An Opinionated Introduction to AutoML

Yi Zhang

2019.4.11

Adapted from AutoML tutorial at NeurIPS 2018

What is AutoML?

- Traditionally, the term has been used to describe *automated* methods for model selection and/or hyperparameter optimization.

- People often feel like they are just guessing as they test out different hyperparameters for a model, and automating the process could make this piece of the machine learning pipeline easier, as well as speeding things up even for experienced machine learning practitioners.
  - AutoWEKA
  - Auto-sklearn

However…

- Model is becoming increasingly complicated.
  - Model selection -> Architecture Selection
- People want to build machine learning products.
  - Reusability
  - Understand what level of performance is possible for a problem.
  - Reproducibility

Much Effort is Necessary

- 1. Hyperparameter Tuning
- 2. Architecture/Pipeline Search
- 3. Meta/Transfer Learning
- 4. MultiTask Learning
- 5. Data/Model Management System
- 6. Distributed learning algorithm
- 7. Code Understanding
- 8. HCI
- …

Today

- 1. Hyperparameter Tuning
- 2. **Architecture/Pipeline Search**
- 3. **Meta/Transfer Learning**
- 4. MultiTask Learning
- 5. Data/Model Management System
- 6. Distributed learning algorithm
- 7. Code Understanding
- 8. HCI
- …

Neural Architecture Search

NAS

- 1. Search Space
- 2. Search Strategy
- 3. Performance Estimation Strategy

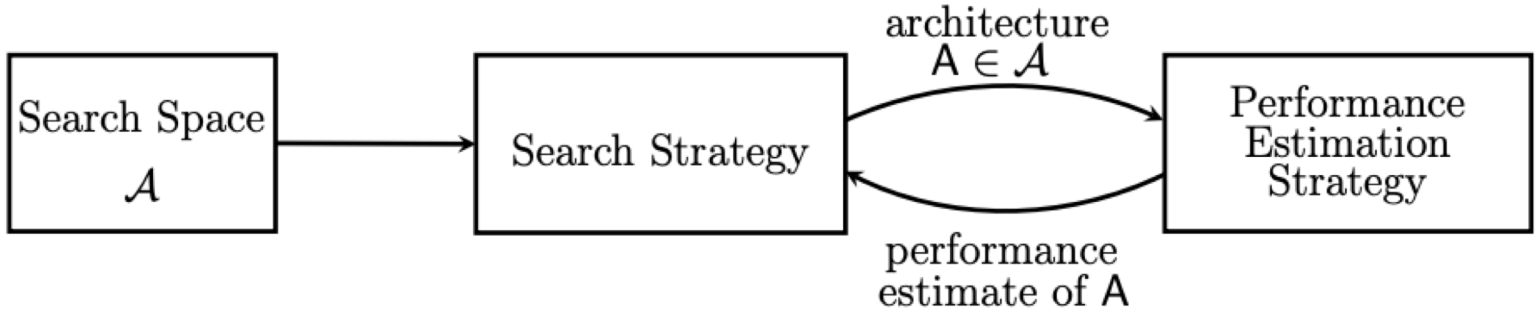
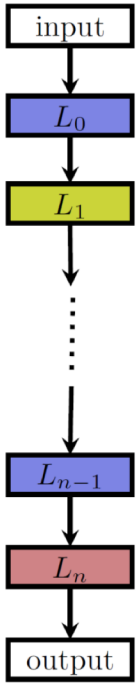
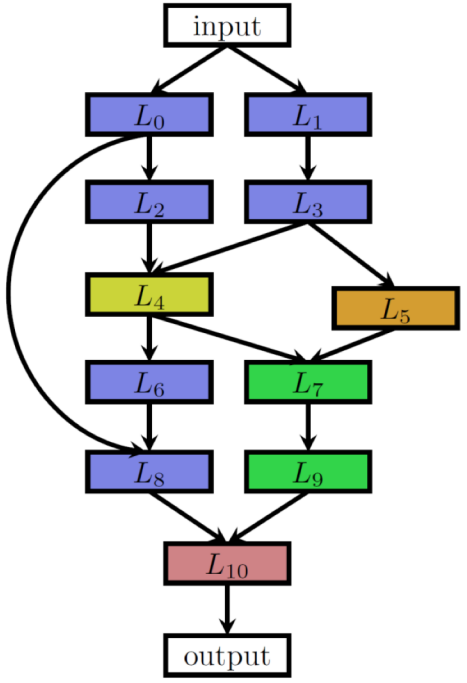


Figure 3.1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

Basic Neural Architecture Search Spaces



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



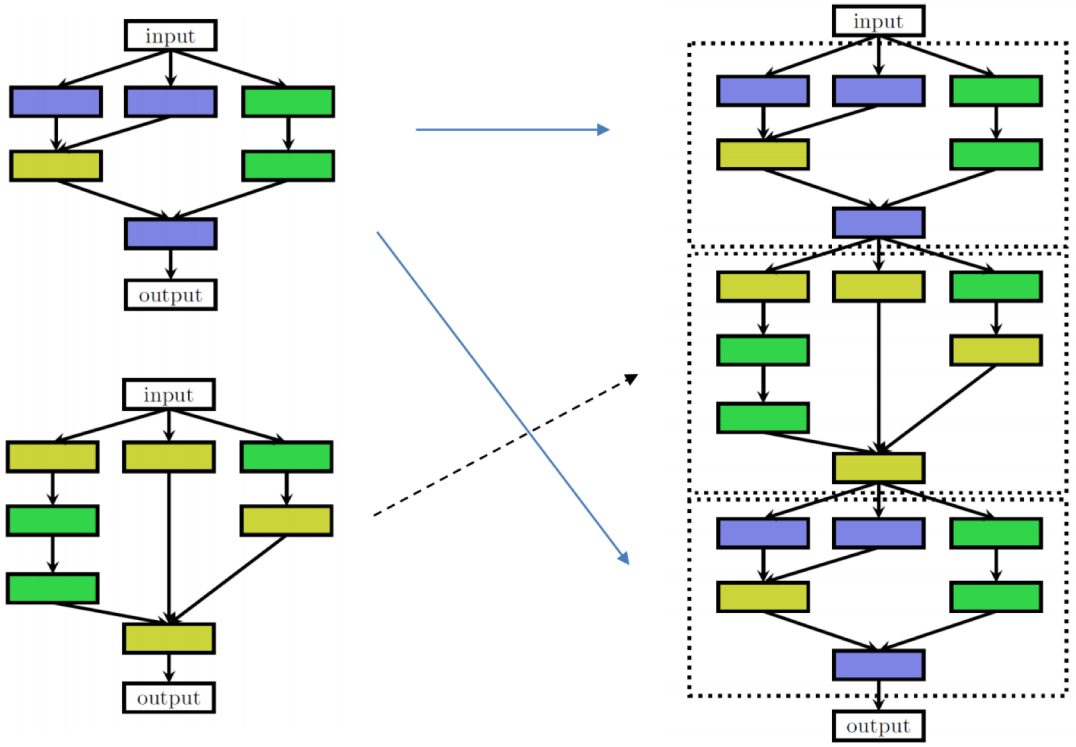
More complex space
with multiple branches
and skip connections

Cell Search Spaces

Introduced by Zoph et al [CVPR 2018]

Normal cell

Reduction cell

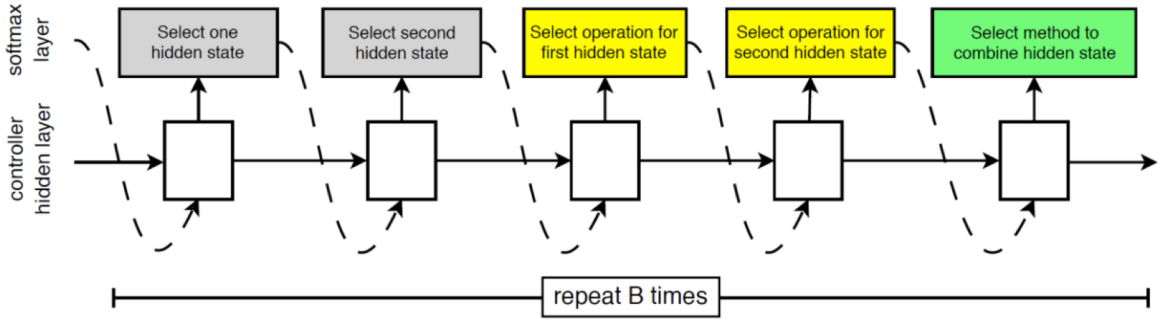


Two possible cells

Architecture composed of stacking together individual cells

NAS as Hyperparameter Optimization

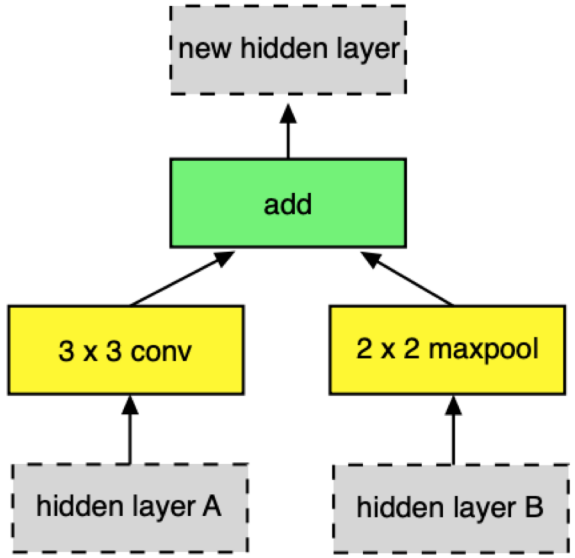
- Cell search space by Zoph et al [CVPR 2018]



  - 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



The difficulty of the optimization:

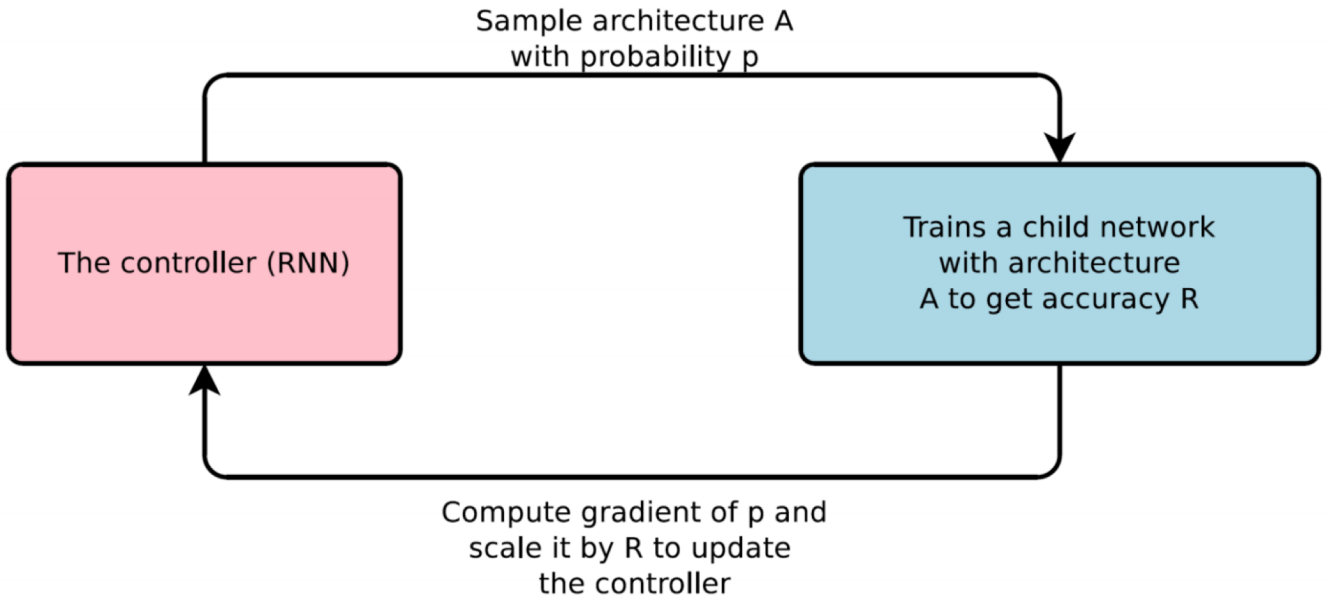
- 1. non-continuous
- 2. relatively high-dimensional

Search Strategy

- Bayesian Optimization
  - Early successes since 2013, leading to some SOTA in vision, and the first automatically-tuned neural networks to win competition datasets against human experts.
- Evolutionary method
- Reinforcement Learning
  - NAS became a mainstream research topic in ML community
- Gradient-based methods

