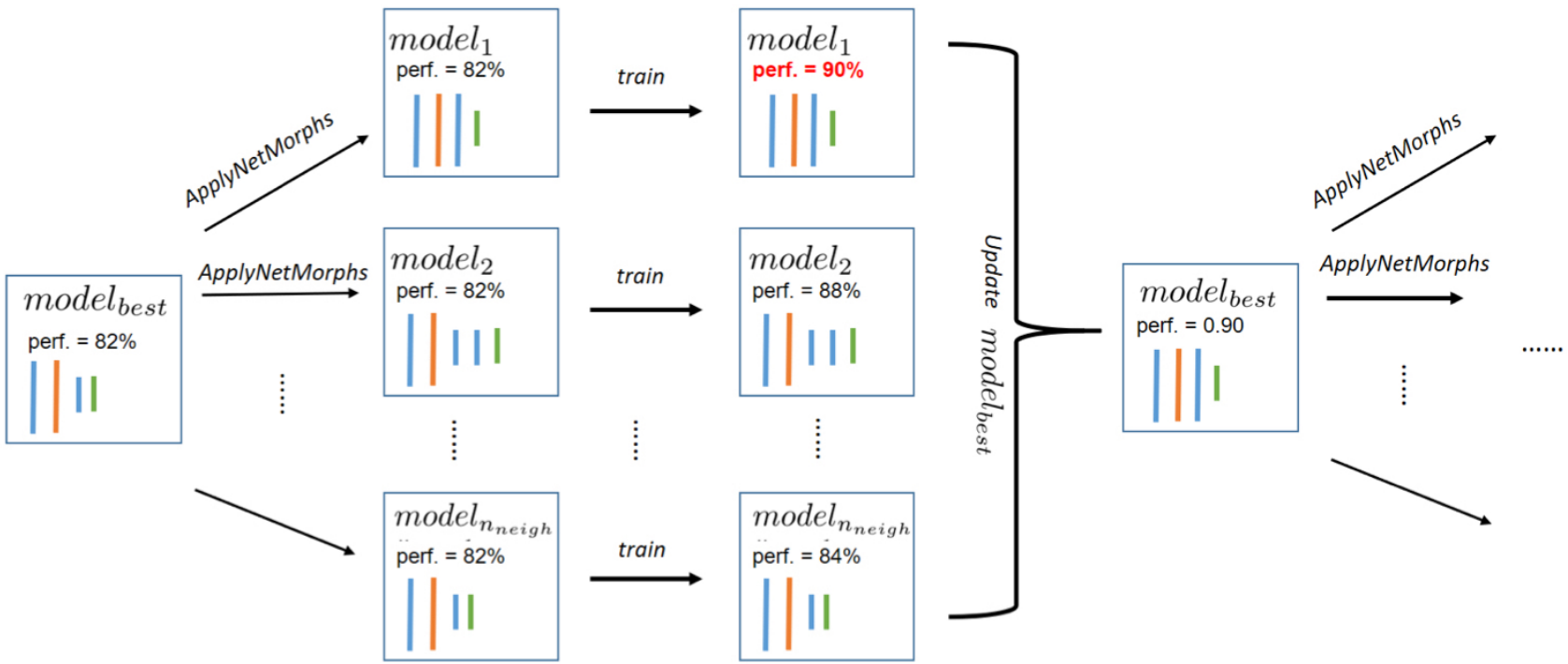
Definition

Network morphism Type II. Assume $f_i^{w_i}$ has the form $f_i^{w_i}(x) = A h^{w_h}(x) + b$ for an arbitrary function h. We replace $f_i^{w_i}$, $w_i = (w_h, A, b)$ by

$$\tilde{f}_i^{\tilde{w}_i}(x) = \begin{pmatrix} A & \tilde{A} \end{pmatrix} \begin{pmatrix} h^{w_h}(x) \\ \tilde{h}^{w_{\tilde{h}}}(x) \end{pmatrix} + b$$

The network morphism equation can trivially be satisfied by setting $\tilde{A} = 0$.

Weight inheritance & network morphisms



$\rightarrow$ enables efficient architecture search

Efficient Performance (CIFAR-10)

	Reference	Error (%)	Params (Millions)	GPU Days
RL	Baker et al. (2017)	6.92	11.18	100
	Zoph and Le (2017)	3.65	37.4	22,400
	Cai et al. (2018a)	4.23	23.4	10
	Zoph et al. (2018)	3.41	3.3	2,000
	Zoph et al. (2018) + Cutout	2.65	3.3	2,000
	Zhong et al. (2018)	3.54	39.8	96
	Cai et al. (2018b)	2.99	5.7	200
	Cai et al. (2018b) + Cutout	2.49	5.7	200
EA	Real et al. (2017)	5.40	5.4	2,600
	Xie and Yuille (2017)	5.39	N/A	17
	Suganuma et al. (2017)	5.98	1.7	14.9
	Liu et al. (2018b)	3.75	15.7	300
	Real et al. (2019)	3.34	3.2	3,150
	Elsken et al. (2018)	5.2	19.7	1
	Wistuba (2018a) + Cutout	3.57	5.8	0.5

NAS with weight inher. / network morphisms

[Wistuba et al., preprint 2019]

Weight Sharing & One-shot Models

- Embed architectures from search space into single network
- Each path through the one-shot model is an architecture
- Only need a single training of the one-shot model
- Weights are shared across architectures embedded in one-shot model
- Search space limited and model kept in GPU-memory

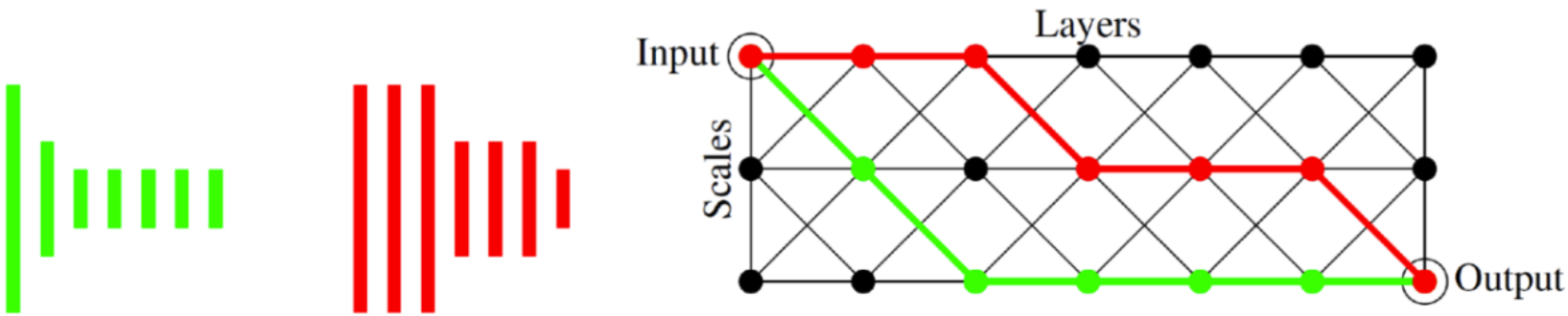
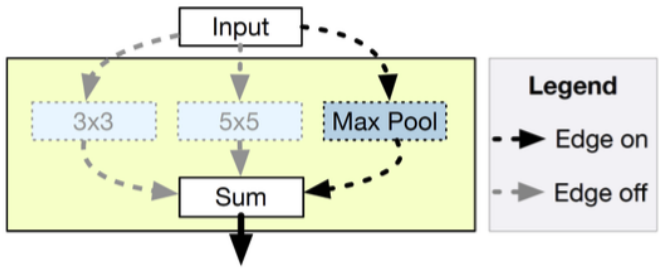


