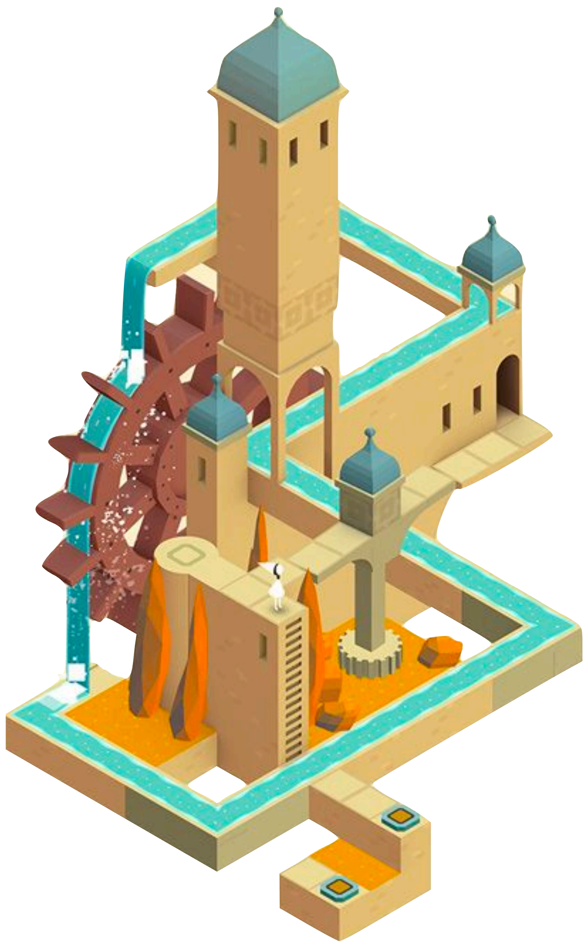


Automatic Machine Learning

A Tutorial

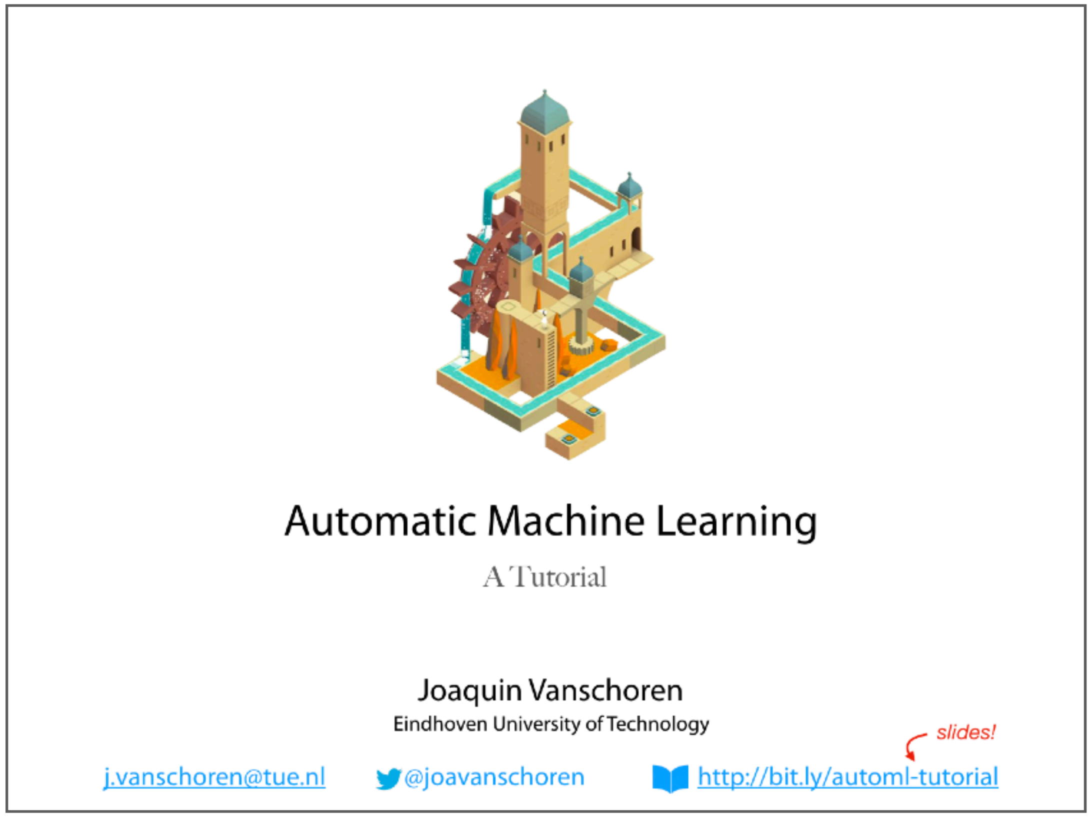
Joaquin Vanschoren

Eindhoven University of Technology

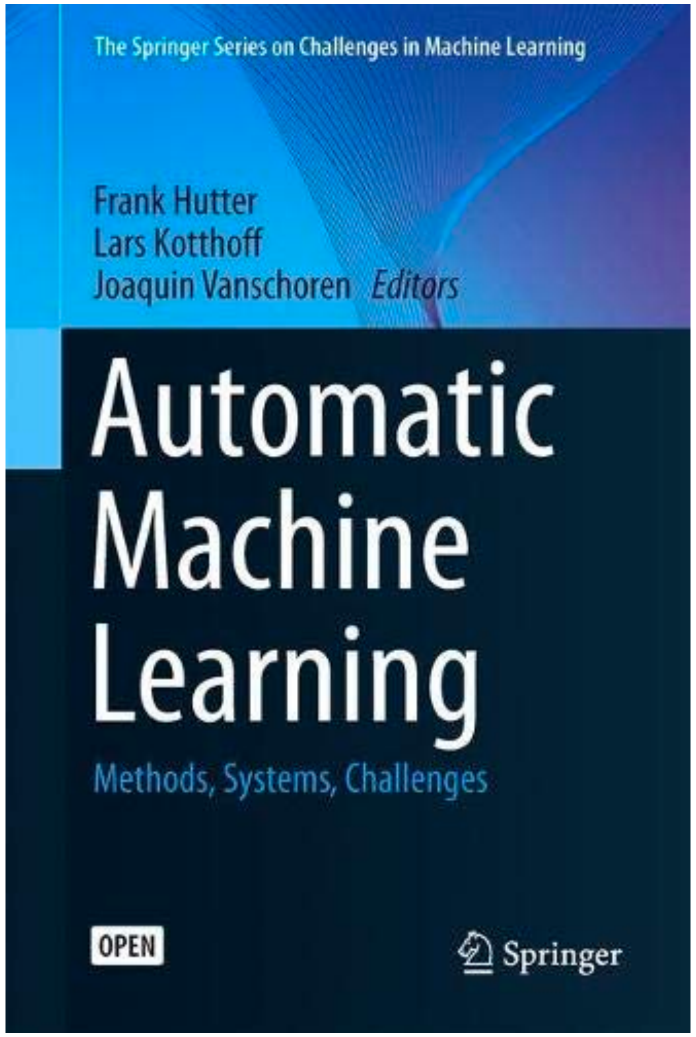
slides!

j.vanschoren@tue.nl @joavanschoren http://bit.ly/automl-course

Slides / Book



Slides
http://bit.ly/automl-course

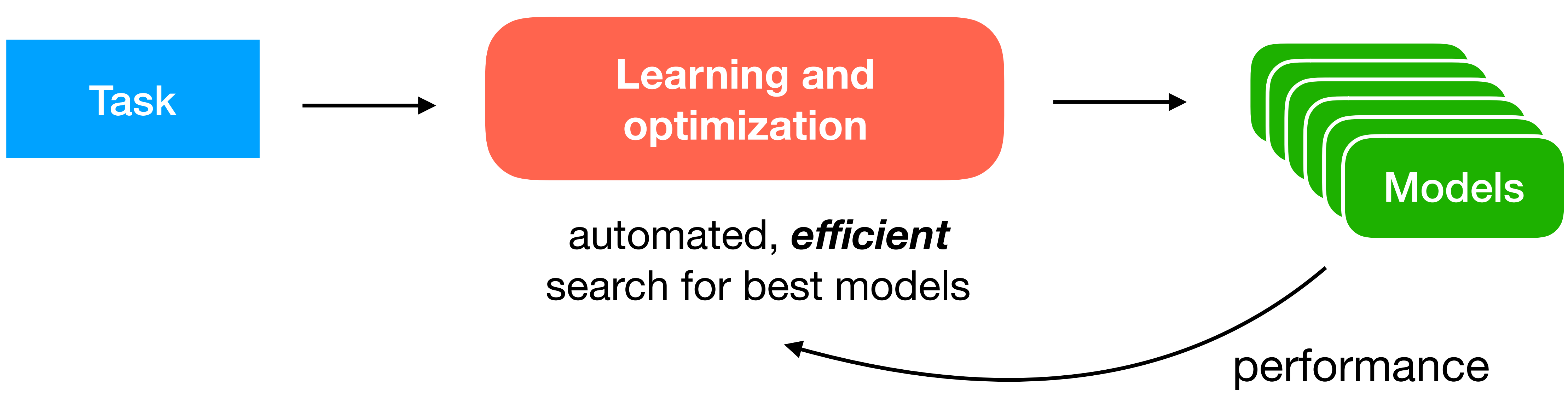


Open access book
(free PDF)
www.automl.org/book
www.amazon.de/dp/3030053172

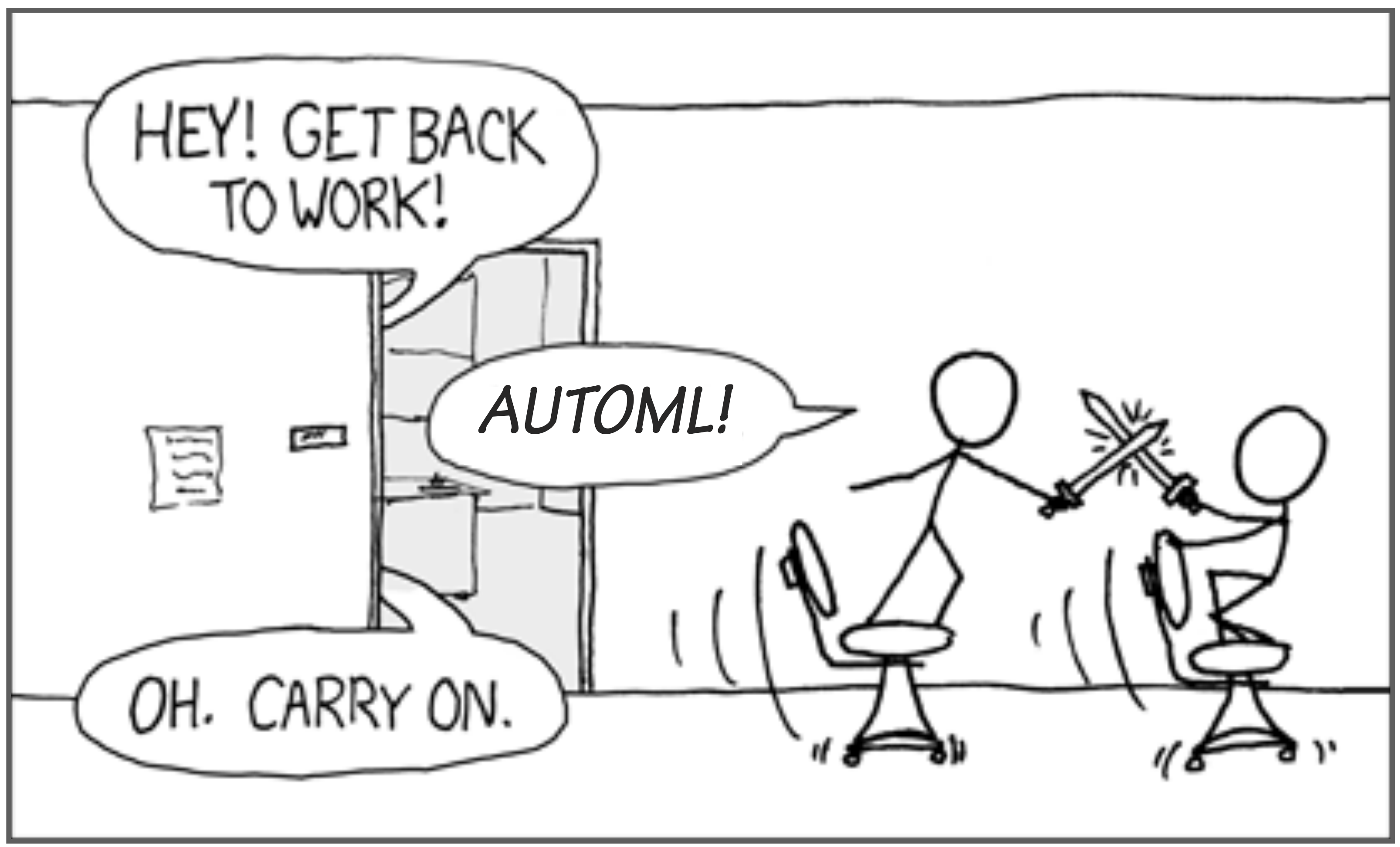
Automated machine learning

Manual machine learning

AutoML: build models in a *data-driven, intelligent, purposeful* way



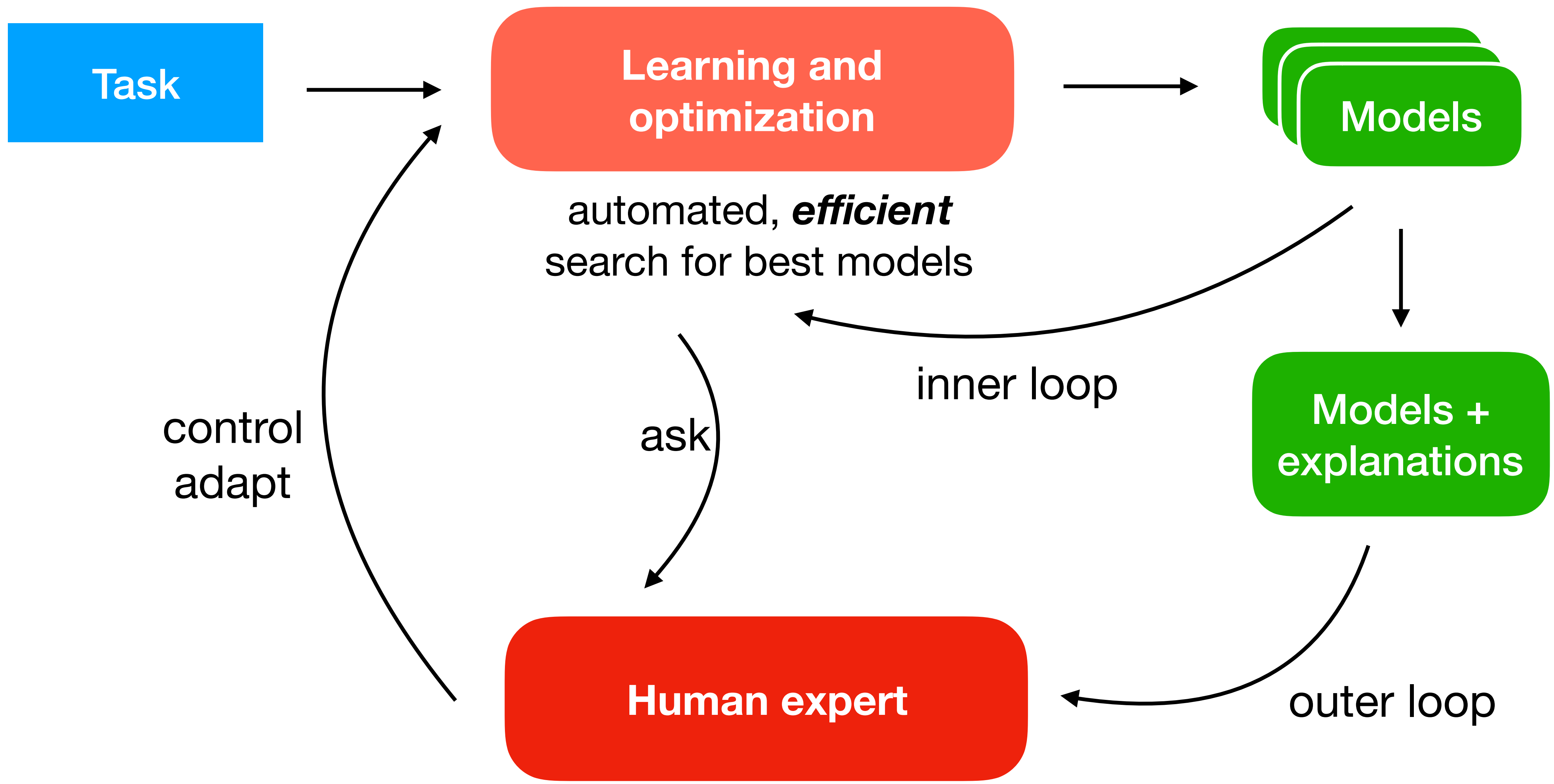
THE DATA SCIENTIST'S #1 EXCUSE FOR LEGITIMATELY SLACKING OFF: "THE AUTOML TOOL IS OPTIMIZING MY MODELS!"



Adapted from XKCD (Randall Munroe)

Semi-Automated machine learning

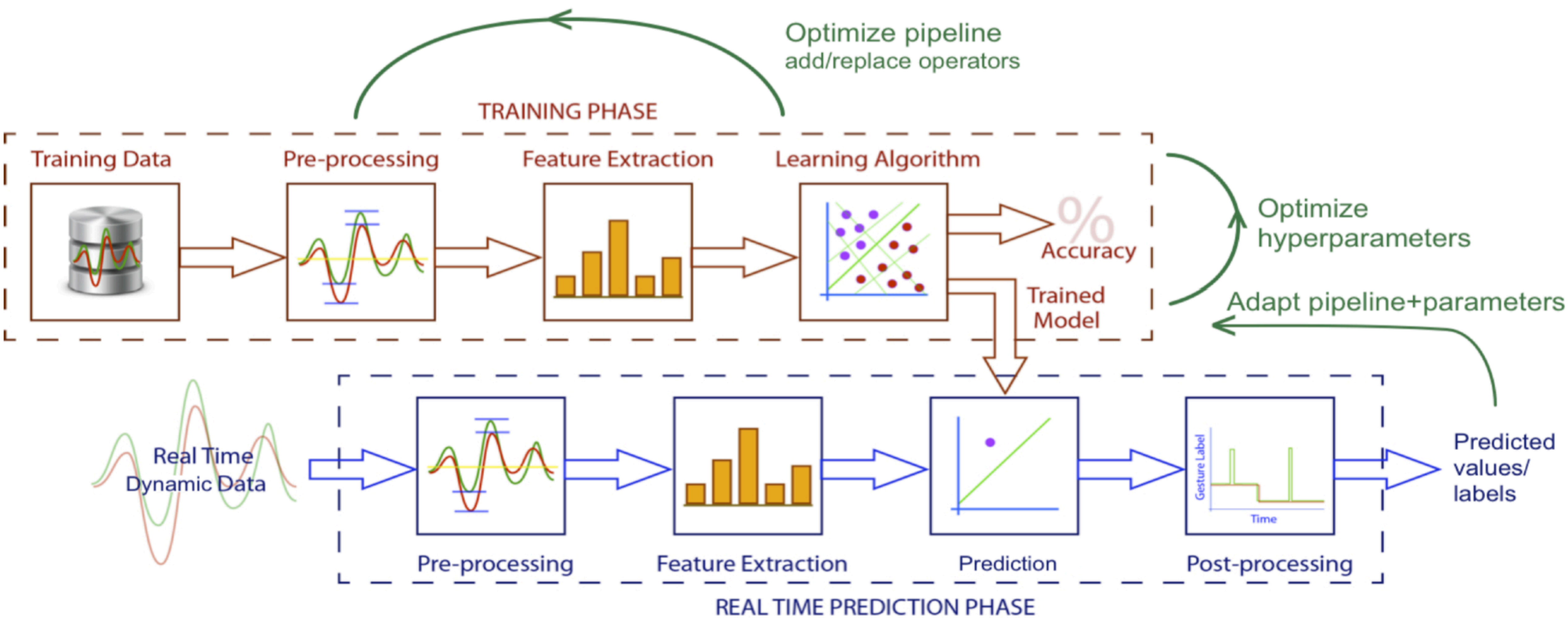
Human-in-the-loop



AutoML automates some aspects that require intelligence, but not all
Domain knowledge and human expertise remain crucial

Doing machine learning requires a lot of expertise

Pipeline synthesis

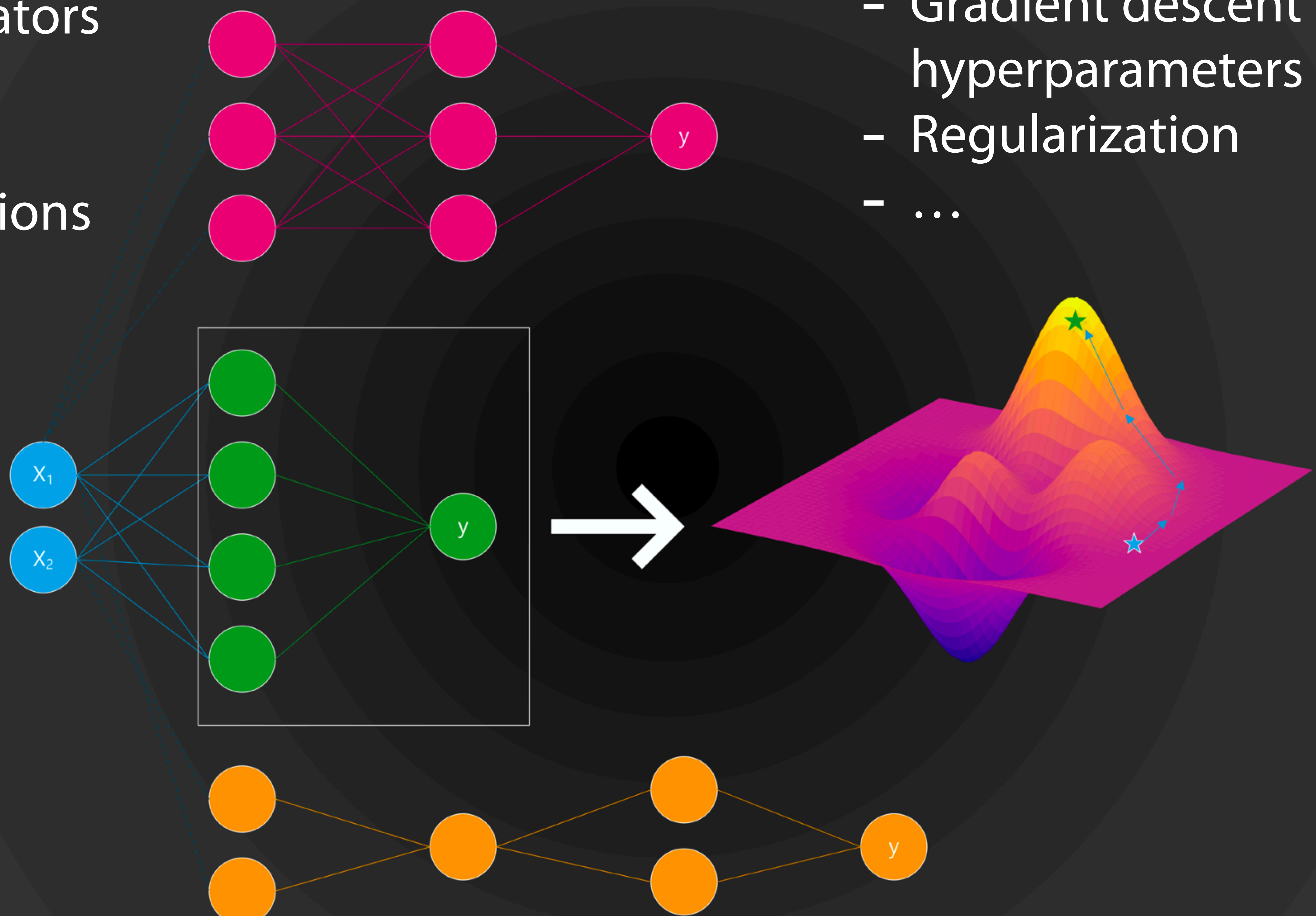


- Excellent tools (e.g. scikit-learn), but little guidance
- Cleaning, preprocessing, feature selection/engineering features, model selection, hyperparameter tuning, adapting to concept drift,…

Doing machine learning requires a lot of expertise

Neural Architecture Search (NAS)

- Type of operators
- Size of layers
- Filter sizes
- Skip connections
- …

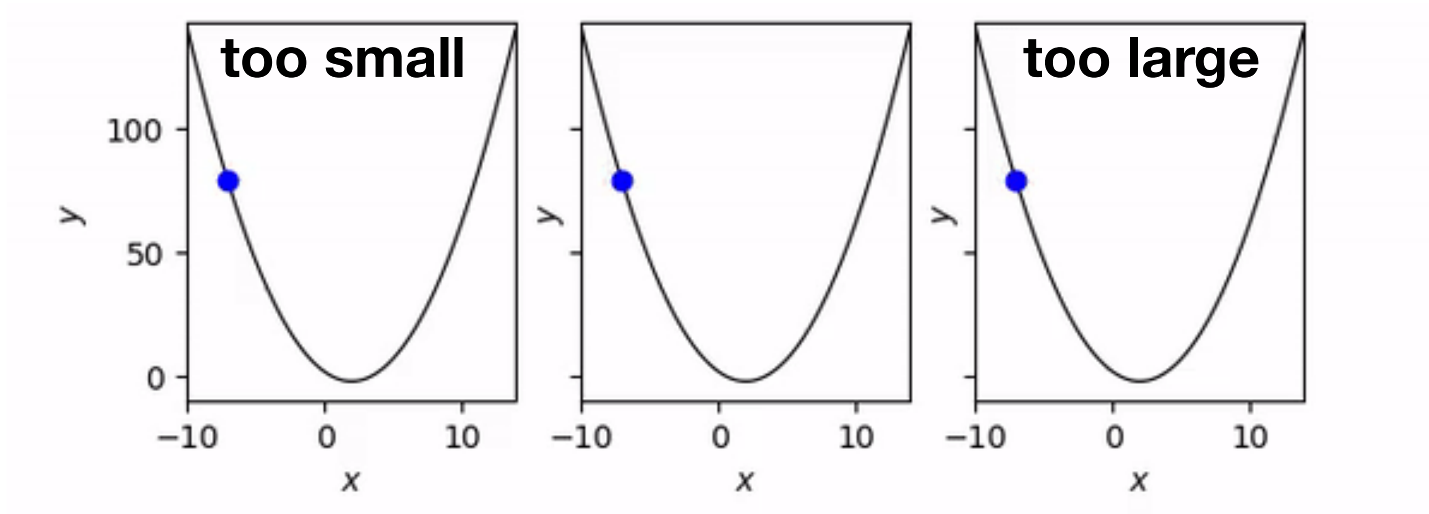


- Gradient descent hyperparameters
- Regularization
- …

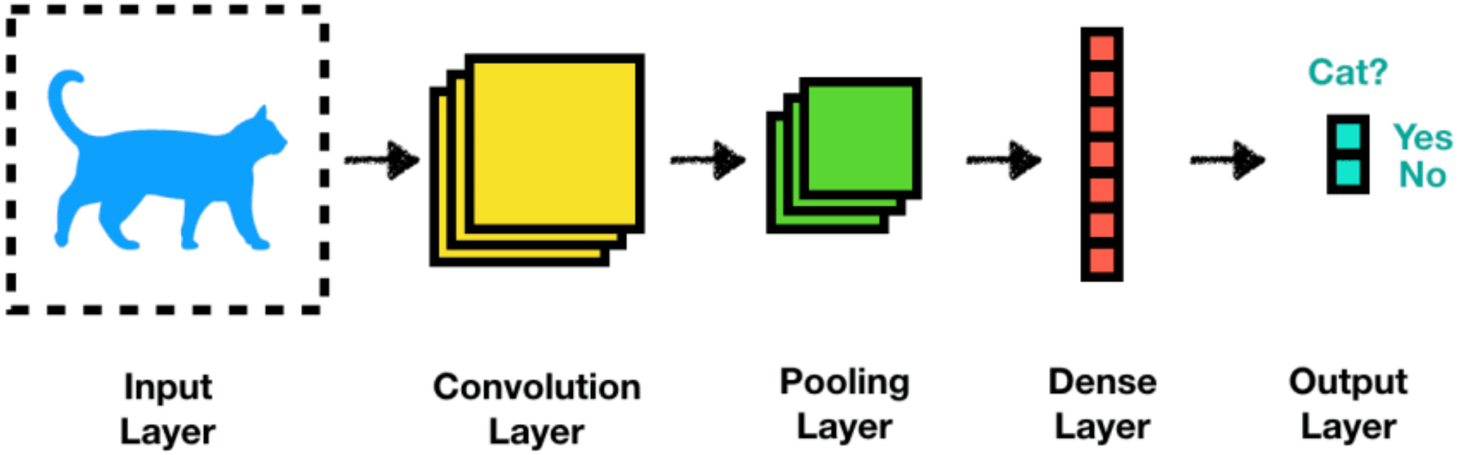
Hyperparameters

Every design decision made by the *user (architecture, operators, tuning,…)*

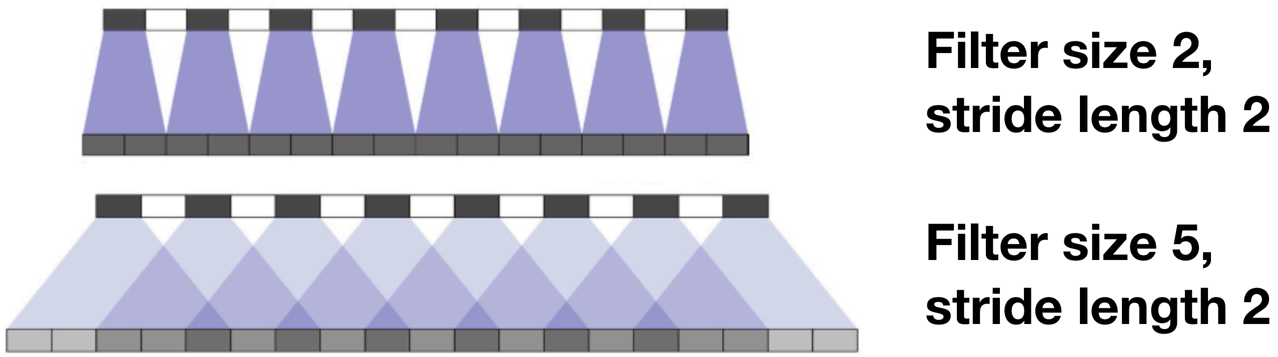
- Numeric
  - e.g. learning rate



- Categorical
  - e.g. layer type



- Conditional
  - ConvLayer -> filter size, stride length,…

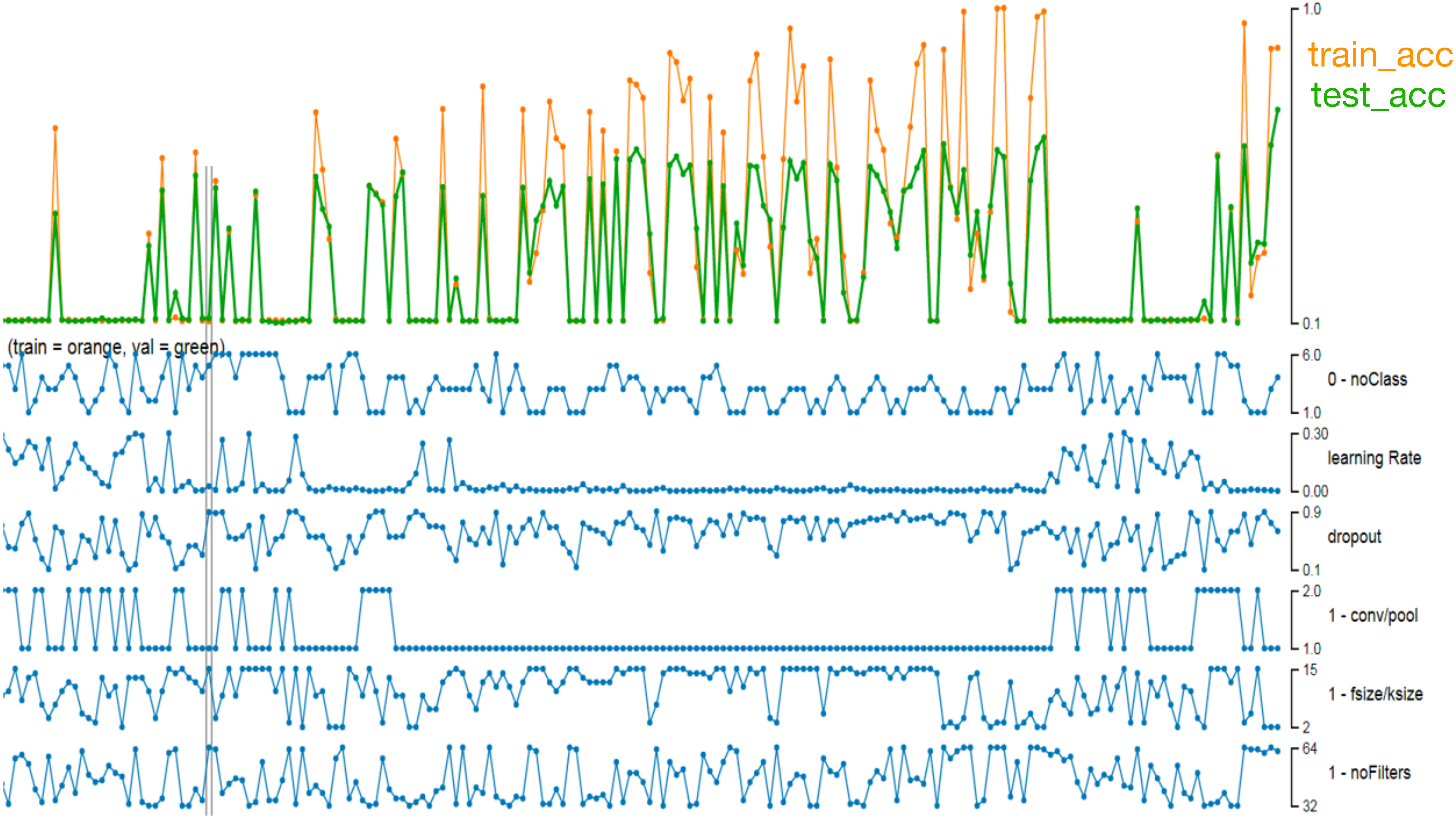


Hyperparameters

Some are very sensitive, others not. Many interact.

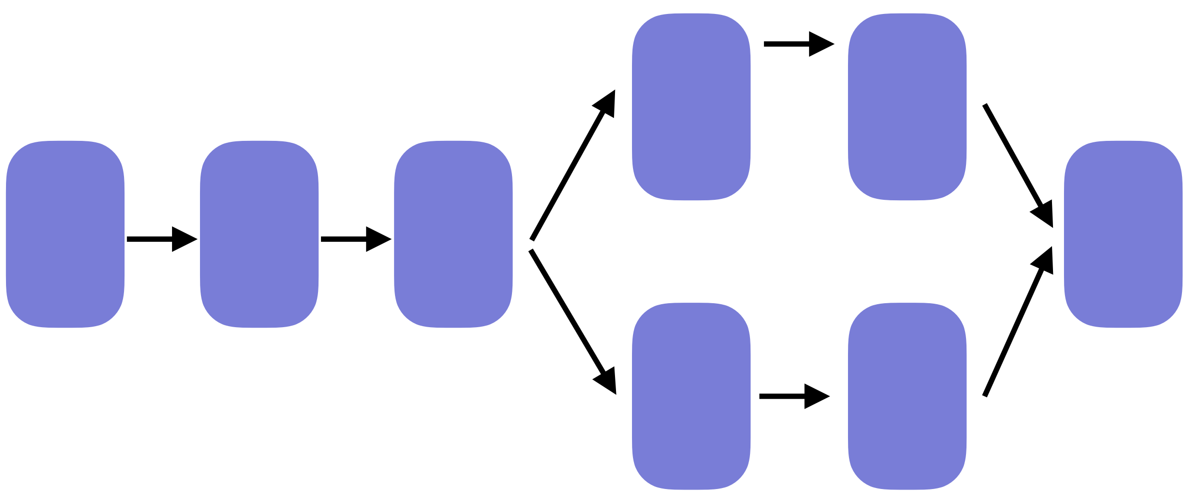
What is the right strategy to find the right ***configuration***?

Can we do better than random search?



AutoML: subproblems

- **Architecture search**: *represent* and *search* all pipelines or neural nets
  - Pipeline operators, neural layers, interconnections,…
  - Defines the search space

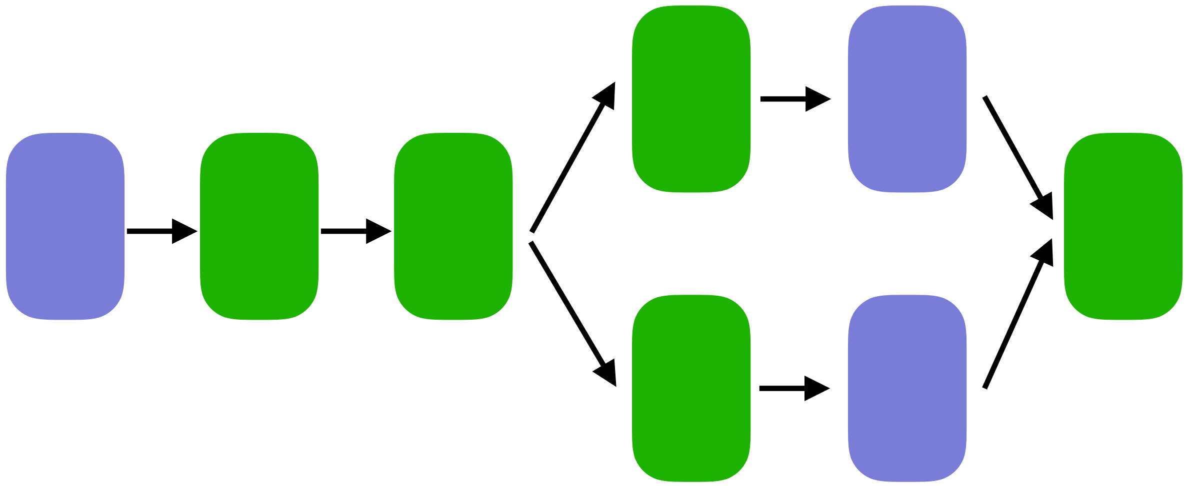


```
make_pipeline(
  OneHotEncoder(),
  Imputer(),
  StandardScaler(),
  SVC())
```

```
model.add(Conv2D(32, (3, 3))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3))
```

AutoML: subproblems

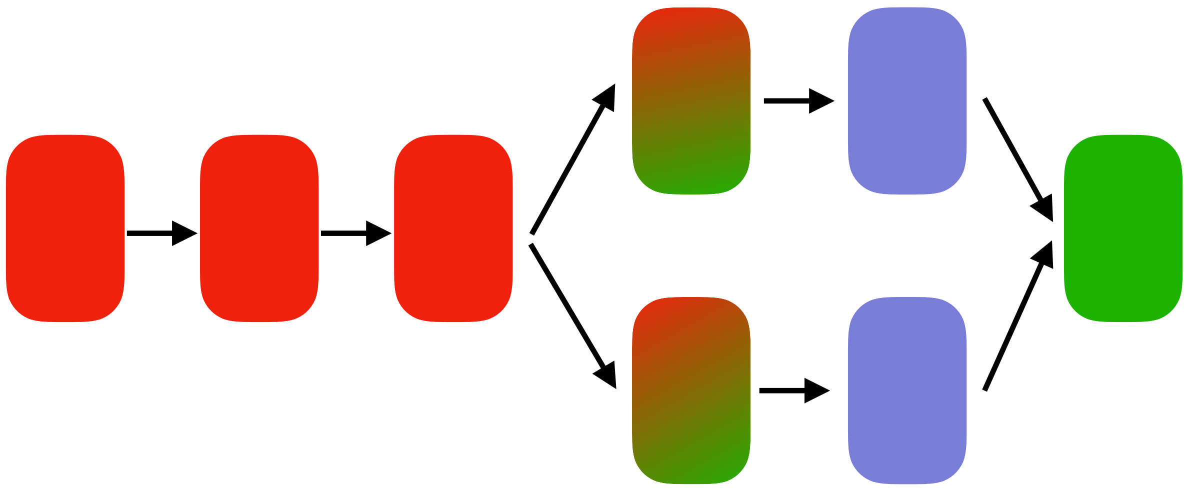
- **Architecture search**: *represent* and search all possible architectures
- **Hyperparameter optimization**:
    - Which hyperparameters are important? How to optimize them?
    - What is the (multi-)objective function?



```
def build_model(hp){
    m.add(Dense(units=hp.Int('units', min_value=32, max_value=512, step=32)))
    m.compile(optimizer=Adam(hp.Choice('learning rate', [1e-2, 1e-3, 1e-4])))
    return model;}
tuner = RandomSearch(build_model, max_trials=5)   //or HyperBand
```

AutoML: subproblems

- **Architecture search**: *represent* and search all possible architectures
- **Hyperparameter optimization**: *optimize* remaining hyperparameters
- **Meta-learning**: how can we transfer *experience* from previous tasks?
  - Don't start from scratch (search space is too large)



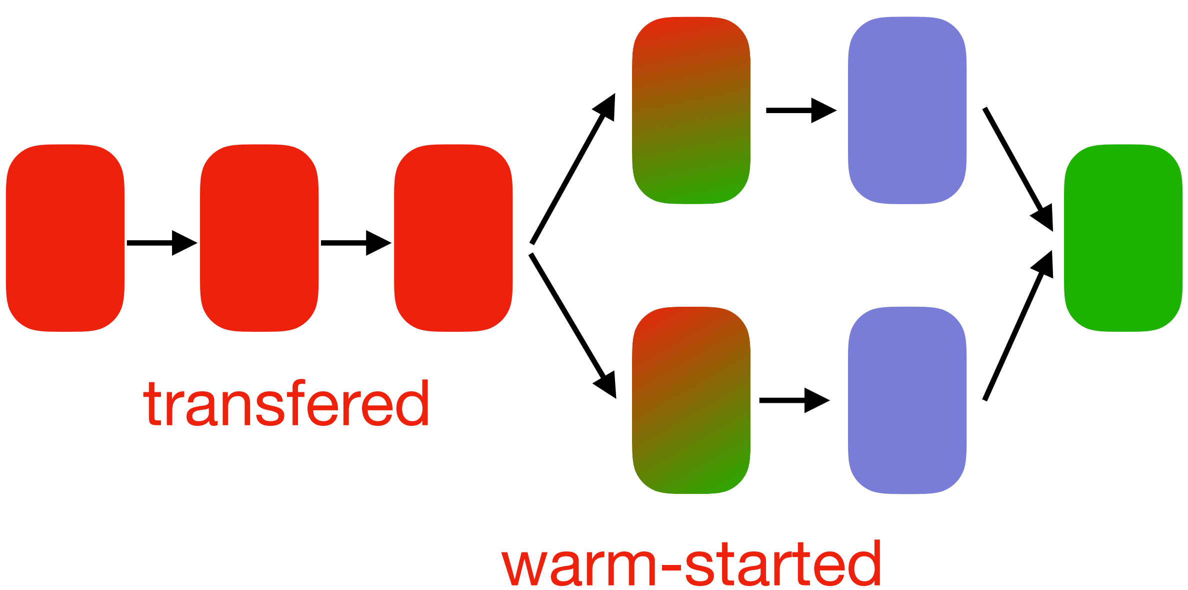
Goal:

```
model = AutoML_Learner().fit(X_train,y_train)
model2 = AutoML_Learner().fit(X2_train,y2_train)
```

meta-learn

AutoML: subproblems

- **Architecture search**: *represent* and search all possible architectures
- **Hyperparameter optimization**: *optimize* important hyperparameters
- **Meta-learning**: how can we transfer experience from previous tasks?
    - Transfer learning: reuse good architectures/configurations
    - Warm starting: start from promising architectures/configurations
    - …



AutoML: subproblems

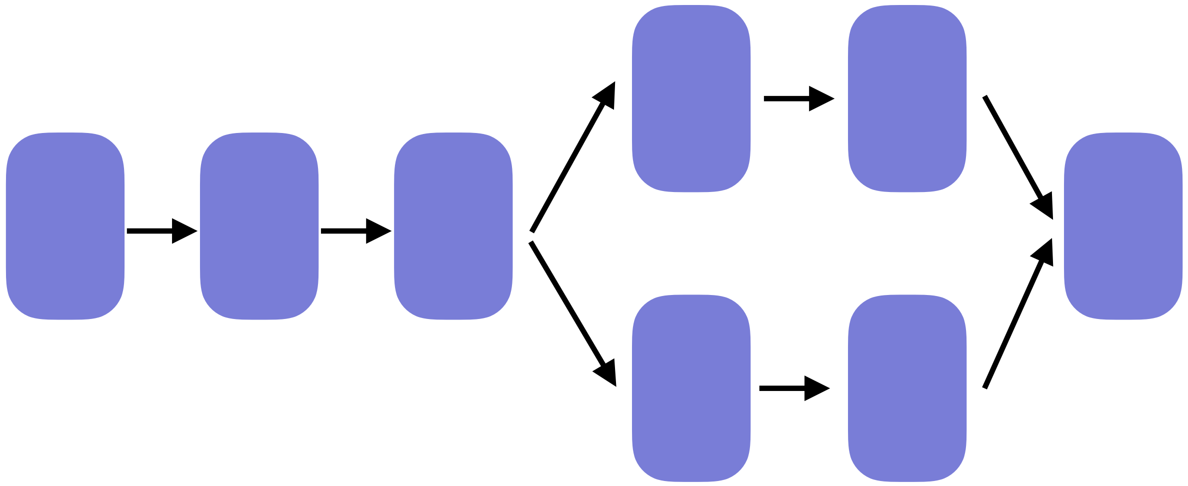
- **Architecture search**: *represent* and search all possible architectures
- **Hyperparameter optimization**: *optimize* remaining hyperparameters
- **Meta-learning**: how can we transfer experience from previous tasks?

