Automatic Machine Learning (AutoML): A Tutorial

Frank Hutter

University of Freiburg
fh@cs.uni-freiburg.de

Joaquin Vanschoren

Eindhoven University of Technology
j.vanschoren@tue.nl

Slides available at automl.org/events -> AutoML Tutorial
(all references are clickable links)

Motivation: Successes of Deep Learning

Speech recognition

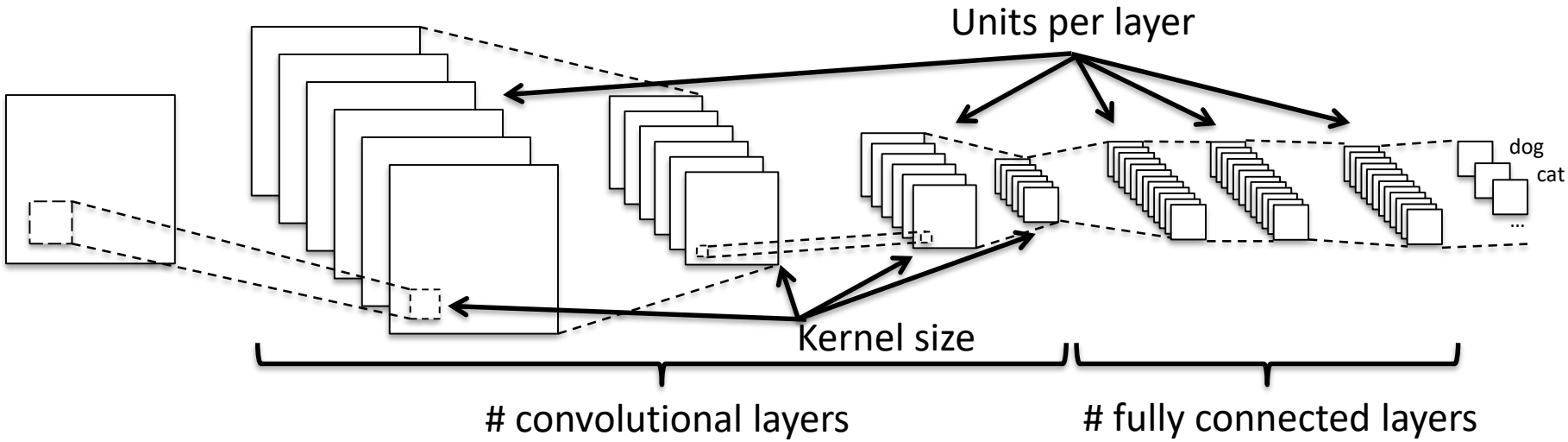
Computer vision in self-driving cars

Reasoning in games

One Problem of Deep Learning

Performance is very **sensitive** to **many hyperparameters**

- Architectural hyperparameters

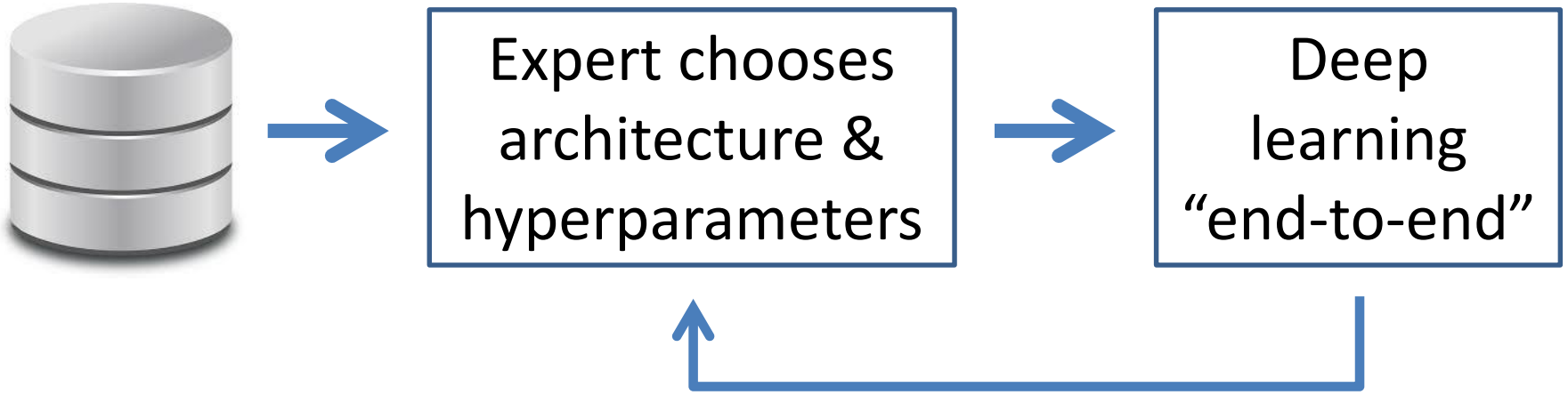


- Optimization algorithm, learning rates, momentum, batch normalization, batch sizes, dropout rates, weight decay, data augmentation, …

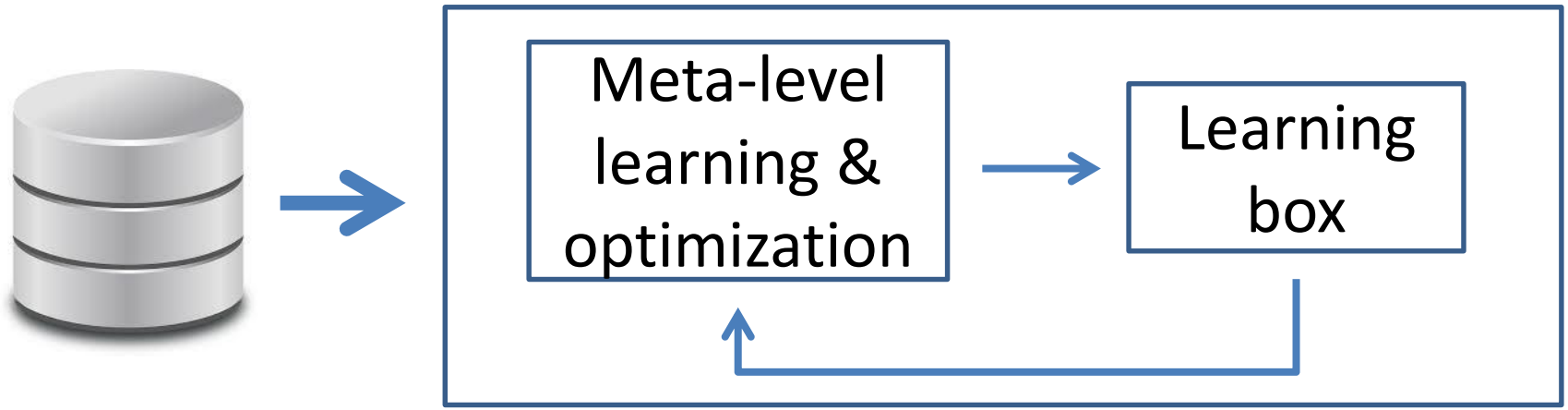
→ Easily 20-50 design decisions

Deep Learning and AutoML

Current deep learning practice



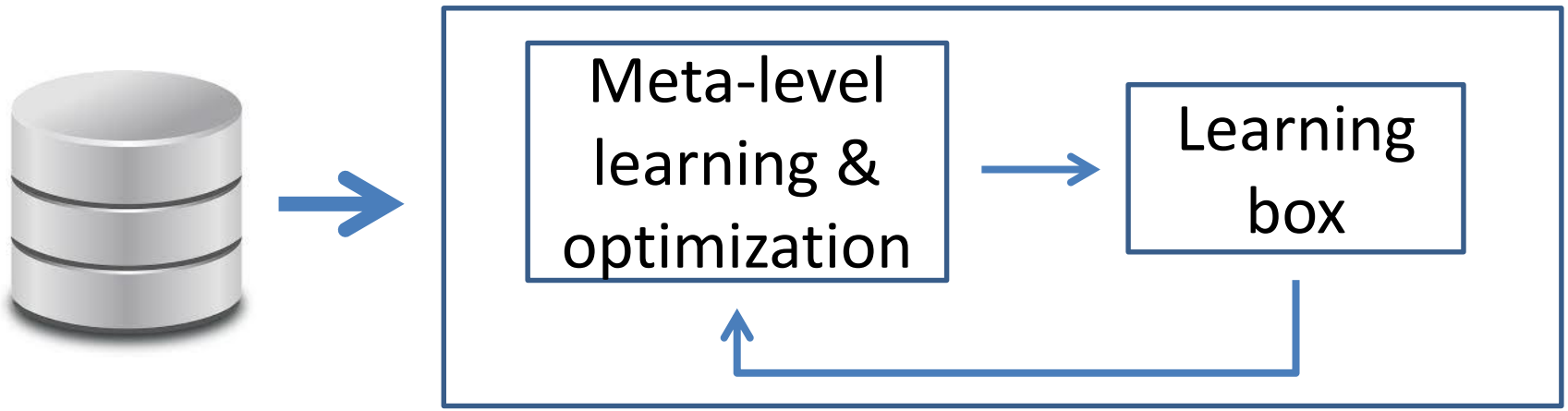
AutoML: true end-to-end learning



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
  - …

AutoML: true end-to-end learning

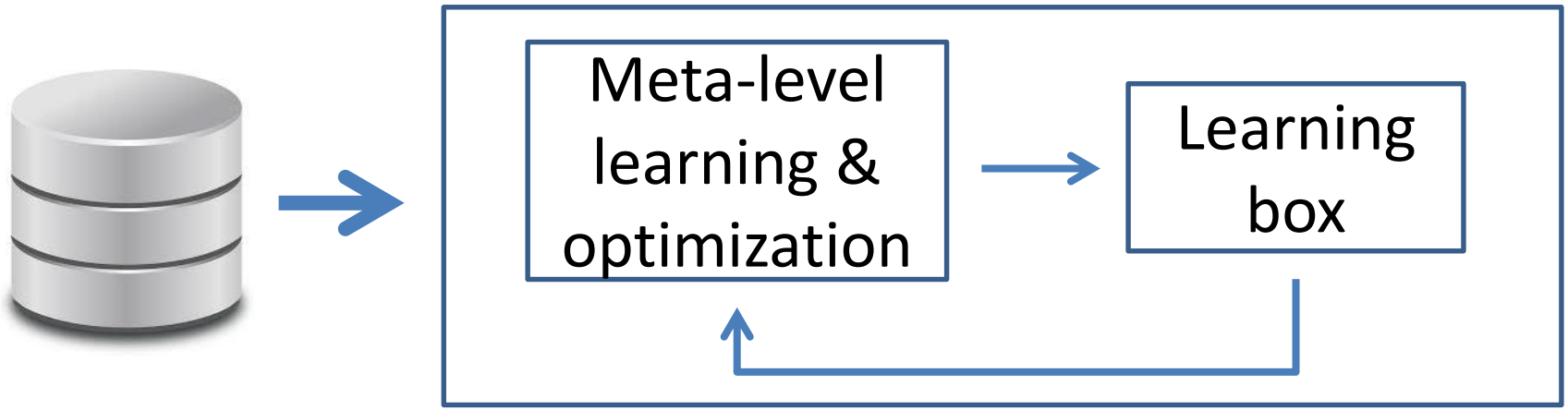


Outline

1. Modern Hyperparameter Optimization
2. Neural Architecture Search
3. Meta Learning

For more details, see: automl.org/book

AutoML: true end-to-end learning



Outline

1. Modern Hyperparameter Optimization

- AutoML as Hyperparameter Optimization
- Blackbox Optimization
- Beyond Blackbox Optimization

Based on: Feurer & Hutter: Chapter 1 of the AutoML book: Hyperparameter Optimization

2. Neural Architecture Search

- Search Space Design
- Blackbox Optimization
- Beyond Blackbox Optimization

Hyperparameter Optimization

Definition: Hyperparameter Optimization (HPO)

Let

- $\boldsymbol{\lambda}$ be the hyperparameters of a ML algorithm A with domain $\boldsymbol{\Lambda}$,
- $\mathcal{L}(A_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$ denote the loss of A, using hyperparameters $\boldsymbol{\lambda}$ trained on D_{train} and evaluated on D_{valid}.

The hyperparameter optimization (HPO) problem is to find a hyperparameter configuration $\boldsymbol{\lambda}^*$ that minimizes this loss:

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

Types of Hyperparameters

- Continuous
  - Example: learning rate
- Integer
  - Example: #units
- Categorical
  - Finite domain, unordered
    - Example 1: algo ∈ {SVM, RF, NN}
    - Example 2: activation function ∈ {ReLU, Leaky ReLU, tanh}
    - Example 3: operator ∈ {conv3x3, separable conv3x3, max pool, …}
  - 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 second 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)

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:

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

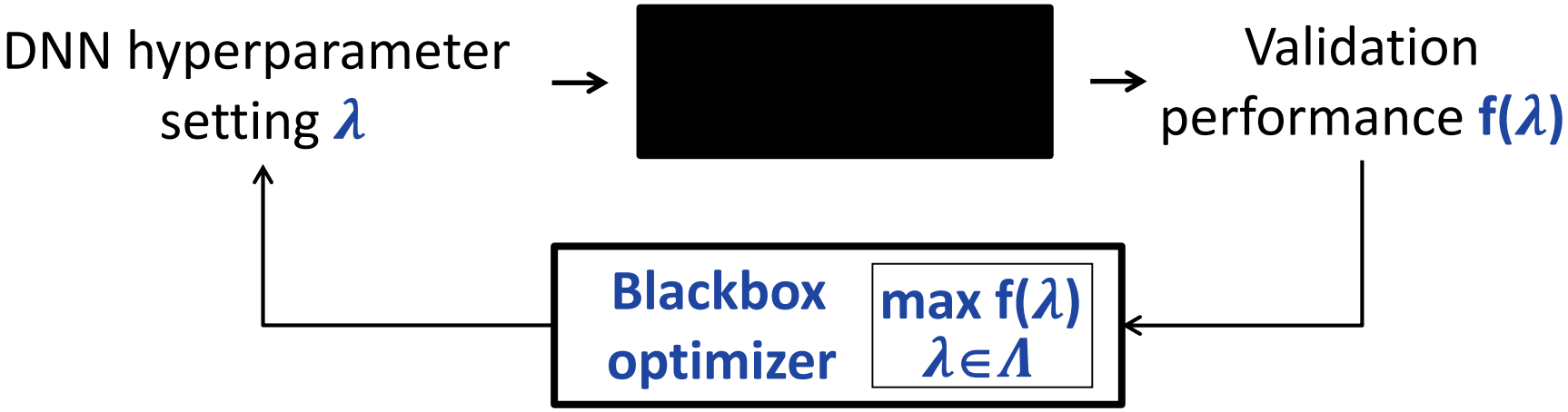
→ Simply a HPO problem with a top-level hyperparameter (choice of algorithm) that all other hyperparameters are conditional on

- E.g., Auto-WEKA: 768 hyperparameters, 4 levels of conditionality

Outline

1. Modern Hyperparameter Optimization
   - AutoML as Hyperparameter Optimization
   - Blackbox Optimization
   - Beyond Blackbox Optimization
2. Neural Architecture Search
   - Search Space Design
   - Blackbox Optimization
   - Beyond Blackbox Optimization

Blackbox Hyperparameter Optimization



- The blackbox function is expensive to evaluate
  $\rightarrow$ sample efficiency is important

Grid Search and Random Search

- Both completely uninformed
- Random search handles unimportant dimensions better
- Random search is a useful baseline

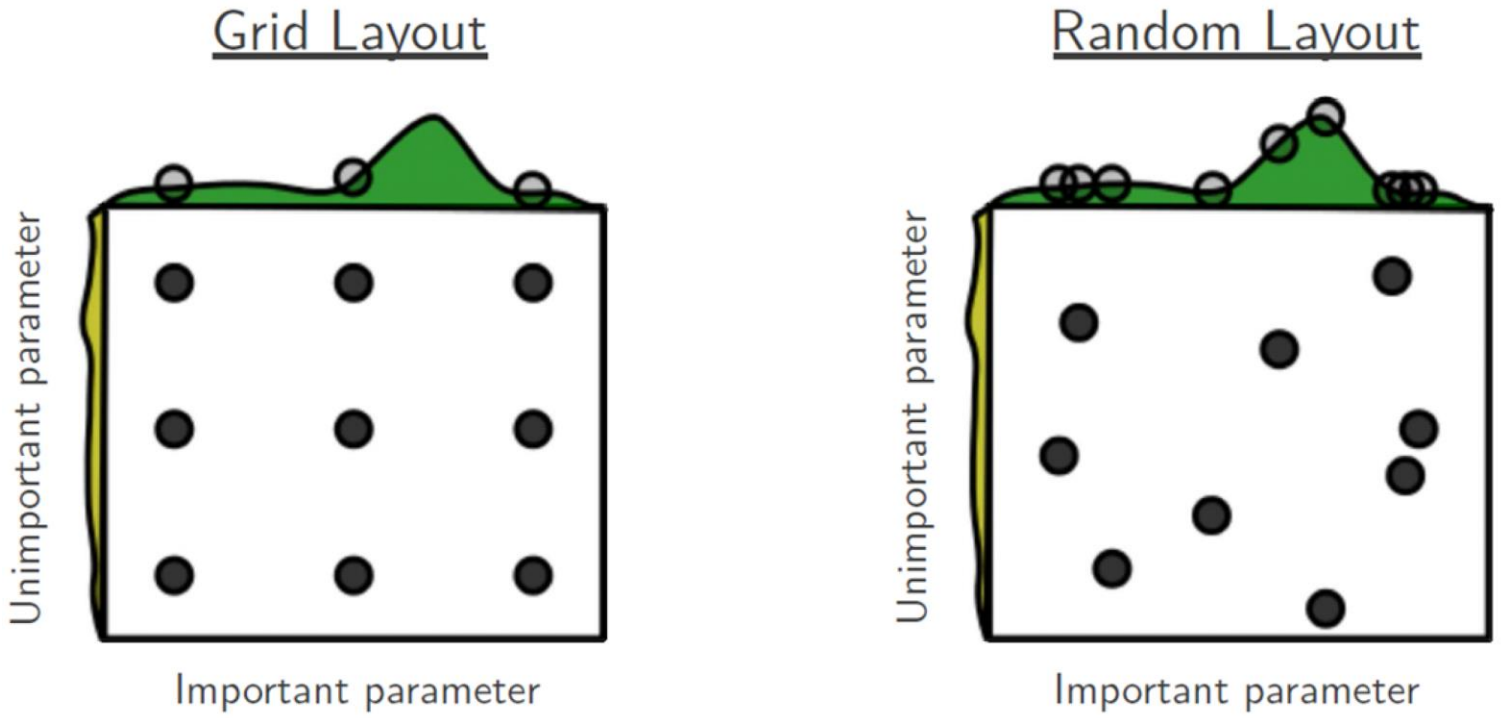


Image source: Bergstra & Bengio, JMLR 2012

Bayesian Optimization

- Approach
  - Fit a proabilistic model to the function evaluations $\langle \lambda, f(\lambda) \rangle$
  - Use that model to trade off exploration vs. exploitation

- Popular since Mockus [1974]
  - Sample-efficient
  - Works when objective is nonconvex, noisy, has unknown derivatives, etc
  - Recent convergence results [Srinivas et al, 2010; Bull 2011; de Freitas et al, 2012; Kawaguchi et al, 2016]

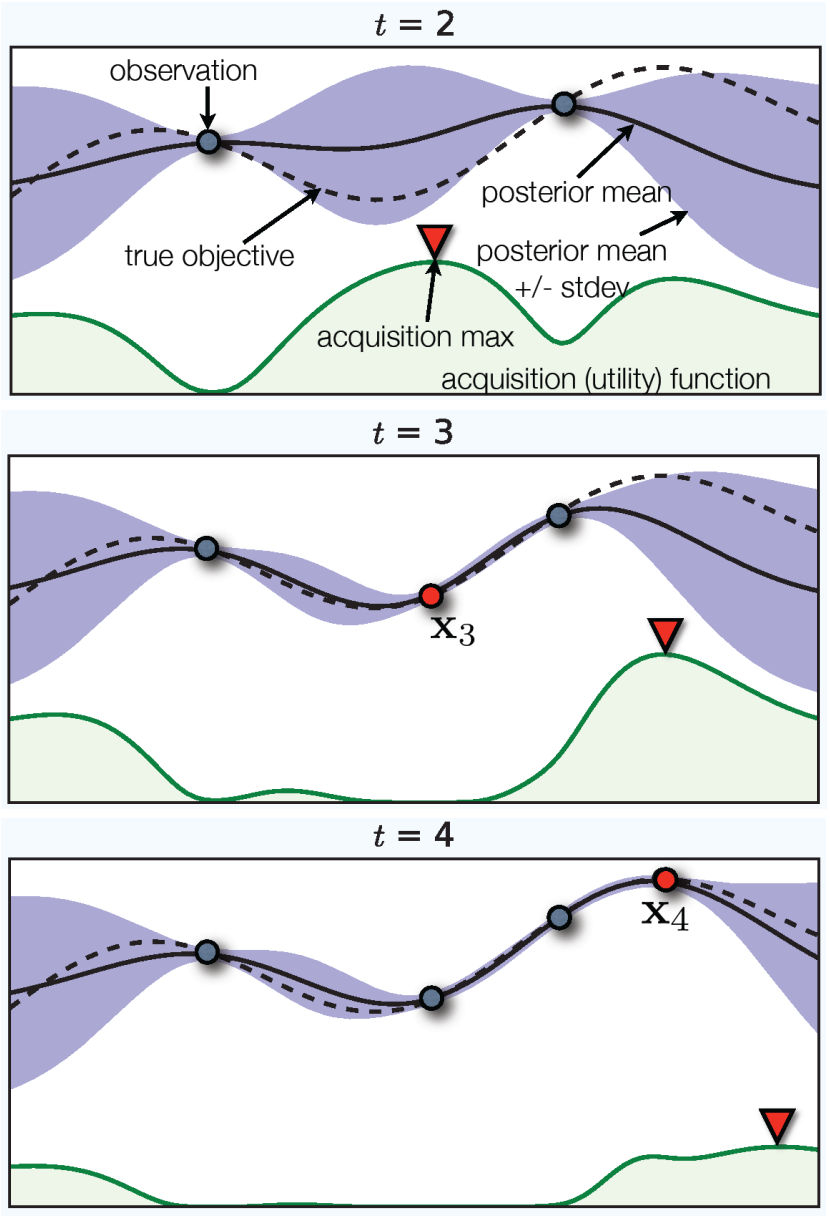


Image source: Brochu et al, 2010

Example: Bayesian Optimization in AlphaGo

[Source: email from Nando de Freitas, today; quotes from Chen et al, forthcoming]

- 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:
  - Complex hyperparameter space
    - High-dimensional (low effective dimensionality) [e.g., Wang et al, 2013]
    - Mixed continuous/discrete hyperparameters [e.g., Hutter et al, 2011]
    - Conditional hyperparameters [e.g., Swersky et al, 2013]
  - Noise: sometimes heteroscedastic, large, non-Gaussian
  - Robustness (usability out of the box)
  - Model overhead (budget is runtime, not #function evaluations)

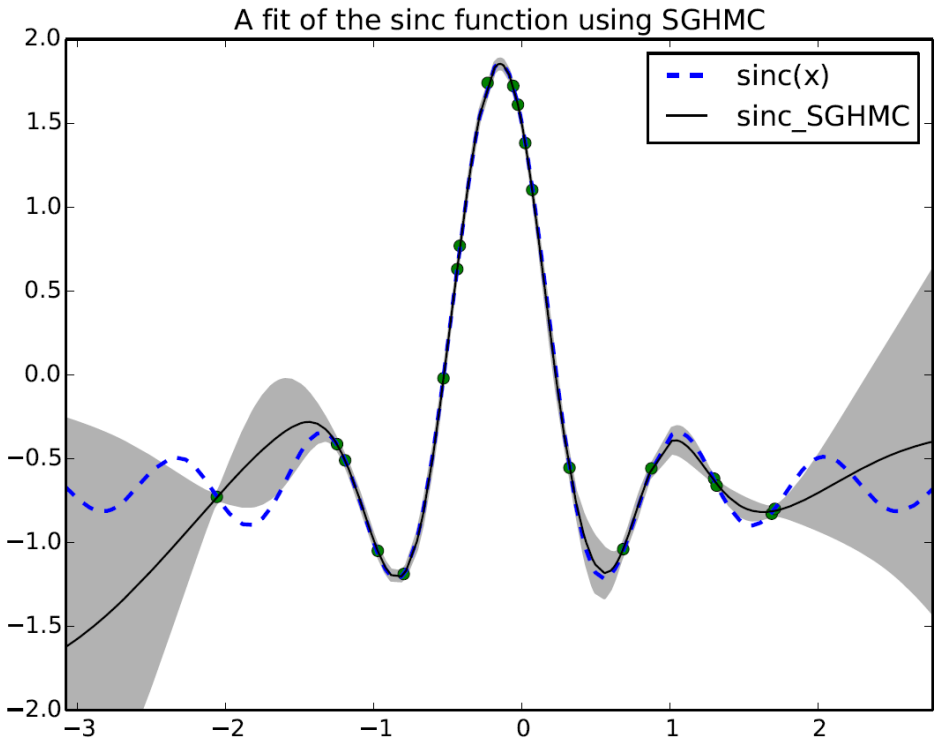
- Simple solution used in SMAC: random forests [Breiman, 2001]
  - Frequentist uncertainty estimate:
    variance across individual trees' predictions [Hutter et al, 2011]

Bayesian Optimization with Neural Networks

- Two recent promising models for Bayesian optimization
  - Neural networks with Bayesian linear regression using the features in the output layer [Snoek et al, ICML 2015]
  - Fully Bayesian neural networks, trained with stochastic gradient Hamiltonian Monte Carlo [Springenberg et al, NIPS 2016]

- Strong performance on low-dimensional HPOlib tasks

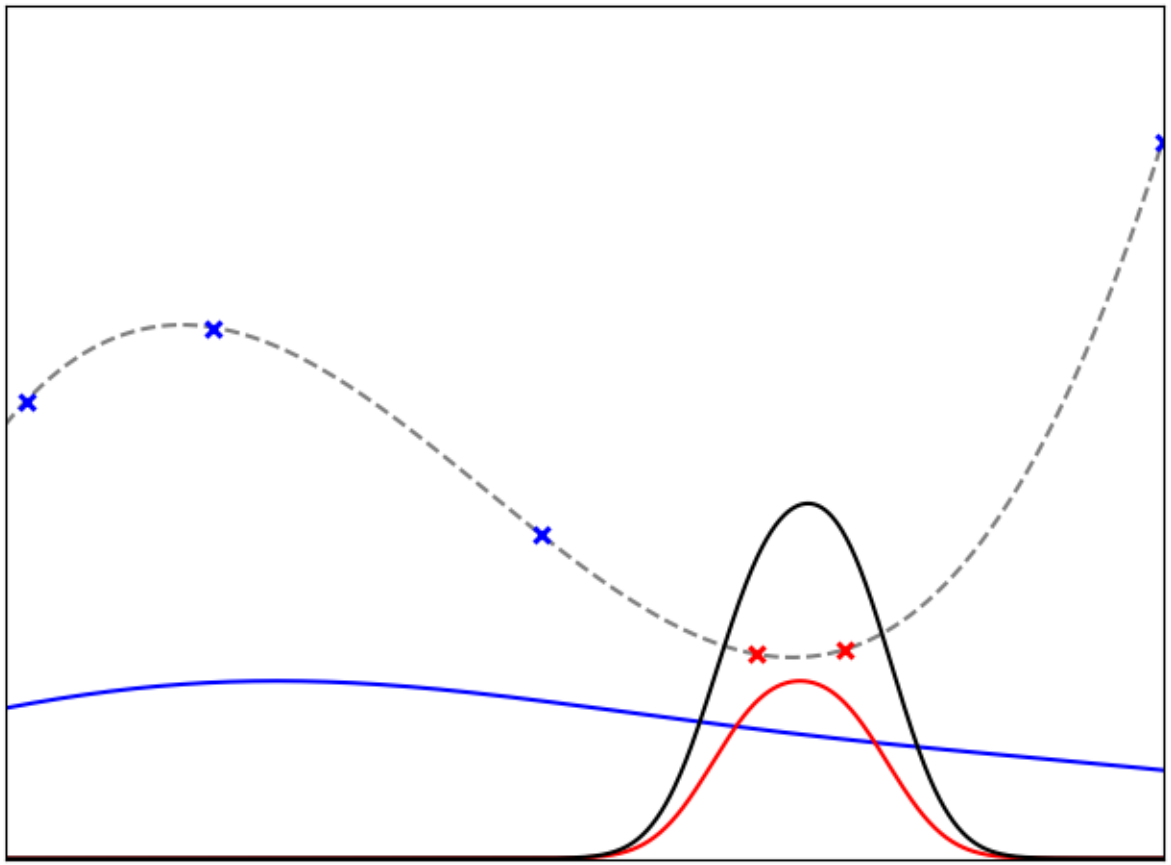
- So far not studied for:
  - High dimensionality
  - Conditional hyperparameters



Tree of Parzen Estimators (TPE)

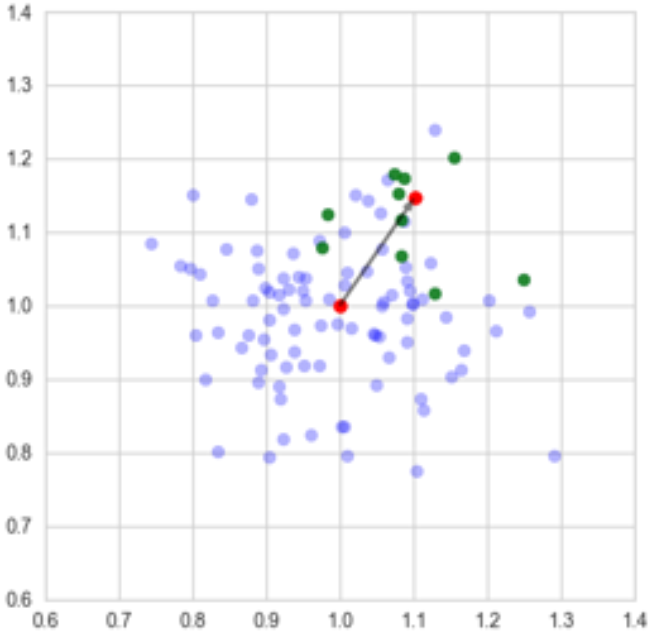
[Bergstra et al, NIPS 2011]

- Non-parametric KDEs for p(λ is good) and p(λ is bad), rather than p(y|λ)
- Equivalent to expected improvement
- Pros:
  - Efficient: O(N*d)
  - Parallelizable
  - Robust
- Cons:
  - Less sample-efficient than GPs