Figure: embeddings of two 7-layer CNNs (red, green) [Saxena & Verbeek, NeurIPS 2016]

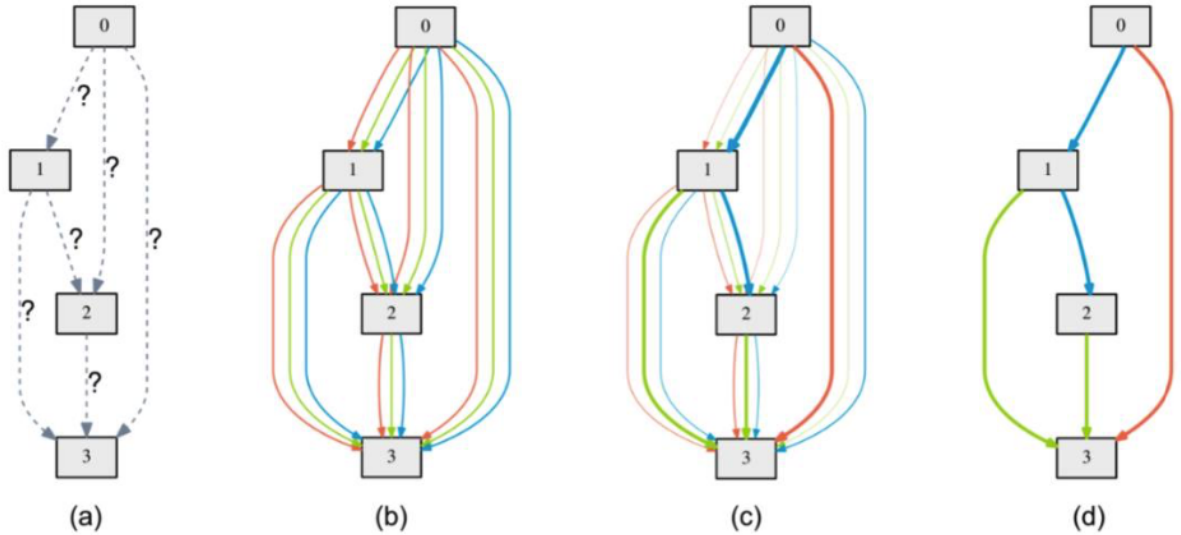
Weight Sharing & One-shot Models

- Simplifying One-Shot Architecture Search [Bender et al., ICML 2018]
  - **Use path dropout to make sure the individual models perform well by themselves**



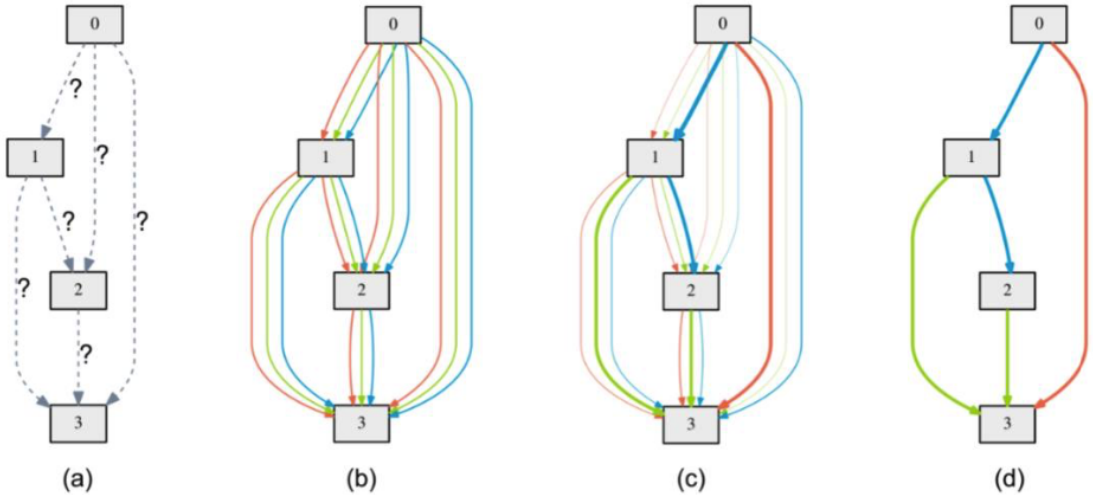
- ENAS [Pham et al., ICML 2018]
  - Use RL to sample paths (=architectures) from one-shot model
- SMASH [Brock et al., MetaLearn 2017]
  - Train hypernetwork that generates weights of models

DRATS: Differentiable Architecture Search



- (a) The data can only flow from lower-level to higher-level nodes, and the operations on edges are initially unknown
- (b) The initial operation on each edge is a mixture of candidate operations, each having equal weight
- (c) Weight of each operation is learnable and ranges from 0 to 1 (previous discrete methods' weight can only be 0 or 1)
- (d) The final structure is constructed by preserving the max-weighted operation on each edge to improve efficiency

DRATS: Differentiable Architecture Search



- Relax the discrete NAS problem (from (a) to (b))
  - One-shot model with continuous architecture weight α for each operator
  - Relax by a softmax function:

$$\overline{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o' \in \mathcal{O}} \exp\left(\alpha_{o'}^{(i,j)}\right)} o(x)$$

- Solve a bi-level optimization problem in (c)

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}_{val}\left(w^*(\alpha), \alpha\right) \\ \text{s.t.} \quad & w^*(\alpha) = \operatorname{argmin}_{\theta} \mathcal{L}_{train}(\theta, \alpha) \end{aligned}$$

DRATS: Differentiable Architecture Search

- Very fast:
  - By alternating SGD steps for α and w, runtime only a bit higher than w alone
- Very brittle optimization:
  - Requires hyper-parameter tuning for new problems
  - Discretization in step (d) can deteriorate performance a lot
- One-shot model needs to fit into GPU memory
- Already lots of follow-up work to solve these problems
  - [Xie et al., ICLR 2019],[Cai et al., ICLR 2019]

One-shot Performance

	Reference	Error (%)	Params (Millions)	GPU Days
RL	Zoph and Le (2017)	3.65	37.4	22,400
	Cai et al. (2018a)	4.23	23.4	10
	Zoph et al. (2018)	3.41	3.3	2,000
	Zoph et al. (2018) + Cutout	2.65	3.3	2,000
	Cai et al. (2018b)	2.99	5.7	200
	Cai et al. (2018b) + Cutout	2.49	5.7	200
EA	Real et al. (2017)	5.40	5.4	2,600
	Liu et al. (2018b)	3.75	15.7	300
	Real et al. (2019)	3.34	3.2	3,150
	Elsken et al. (2018)	5.2	19.7	1
	Wistuba (2018a) + Cutout	3.57	5.8	0.5
One-Shot	Pham et al. (2018)	3.54	4.6	0.5
	Pham et al. (2018) + Cutout	2.89	4.6	0.5
	Bender et al. (2018)	4.00	5.0	N/A
	Casale et al. (2019) + Cutout	2.81	3.7	1
	Liu et al. (2019b) + Cutout	2.76	3.3	4
	Xie et al. (2019b) + Cutout	2.85	2.8	1.5
	Cai et al. (2019) + Cutout	2.08	5.7	8.33
	Brock et al. (2018)	4.03	16.0	3
	Zhang et al. (2019)	2.84	5.7	0.84
Random	Liu et al. (2018b)	3.91	N/A	300
	Luo et al. (2018)	3.92	3.9	0.3
	Liu et al. (2019b) + Cutout	3.29	3.2	4
	Li and Talwalkar (2019) + Cutout	2.85	4.3	2.7