Many combinations are possible!
They can be done *consecutively*, *simultaneously* or *interleaved*

So… yeah, this is a meta-meta-learning problem

AutoML: subproblems

- **Architecture search**: *many different approaches*
  - Parameterized (fixed) architectures
  - Reinforcement learning
  - Evolution
  - Heuristic search
  - …

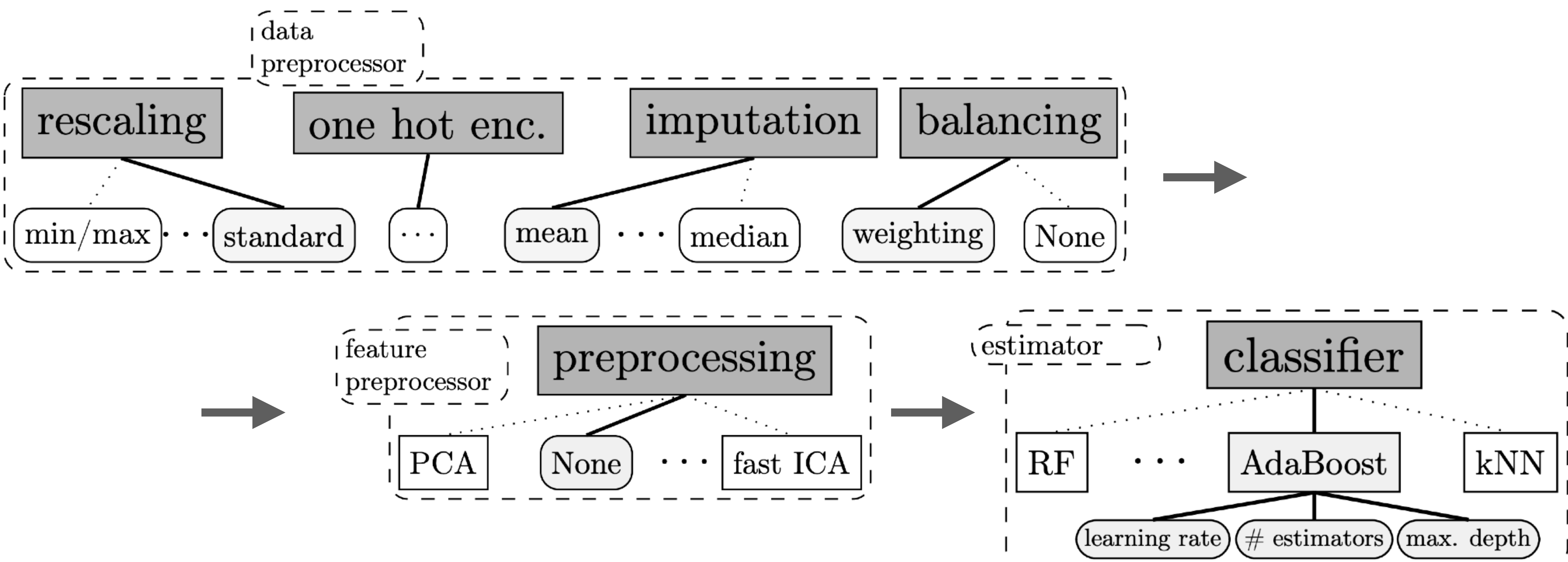


Architecture Search

Parameterized architecture

autosklearn *Feurer et al. 2015*
autoWEKA *Thornton et al. 2013*
hyperopt-sklearn *Komer et al. 2014*

- From experience: most successful pipelines have a similar structure
- Fix architecture, encode all choices as extra hyperparameters
  - *Architecture search becomes hyperparameter optimization*



+ smaller search space
- you can't learn entirely new architectures

Elsken et al. 2019

Parameterized neural architectures

Parameterized Sequential

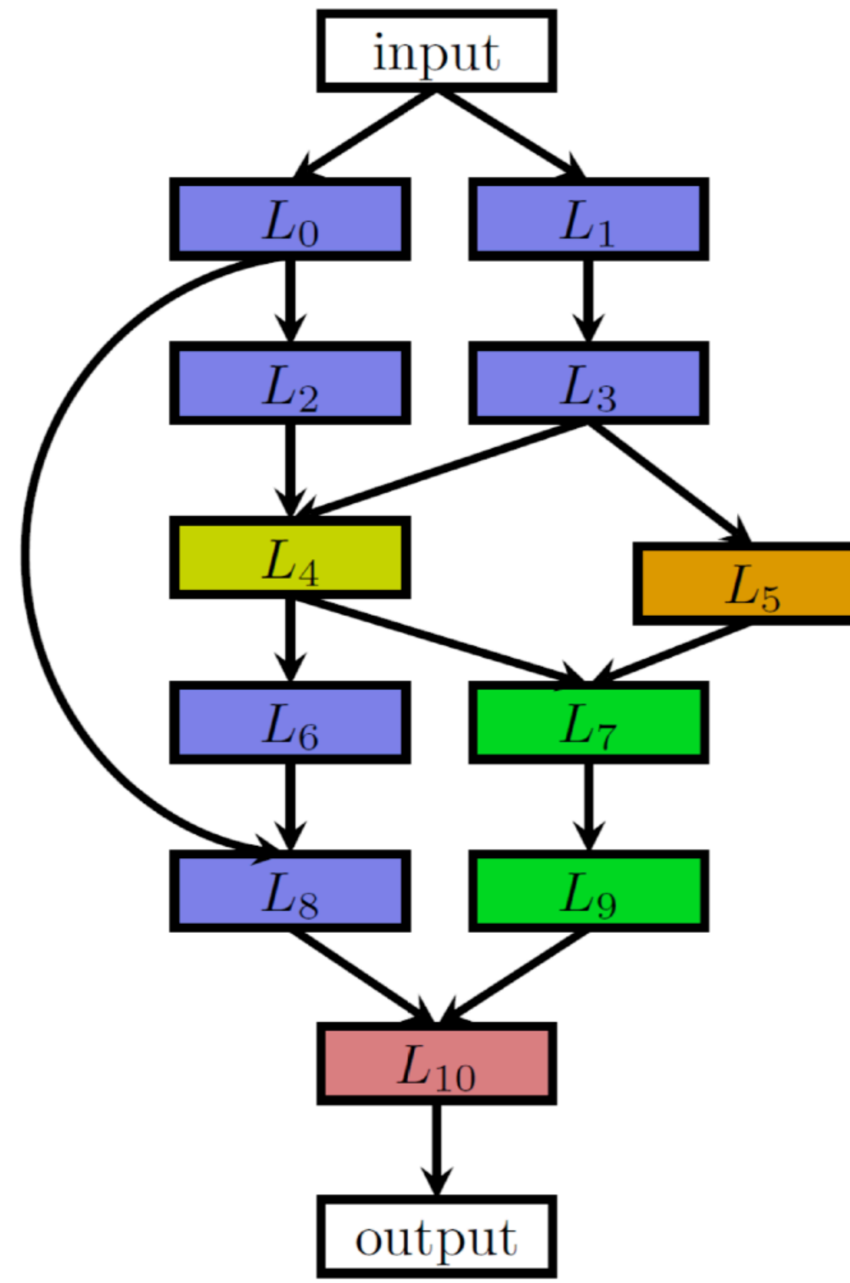
Choose:
- number of layers
- type of layers
  - dense
  - convolutional
  - max-pooling
  - …
- hyperparameters of layers

+ easier to search
- sometimes too simple

Parameterized Graph

Choose:
- branching
- joins
- skip connections
- types of layers
- hyperparameters of layers

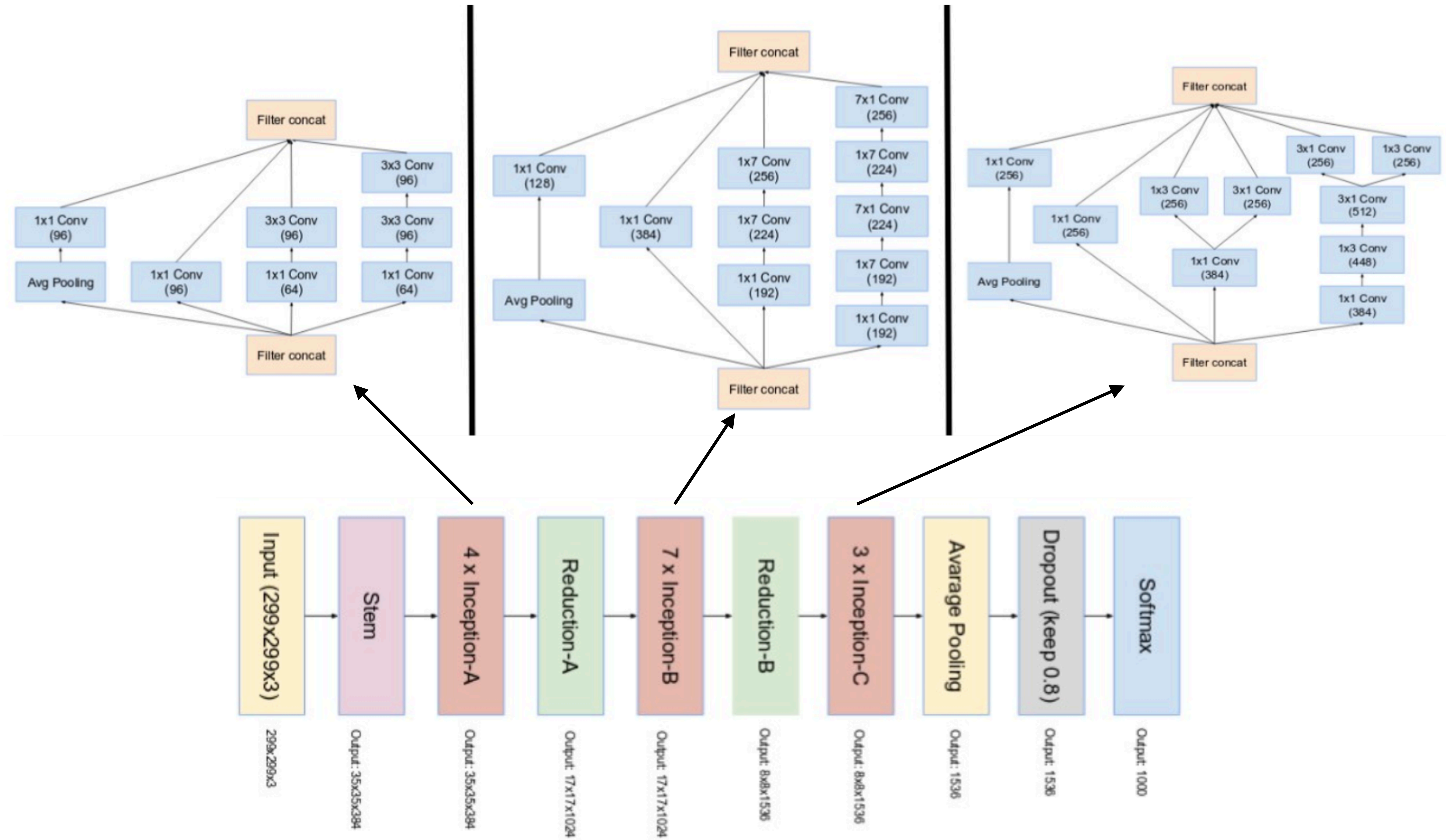
+ more flexible
- much harder to search



Neural Architecture Search

From experience: successful deep networks have repeated motifs (cells)

e.g. Inception v4:

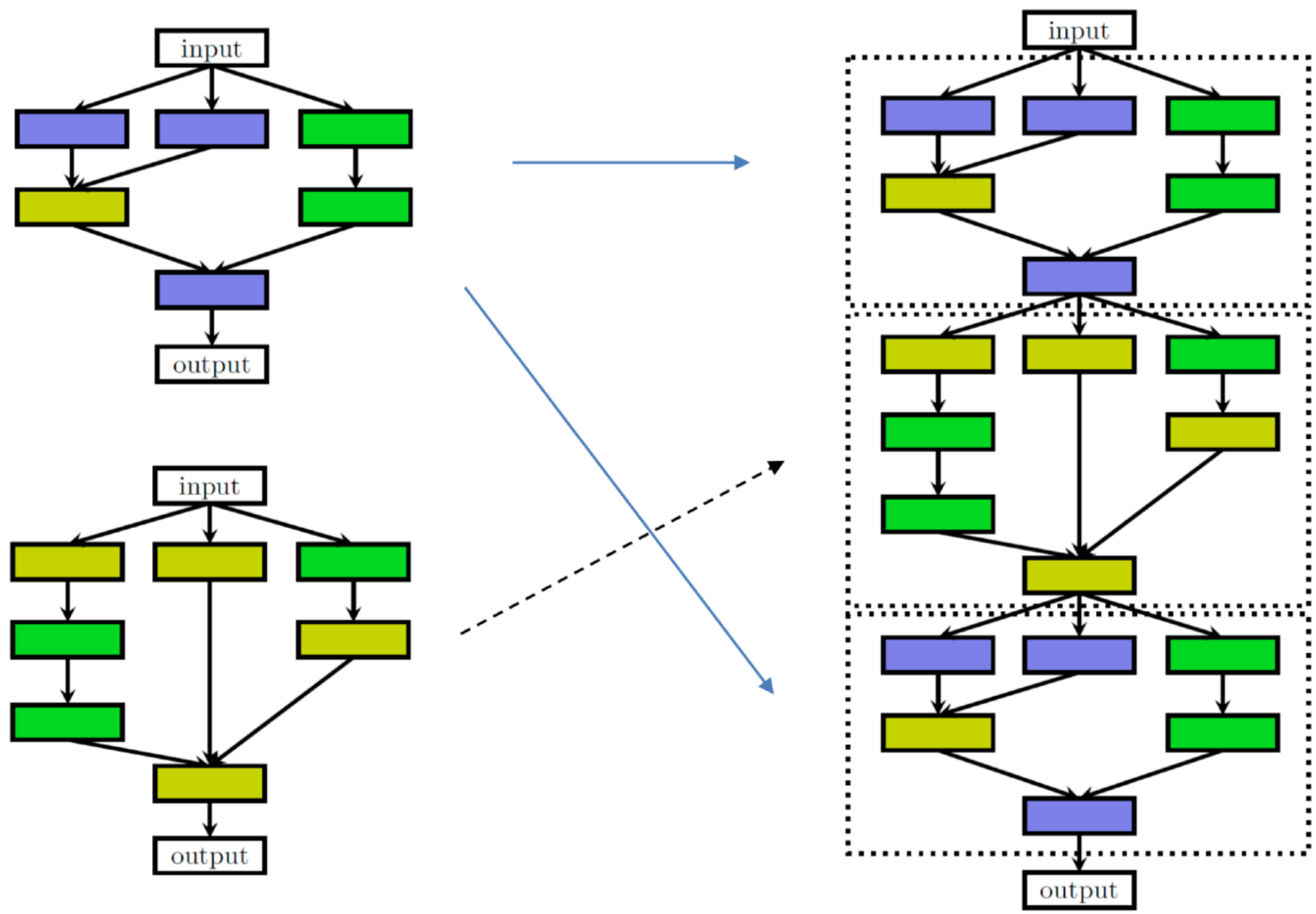


Meta-architectures

Compositionality: learn hierarchical building blocks to simplify the task

Cell search space

- learn parameterized building blocks *(cells)*
- stack cells together in macro-architecture



+ smaller search space
+ cells can be learned on a small dataset & transferred to a larger dataset
- strong domain priors, doesn't generalize well

***Idea:** pipeline search could also use compositionality/hierarchies?*

Zoph & Le 2017

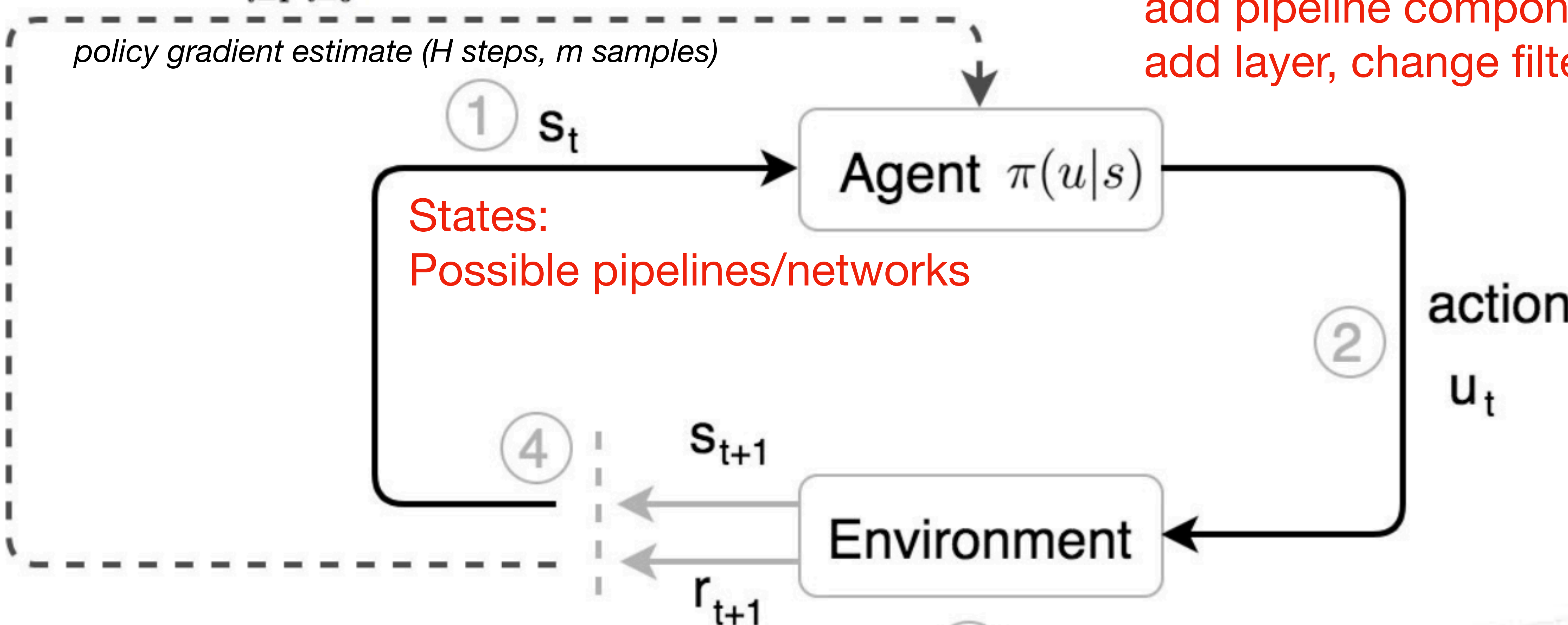
Levine Hui

Reinforcement learning

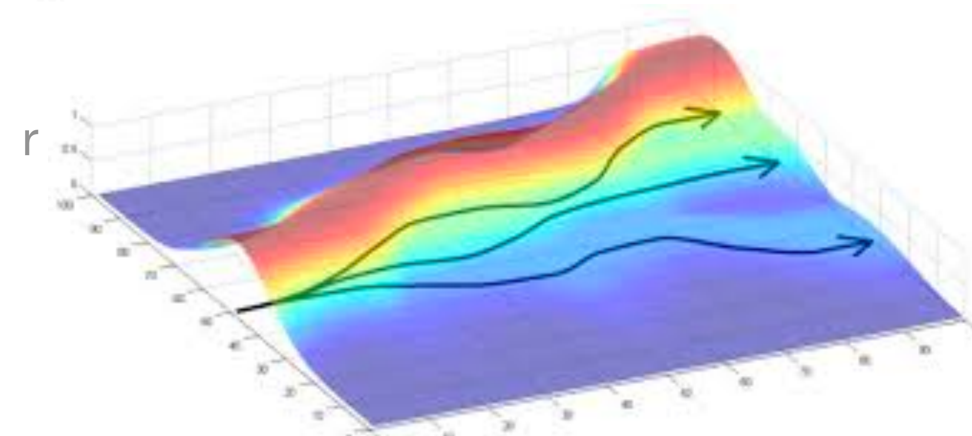
Build pipeline or network step-by-step, learn general strategy (policy)

5 $$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \, R(\tau^{(i)})$$

Actions:
add pipeline component, tune,…
add layer, change filter size,…



Reward:
Model performance after training
After trajectory τ of actions, adjust policy based on total reward $R(\tau)$

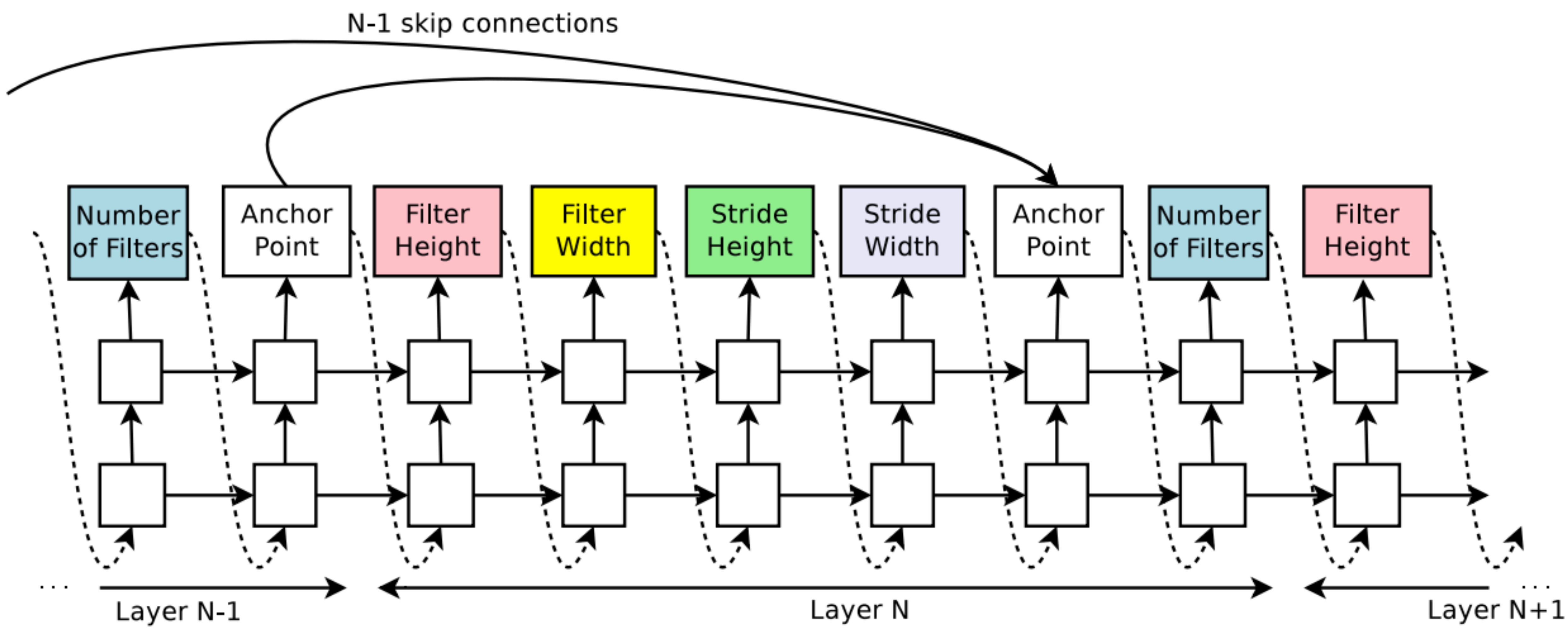


Zoph & Le 2017

NAS with Reinforcement learning

2-layer LSTM (REINFORCE), chains of convolutional layers

- State of the art on CIFAR-10, Penn Treebank
- 800 GPUs for 3-4 weeks, 12800 architectures



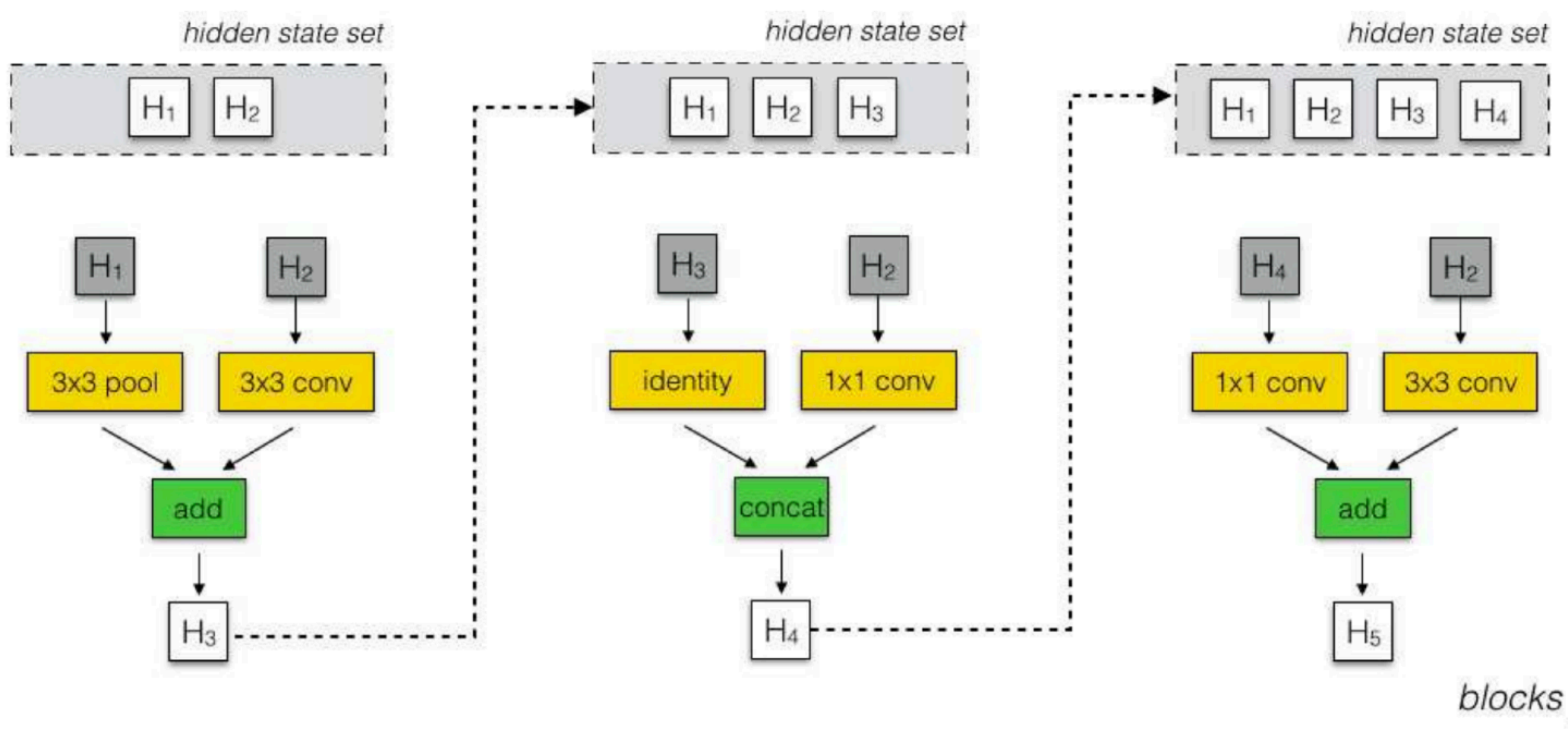
Zoph et al 2018

NAS with Reinforcement learning

1-layer LSTM (PPO), cell space search

- State of the art on ImageNet
- 450 GPUs, 3-4 days, 20000 architectures

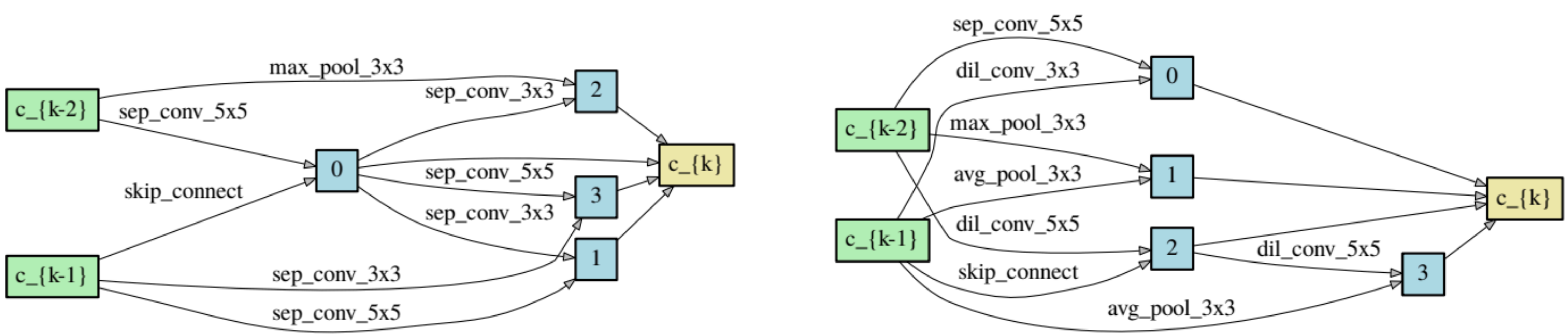
- Cell construction:
  - Select existing layers (hidden states, e.g. cell input) H_i to build on
  - Add operation (e.g. 3x3conv) on H_i
  - Combine into new hidden state (e.g. concat, add,…)
  - Iterate over B blocks



NAS with random search?

Li & Talwalkar 2019
Yu et al. 2019
Real et al. 2019

If you constrain the search space enough, you can get SOTA results with random search!



(a) Normal Cell (b) Reduction Cell

Convolutional Cells on CIFAR-10 Benchmark: Best architecture found by random search with weight-sharing.

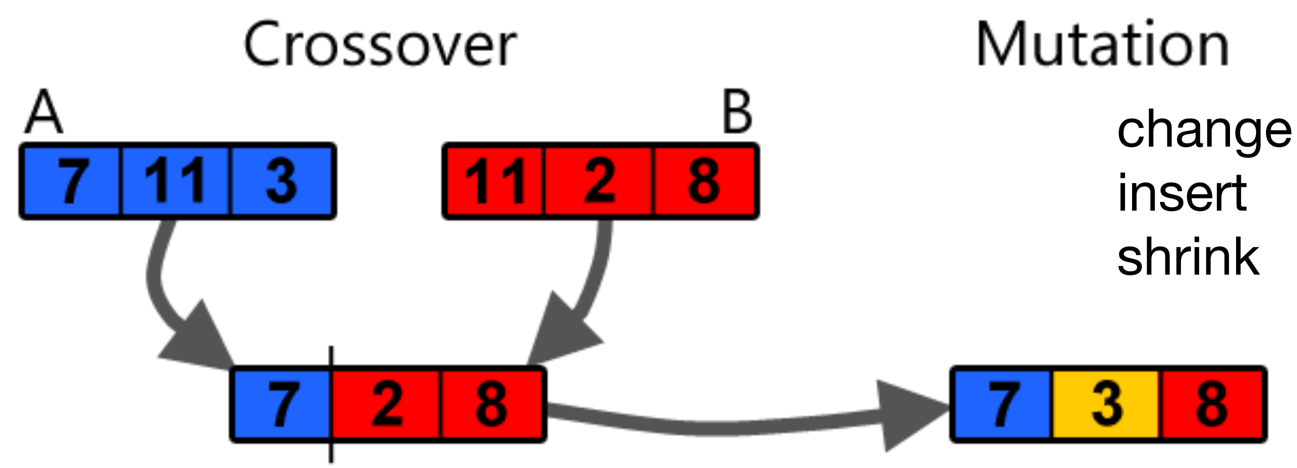
Architecture Search

Evolving pipelines

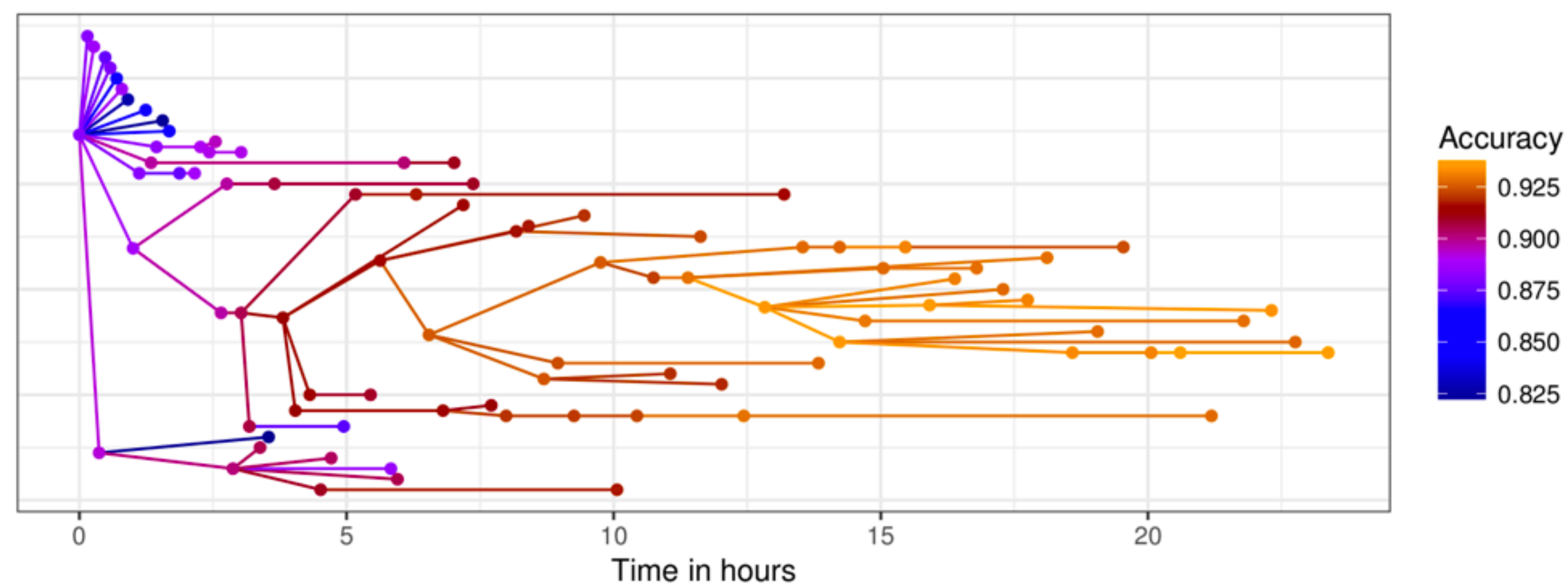
TPOT - *Olson & Moore 2016*

GAMA - *Gijsbers & Vanschoren 2018*

- Start with initial pipeline
- Best pipelines *evolve*: cross-over or mutation
    - Generation-based (TPOT)
    - Asynchronous (GAMA)
- No fixed maximal length
    - Adapts to complexity of the problem



+ more flexible
- larger search space, can be slower

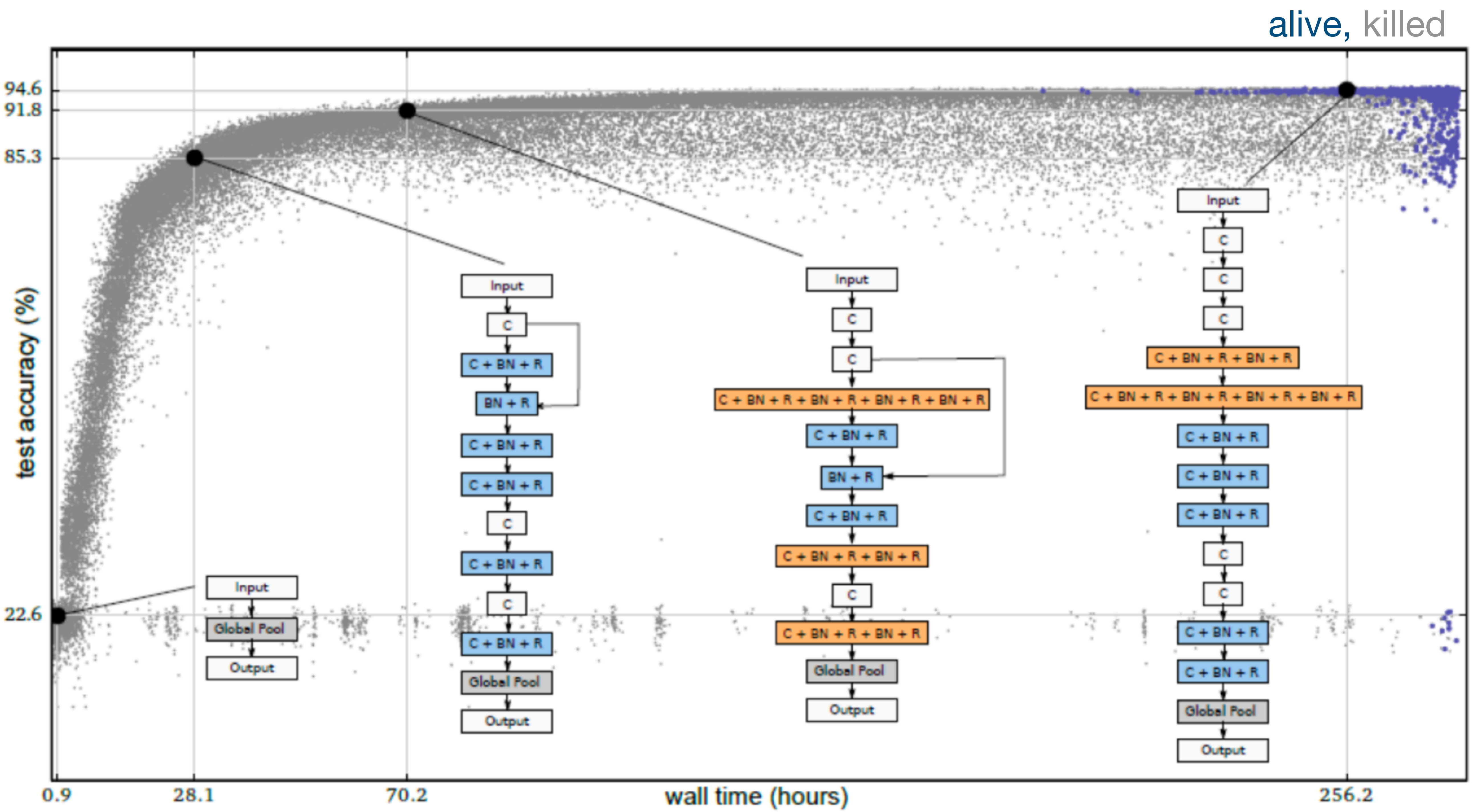


Architecture Search

Neuro-evolution

Real et al 2017
Miikkulainen et al 2017
Wistuba et al 2018

- same idea: learn the neural architecture through evolution
- mutations: add, change, remove layer

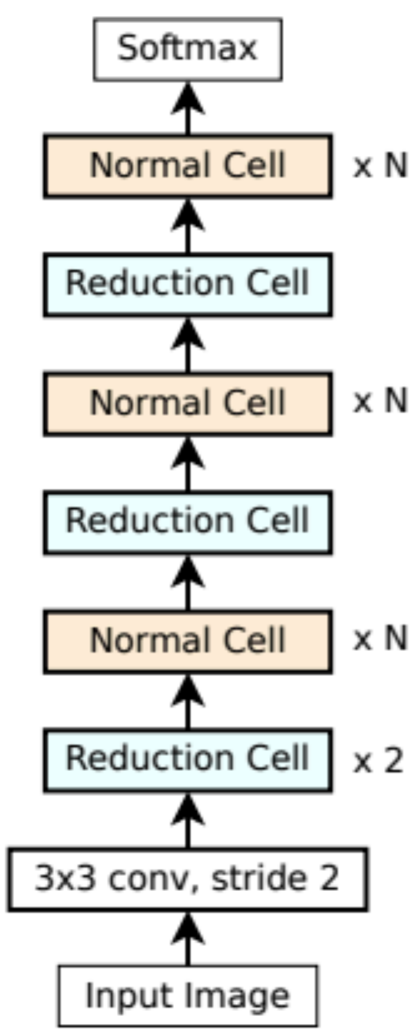


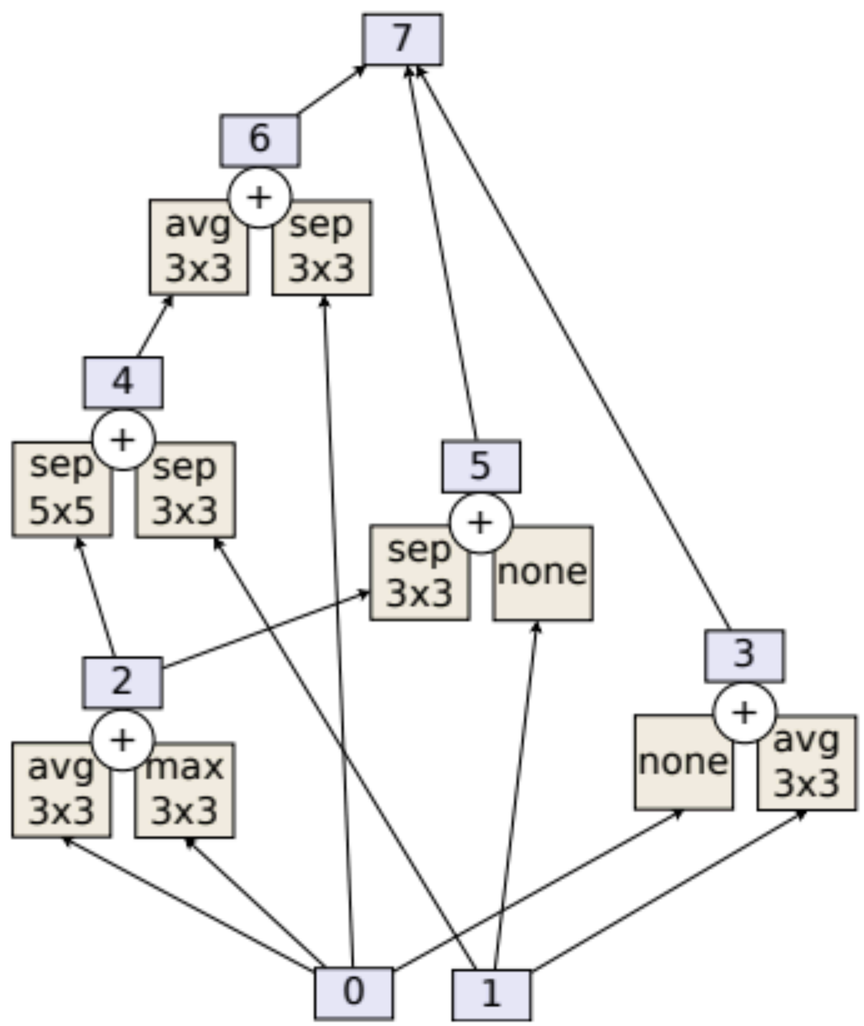
Real et al 2019

Neuro-evolution

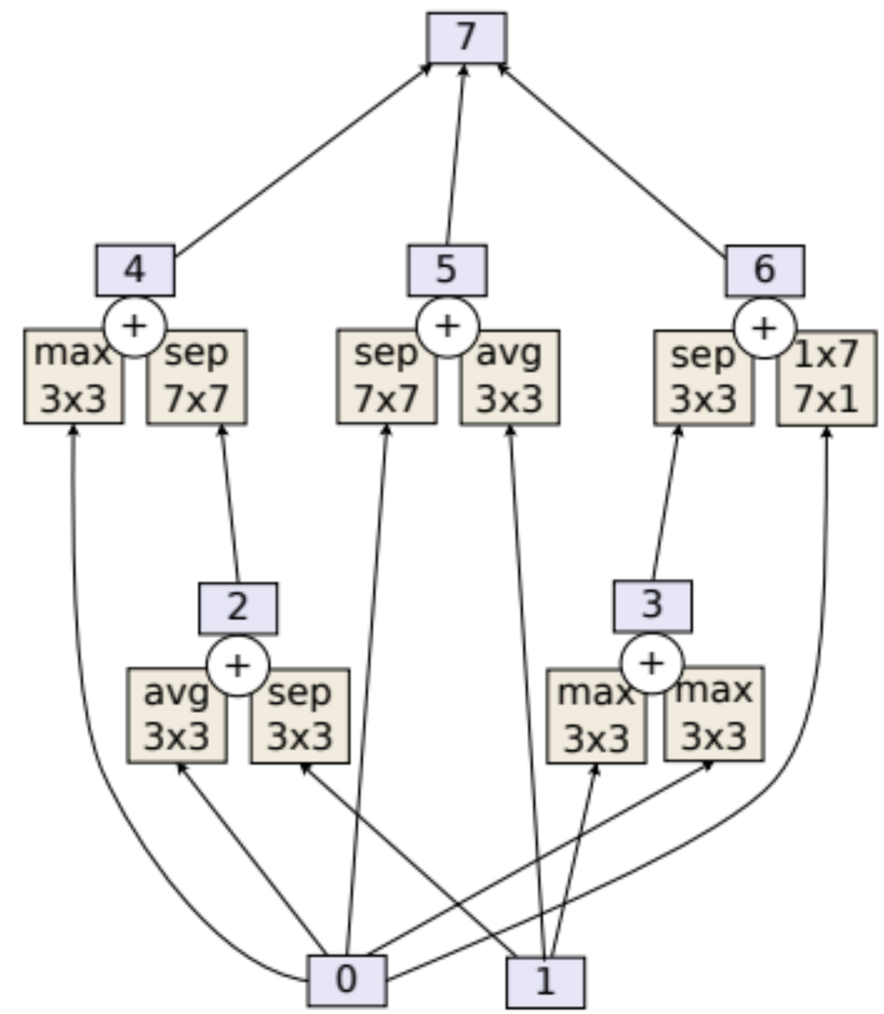
AmoebaNet: State of the art on ImageNet, CIFAR-10