Population-based Methods

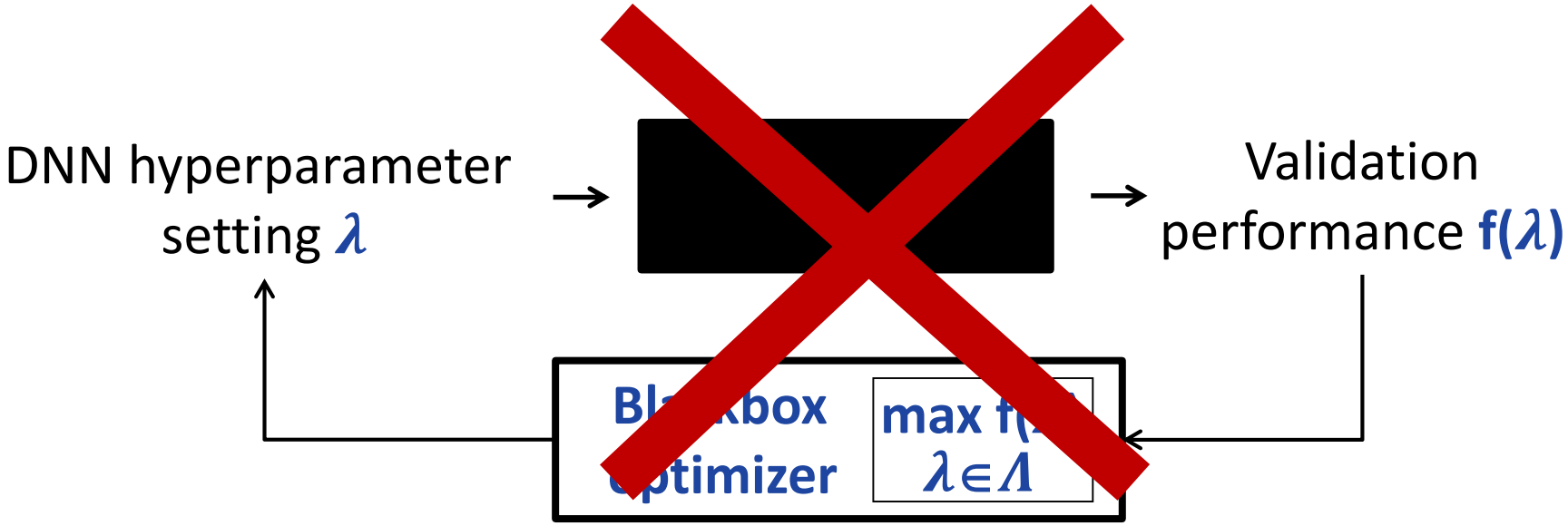
- Population of configurations
  - Maintain diversity
  - Improve fitness of population
- E.g, evolutionary strategies
  - Book: Beyer & Schwefel [2002]
  - Popular variant: CMA-ES [Hansen, 2016]
    - Very competitive for HPO of deep neural nets [Loshchilov & Hutter, 2016]
    - Embarassingly parallel
    - Purely continuous



Outline

1. Modern Hyperparameter Optimization
   - AutoML as Hyperparameter Optimization
   - Blackbox Optimization
   - Beyond Blackbox Optimization
2. Neural Architecture Search
   - Search Space Design
   - Blackbox Optimization
   - Beyond Blackbox Optimization

Beyond Blackbox Hyperparameter Optimization



Too slow for DL / big data

Main Approaches Going Beyond Blackbox HPO

- Hyperparameter gradient descent
- Extrapolation of learning curves
- Multi-fidelity optimization
- Meta-learning [part 3 of this tutorial]

Hyperparameter Gradient Descent

- Formulation as bilevel optimization problem
  [e.g., Franceschi et al, ICML 2018]

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

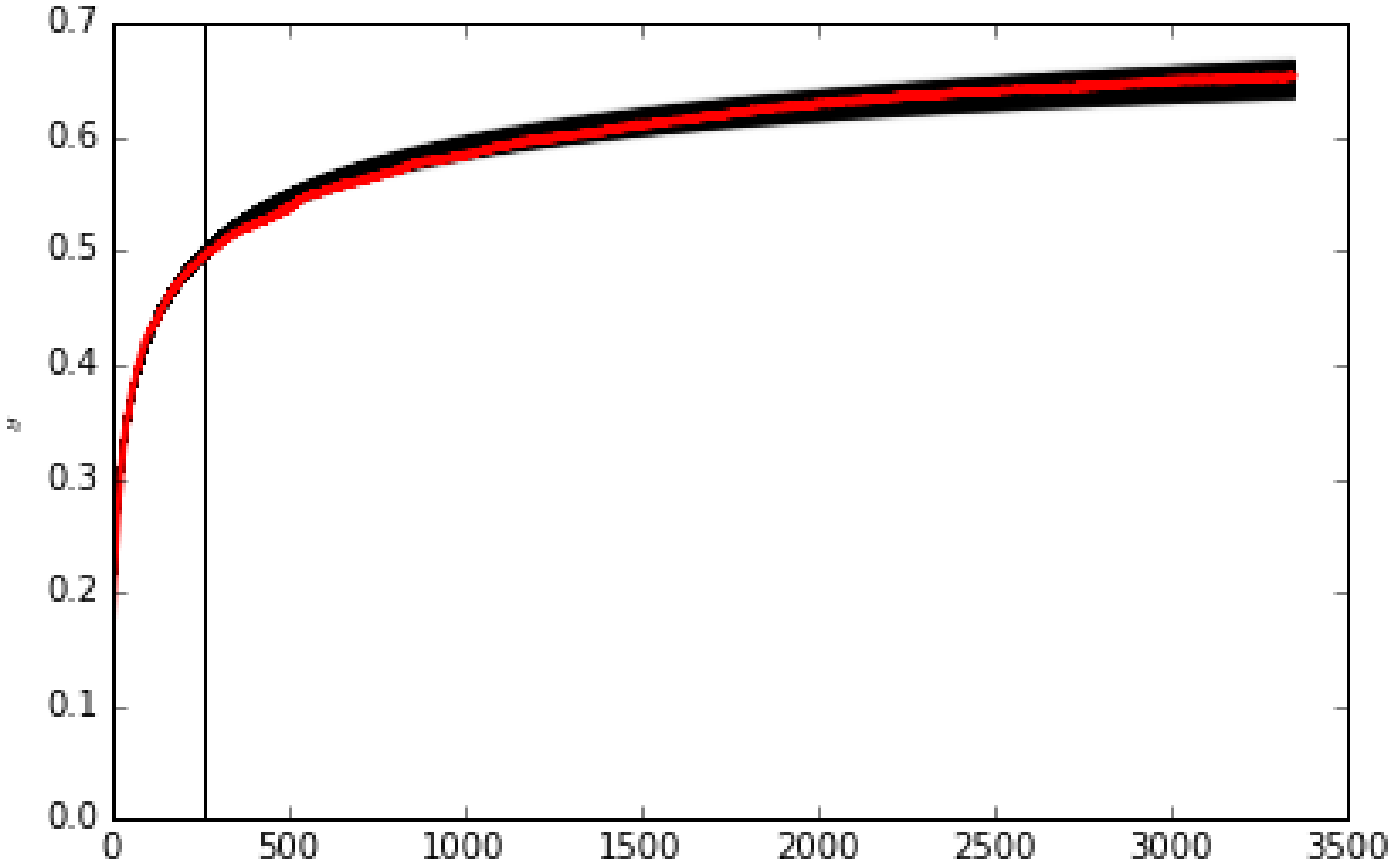
- Derive through the entire optimization process
  [MacLaurin et al, ICML 2015]

- 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



- Parametric learning curve models [Domhan et al, IJCAI 2015]
- Bayesian neural networks [Klein et al, ICLR 2017]

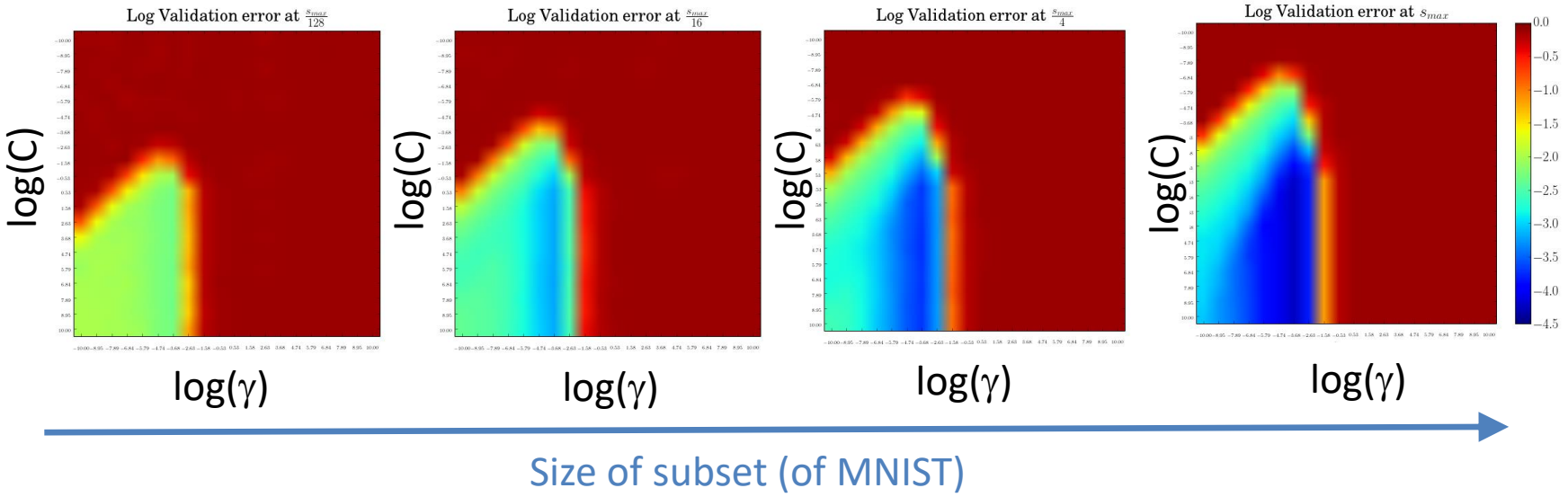
Multi-Fidelity Optimization

- 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

  - Also applicable in different domains, e.g., fluid simulations:
    - Less particles
    - Shorter simulations

Multi-fidelity Optimization

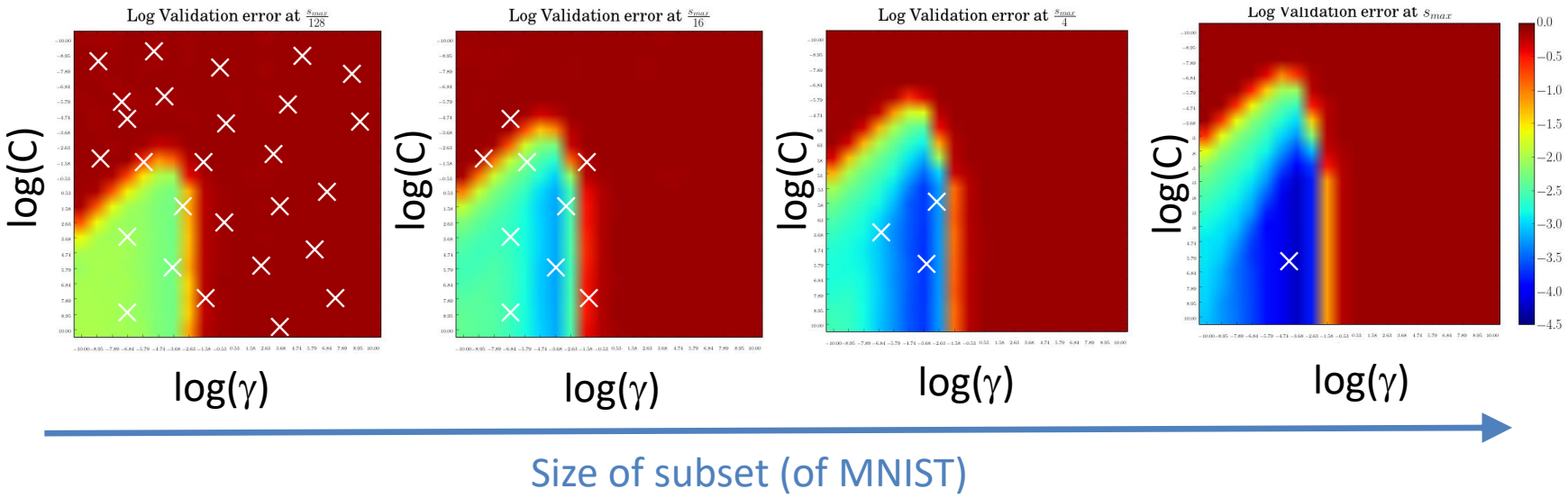
- **Make use of cheap low-fidelity evaluations**
  - E.g.: subsets of the data (here: SVM on MNIST)



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

Multi-fidelity Optimization

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

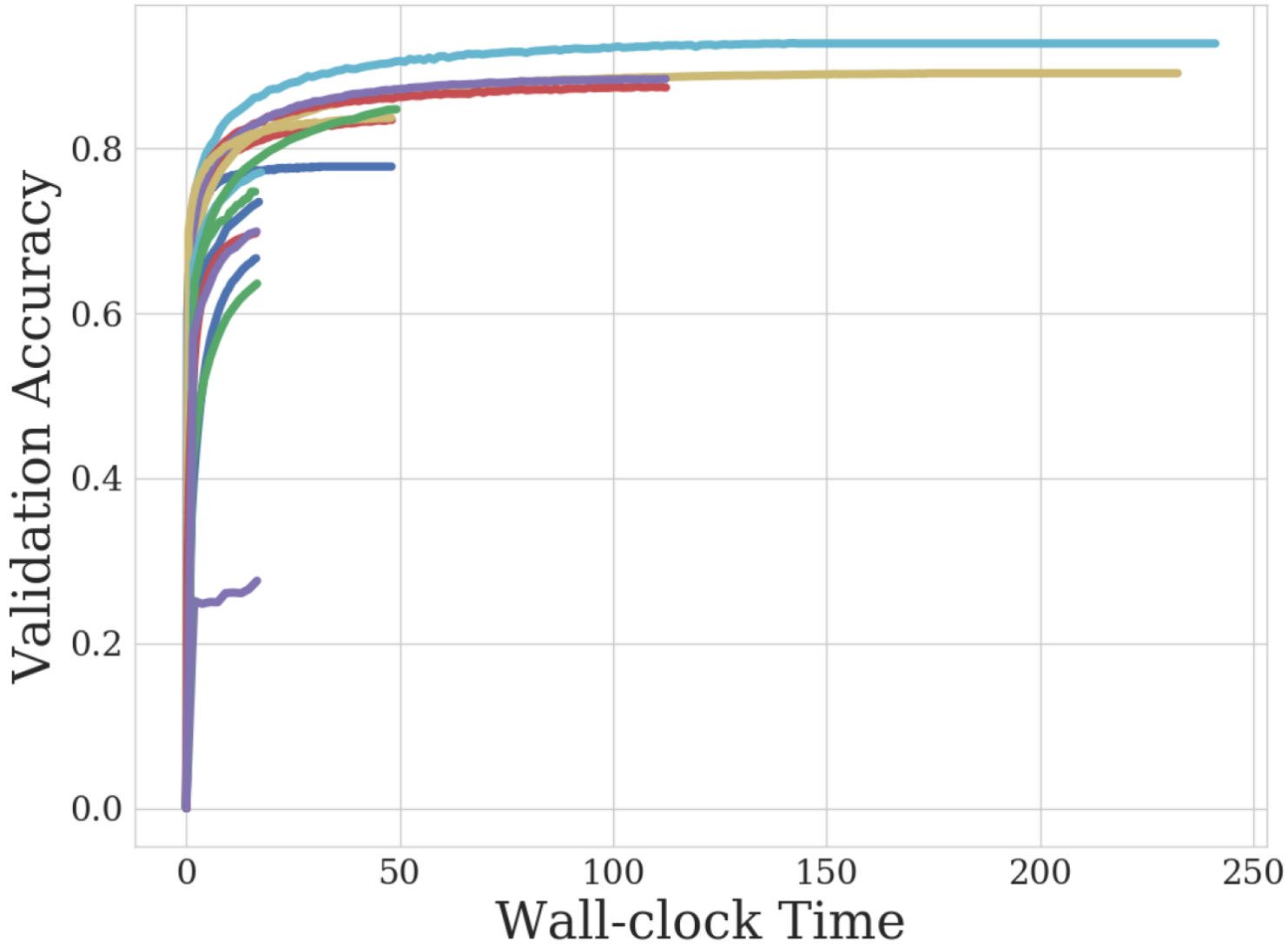


  - Fit a Gaussian process model f(λ,b) to predict performance as a function of hyperparameters λ and budget b
  - Choose both λ and budget b to maximize "bang for the buck"

[Swersky et al, NIPS 2013; Swersky et al, arXiv 2014; Klein et al, AISTATS 2017; Kandasamy et al, ICML 2017]

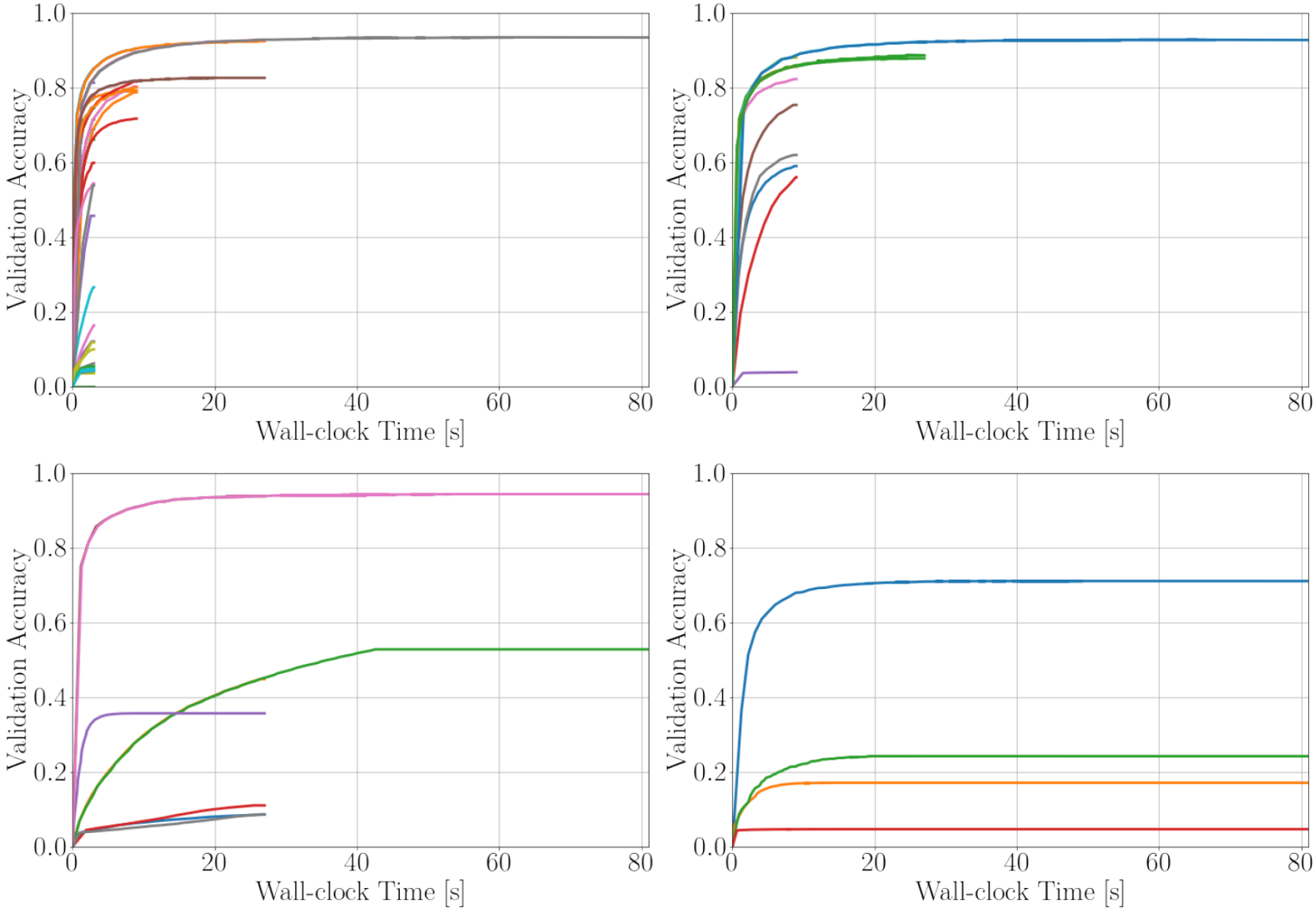
A Simpler Approach: Successive Halving (SH)

[Jamieson & Talwalkar, AISTATS 2016]



Hyperband (its first 4 calls to SH)

[Li et al, ICLR 2017]

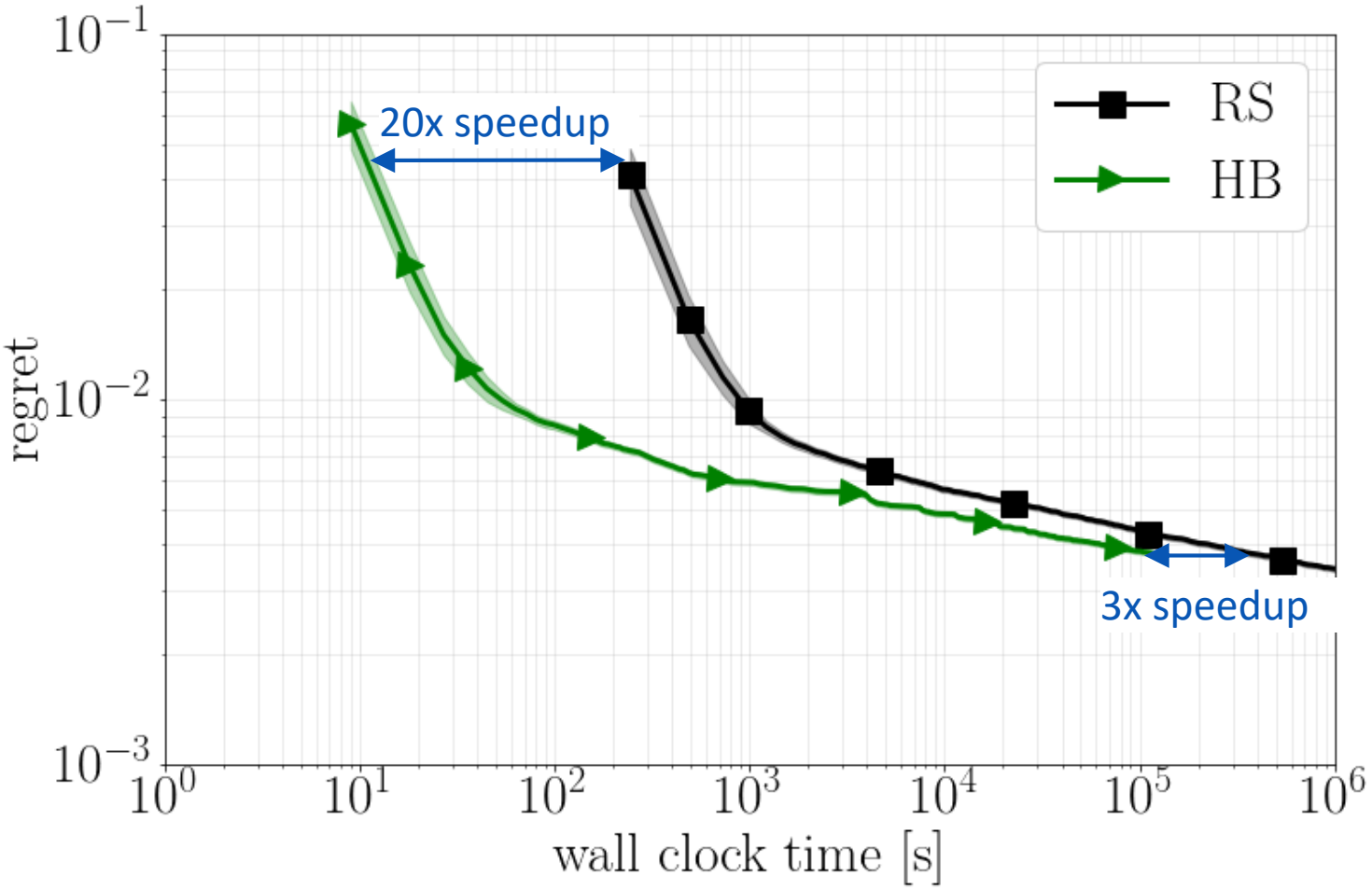


BOHB: Bayesian Optimization & Hyperband

[Falkner, Klein & Hutter, ICML 2018]

- Advantages of Hyperband
  - Strong anytime performance
  - General-purpose
    - Low-dimensional continuous spaces
    - High-dimensional spaces with conditionality, categorical dimensions, etc
  - Easy to implement
  - Scalable
  - Easily parallelizable
- Advantage of Bayesian optimization: strong final performance
- Combining the best of both worlds in BOHB
  - Bayesian optimization
    - for choosing the configuration to evaluate (using a TPE variant)
  - Hyperband
    - for deciding how to allocate budgets