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**



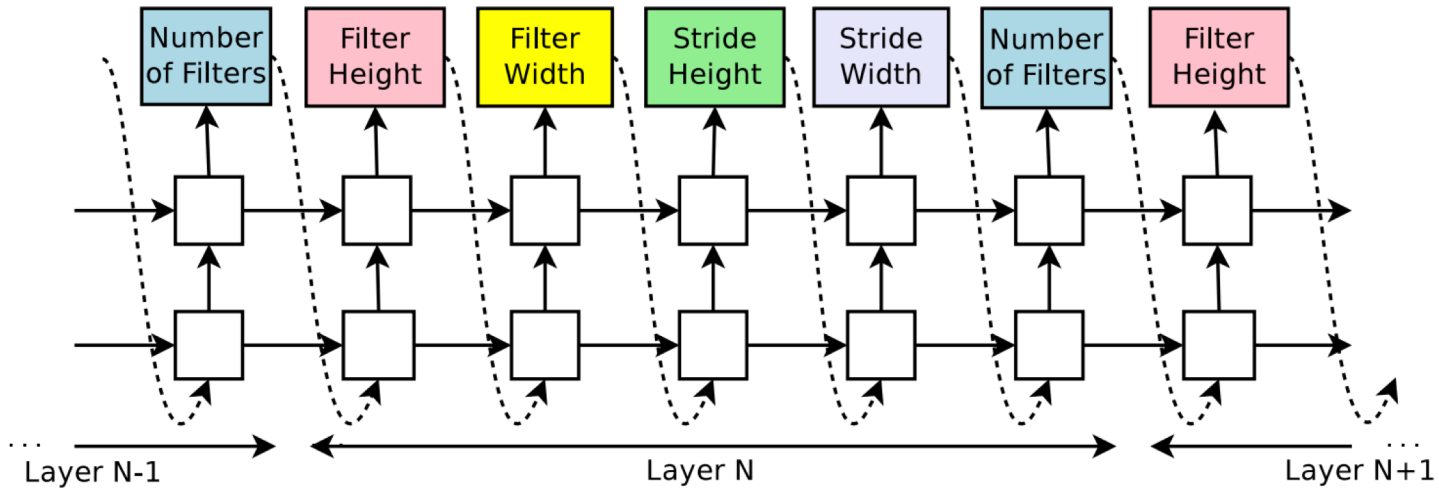
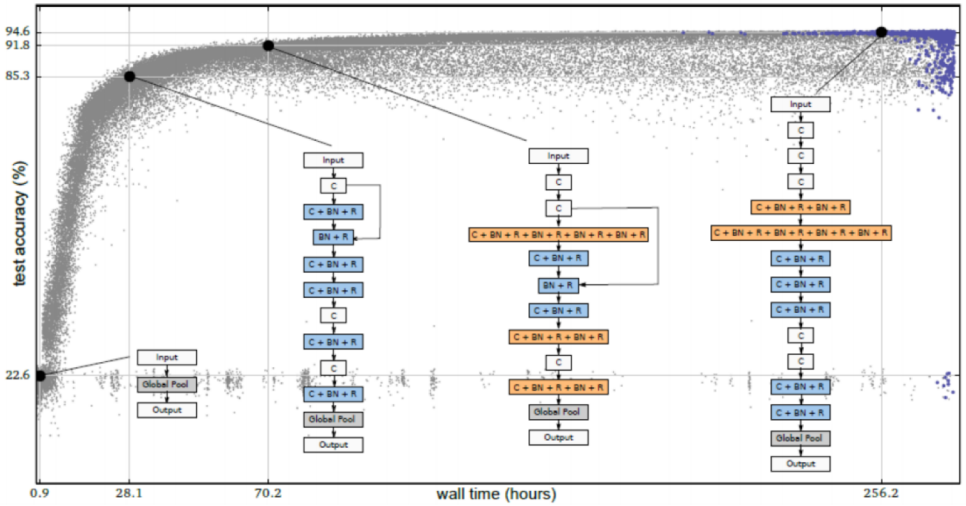


Figure 2: How our controller recurrent neural network samples a simple convolutional network. It predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by a softmax classifier and then fed into the next time step as input.

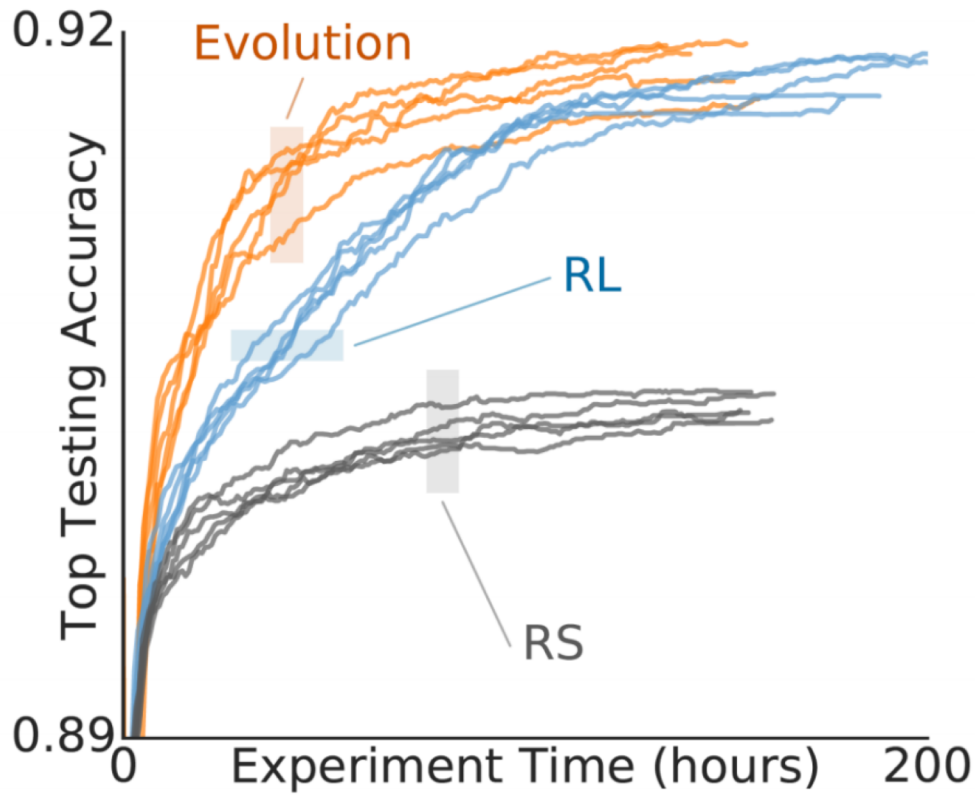
Evolution

- 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]



- 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

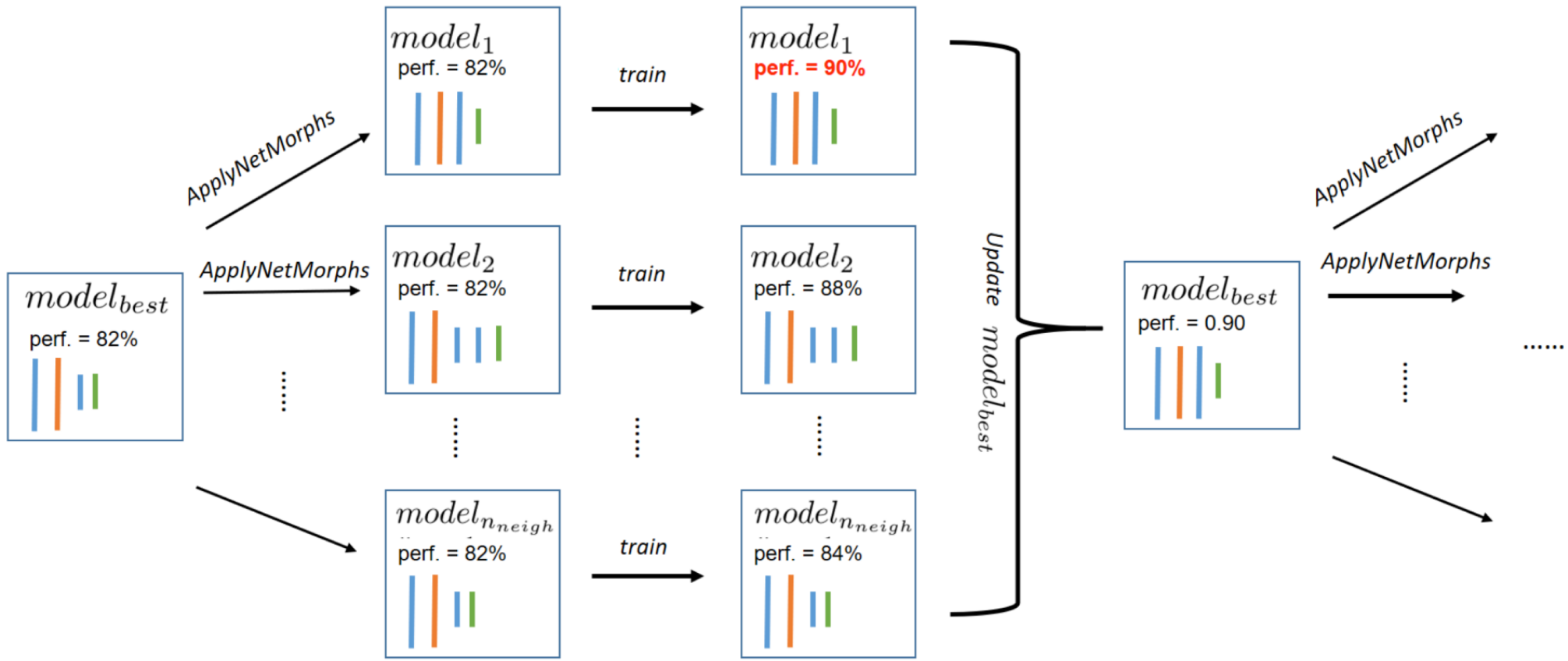
- BO is one of the most popular methods for hyperparameter optimization, but it has not been applied to NAS by many groups since typical BO toolboxes are based on Gaussian processes and focus on low-dimensional continuous optimization problems.

- Variations exist, while a full comparison is lacking.

Beyond Blackbox Optimization

- **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

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



→ enables efficient architecture search

One-shot Architecture

- Treats all architectures as different subgraphs of a supergraph (the one-shot model) and shares weights between architectures that have edges of this supergraph in common.
- Only the weights of a single one-shot model need to be trained, and architectures can then be evaluated without any separate training by inheriting trained weights from the one-shot model.

ENAS

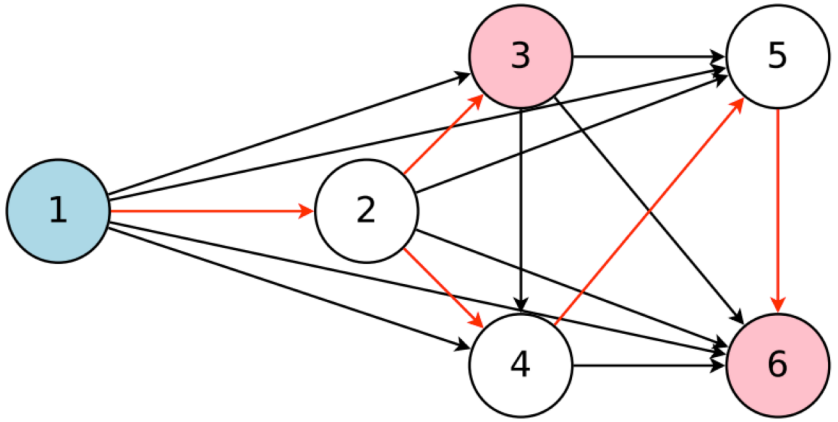


Figure 2. The graph represents the entire search space while the red arrows define a model in the search space, which is decided by a controller. Here, node 1 is the input to the model whereas nodes 3 and 6 are the model's outputs.

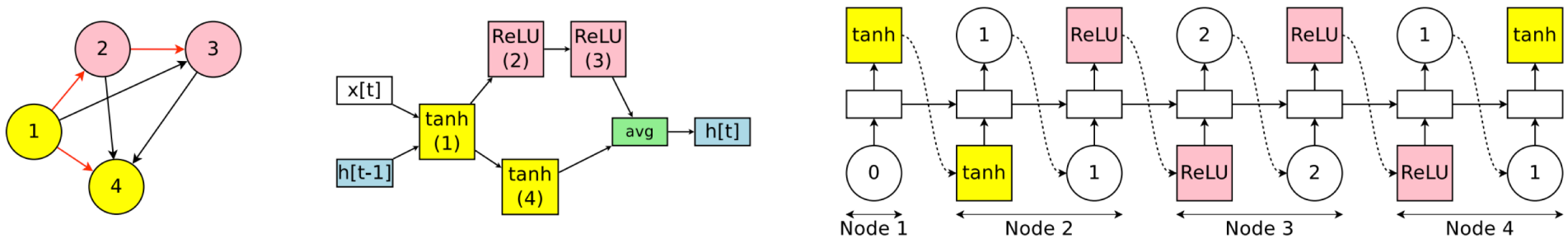


Figure 1. An example of a recurrent cell in our search space with 4 computational nodes. *Left:* The computational DAG that corresponds to the recurrent cell. The red edges represent the flow of information in the graph. *Middle:* The recurrent cell. *Right:* The outputs of the controller RNN that result in the cell in the middle and the DAG on the left. Note that nodes 3 and 4 are never sampled by the RNN, so their results are averaged and are treated as the cell's output.