- Cell search space, aging evolution (kill oldest networks)





normal cell



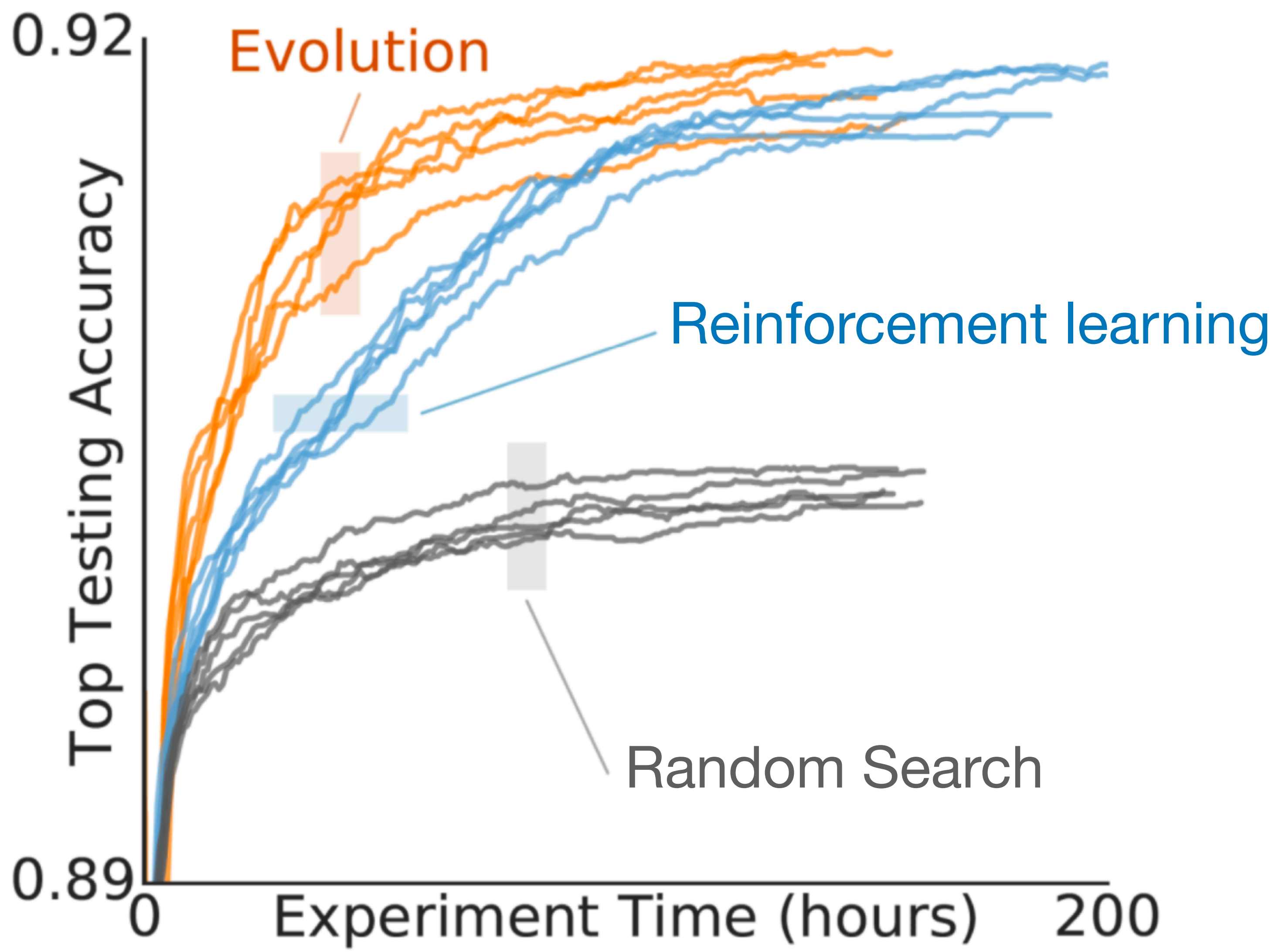
reduction cell

Real et al 2019

Neuro-evolution

AmoebaNet: State of the art on ImageNet, CIFAR-10

- Cell search space + aging evolution (kill oldest networks)
- More efficient than reinforcement learning

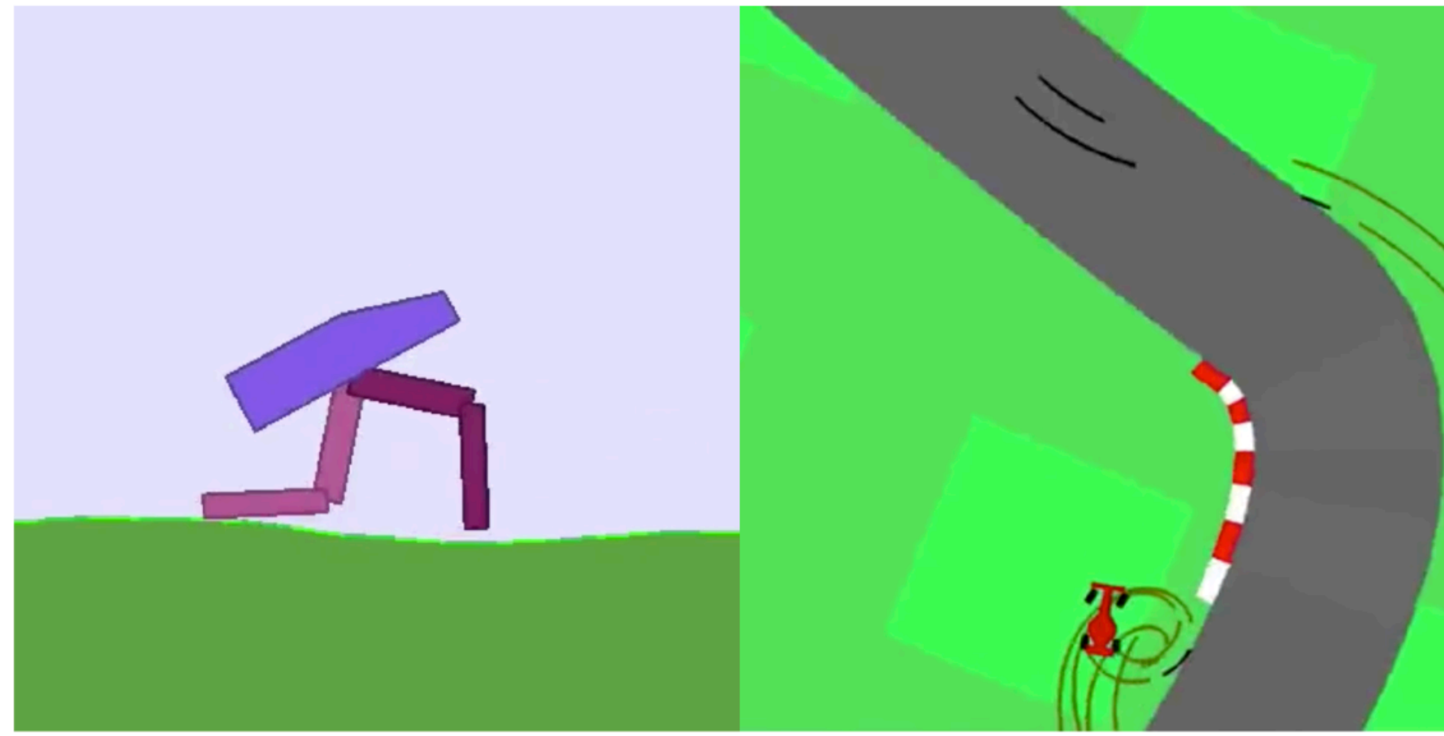


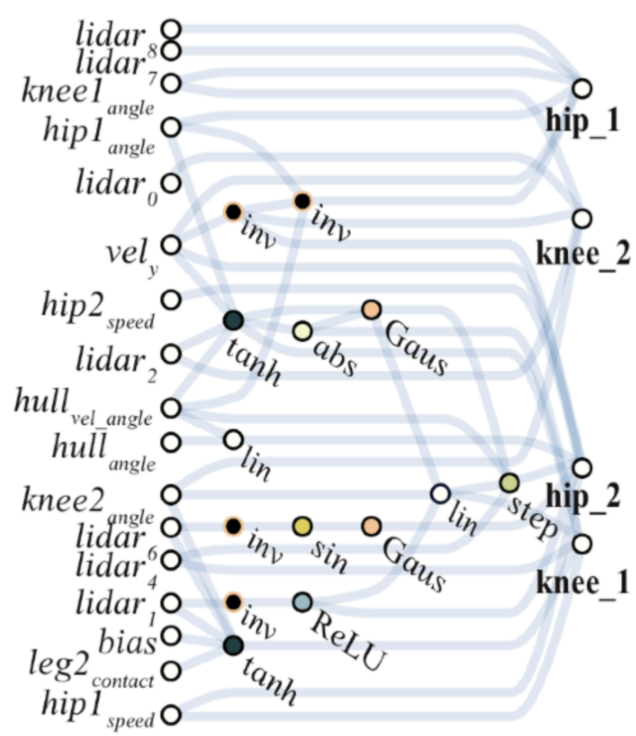
Gaier & Ha 2019

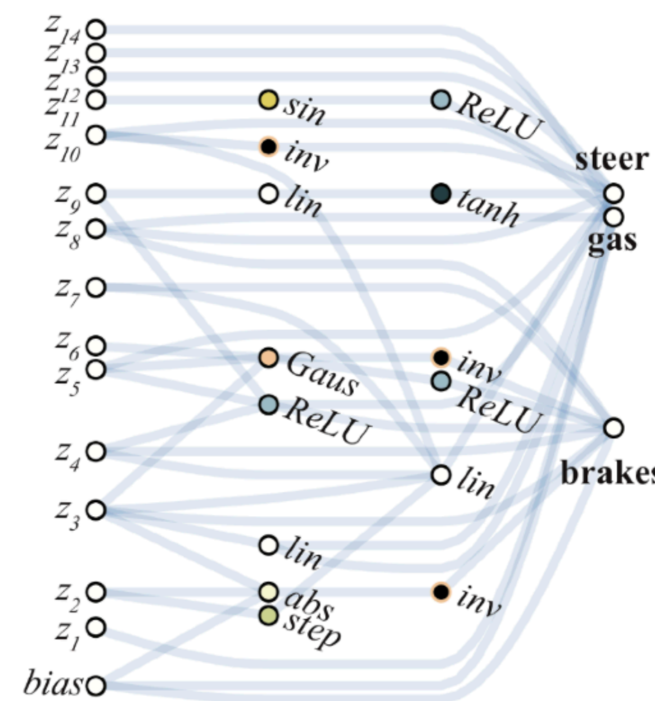
Weight-agnostic neural networks

Evolving the networks while training the model weights is expensive

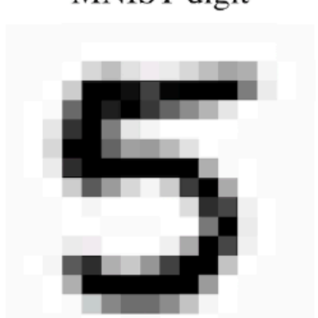
- Can we just fix the model weights and only evolve the architecture

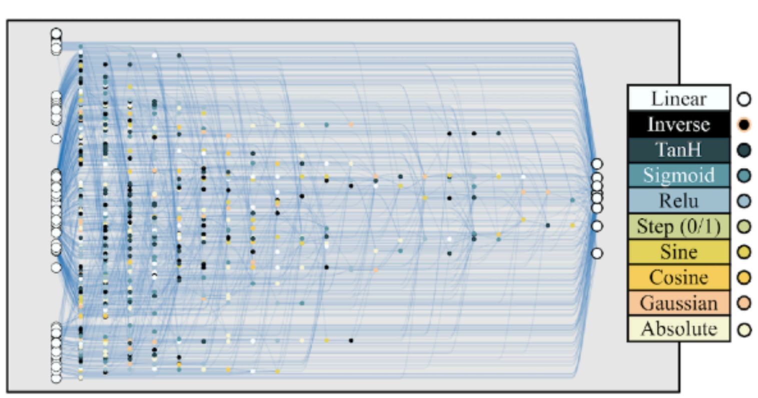






MNIST digit

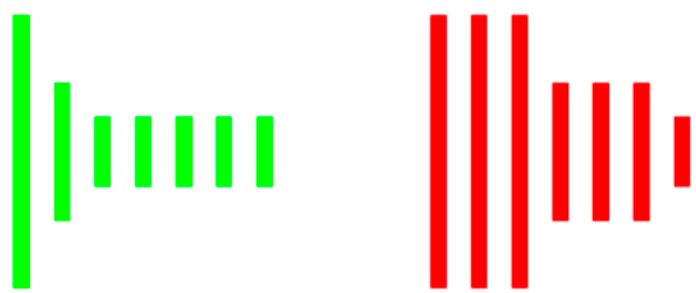


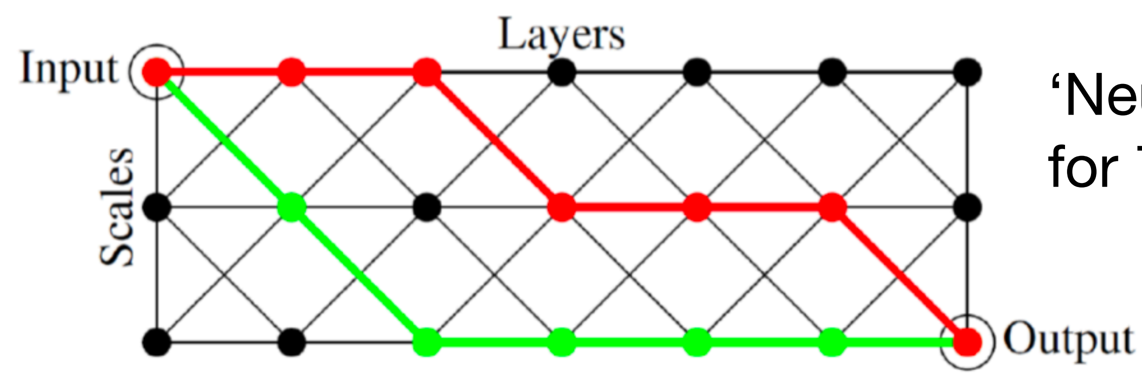


[1] Saxena & Verbeek, 2016 [2] Pham et al. 2018
[3] Bender et al. 2018 [4] Brock et al. 2017

Hypernetworks

- Every candidate network belongs to a *hypernetwork*



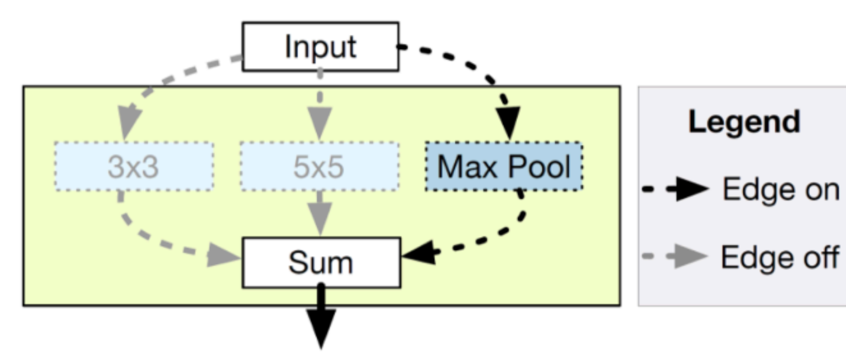


'Neural fabric' for 7-layer CNNs [1]

  - Train in parallel, share the weights of 'common' layers: faster
  - Hypernetwork can be designed to reduce the search space

- *EfficientNAS* [2]
  - use RL to sample paths from the graph, share weights for evaluation

- *One-shot models* [3]
  - Train all nets in parallel, with *path dropout*

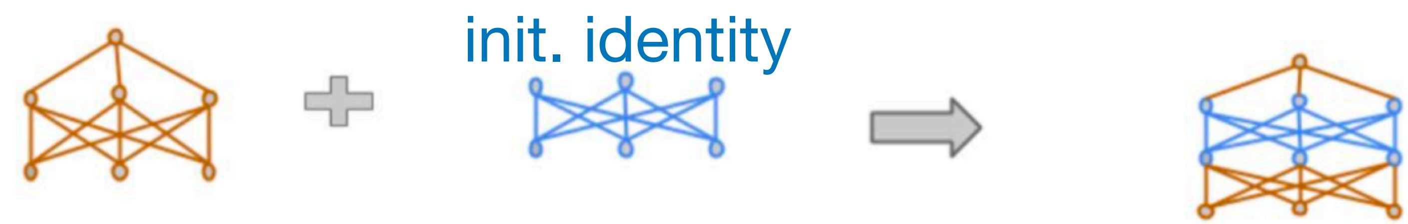


- *SMASH* [4]
  - Shared hypernetwork that predicts weights given architecture

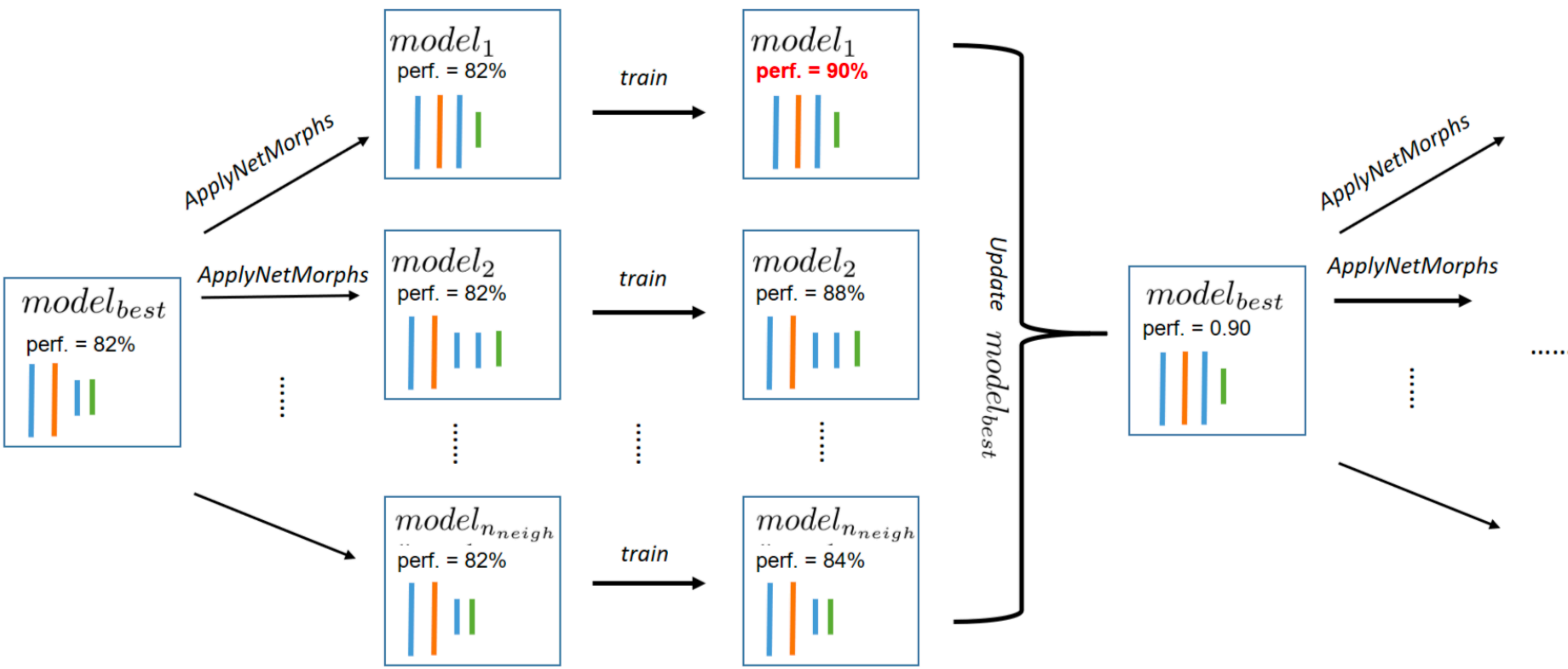
Chen et al. 2016, Wei et al. 2016, Cai et al. 2017
Cai et al. 2018, Elsken et al. 2017, Cortes et al. 2017, Cai et al. 2018

Network Morphisms

[AutoKeras]

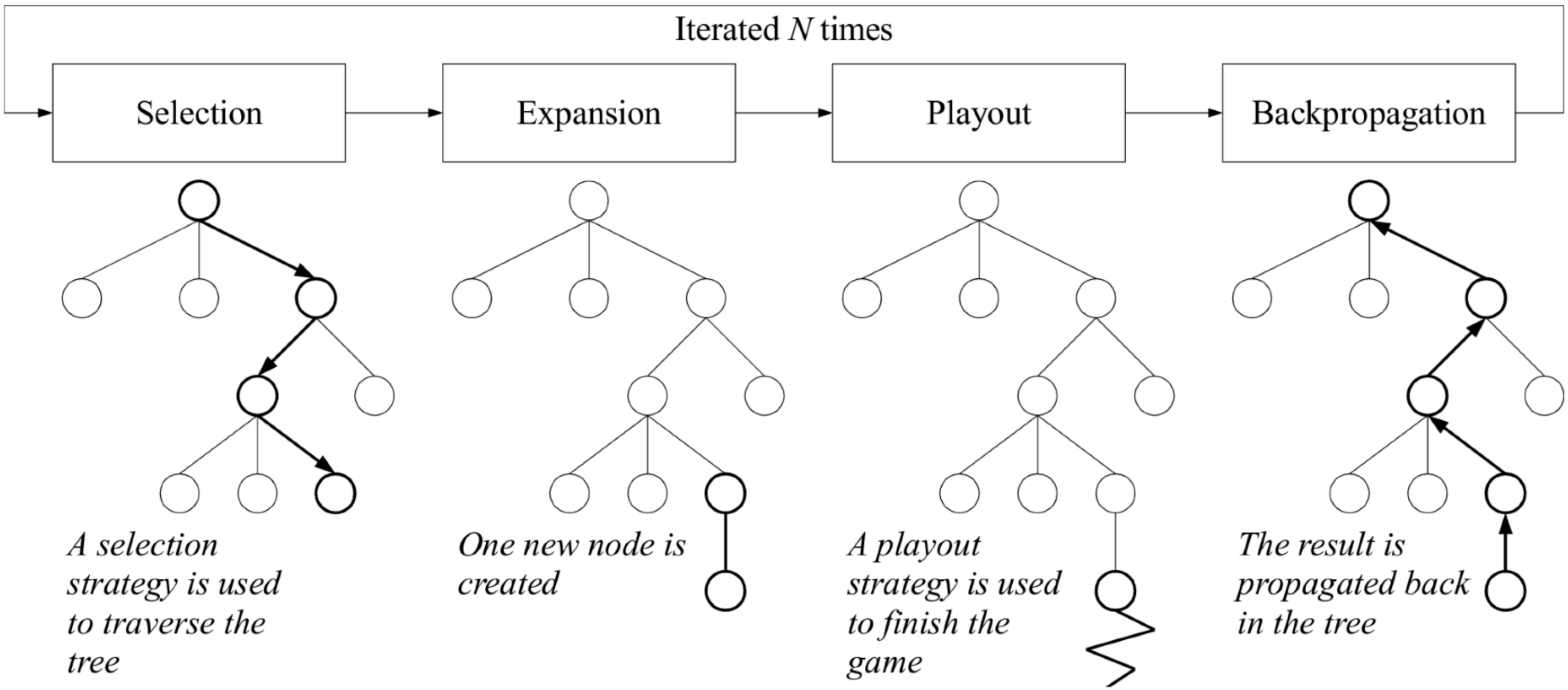


- Change architecture, but not modelled function
- Keep the initial weights, only update the derived architectures



Monte Carlo Tree Search (MCTS)

- Select partial pipeline from tree of all pipelines
- Add one new operator, do random completion to estimate accuracy
- Update nodes according to performance

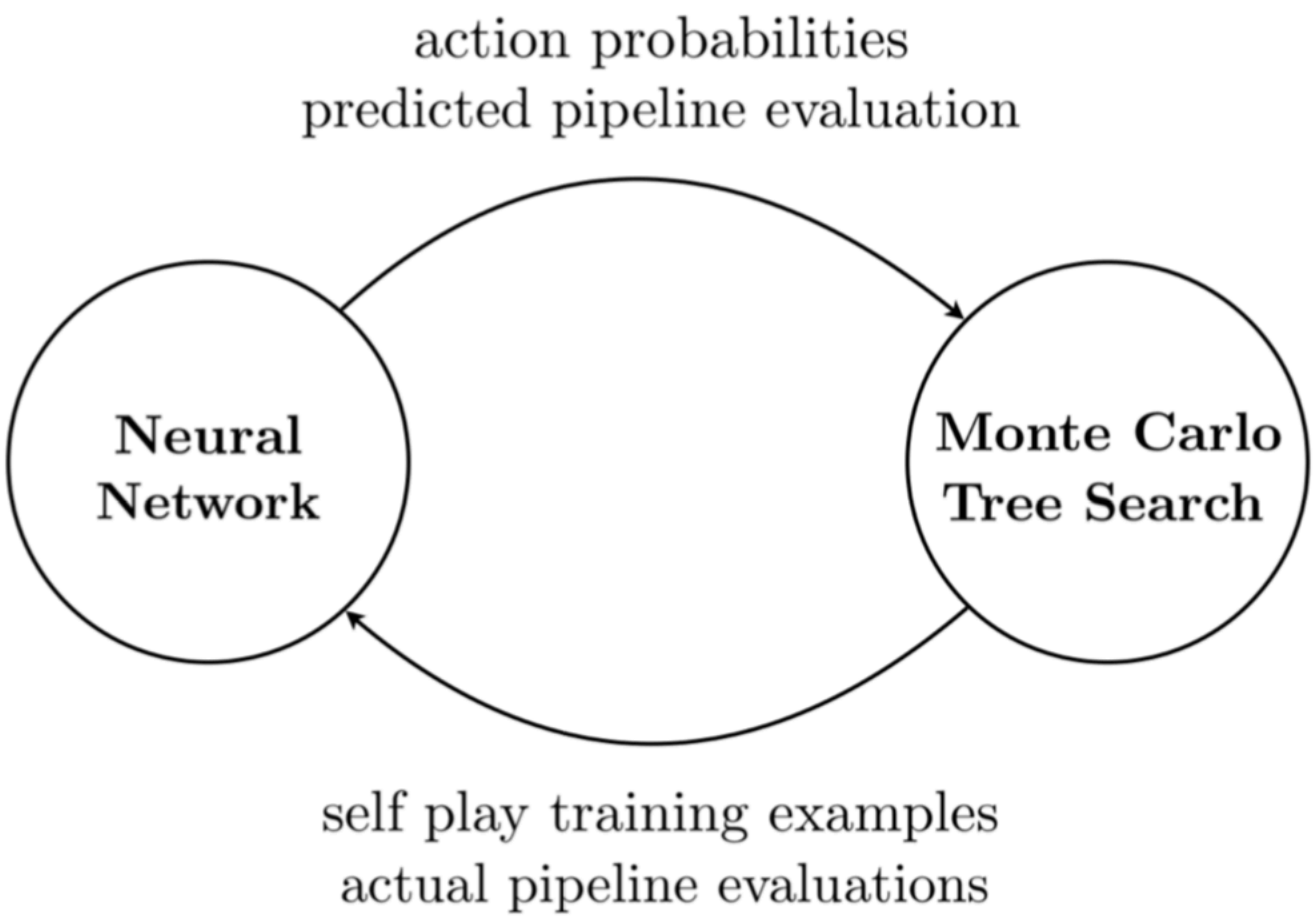


[Chaslot et al., 2008]

MOSAIC [Rakotoarison et al. 2019]
AlphaD3M [Drori et al. 2018]

MCTS + reinforcement learning

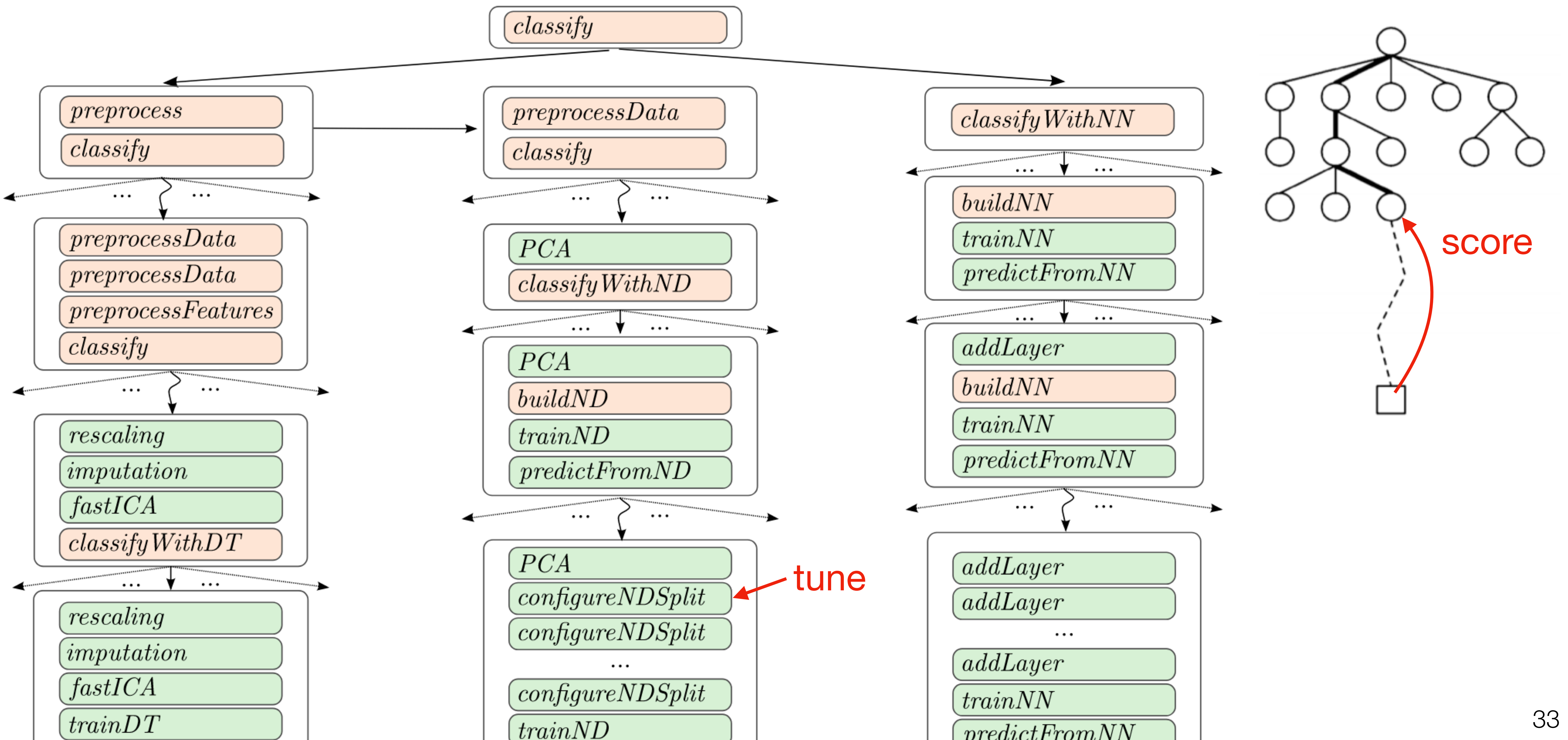
- Build pipelines by inserting, deleting, replacing components (actions)
- *Self-play:*
  - Monte Carlo Tree Search builds pipelines given action probabilities
  - Neural network (LSTM) Predicts pipeline performance



ML-Plan [Mohr et al. 2018]
P4ML [Gil et al. 2018]

Hierarchical planning

- Use planning to ‘plan' pipeline, e.g. with best first search
- Use random path completion (as in MCTS) to evaluate each node

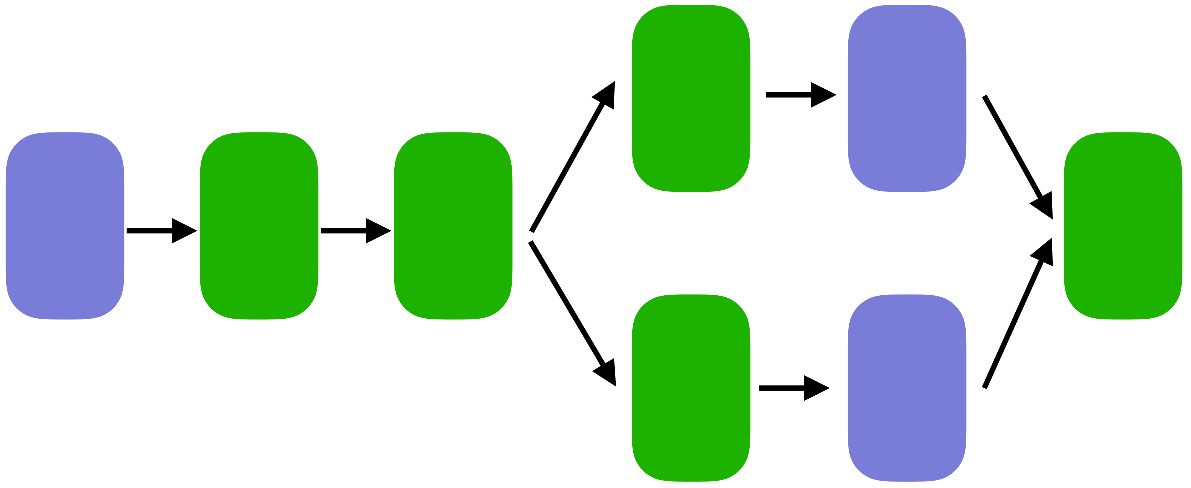


Hyperparameter Optimization

Searching the hyperparameter space ***efficiently***

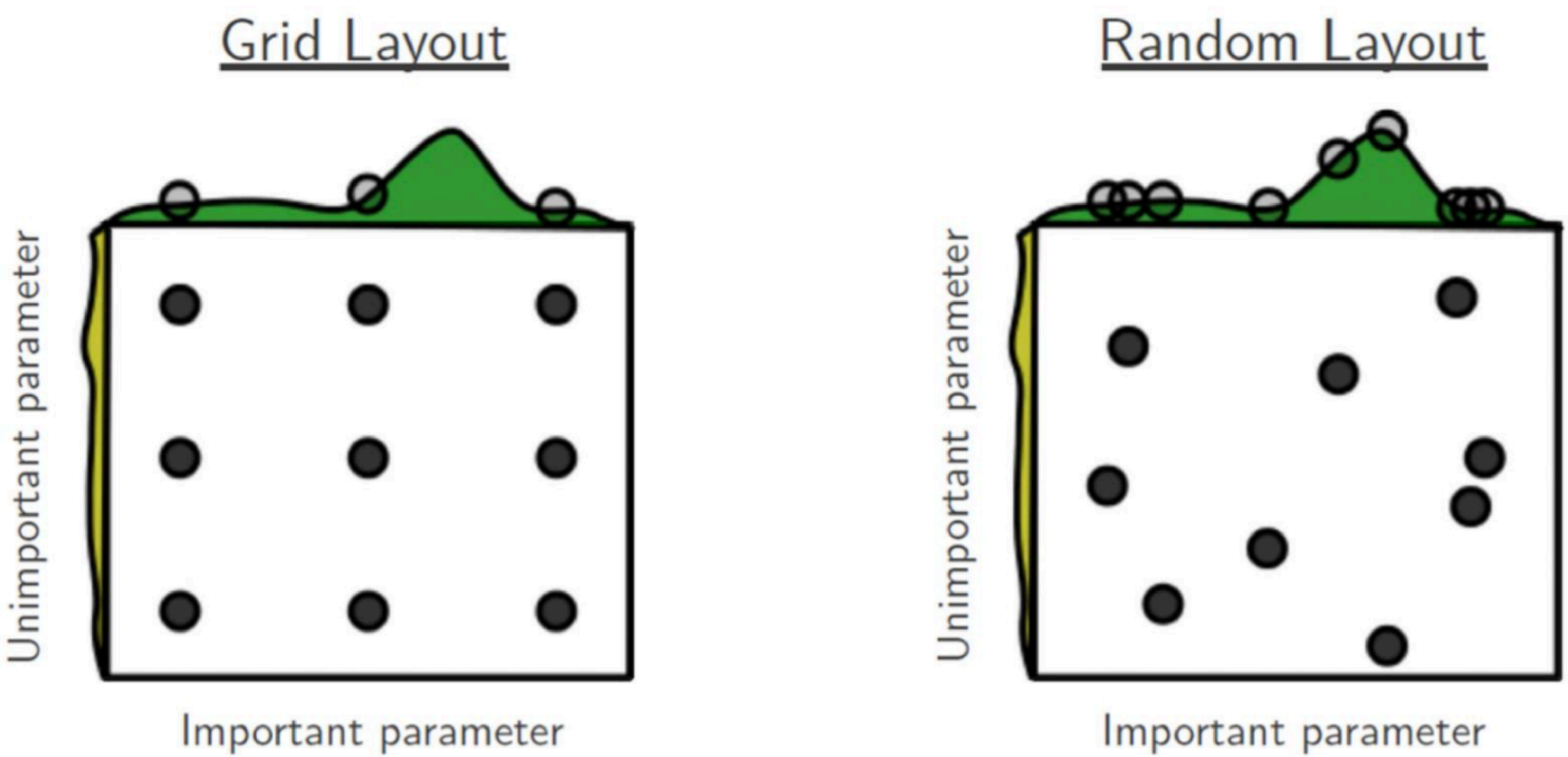
AutoML: subproblems

- **Hyperparameter optimization**: *many different approaches*
  - Random search
  - Bayesian optimization
  - Population-based methods (evolution)
  - Multi-fidelity optimization
  - Differentiable optimization
  - …



Random search

- Handles unimportant dimensions better than grid search
- Easily parallelizable, but uninformed (no learning)

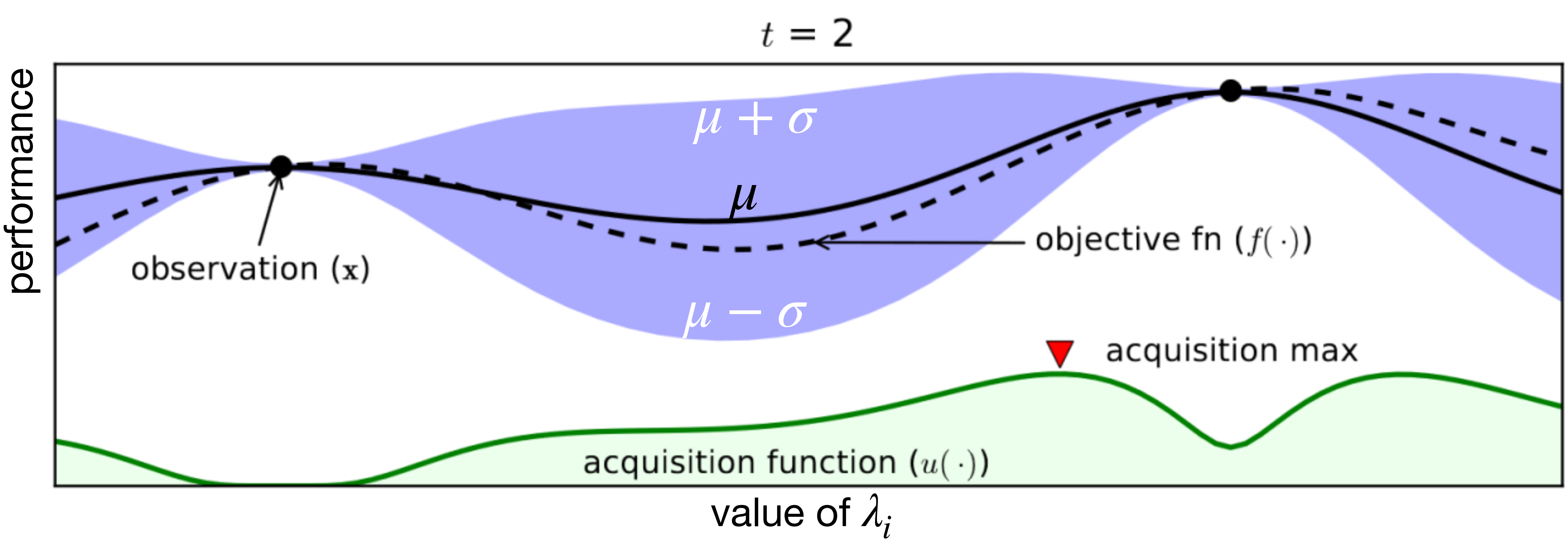


Bergstra & Bengio, JMLR 2012

Mockus [1974]

Bayesian Optimization

- Start with a few (random) hyperparameter configurations
- Fit a ***surrogate model*** to predict other configurations
- Probabilistic regression: mean μ and standard deviation σ (blue band)
- Use an ***acquisition function*** to trade off exploration and exploitation, e.g. Expected Improvement (EI)
- Sample for the best configuration under that function