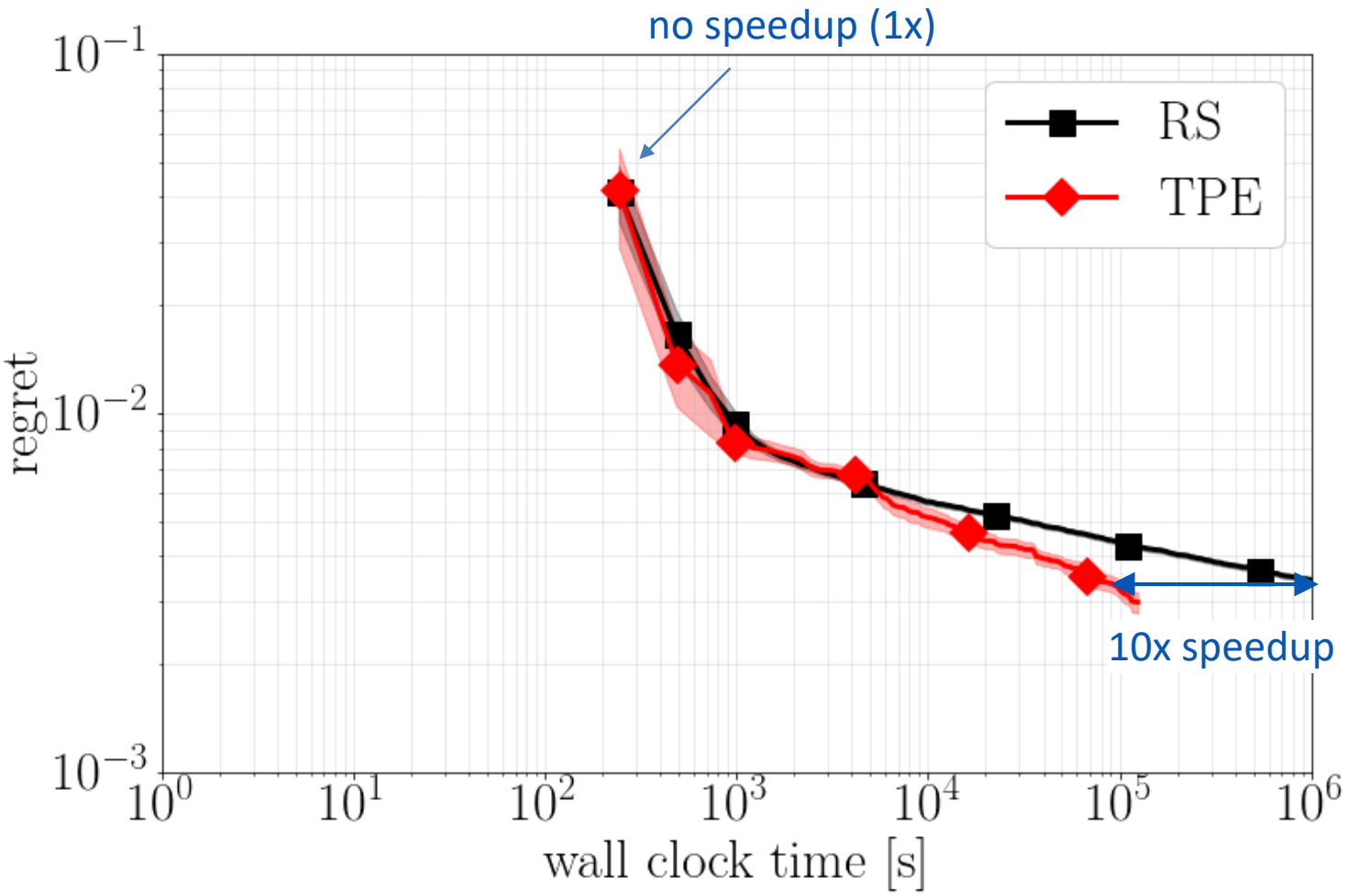
Hyperband vs. Random Search



Biggest advantage: much improved **anytime performance**

Auto-Net on dataset adult

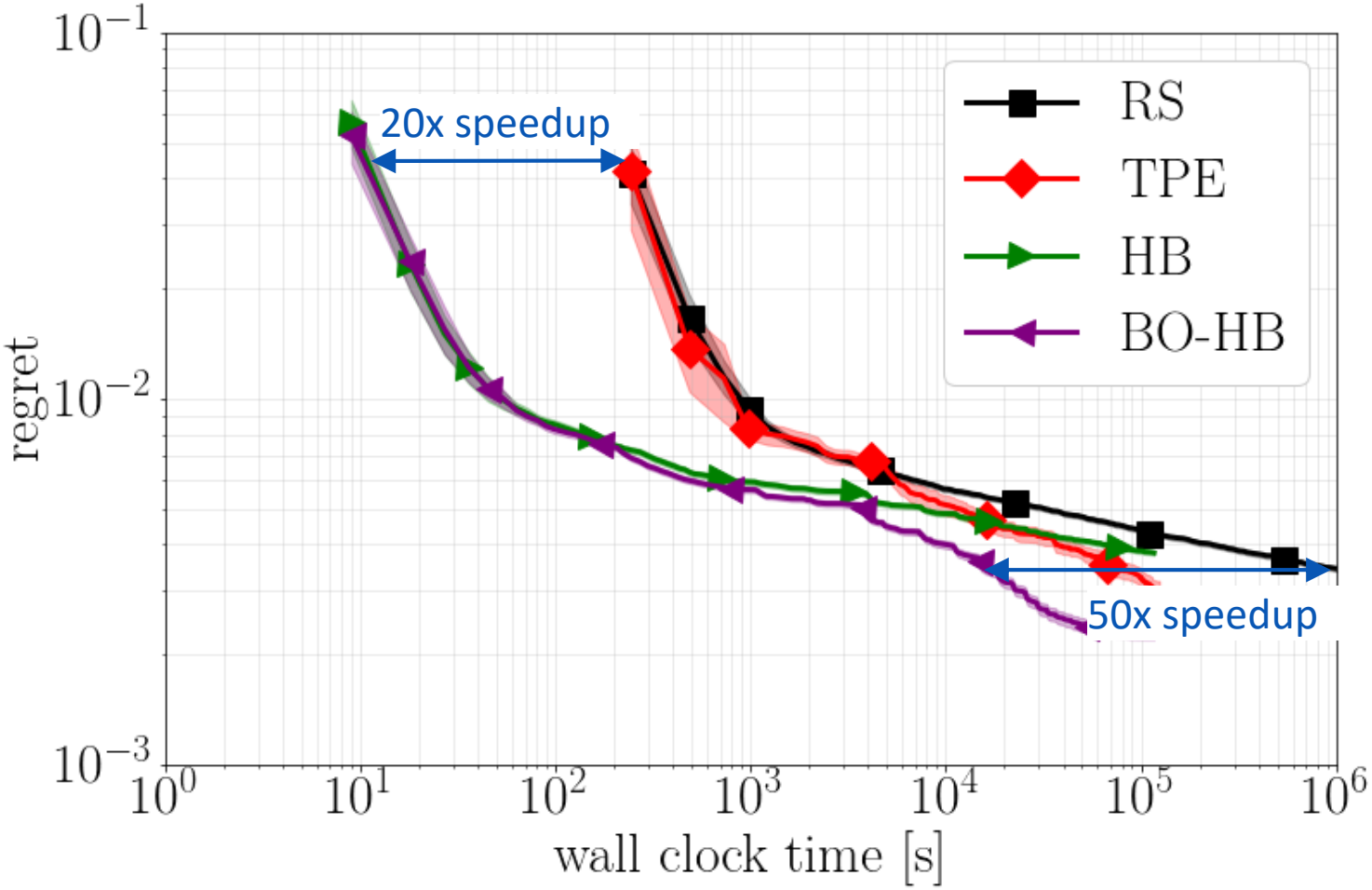
Bayesian Optimization vs Random Search



Biggest advantage: much improved **final performance**

Auto-Net on dataset adult

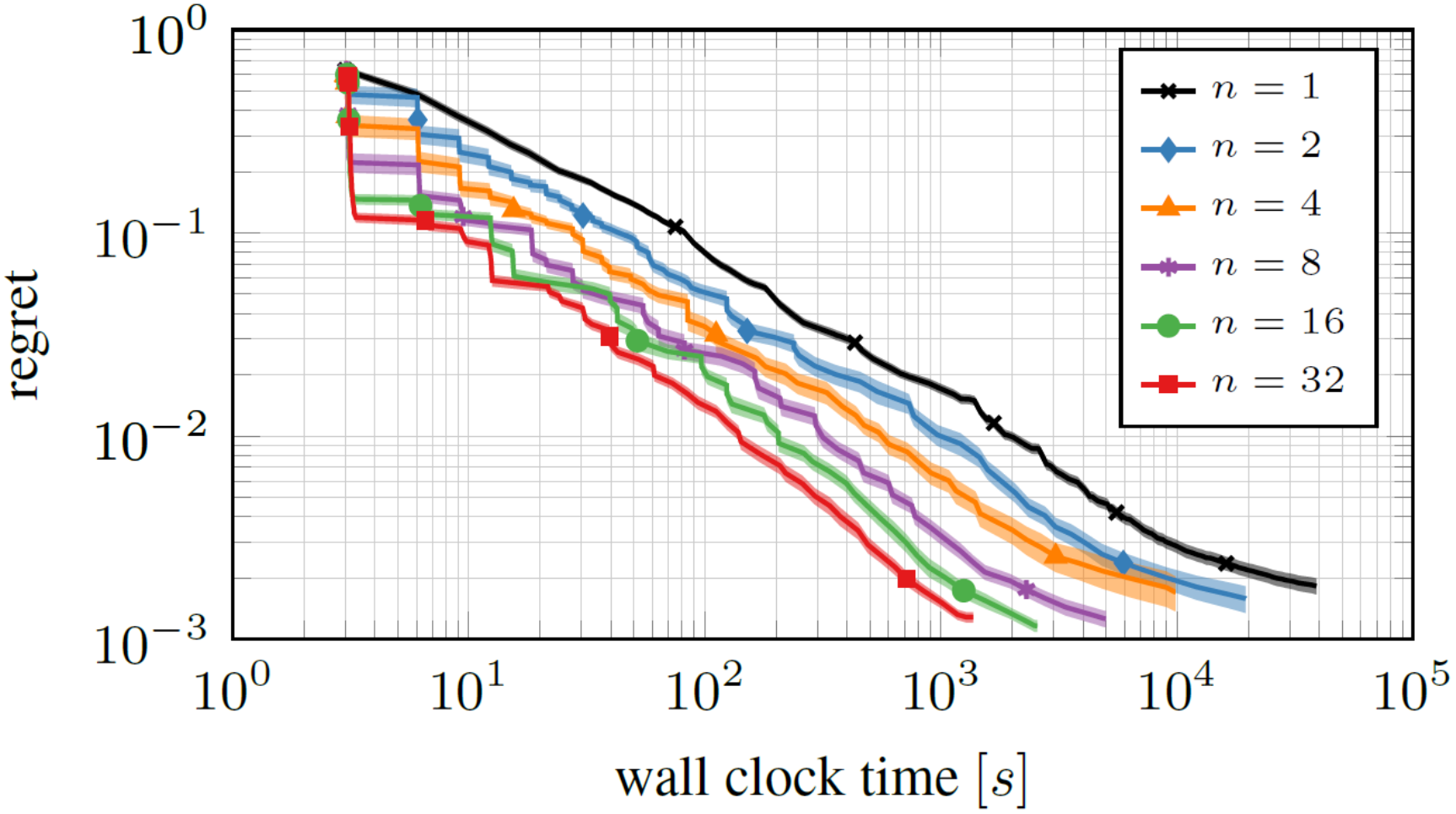
Combining Bayesian Optimization & Hyperband



Best of both worlds: strong **anytime and final performance**

Auto-Net on dataset adult

Almost Linear Speedups By Parallelization



Auto-Net on dataset letter

HPO for Practitioners: Which Tool to Use?

- If you have access to multiple fidelities
  - We recommend BOHB [Falkner et al, ICML 2018]
  - https://github.com/automl/HpBandSter
  - Combines the advantages of TPE and Hyperband

- If you do not have access to multiple fidelities
  - Low-dim. continuous: GP-based BO (e.g., Spearmint)
  - High-dim, categorical, conditional: SMAC or TPE
  - Purely continuous, budget >10x dimensionality: CMA-ES

Open-source AutoML Tools based on HPO

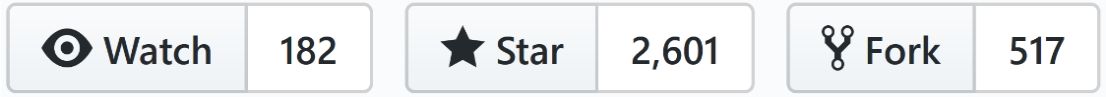
- Auto-WEKA [Thornton et al, KDD 2013]
  - 768 hyperparameters, 4 levels of conditionality
  - Based on WEKA and SMAC
- Hyperopt-sklearn [Komer et al, SciPy 2014]
  - Based on scikit-learn & TPE
- Auto-sklearn [Feurer al, NIPS 2015]
  - Based on scikit-learn & SMAC / BOHB
  - Uses meta-learning and posthoc ensembling
  - Won AutoML competitions 2015-2016 & 2017-2018
- TPOT [Olson et al, EvoApplications 2016]
  - Based on scikit-learn and evolutionary algorithms
- H2O AutoML [so far unpublished]
  - Based on random search and stacking

AutoML: Democratization of Machine Learning

- Auto-sklearn also won the last two phases of the AutoML challenge human track (!)
  - It performed better than up to 130 teams of human experts
  - It is open-source (BSD) and trivial to use:

```
import autosklearn.classification as cls
automl = cls.AutoSklearnClassifier()
automl.fit(X_train, y_train)
y_hat = automl.predict(X_test)
```

https://github.com/automl/auto-sklearn



→ Effective machine learning for everyone!

Example Application: Robotic Object Handling

- Collaboration with Freiburg's robotics group
- Binary classification task for object placement: will the object fall over?

Video credit: Andreas Eitel

- Dataset
  - 30000 data points
  - 50 features -- manually defined [BSc thesis, Hauff 2015]
- Performance
  - Caffe framework & BSc student for 3 months: 2% error rate
  - **Auto-sklearn: 0.6% error rate** (within 30 minutes)

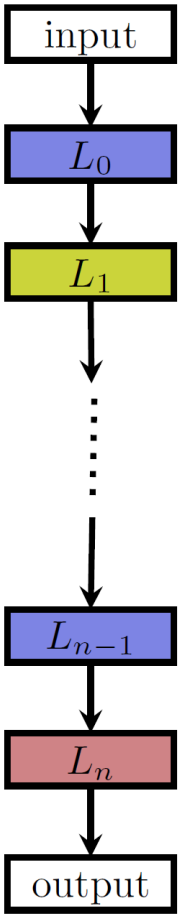
Outline

1. Modern Hyperparameter Optimization
   - AutoML as Hyperparameter Optimization
   - Blackbox Optimization
   - Beyond Blackbox Optimization

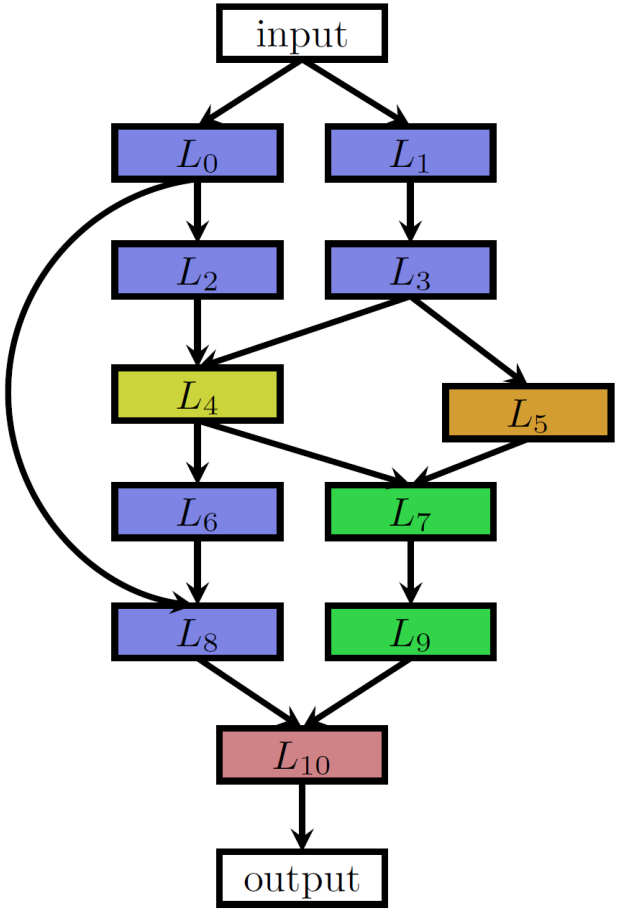
2. Neural Architecture Search
   - Search Space Design
   - Blackbox Optimization
   - Beyond Blackbox Optimization

Based on: Elsken, Metzen and Hutter
[Neural Architecture Search: a Survey, arXiv 2018; also Chapter 3 of the AutoML book]

Basic Neural Architecture Search Spaces



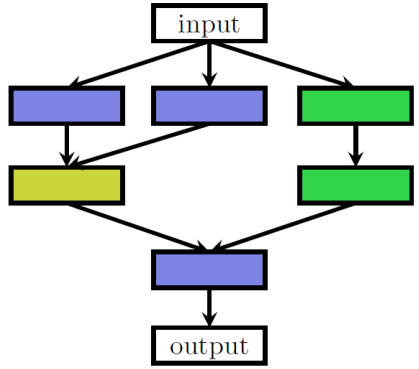
Chain-structured space
(different colours:
different layer types)

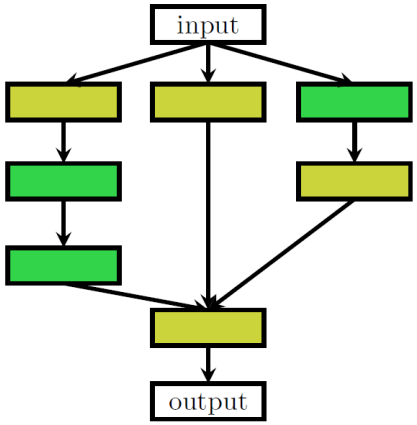


More complex space
with multiple branches
and skip connections

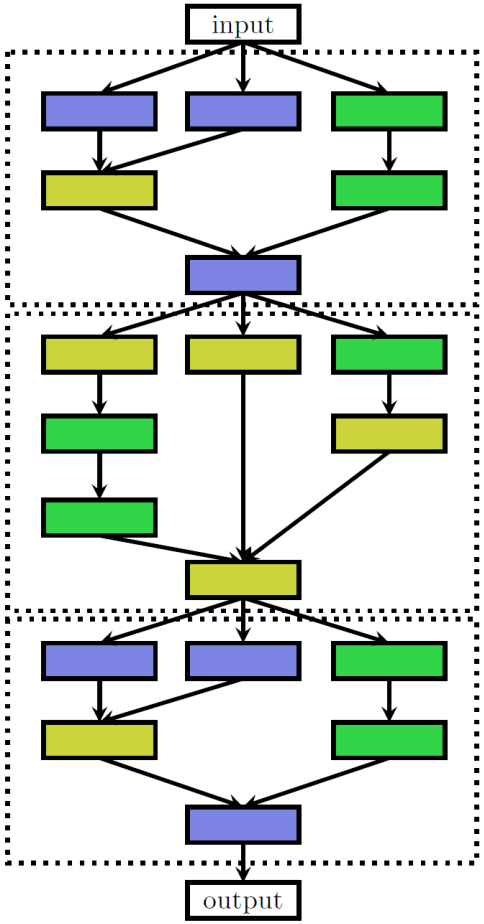
Cell Search Spaces

Introduced by Zoph et al [CVPR 2018]





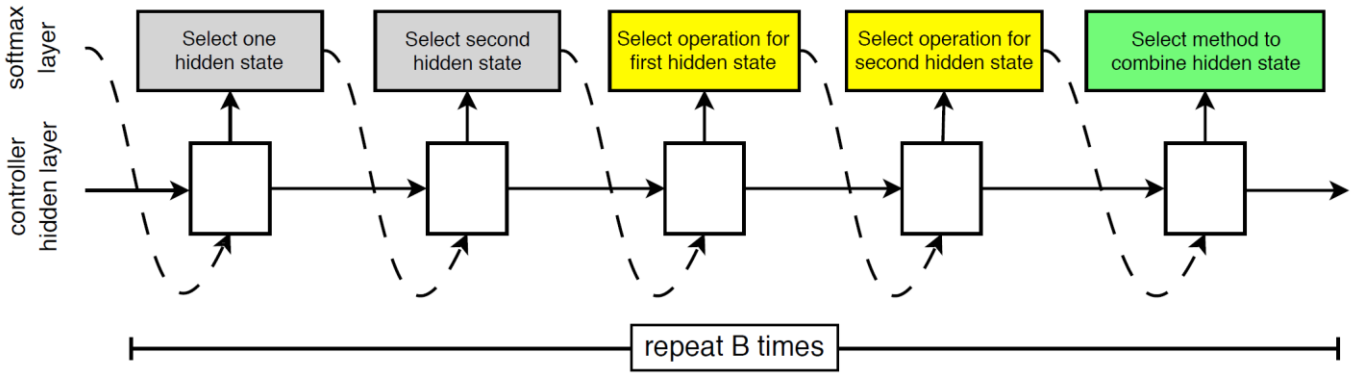
Two possible cells



Architecture composed of stacking together individual cells

NAS as Hyperparameter Optimization

- Cell search space by Zoph et al [CVPR 2018]



  - 5 categorical choices for Nth block:
    - 2 categorical choices of hidden states, each with domain {0, ..., N-1}
    - 2 categorical choices of operations
    - 1 categorical choice of combination method

    → Total number of hyperparameters for the cell: 5B (with B=5 by default)

- Unrestricted search space
  - Possible with conditional hyperparameters (but only up to a prespecified maximum number of layers)
  - Example: chain-structured search space
    - Top-level hyperparameter: number of layers L
    - Hyperparameters of layer k conditional on L >= k

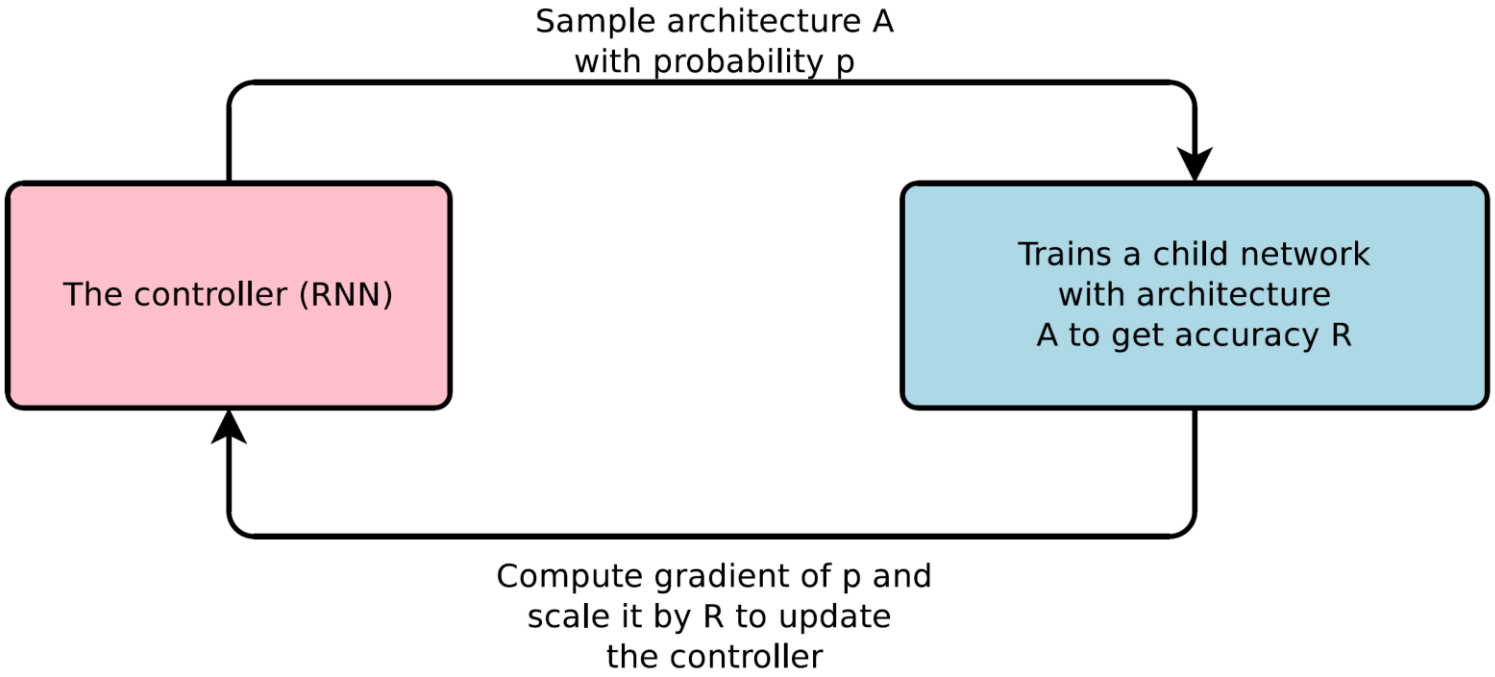
Outline

1. Modern Hyperparameter Optimization
   - AutoML as Hyperparameter Optimization
   - Blackbox Optimization
   - Beyond Blackbox Optimization

2. Neural Architecture Search
   - Search Space Design
   - Blackbox Optimization
   - Beyond Blackbox Optimization

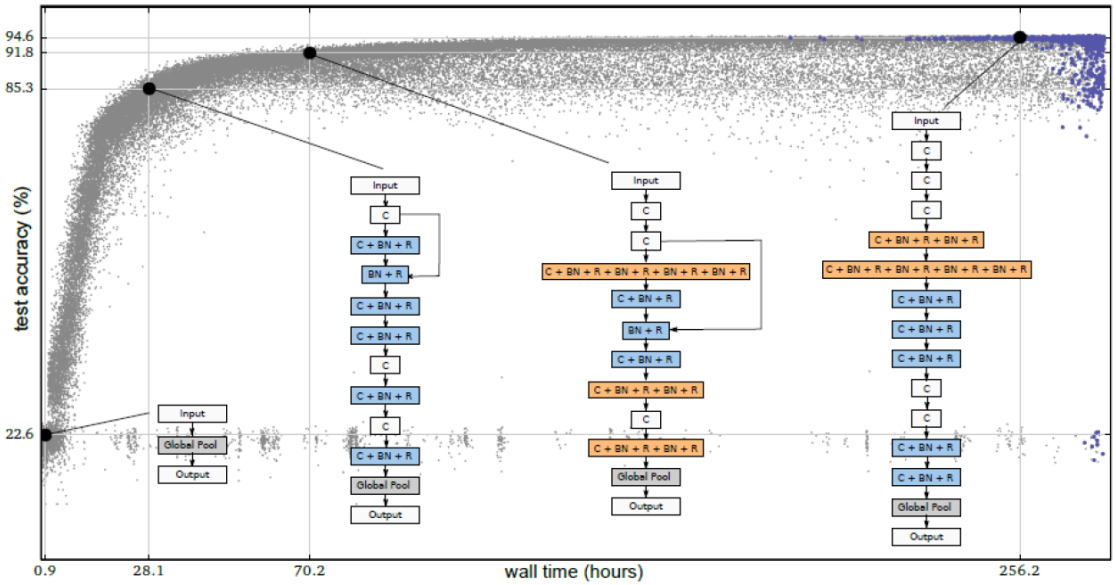
Reinforcement Learning

- NAS with Reinforcement Learning [Zoph & Le, ICLR 2017]
  - State-of-the-art results for CIFAR-10, Penn Treebank
  - Large computational demands
    - **800 GPUs for 3-4 weeks, 12.800 architectures evaluated**



Evolution

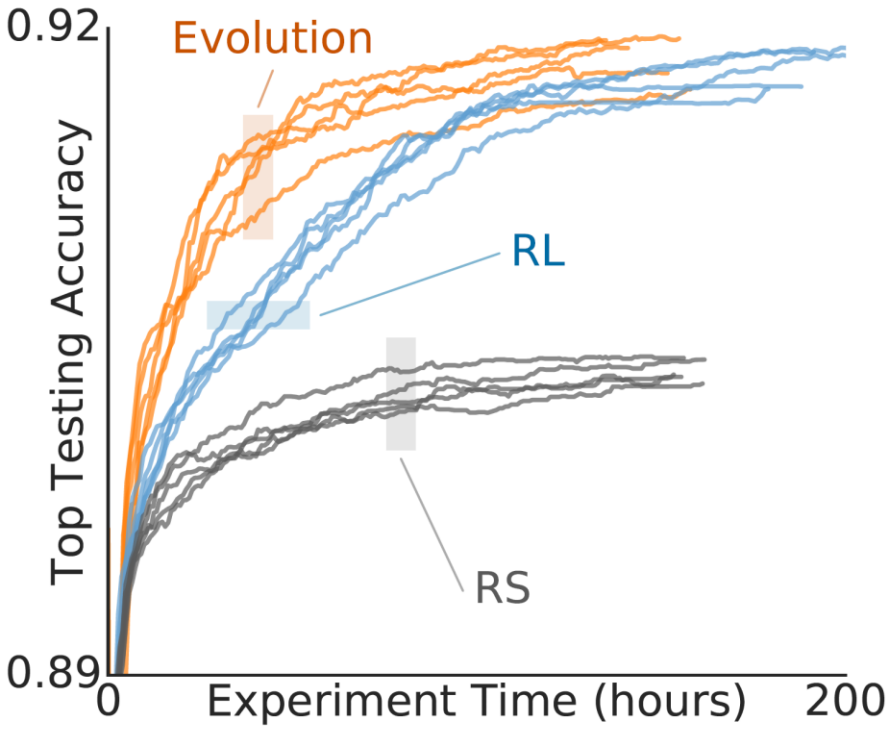
- Neuroevolution (already since the 1990s)
  - Typically optimized both architecture and weights with evolutionary methods
    [e.g., Angeline et al, 1994; Stanley and Miikkulainen, 2002]
  - Mutation steps, such as adding, changing or removing a layer
    [Real et al, ICML 2017; Miikkulainen et al, arXiv 2017]



Regularized / Aging Evolution

- Standard evolutionary algorithm [Real et al, AAAI 2019]
  - But oldest solutions are dropped from the population (even the best)
- State-of-the-art results (CIFAR-10, ImageNet)
  - Fixed-length cell search space

Comparison of evolution, RL and random search



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, BayesOpt 2013]
  - NASBOT [Kandasamy et al, NIPS 2018]
- Sequential model-based optimization
  - PNAS [Liu et al, ECCV 2018]

Outline

1. Modern Hyperparameter Optimization
   - AutoML as Hyperparameter Optimization
   - Blackbox Optimization
   - Beyond Blackbox Optimization
2. Neural Architecture Search
   - Search Space Design
   - Blackbox Optimization
   - Beyond Blackbox Optimization

Main approaches for making NAS efficient

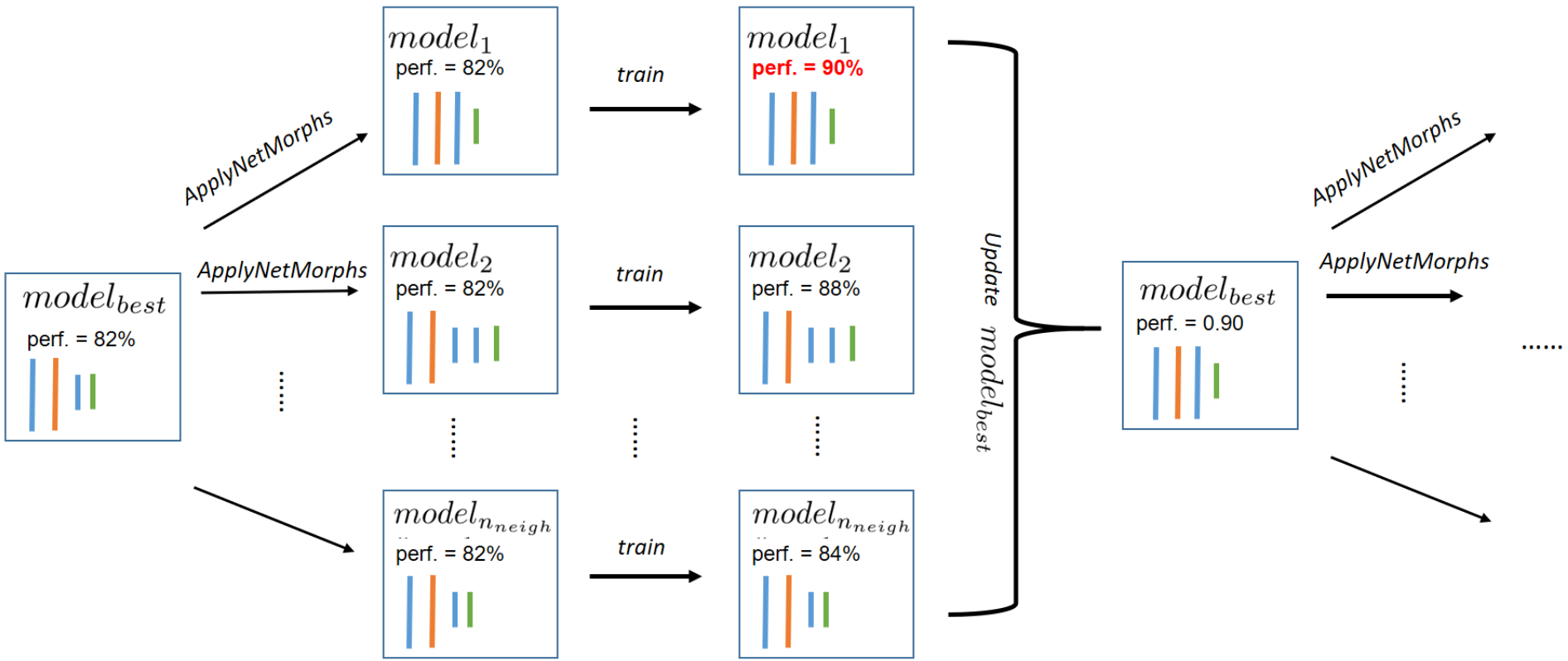
- Weight inheritance & network morphisms
- Weight sharing & one-shot models
- Multi-fidelity optimization
  [Zela et al, AutoML 2018, Runge et al, MetaLearn 2018]
- Meta-learning [Wong et al, NIPS 2018]

Network morphisms

- Network morphisms [Chen et al, 2016; Wei et al, 2016; Cai et al, 2017]
  - 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 morphism
  - Allow efficient moves in architecture space

Weight inheritance & network morphisms

[Cai et al, AAAI 2018; Elsken et al, MetaLearn 2017; Cortes et al, ICML 2017; Cai et al, ICML 2018]



→ enables efficient architecture search

Weight Sharing & One-shot Models

- Convolutional Neural Fabrics [Saxena & Verbeek, NIPS 2016]
  - Embed an exponentially large number of architectures
  - Each path through the fabric is an architecture

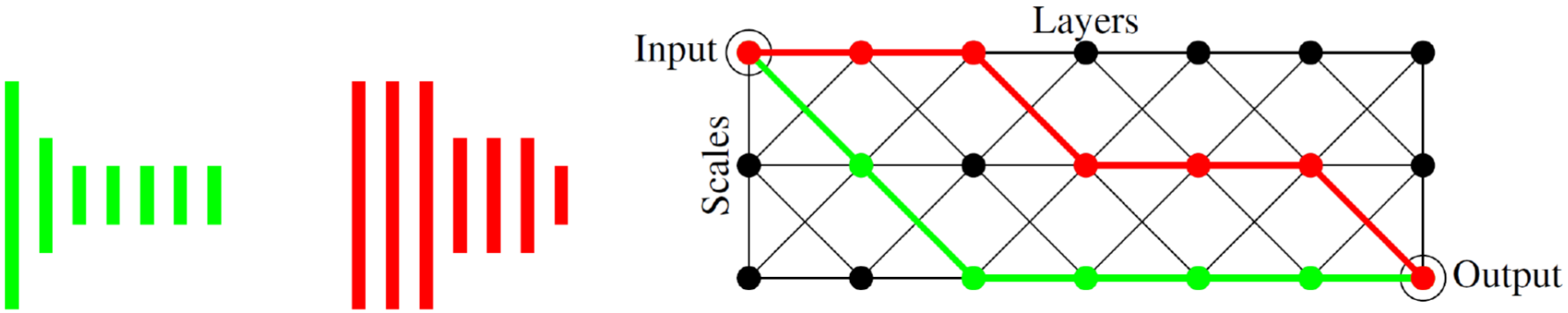
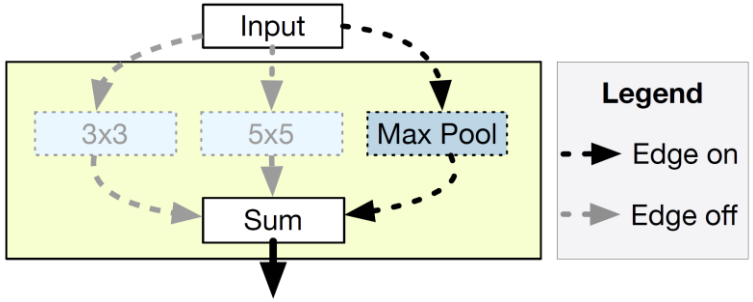


Figure: Fabrics embedding two 7-layer CNNs (red, green).
Feature map sizes of the CNN layers are given by height.