Outline

Meta-learning

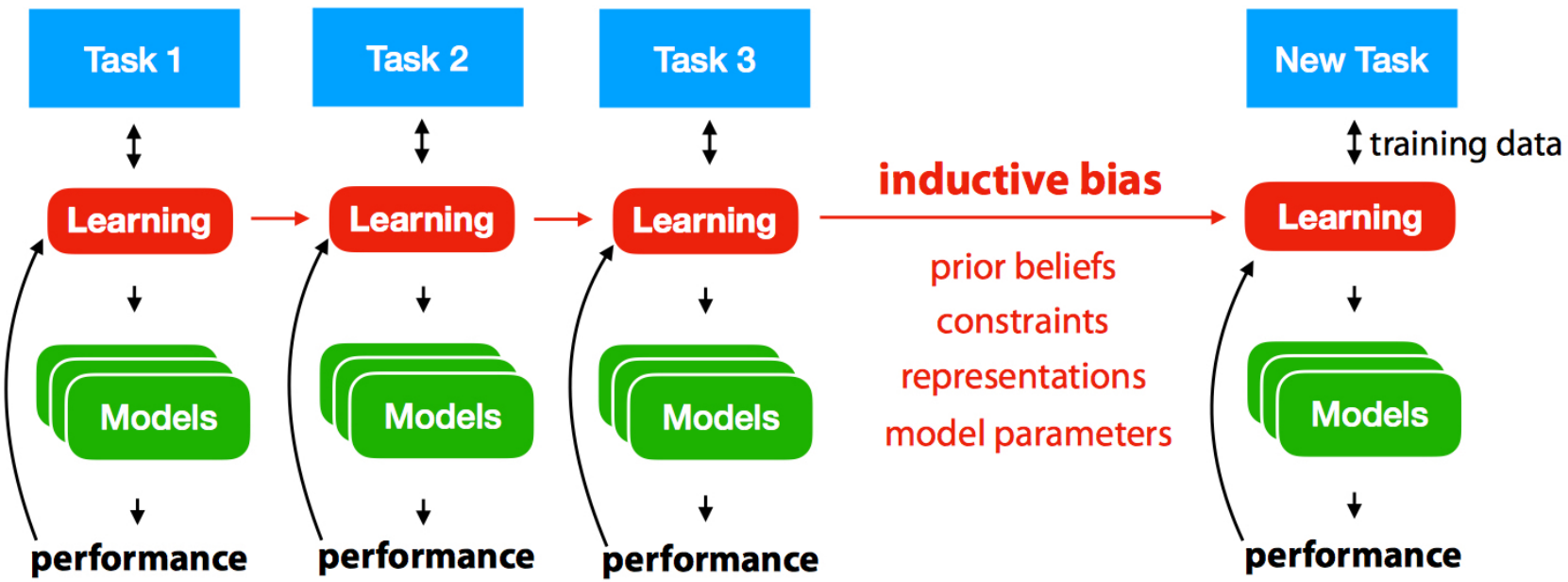
- Given a new unknown ML task, ML methods usually start from scratch to build an ML pipeline
- Meta-learning is the science of learning to learn
- Based on the observation of various configurations on previous ML tasks, meta-learning builds a model to construct promising configurations for a new unknown ML task leading to faster convergence with less trial and error

Meta-learning v.s. Multi-task learning v.s. Ensemble learning

- Multi-task learning learns multiple related tasks simultaneously
- Ensemble learning builds multiple models on the same task
- They do not in themselves involve learning from prior experience on other tasks

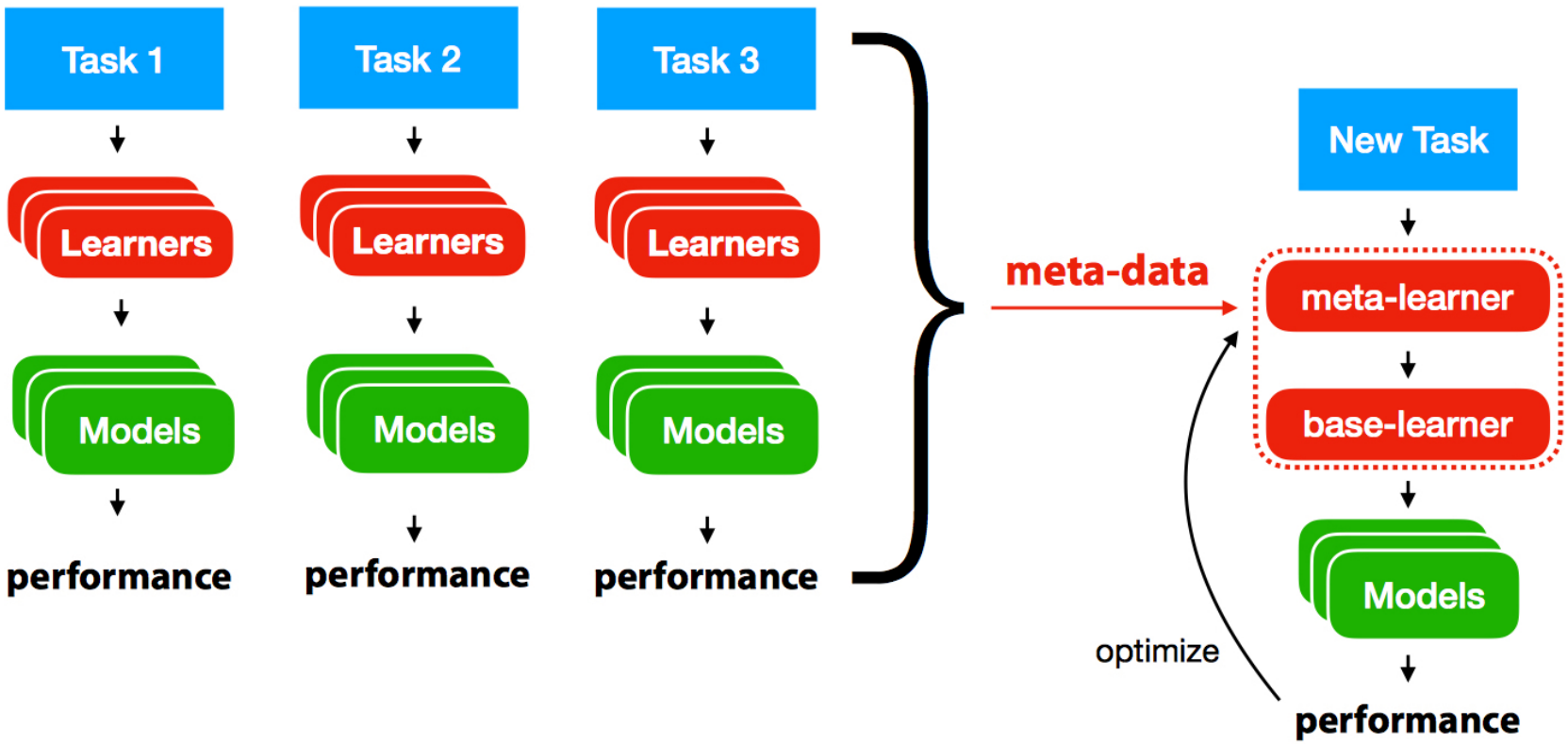
Learning to learn

- Inductive bias: all assumptions added to the training data to learn effectively
- If prior tasks are similar, we can transfer prior knowledge to new tasks
- if not it may actually harm learning