Training

- Controller network is an LSTM with 100 hidden units.
- The LSTM samples decisions via softmax classifiers.
- Two sets of learnable parameters:
  - 1. the parameters of the controller LSTM, θ
  - 2. the shared parameters of the child models, ω
- 1. Fix the controller's policy and perform stochastic gradient descent ω to minimize the expected loss function.
- 2. Fix ω and update the policy parameters θ, aiming to maximize the expected reward.

DARTS

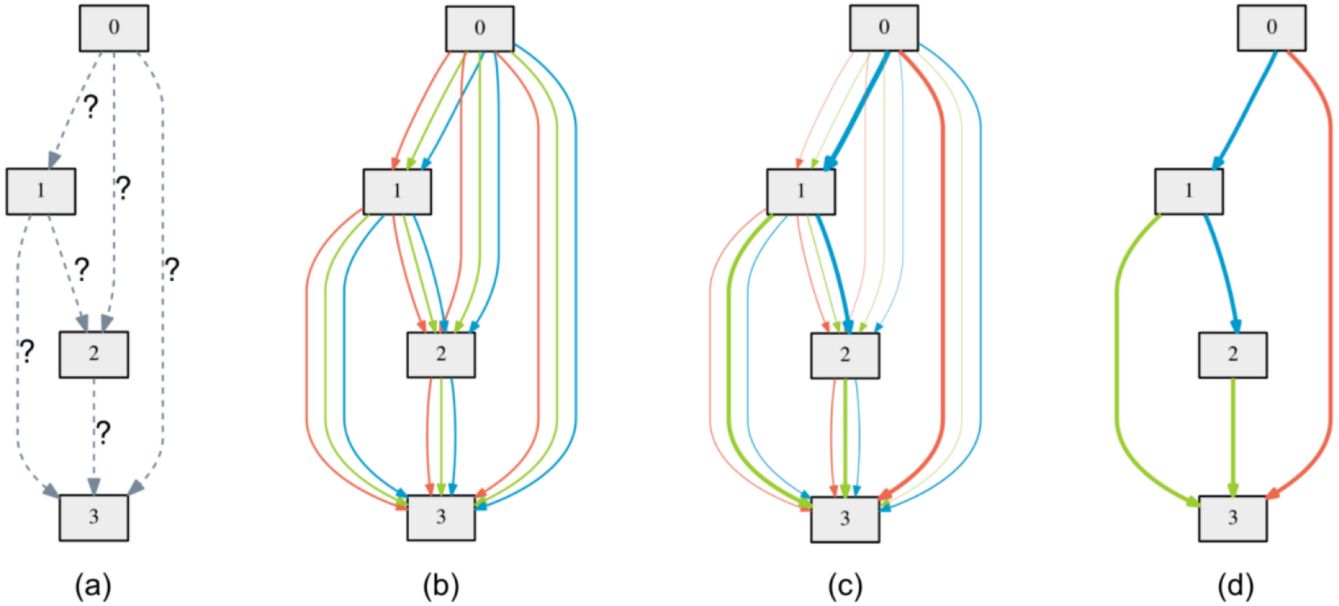


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem.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

- 1. Relax the categorical choice of a particular operation as a softmax over all possible operations:

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

- The task of architecture search reduces to learning a set of continuous variables $\alpha = \left\{\alpha^{(i,j)}\right\}$.
- A discrete architecture is obtained by replacing each mix operation $\bar{o}^{(i,j)}$ with the most likely operation.

Table 1: Comparison with state-of-the-art image classifiers on CIFAR-10. Results marked with † were obtained by training the corresponding architectures using our setup.

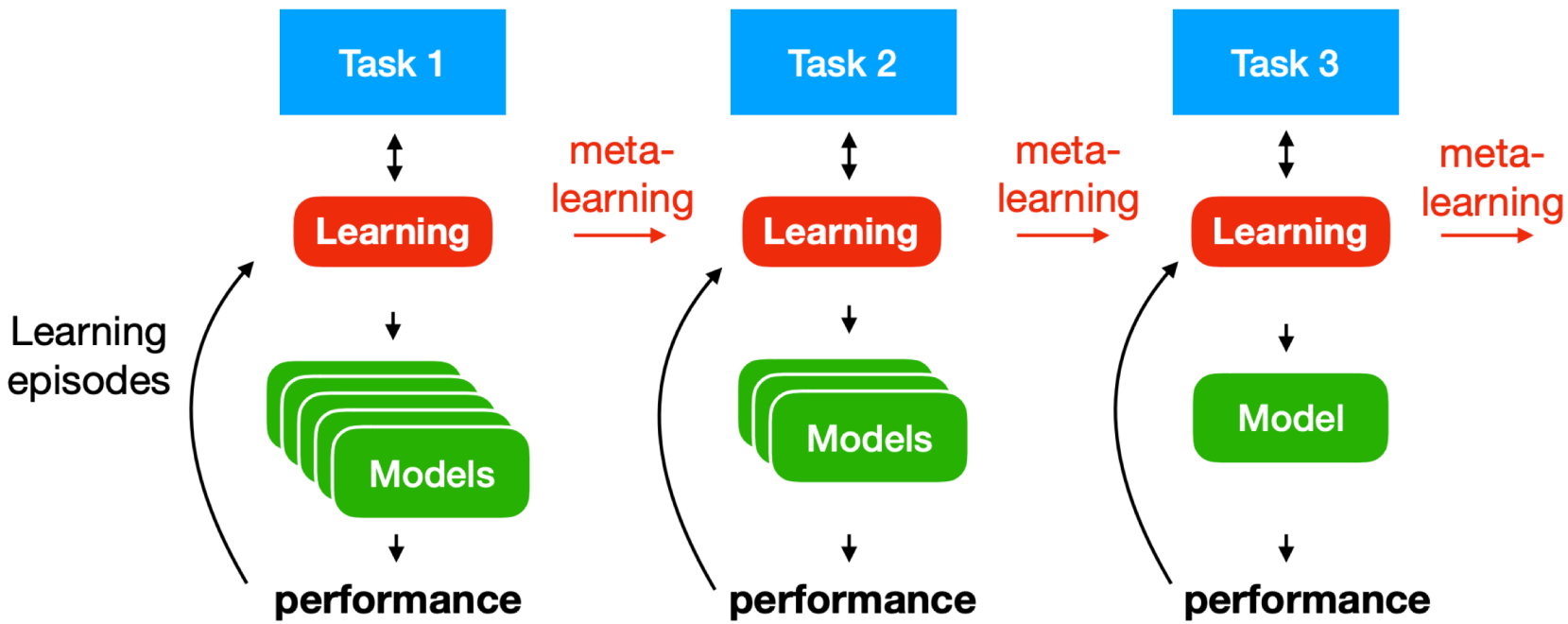
Architecture	**Test Error (%)**	**Params (M)**	**Search Cost (GPU days)**	**Search Method**
DenseNet-BC (Huang et al., 2017)	3.46	25.6	–	manual
NASNet-A + cutout (Zoph et al., 2017)	2.65	3.3	1800	RL
NASNet-A + cutout (Zoph et al., 2017)†	2.83	3.1	3150	RL
AmoebaNet-A + cutout (Real et al., 2018)	3.34 ± 0.06	3.2	3150	evolution
AmoebaNet-A + cutout (Real et al., 2018)†	3.12	3.1	3150	evolution
AmoebaNet-B + cutout (Real et al., 2018)	2.55 ± 0.05	2.8	3150	evolution
Hierarchical Evo (Liu et al., 2017b)	3.75 ± 0.12	15.7	300	evolution
PNAS (Liu et al., 2017a)	3.41 ± 0.09	3.2	225	SMBO
ENAS + cutout (Pham et al., 2018b)	2.89	4.6	0.5	RL
Random + cutout	3.49	3.1	–	–
DARTS (first order) + cutout	2.94	2.9	1.5	gradient-based
DARTS (second order) + cutout	2.83 ± 0.06	3.4	4	gradient-based

Meta-Learning

Learning is a never-ending process

Tasks come and go, but learning is forever

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

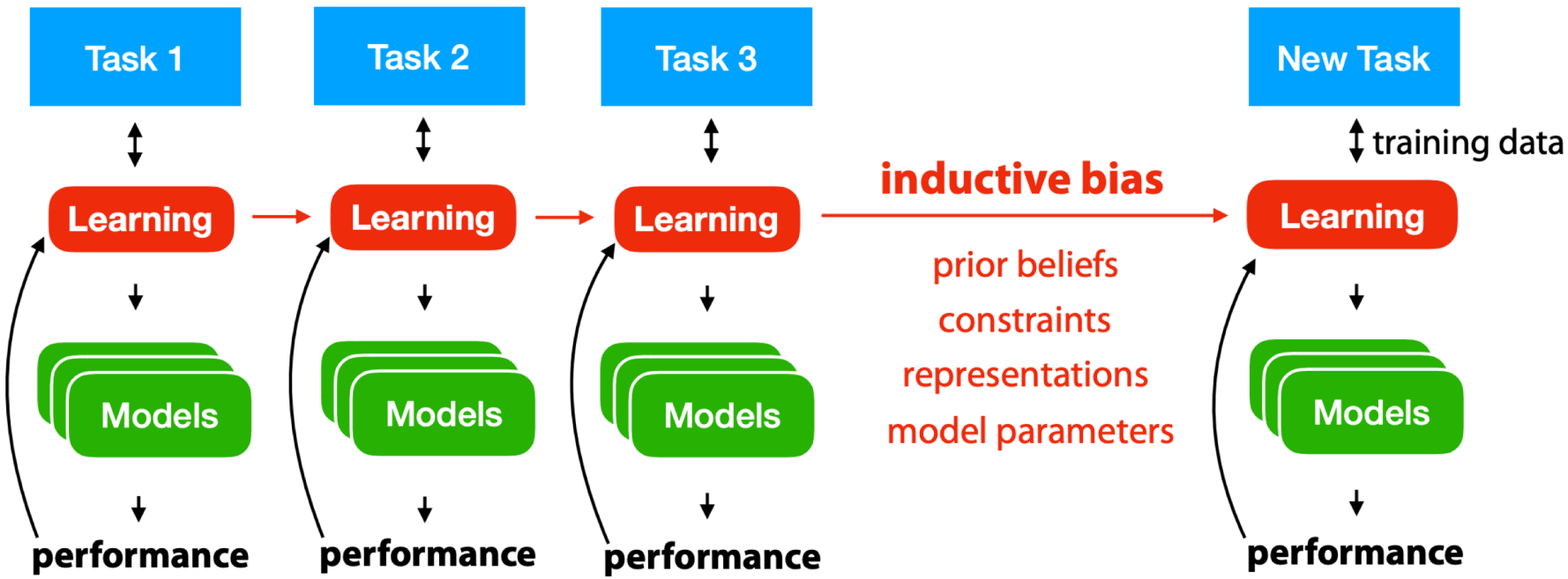


Learning to learn

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: Step 1

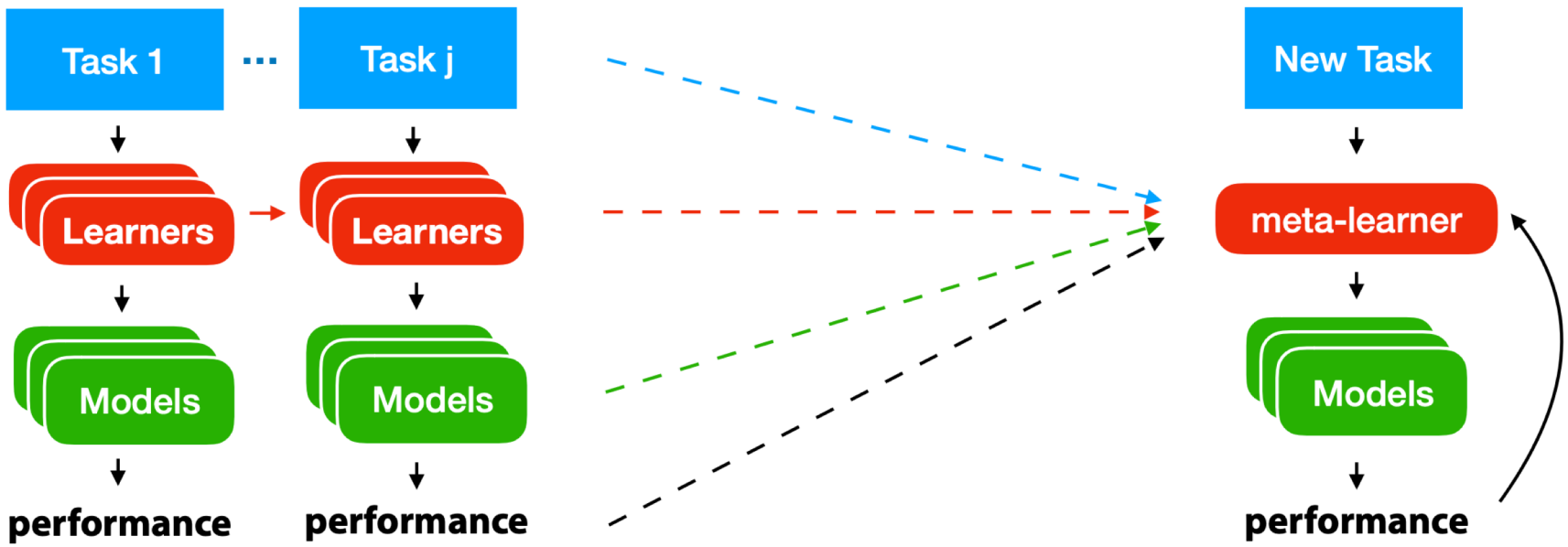
- Meta-data: describe prior learning tasks and previously learned model.
  - Exact algorithm configurations
    - Hyperparameter settings
    - Pipeline compositions
    - Network architectures
  - Model evaluations
    - Accuracy
    - Training time
    - Learned model parameters
  - Mete-features: measurable properties of the task

Meta-learning: Step 2

- Learn a (base-) learning algorithm
  - Learn from prior meta-data, to extract and transfer knowledge that guides the search for optimal models for new tasks.