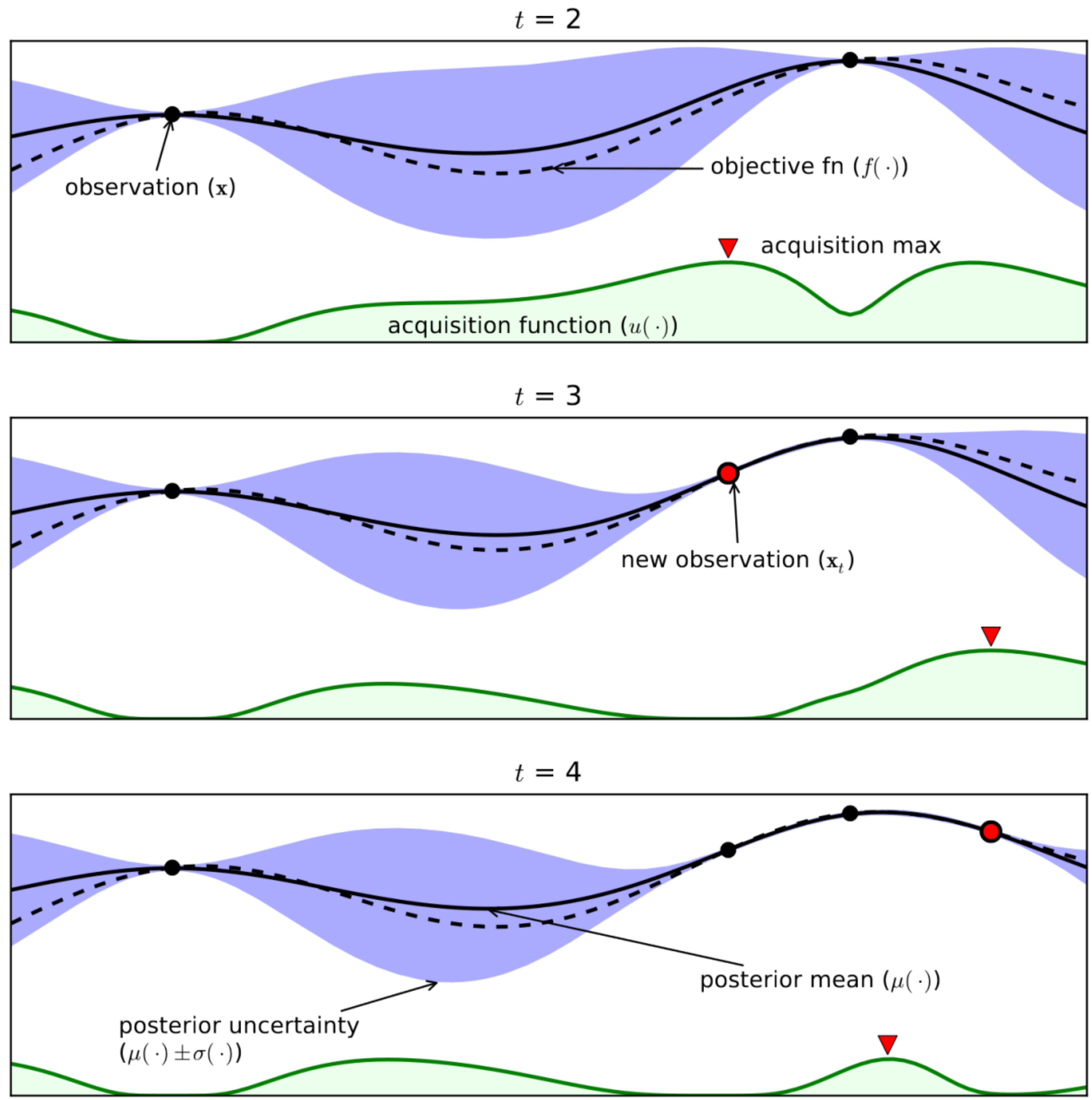


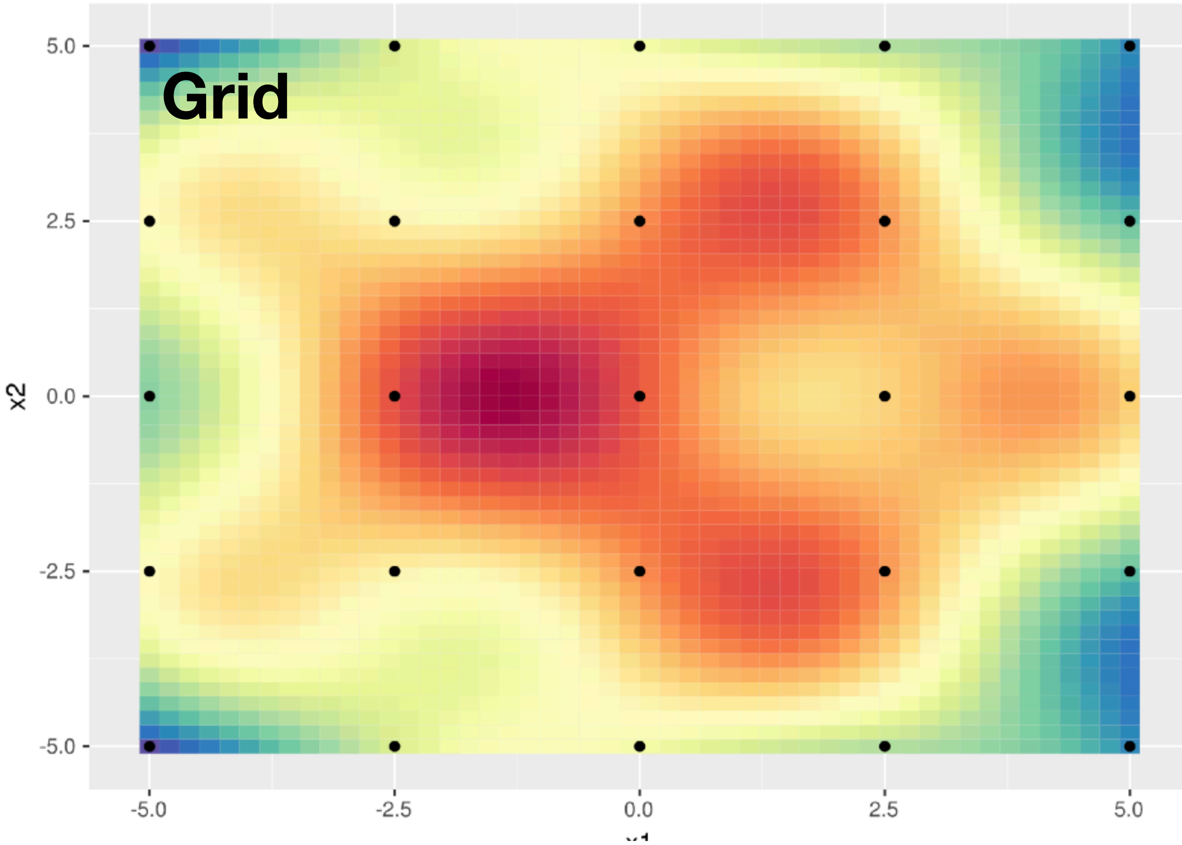
Bayesian Optimization

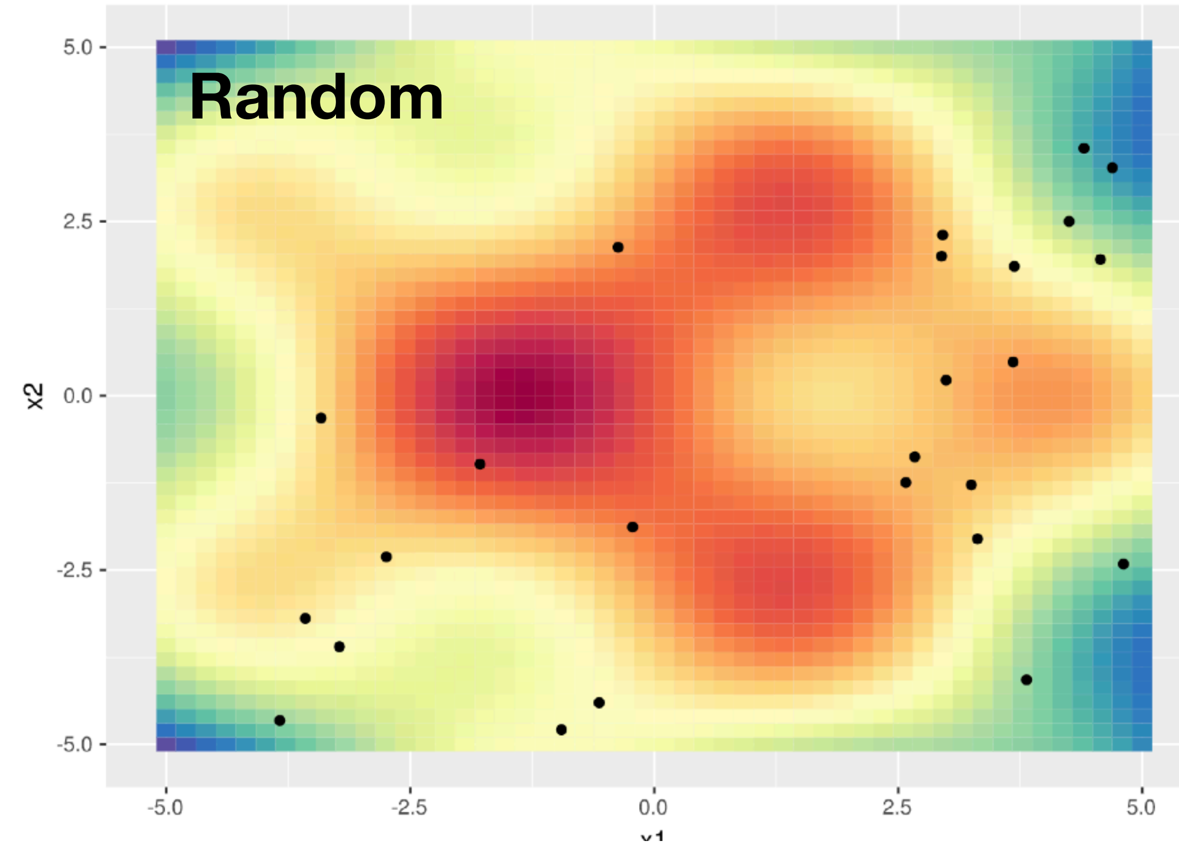
- Repeat until some stopping criterion:
  - Fixed budget
  - Convergence
  - EI threshold

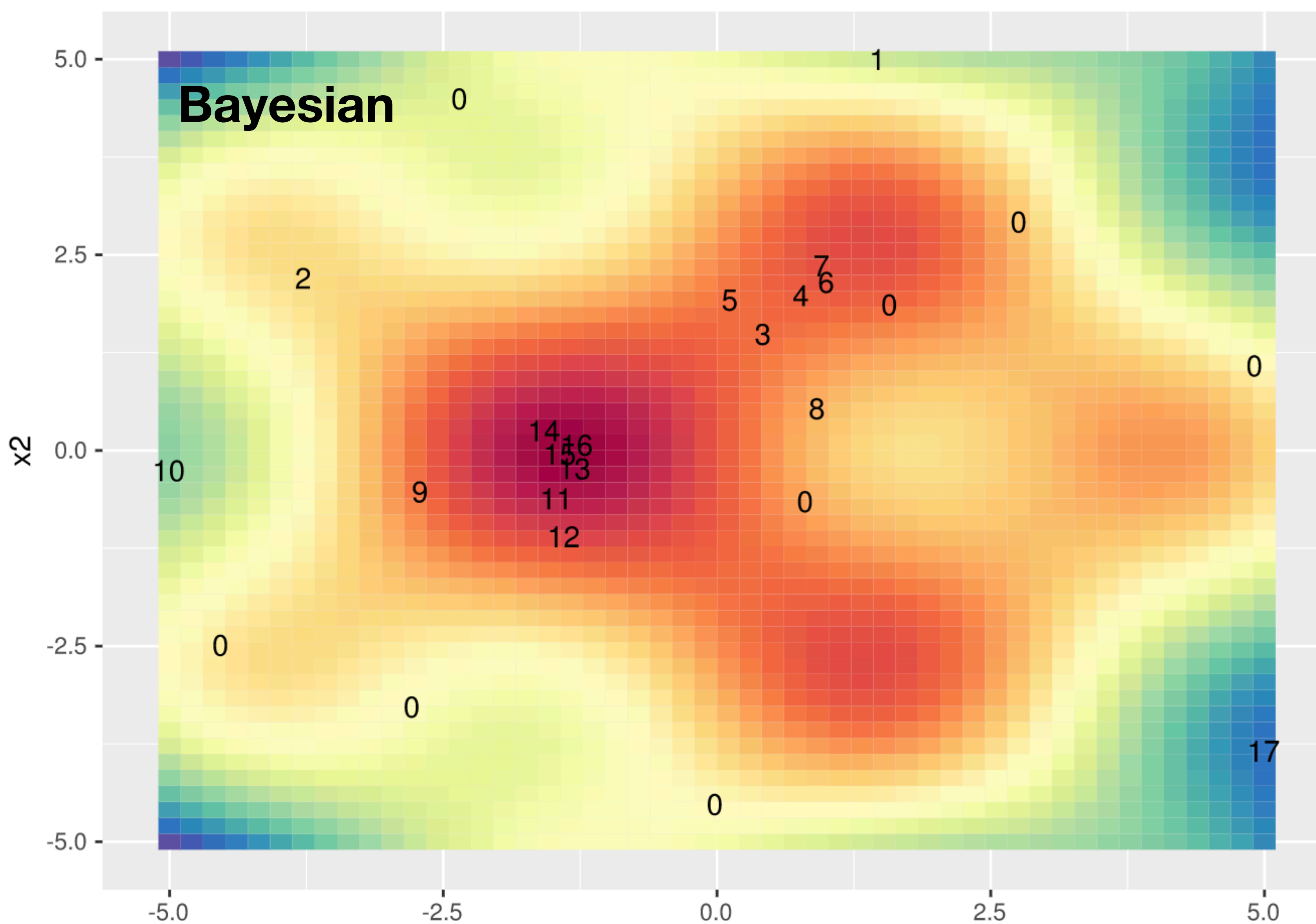
- Theoretical guarantees
  Srinivas et al. 2010, Freitas et al. 2012, Kawaguchi et al. 2016

- Also works for non-convex, noisy data
- Used in AlphaGo




Grid
x2
5.0
2.5
0.0
-2.5
-5.0
-5.0
-2.5
0.0
2.5
5.0

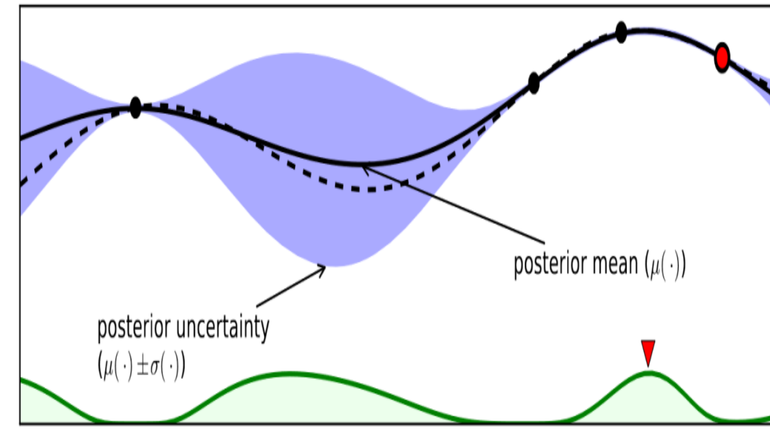

Random
x2
5.0
2.5
0.0
-2.5
-5.0
-5.0
-2.5
0.0
2.5
5.0


Bayesian
x2
5.0
2.5
0.0
-2.5
-5.0
-5.0
-2.5
0.0
2.5
5.0
1
0
0
2
5
4
7
6
0
3
0
8
14
15
16
13
10
9
11
0
12
0
0
17
0

Hyperparameter optimization

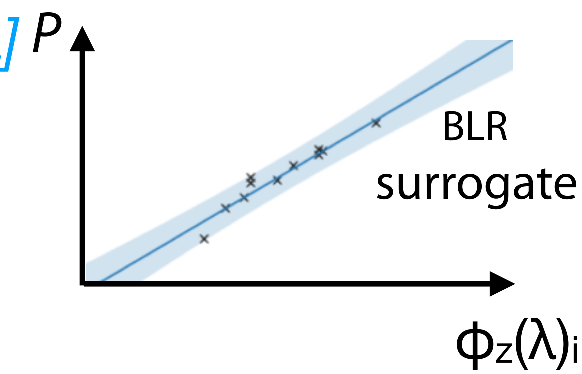
Bayesian Optimization: which surrogate?

Gaussian processes [skopt]



+ handles uncertainty, extrapolates well
+ ideal for few numeric hyperparameters
- scales badly (cubic), try random embeddings [Wang et al. 2013]

Bayesian Linear Regression [Snoek et al. 2015] [Amazon AutoML]

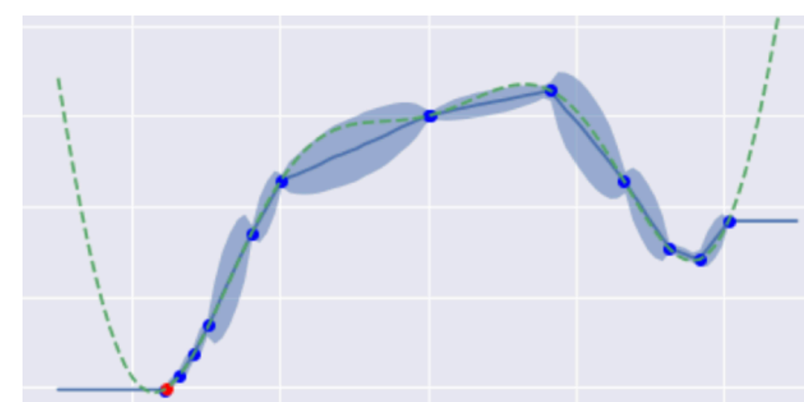


+ Scalable, Bayesian, - requires good basis expansion

Bayesian neural networks [Springenberg et al. 2016]

+ Bayesian - Scalability?

Random Forests [Hutter et al. 2011, Feurer et al. 2015] [SMAC]

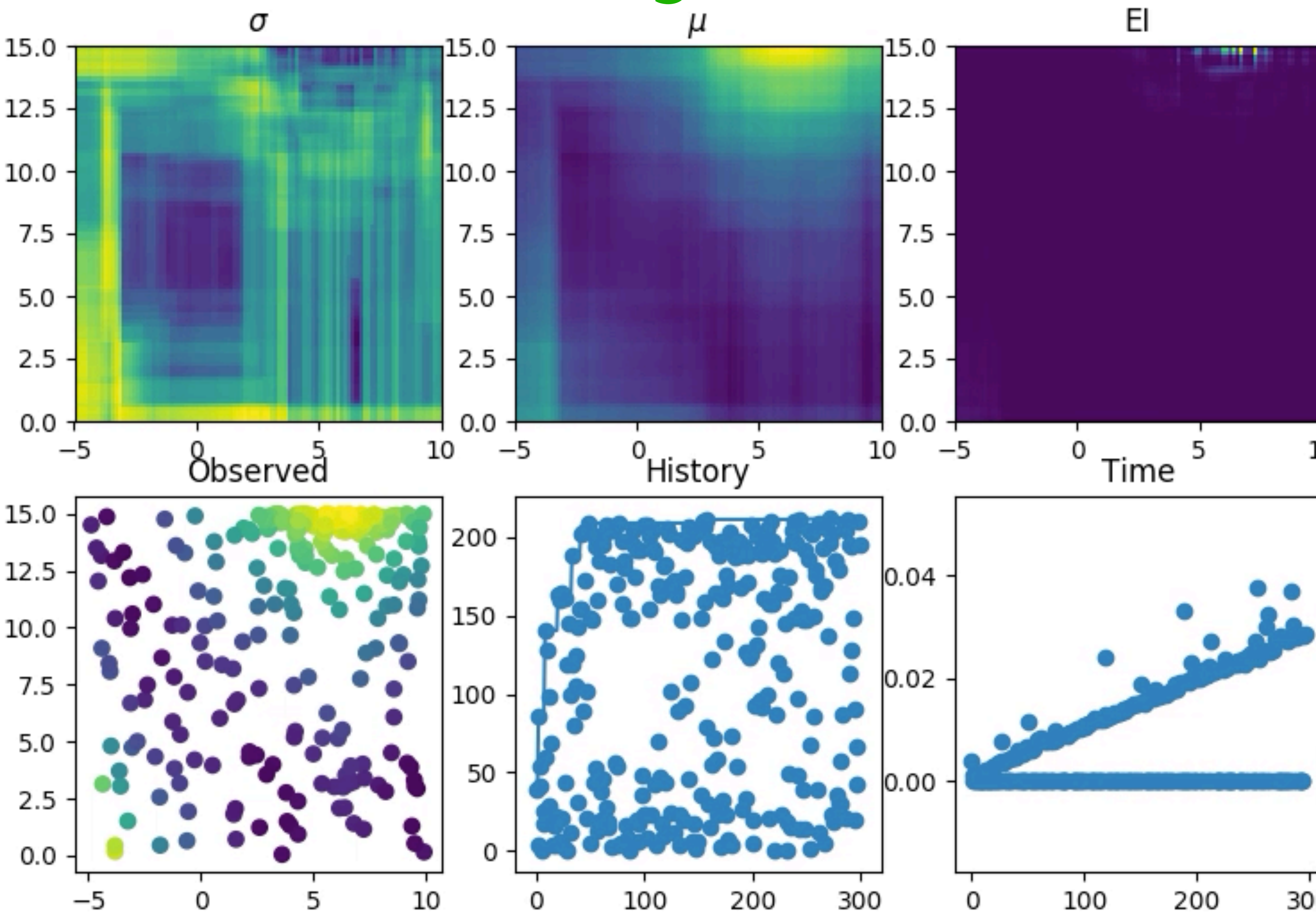


+ scalable, handles conditional hyperparams
- bad uncertainty estimates, extrapolation

Boosting + quantile regression [van Hoof, Vanschoren 2019] [HyperBoost]

+ better fit than RF, handles conditionals, adapts to drift - new

Random Forest Surrogate



animation by Jeroen van Hoof

[Komer et al. 2019]

Tree of Parzen Estimators

1.Test some hyperparameters

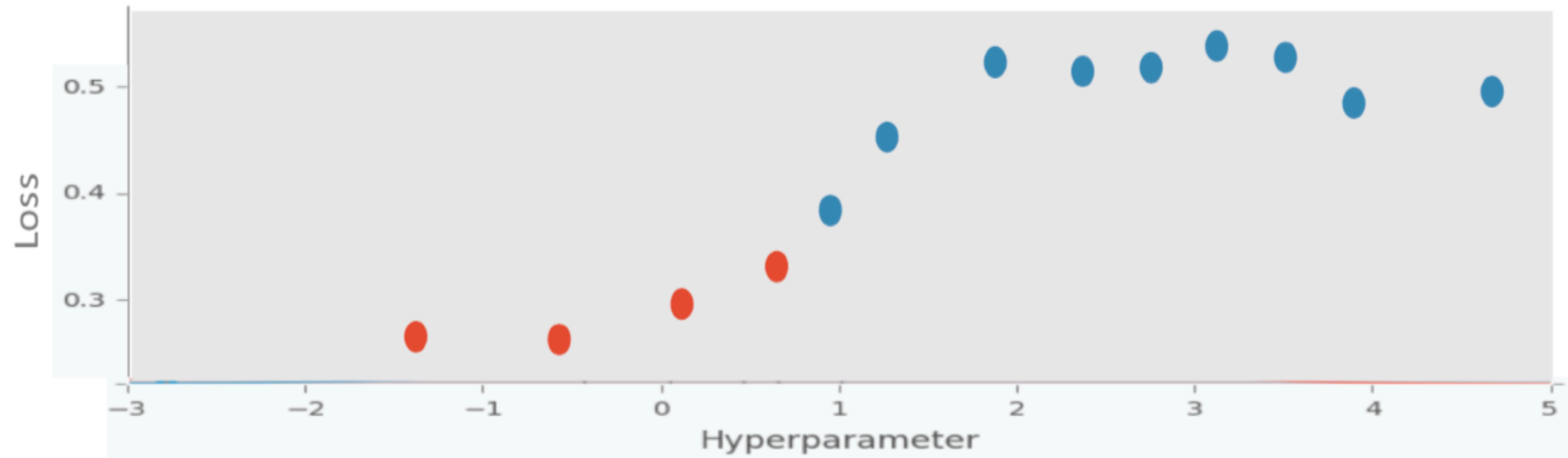
2. Separate into good and bad hyperparameters (with some quantile)

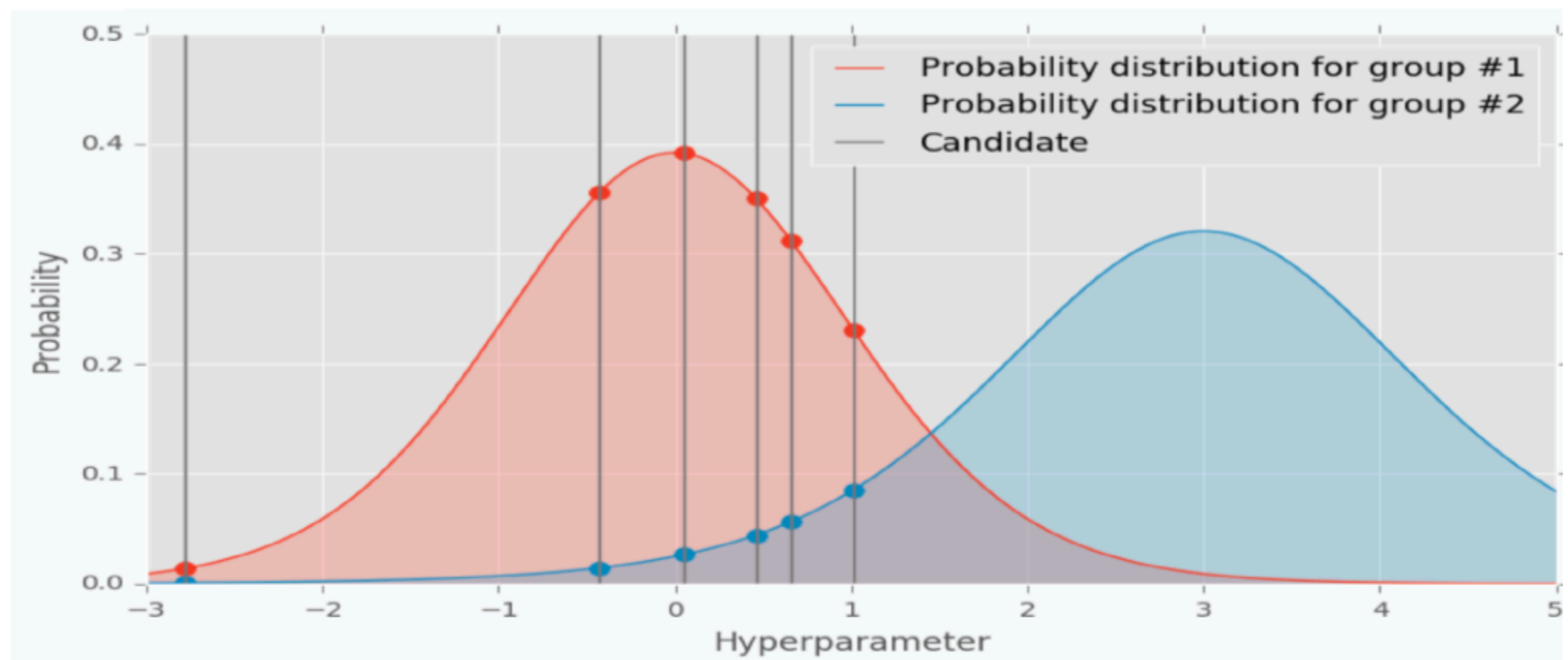
3. Fit non-parametric KDE for $p(\lambda = \text{good})$ and $p(\lambda = \text{bad})$

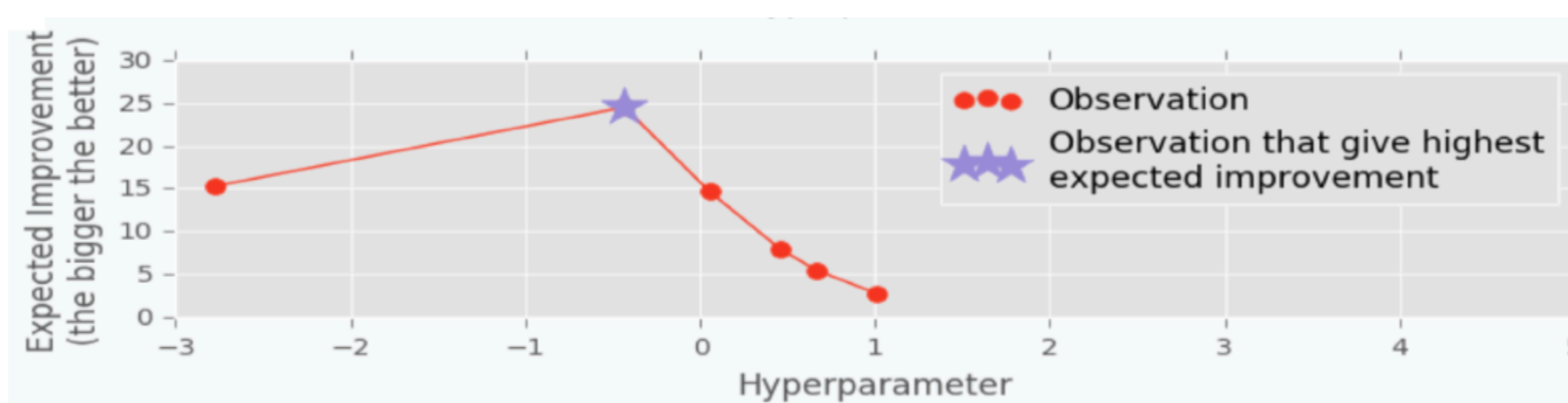
4. For a few samples, evaluate $\dfrac{p(\lambda = \text{good})}{p(\lambda = \text{bad})}$

Shown to be equivalent to EI!

Efficient, **parallelizable**, robust, but less sample efficient than GPs







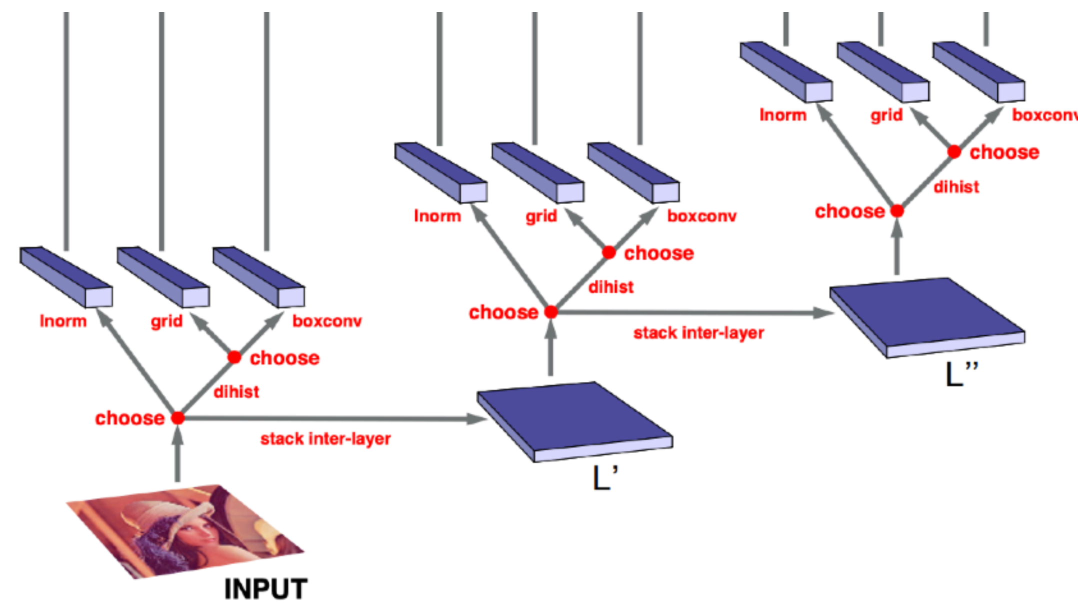
Hyperparameter optimization

Bayesian Optimization

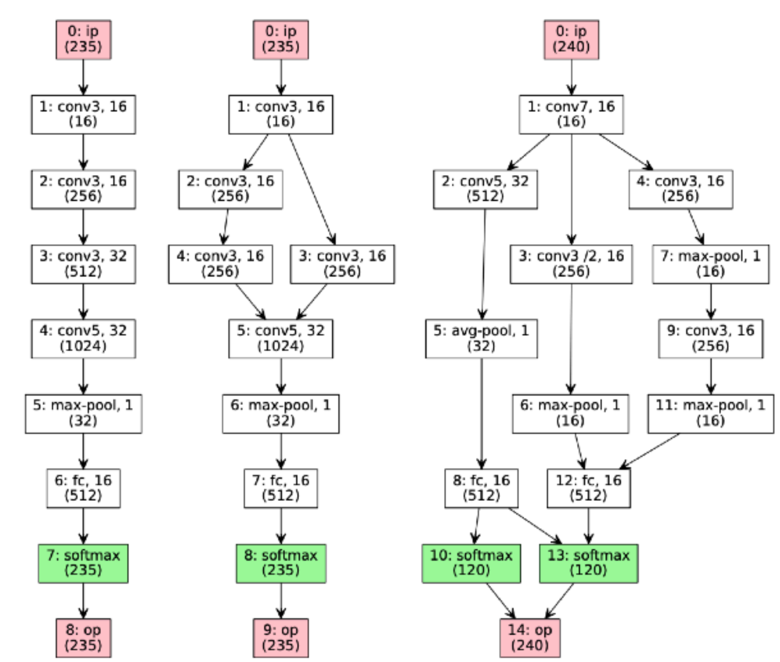
[1] Bergstra et al. 2013
[2] Swersky et al 2013
[3] Kandasamy et al. 2018

Examples:

- Image classification pipelines[1]
    - 238 hyperparameters, tuned with TPE



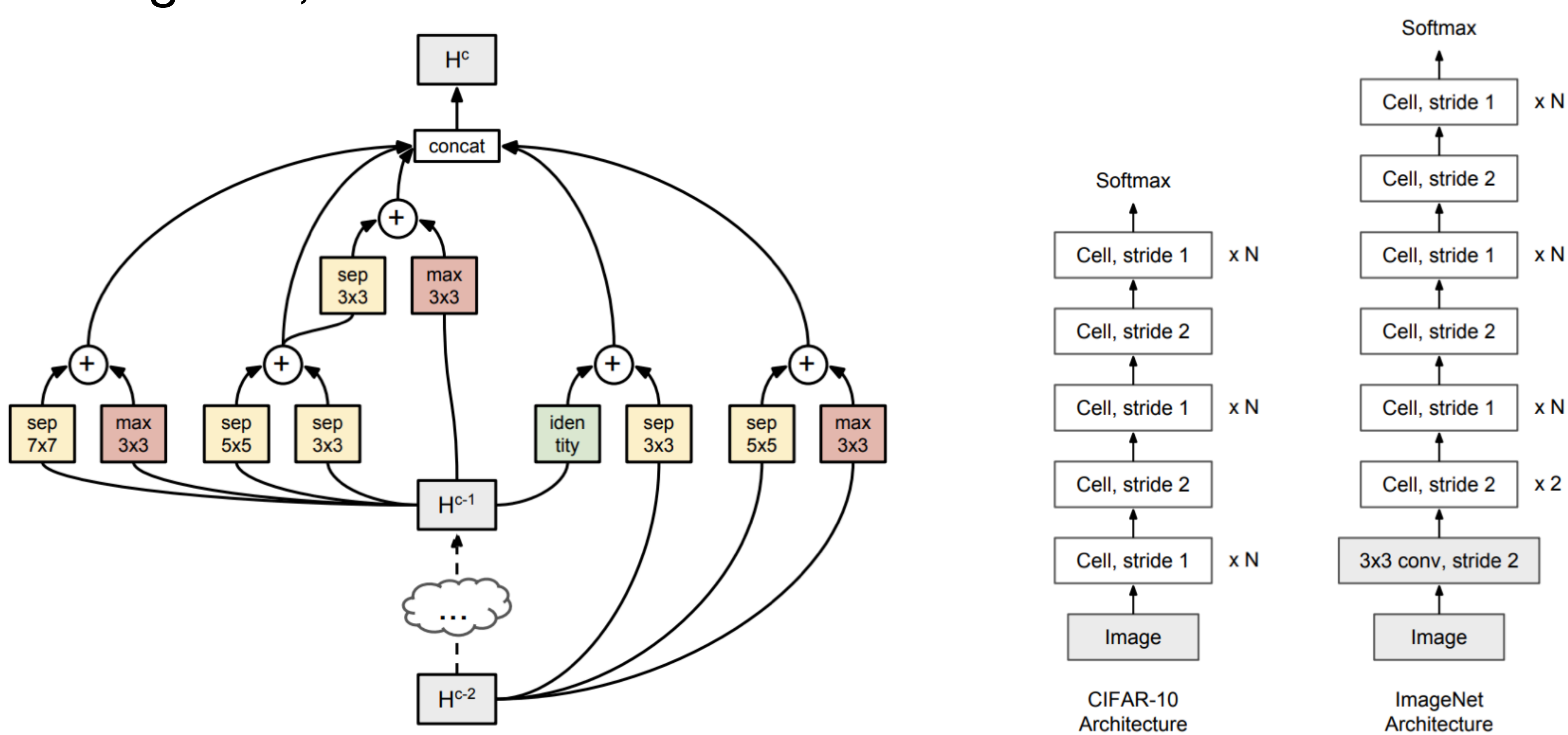
- Kernels to optimize neural nets with GP-based Bayesian optimization
    - ArcNet[2]: DNNs, 23 hyperparameters, 6 kernels
    - NASBOT[3]: DNNs and CNNs



Bayesian Optimization

[1] Mendoza et al. 2016
[2] Zela et al. 2018
[3] Liu et al. 2018

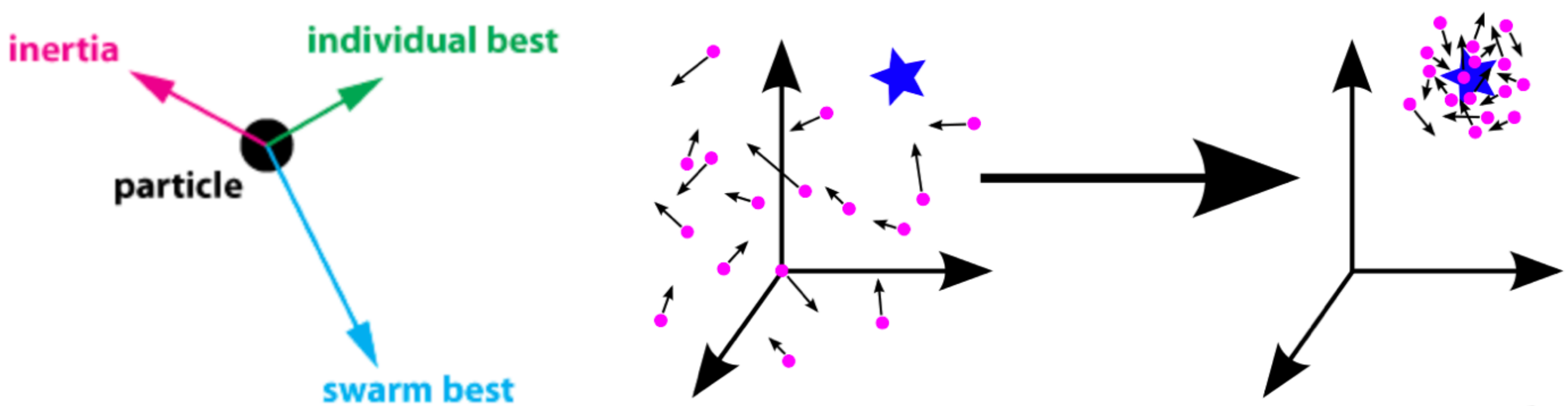
- AutoNet [1] : DNNs, 63 hyperparameters, tuned with SMAC
- Joint NAS + HPO [2]: ResNets, tuned with BO-HB
- PNAS (Progressive NAS) [3]
  - Cell search space, optimized with SMAC, HPO afterwards
  - SotA on ImageNet, CIFAR



Population-based methods

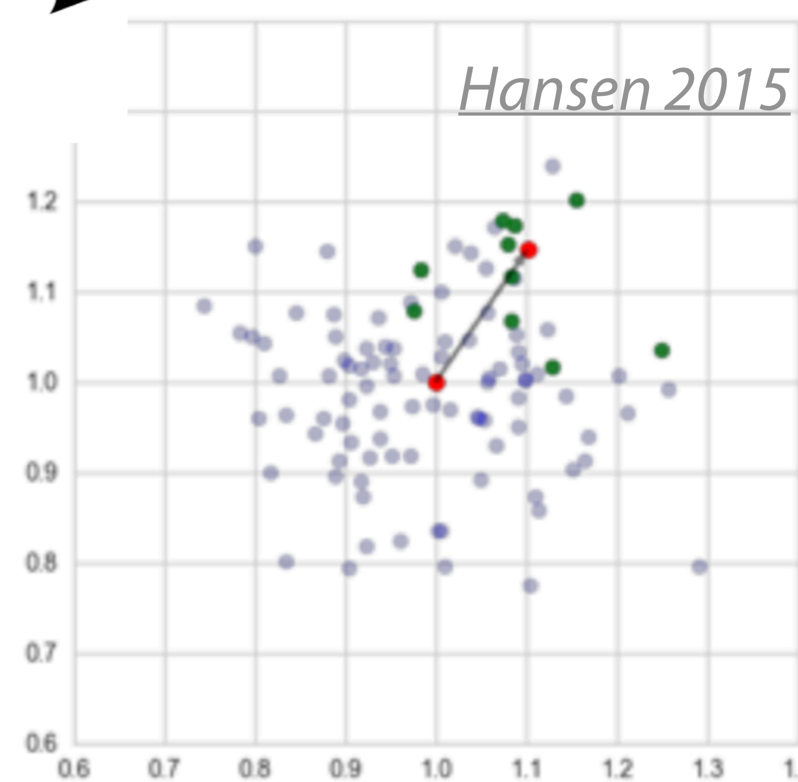
- Less sample efficient, but easy to parallelize, and adapts to changes
- Genetic programming Olson, Moore 2016, 2019
  - Mutations: add, mutate/tune, remove HP

- Particle swarm optimization Mantovani et al 2015



- Covariance matrix adaptation evolution (CMA-ES)
  - Purely continuous, expensive
  - Very competitive to optimize deep neural nets

[Loshilov, Hutter 2016]



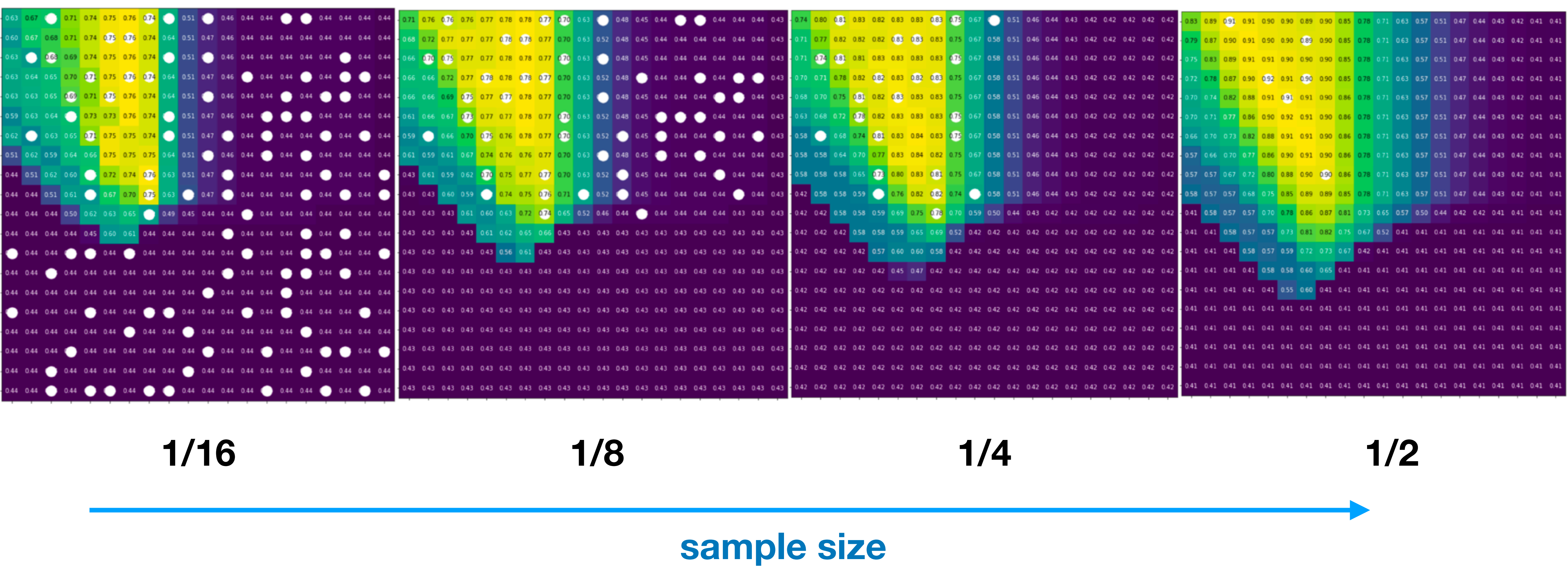
Hyperparameter optimization

Jamieson & Talwalkar 2016

Multi-fidelity optimization

Successive halving:

- train on small subsets, infer which regions may be interesting to evaluate in more depth
- Randomly sample candidates and evaluate on a small data sample
- retrain the 50% best candidates on twice the data