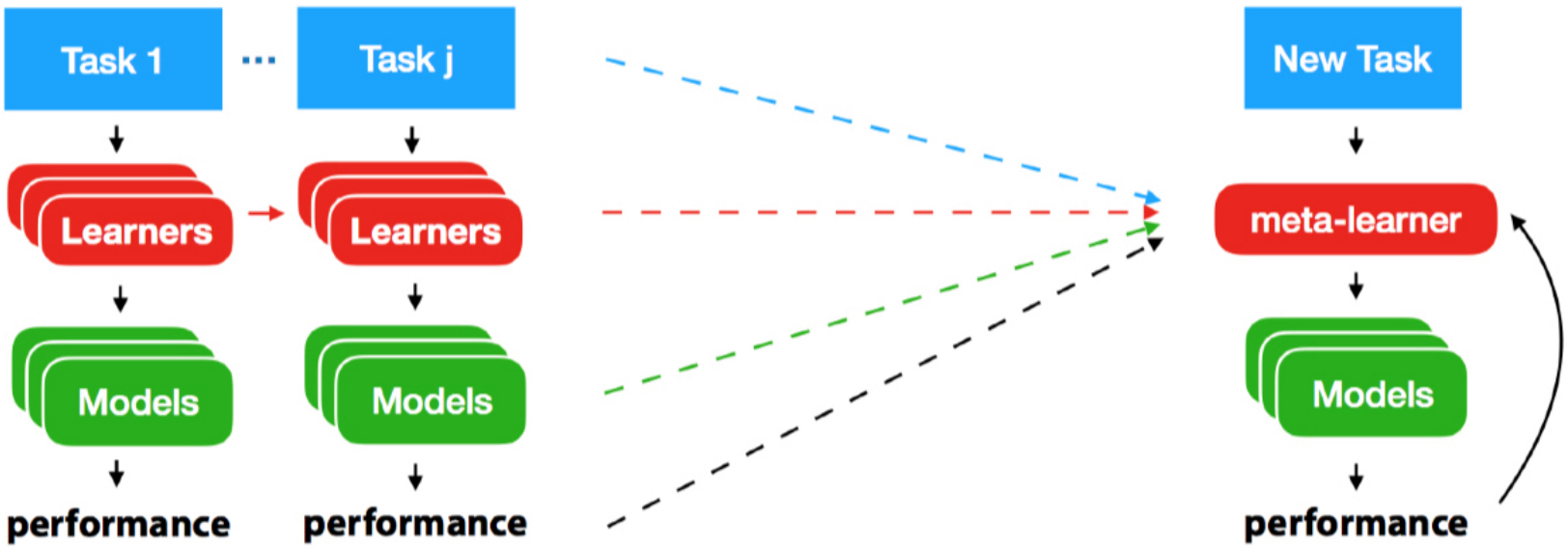
Meta-learning

- Collect meta-data about learning episodes and learn from them
- Meta-learner learns a (base-)learning algorithm, end-to-end

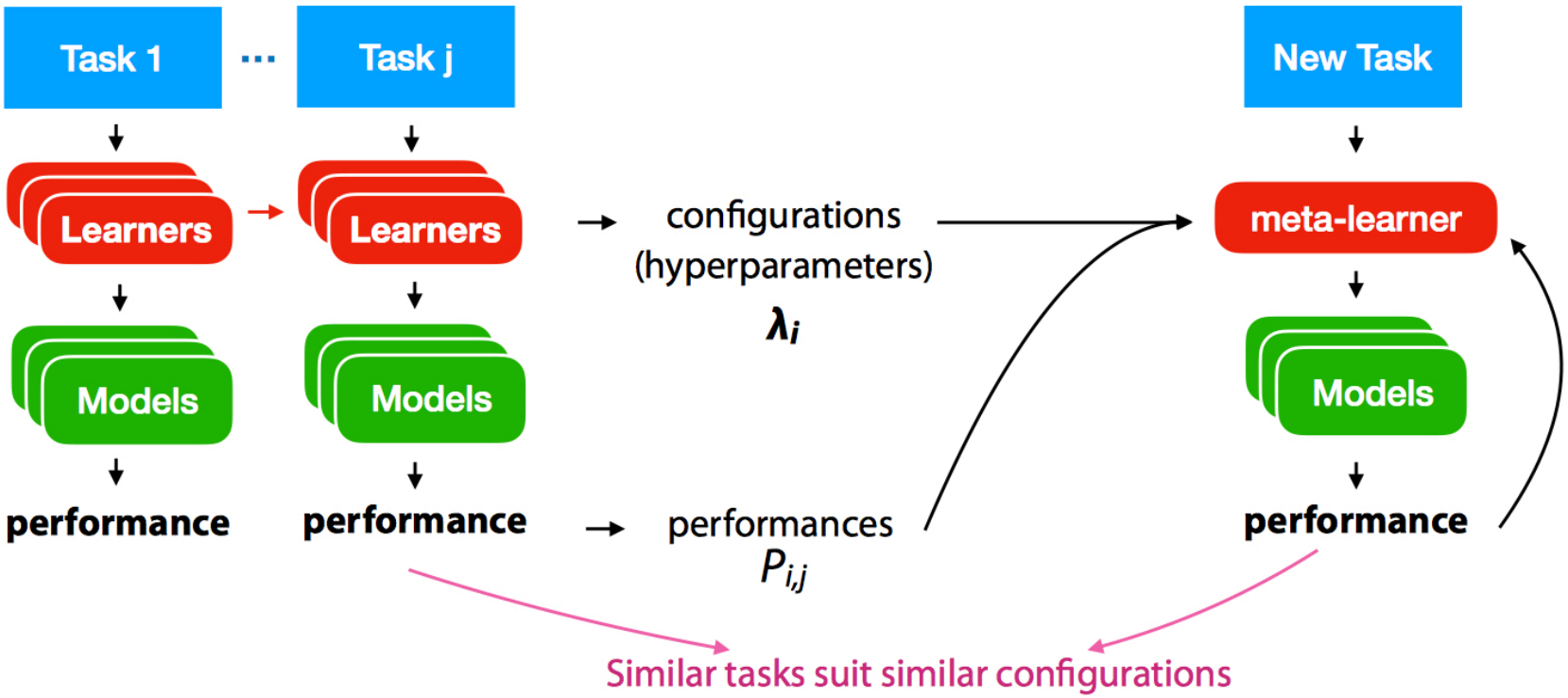


Three approaches

- Learning from Model Evaluations
- Learning from Task Properties
- Learning from Prior Models