Three approaches for increasingly similar tasks

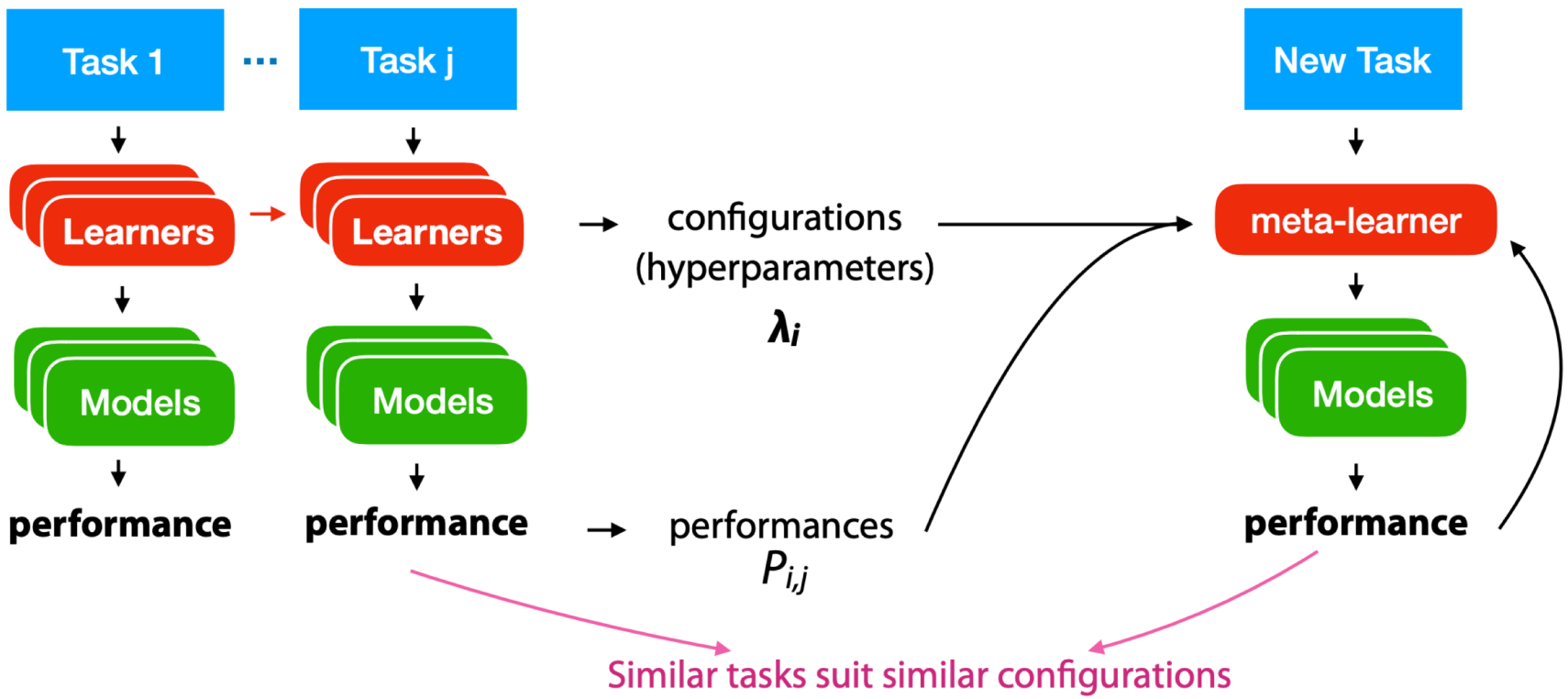
1. Transfer prior knowledge about what generally works well
2. Reason about model performance across tasks
3. Start from models trained earlier on similar tasks



1. Learning from prior evaluations

Configurations: settings that uniquely define the model

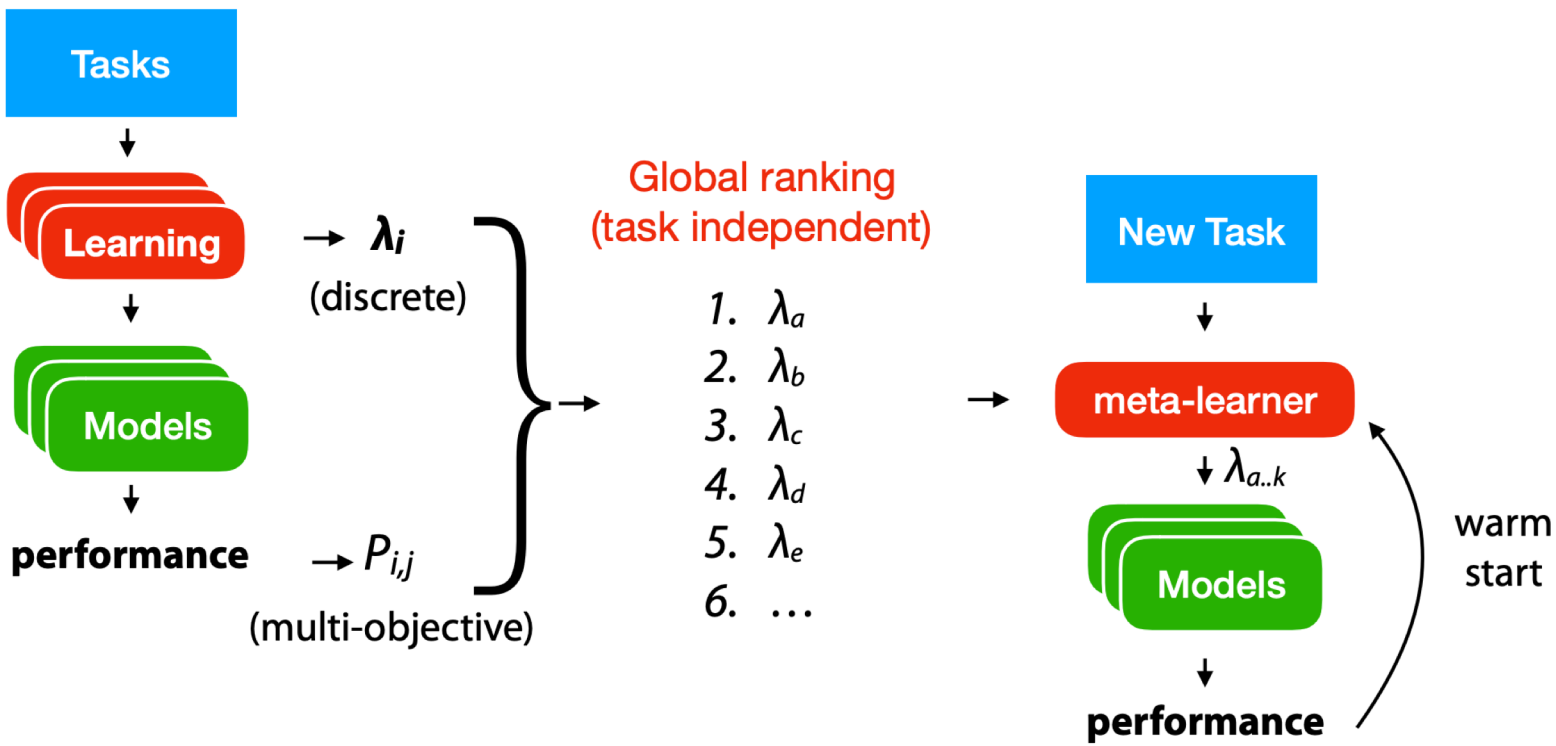
(algorithm, pipeline, neural architecture, hyper-parameters, …)



Leite et al. 2012
Abdulrahman et al. 2018

Top-K recommendation

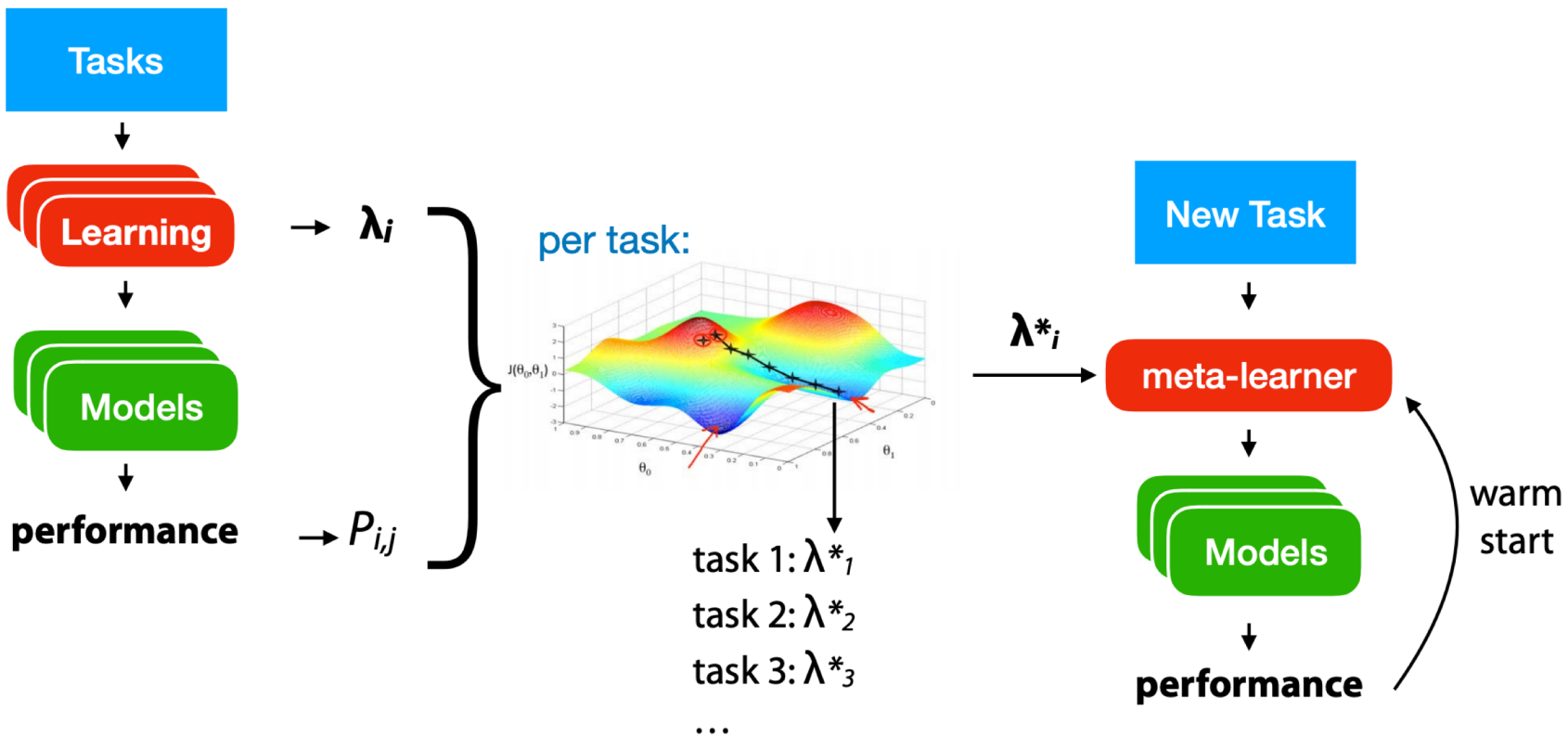
- Build a *global (multi-objective) ranking,* recommend the top-K
- Requires fixed selection of candidate configurations *(portfolio)*
- Can be used as a warm start for optimization techniques