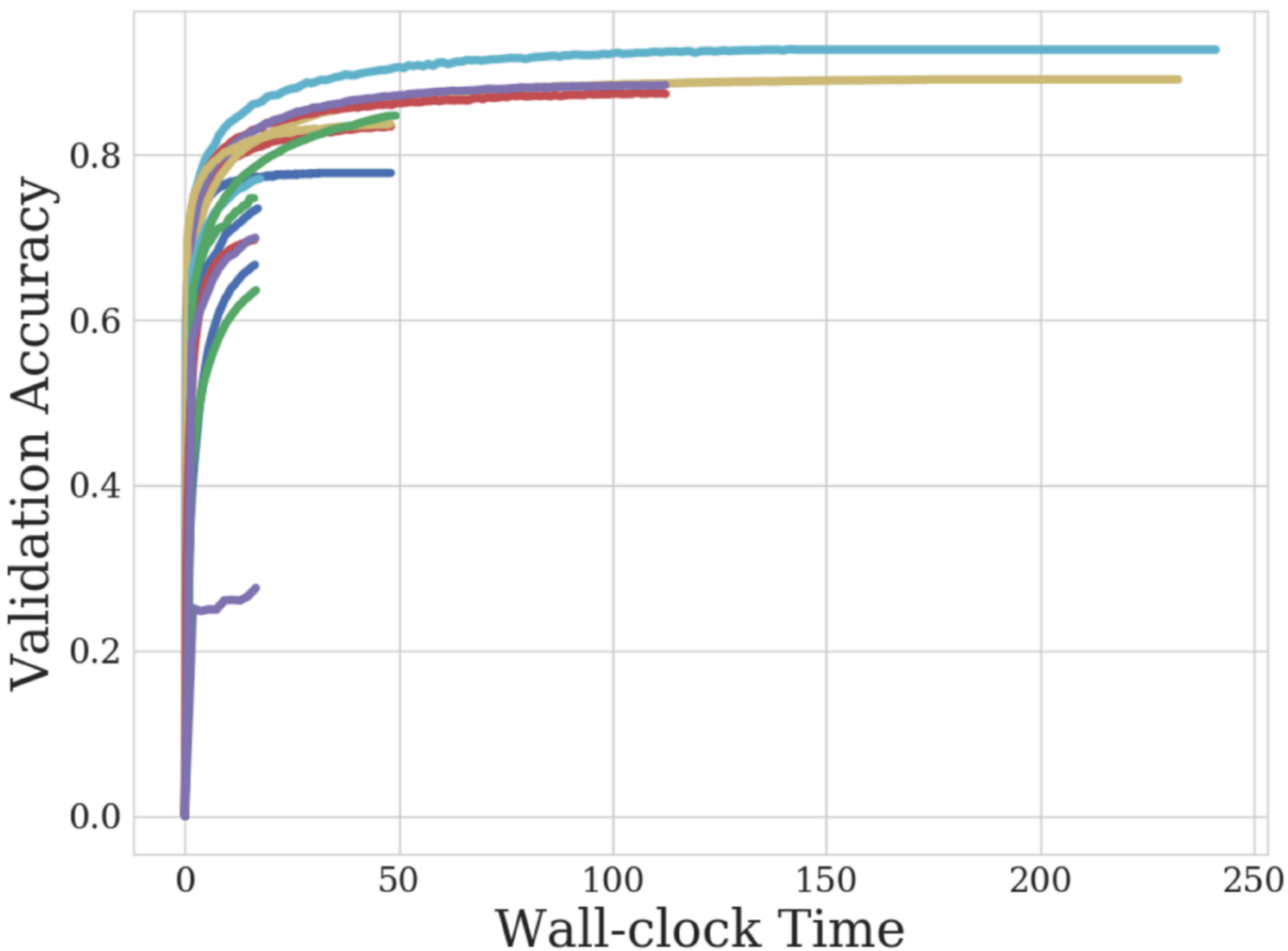


Jamieson & Talwalkar 2016

Multi-fidelity optimization

Successive halving:

- Randomly sample candidates and evaluate on a small data sample
- retrain the best half candidates on twice the data

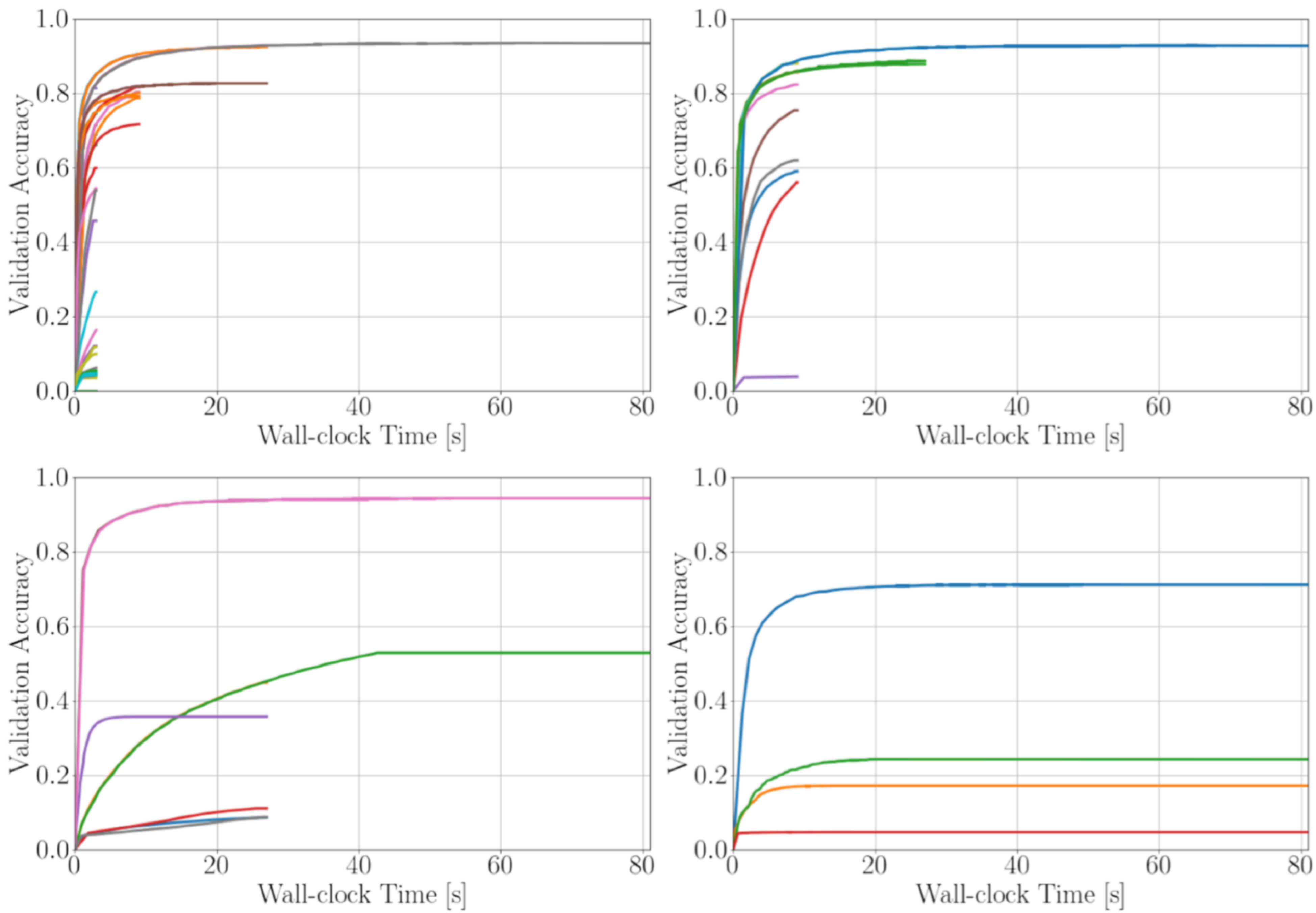


Li et al. 2017

Multi-fidelity optimization

Hyperband (HB): Repeated, decreasingly aggressive successive halving

- Minimizes (doesn’t eliminate) chance that candidate was pruned to early
- Strong anytime performance, easy to implement, scalable, parallelizable

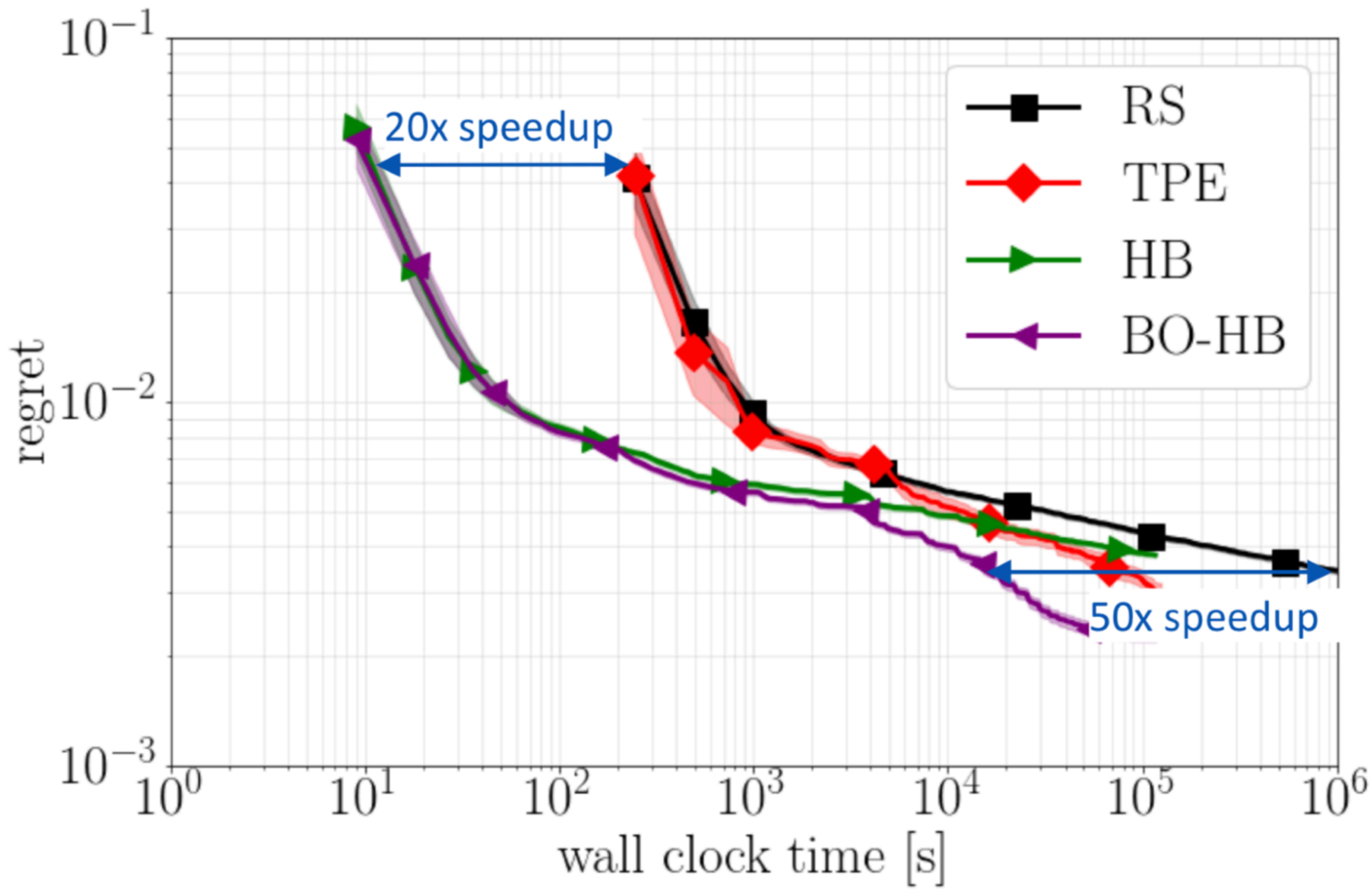


Falkner et al. 2018

Multi-fidelity optimization

Combined Bayesian Optimization and Hyperband (BO-HB)

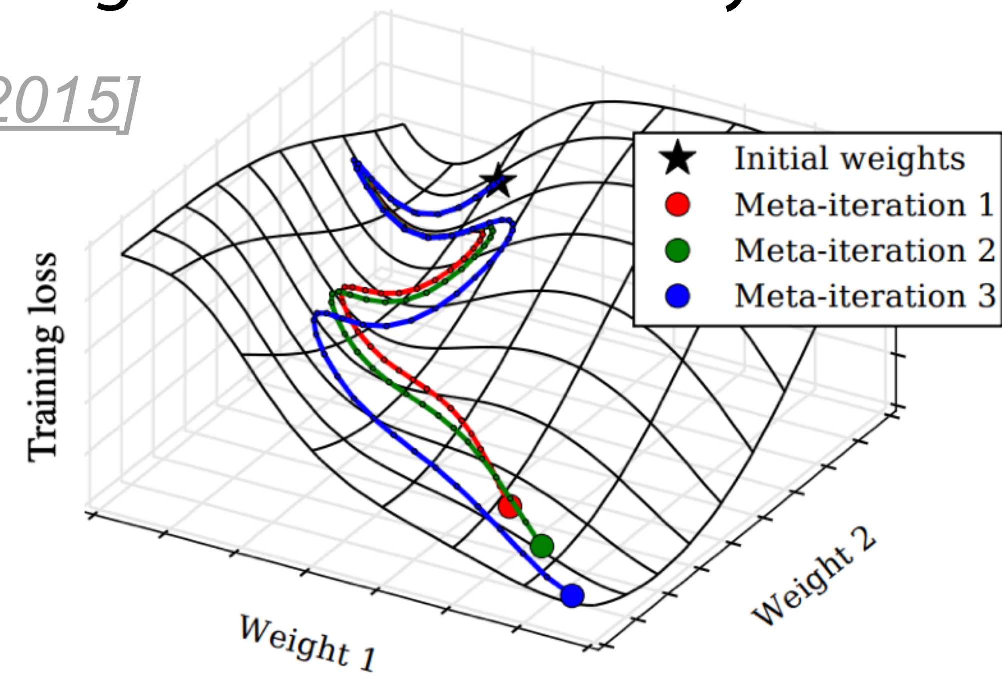
- Choose which configurations to evaluate (with TPE)
- Hyperband: allocate budgets more efficiently
- Strong anytime and final performance



Hyperparameter gradient descent

Optimize neural network hyperparameters and weights simultaneously

- Derive through the entire SGD [MacLaurin et al 2015]
  - Get *hypergradients* wrt. validation loss
  - Expensive! But useful if you have many hyper parameters and high parallelism



- Bilevel program [Franceschi et al 2018]
  - Outer obj.: optimize λ
  - Inner obj.: optimize weights given λ
    - Typically approximated

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

- Interleave optimization steps [Luketina et al 2016]
  - Alternate SGD steps for ω and λ

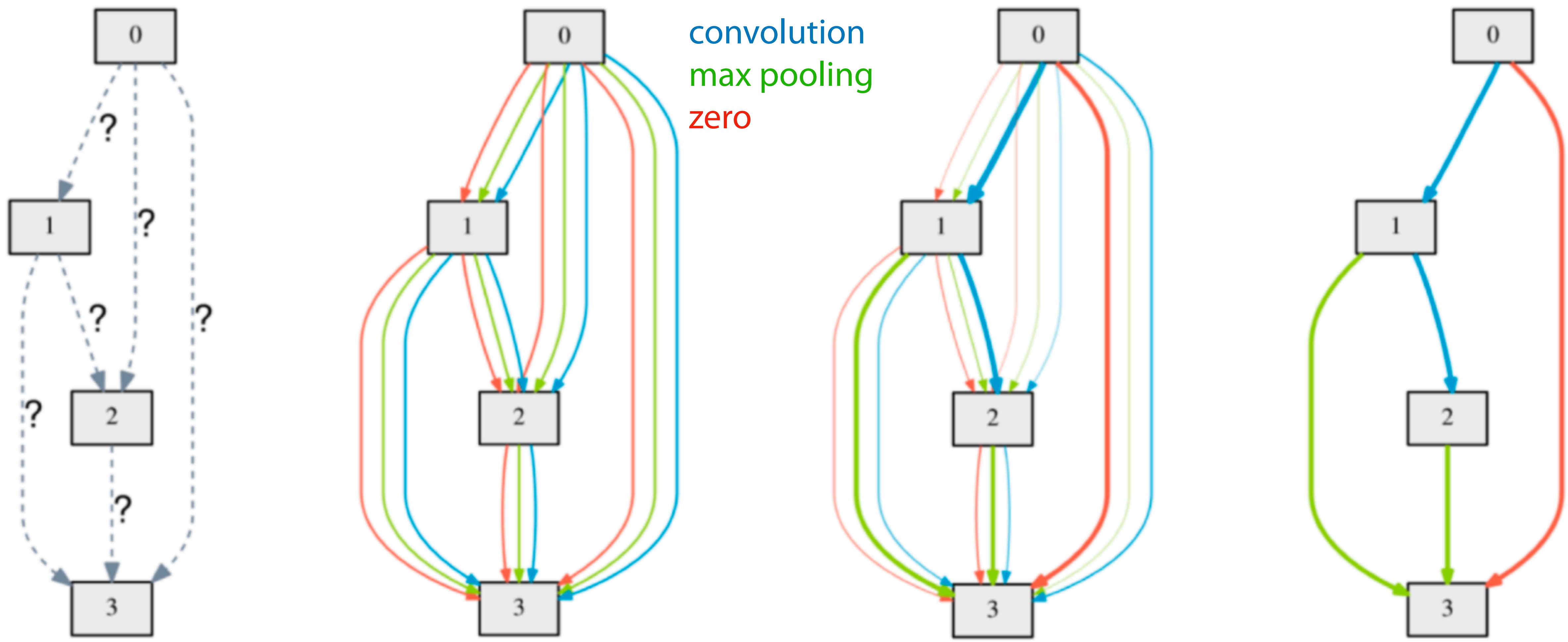
Hyperparameter gradient step w.r.t. $\nabla_{\lambda}\mathcal{L}_{val}$

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

Liu et al. 2018

DARTS: Differentiable NAS

- Fixed (one-shot) structure, learn which operators to use
- Give all operators a weight α_i
- Optimize α_i and model weights ω_j using bilevel optimization
  - approximate $\omega_j^*(\alpha_i)$ by adapting ω_j after every training step



One-shot model **operator weights α_i** **interleaved optimization of α_i and ω_j with SGD** **argmax α_i**

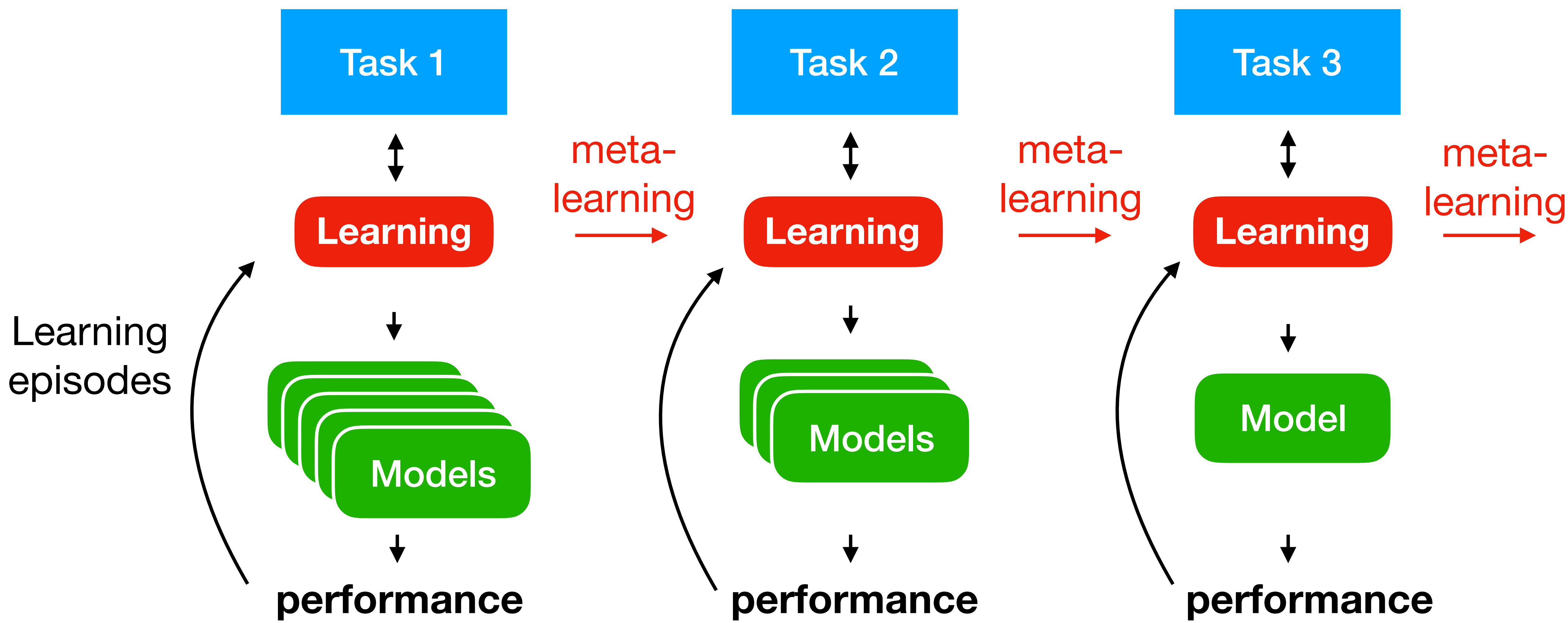
Learning is a never-ending process

Humans don't learn from scratch

Learning is a never-ending process

Humans learn *across* tasks

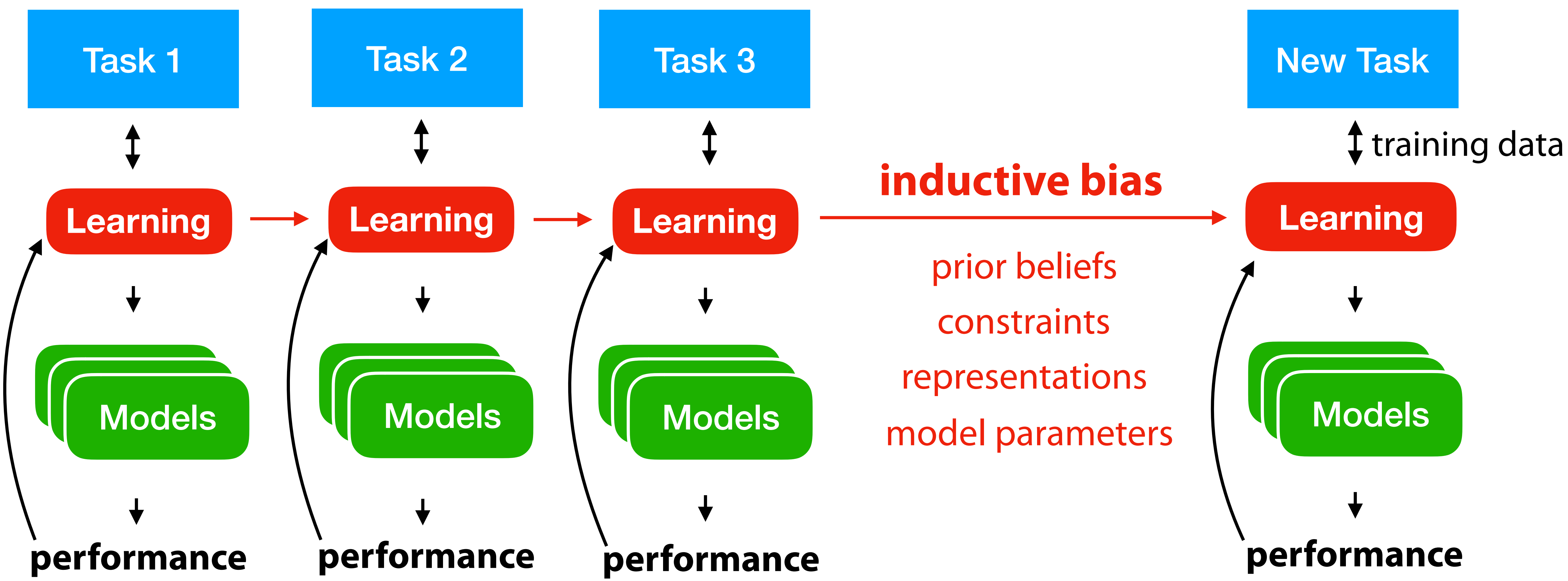
Why? Requires less trial-and-error, less data



Learning to learn

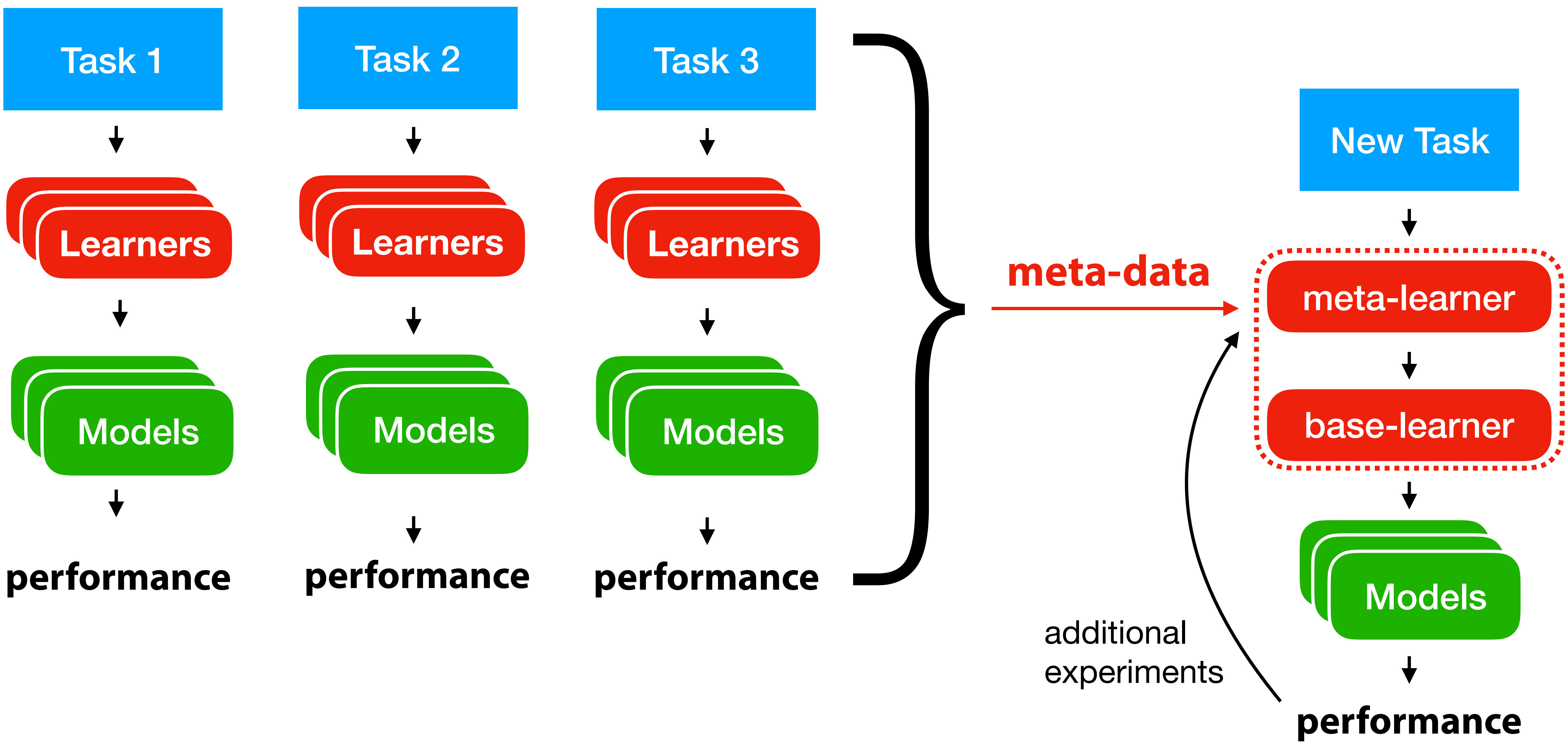
If prior tasks are *similar*, we can ***transfer*** prior knowledge to new tasks

Inductive bias: assumptions added to the training data to learn effectively



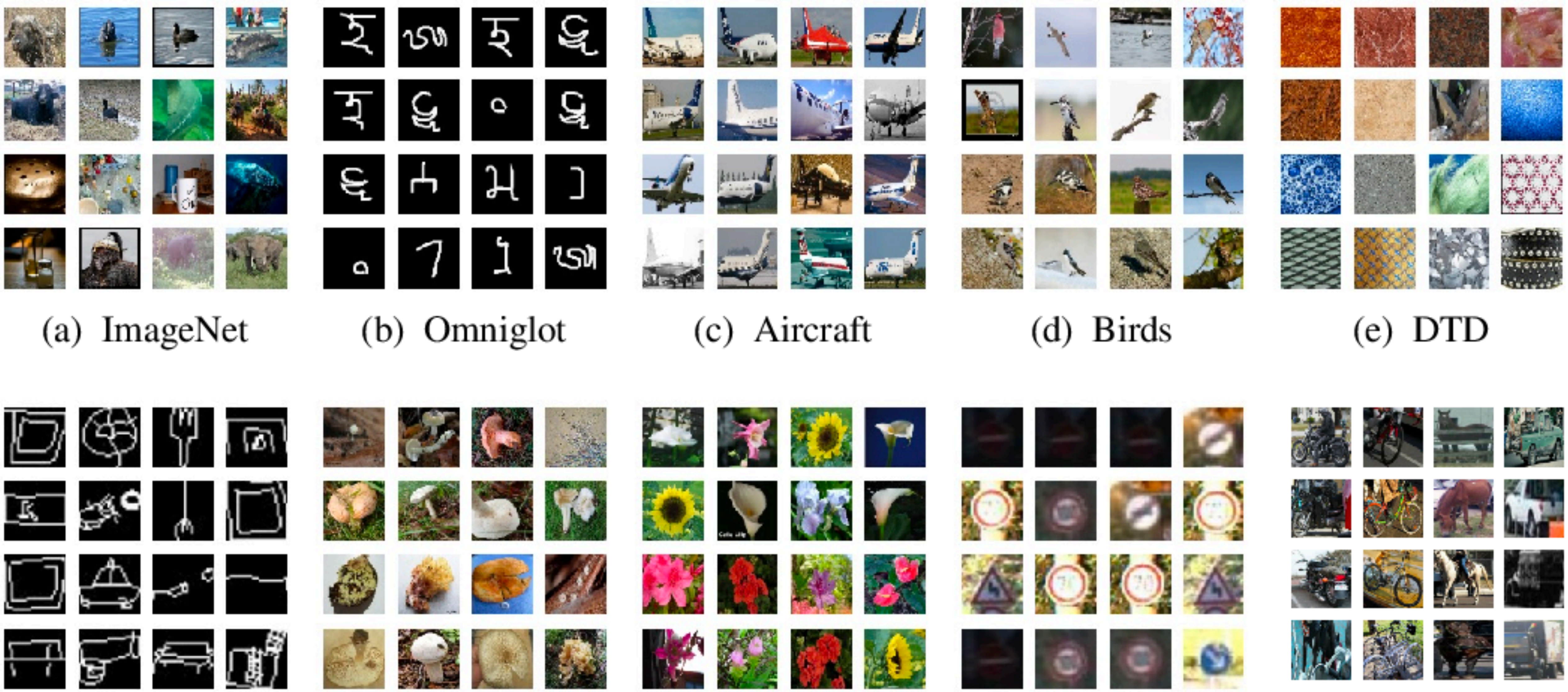
Meta-learning

Meta-learner *learns* a (base-)learning algorithm, based on *meta-data*



Triantafillou et al. 2019

Example: meta-dataset



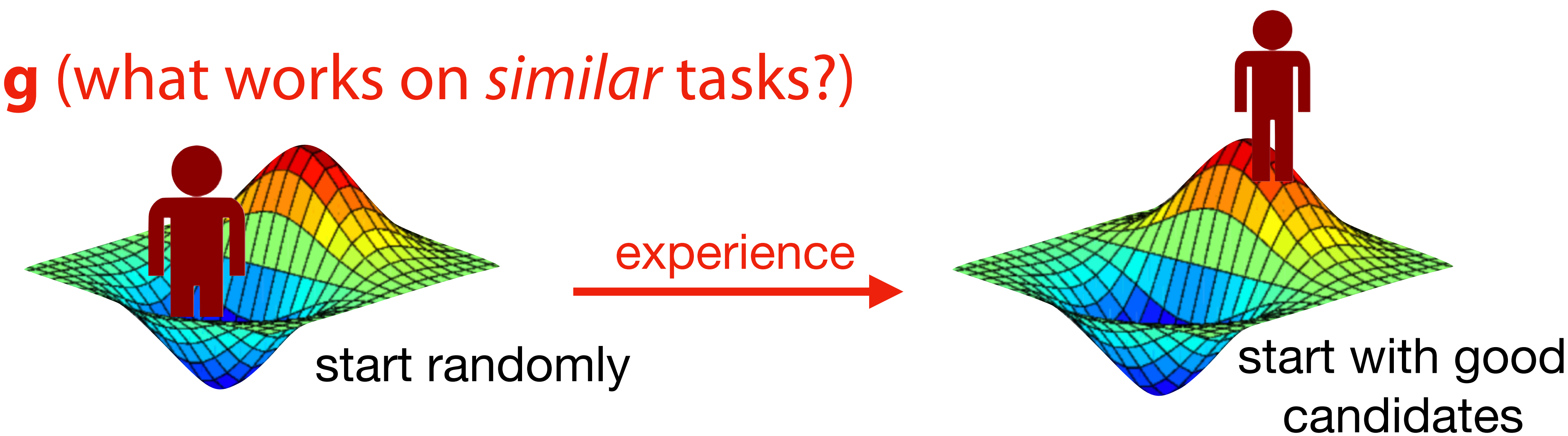
(a) ImageNet (b) Omniglot (c) Aircraft (d) Birds (e) DTD

(f) Quick Draw (g) Fungi (h) VGG Flower (i) Traffic Signs (j) MSCOCO

How?

Search space design (what works in general?)

Warm starting (what works on *similar* tasks?)

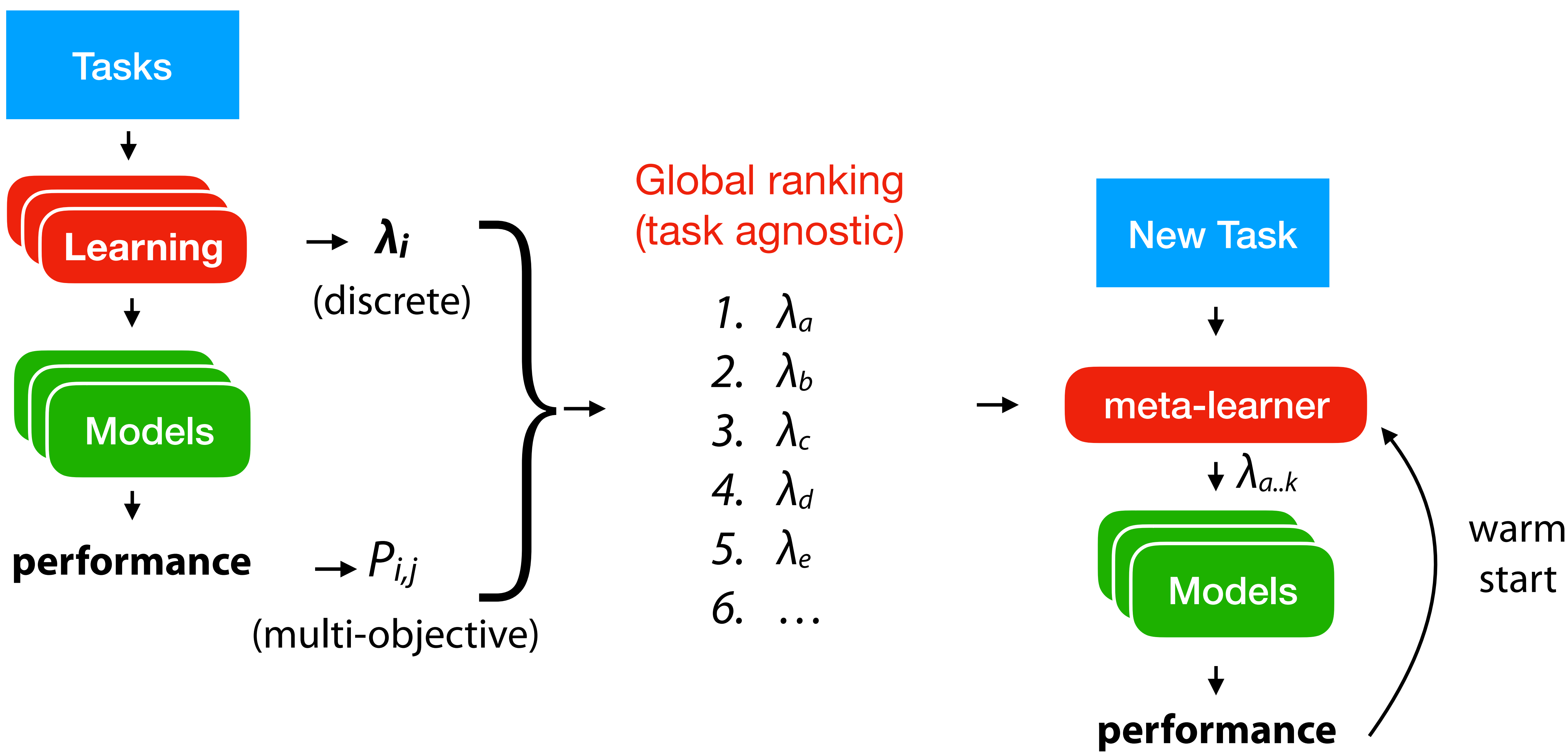


Model transfer (reuse on very similar tasks)

Rankings

Leite et al. 2012
Abdulrahman et al. 2018

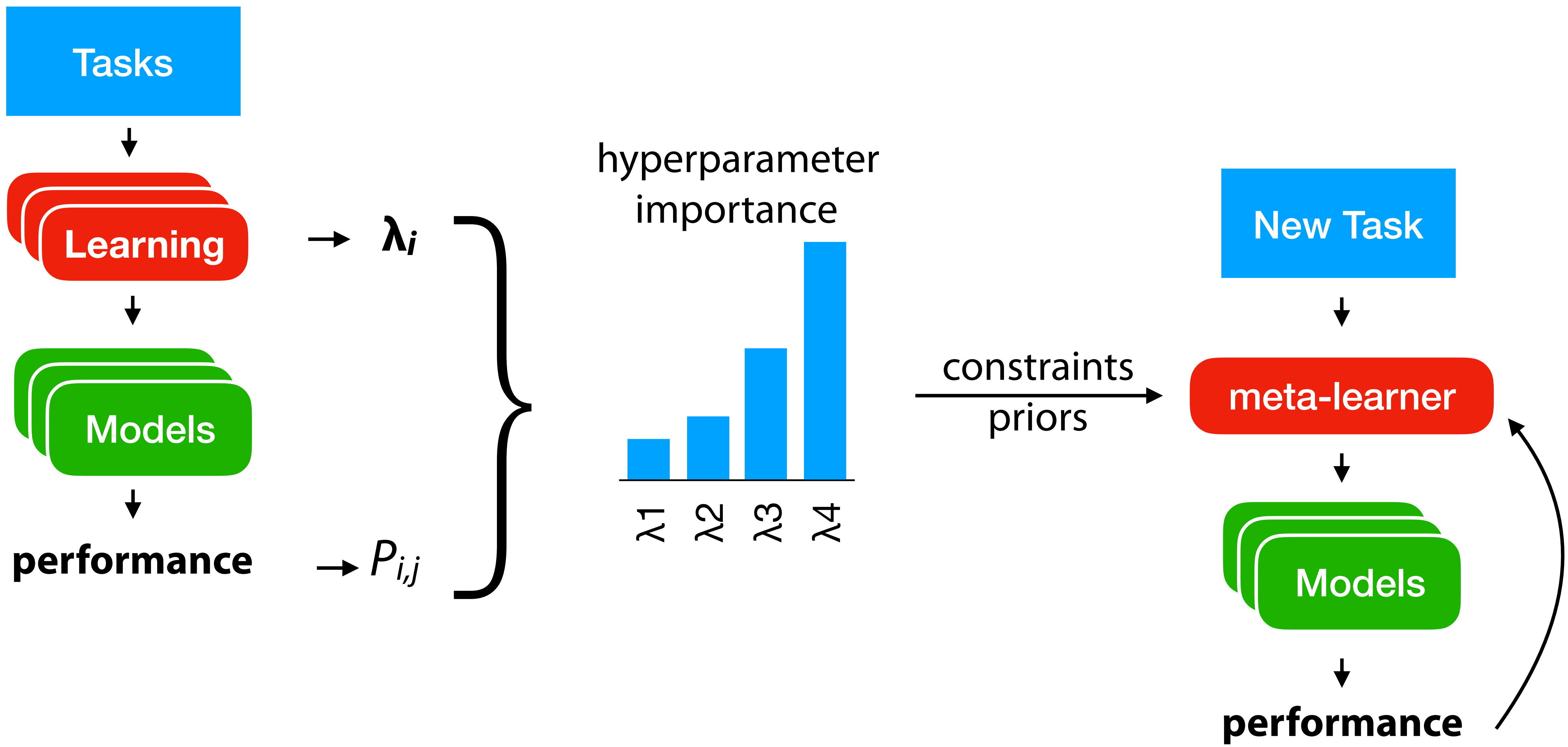
- Build a *global (multi-objective) ranking,* recommend the top-K
- Can be used as a *warm start* for optimization techniques
  - E.g. Bayesian optimization, evolutionary techniques,…



[1] van Rijn & Hutter 2018

Hyperparameter importance

- **Functional ANOVA** [1]
  Select hyperparameters that cause variance in the evaluations.

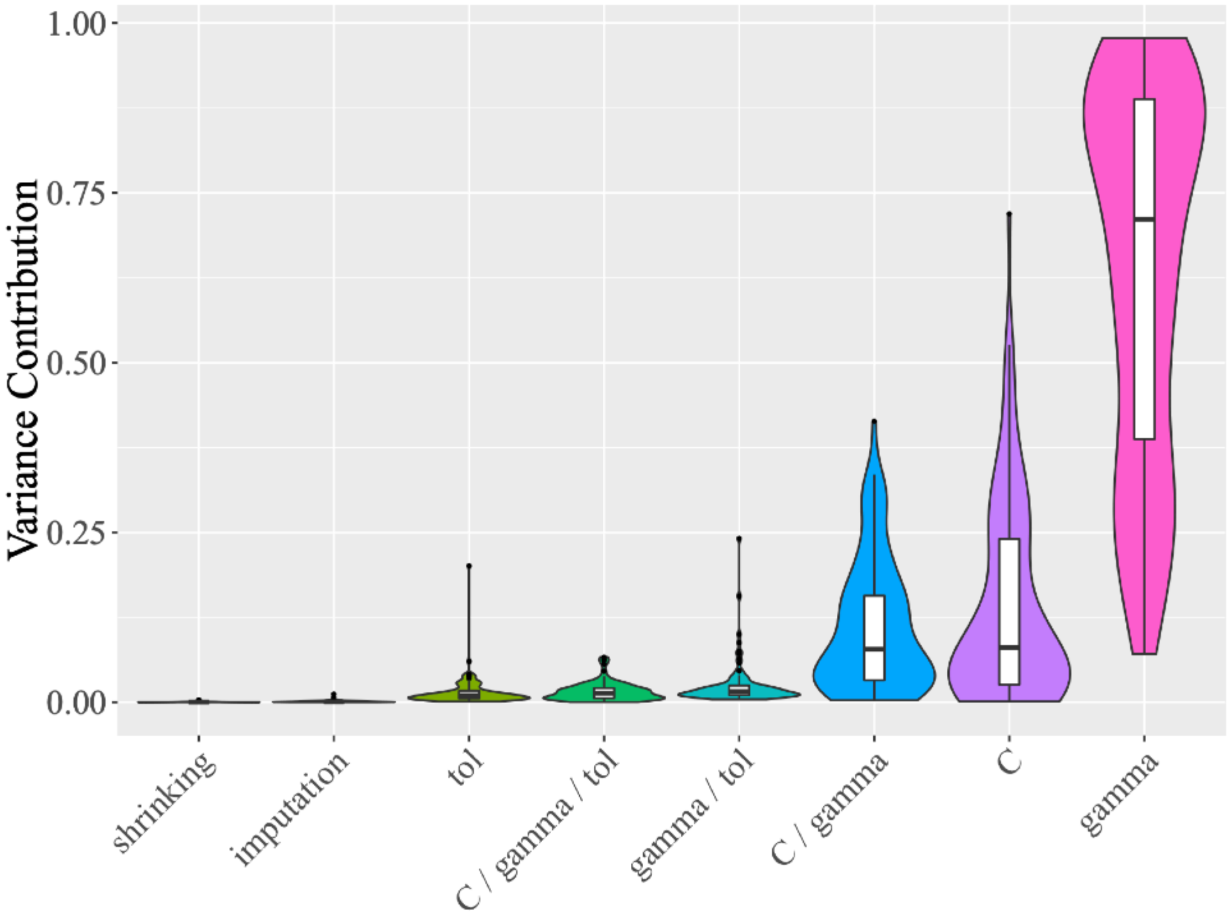


Meta-Learning

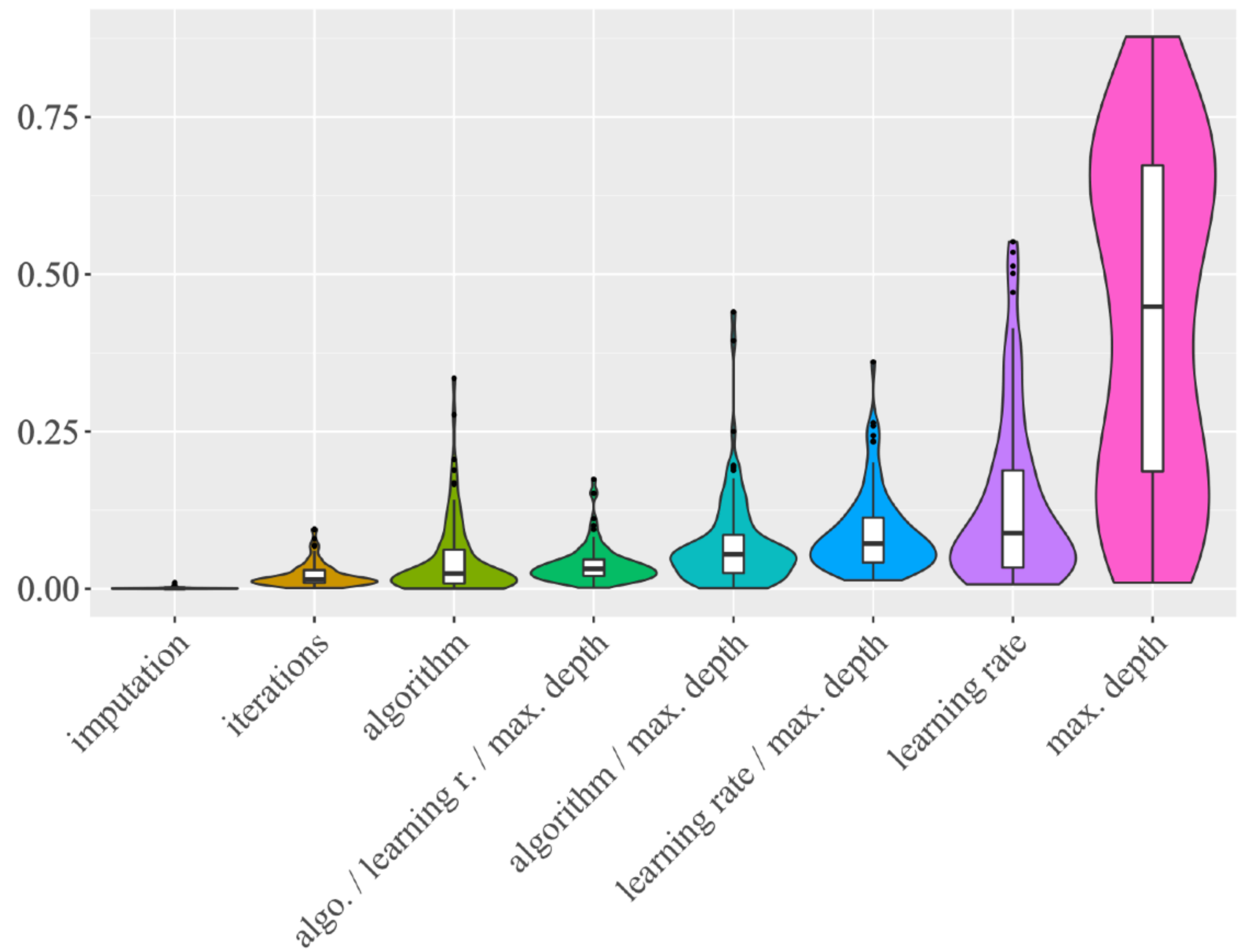
[1] van Rijn & Hutter 2018

Hyperparameter importance

- **Functional ANOVA** [1]
  Select hyperparameters that cause variance in the evaluations.