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

DARTS: Differentiable Neural Architecture Search

[Liu et al, Simonyan, Yang, arXiv 2018]

- Relax the discrete NAS problem
  - One-shot model with continuous architecture weight α for each operator
  - Use a similar approach as Luketina et al [ICML'16] to interleave optimization steps of α (using validation error) and network weights

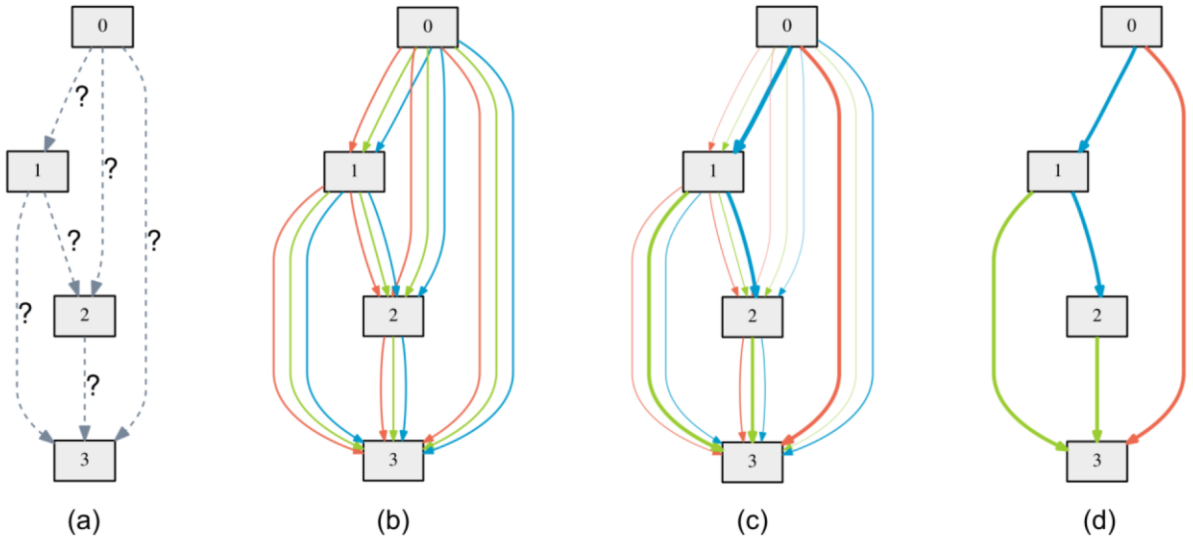


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

Some Promising Work Under Review

- Anonymous ICLR submissions based on DARTS
  - SNAS: Use Gumbel softmax on architecture weights α [link]
  - Single shot NAS: use L1 penalty to sparsify architecture [link]
  - Proxyless NAS: (PyramidNet-based) memory-efficient variant of DARTS that trains sparse architectures only [link]
- Graph hypernetworks for NAS [Anonymous ICLR submission]
- Multi-objective NAS
  - MNasNet: scalarization [Tan et al, arXiv 2018]
  - LEMONADE: evolution & (approximate) network morphisms [Anonymous ICLR submission]

Remarks on Experimentation in NAS

- Final results are often incomparable due to
  - Different training pipelines without available source code
    - Releasing the final architecture does not help for comparisons
  - Different hyperparameter choices
    - Very different hyperparameters for training and final evaluation
  - Different search spaces / initial models
    - Starting from random or from PyramidNet?

→ Need for looking beyond the error numbers on CIFAR

→ Need for benchmarks including training pipeline & hyperparams

- Experiments are often very expensive

→ Need for cheap benchmarks that allow for many runs

HPO and NAS Wrapup

- Exciting research fields, lots of progress
- Several ways to speed up blackbox optimization
  - Multi-fidelity approaches
  - Hyperparameter gradient descent
  - Weight inheritance
  - Weight sharing & hypernetworks
- More details in AutoML book: automl.org/book
- Advertisement: we‘re building up an Auto-DL team
  - Building research library of building blocks for efficient NAS
  - Building open-source framework Auto-PyTorch
  - We have several openings on all levels (postdocs, PhD students, research engineers); see automl.org/jobs

AutoML and Job Loss Through Automation

- Concern about too much automation, job loss
  - AutoML will allow humans to become more productive
  - Thus, it will eventually reduce the work left for data scientists
  - But it will also help many domain scientists use machine learning that would otherwise not have used it
    - This creates more demand for interesting and creative work

- Call to arms: let‘s use AutoML to create and improve jobs
  - If you can think of a business opportunity that‘s made feasible by AutoML (robust, off-the-shelf, effective ML), now is a good time to act on it ...

AutoML: Further Benefits and Concerns

+ Democratization of data science
+ We directly have a strong baseline
+ We can codify best practices
+ 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

NeurIPS 2018 Tutorial on Automatic Machine Learning

Learning to *Learn*

automl.org/events -> AutoML Tutorial -> Slides

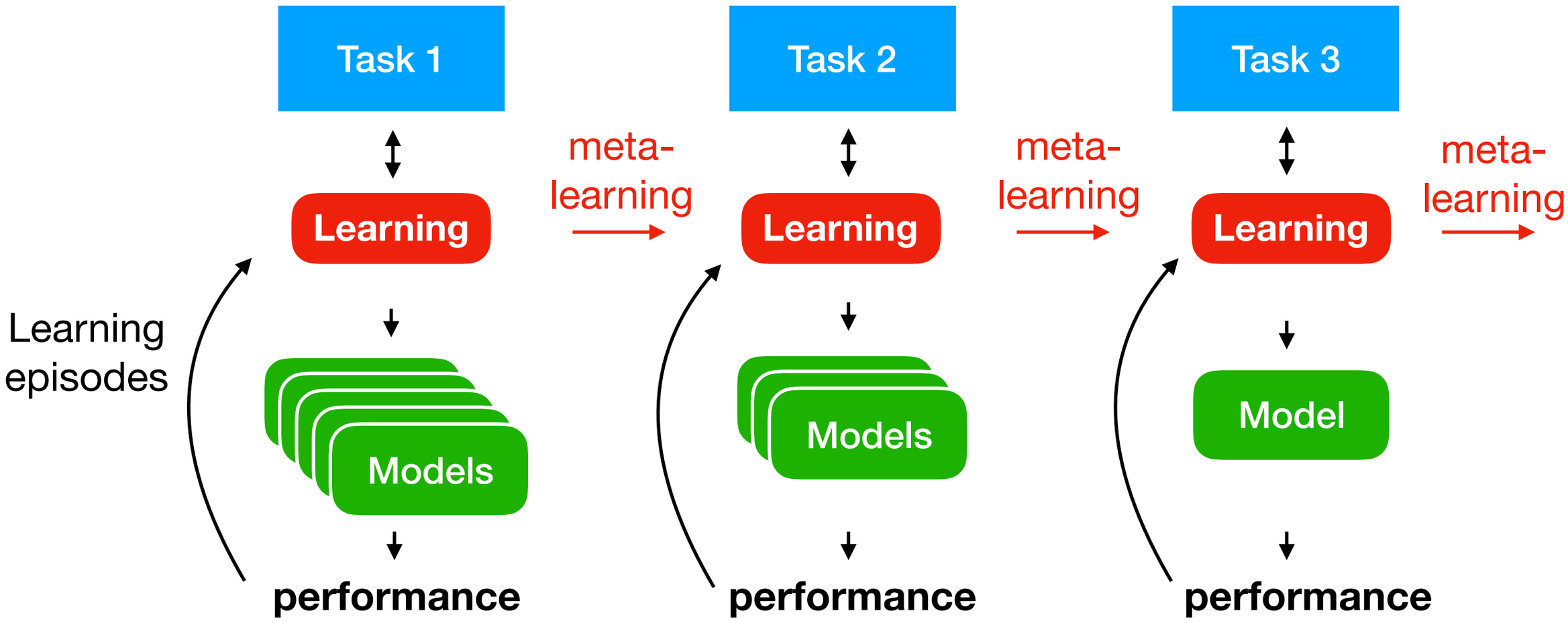
Frank Hutter
University of Freiburg
fh@cs.uni-freiburg.de

Joaquin Vanschoren
Eindhoven University of Technology
j.vanschoren@tue.nl
@joavanschoren

Learning is a never-ending process

Tasks come and go, but learning is forever

Learn more effectively: less trial-and-error, less data

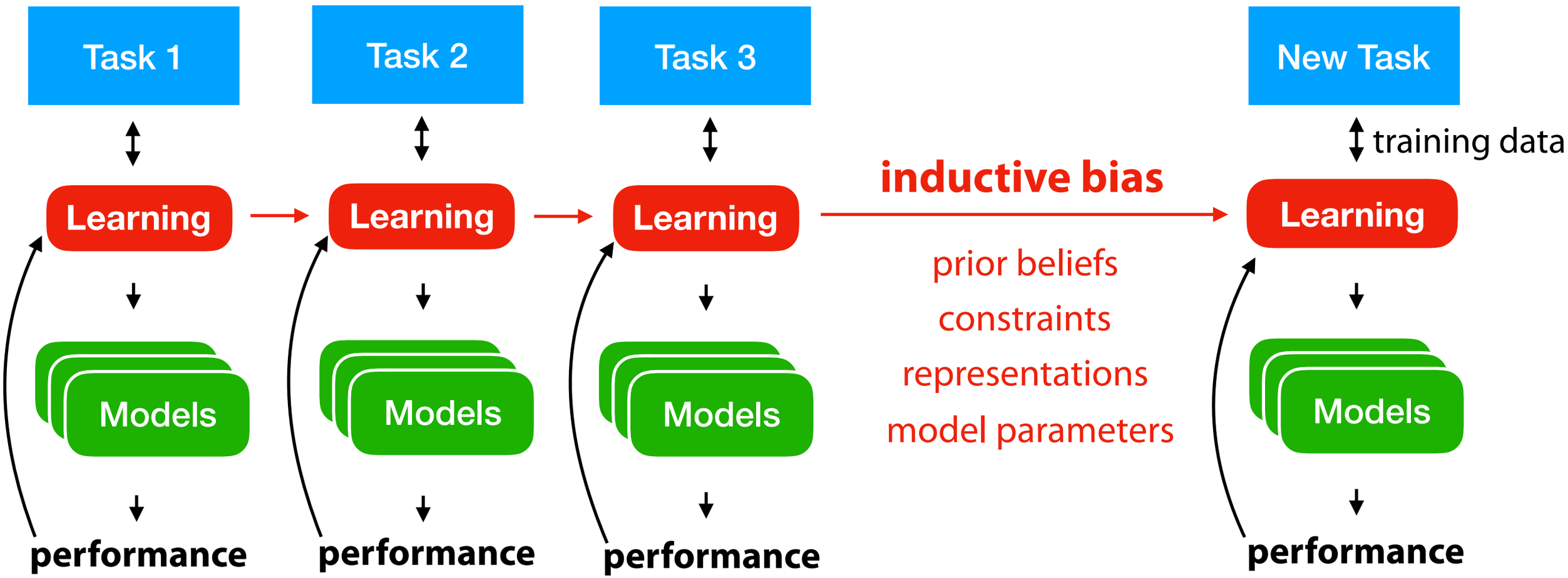


Learning to learn

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

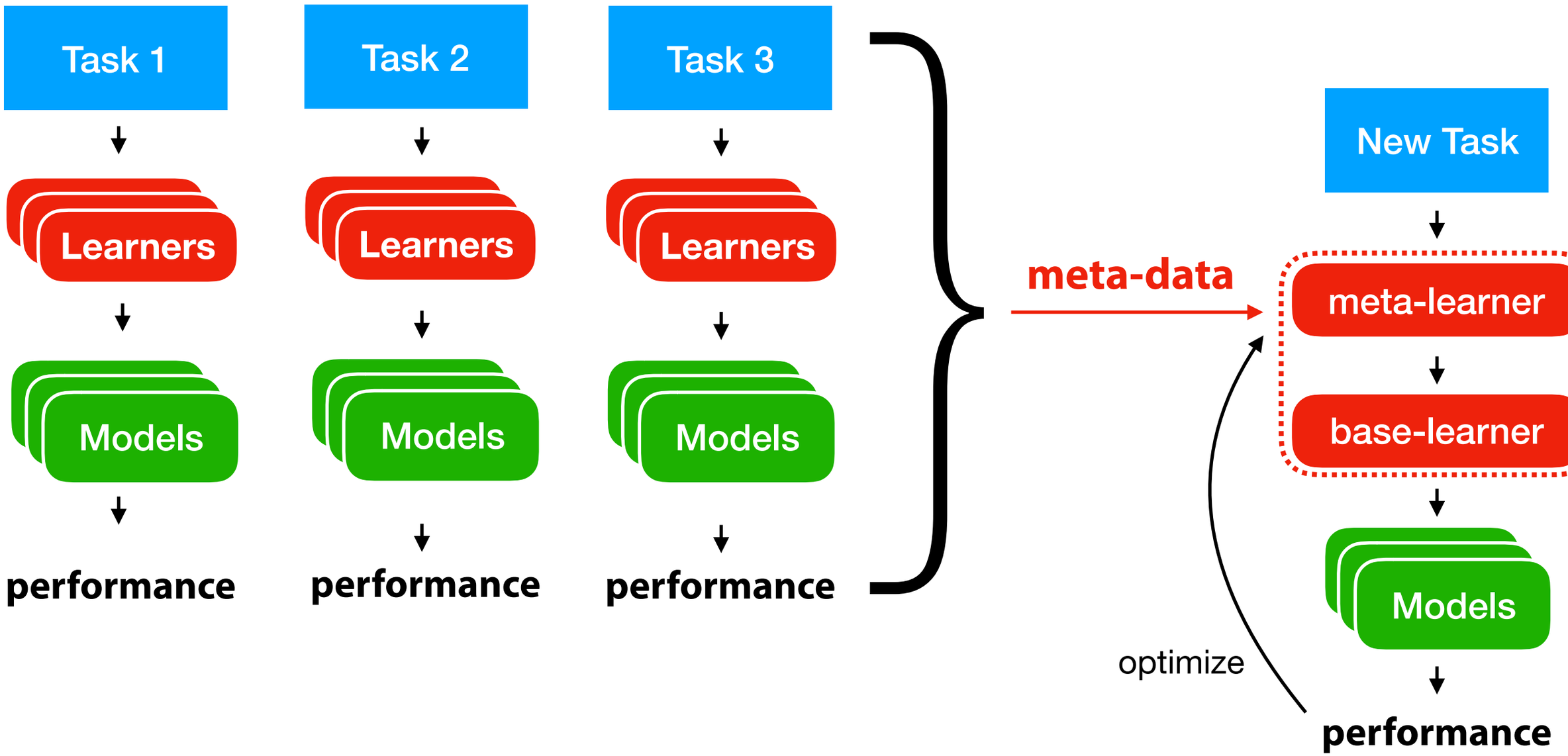
(if not it may actually harm learning)



Meta-learning

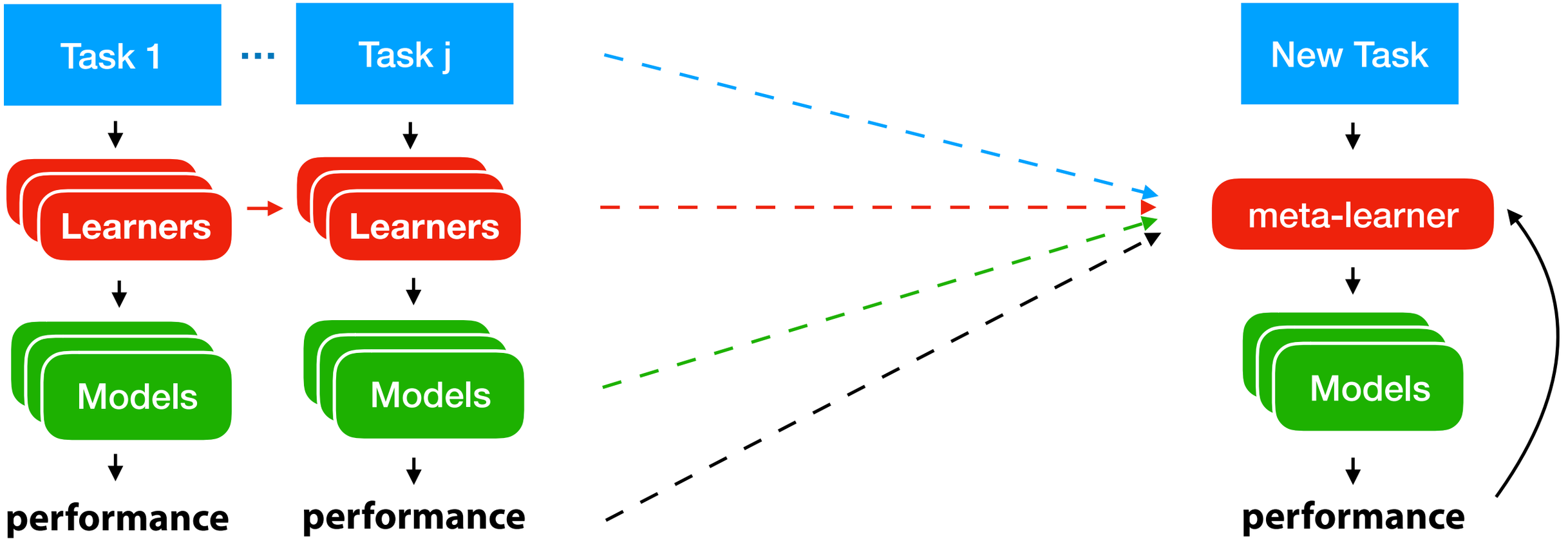
Collect meta-data about learning episodes and learn from them

Meta-learner *learns* a (base-)learning algorithm, *end-to-end*



Three approaches for increasingly similar tasks

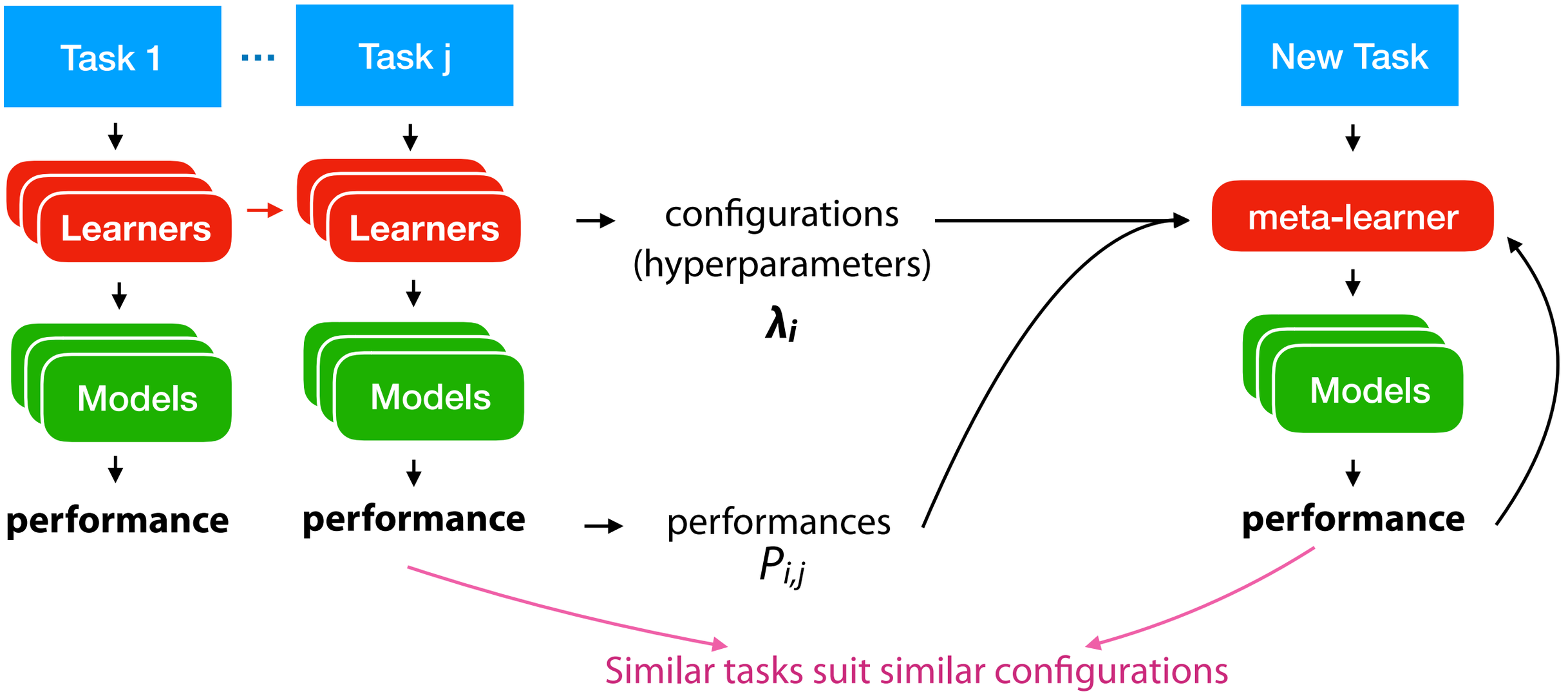
1. Transfer prior knowledge about what generally works well
2. Reason about model performance across tasks
3. Start from models trained earlier on similar tasks



1. Learning from prior evaluations

Configurations: settings that uniquely define the model

(algorithm, pipeline, neural architecture, hyper-parameters, …)

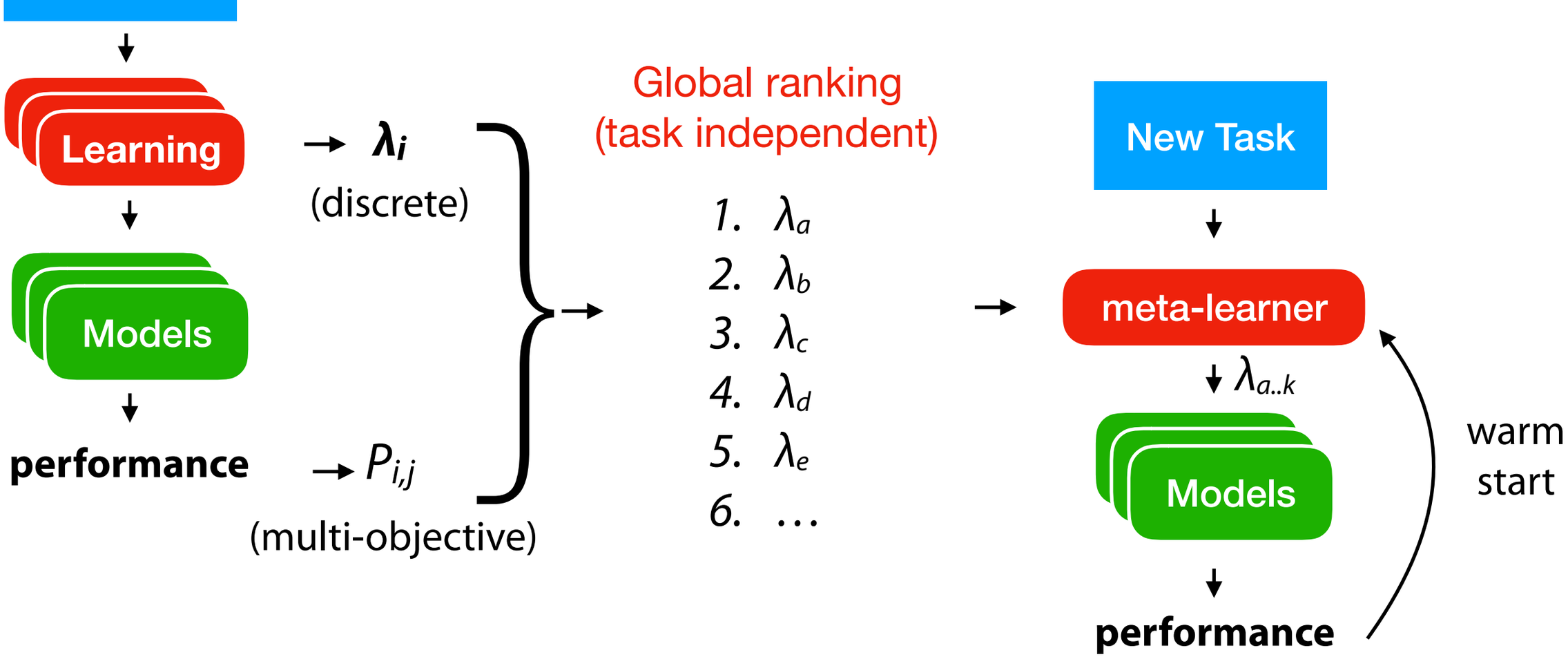


[Leite et al. 2012]

[Abdulrahman et al. 2018]

Top-K recommendation

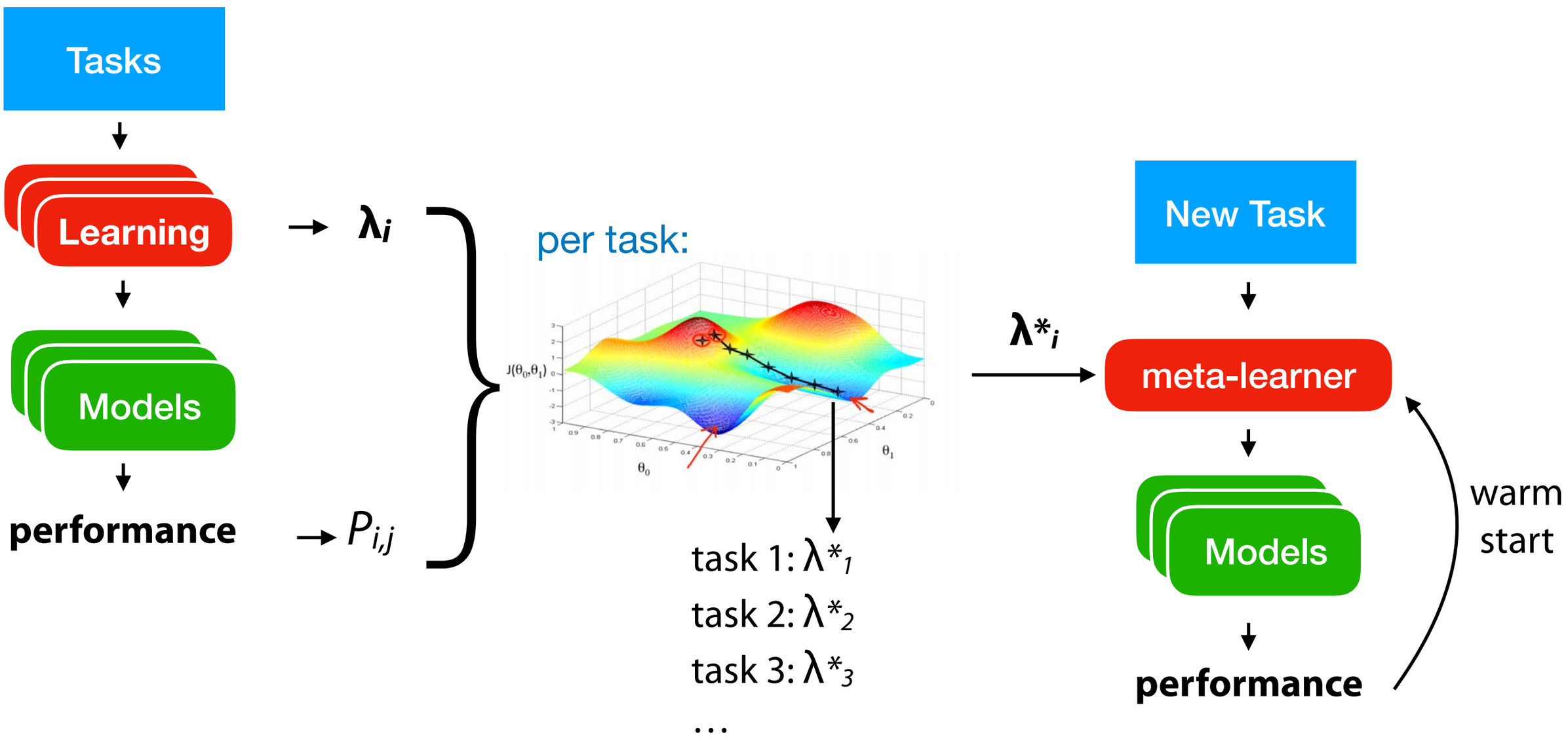
- Build a *global (multi-objective) ranking,* recommend the top-K
- Requires fixed selection of candidate configurations *(portfolio)*
- Can be used as a warm start for optimization techniques