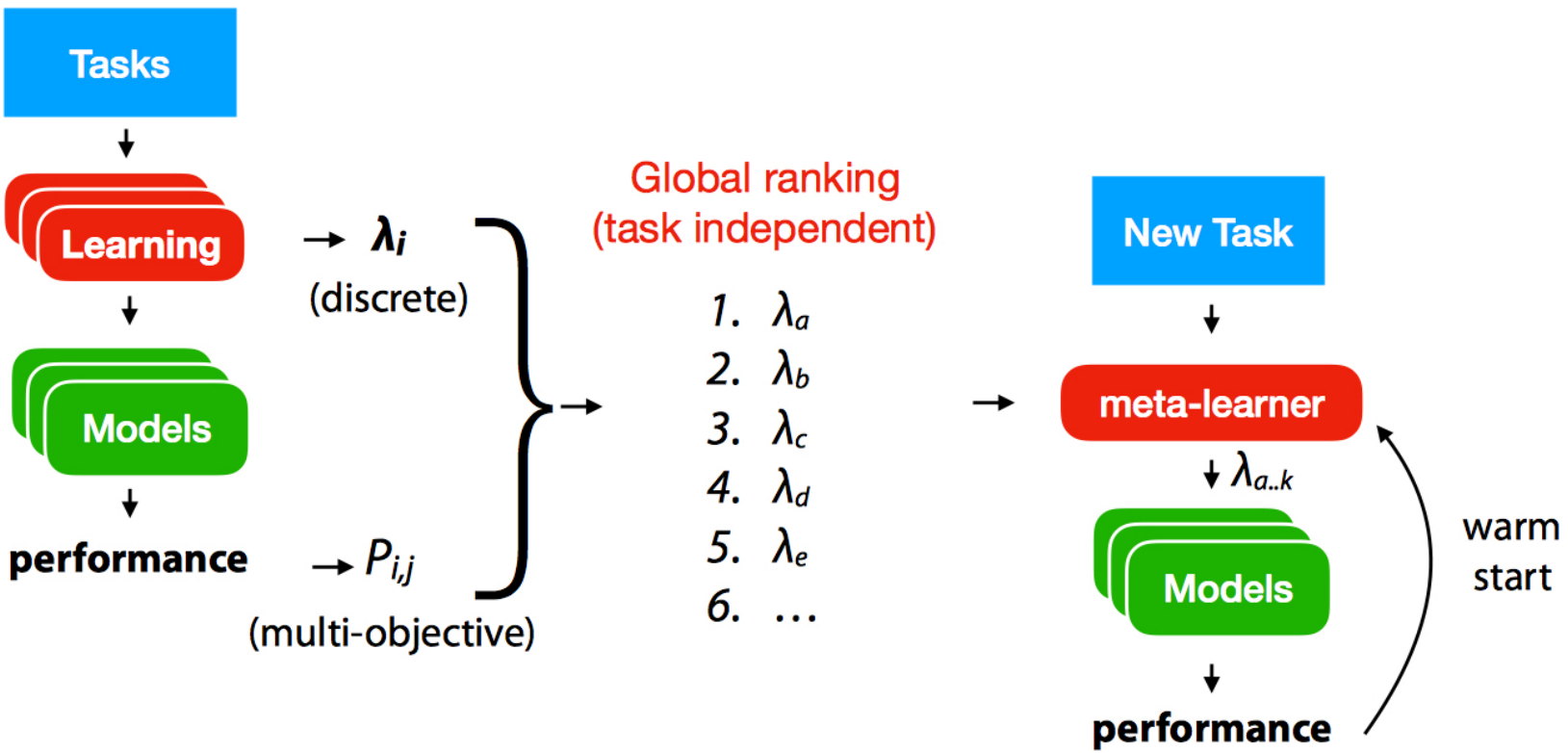


Learning from Model Evaluations



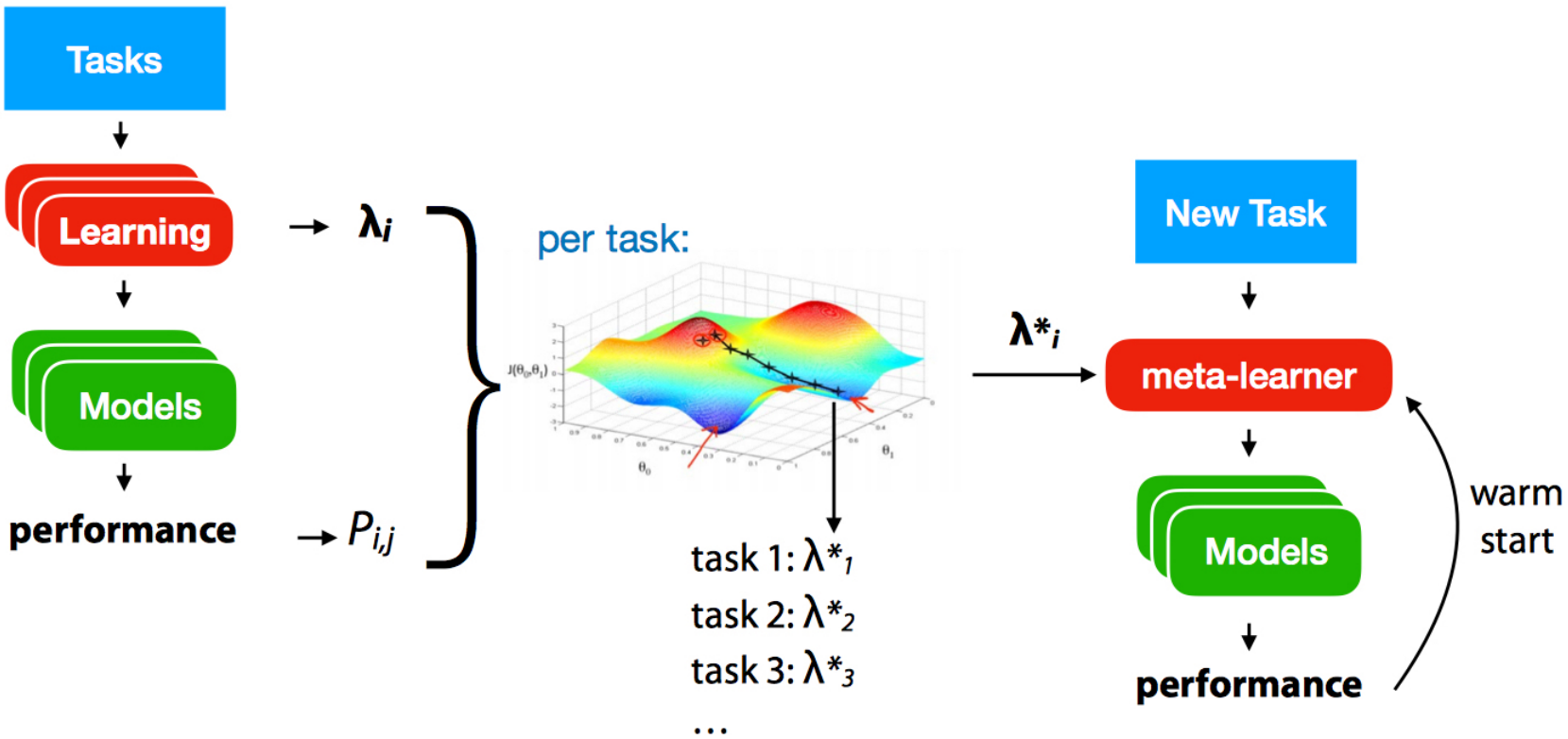
Top-K recommendation

- Build a global (multi-objective) ranking, recommend the top-K
- Requires fixed selection of candidate configurations
- Can be used as a warm start for optimization techniques