SVM (RBF)

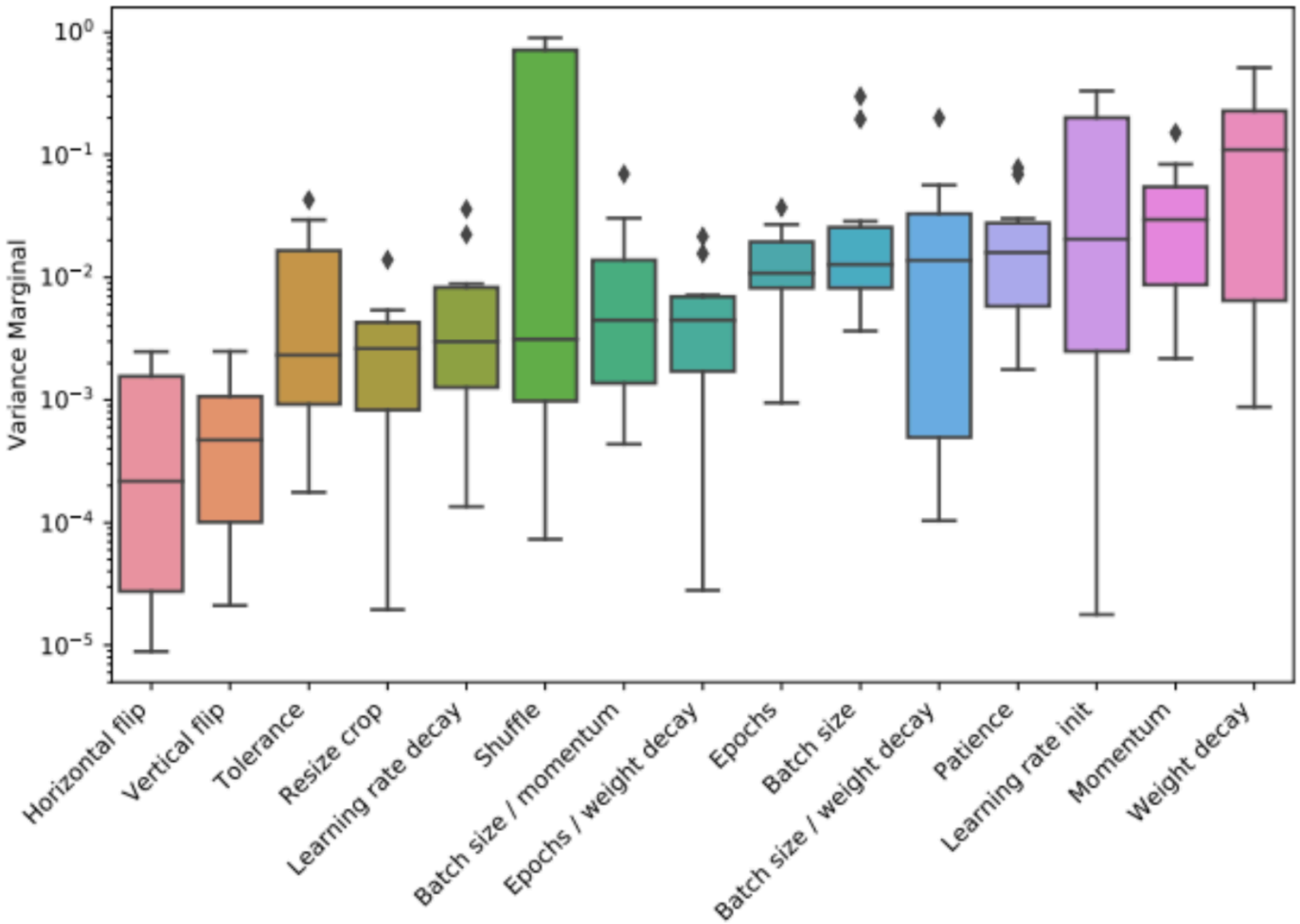


AdaBoost

[1] van Rijn & Hutter 2018

Hyperparameter importance

- **Functional ANOVA** [1]
  Select hyperparameters that cause variance in the evaluations.



ResNets for image classification

Hyperparameter importance

[1] van Rijn & Hutter 2018

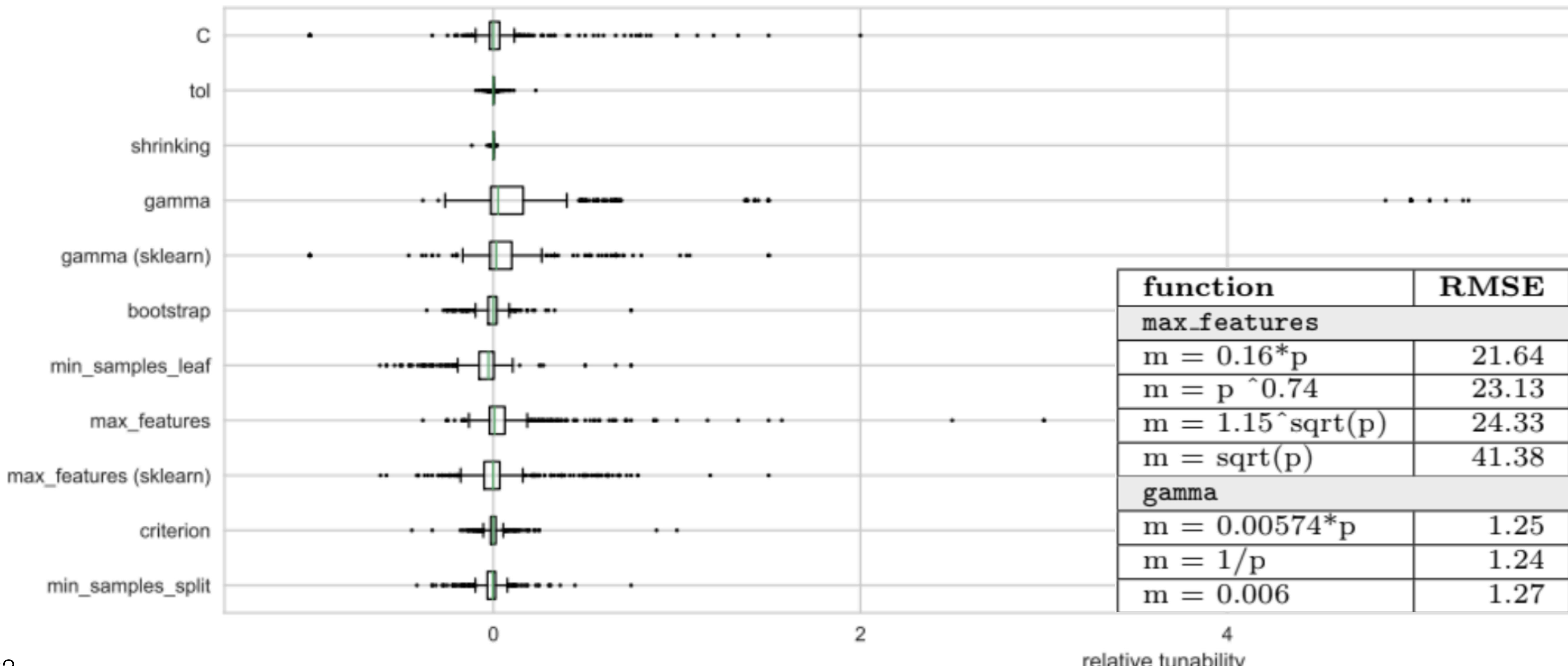
[2] Probst et al. 2018

[3] Weerts et al. 2018

[4] van Rijn et al. 2018

- **Functional ANOVA** [1]
  Select hyperparameters that cause variance in the evaluations.

- **Tunability** [2,3,4]
  Learn good defaults, measure improvement from tuning over defaults



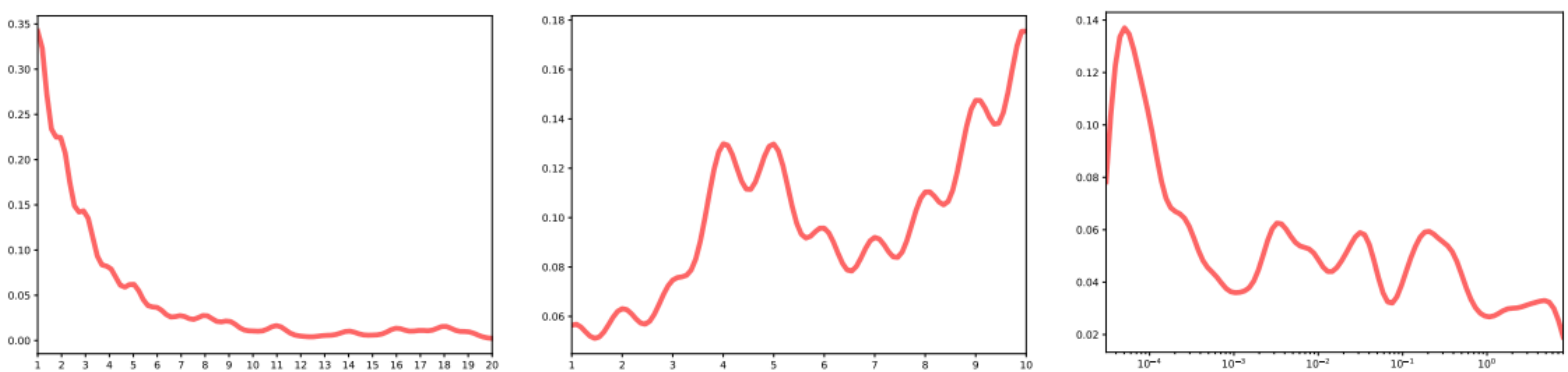
function	RMSE
max_features	
m = 0.16*p	21.64
m = p ^0.74	23.13
m = 1.15^sqrt(p)	24.33
m = sqrt(p)	41.38
gamma	
m = 0.00574*p	1.25
m = 1/p	1.24
m = 0.006	1.27

Meta-Learning

van Rijn & Hutter 2018

Hyperparameter priors

- Select hyperparameters that cause variance in the evaluations.
  Learn priors for hyperparameter tuning (e.g. Bayesian optimization)



(a) RF: min. samples per leaf (b) Adaboost: max. depth of tree (c) SVM (RBF kernel): gamma

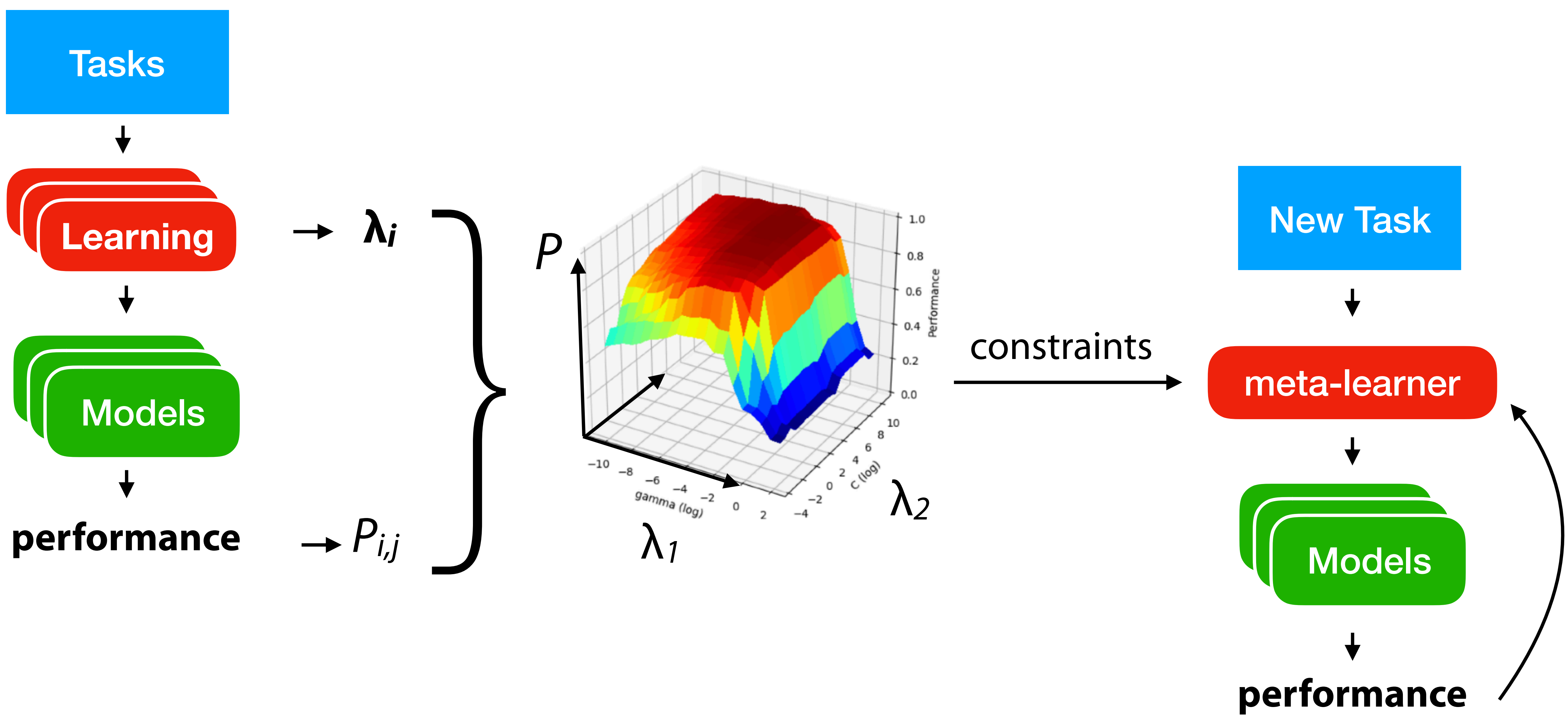
Priors (KDE) for the most important hyperparameters

Meta-Learning

Wistuba et al. 2015

Hyperparameter priors

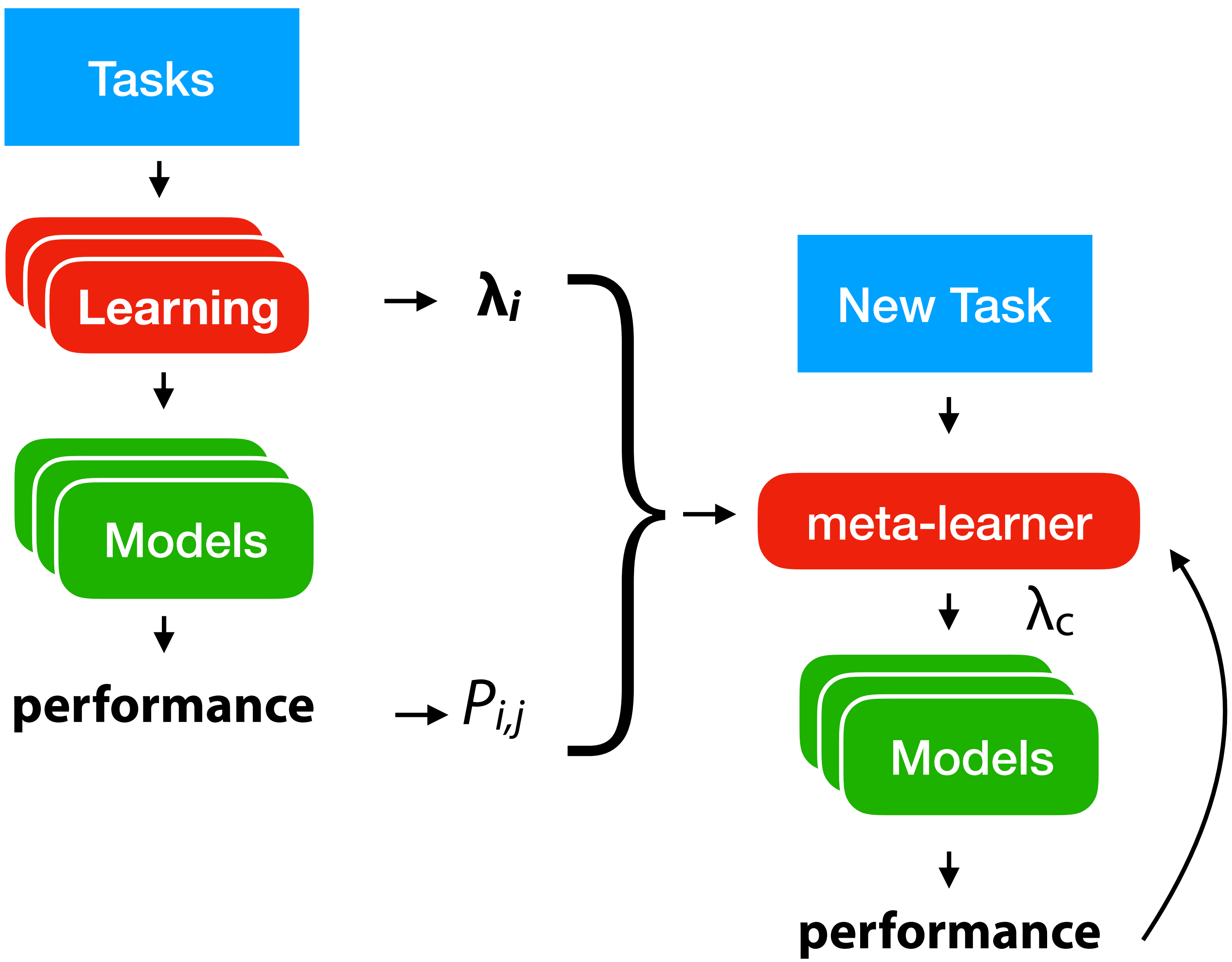
- **Search space pruning**
  Exclude regions yielding bad performance on (similar) tasks



Leite et al. 2012

Learning task similarity

- Experiments on the *new* task can tell us how it is similar to *previous* tasks
- **Task are *similar*** if observed *performance* of configurations is similar
- Use this to recommend new configurations, run, repeat *(active testing)*

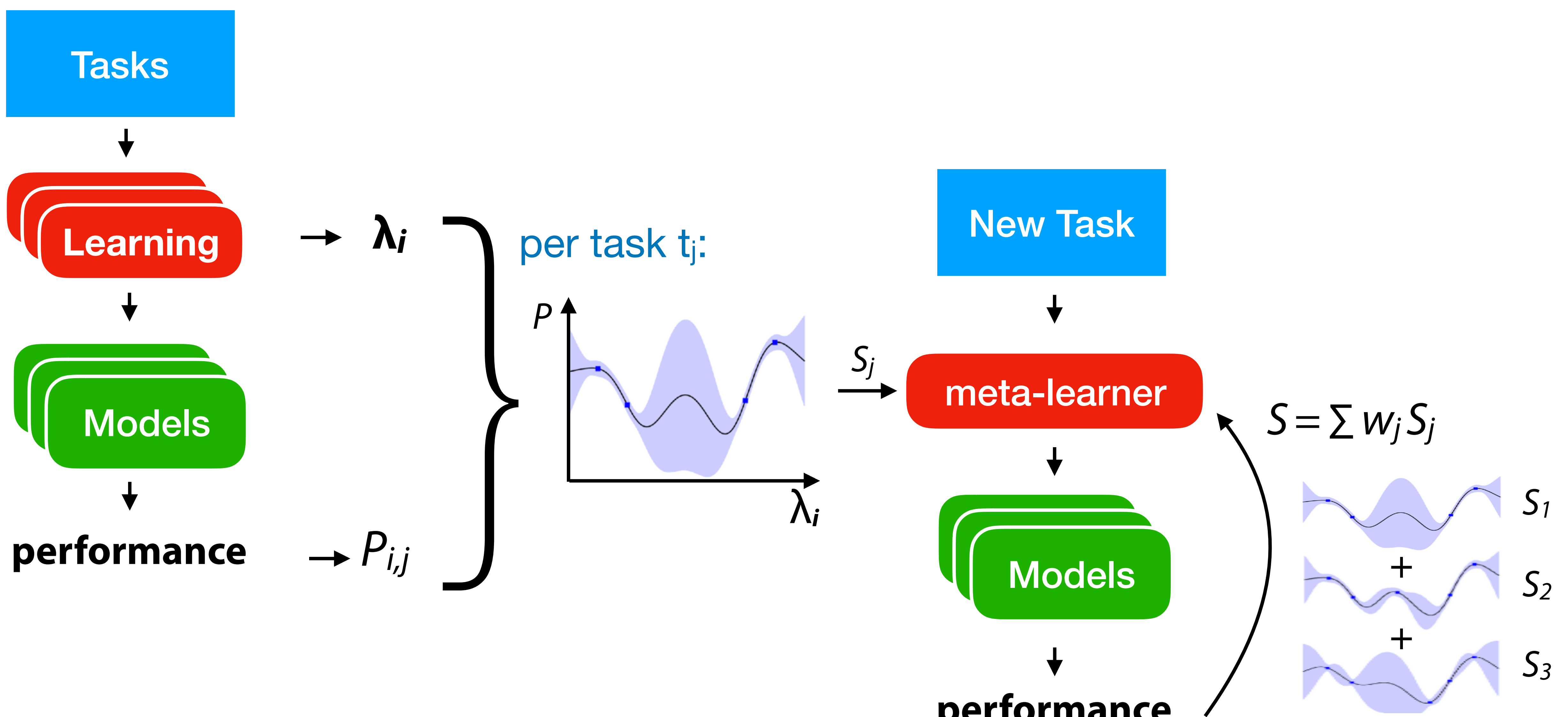


$$P_{c,new} \cong P_{c,j}$$

Surrogate model transfer

[1] Wistuba et al. 2018
[2] Feurer et al. 2018

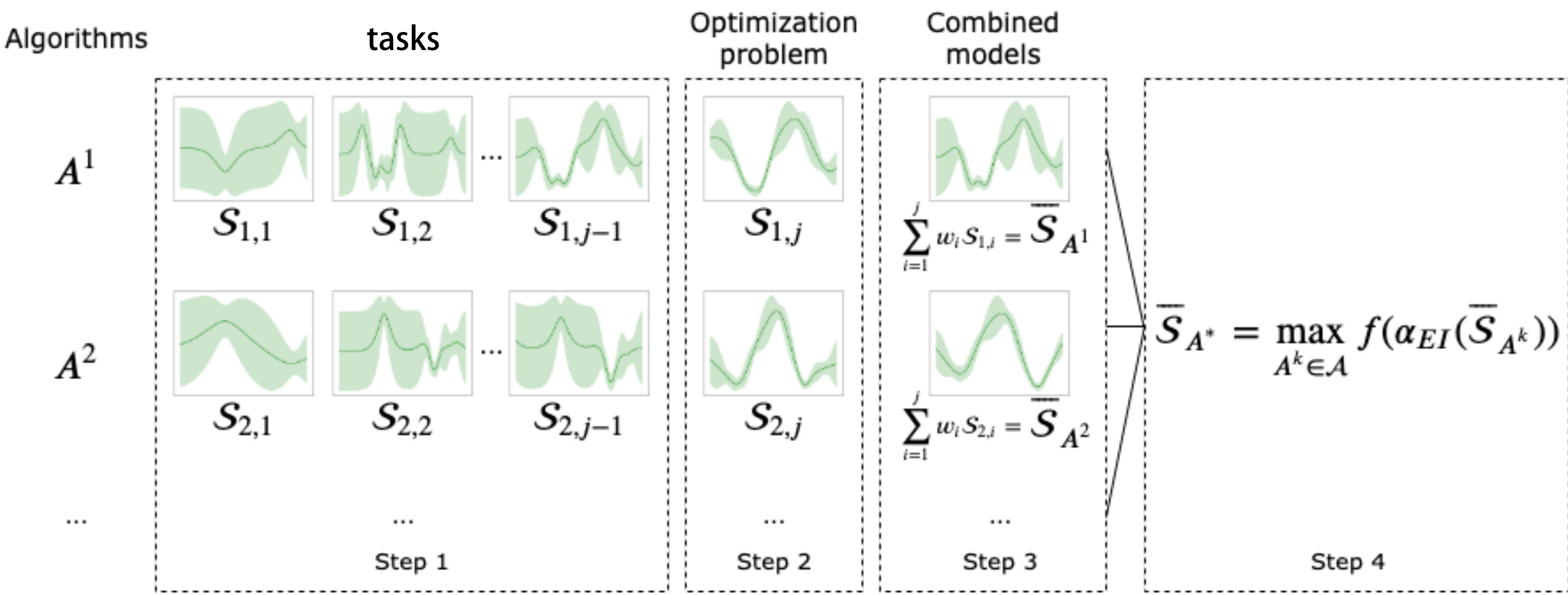
- If task j is *similar* to the new task, its surrogate model S_j will likely transfer well
- Sum up all S_j predictions, weighted by task similarity (as in active testing)[1]
- Build combined Gaussian process, *weighted by current performance* on new task[2]



Feuer et al., ICML 2018
Manolache & Vanschoren, NeurIPS 2019

Surrogate Model Transfer

- Store surrogate model S_{ij} for every pair of task i and algorithm j
- Simpler surrogates, better transfer
- Learn weighted ensemble -> significant speed up in optimization

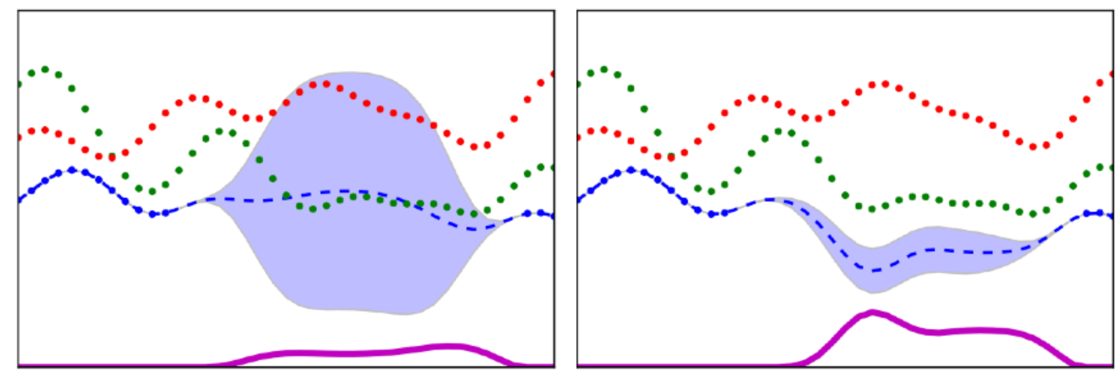


Meta-Learning

Multi-task Bayesian optimization

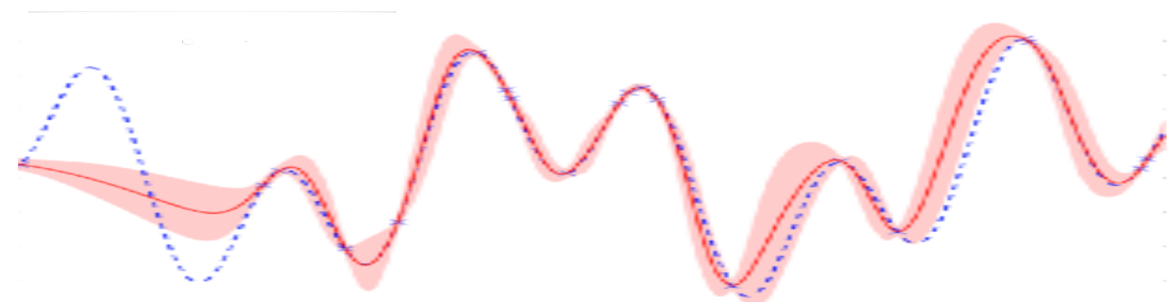
[1] Swersky et al. 2013
[2] Springenberg et al. 2016
[3] Golovin et al. 2017

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

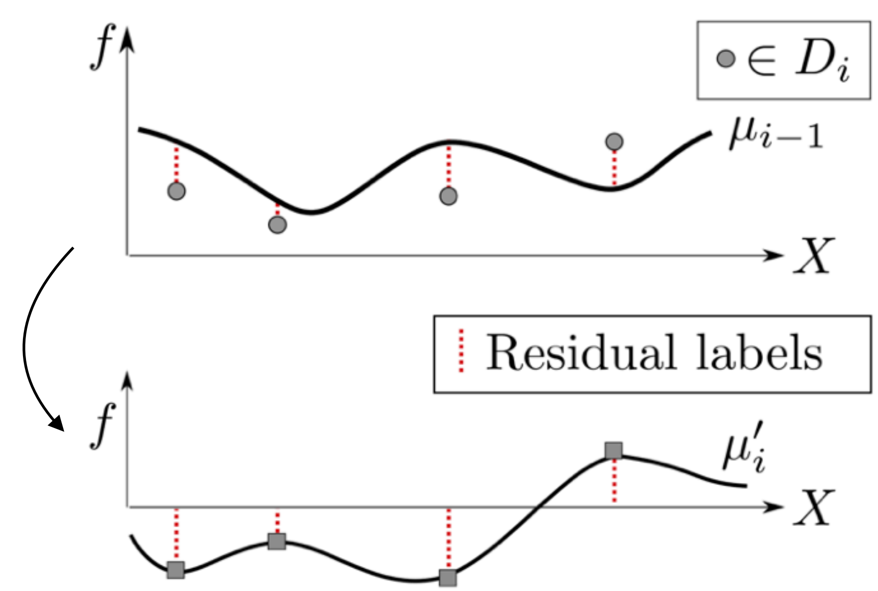


Independent GP predictions   Multi-task GP predictions

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



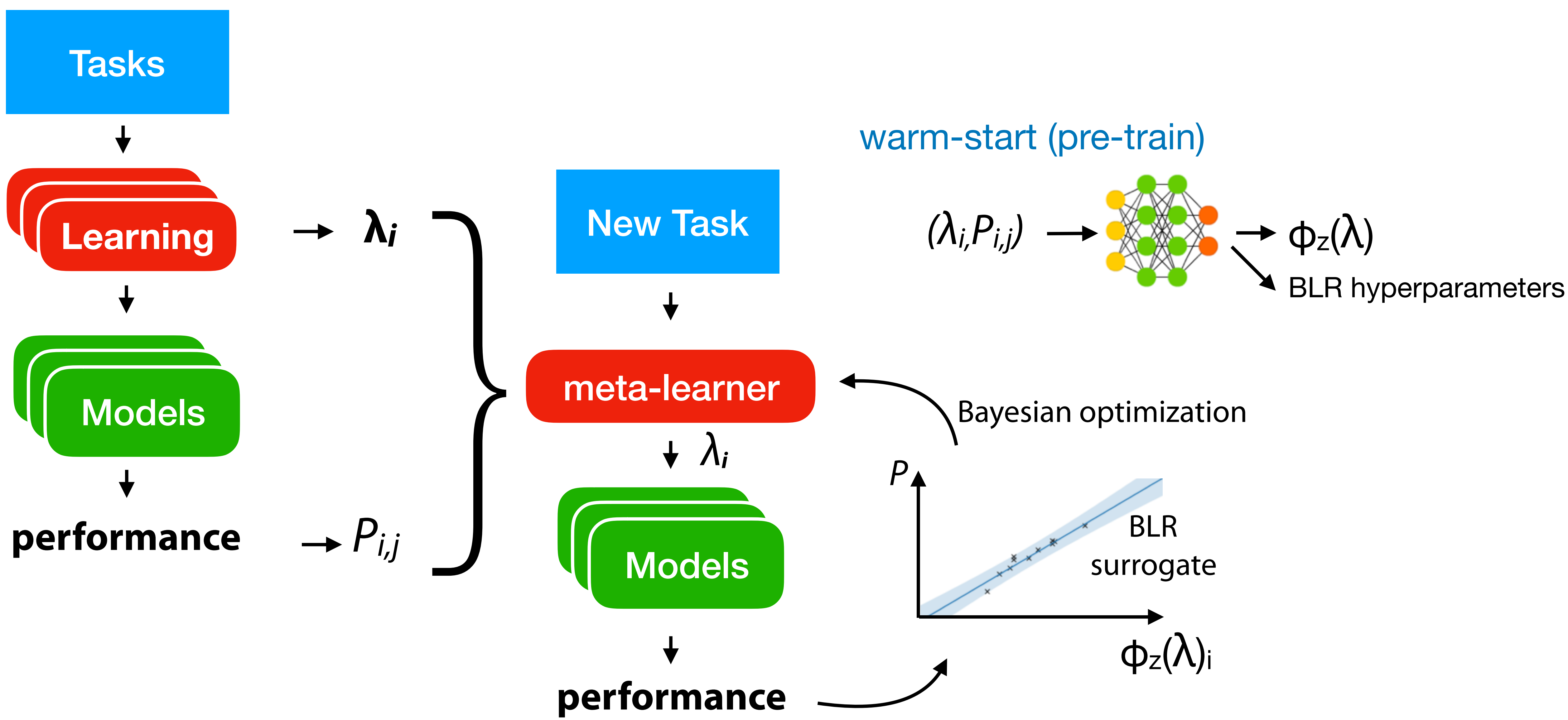
- **Stacking** Gaussian Process regressors (Google Vizier)[3]
  - Continual learning (sequential similar tasks)
  - Transfers a prior based on residuals of previous GP



Perrone et al. 2018

Learn basis expansions for hyperparameters

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



Meta-features

[1] Vanschoren 2018

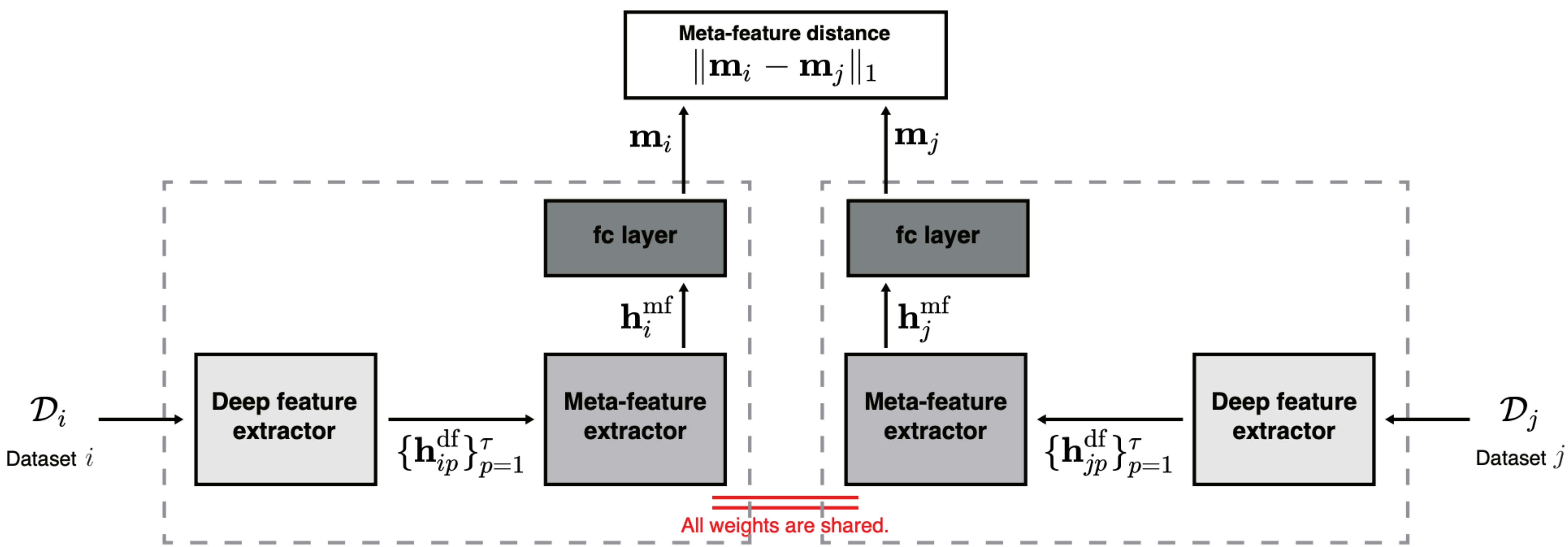
[2] Rivolli et al 2019

- Task descriptions, allow to compute a distance between task
- **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
- Optimized (recommended) representations exist [2]

Meta-features

[1] Kim et al. 2017
[2] Achille et al. 2019
[2] Zamir et al. 2018

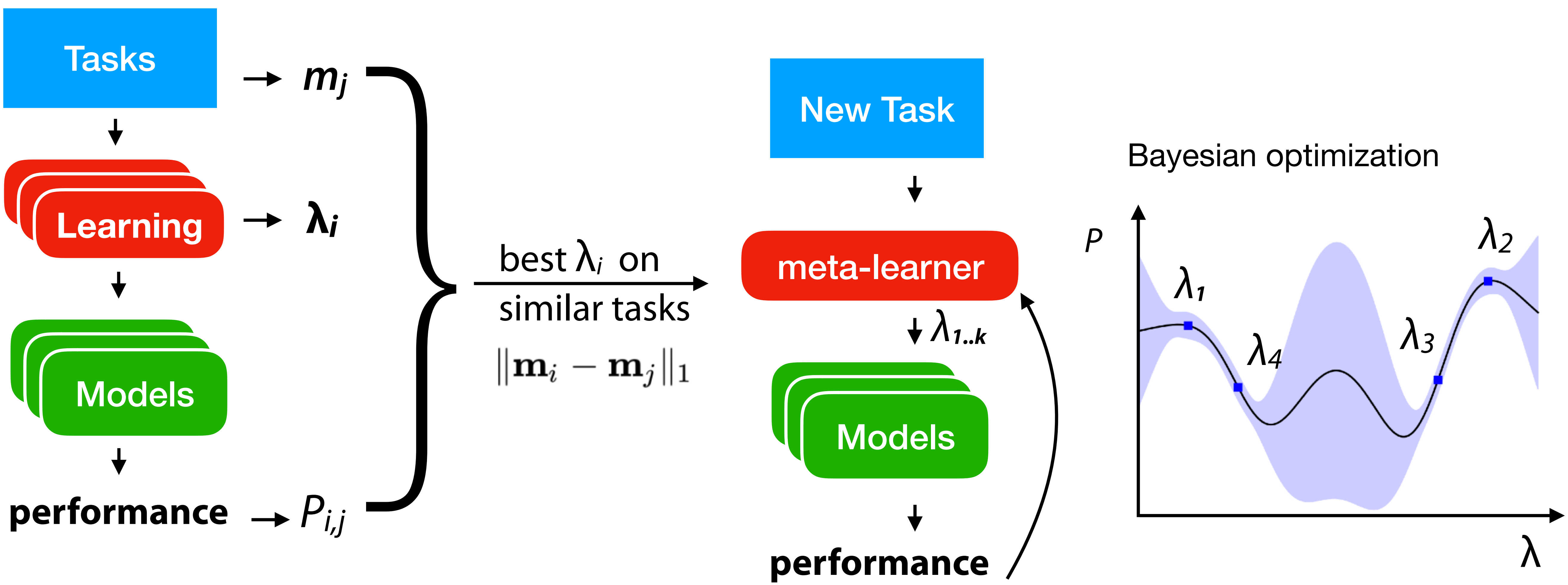
- ***Learning* a joint task representation**
  - Deep metric learning: learn a representation *h^mf^* using ground truth distance
    - Siamese Network: similar task, similar representation [1]
    - Feed through pretrained CNN, extract vector from weights [2]
  - Recommend neural architectures, or warm-start neural architecture search
  - Ground truth can be obtained by brute force evaluation, e.g. taskonomy [3]



Feurer et al. 2015

Warm-starting from similar tasks

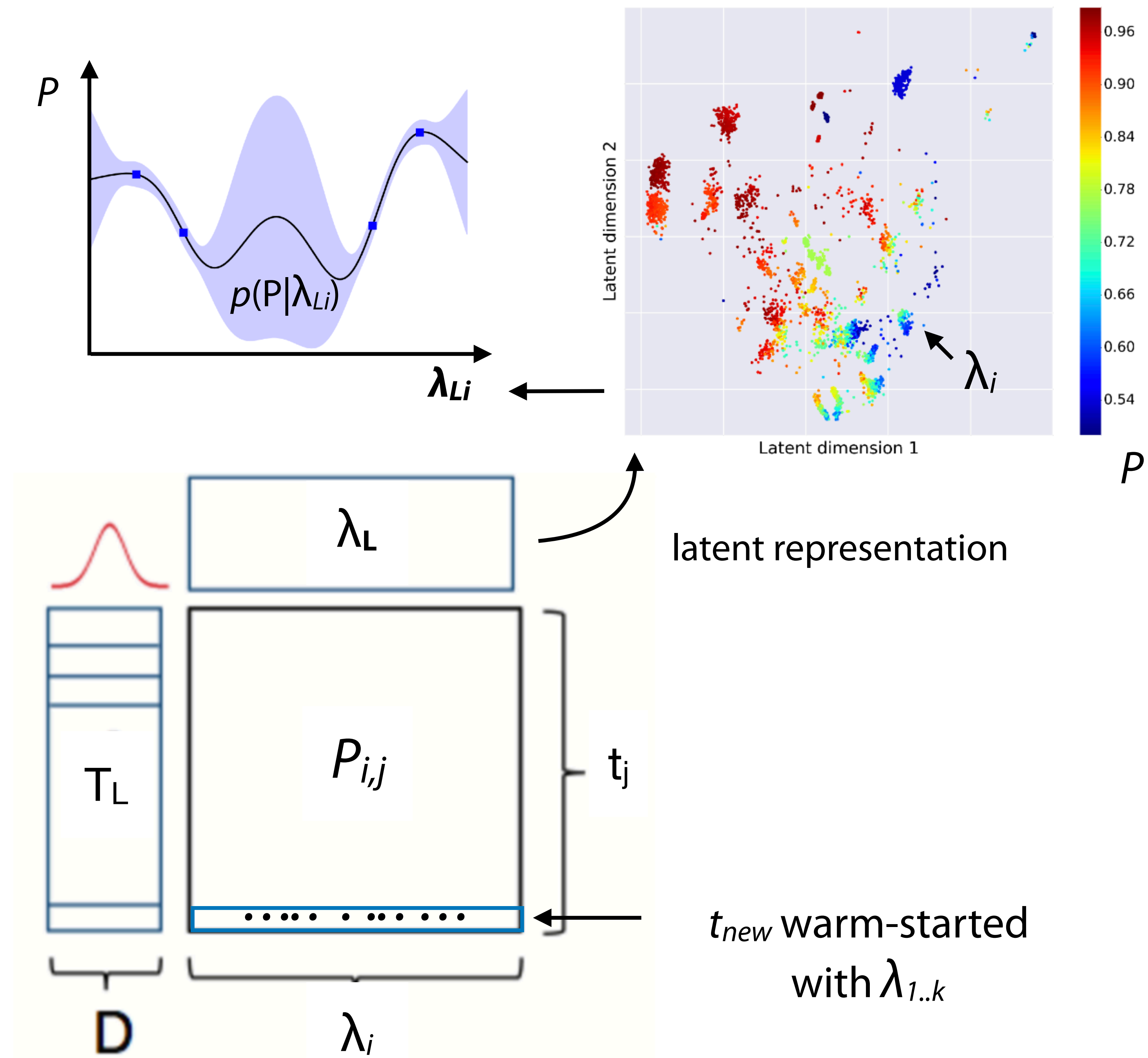
- Find k most similar tasks, warm-start search with best λ_i
  - Auto-sklearn: Bayesian optimization (SMAC)
    - Meta-learning yield better models, faster
    - Winner of AutoML Challenges