Wistuba et al. 2015

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



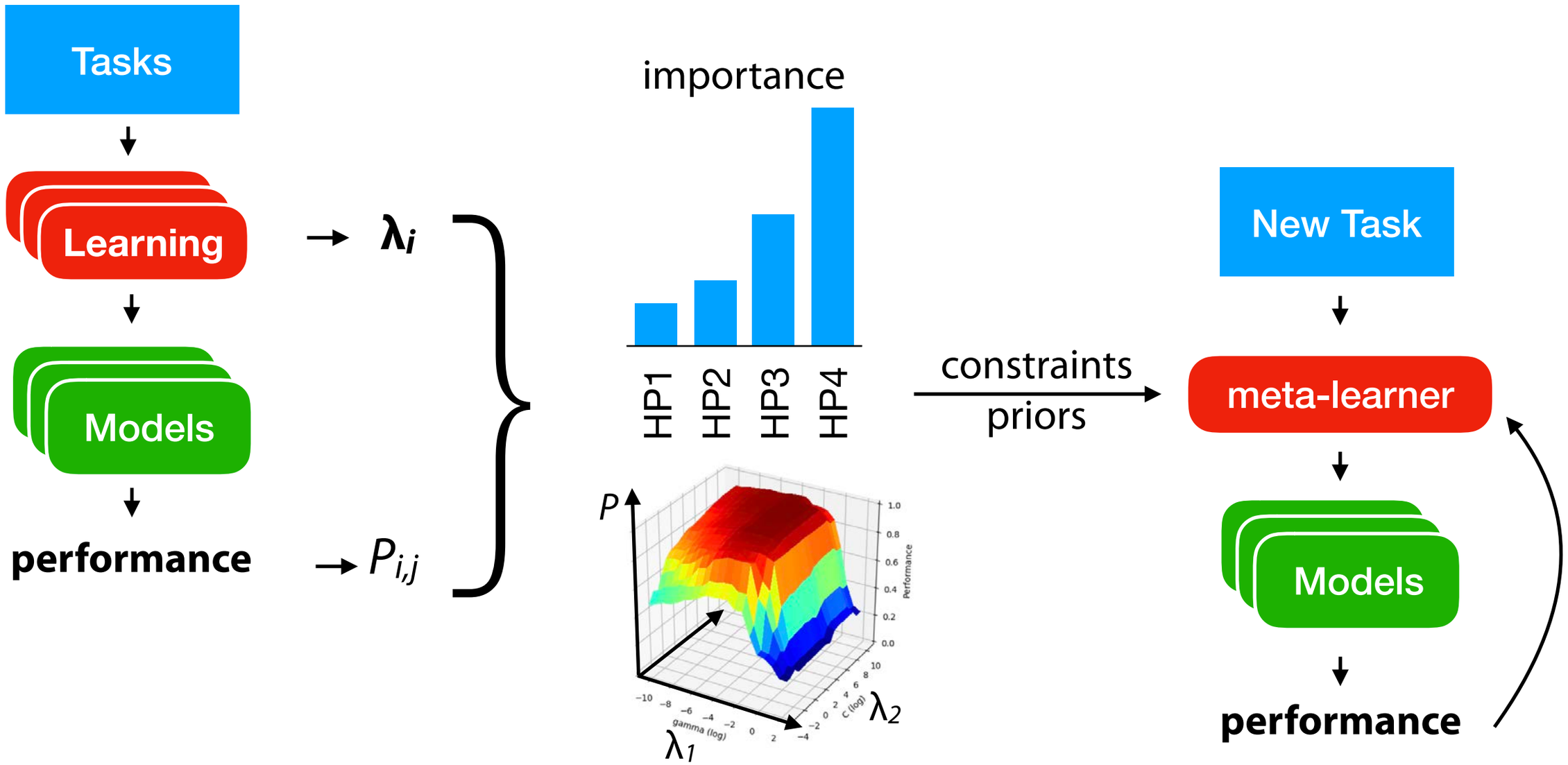
[1] van Rijn & Hutter 2018

[2] Probst et al. 2018

[3] Wistuba et al. 2015

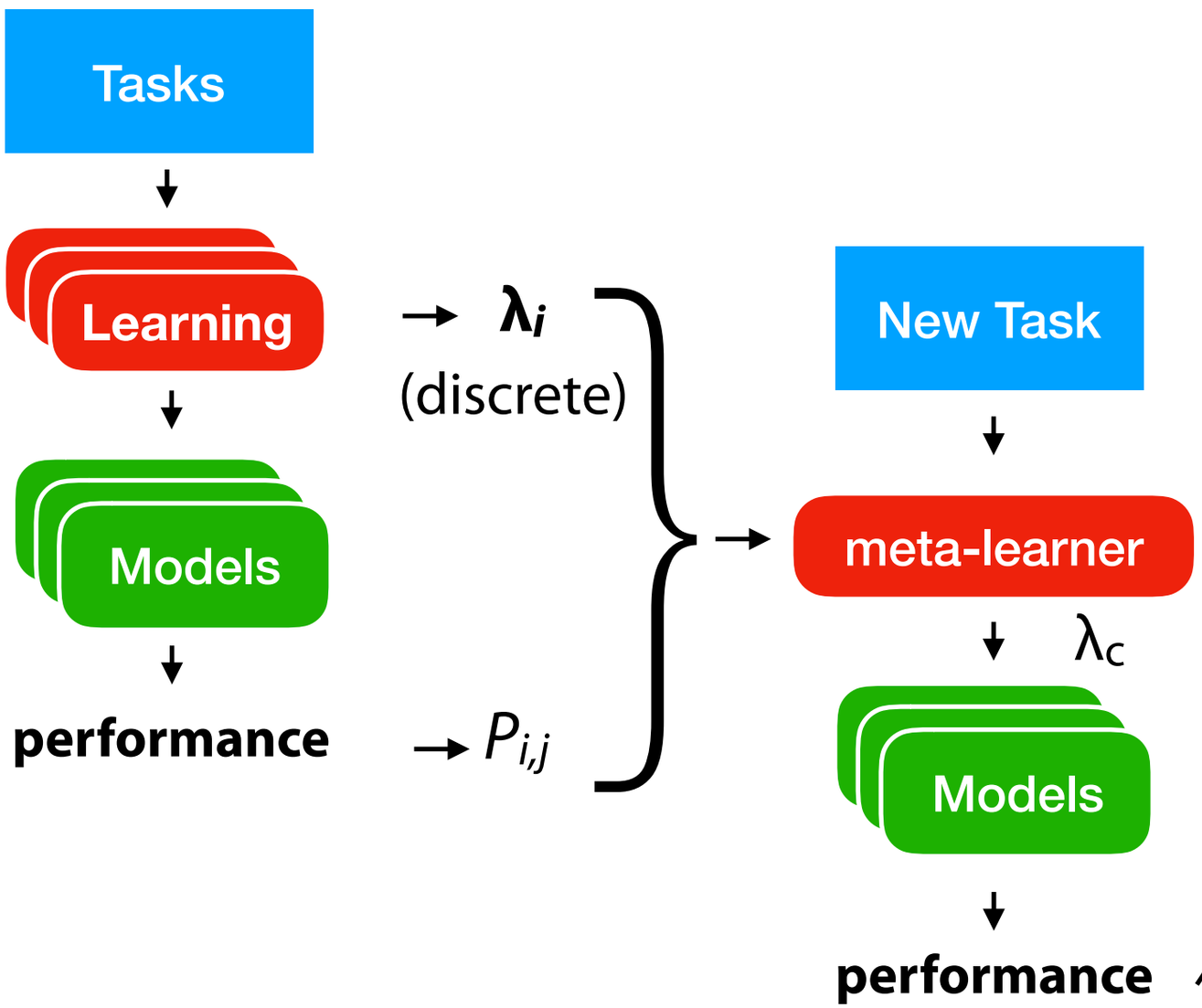
Configuration space design

- **Functional ANOVA**: select hyperparameters that cause variance in the evaluations[1]
- **Tunability**: improvement from tuning a hyperparameter vs. using a good default[2]
- **Search space pruning**: exclude regions yielding bad performance on similar tasks[3]



Active testing

- **Task are *similar*** if observed *relative performance* of configurations is similar
- Tournament-style selection, warm-start with overall best configurations λ_{best}
- Next candidate λ_c : the one that beats current λ_{best} on similar tasks (from portfolio)



Relative landmark on λ_a,λ_b, task t_j:

$$RL_{a,b,j} = P_{a,j} - P_{b,j}$$

Update:

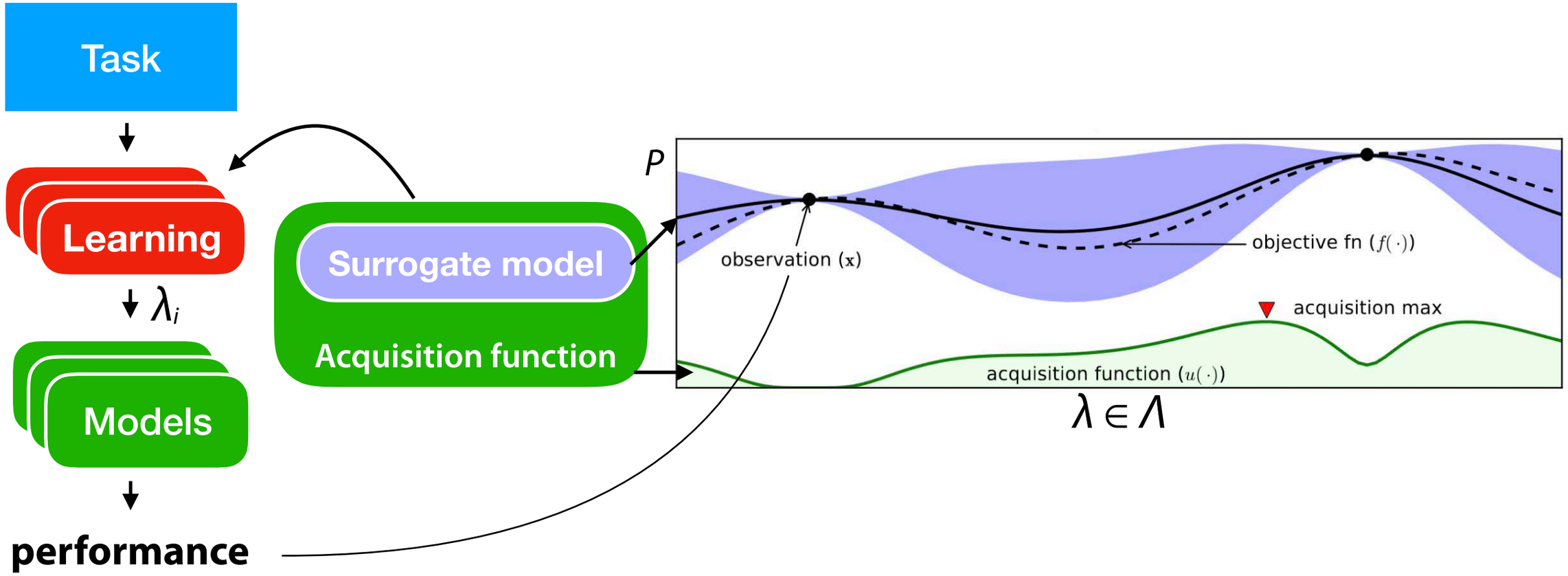
$$Sim(t_j,t_{new}) = Corr([RL_{a,b,j}],[RL_{a,b,new}])$$

Select $\lambda_c >_{RL} \lambda_{best}$ on similar tasks

Rasmussen 2014

Bayesian optimization (refresh)

- Learns how to learn within a single task (short-term memory)
- Surrogate model: *probabilistic* regression model of configuration performance
- **Can we transfer what we learned to *new* tasks (long term memory)?**

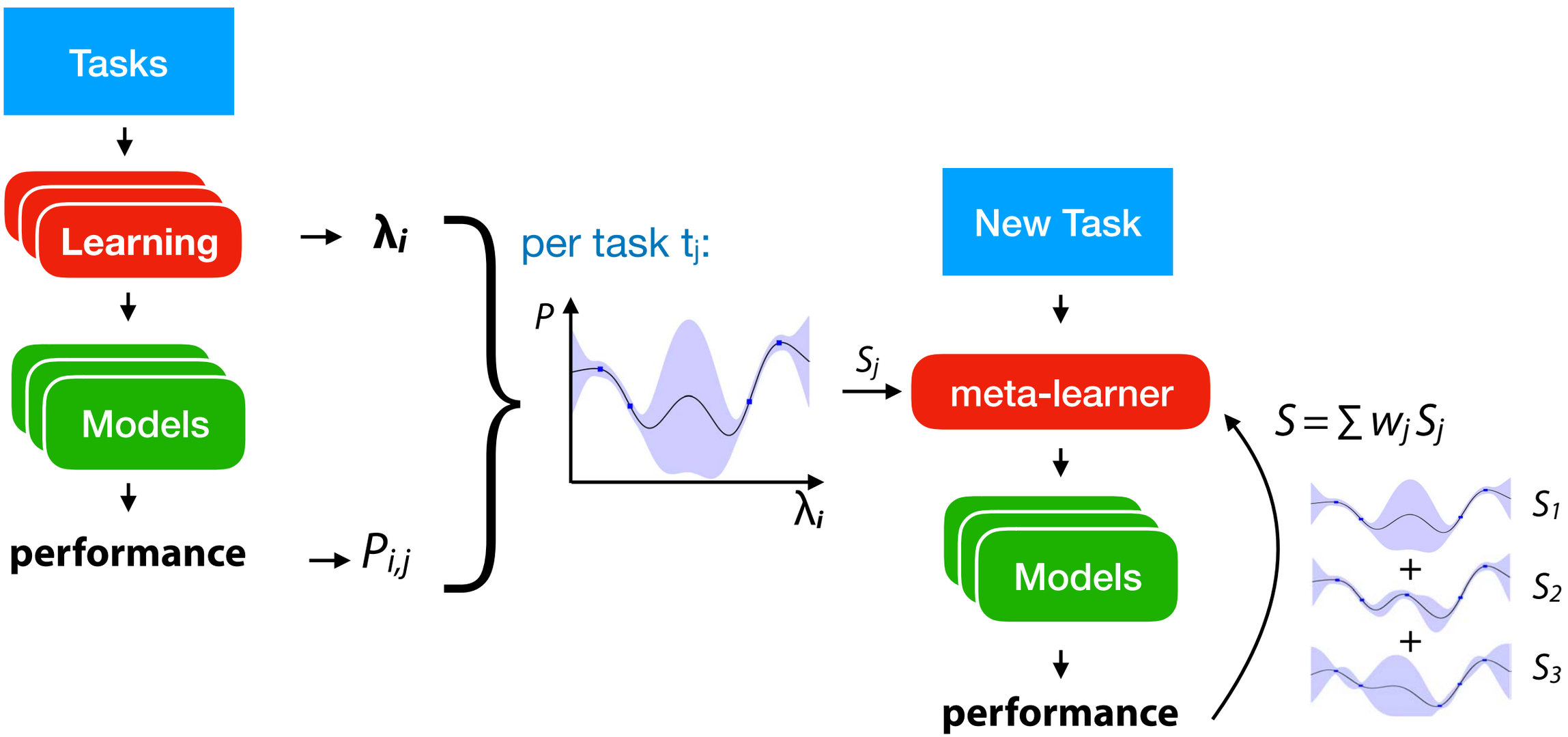


[1] Wistuba et al. 2018

[2] Feurer et al. 2018

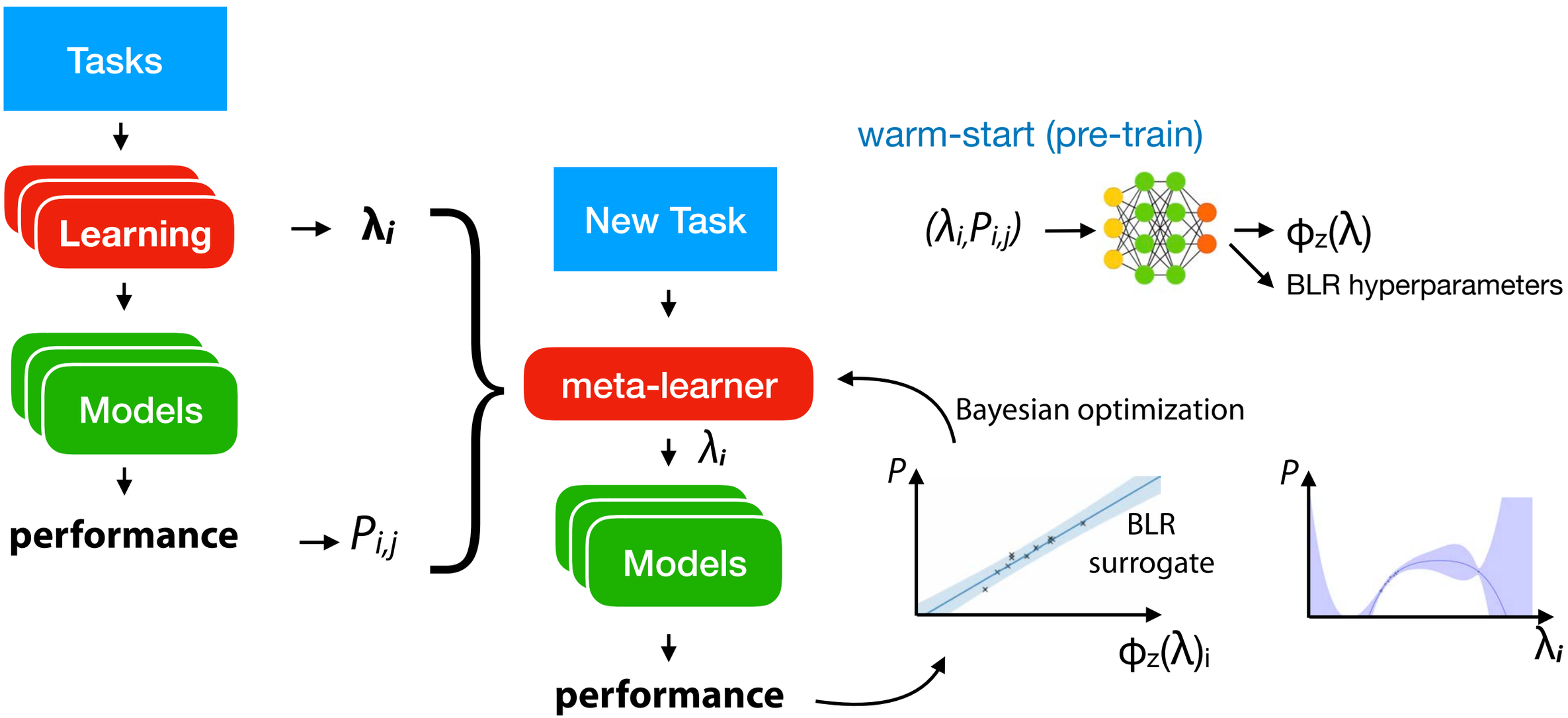
Surrogate model transfer

- If task j is *similar* to the new task, its surrogate model S_j will do well
- Sum up all S_j predictions, weighted by task similarity (relative landmarks)[1]
- Build combined Gaussian process, weighted by current performance on new task[2]



Warm-started multi-task learning

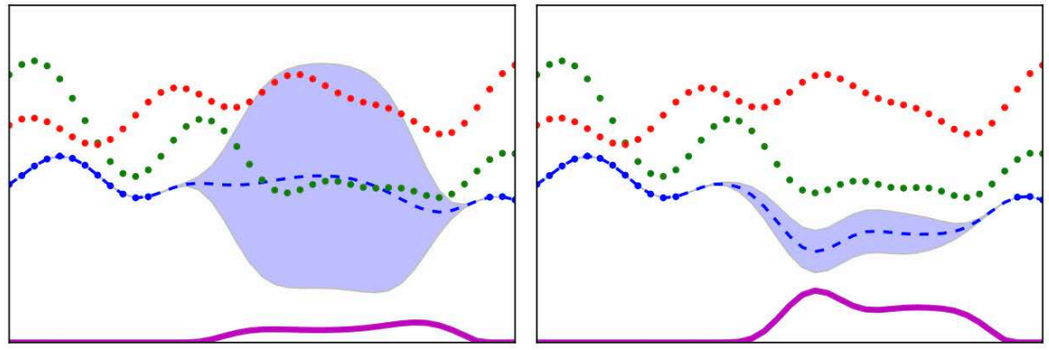
- Bayesian linear regression (BLR) surrogate model on every task
- Learn a suitable basis expansion $\phi_z(\lambda)$, joint representation for all tasks
- Scales linearly in # observations, transfers info on configuration space



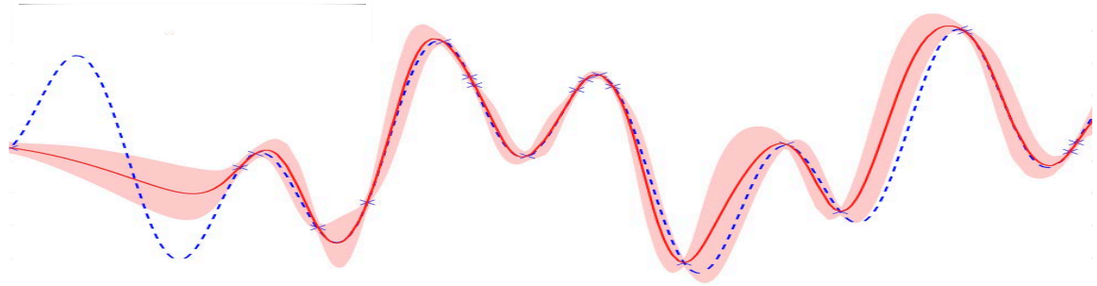
[1] Swersky et al. 2013
[2] Springenberg et al. 2016
[3] Golovin et al. 2017

Multi-task Bayesian optimization

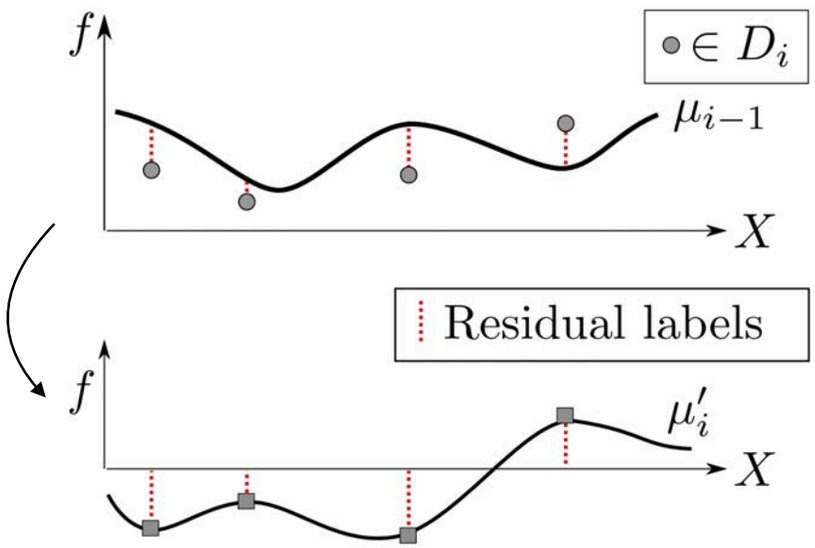
- **Multi-task Gaussian processes:** train surrogate model on t tasks simultaneously[1]
  - If tasks are similar: transfers useful info
  - Not very scalable



- **Bayesian Neural Networks** as surrogate model[2]
  - Multi-task, more scalable



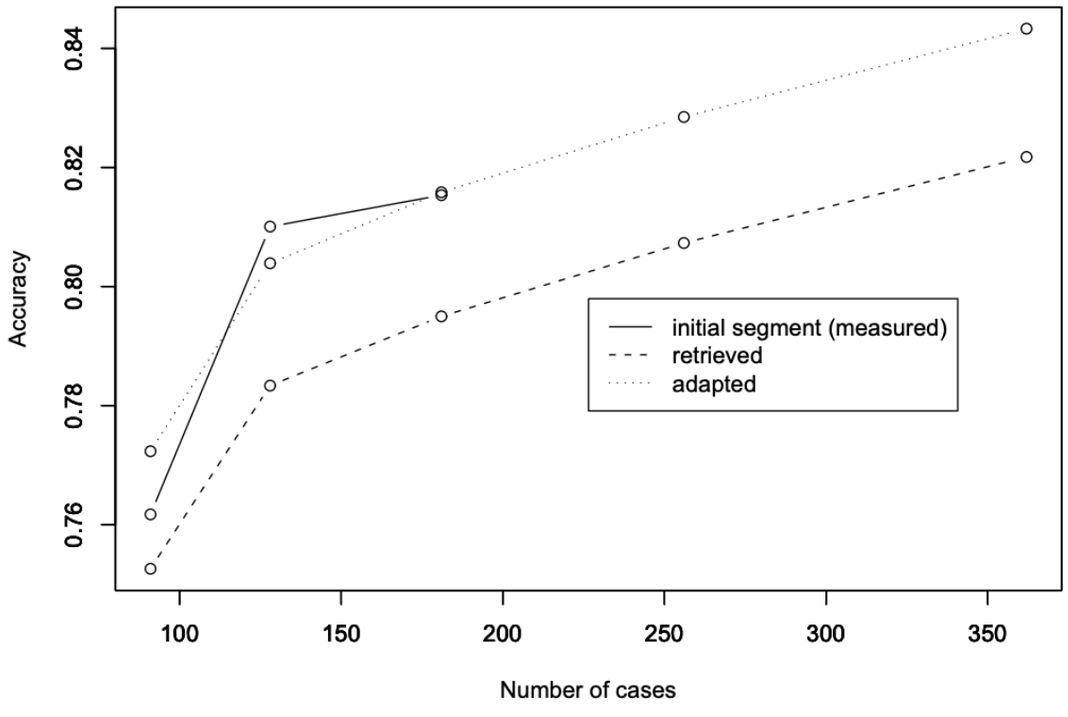
- **Stacking** Gaussian Process regressors (Google Vizier)[3]
  - Sequential tasks, each similar to the previous one
  - Transfers a prior based on residuals of previous GP



[1] *Ramachandran et al. 2018*
[2] *Leite et al. 2005*
[3] *van Rijn et al. 2015*

Other techniques

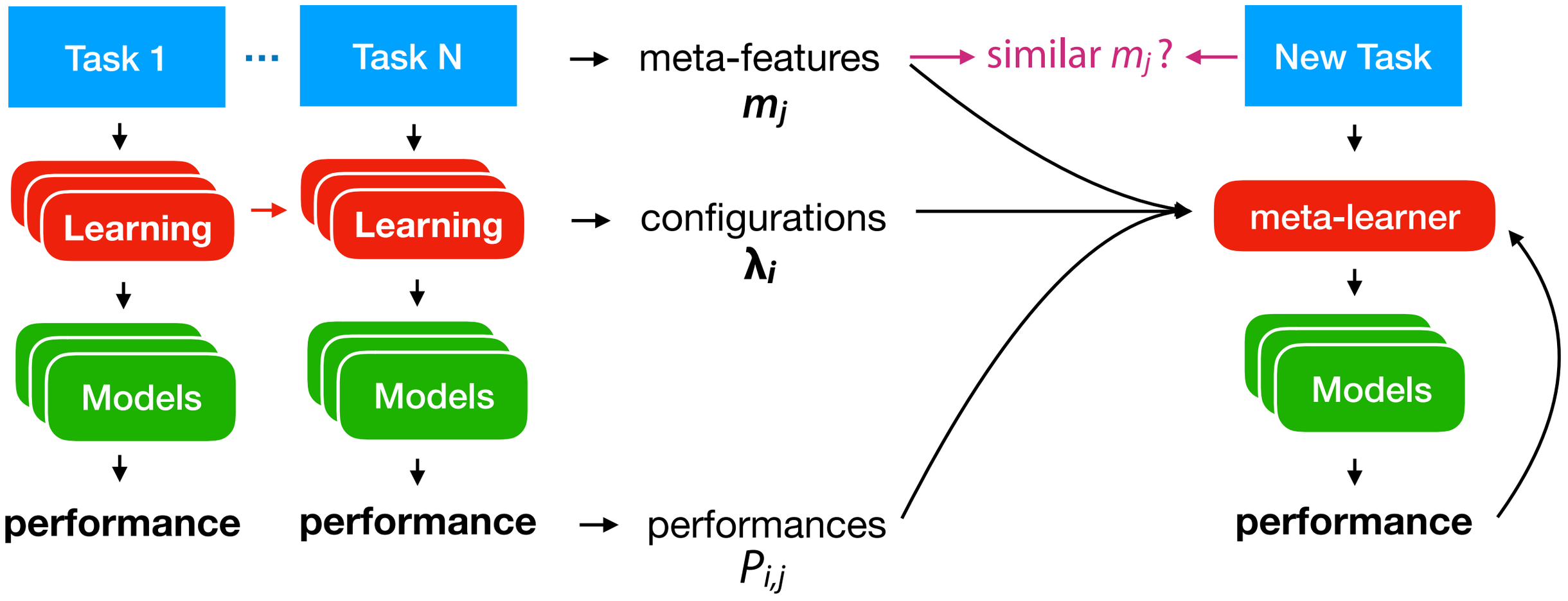
- Transfer learning with multi-armed bandits[1]
  - View every task as an arm, learn to `pull` observations from the most similar tasks
  - Reward: accuracy of configurations recommended based on these observations
- Transfer learning curves[2,3]
  - Learn a partial learning curve on a new task, find best matching earlier curves
  - Predict the most promising configurations based on earlier curves



2. Reason about model performance across tasks

Meta-features: measurable properties of the tasks

(number of instances and features, class imbalance, feature skewness,…)