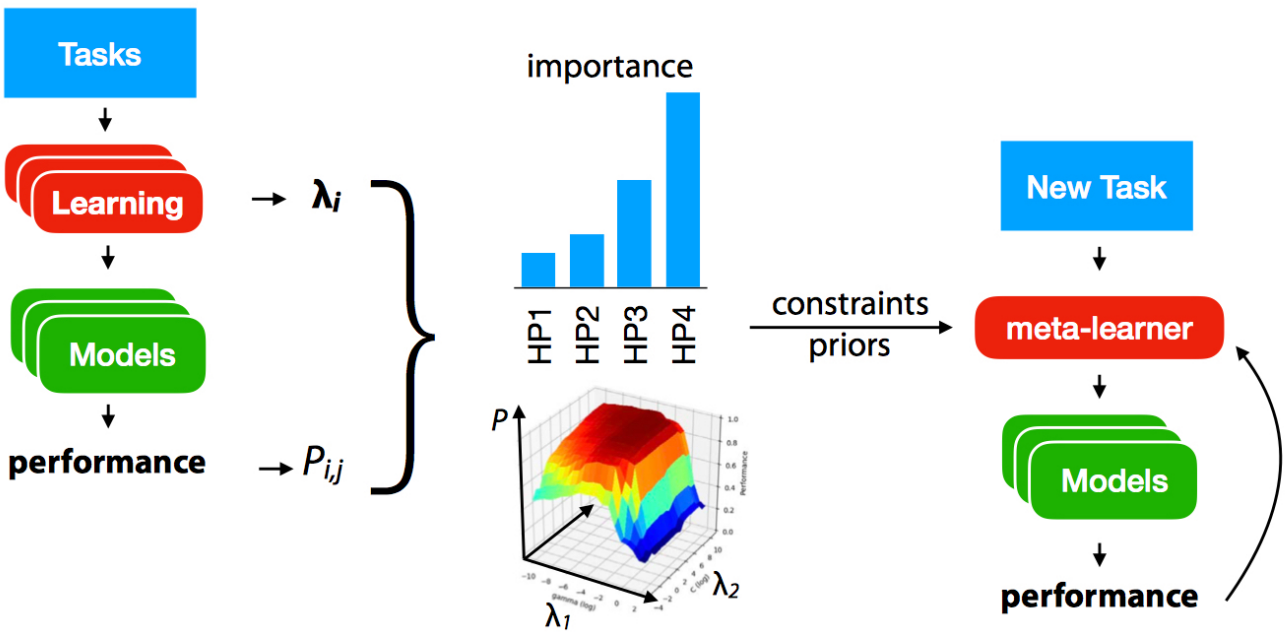
Warm-starting with plugin estimators

- What if prior configurations are not optimal?
- Per task, fit a differentiable plugin estimator on all evaluated configurations
- Do gradient descent to find optimized configurations, recommend those



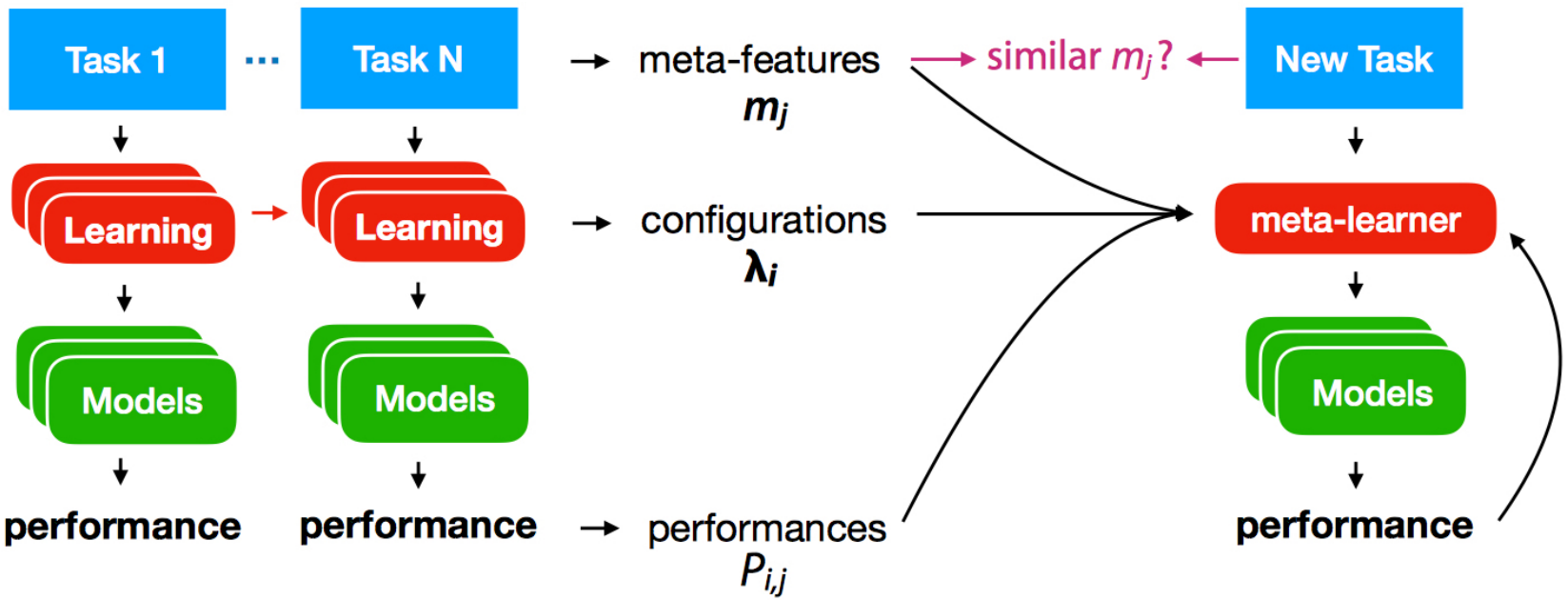
Configuration space design

- Prior evaluations can also be used to learn a better configuration space Θ^*
  - speed up the search as more relevant regions of the configuration space are explored
- Functional ANOVA: hyperparameters are important if they explain most of variance
- Tunability: learn an optimal hyperparameter, and define hyperparameter importance as the performance gain by tuning



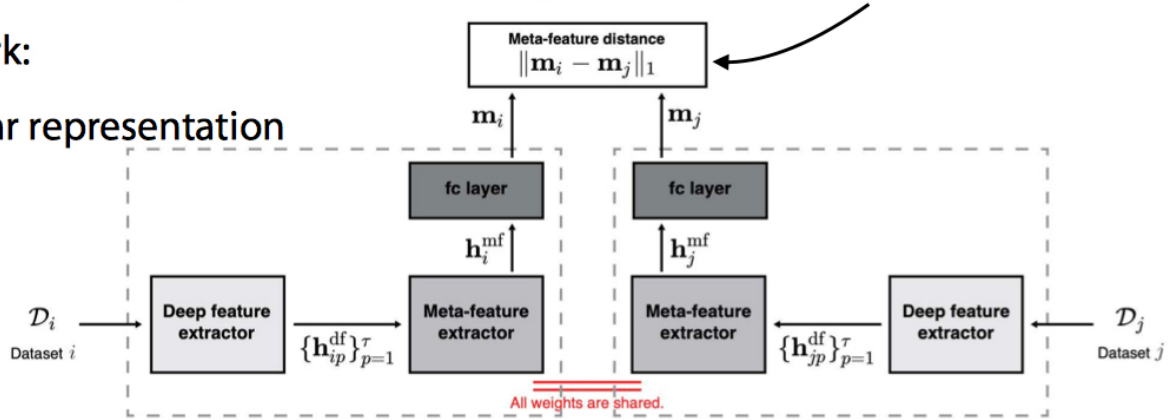
Learning from Task Properties

- Another rich source of meta-data are characterizations (meta-features) of the task at hand