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



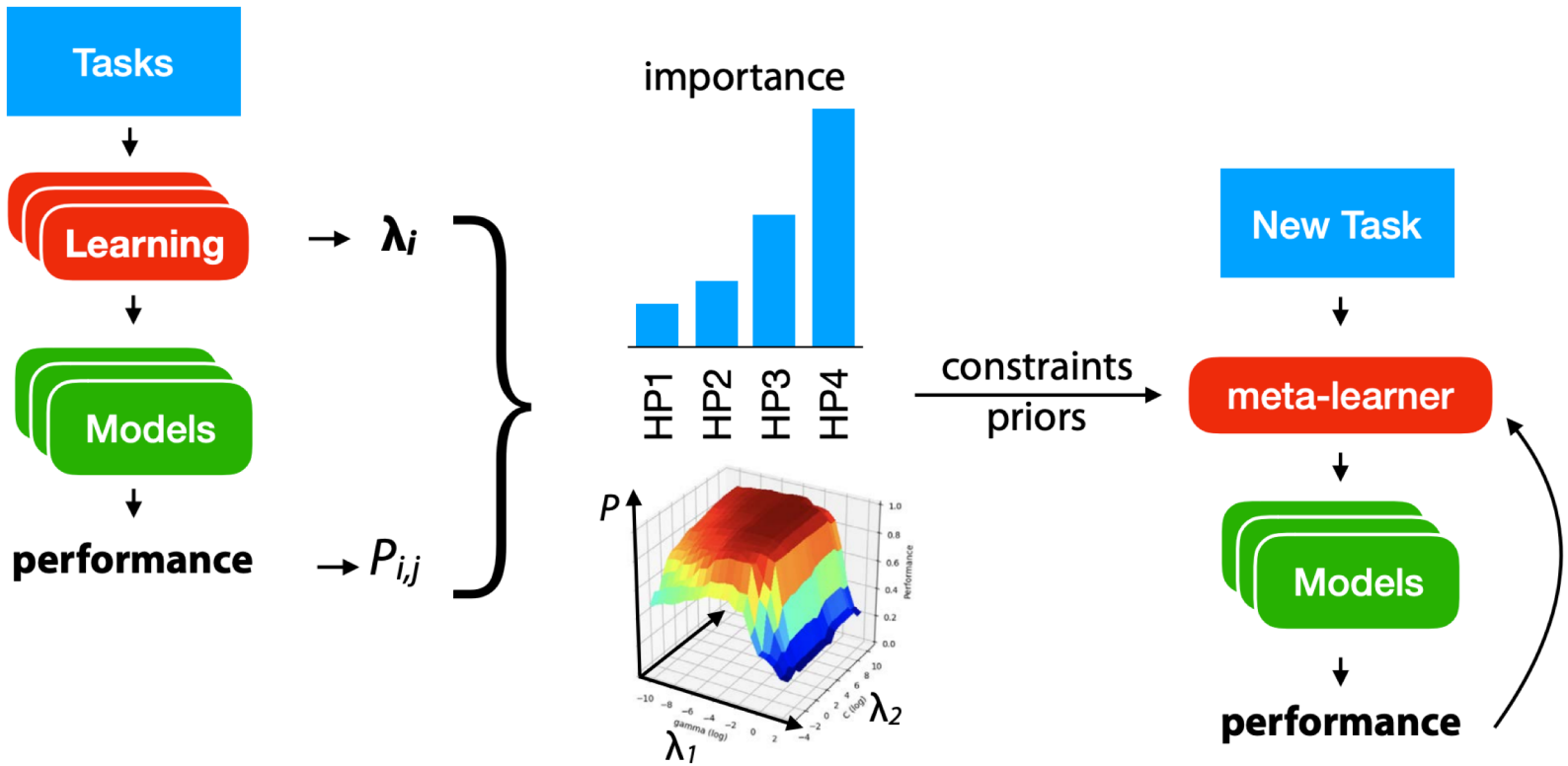
Configuration space design

[1] van Rijn & Hutter 2018

[2] Probst et al. 2018

[3] Wistuba et al. 2015

- **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]



Configuration Transfer

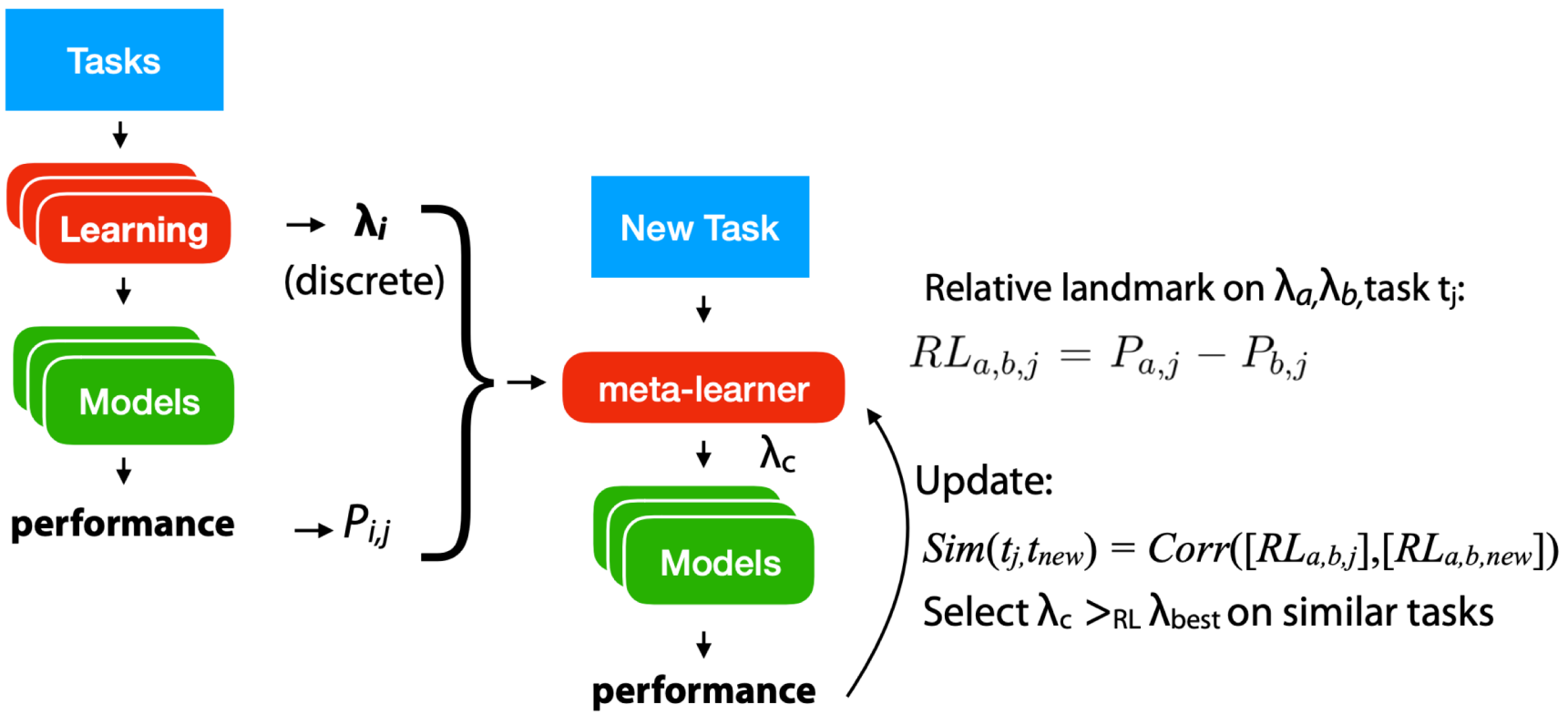
- If we want to provide recommendations for a specific new task t_n, we need information of how similar t_n is to some prior task t_j.

- 1. Evaluate recommended configurations on t_n, yielding new evidence P_n. If we can observe that evaluations $P_{i,n}$ are similar to $P_{i,j}$, then t_n, t_j can be considered intrinsically similar, based on empirical evidence.

Leite et al. 2012

Active testing

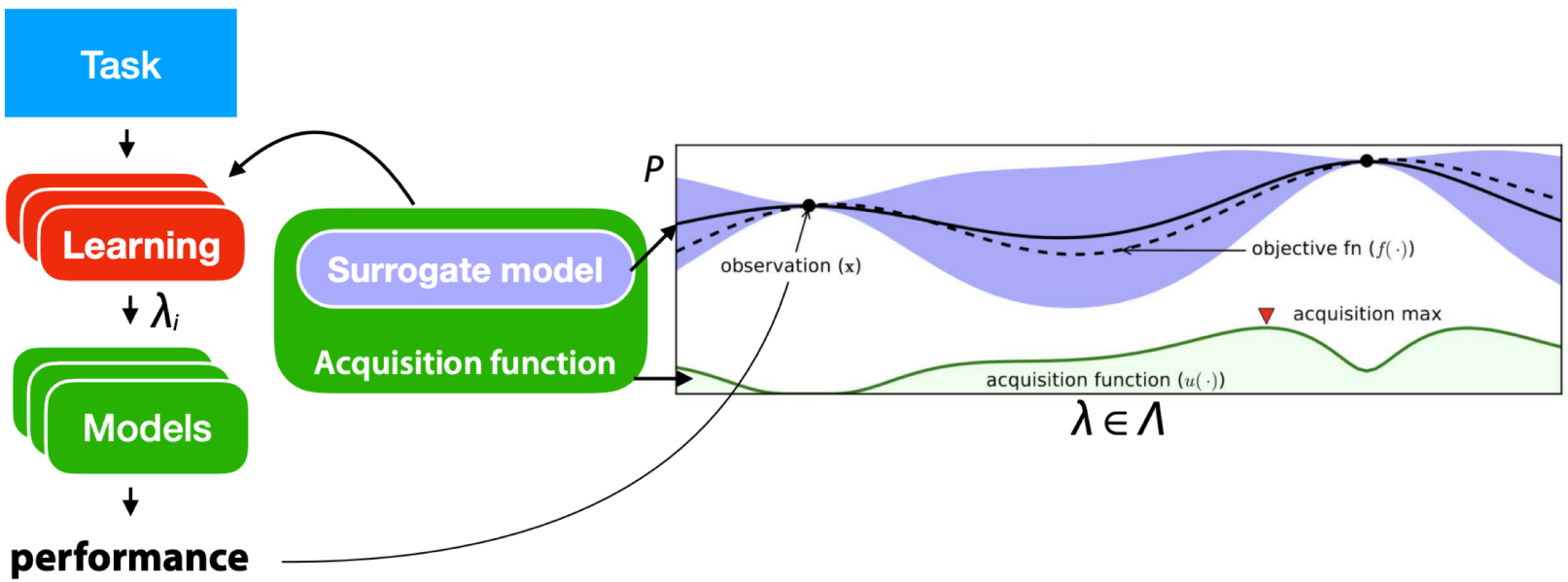
- **Task are *similar*** if observed *relative performance* of configurations is similar
- Tournament-style selection, warm-start with overall best configurations λ_{best}
- Next candidate λ_c : the one that beats current λ_{best} on similar tasks (from portfolio)



Rasmussen 2014

Bayesian optimization

- 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)?**

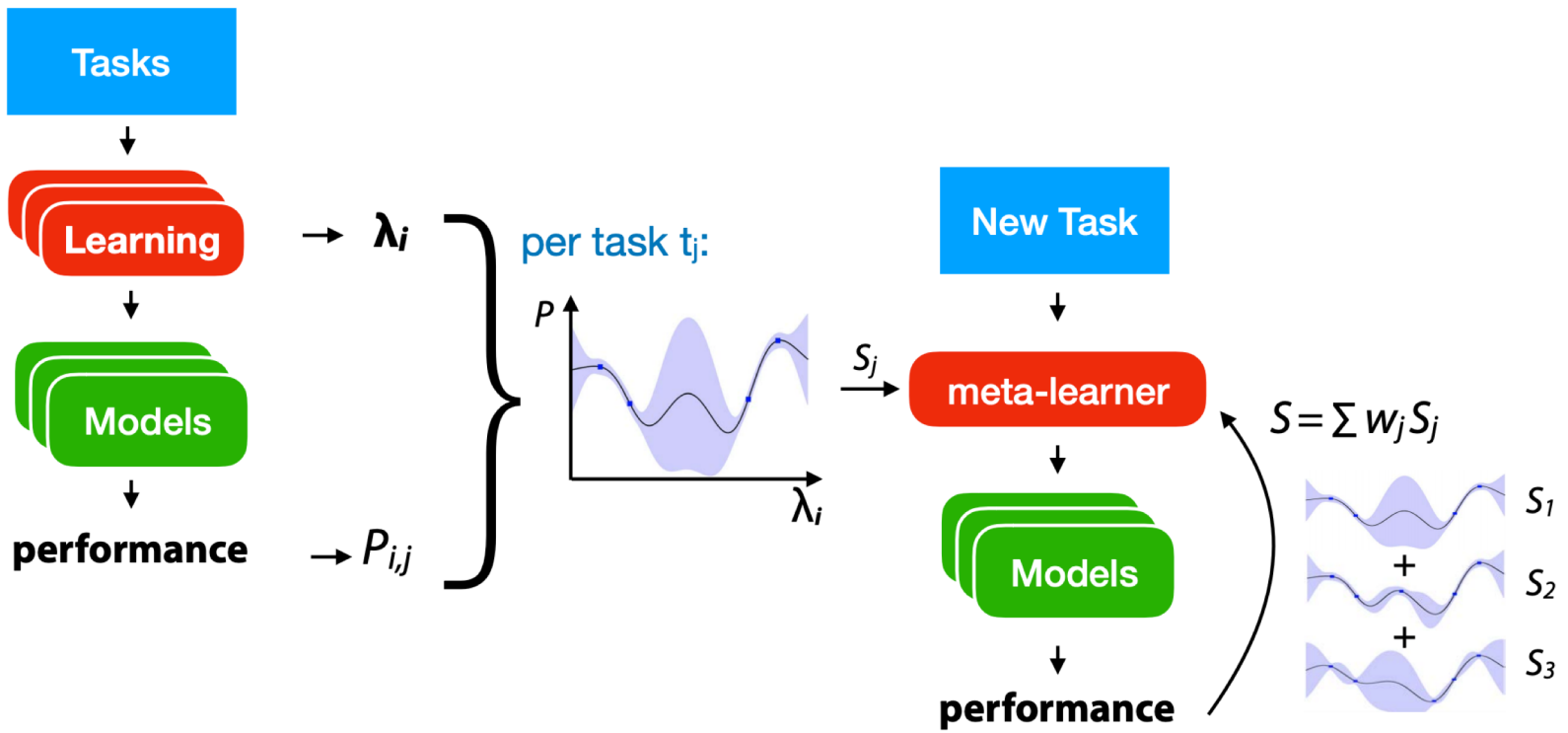


[1] Wistuba et al. 2018

[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

- Learn a joint task representation using P.
- Use a surrogate model: $s_j(\lambda_i)$
  - Trained and combined in a fnn $NN(\lambda_i)$ which learns a joint task representation that can accurately predict P_{in}.