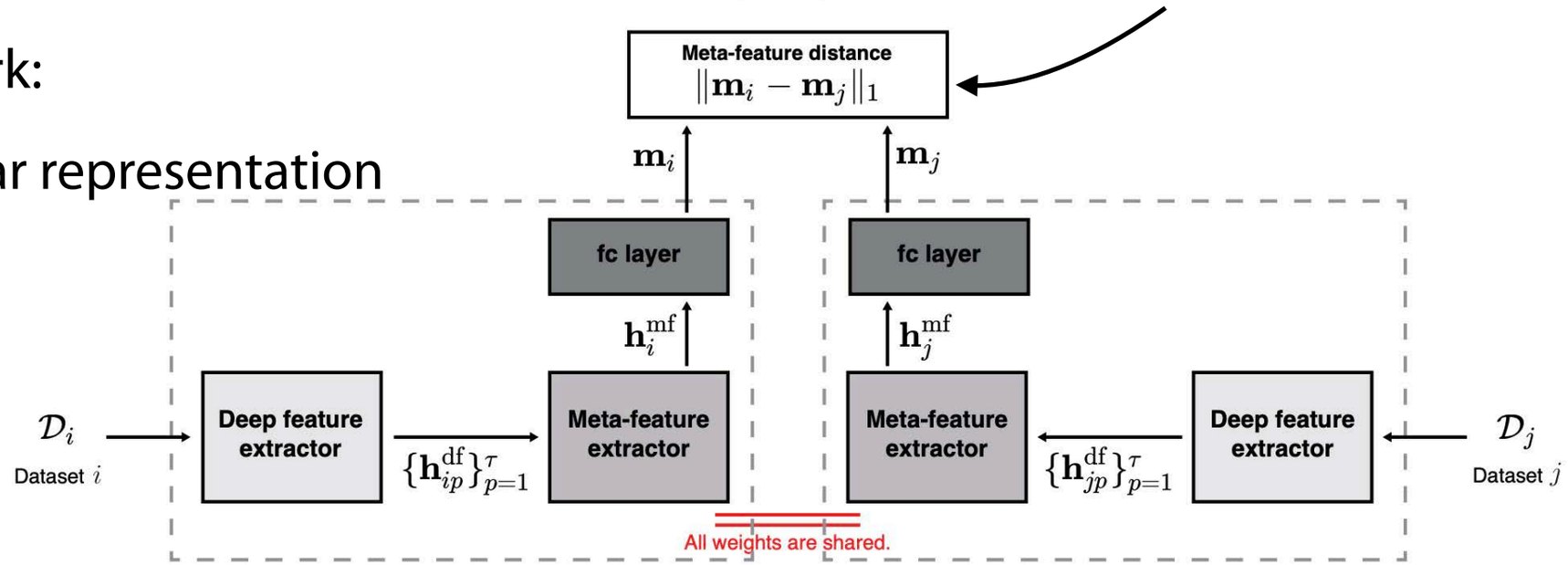


[1] Vanschoren 2018

[2] Kim et al. 2017

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

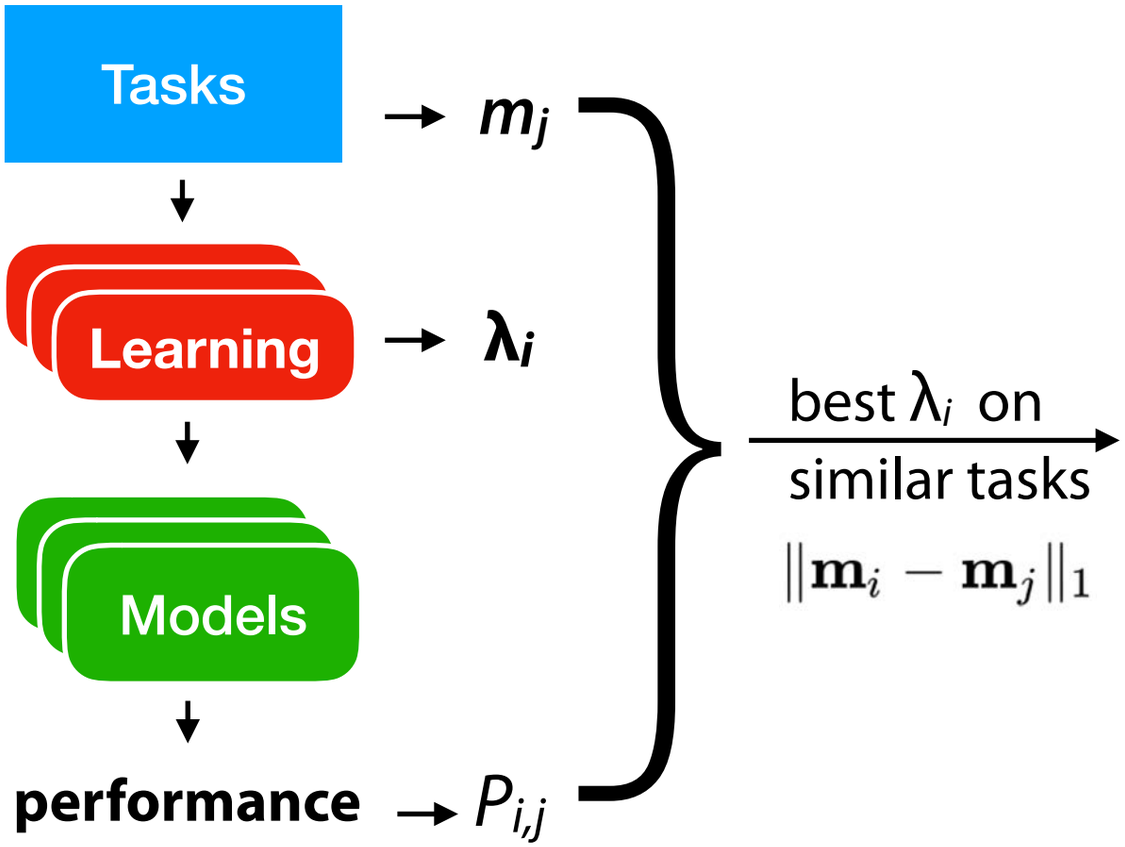


[1] Gomes et al. 2012, Reif et al. 2012

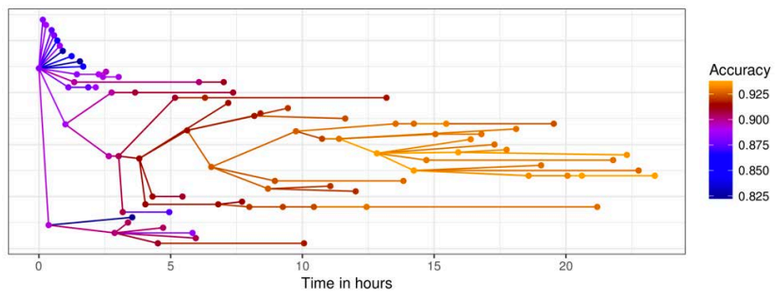
[2] Feurer et al. 2015

Warm-starting from similar tasks

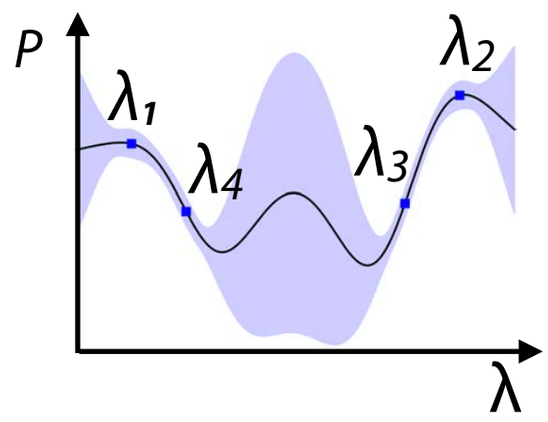
- Find k most similar tasks, warm-start search with best θ_i
  - Genetic hyperparameter search [1]
  - Auto-sklearn: Bayesian optimization (SMAC) [2]
    - Scales well to high-dimensional configuration spaces



Genetic optimization

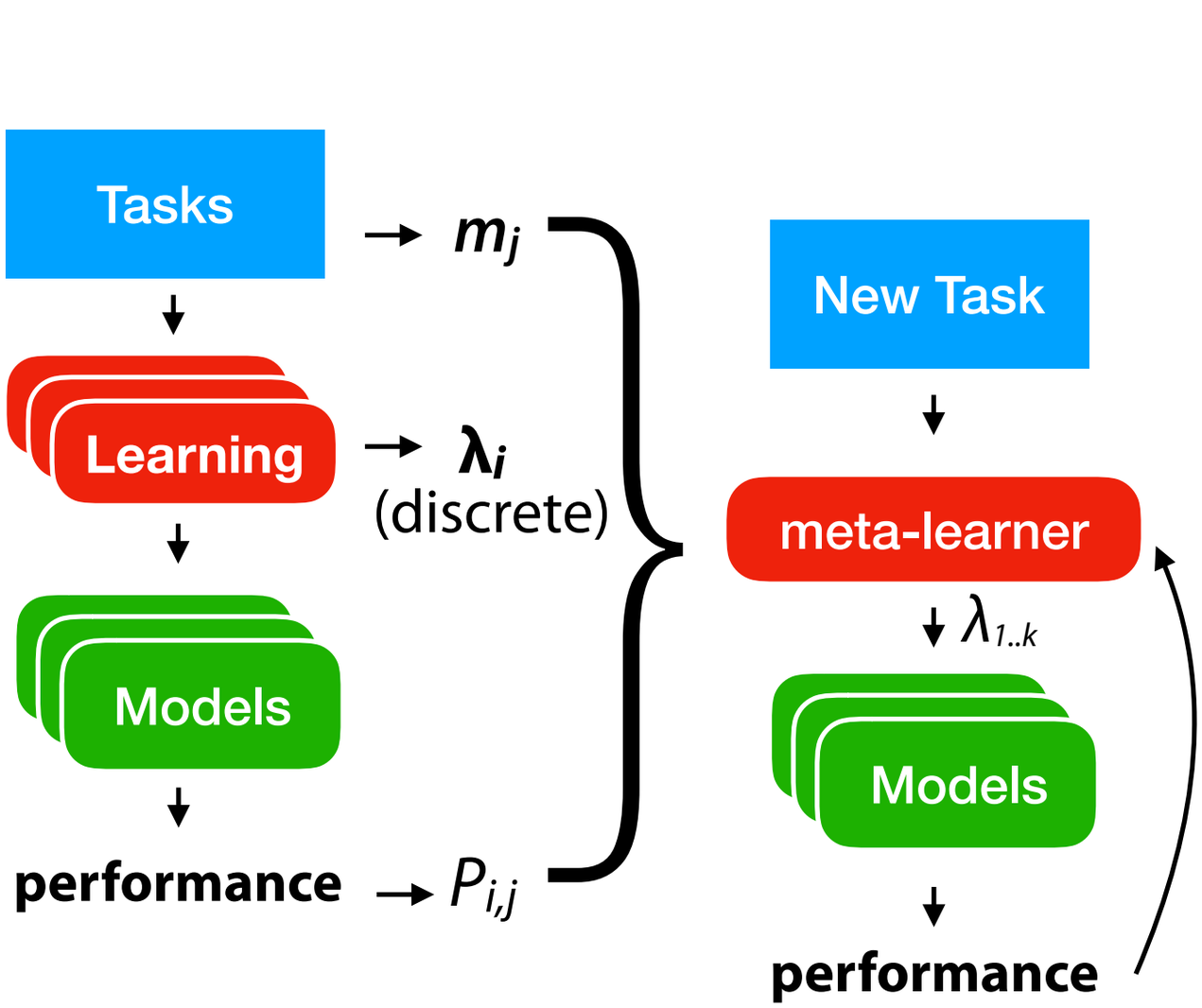


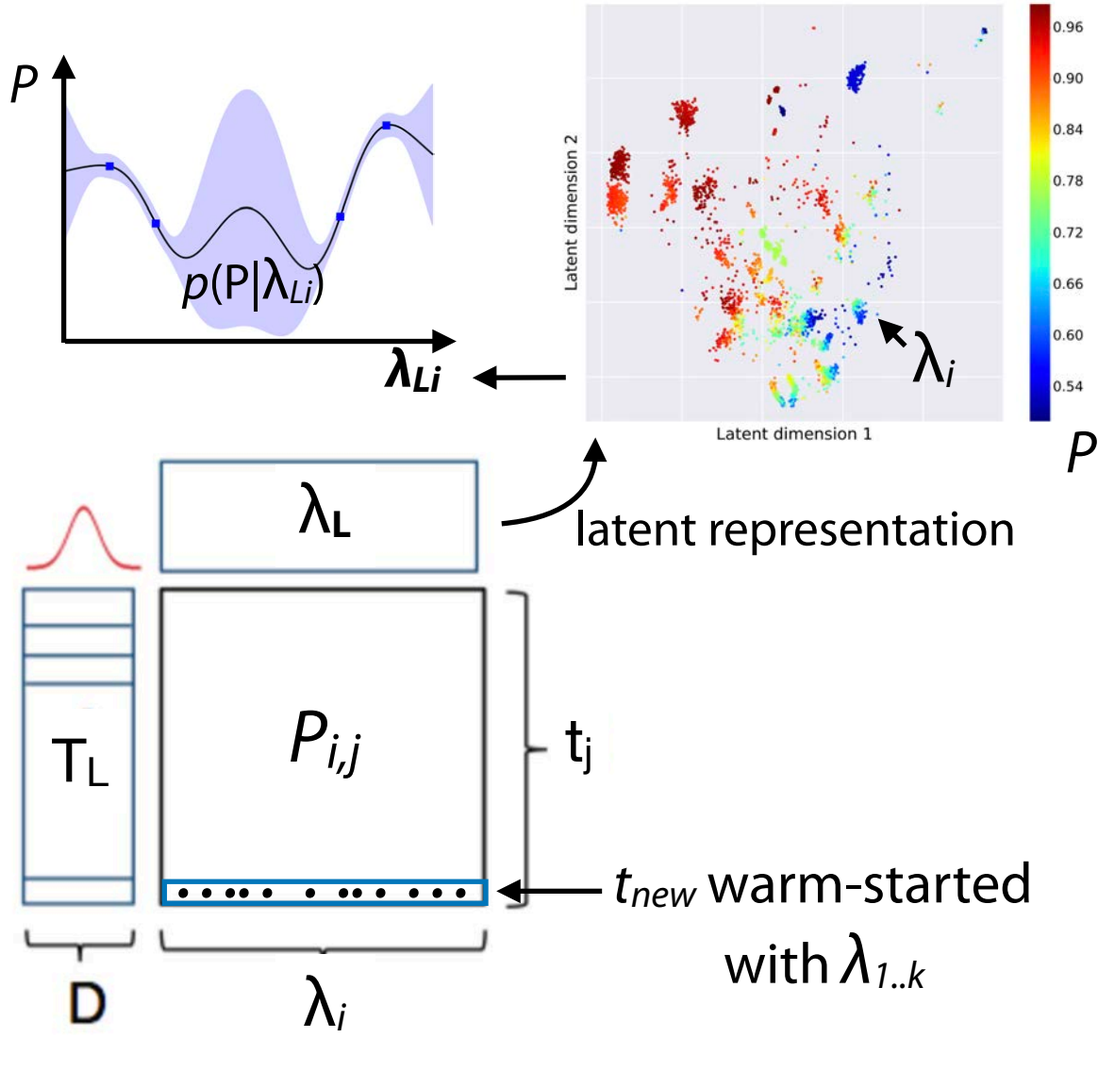
Bayesian optimization



Warm-starting from similar tasks

- Collaborative filtering: configurations λ_i are `rated' by tasks t_j
  - Probabilistic matrix factorization
    - Learns a latent representation for tasks and configurations
    - Returns probabilistic predictions for Bayesian optimization
    - Use meta-features to warm-start on new task

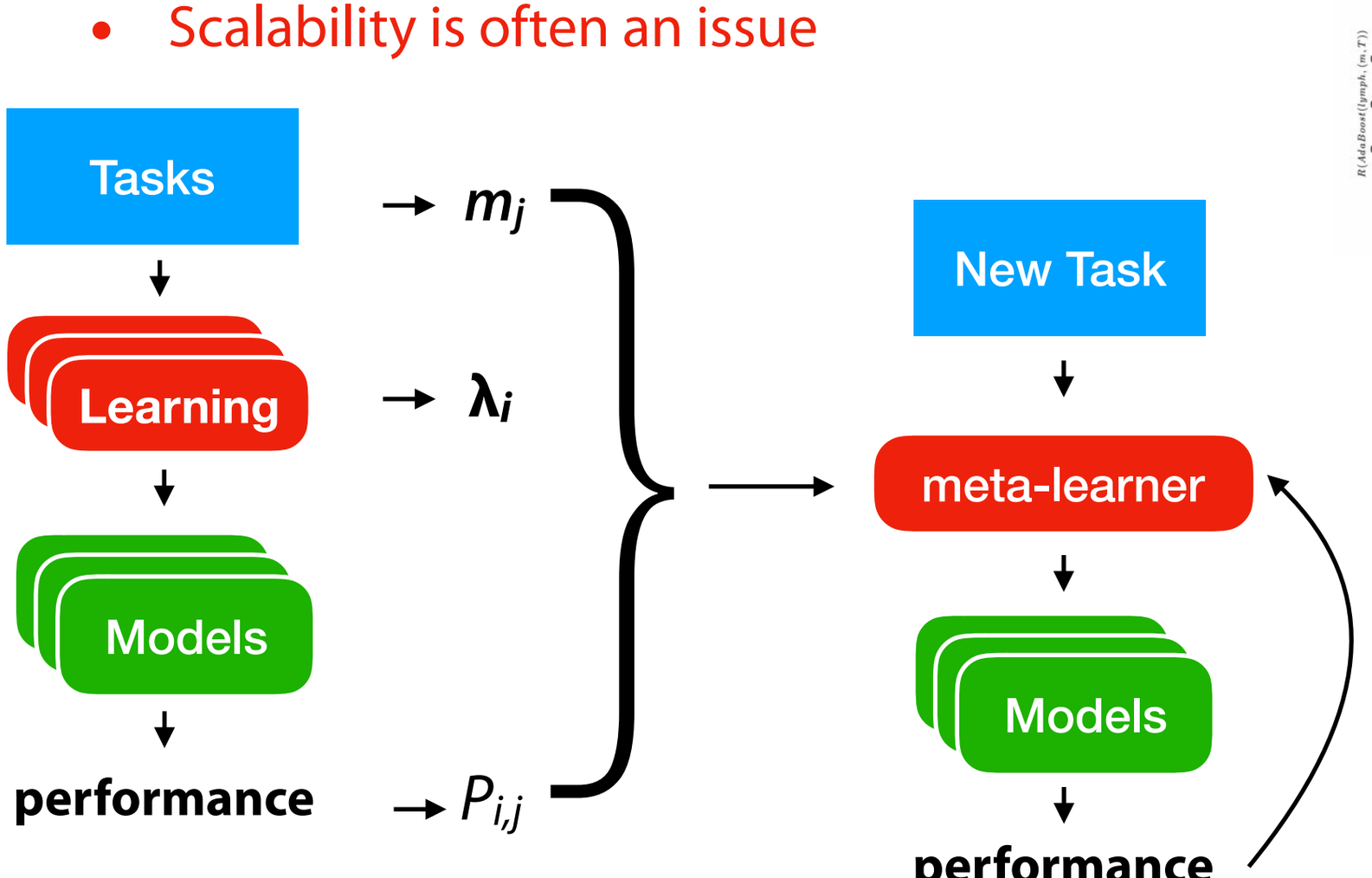


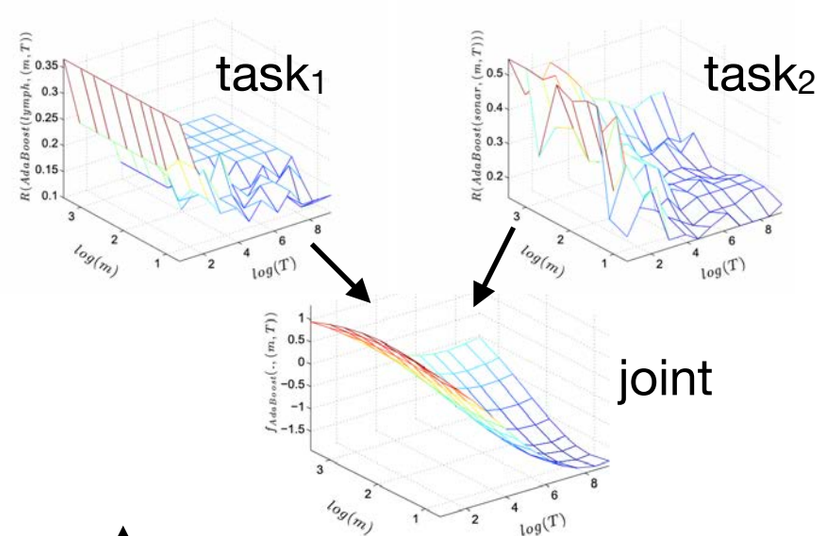


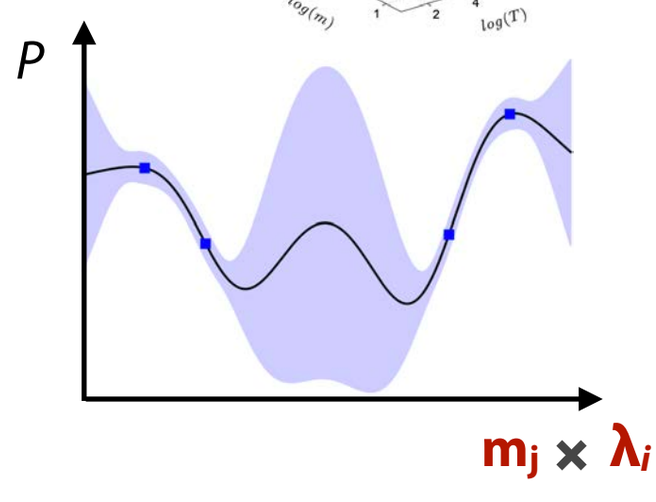
[1] Bardenet et al. 2013
[2] Schilling et al. 2015
[3] Yogatama et al. 2014

Global surrogate models

- Train a task-independent surrogate model with meta-features in inputs
  - SCOT: Predict *ranking* of λ_i with surrogate ranking model + m_j. [1]
  - Predict $P_{i,j}$ with multilayer Perceptron surrogates + m_j. [2]
  - Build joint GP surrogate model on most similar ($\|\mathbf{m}_i - \mathbf{m}_j\|_2$) tasks. [3]
  - Scalability is often an issue







[1] Brazdil et al. 2009, Lemke et al. 2015

[2] Sun and Pfahringer 2013, Pinto et al. 2017

[3] Sanders and C. Giraud-Carrier 2017

[4] Yang et al. 2018

Meta-models

- *Learn* direct mapping between meta-features and $P_{i,j}$
  - Zero-shot meta-models: predict best λ_i given meta-features [1]

  - Ranking models: return ranking $\lambda_{1..k}$ [2]

  - Predict which algorithms / configurations to consider / tune [3]

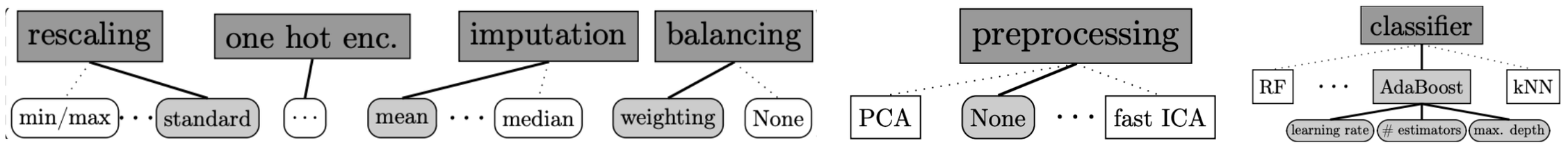
  - Predict performance / runtime for given θ_i and task [4]

- Can be integrated in larger AutoML systems: warm start, guide search,…

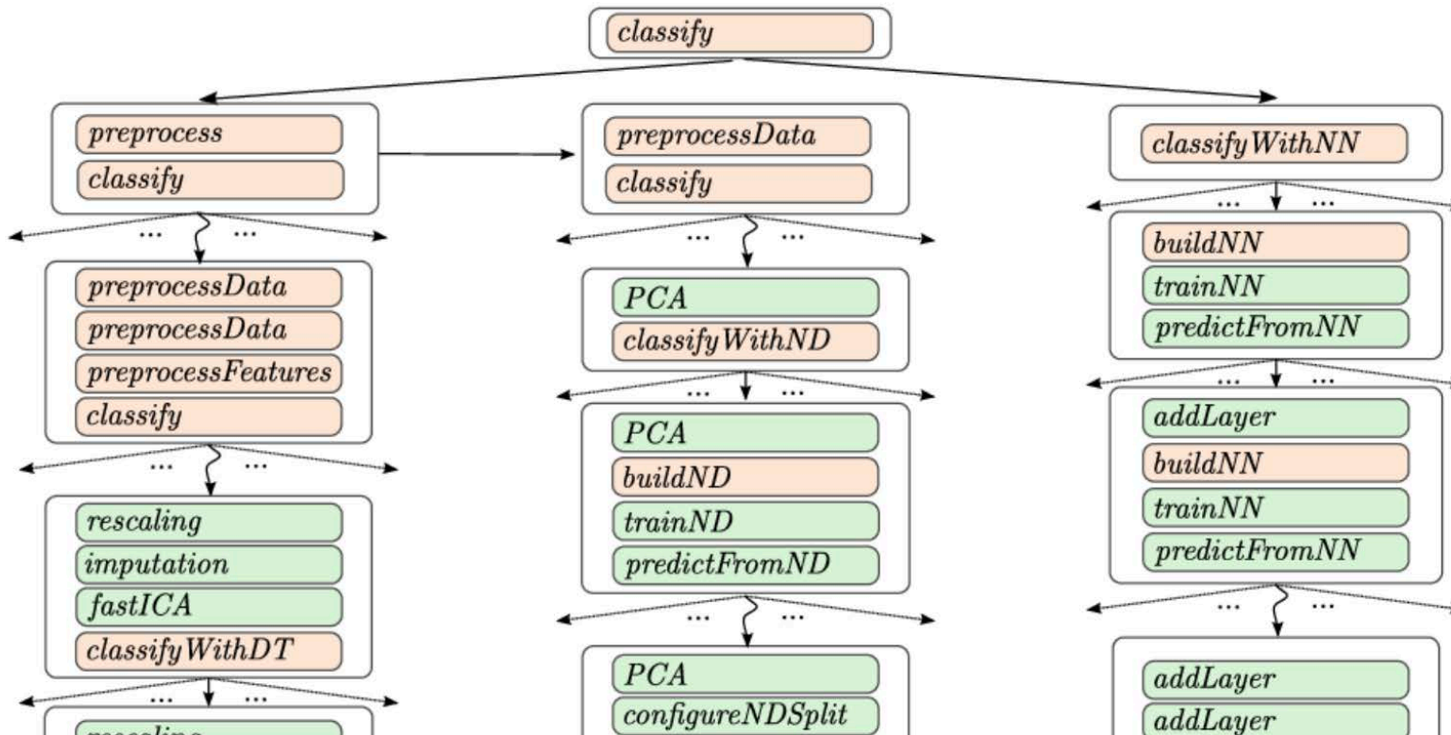
[1] Fusi et al. 2017
[2] Feurer et al. 2015
[3] Mohr et al. 2018

Learning Pipelines

- **Compositionality**: the learning process can be broken down into smaller tasks
  - Easier to learn, more transferable, more robust
- Pipelines are one way of doing this, but how to control the search space?
  - Select a fixed set of possible pipelines. Often works well (less overfitting) [1]
  - Impose a fixed structure on the pipeline [2]



  - (Hierarchical) Task Planning [3]
    - Break down into smaller tasks

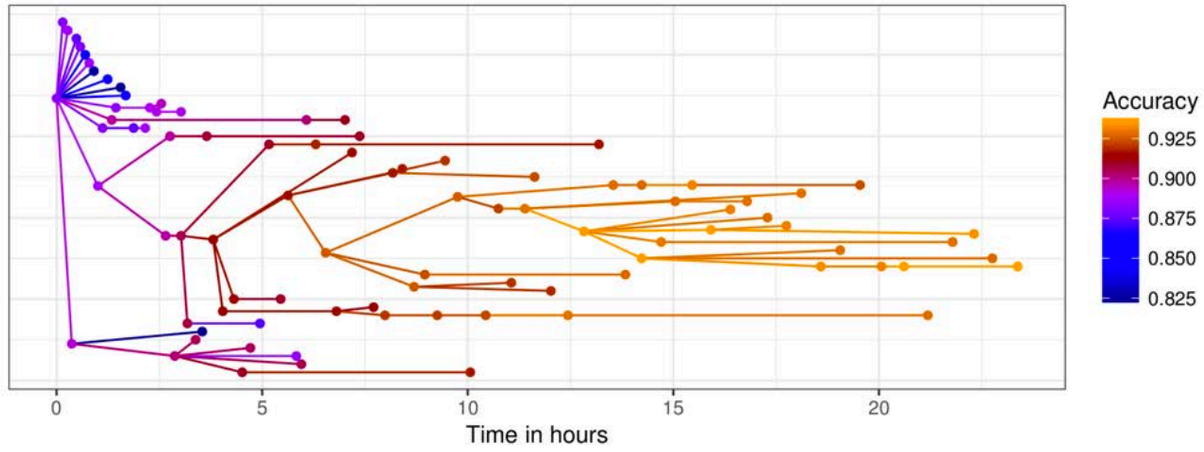


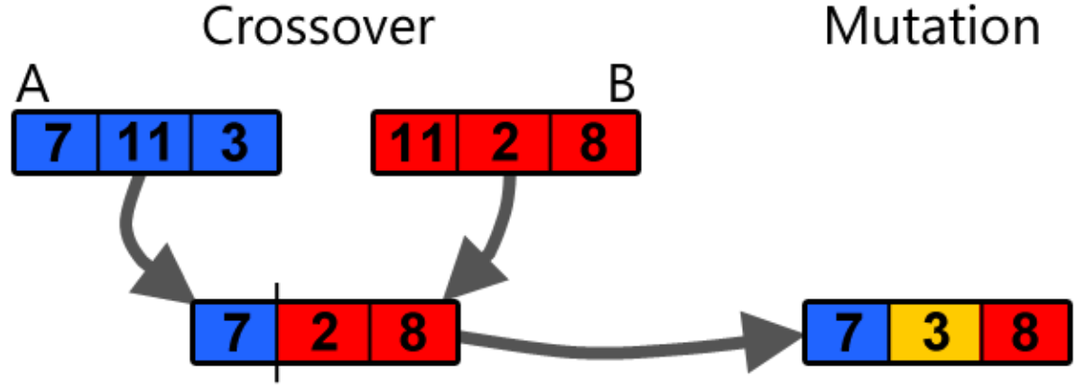
- Meta-learning:
  - Mostly warm-starting

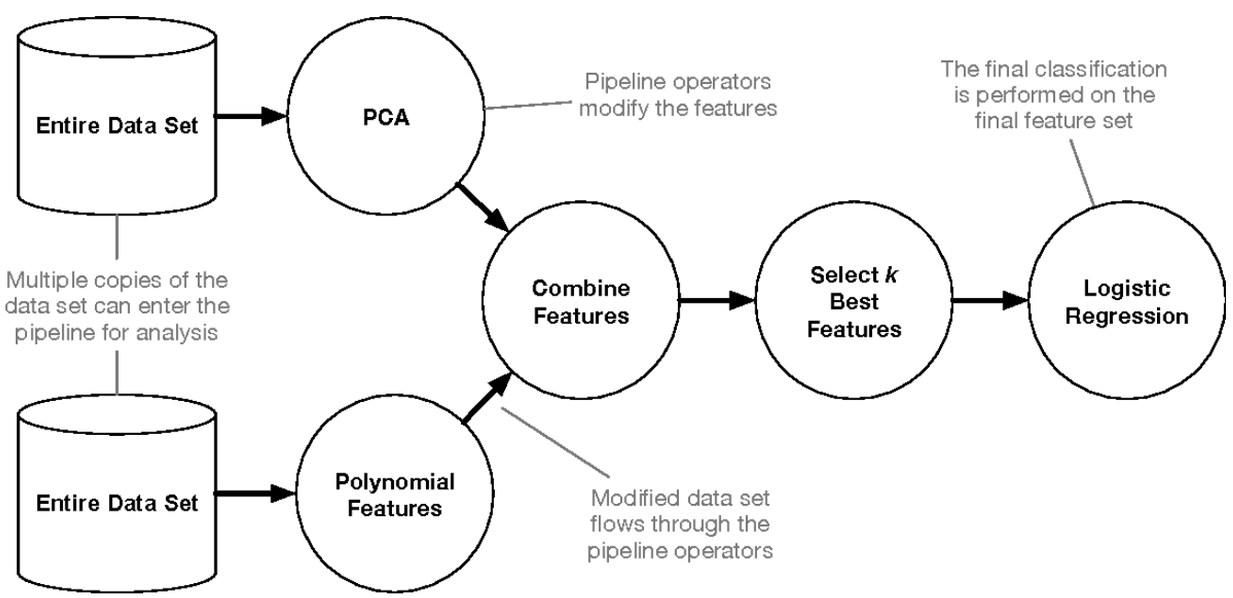
[1] Olson et al. 2017
[2] Gijsbers et al. 2018
[3] De Sa et al. 2017