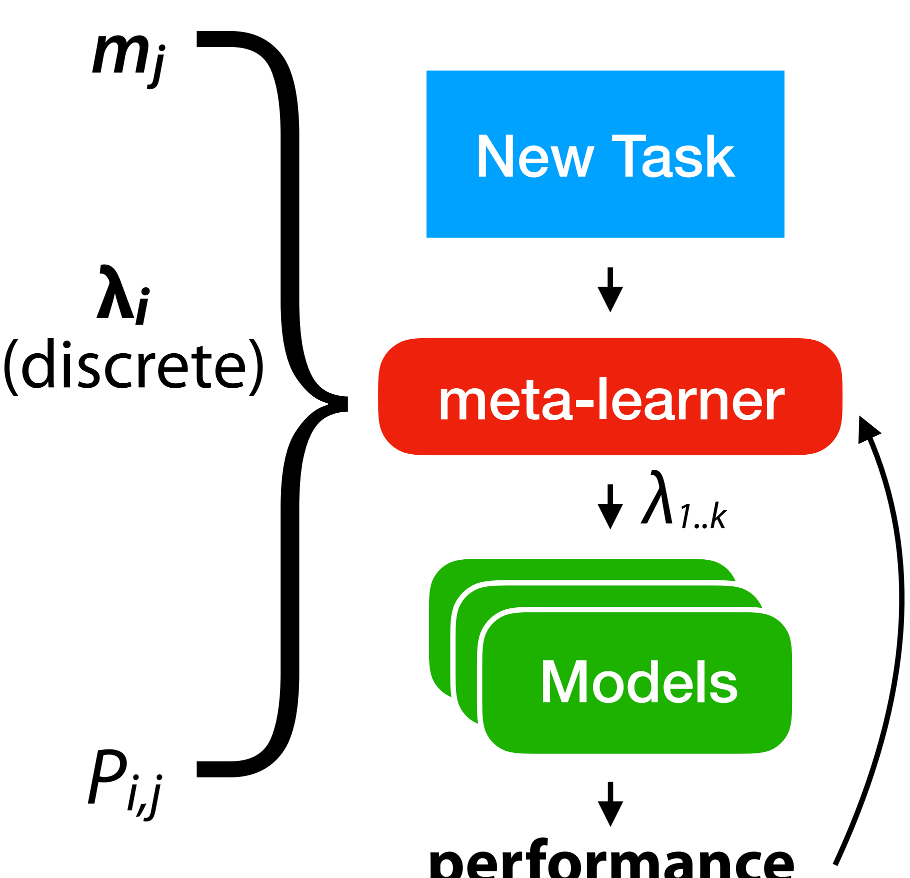


Warm-starting from similar tasks

Fusi et al. 2017

- Collaborative filtering: configurations λ_i are `rated' by tasks t_j
  - Learn latent representation for tasks and configurations
  - Use meta-features to warm-start on new task
  - Returns probabilistic predictions for BayesOpt



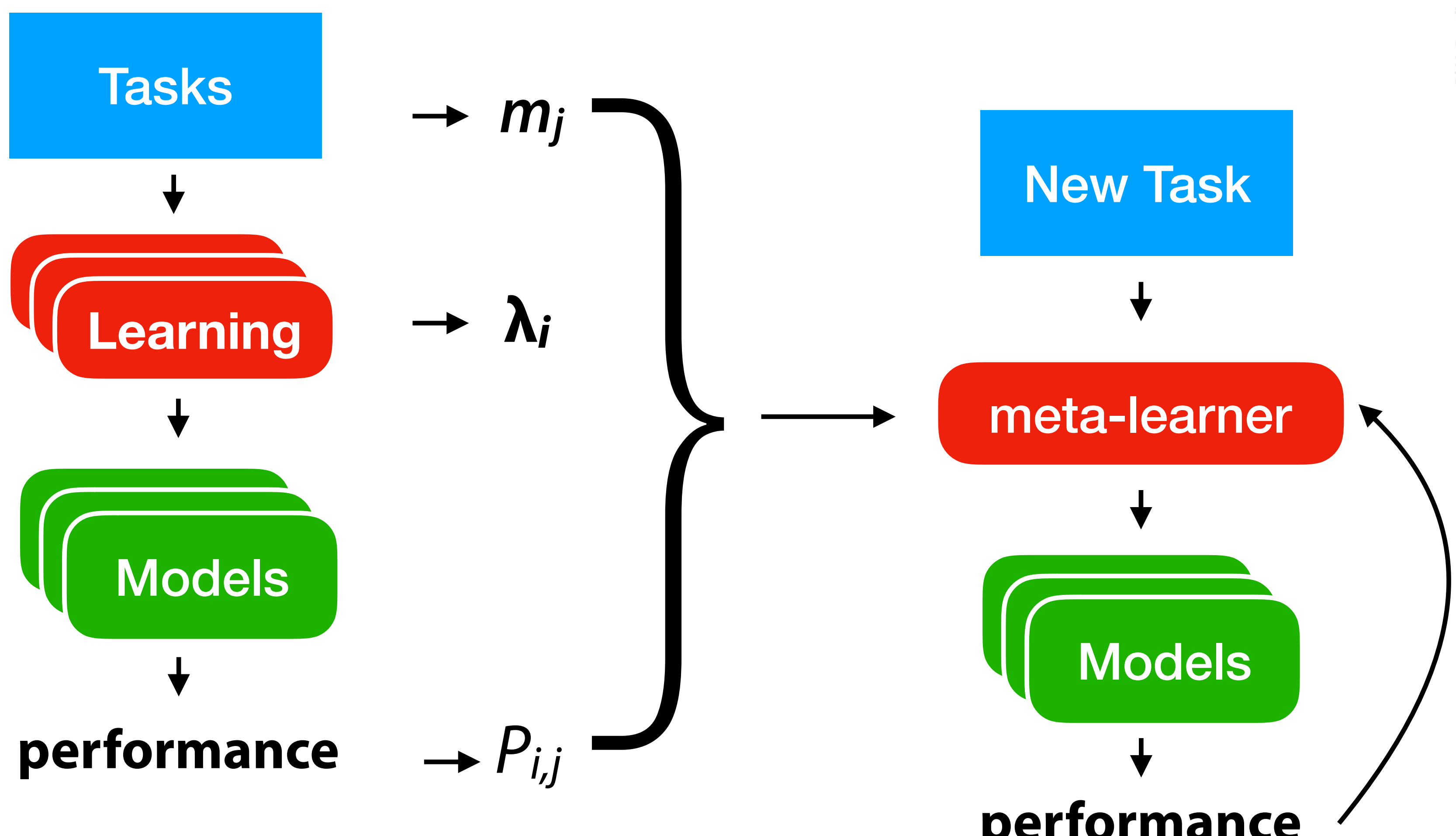


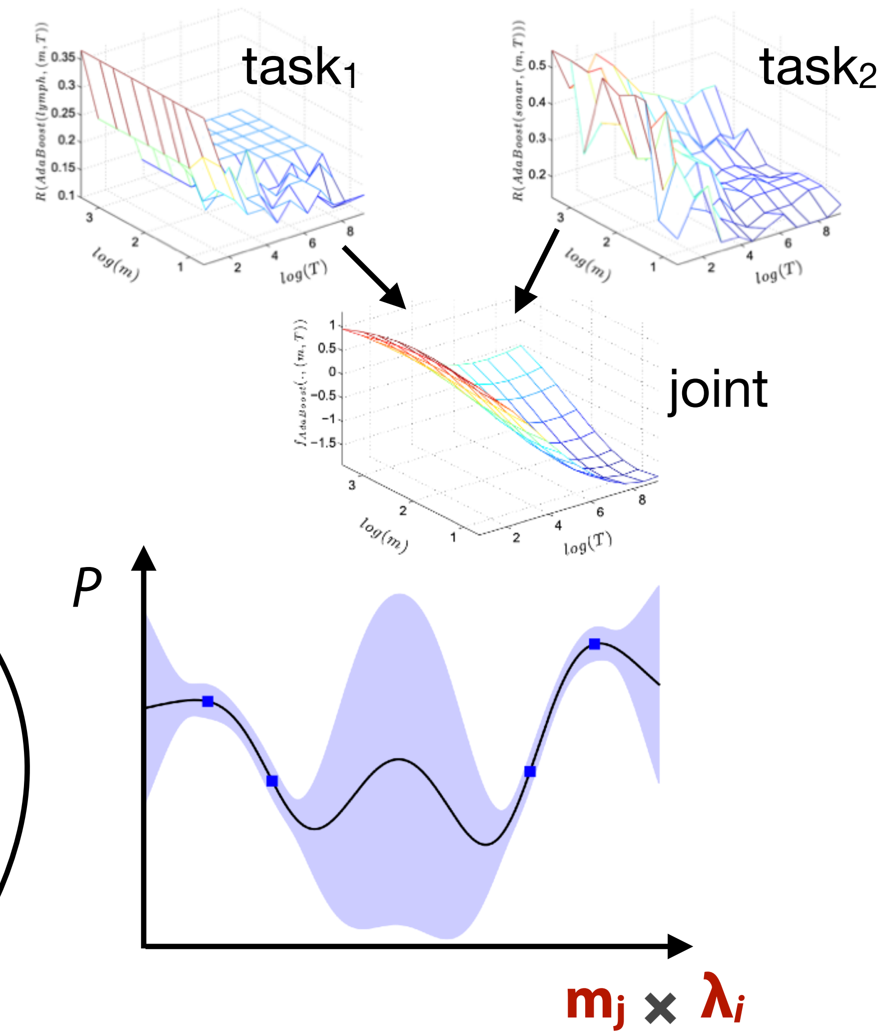
Meta-Learning

Global surrogate models

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

- 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





Meta-Learning

Meta-models

[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

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

    $m_j \rightarrow$ **meta-learner** $\rightarrow \lambda_{best}$

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

    $m_j \rightarrow$ **meta-learner** $\rightarrow \lambda_{1..k}$

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

    $m_j \rightarrow$ **meta-learner** $\rightarrow \Lambda$

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

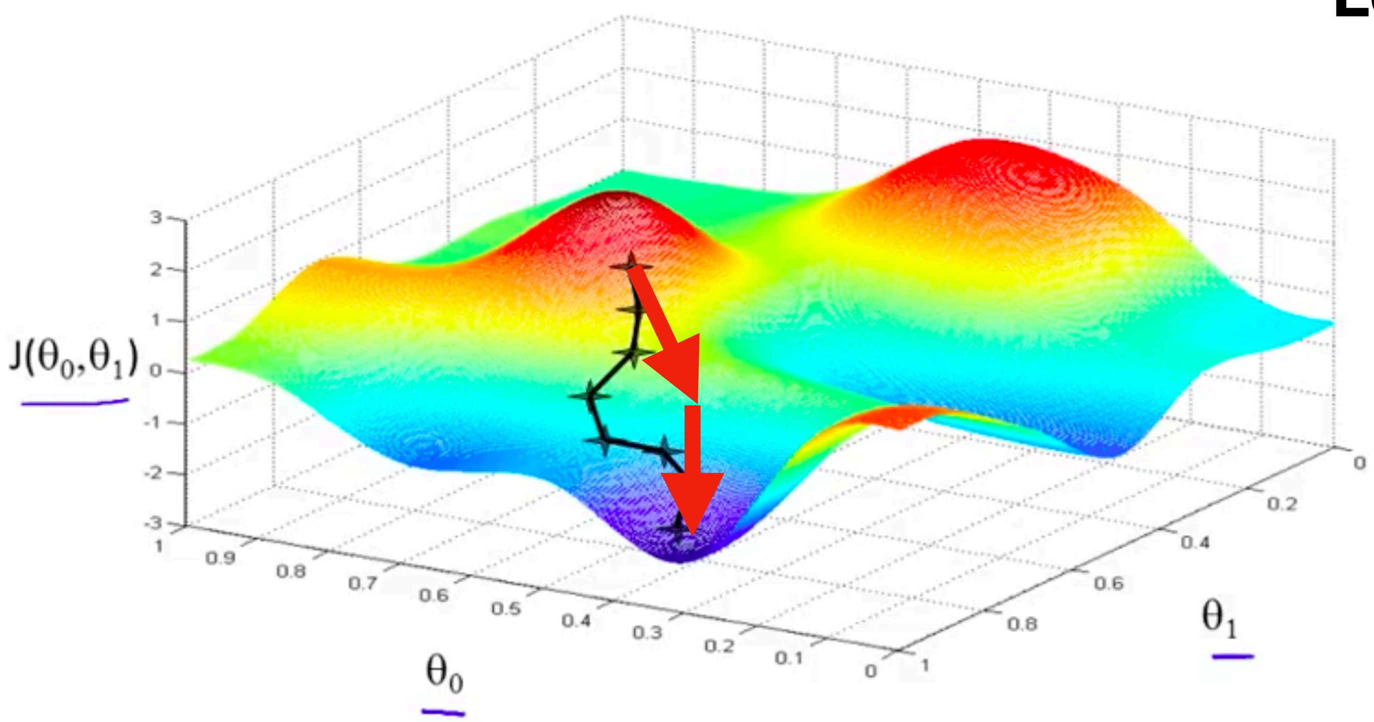
    $m_j, \lambda_i \rightarrow$ **meta-learner** $\rightarrow P_{ij}$

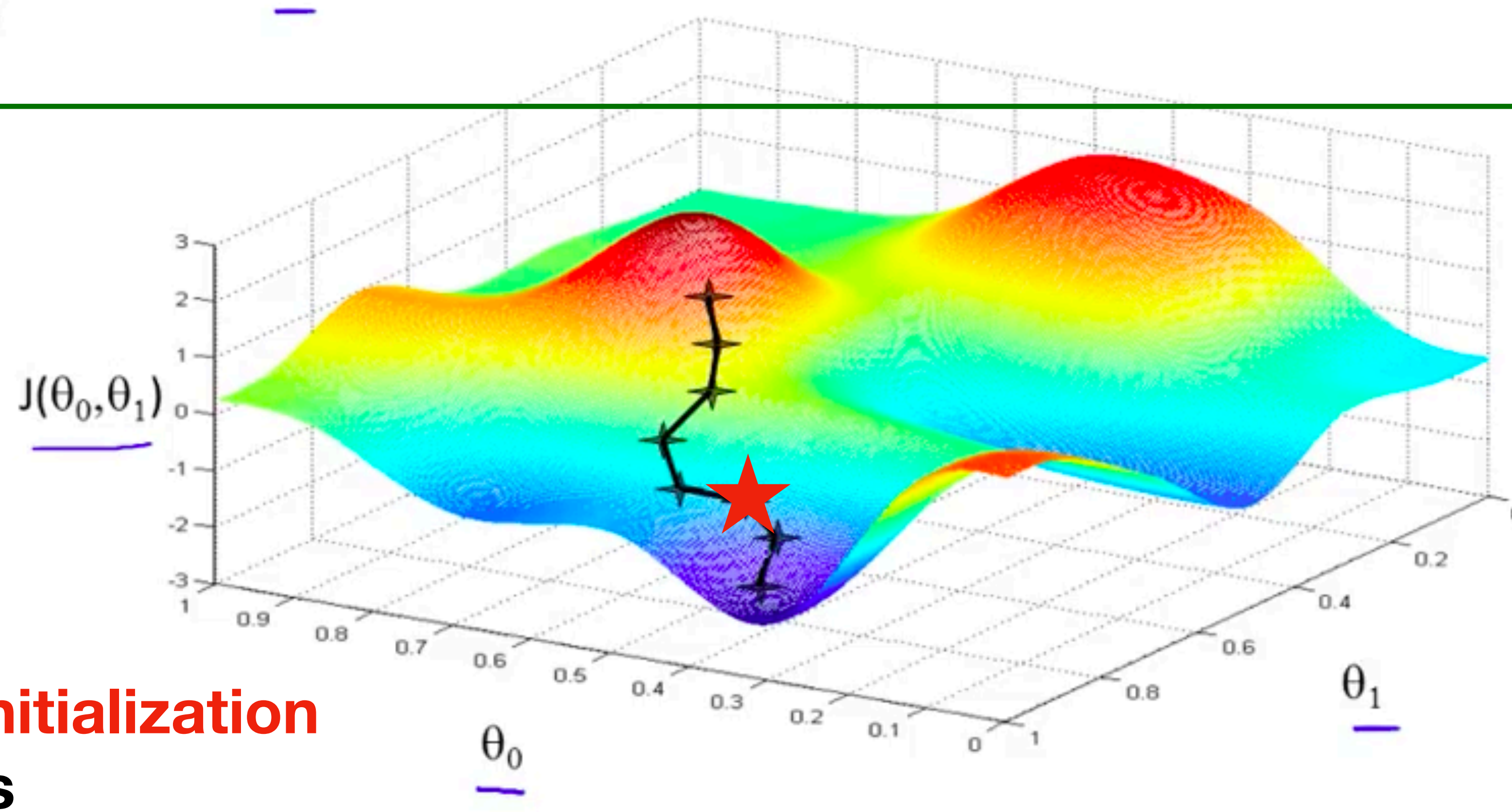
- Can be integrated in larger AutoML systems: warm start, guide search,…

Learning to learn

Transfer knowledge between very similar tasks

Learn and transfer a gradient descent update rule for a group of tasks





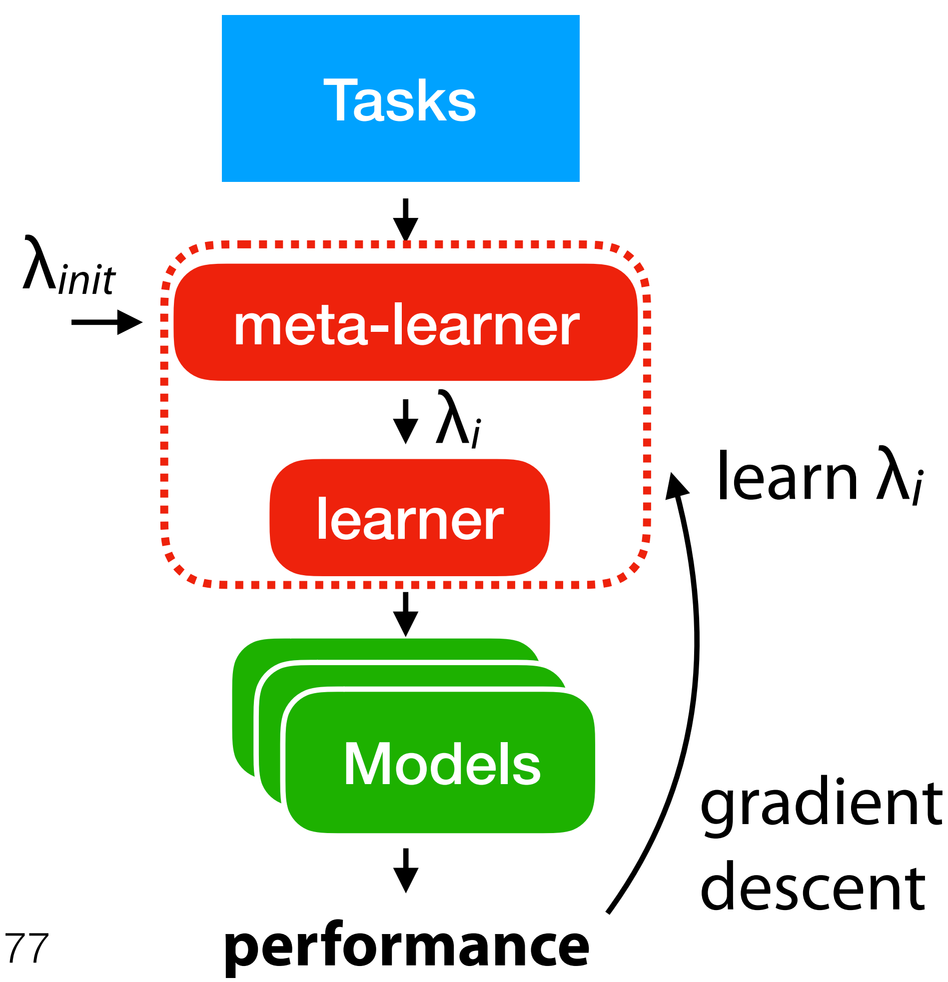
Learn and transfer a weight initialization for a group of tasks

[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: $\cancel{\Delta w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}}}$
  - 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)



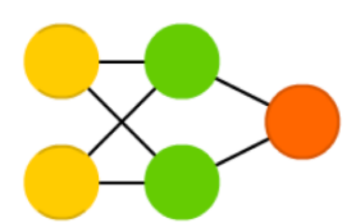
Bengio et al.

presynaptic activity

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

learning rate   reinforcing signal

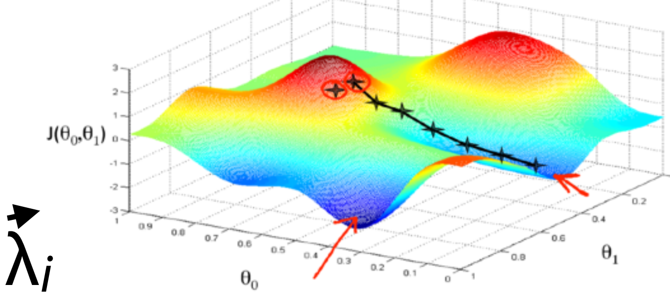
Runarsson and Jonsson

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

learning rate

weights λ_i

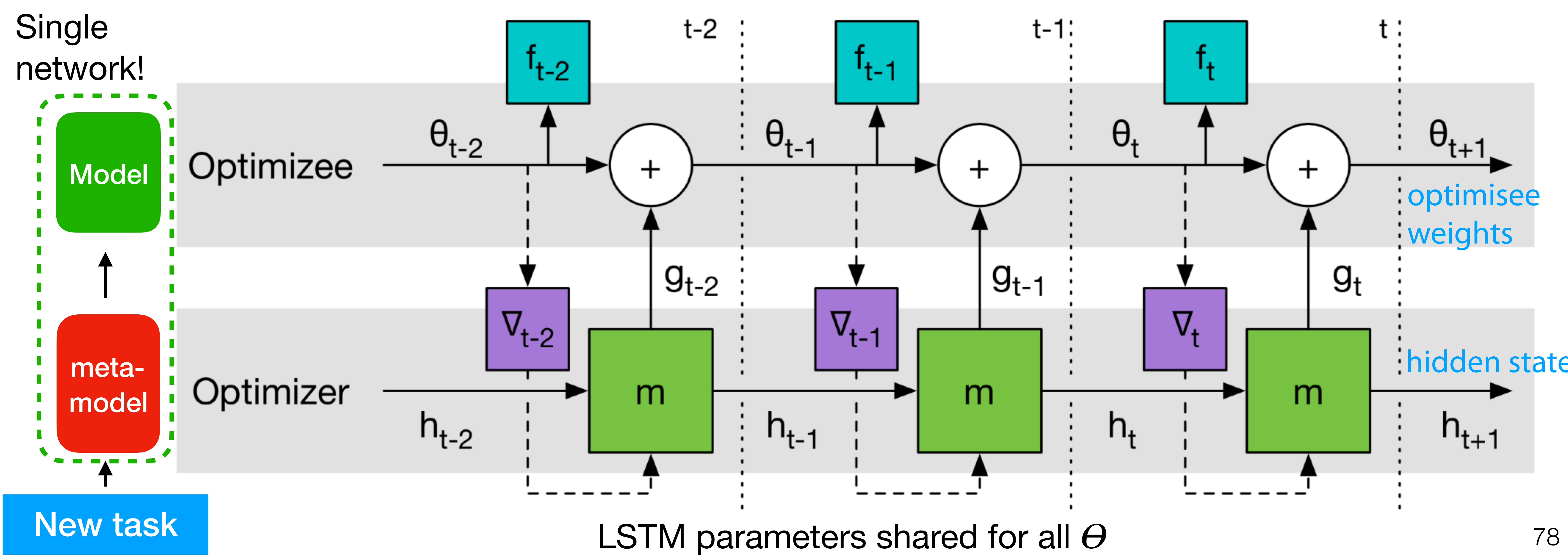
meta-gradient



[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



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

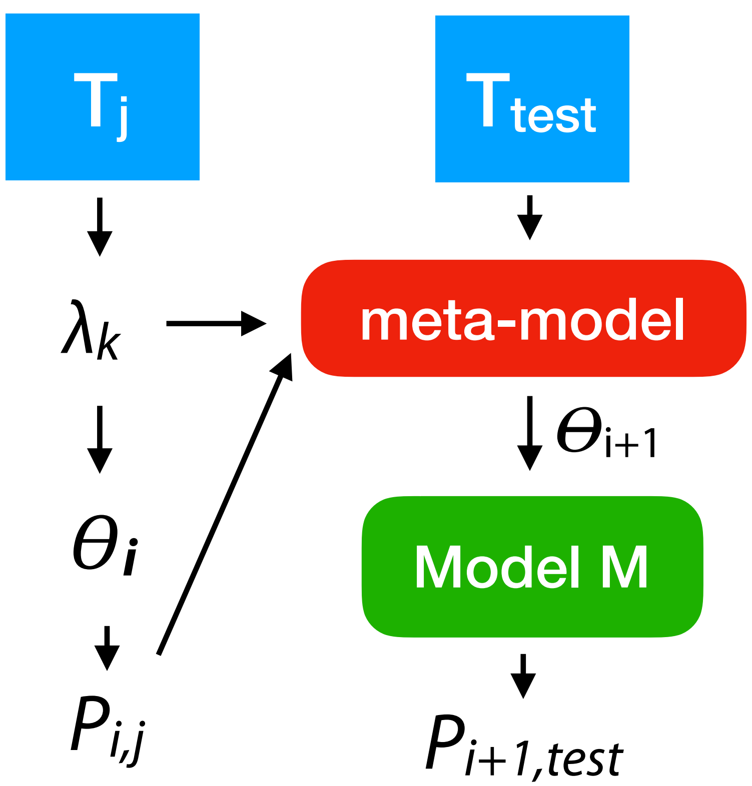
Learning to learn gradient descent by gradient descent

- Left: optimizer and optimizee trained to do style transfer
- Right: optimizer solves similar tasks (different style, content and resolution) without any more training data

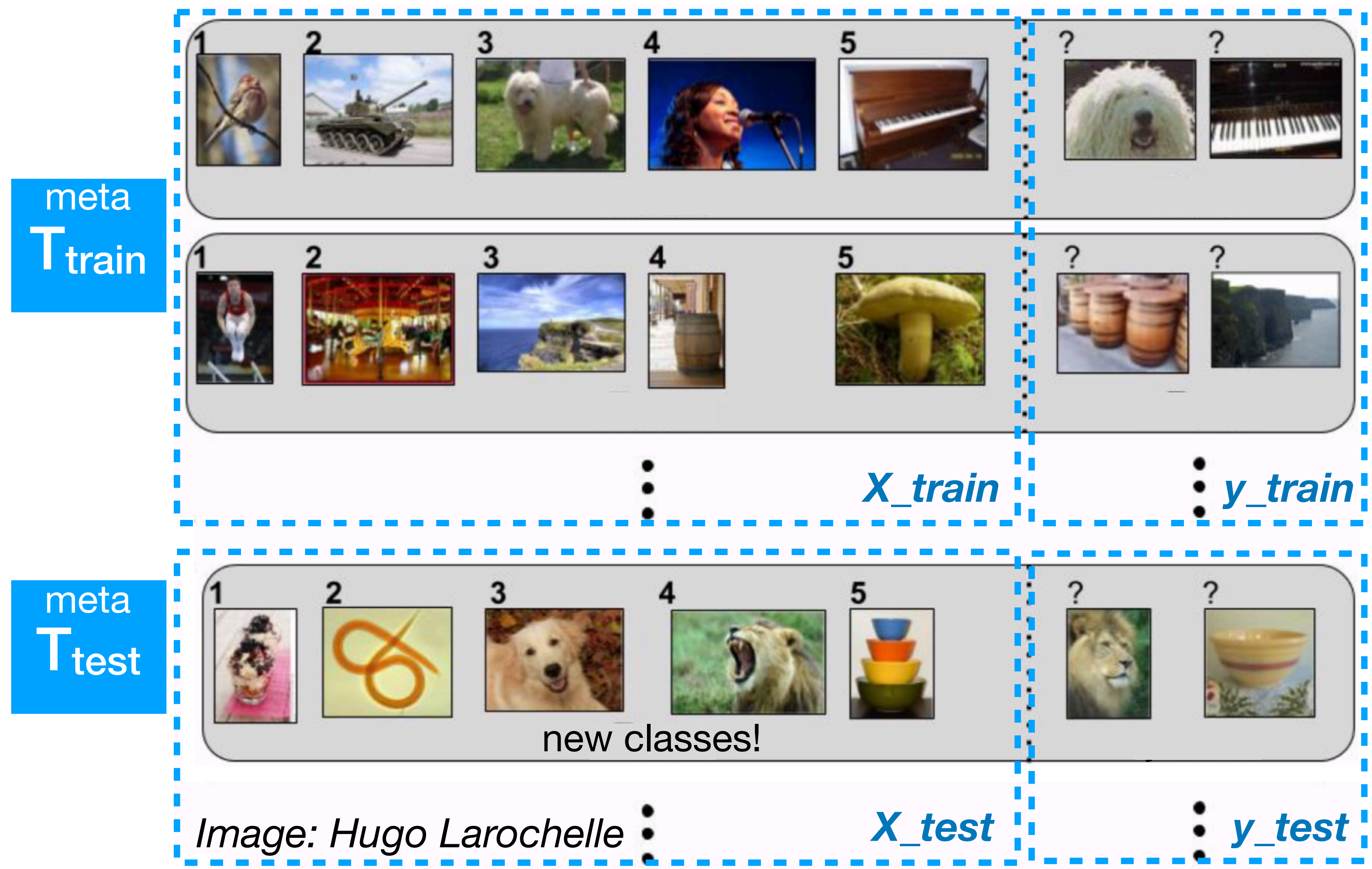
Application: Few-shot learning

- 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

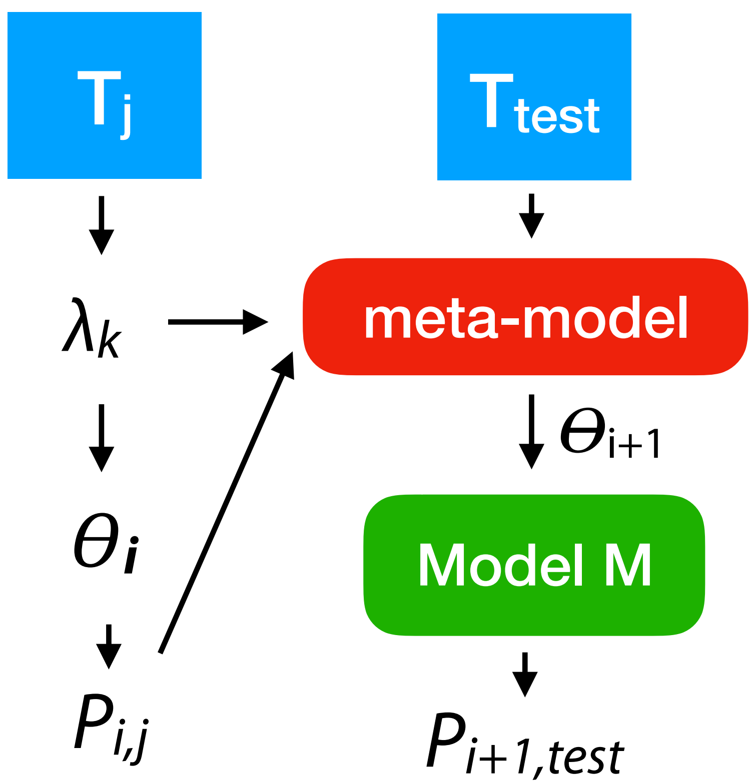
$$Cost(\theta_i) = \frac{1}{|T_{test}|} \sum_{t \in T_{test}} loss(\theta_i, t)$$



1-shot, 5-class:



Few-shot learning: approaches

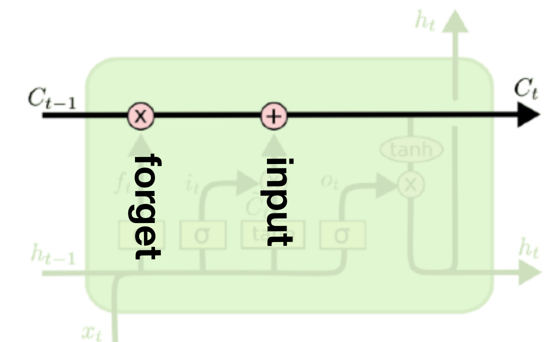


$$Cost(\theta_i) = \frac{1}{|T_{test}|} \sum_{t \in T_{test}} loss(\theta_i, t)$$

- 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

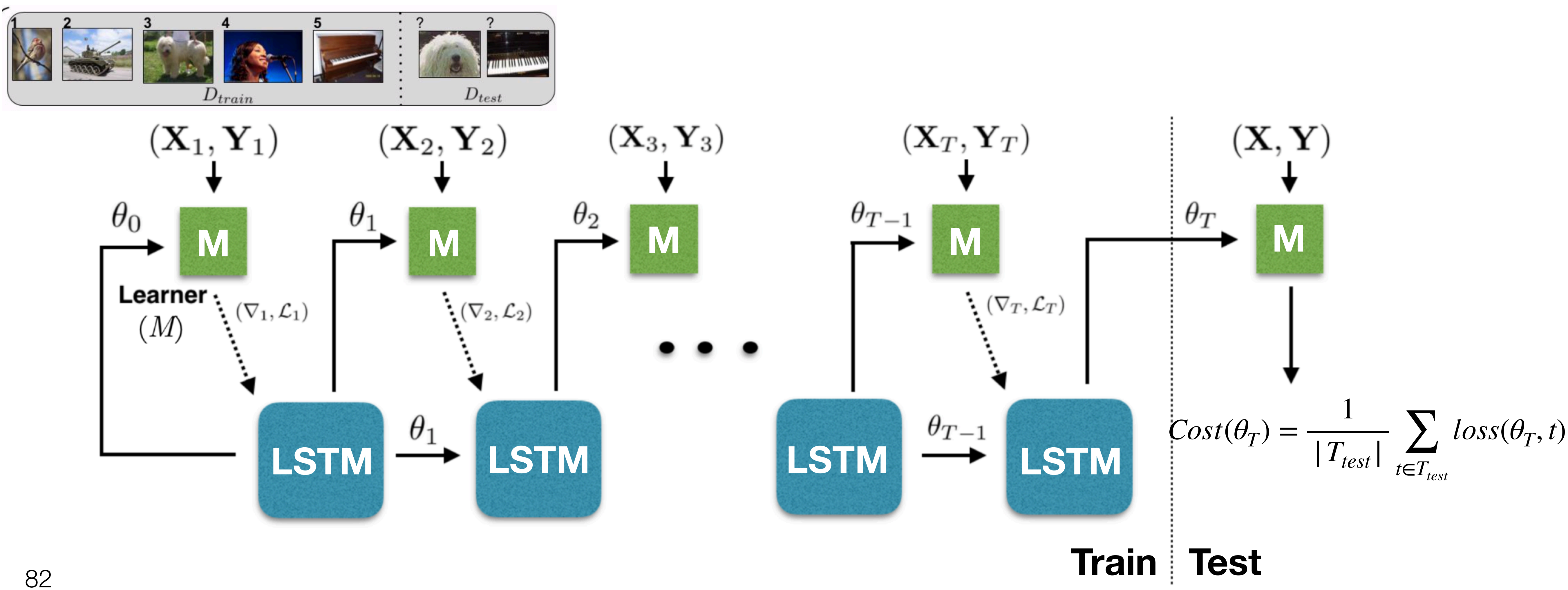
LSTM meta-learner + gradient descent



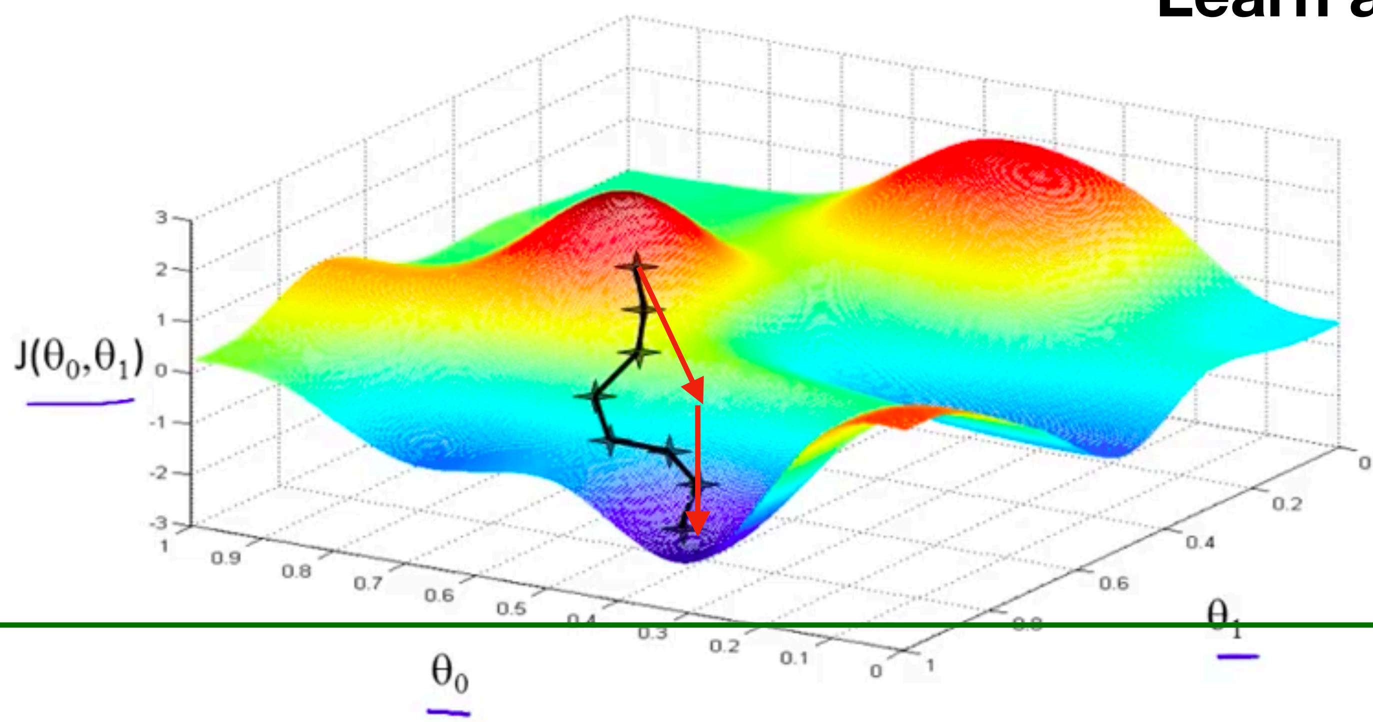
- Gradient descent update Θ_t is similar to LSTM cell state update c_t

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

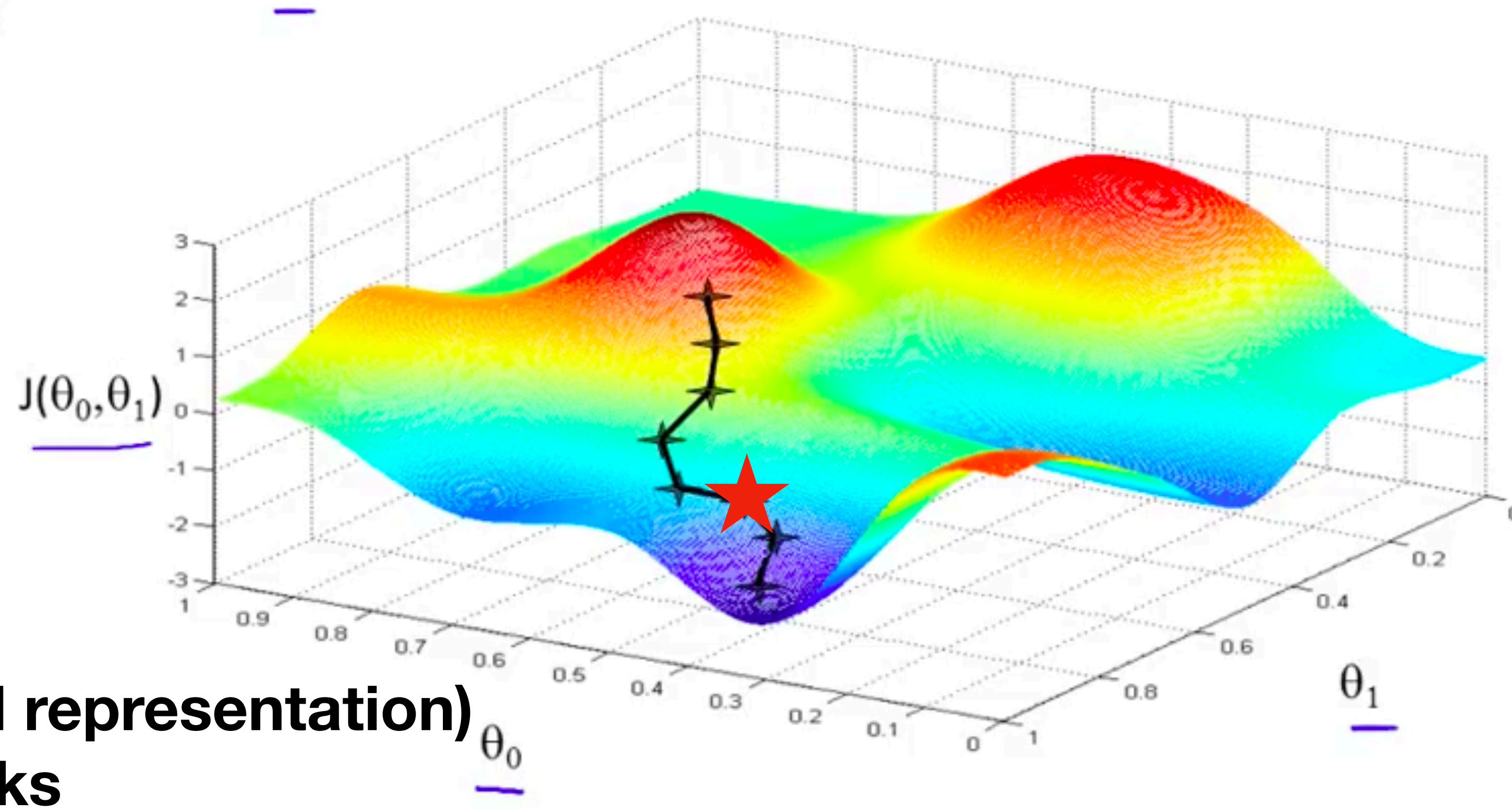
- Hence, training a meta-learner LSTM yields an update rule for training M
    - Start from initial Θ_0, train model on first batch, get gradient and loss update
    - Predict Θ_{t+1}, continue to t=T, get cost, backpropagate to learn LSTM weights, optimal Θ_0



Model transfer (Learning to learn)