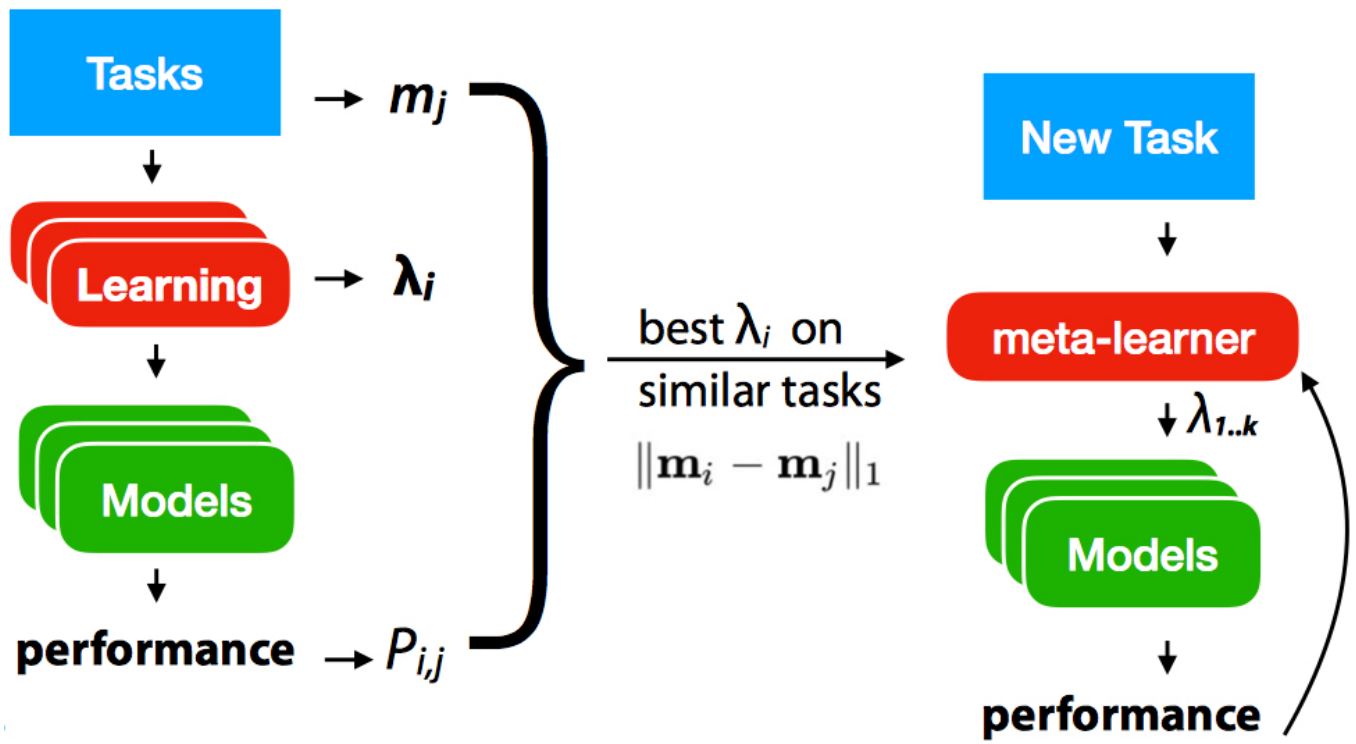
Meta-Features

- **Hand-crafted (interpretable) meta-features**[1]
  - **Number of** instances, features, classes, missing values, outliers,…
  - **Statistical:** skewness, kurtosis, correlation, covariance, sparsity, variance,…
  - **Information-theoretic**: class entropy, mutual information, noise-signal ratio,…
  - **Model-based**: properties of simple models trained on the task
  - **Landmarkers**: performance of fast algorithms trained on the task
  - Domain specific task properties
- **Learning a joint task representation**
  - Deep metric learning: learn a representation h^{mf} using a ground truth distance[2]
  - With Siamese Network:
    - Similar task, similar representation

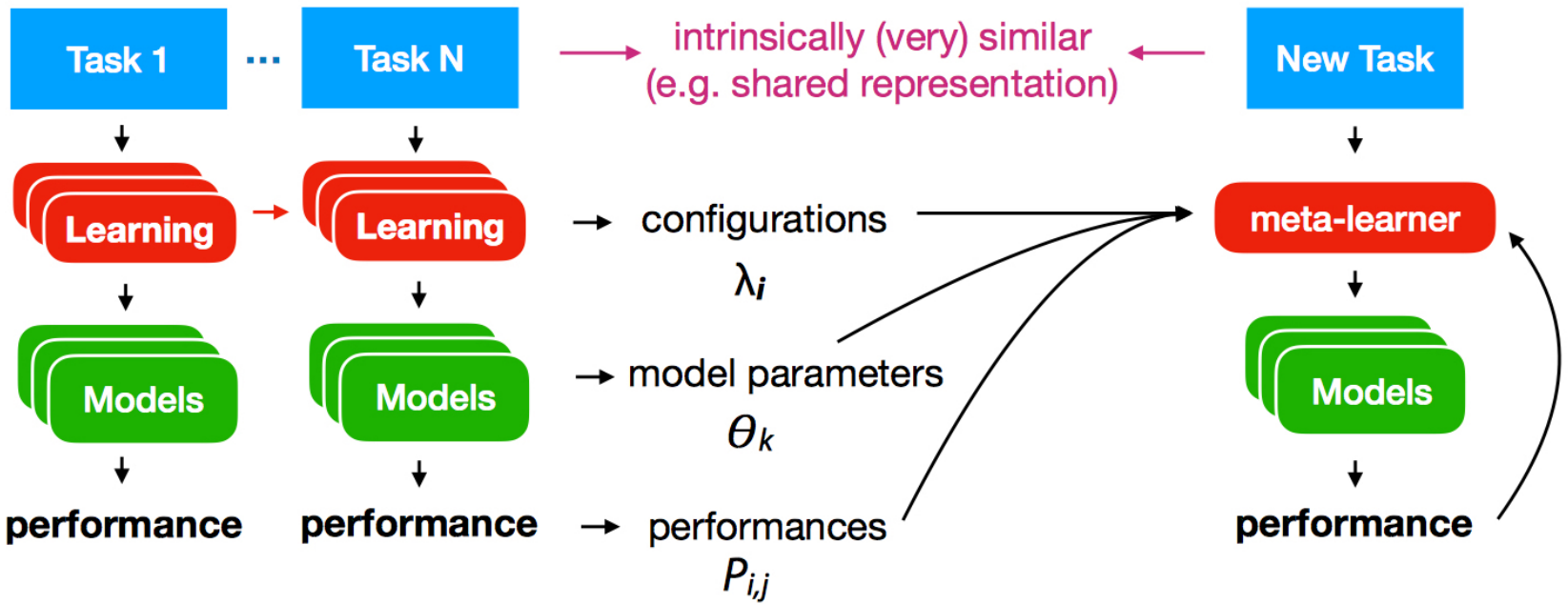


Warm-starting from similar tasks

- Find k most similar tasks, warm-start search with best λ_i
- Collaborative filtering: configurations λ_i are "related" by tasks t_j