2. Learning from Task Prope

- Each task $t_j \in T$, is represented with a ve
  - $m(t_j) = (m_{j1}, \ldots, m_{jK})$

Name	Formula	Rationale	Variants
Nr instances	n	Speed, Scalability [99]	p/n, $log(n)$, log(n/p)
Nr features	p	Curse of dimensionality [99]	$log(p)$, % categorical
Nr classes	c	Complexity, imbalance [99]	ratio min/maj class
Nr missing values	m	Imputation effects [70]	% missing
Nr outliers	o	Data noisiness [140]	o/n
Skewness	$\frac{E(X-\mu_X)^3}{\sigma_X^3}$	Feature normality [99]	min,max,μ,σ,q_1, q_3
Kurtosis	$\frac{E(X-\mu_X)^4}{\sigma_X^4}$	Feature normality [99]	min,max,μ,σ,q_1, q_3
Correlation	$\rho_{X_1X_2}$	Feature interdependence [99]	min,max,μ,σ,ρ_{XY} [157]
Covariance	$cov_{X_1X_2}$	Feature interdependence [99]	min,max,μ,σ,cov_{XY}
Concentration	$\tau_{X_1X_2}$	Feature interdependence [72]	min,max,μ,σ,τ_{XY}
Sparsity	sparsity(X)	Degree of discreteness [142]	min,max,μ,σ
Gravity	gravity(X)	Inter-class dispersion [5]	
ANOVA p-value	$p_{val_{X_1X_2}}$	Feature redundancy [70]	$p_{val_{XY}}$ [157]
Coeff. of variation	$\frac{\sigma_Y}{\mu_Y}$	Variation in target [157]	
PCA ρ_{λ_1}	$\sqrt{\frac{\lambda_1}{1+\lambda_1}}$	Variance in first PC [99]	$\frac{\lambda_1}{\sum_i^i \lambda_i}$ [99]
PCA skewness		Skewness of first PC [48]	PCA kurtosis [48]

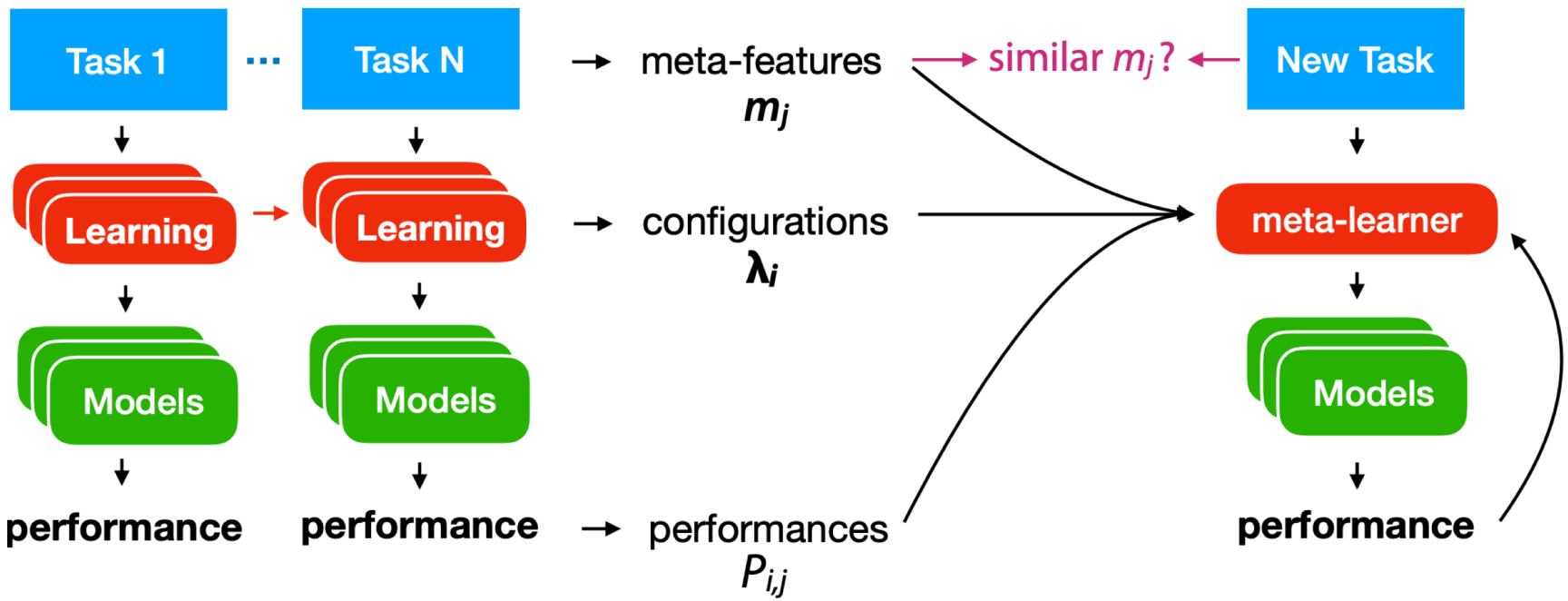
- **Number of** instances, features, classes, missing values, outliers,…
- **Statistical:** skewness, kurtosis, correlation, covariance, sparsity, variance,…
- **Information-theoretic**: class entropy, mutual information, noise-signal ratio,…
- **Model-based**: properties of simple models trained on the task
- **Landmarkers**: performance of fast algorithms trained on the task
- Domain specific task properties

			gini
Landmarker(Tree)	$P(\theta_{Tree}, t_j)$	Data separability [115]	Stump,RandomTree
Landmarker(Lin)	$P(\theta_{Lin}, t_j)$	Linear separability [115]	Lin.Disciminant
Landmarker(NB)	$P(\theta_{NB}, t_j)$	Feature independence [115]	More models [14, 88]
Relative LM	$P_{a,j} - P_{b,j}$	Probing performance [53]	
Subsample LM	$P(\theta_i, t_j, s_t)$	Probing performance [159]	

2. Reason about model performance across tasks

Meta-features: measurable properties of the tasks

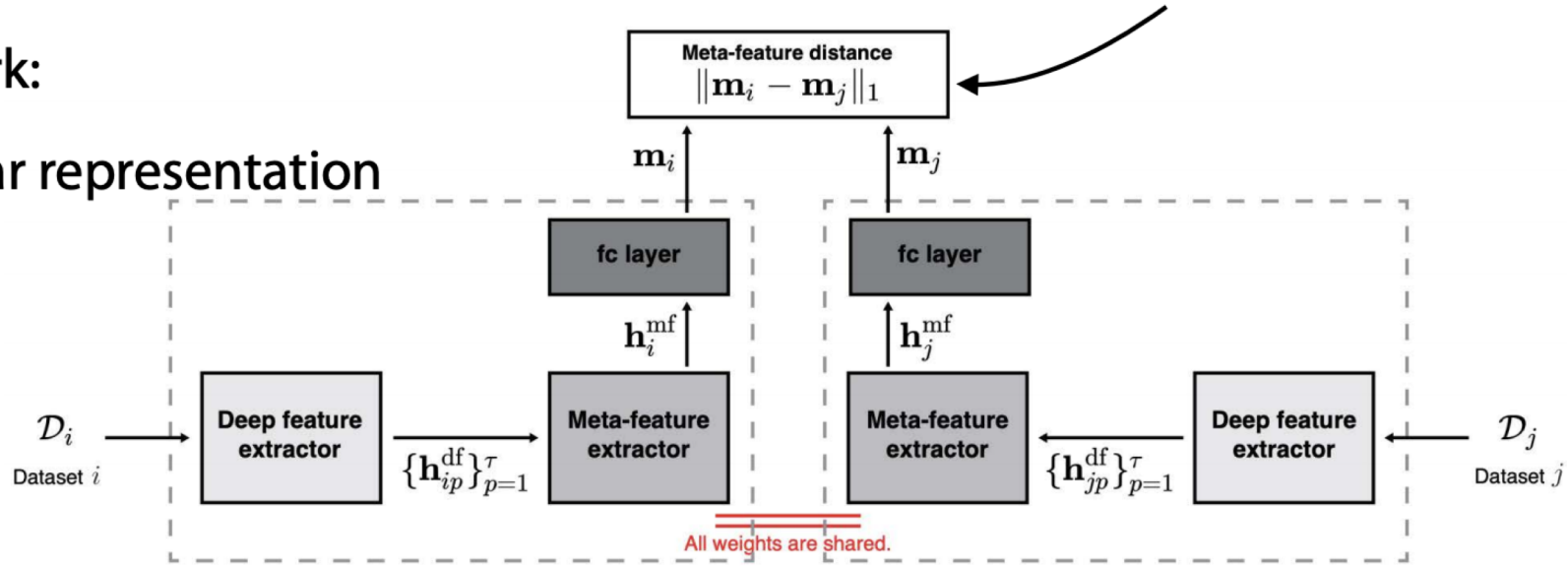
(number of instances and features, class imbalance, feature skewness,…)