Learn a better gradient update rule for a group of tasks

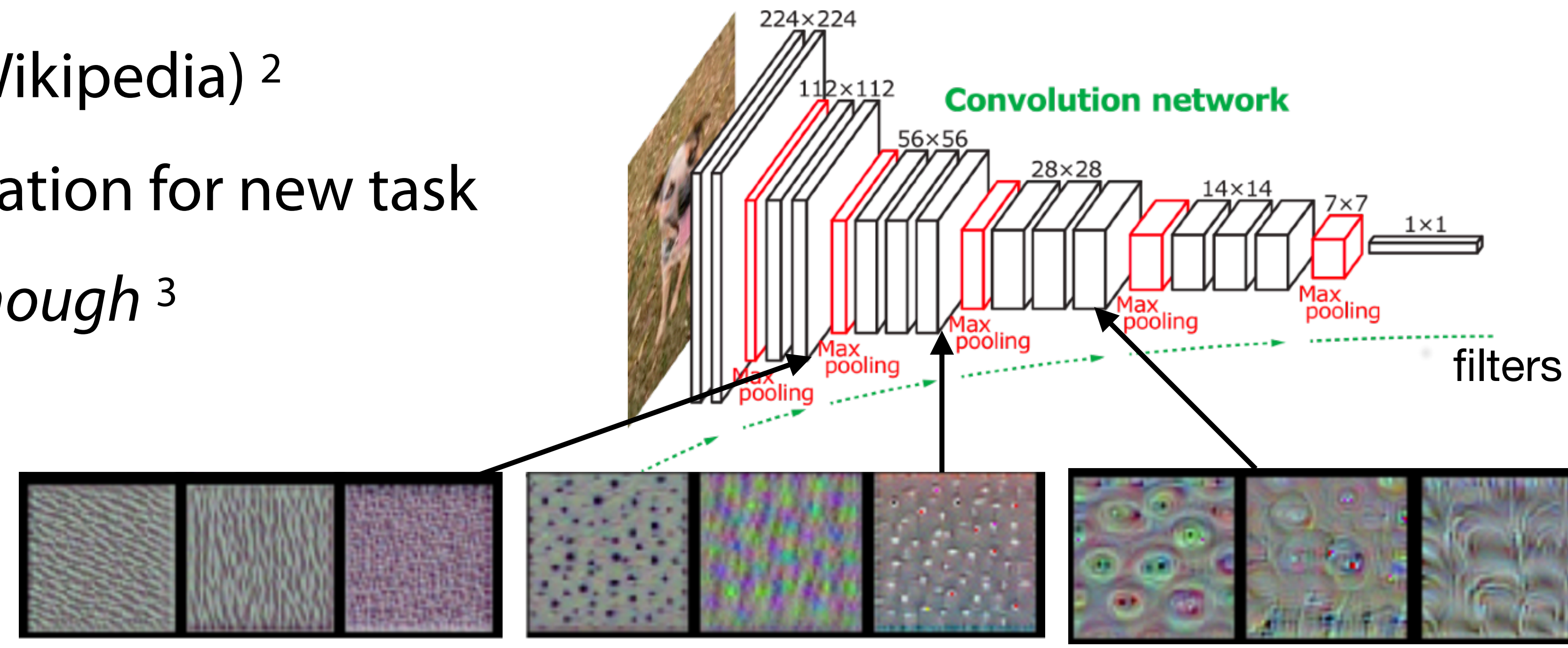


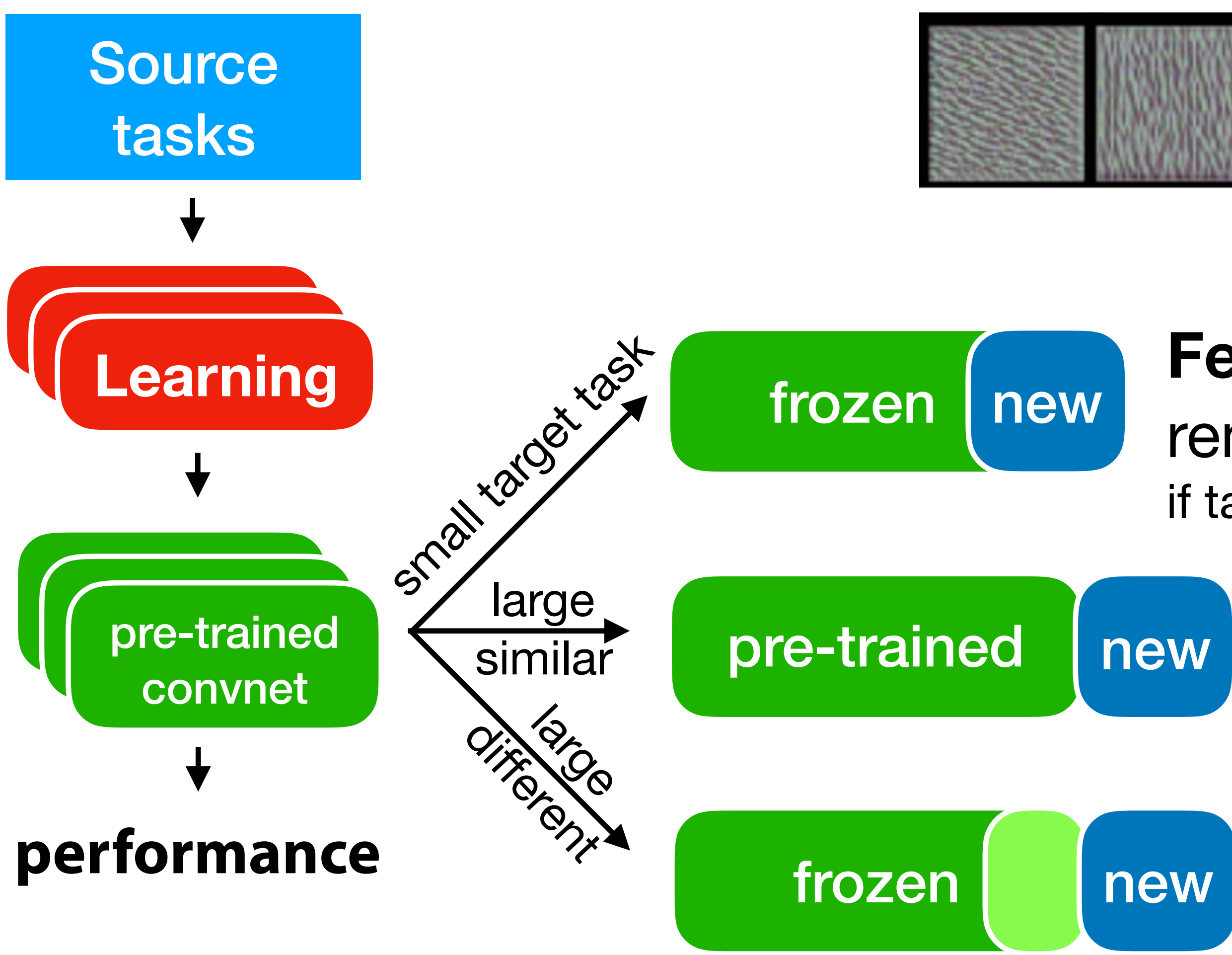
Learn a better initialization (and representation) for a group of tasks

Transfer learning

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

- Pre-train *weights* from:
  - Large image datasets (e.g. ImageNet) [1]
  - Large text corpora (e.g. Wikipedia) [2]
- Use these weights as initialization for new task
- 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

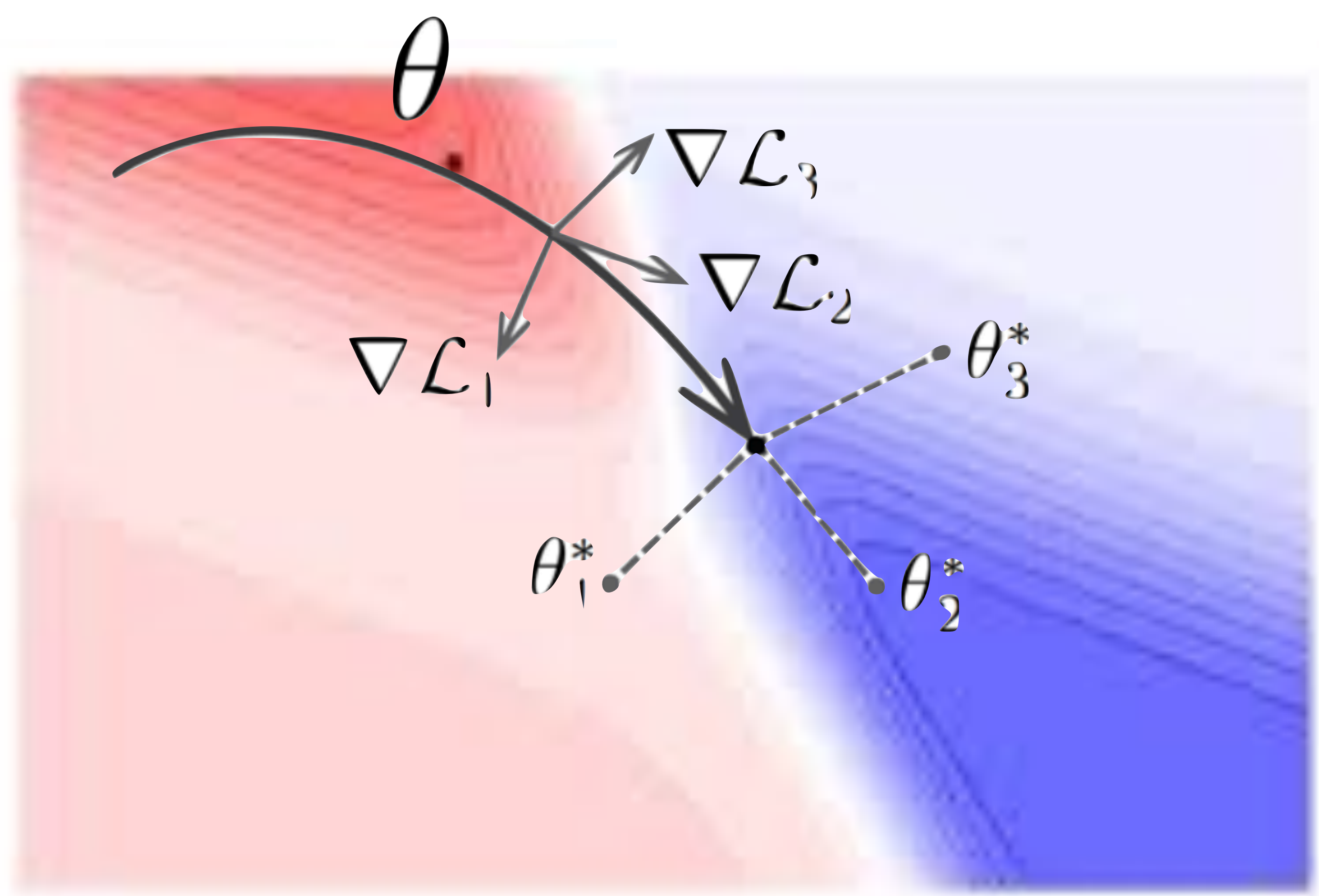
[1] Finn et al. 2017

Model-agnostic meta-learning

- Solve new tasks faster by learning a model *initialization* from 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

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

  - Repeat



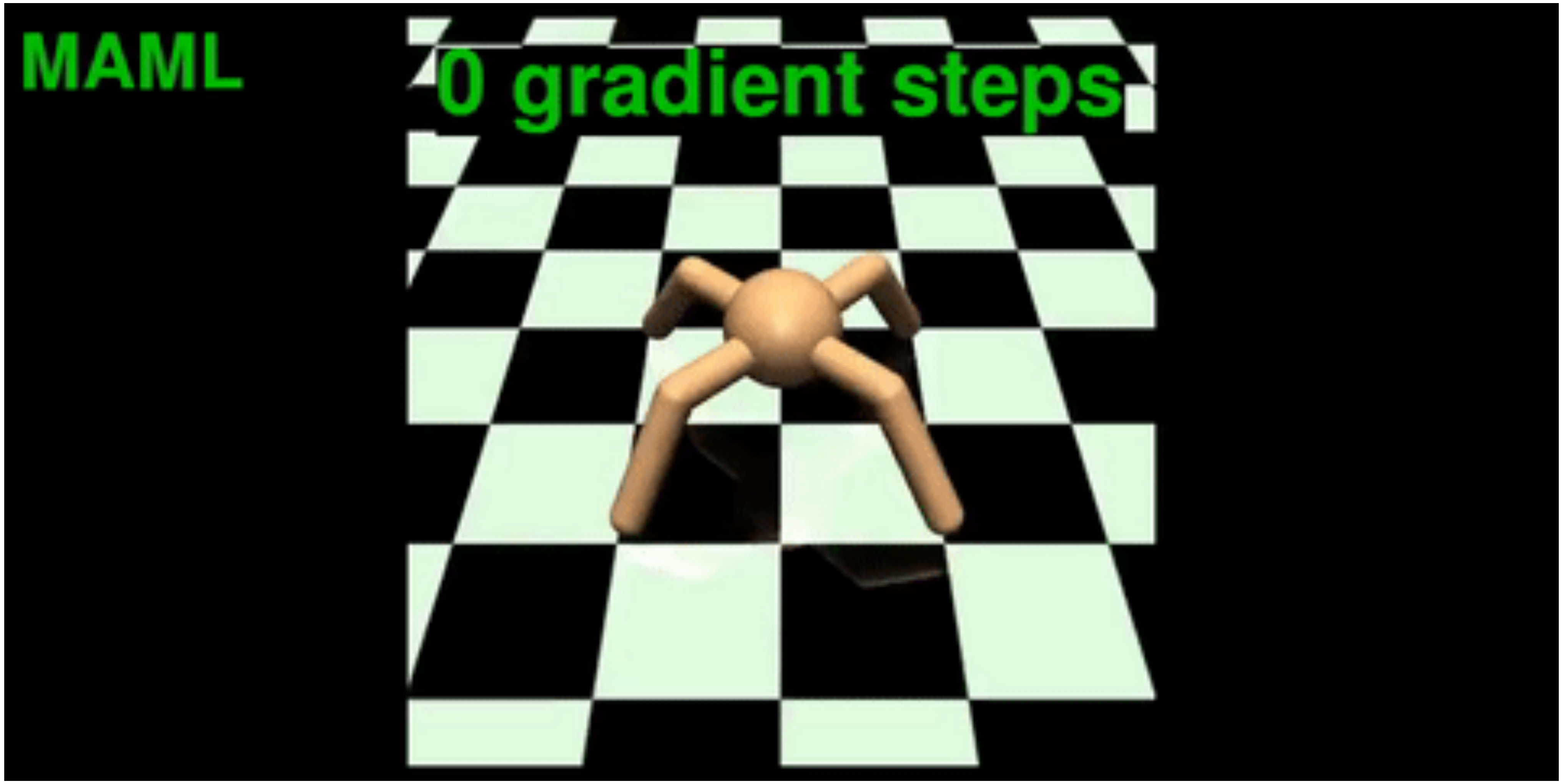
[1] Finn et al. 2018
[2] Nichol et al. 2018

Model-agnostic meta-learning

- More resilient to overfitting
- Generalizes better than LSTM approaches for few shot learning
- *Universality* [1]: no theoretical downsides in terms of expressivity when compared to alternative meta-learning models.
- Many variants and applications: Finn & Levine 2019
  - REPTILE: do SGD for k steps in one task, only then update init. weights [2]
  - PLATIPUS: probabilistic MAML
  - Bayesian MAML (Bayesian ensemble)
  - Online MAML
  - …
- Lots of current research
  - What does it actually learn?
  - Can it work on out-of-distribution, multi-modal task distributions,…?

Model-agnostic meta-learning

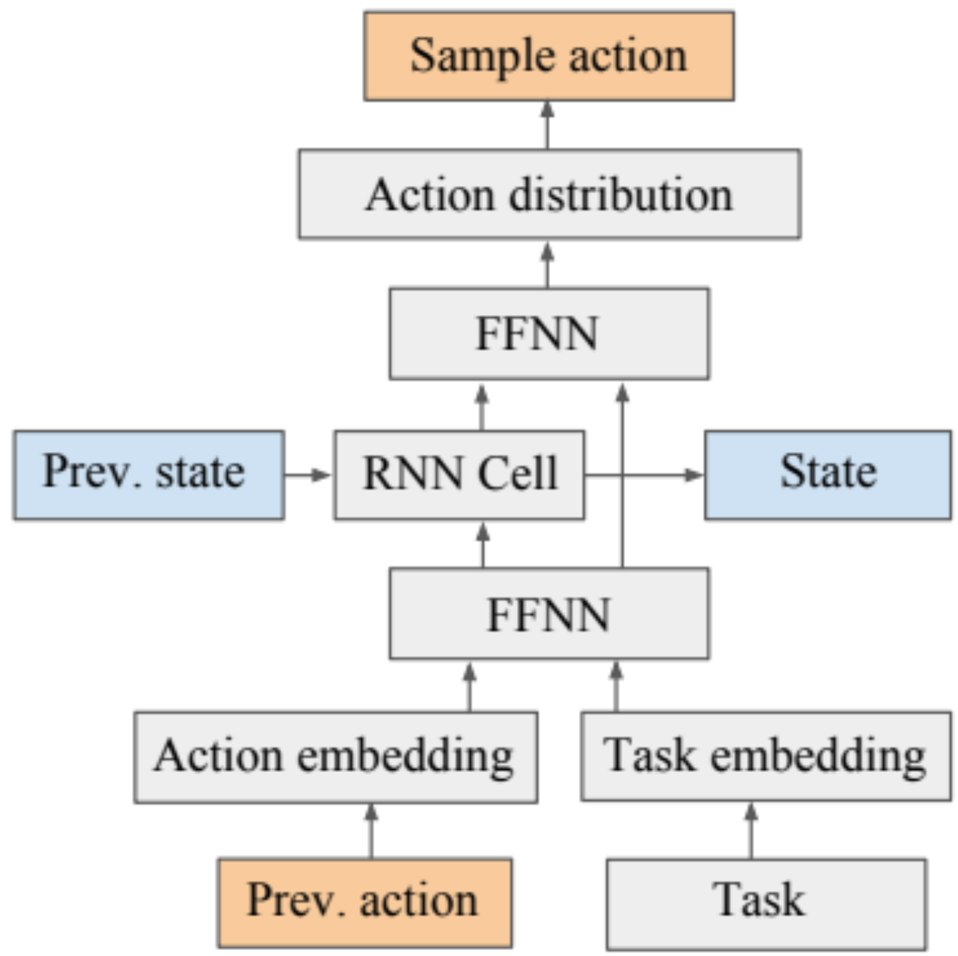
- For reinforcement learning:



Neural Architecture Meta-Learning

- Warm-start a deep RL controller based on prior tasks
- Much faster than single-task equivalent

Wong et al. 2018

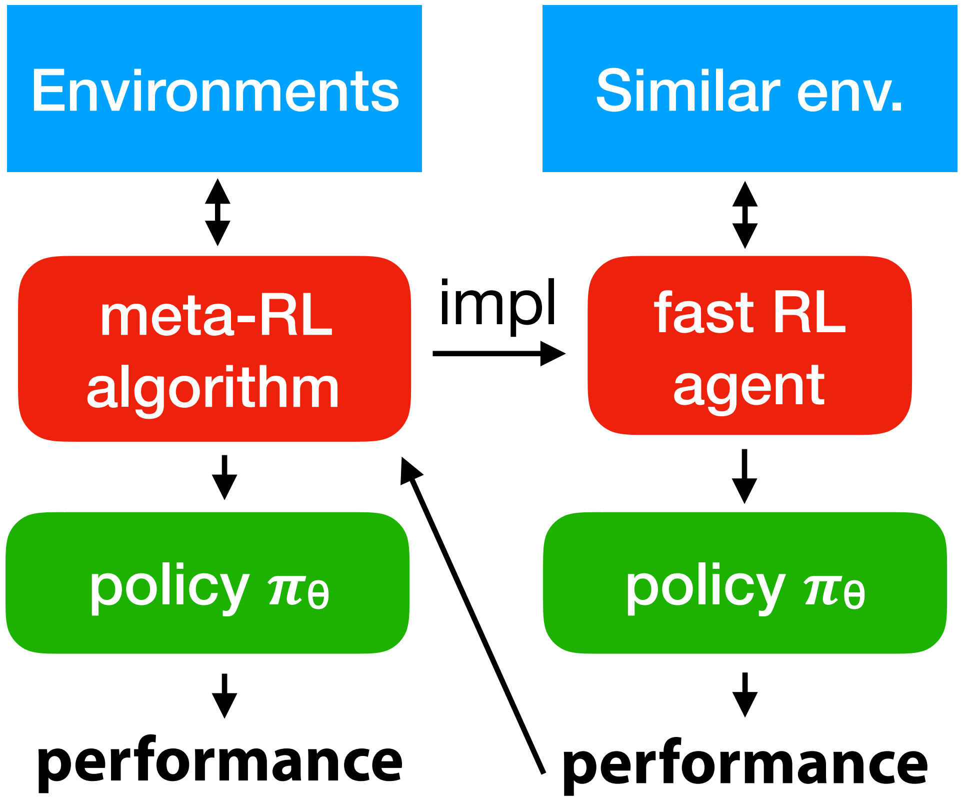


- Another idea: warm-start DARTS with task-specific one-shot model

Learning to reinforcement learn

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

- 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



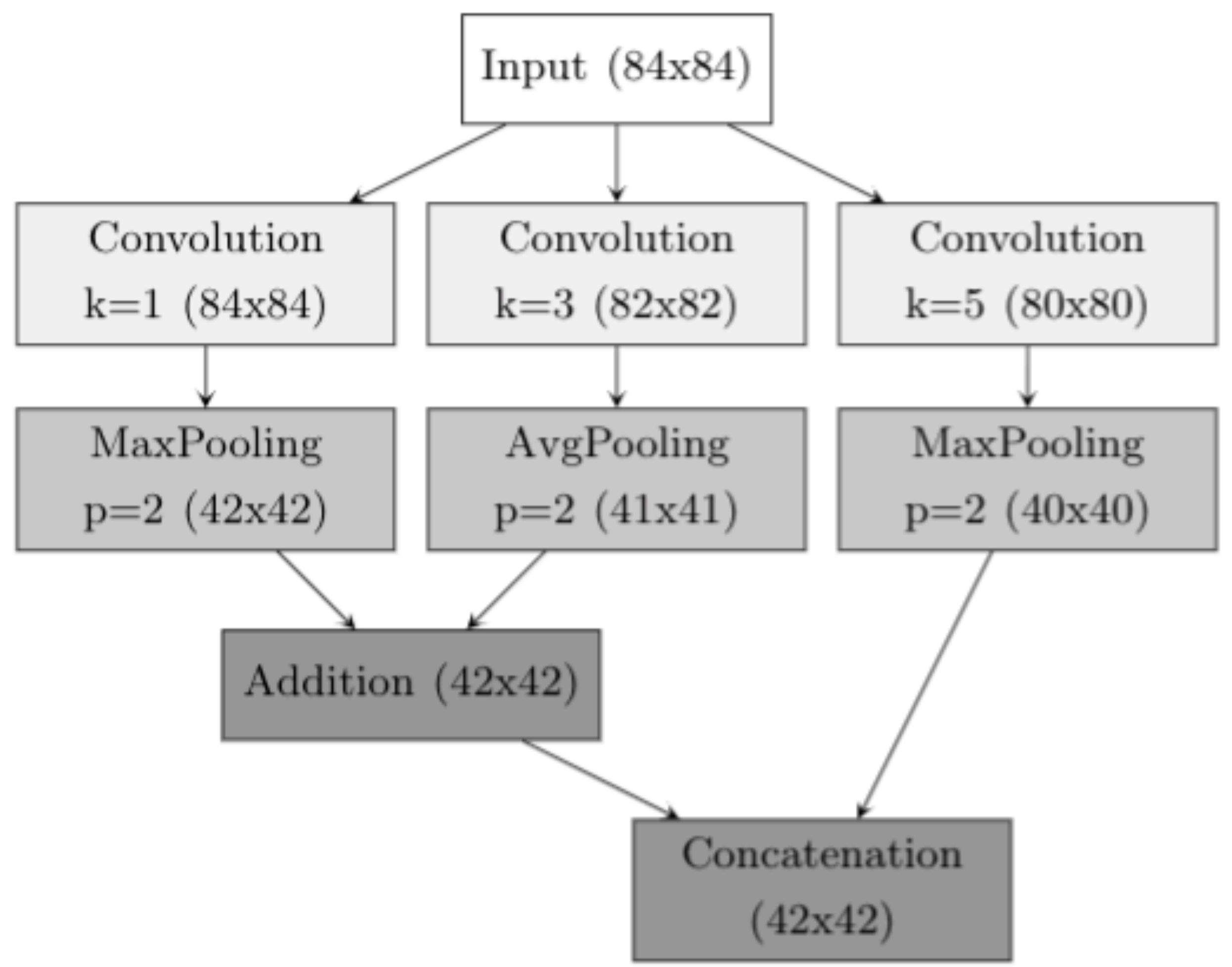
- Learning to reinforcement learn [1,2]
  - Use RNN-based deep RL to train a recurrent network on many tasks
  - Learns to implement a 'fast' RL agent, encoded in its weights

Gomez & Vanschoren, NeurIPS 2019

Meta-Reinforcement Learning for NAS

- Train an agent how to build a neural net, across tasks
- Should transfer but also adapt to new tasks

[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[1, 1, 1, 0, 0]
[2, 2, 2, 1, 0]
[3, 1, 3, 0, 0]
[4, 3, 2, 3, 0]
[5, 1, 5, 0, 0]
[6, 2, 2, 5, 0]
[7, 5, 0, 2, 4]
[8, 7, 0, 0, 0]



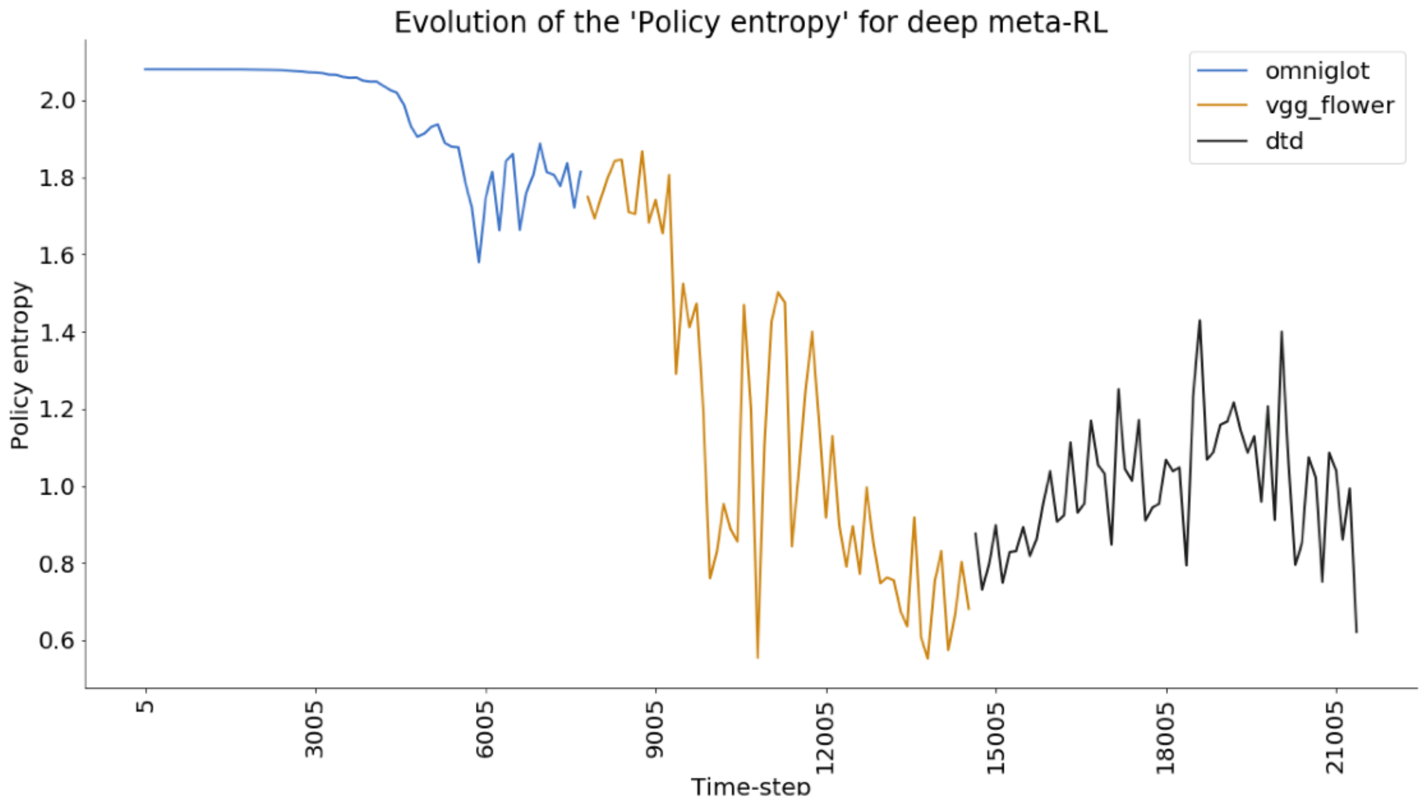
Actions: add certain layers in certain locations

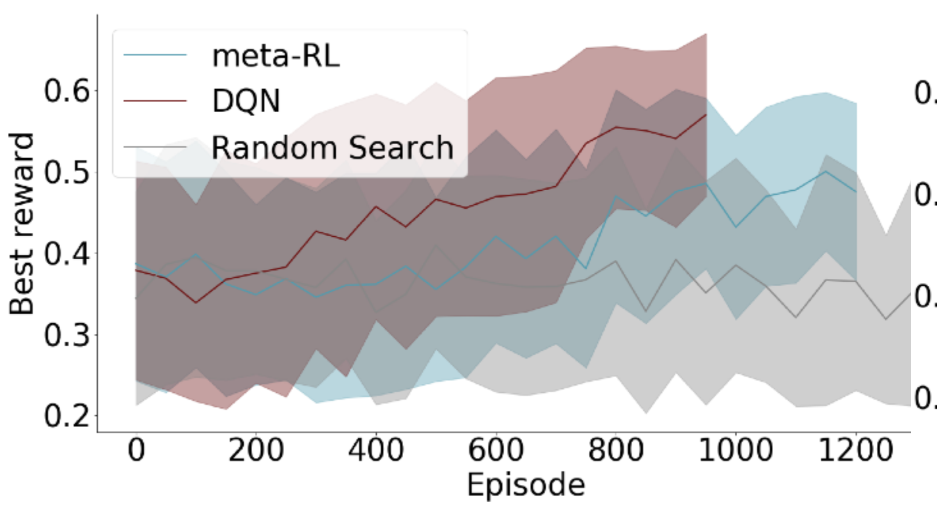
Gomez & Vanschoren, NeurIPS 2019

Meta-Reinforcement Learning for NAS

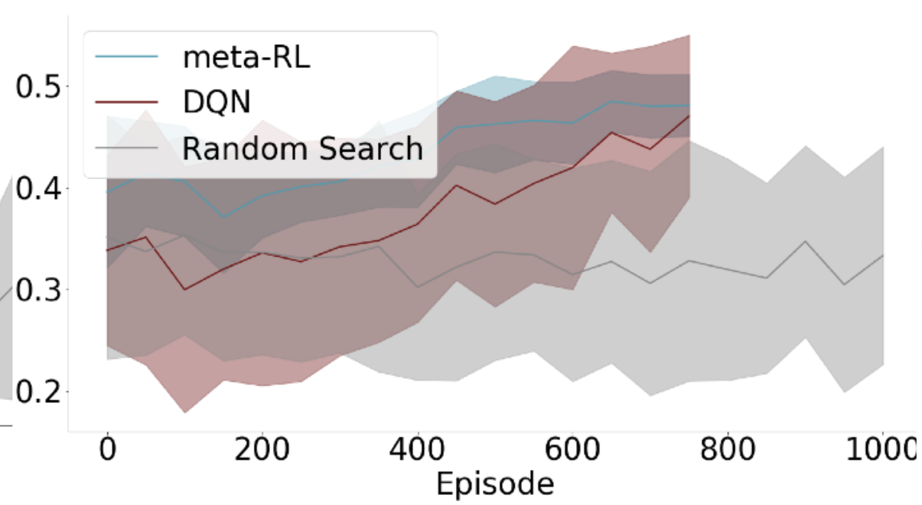
Results on increasingly difficult tasks:

- omniglot
- vgg_flower
- dtd

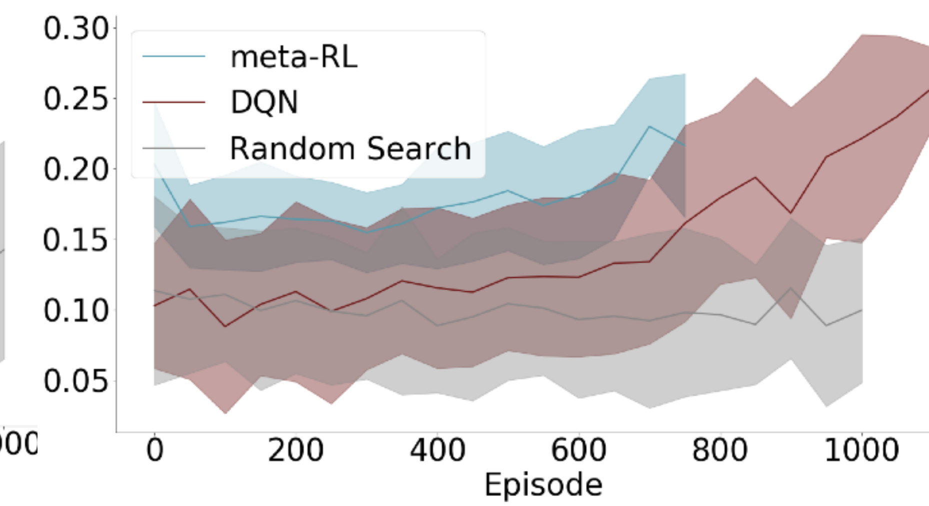




omniglot



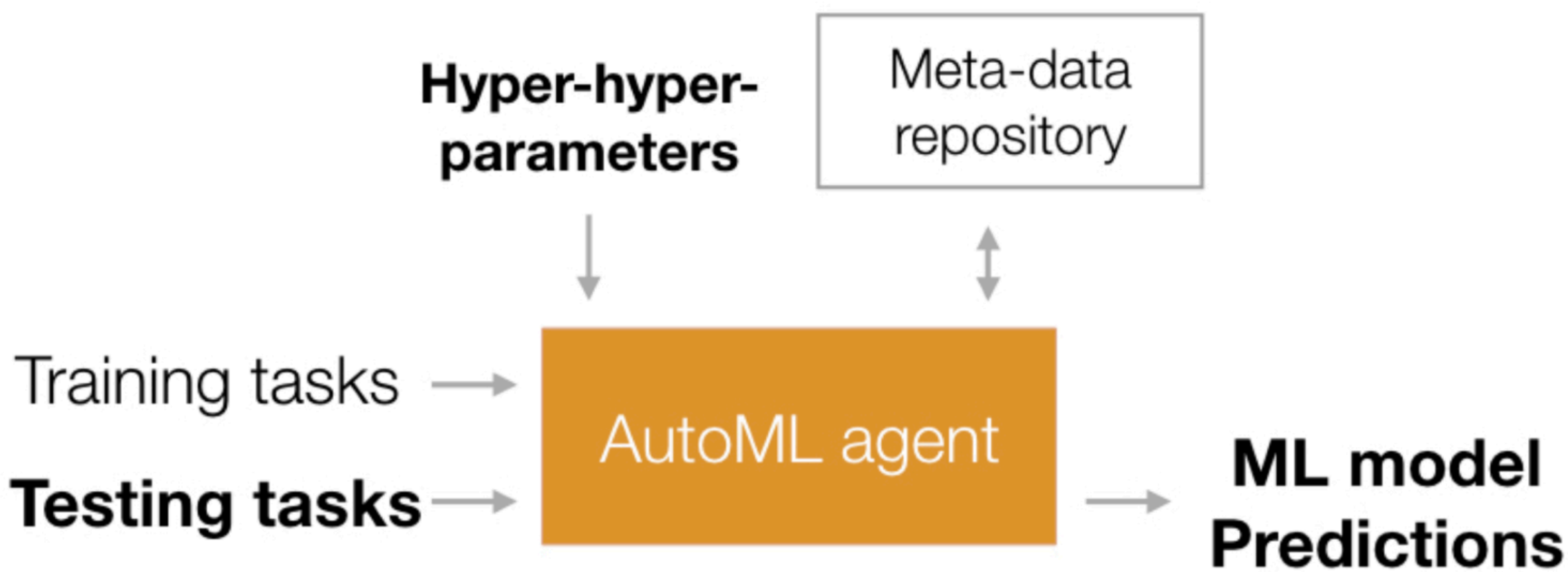
vgg_flower



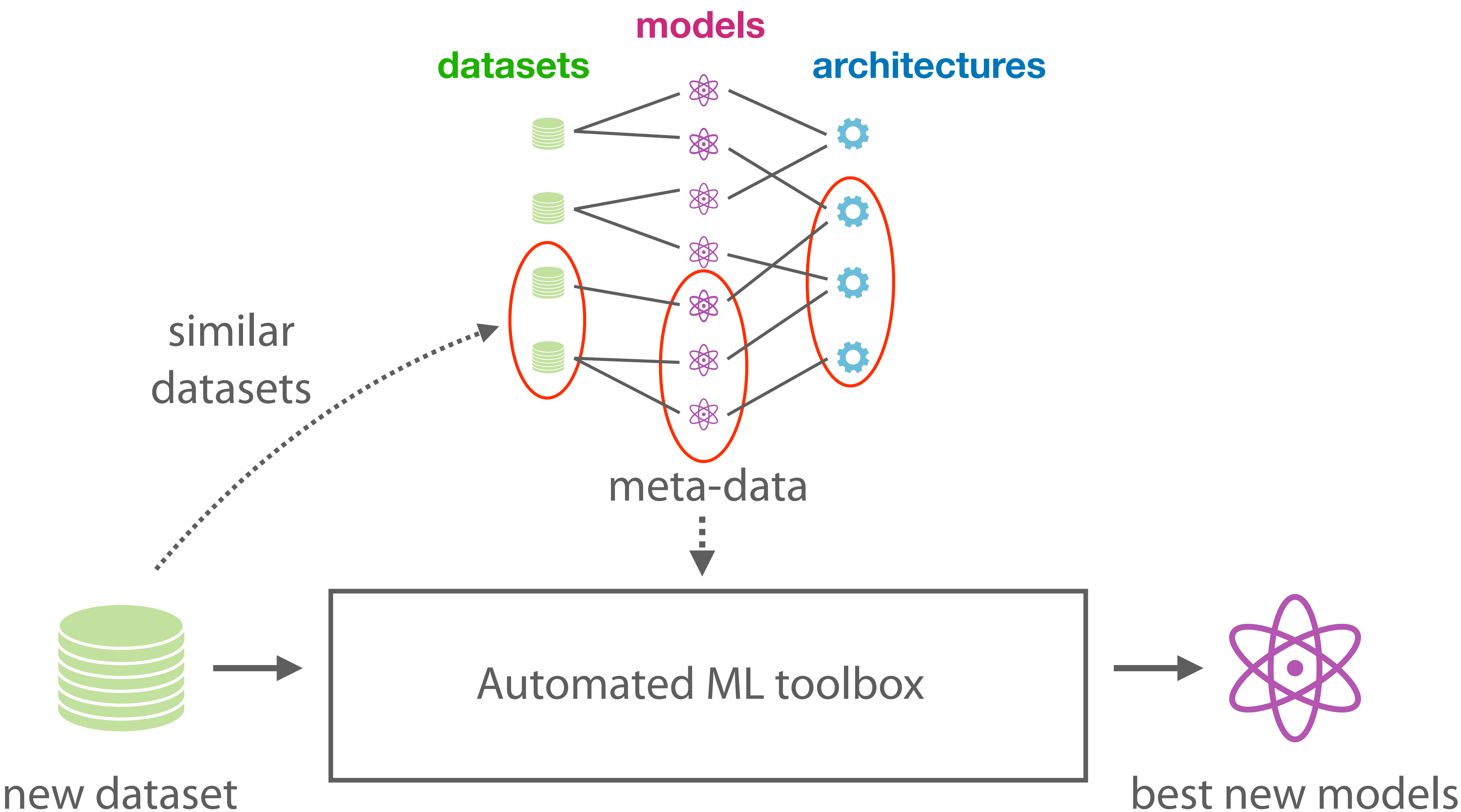
dtd

Meta-learning in practice

- We need a meta-data repository of relevant prior machine learning experiments
  - e.g. OpenML.org (needs more deep learning experiments)
- Ideally, a *shared* memory that all AutoML tools can access



Meta-learning with OpenML



AutoML *open source* tools

	Architect. search	Operators	Hyperpar. search	Improvements	Metalearnin
Auto-WEKA	Param. pipeline	WEKA	Bayesian Opt. (RF)		
auto-sklearn	Param. pipeline	sklearn	Bayesian Opt. (RF)	Ensemble	warm-start
mlr-mbo	Param. pipeline	mlr	Bayesian Opt.	multi-obj.	
BO-HB	Param. pipeline	sklearn	Tree of Parzen Estim.	Ensemble, HB	
hyperopt-sklearn	Param. pipeline	sklearn	Tree of Parzen Estim.		
skopt	Param. pipeline	sklearn	Bayesian Opt. (GP)		
TPOT	Evolving pipelines	sklearn	Population-based		
GAMA	Evolving pipelines	sklearn	Population-based	Ensemble, ASHA	
H2O AutoML	Param. pipeline	H2O	Random search	Stacking	
OBOE	Single algorithms	sklearn	Low rank approx.	Ensembling	runtime pred
Auto-Keras	Param. NAS	keras	Bayesian Opt.	Net Morphisms	
Auto-pyTorch	Param. pipeline	pyTorch	BO-HB		
TensorFlow 2	/	keras	RS or HB		
Talos	/	keras	RS variants		

Many other tools for hyperparameter optimization alone

Thank you!

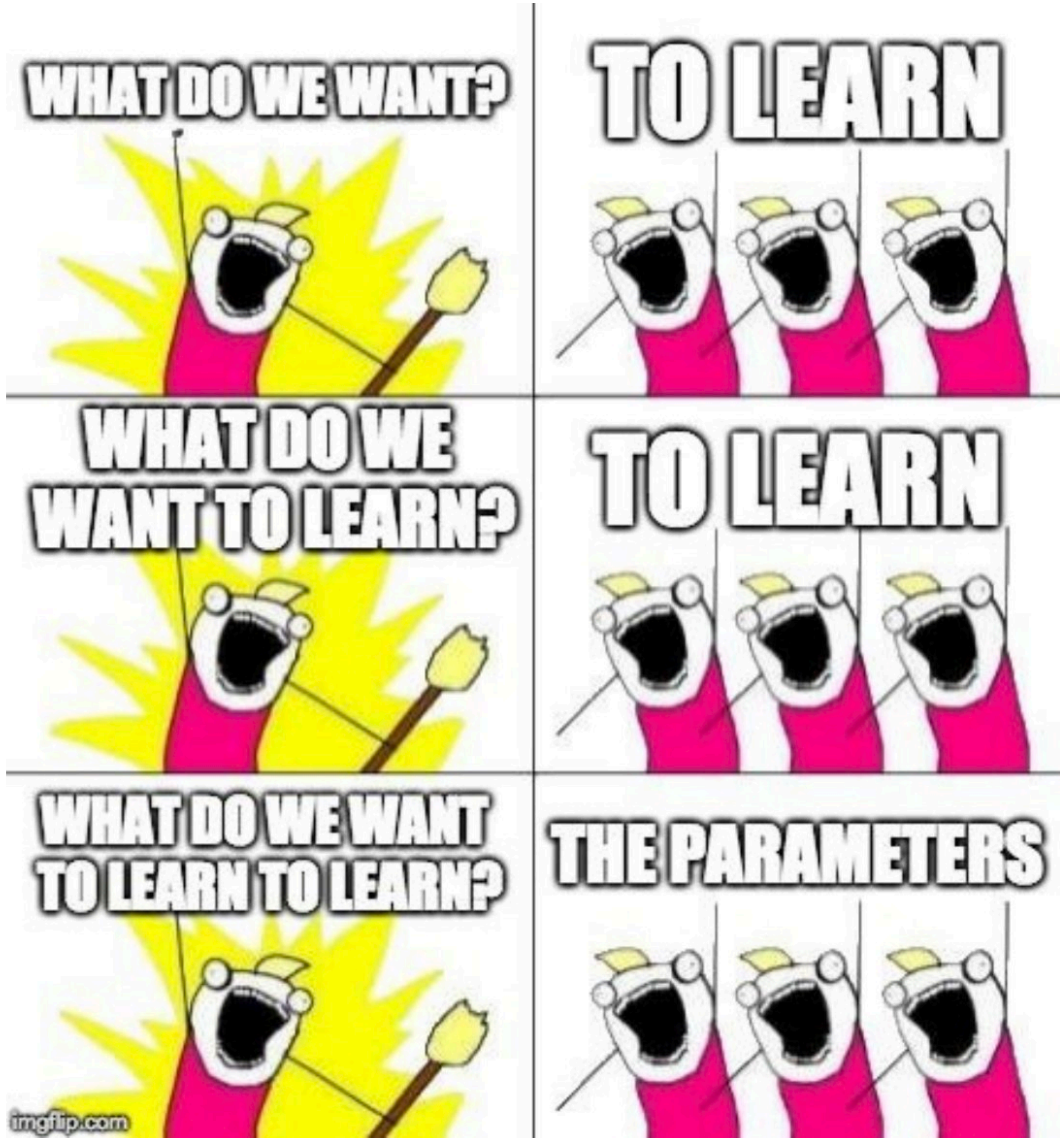


Image: Pesah et al. 2018

Andreas Mueller
Bernd Bischl
Bilge Celik
Erin LeDell
Frank Hutter
Guiseppe Casalicchio
Heidi Seibold
Jakub Smid
Jan van Rijn
Janek Thomas
Joaquin Vanschoren
Matthias Feurer
Marcel Wever
Markus Weimer
Michel Lang
Mitar Milutinovic
Neeratjoy Mallik
Neil Lawrence
Pieter Gijsbers
Prabhant Singh
Sahithya Ravi
William Raynaut

Thanks to the OpenML team!

Join us :)