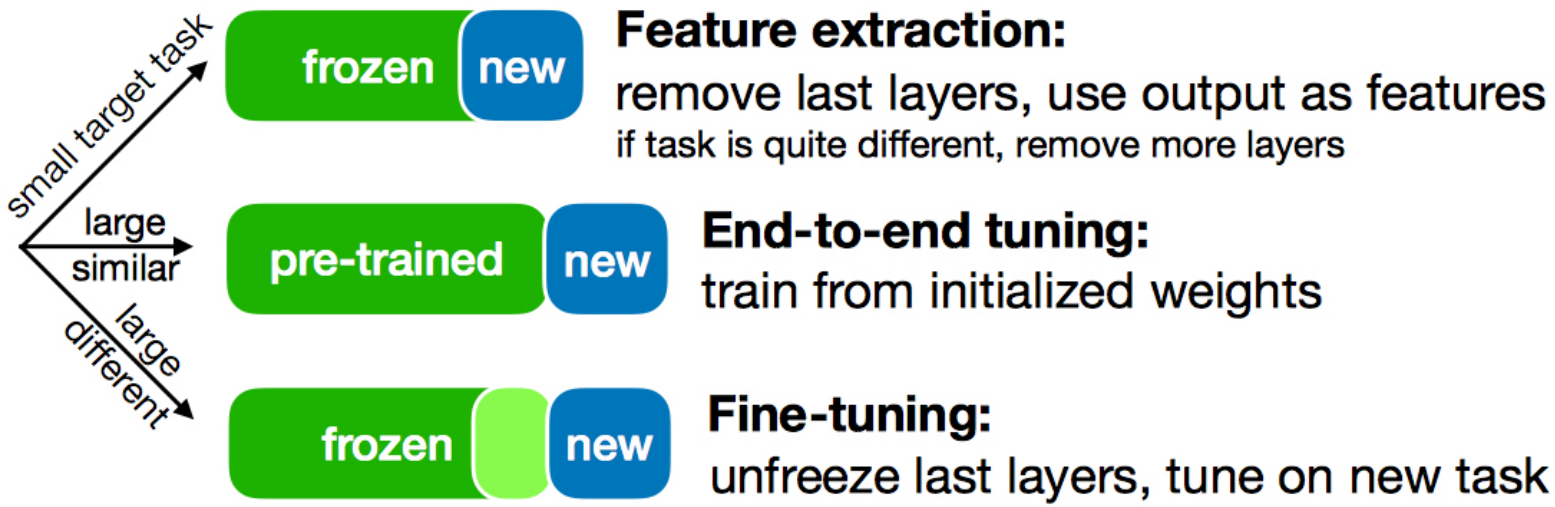


Learning from Prior Models



Transfer Learning

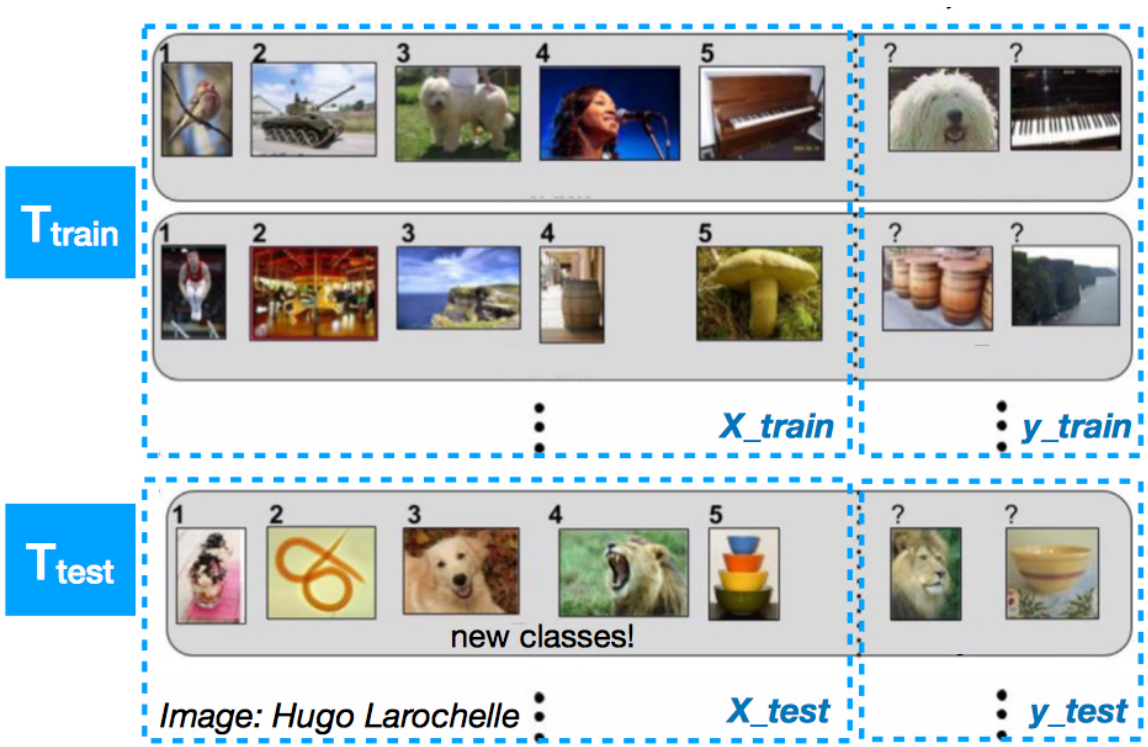
- Select source tasks, transfer trained models to similar target task
- Use as starting point for tuning, or freeze certain aspects
- Reinforcement learning: start policy search from prior policy
- Neural networks: both structure and weights can be transferred
  - Large image datasets (e.g. ImageNet)
  - Large text corpora (e.g. Wikipedia)
- Fails if tasks are not similar enough



Few-shot learning

- Learn how to learn from few examples (given similar tasks)
- Meta-learner must learn how to train a base-learner based on prior experience
- Parameterize base-learner model and learn the parameters

$$\text{cost}(\theta_i) = \frac{1}{|T_{\text{test}}|} \sum_{t \in T_{\text{test}}} \text{loss}(\theta_i, t)$$



Few-shot learning: approaches

- Existing algorithm as meta-learner:
  - LSTM + gradient descent
  - Learn $\Theta_{\text{init}+}$ gradient descent
  - KNN-like: Memory + similarity
  - Learn embedding + classifier
  - ...
- Black-box meta-learner:
  - Neural Turing machine (with memory)
  - Neural attentive learner
  - ...

Model-agnostic meta-learning[4]

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks

Require: α, β: step size hyperparameters

1: randomly initialize θ

2: **while** not done **do**

3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$

4: **for all** $\mathcal{T}_i$ **do**

5: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples

6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$

7: **end for**

8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$

9: **end while**

[4] Finn, Chelsea, Pieter Abbeel, and Sergey Levine. 2017. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks." International Conference on Machine Learning, 1126–35.

Outline

AutoML: Further Benefits and Concerns

- Democratization of data science :)
- We directly have a strong baseline :)
- Reducing the tedious part of our work, freeing time to focus on problems humans do best (creativity, interpretation,...) :)
- People will use it without understanding anything :(

Thanks.

HP: http://keg.cs.tsinghua.edu.cn/jietang/
Email: jietang@tsinghua.edu.cn