Learning Meta-Features

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

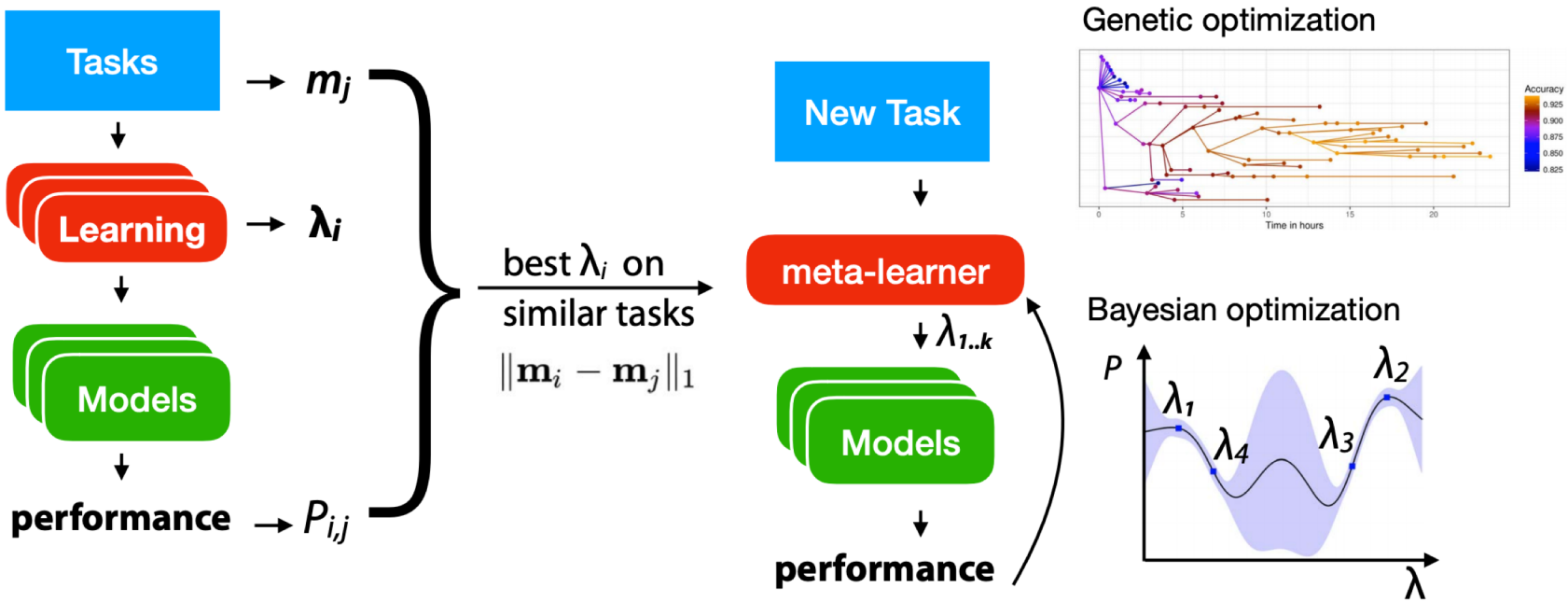


[1] Gomes et al. 2012, Reif et al. 2012

[2] Feurer et al. 2015

Warm-starting from similar tasks

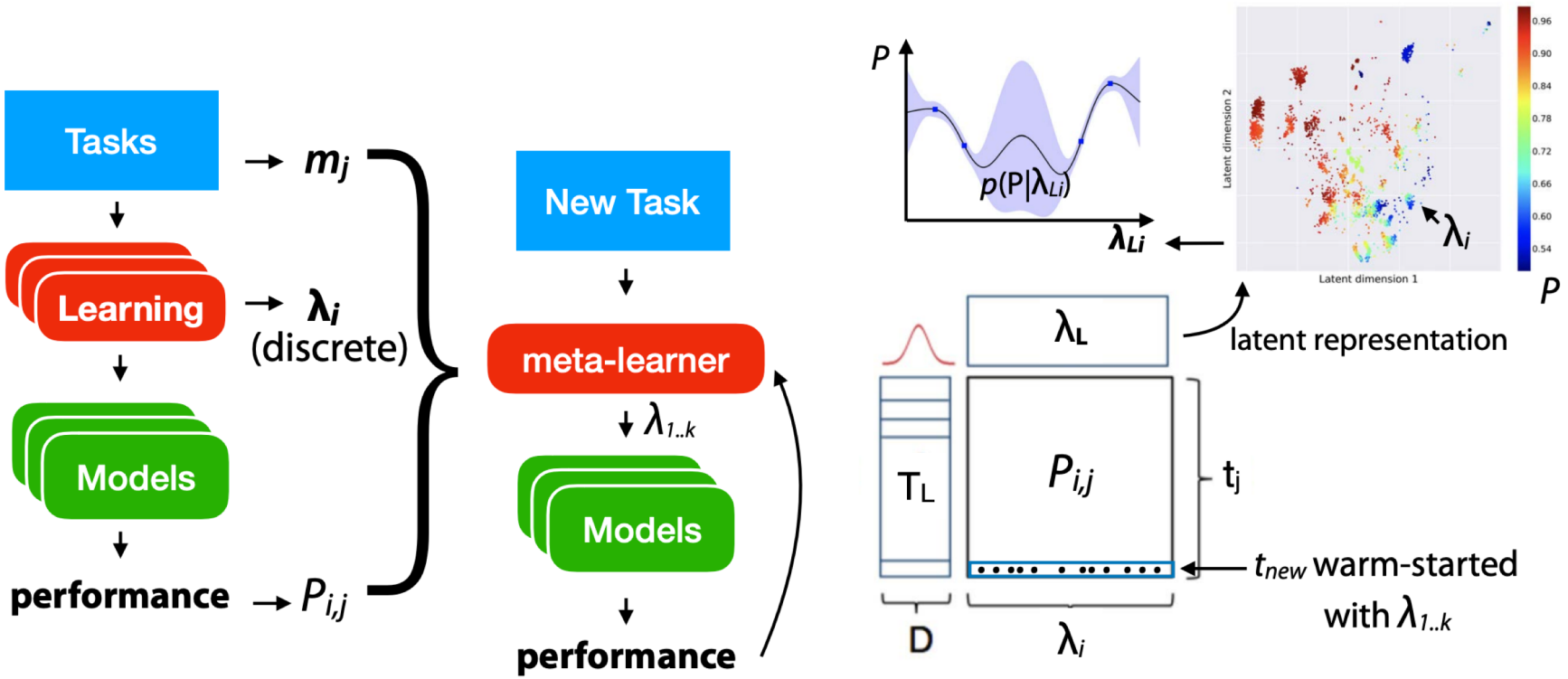
- Find k most similar tasks, warm-start search with best θ_i
  - Genetic hyperparameter search [1]
  - Auto-sklearn: Bayesian optimization (SMAC) [2]
    - Scales well to high-dimensional configuration spaces



Fusi et al. 2017

Warm-starting from similar tasks

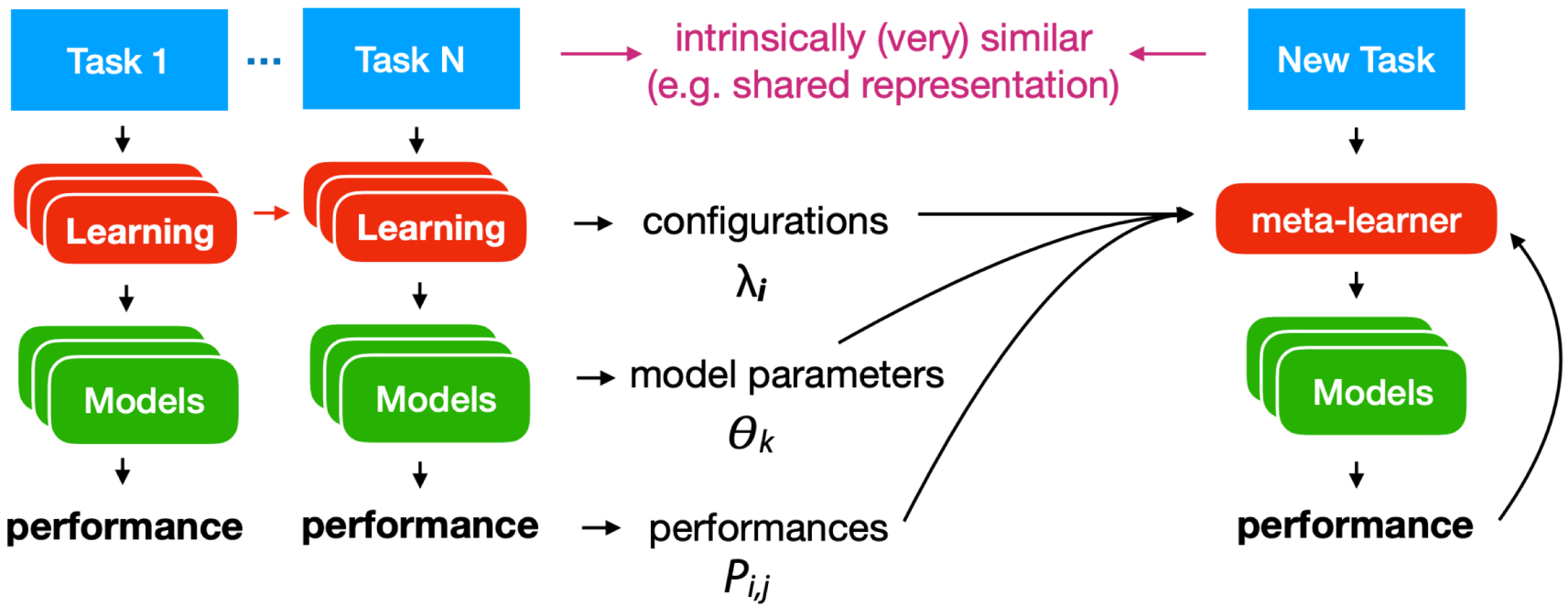
- Collaborative filtering: configurations λ_i are `rated' by tasks t_j
  - Probabilistic matrix factorization
    - Learns a latent representation for tasks and configurations
    - Returns probabilistic predictions for Bayesian optimization
    - Use meta-features to warm-start on new task



3. Learning from trained models

Models trained on similar tasks

(model parameters, features,…)



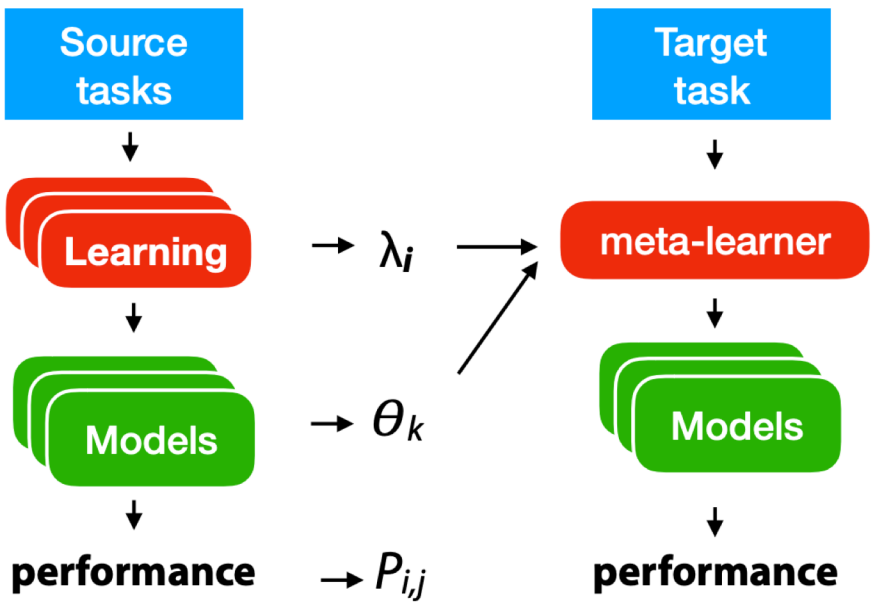
[1] Thrun and Pratt 1998

[2] Niculescu-Mizil and Caruana 2005

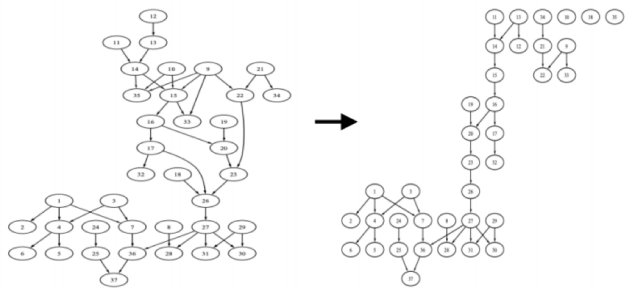
[3] Taylor and Stone 2009

Transfer Learning

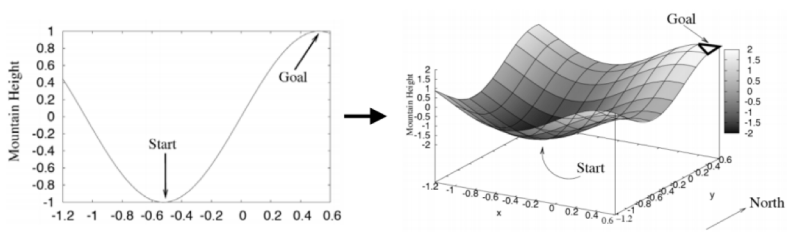
- Select source tasks, transfer trained models to similar target task [1]
- Use as starting point for tuning, or *freeze* certain aspects (e.g. structure)
  - Bayesian networks: start structure search from prior model [2]
  - Reinforcement learning: start policy search from prior policy [3]