Evolving pipelines

- Start from simple pipelines
- *Evolve* more complex ones if needed
- Reuse pipelines that do specific things
- Mechanisms:
  - Cross-over: reuse partial pipelines
  - Mutation: change structure, tuning
- Approaches:
  - TPOT: Tree-based pipelines[1]
  - GAMA: asynchronous evolution[2]
  - RECIPE: grammar-based[3]
- Meta-learning:
  - Largely unexplored
  - Warm-starting, meta-models

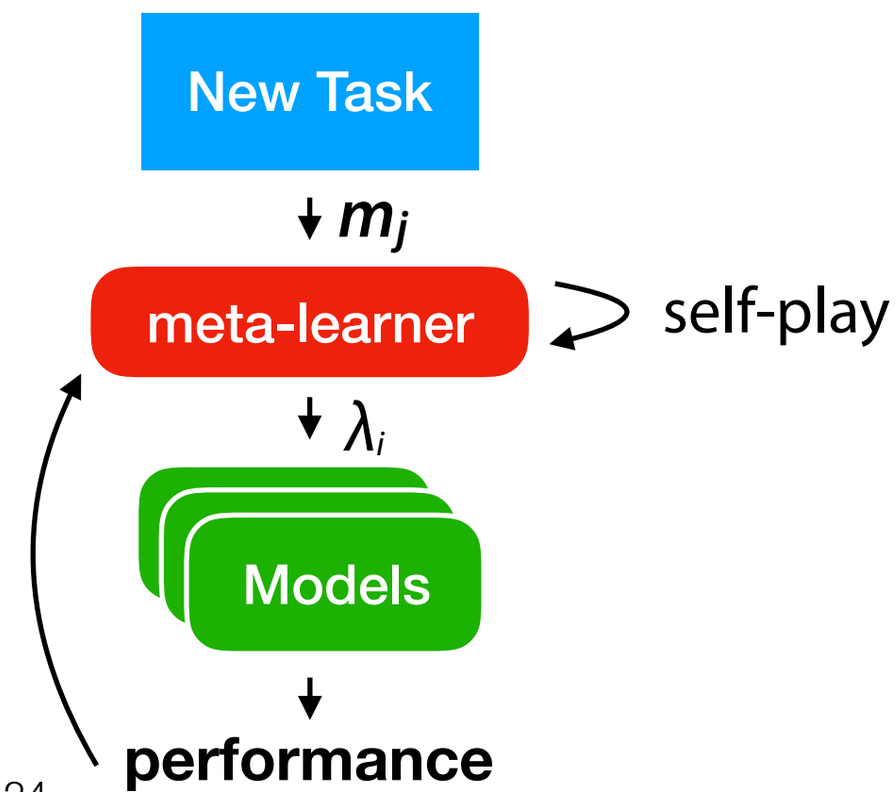


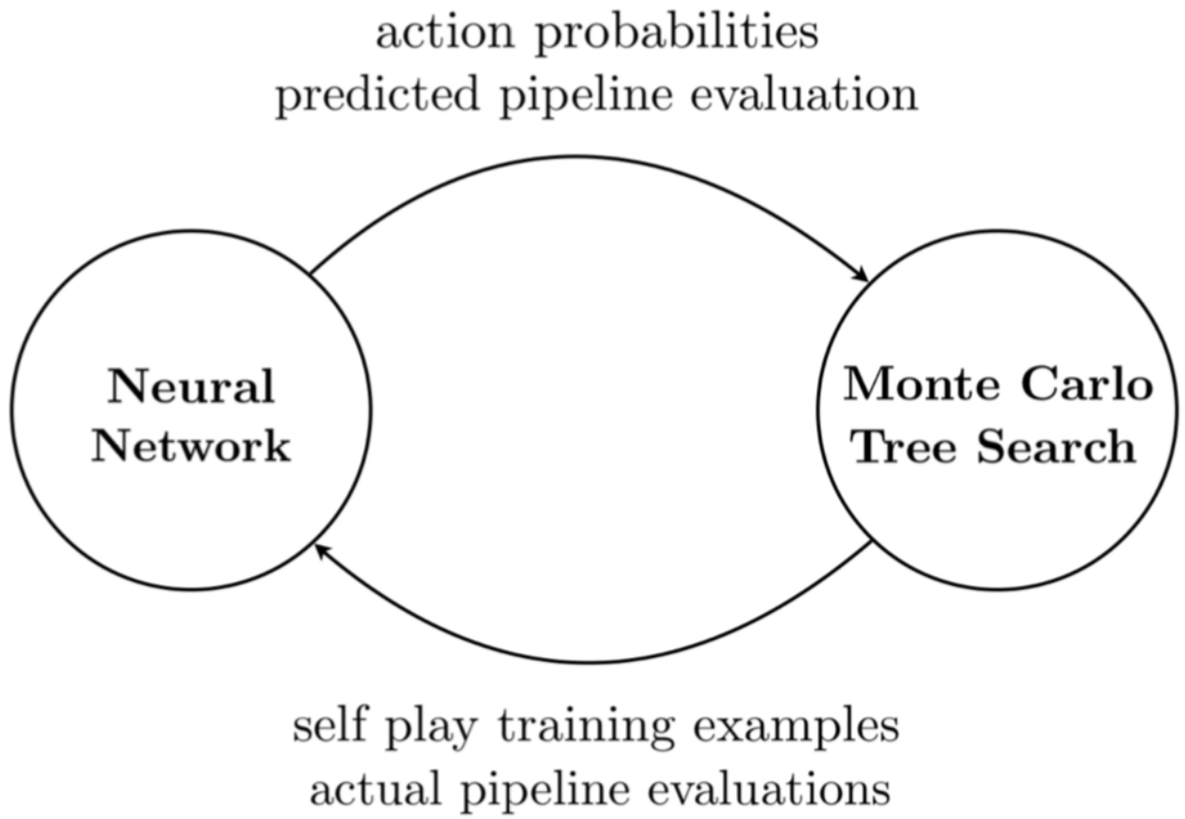




Learning to learn through self-play

- Build pipelines by selecting among actions
  - insert, delete, replace pipeline parts
- Neural network (LSTM) receives task meta-features, pipelines and evaluations
  - Predict pipeline performance and action probabilities
- Monte Carlo Tree Search builds pipelines based on probabilities
  - Runs multiple simulations to search for a better pipeline

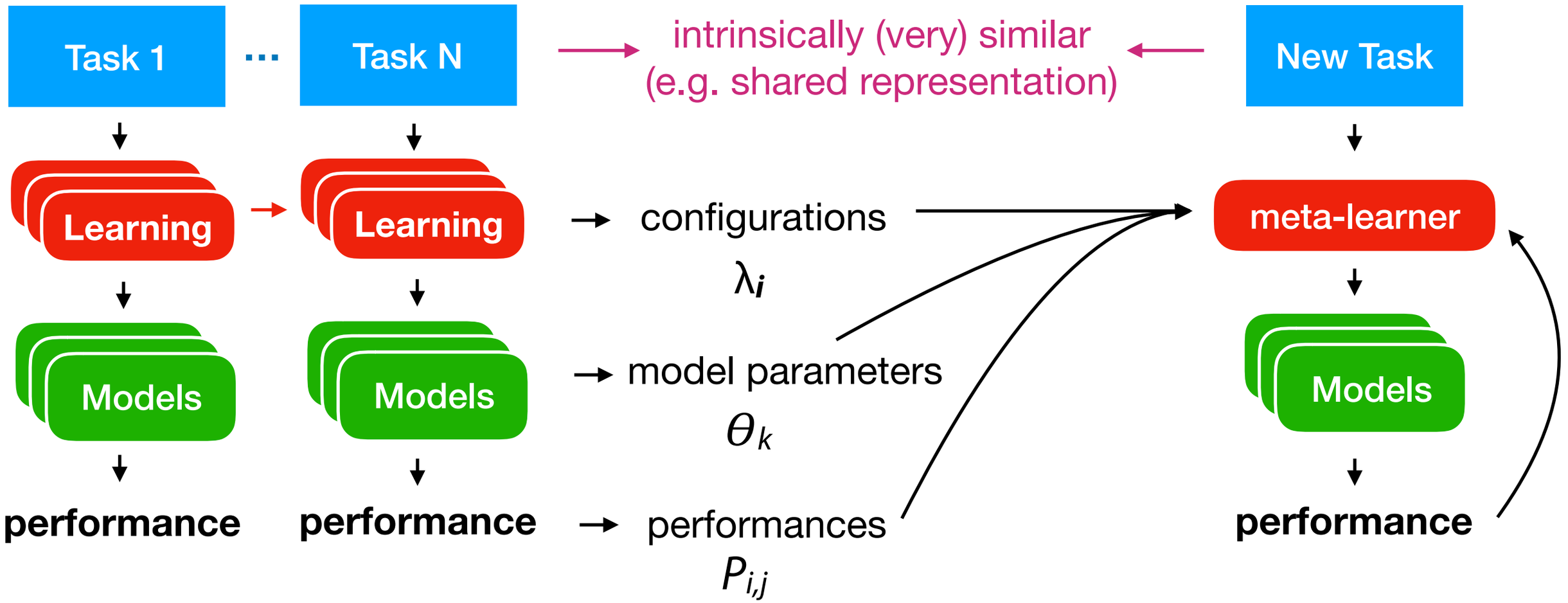




3. Learning from trained models

Models trained on similar tasks

(model parameters, features,…)



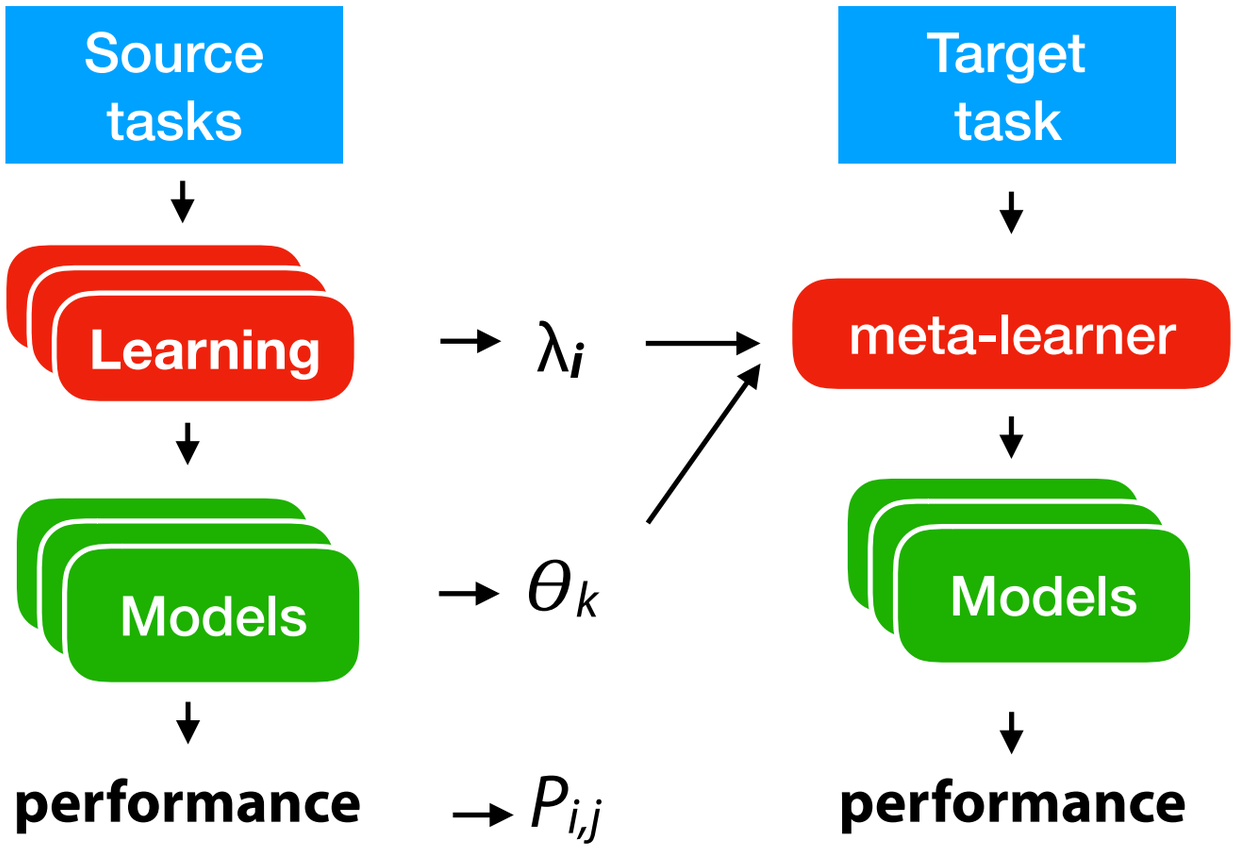
[1] Thrun and Pratt 1998

[2] Niculescu-Mizil and Caruana 2005

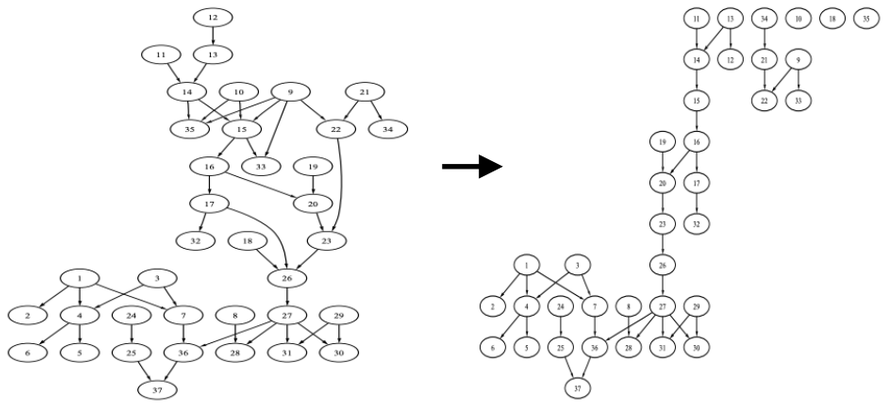
[3] Taylor and Stone 2009

Transfer Learning

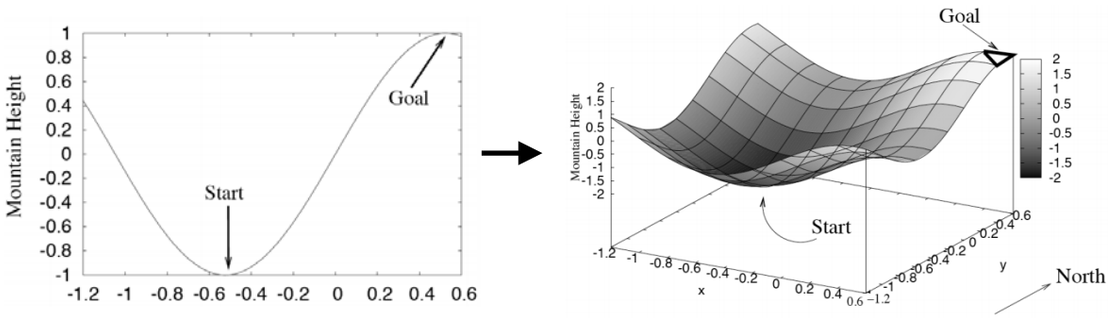
- Select source tasks, transfer trained models to similar target task [1]
- Use as starting point for tuning, or *freeze* certain aspects (e.g. structure)
  - Bayesian networks: start structure search from prior model [2]
  - Reinforcement learning: start policy search from prior policy [3]



Bayesian Network transfer



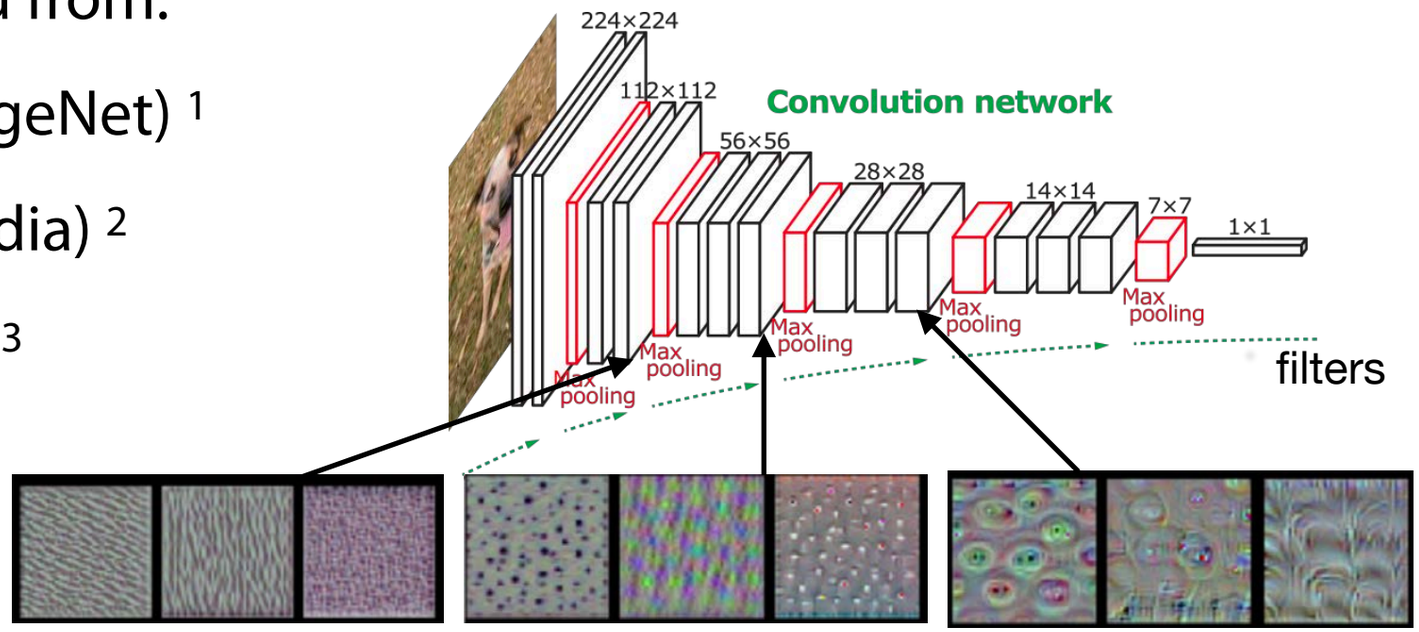
Reinforcement learning: 2D to 3D mountain car

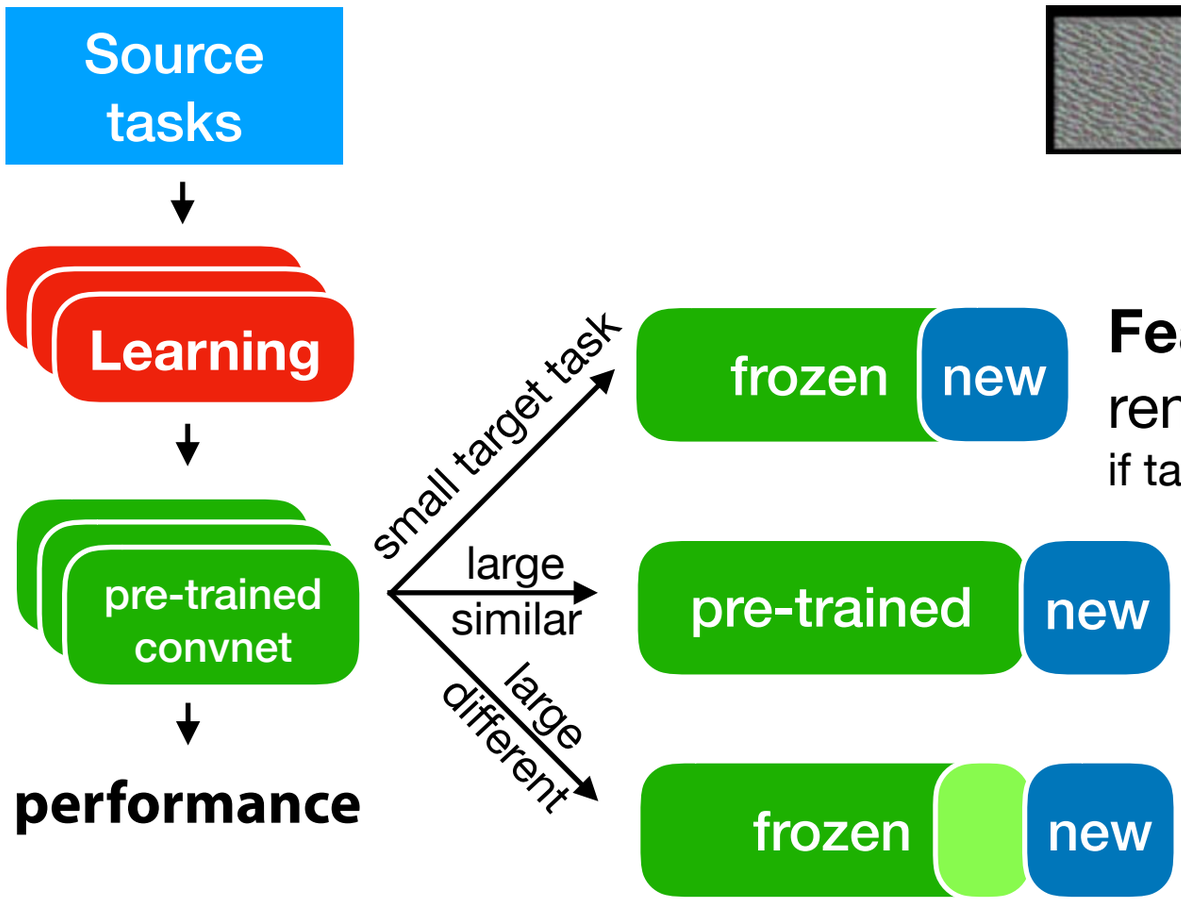


[1] Razavian et al. 2014
[2] Mikolov et al. 2013
[3] Yosinski et al. 2014

Transfer features, initializations

- For neural networks, both structure and weights can be transferred
- Features and initializations learned from:
  - Large image datasets (e.g. ImageNet) [1]
  - Large text corpora (e.g. Wikipedia) [2]
- Fails if tasks are *not similar enough* [3]





Feature extraction:
remove last layers, use output as features
if task is quite different, remove more layers

End-to-end tuning:
train from initialized weights

Fine-tuning:
unfreeze last layers, tune on new task

[1] Bengio et al. 1995

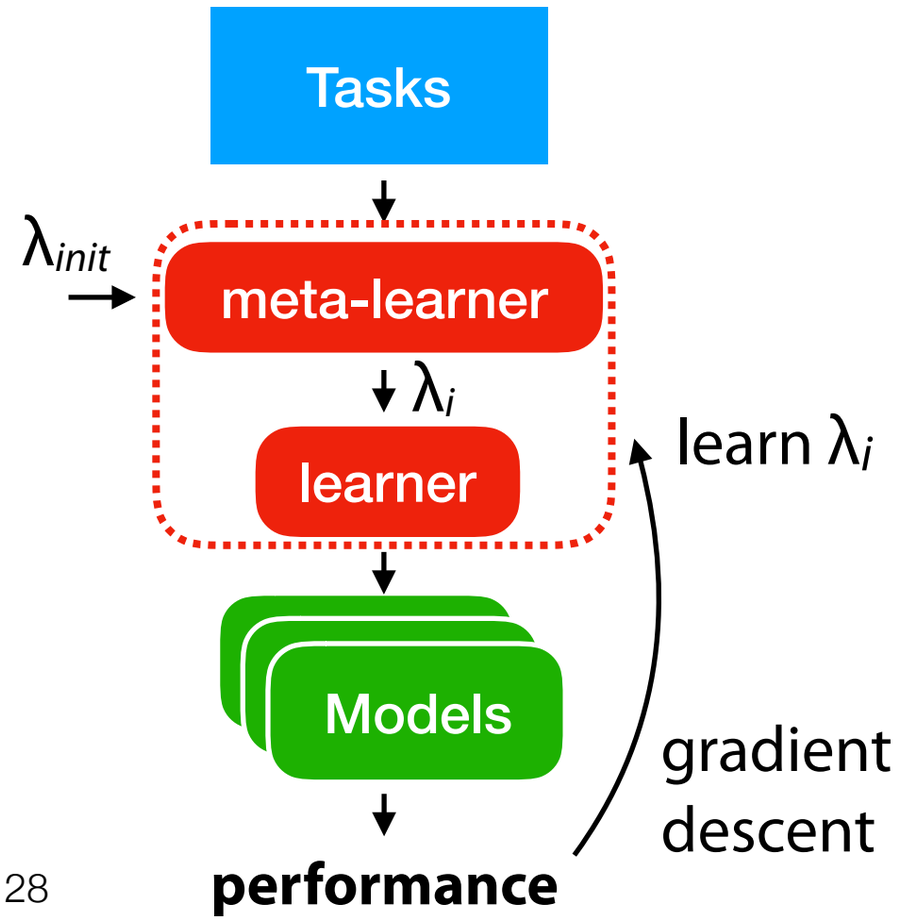
[2] Runarsson and Jonsson 2000

Learning to learn by gradient descent

- Our brains *probably* don't do backprop, replace it with:

$$\Delta w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}}$$

  - Simple *parametric* (bio-inspired) rule to update weights [1]
  - Single-layer neural network to learn weight updates [2]
- Learn parameters across tasks, by gradient descent (meta-gradient)



Bengio et al.

$$\Delta \theta_i = \epsilon \, y_{pre(i)} \, k$$

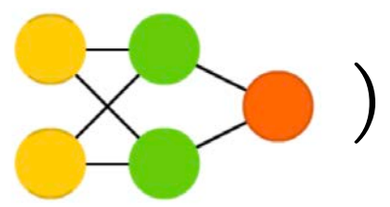
presynaptic activity

learning rate

reinforcing signal

Runarsson and Jonsson

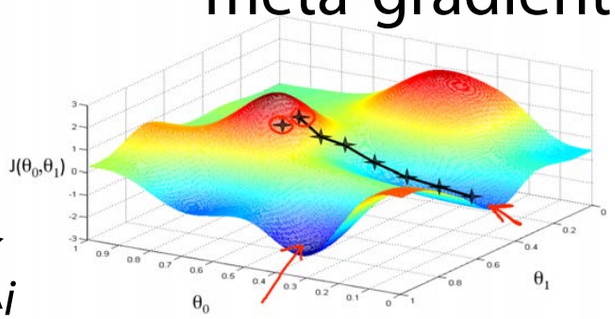
$$\Delta \theta_i = \eta \, (\quad)$$



learning rate

weights λ_i

meta-gradient

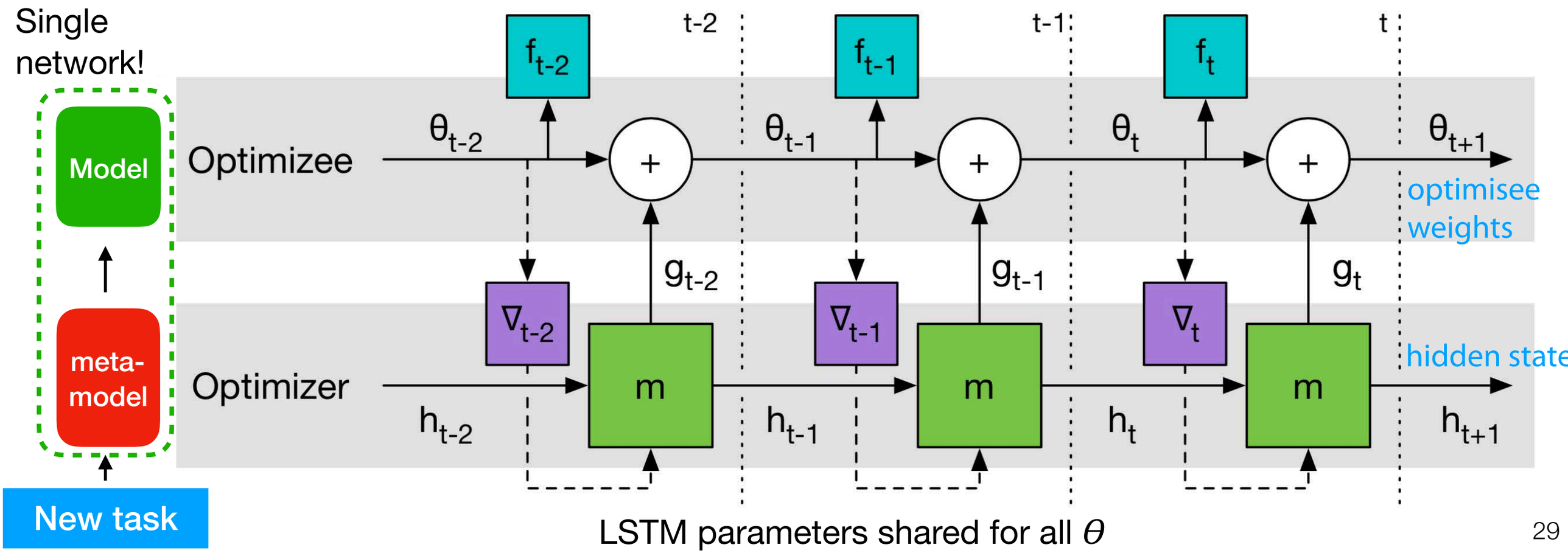


[1] Hochreiter 2001

[2] Andrychowicz et al. 2016

Learning to learn gradient descent by gradient descent

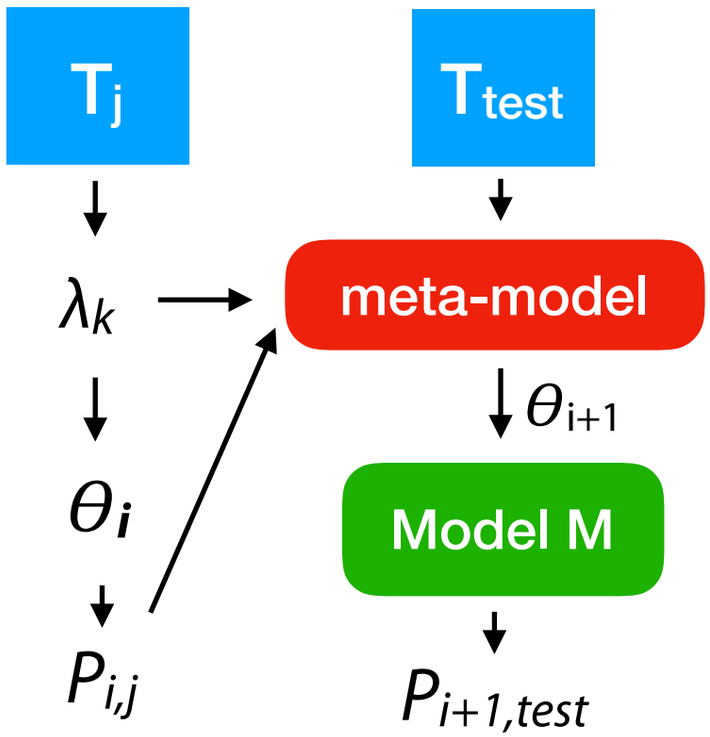
- Replace backprop with a recurrent neural net (LSTM)[1], not so scalable
- Use a coordinatewise LSTM [m] for scalability/flexibility (cfr. ADAM, RMSprop) [2]
  - Optimizee: receives weight update g_t from optimizer
  - Optimizer: receives gradient estimate ∇_t from optimizee
    - Learns how to do gradient descent across tasks

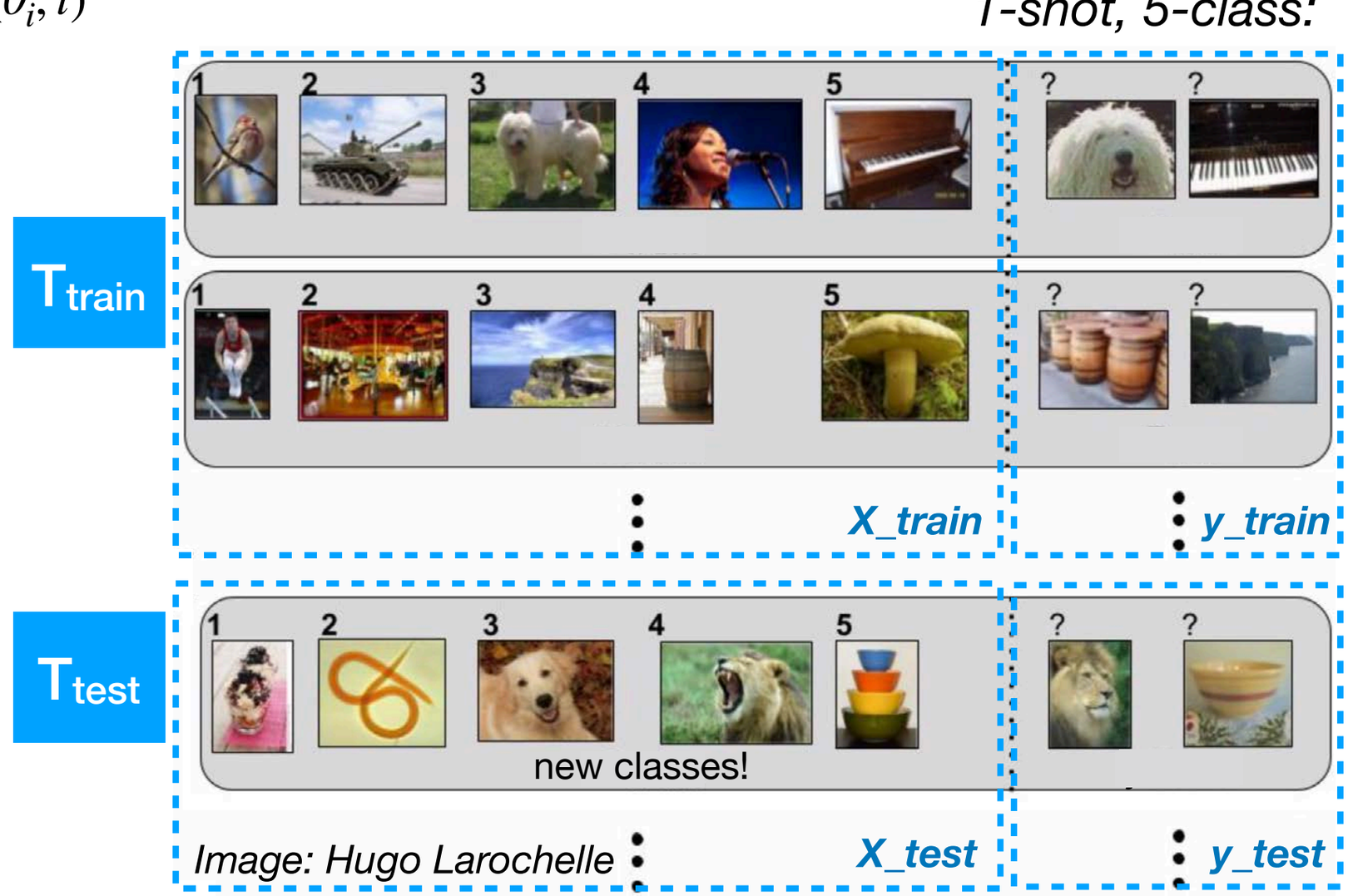


Few-shot learning

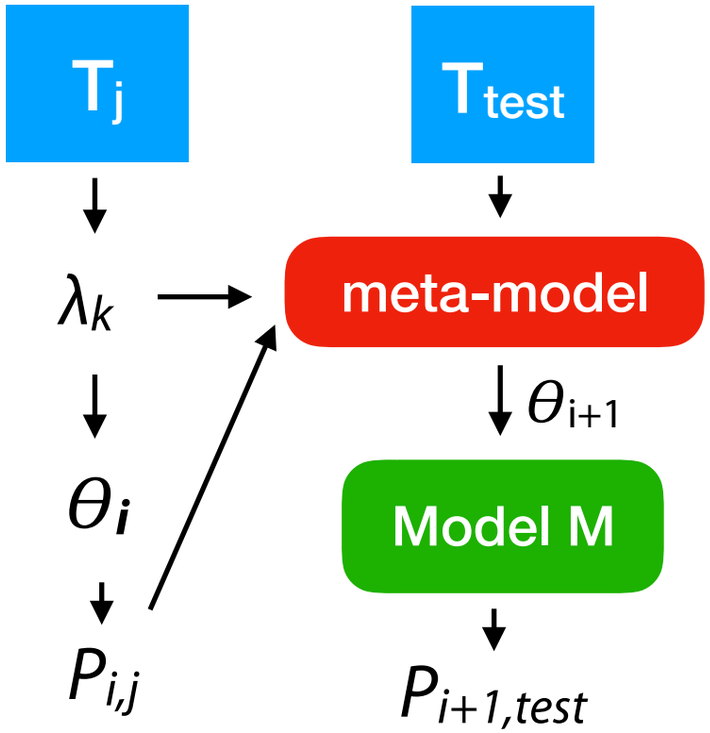
- Learn how to learn from few examples (given similar tasks)
  - Meta-learner must learn how to train a base-learner based on prior experience
    - Parameterize base-learner model and learn the parameters θ_i

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





Few-shot learning: approaches



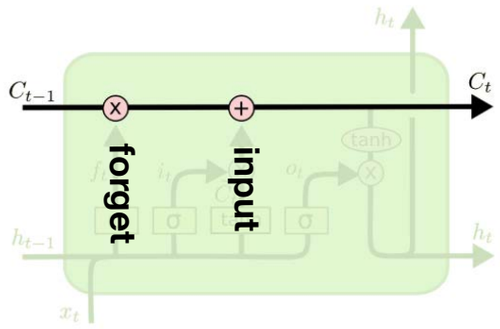
$$Cost(\theta_i) = \frac{1}{|T_{test}|} \sum_{t \in T_{test}} loss(\theta_i, t)$$

- Existing algorithm as meta-learner:
  - LSTM + gradient descent — *Ravi and Larochelle 2017*
  - Learn θ_{init} + gradient descent — *Finn et al. 2017*
  - kNN-like: Memory + similarity — *Vinyals et al. 2016*
  - Learn embedding + classifier — *Snell et al. 2017*
  - …
- Black-box meta-learner
  - Neural Turing machine (with memory) — *Santoro et al. 2016*
  - Neural attentive learner — *Mishra et al. 2018*
  - …

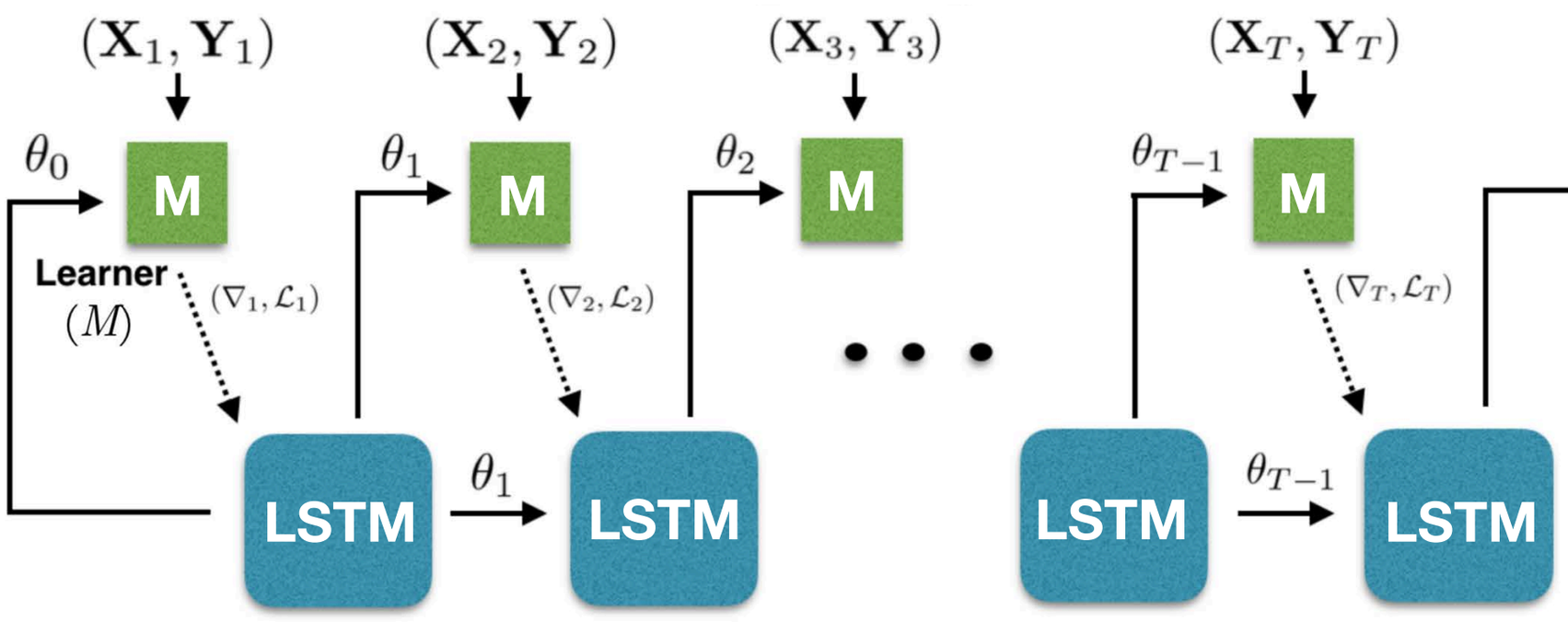
LSTM meta-learner + gradient descent

- Gradient descent update θ_t is similar to LSTM cell state update c_t

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t \qquad c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$



- Hence, training a meta-learner LSTM yields an update rule for training M
  - Start from initial θ_0, train model on first batch, get gradient and loss update
  - Predict θ_{t+1}, continue to t=T, get cost, backpropagate to learn LSTM weights, optimal θ_0

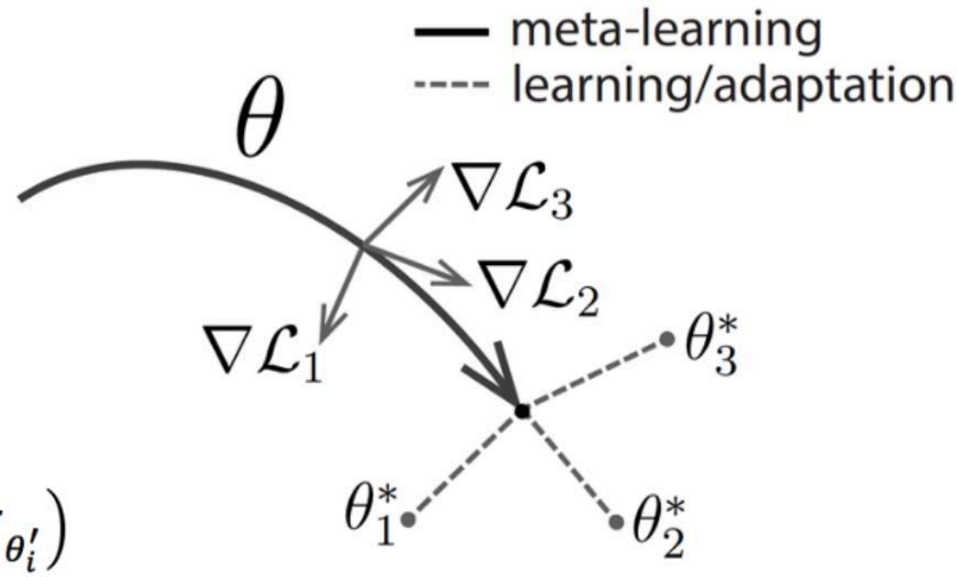


[1] Finn et al. 2017
[2] Finn et al. 2018
[3] Nichol et al. 2018

Model-agnostic meta-learning

- Quickly learn new skills by learning a model *initialization* that generalizes better to similar tasks
  - Current initialization θ
  - On K examples/task, evaluate $\nabla_\theta L_{T_i}(f_\theta)$
  - Update weights for $\theta_1, \theta_2, \theta_3$
  - Update θ to minimize sum of per-task losses
  - Repeat

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}\left(f_{\theta_i'}\right)$$

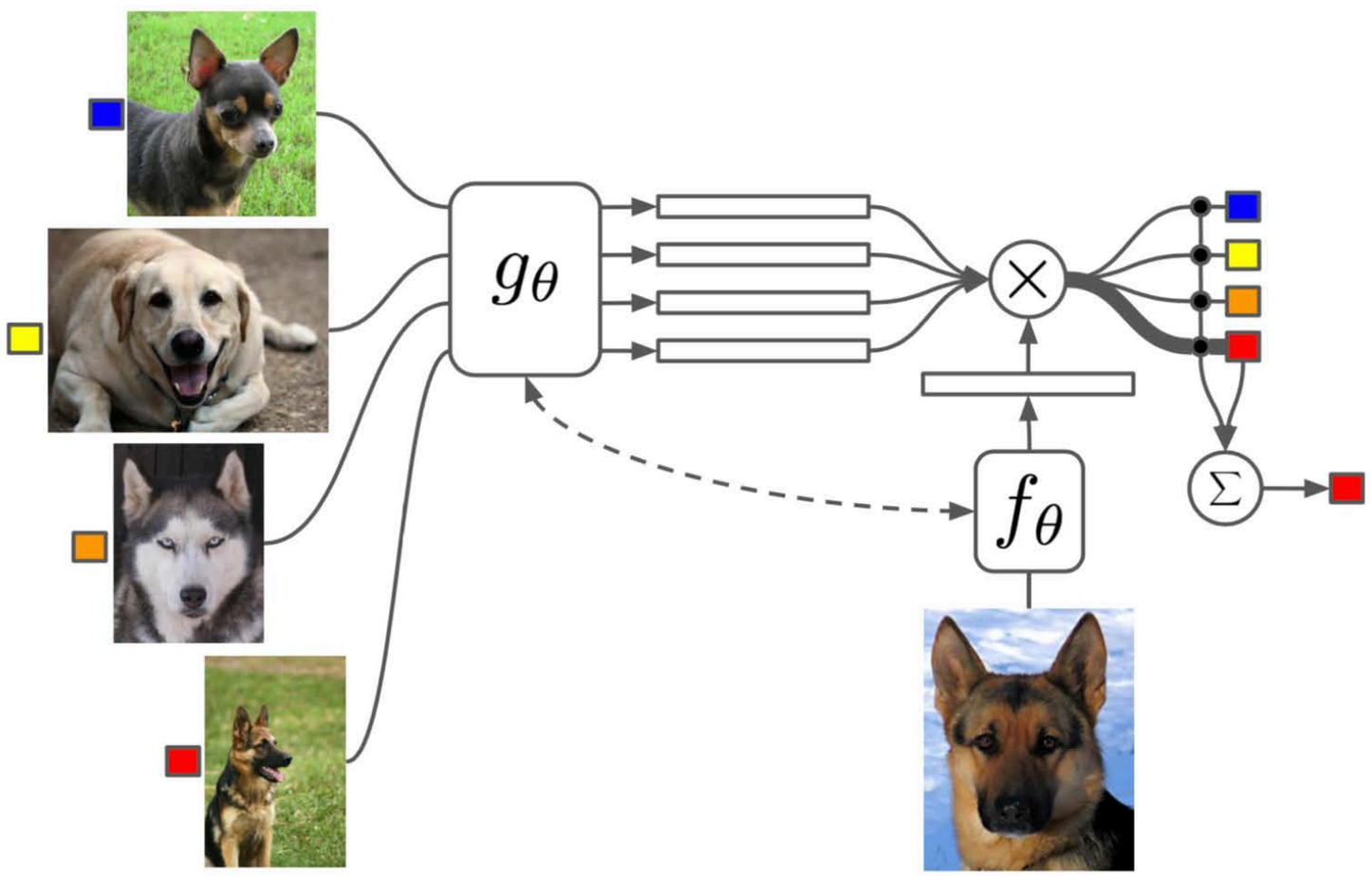


- More resilient to overfitting
- Generalizes better than LSTM approaches
- *Universality*: no theoretical downsides in terms of expressivity when compared to alternative meta-learning models.
- REPTILE: do SGD for k steps in one task, only then update initialization weights[3]

Vinyals et al. 2016

1-shot learning with Matching networks

- Don’t learn model parameters, use non-parameters model (like kNN)
- Choose an embedding network f and g (possibly equal)
- Choose an attention kernel $a(\hat{x}, x_i)$, e.g. softmax over cosine distance
- Train complete network in minibatches with few examples per task



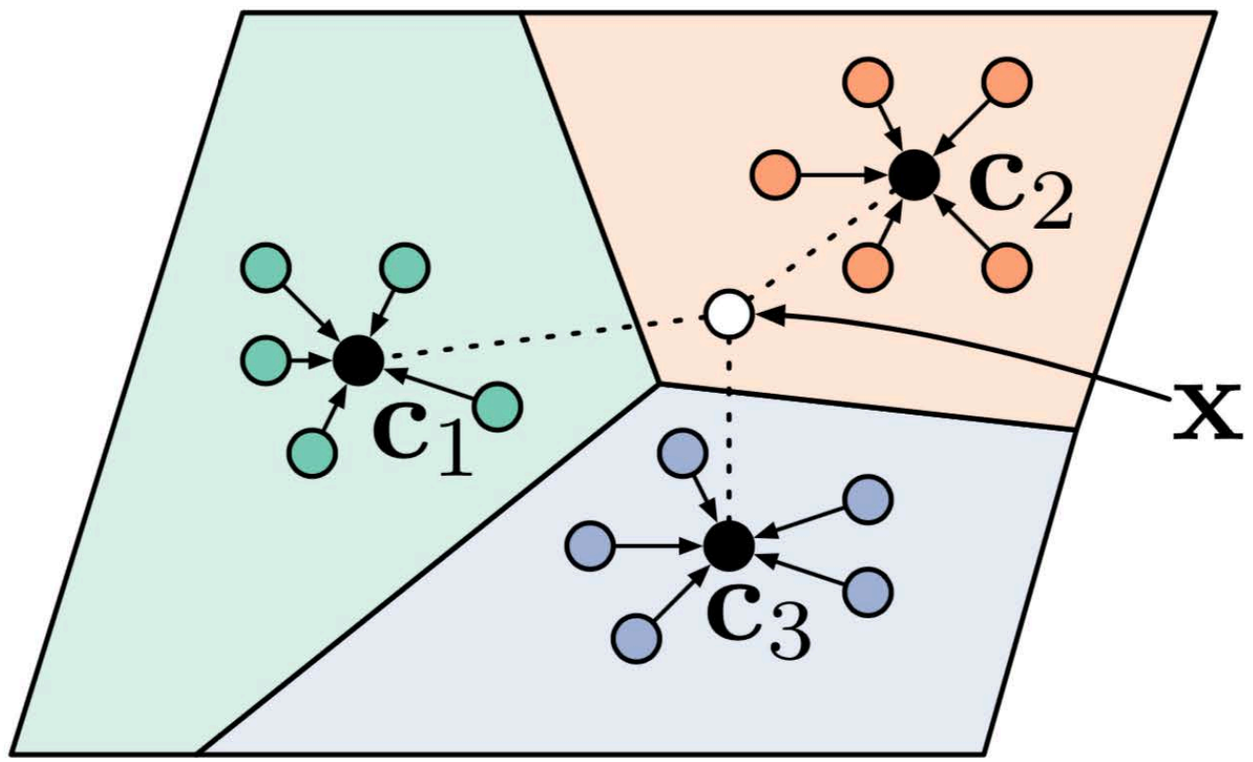
$$\hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i$$

θ = {VGG, Inception,…}

[Snell et al. 2017](#)
[Ren et al. 2018](#)

Prototypical networks

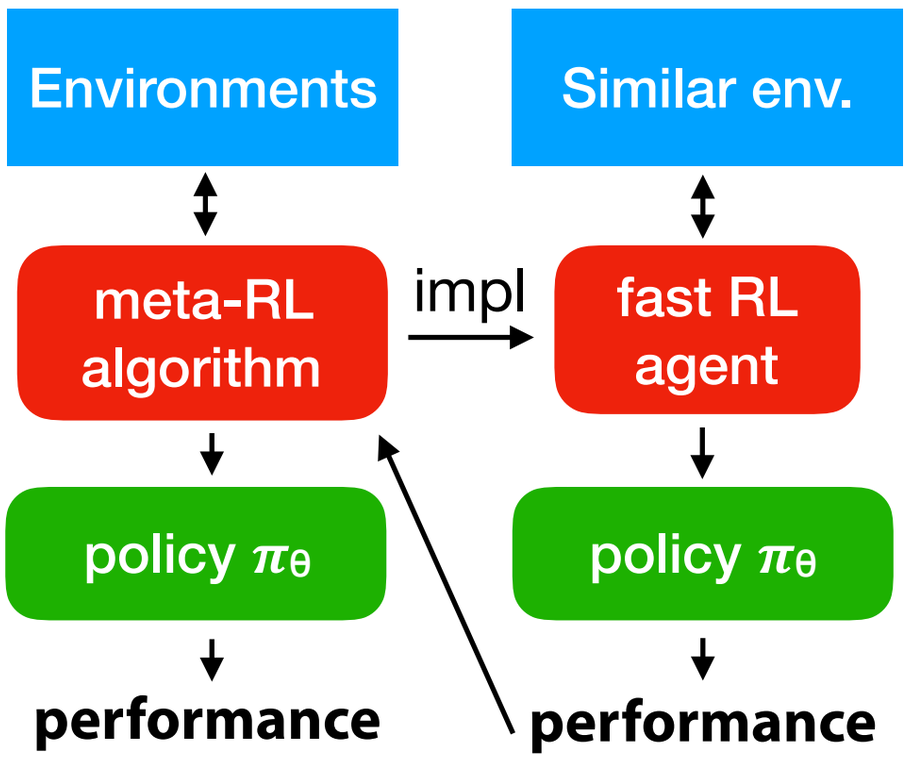
- Train a "prototype extractor" network
- Map examples to p-dimensional embedding so examples of a given class are close together
- Calculate a prototype (mean vector) for every class
- Map test instances to the same embedding, use softmax over distance to prototype
- Using more classes during meta-training works better!

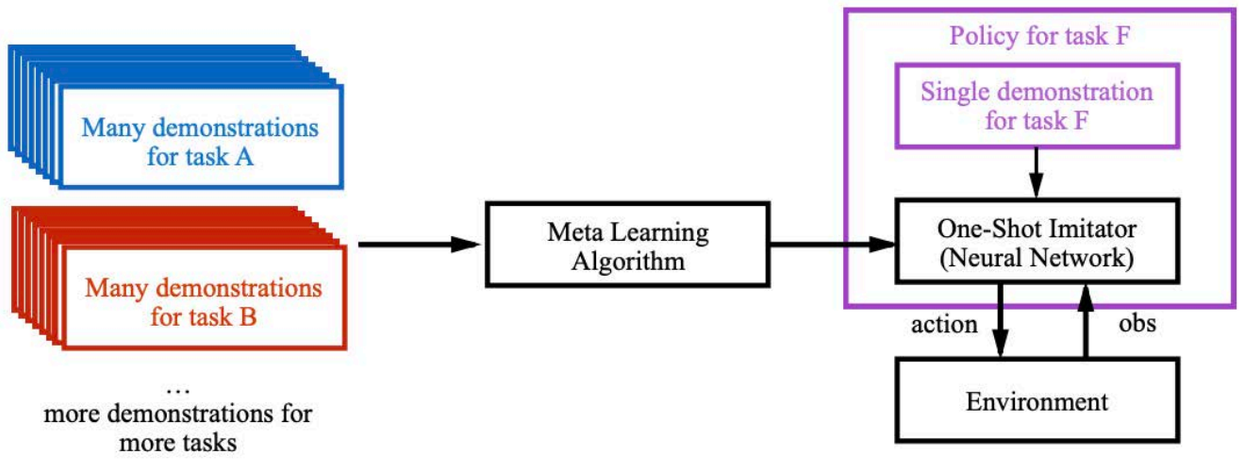


[1] Duan et al. 2017
[2] Wang et al. 2017
[3] Duan et al. 2017

Learning to reinforcement learn

- Humans often learn to play new games much faster than RL techniques do
- Reinforcement learning is very suited for learning-to-learn:
  - Build a learner, then use performance as that learner as a reward
- Learning to reinforcement learn [1,2]
  - Use RNN-based deep RL to train a recurrent network on many tasks
  - Learns to implement a 'fast' RL agent, encoded in its weights





- Also works for few-shot learning [3]
  - Condition on observation + upcoming demonstration
- You don't know what someone is trying to teach you, but you prepare for the lesson

Learning to learn more tasks

- Active learning — *Pang et al. 2018*
  - Deep network (learns representation) + policy network
  - Receives state and reward, says which points to query next
- Density estimation — *Reed et al. 2017*
  - Learn distribution over small set of images, can generate new ones
  - Uses a MAML-based few-shot learner
- Matrix factorization — *Vartak et al. 2017*
  - Deep learning architecture that makes recommendations
  - Meta-learner learns how to adjust biases for each user (task)
- Replace hand-crafted algorithms by learned ones.
- Look at problems through a meta-learning lens!

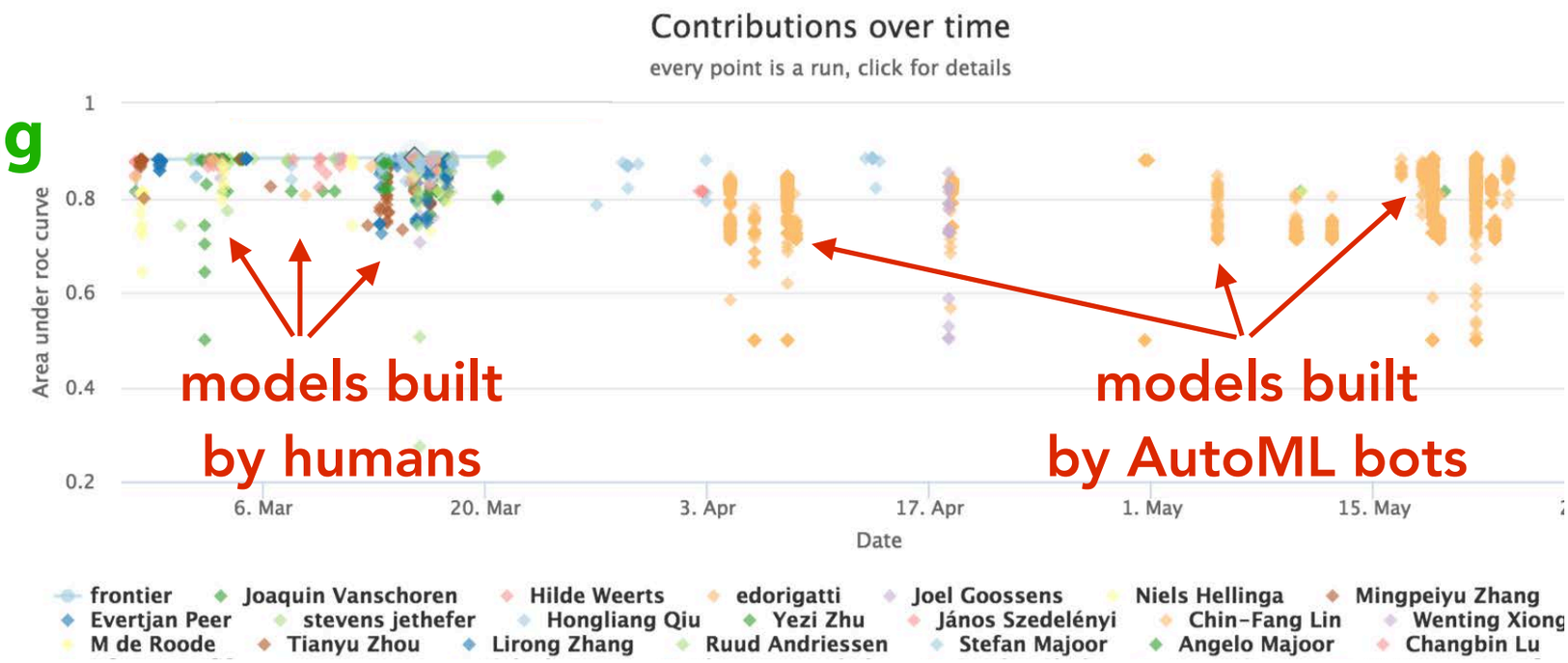
Meta-data sharing building a shared memory

- OK, but how do I get large amounts of meta-data for meta-learning?
- OpenML.org
  - Thousands of uniform datasets
  - 100+ meta-features
  - Millions of evaluated runs
    - Same splits, 30+ metrics
    - Traces, models *(opt)*
- **APIs in Python, R, Java,…**
- Publish your own runs
- **Never ending learning**
- Benchmarks

```
import openml as oml
from sklearn import tree

task = oml.tasks.get_task(14951)
clf = tree.ExtraTreeClassifier()
flow = oml.flows.sklearn_to_flow(clf)
run = oml.runs.run_flow_on_task(task, flow)
myrun = run.publish()
```

run *locally,* share *globally*



Open positions!
Scientific programmer
Teaching PhD

Towards human-like learning to learn

- Learning-to-learn gives humans a significant advantage
  - **Learning how to learn any task empowers us far beyond knowing how to learn specific tasks.**
  - It is a **universal** aspect of life, and how it evolves
- Very exciting field with many unexplored possibilities
  - Many aspects not understood (e.g. task similarity), need more experiments.
- **Challenge**:
  - Build learners that **never stop learning**, that **learn from each other**
  - Build a ***global memory*** for learning systems to learn from
  - **Let them explore by themselves**, active learning

Thank you!

Merci!

more to learn

http://www.automl.org/book/

Chapter 2: Meta-Learning

special thanks to

Pavel Brazdil, Matthias Feurer, Frank Hutter, Erin Grant,
Hugo Larochelle, Raghu Rajan, Jan van Rijn, Jane Wang