Bayesian Network transfer



Reinforcement learning: 2D to 3D mountain car

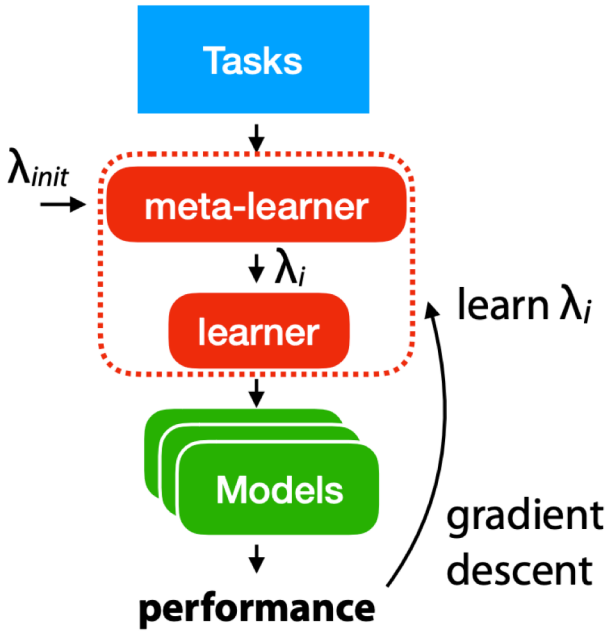


[1] Bengio et al. 1995

[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

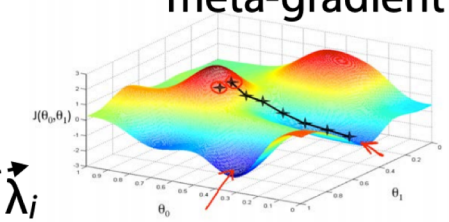
Runarsson and Jonsson

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

learning rate

weights λ_i

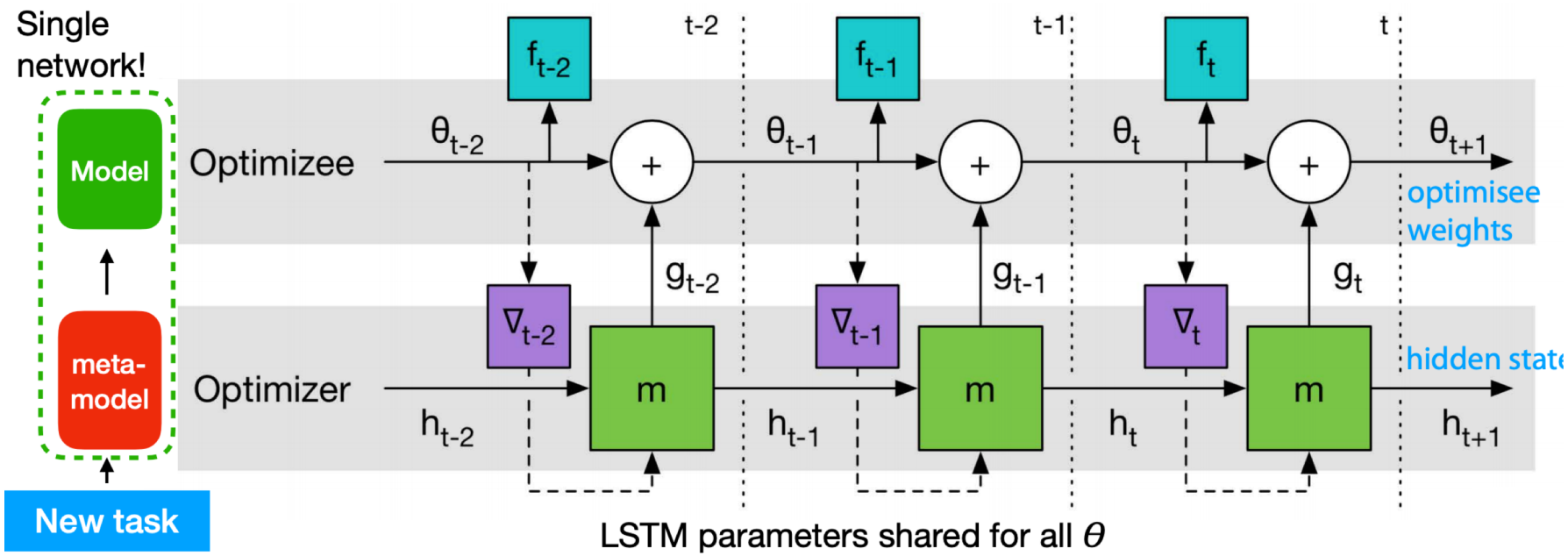
meta-gradient



[1] Hochreiter 2001
[2] Andrychowicz et al. 2016

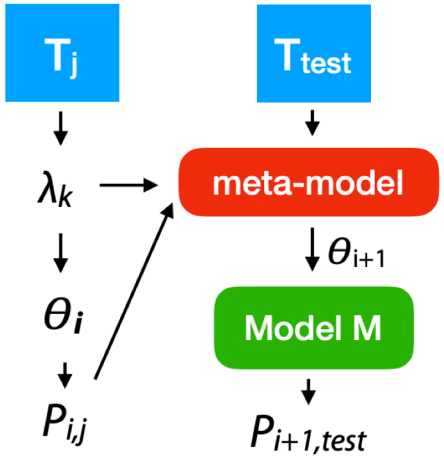
Learning to learn gradient descent by gradient descent

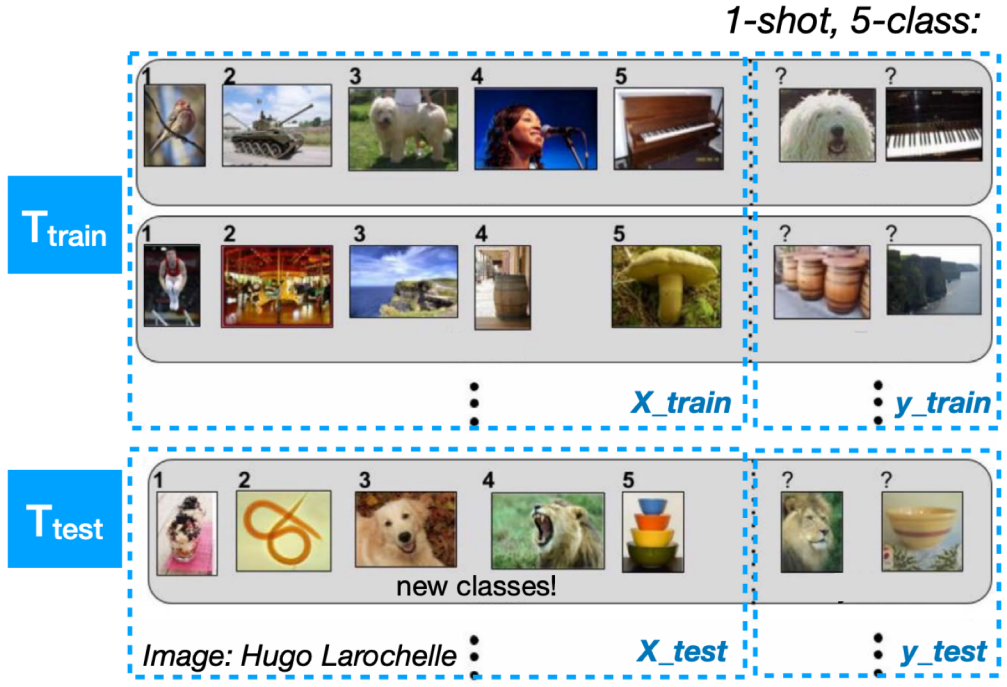
- Replace backprop with a recurrent neural net (LSTM)[1], not so scalable
- Use a coordinatewise LSTM [m] for scalability/flexibility (cfr. ADAM, RMSprop) [2]
  - Optimizee: receives weight update g_t from optimizer
  - Optimizer: receives gradient estimate ∇_t from optimizee
    - Learns how to do gradient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



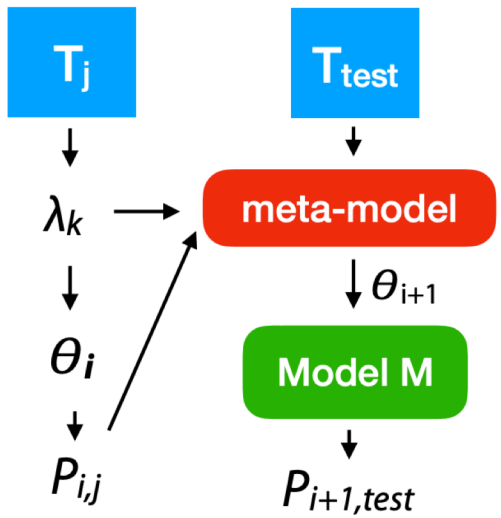
Few-Shot Learning

- A particular challenging meta-learning problem is to train an accurate deep learning model using only a few training examples, given prior_experience with very similar tasks for which we have large training sets available.
- A particular example is 'K-shot N-way' classification, in which we are given many examples of certain classes, and want to learn a classifier l_n able to classify N new classes using only K examples of each.





Few-shot learning: approaches



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

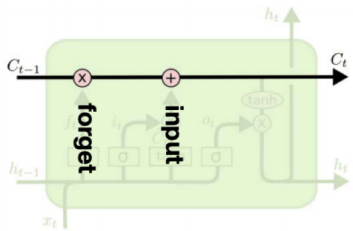
- Existing algorithm as meta-learner:
  - LSTM + gradient descent — *Ravi and Larochelle 2017*
  - Learn θ_{init} + gradient descent — *Finn et al. 2017*
  - kNN-like: Memory + similarity — *Vinyals et al. 2016*
  - Learn embedding + classifier — *Snell et al. 2017*
  - …
- Black-box meta-learner
  -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

Ravi and Larochelle 2017

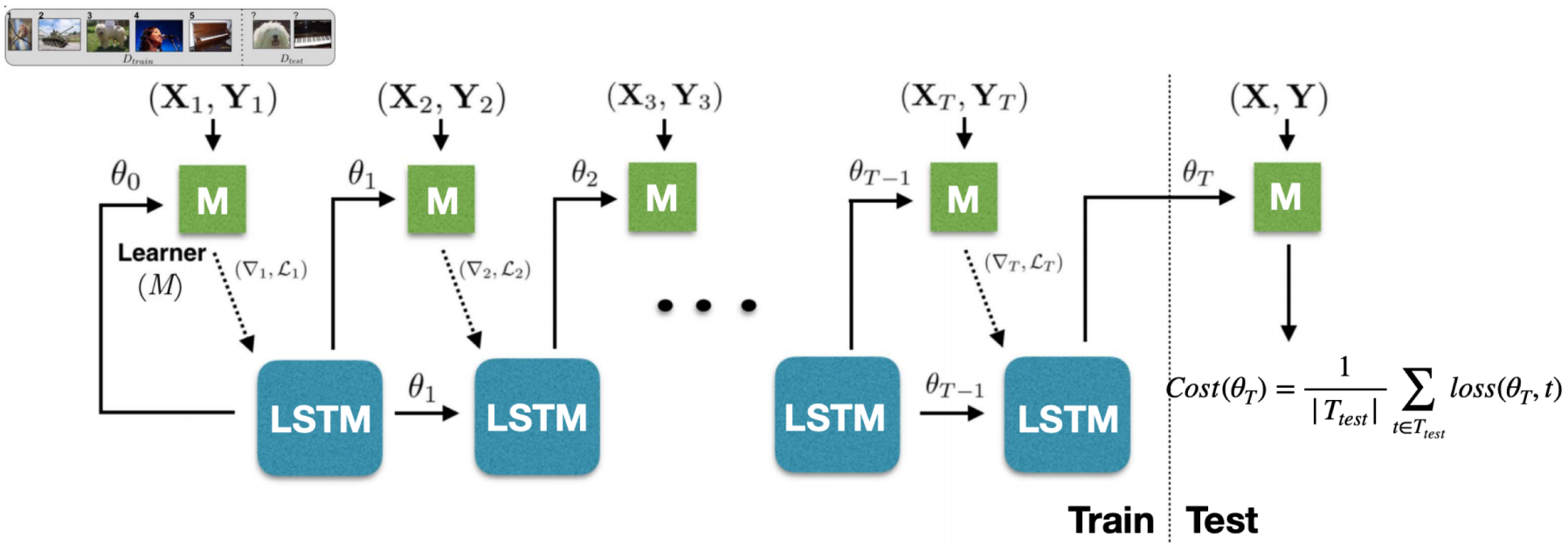
LSTM meta-learner + gradient descent

- 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

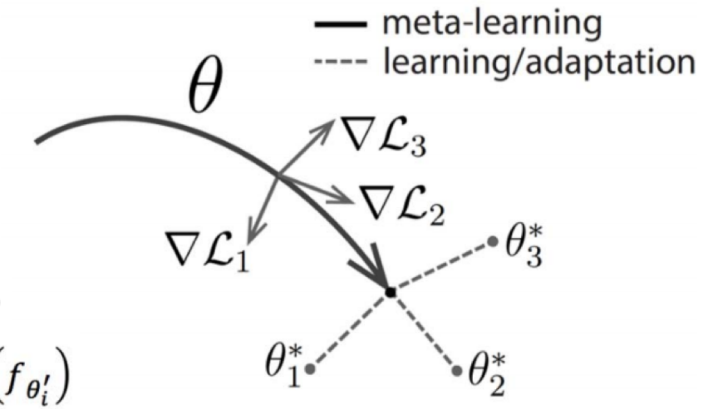


[1] Finn et al. 2017
[2] Finn et al. 2018
[3] Nichol et al. 2018

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

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



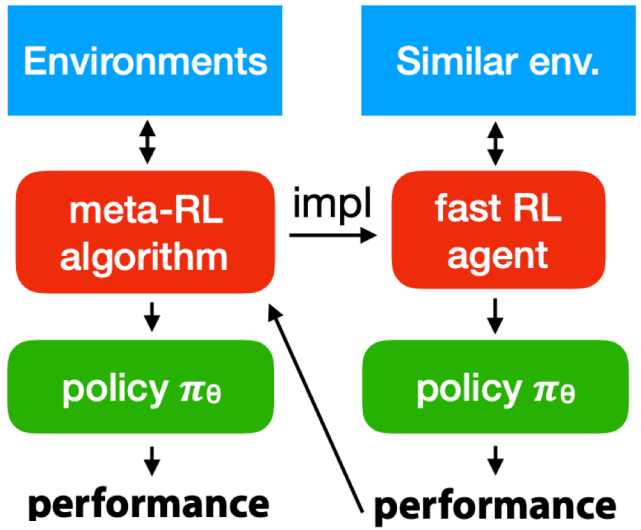
- More resilient to overfitting
- Generalizes better than LSTM approaches
- *Universality*: no theoretical downsides in terms of expressivity when compared to alternative meta-learning models.
- REPTILE: do SGD for k steps in one task, only then update initialization weights[3]

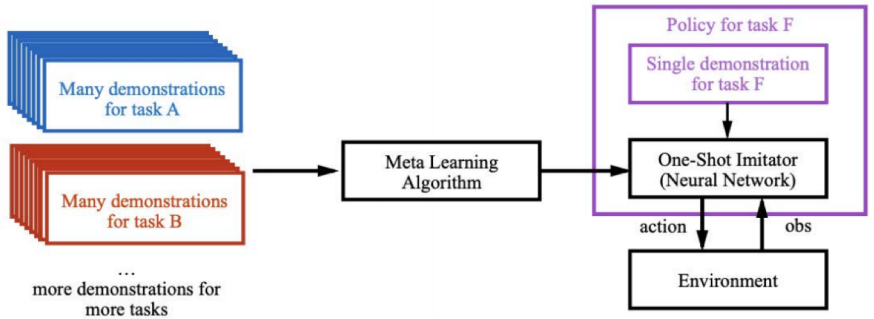
Questions?

[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