

Automatic Machine Learning

A Tutorial

Joaquin Vanschoren
Eindhoven University of Technology

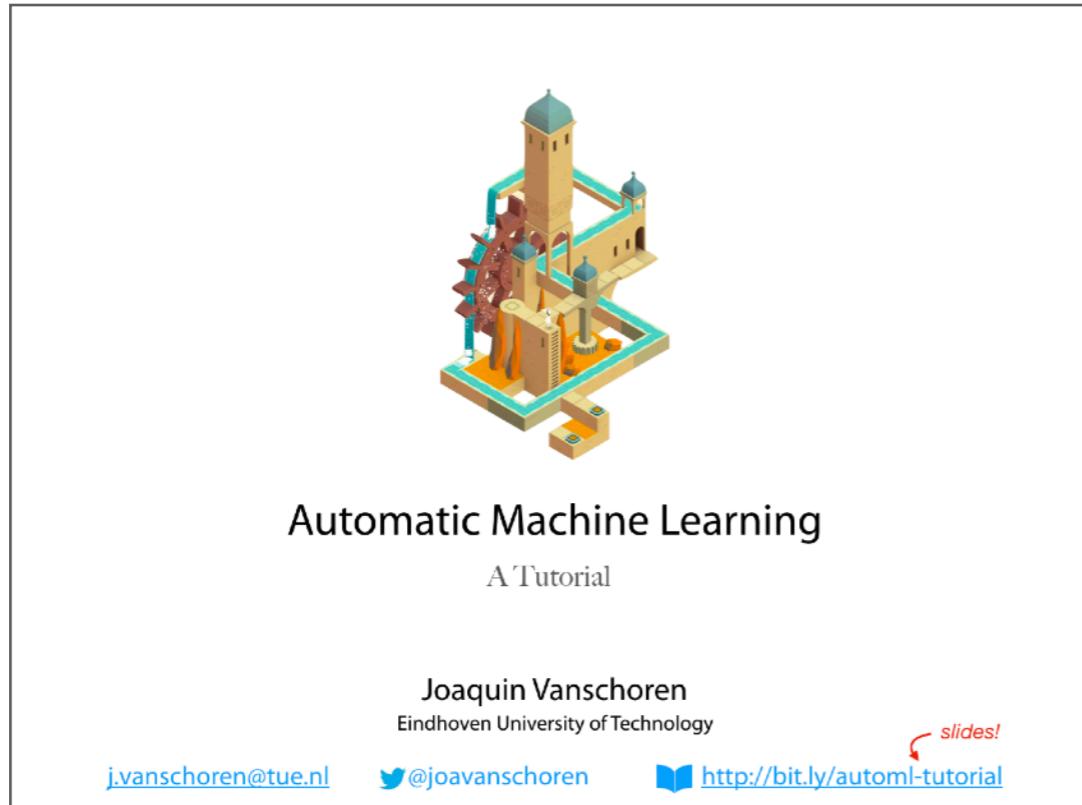
j.vanschoren@tue.nl

 [@joavanschoren](https://twitter.com/joavanschoren)

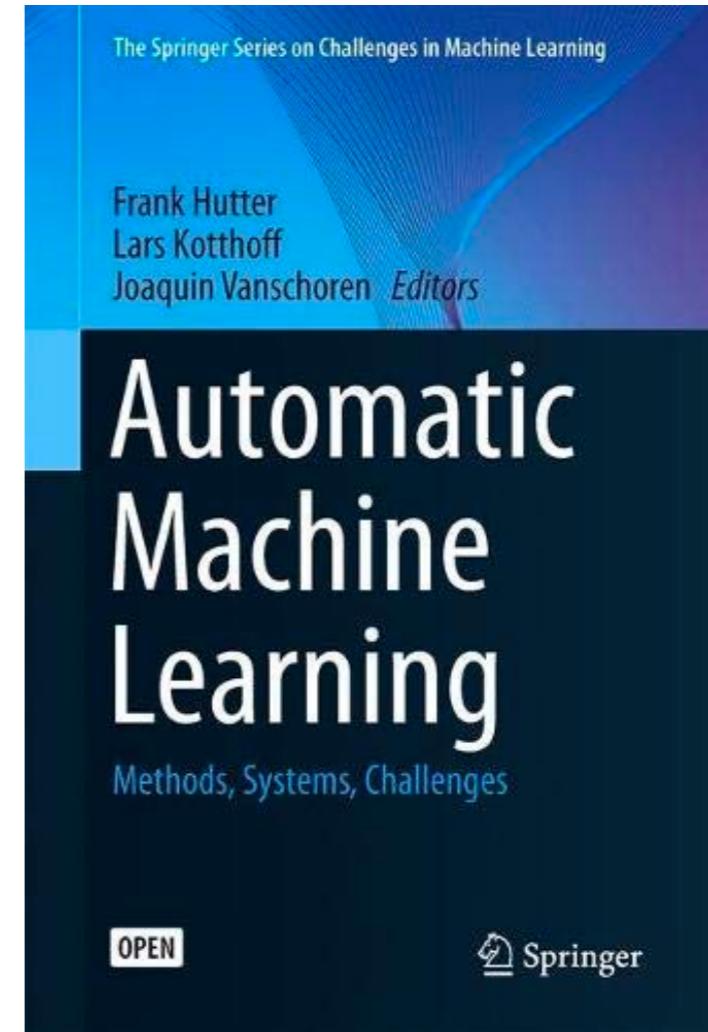
 <http://bit.ly/automl-course>

slides!

Slides / Book



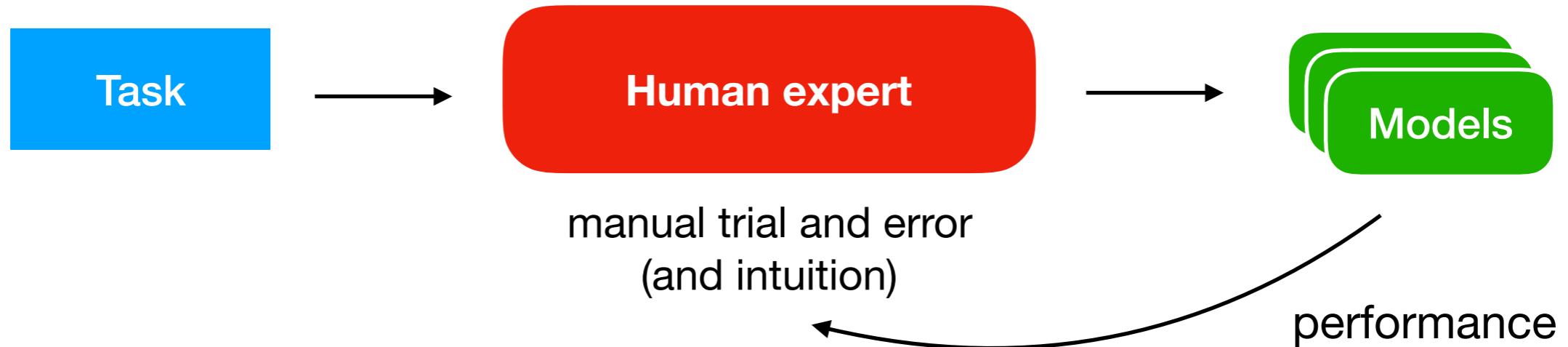
Slides
<http://bit.ly/automl-course>



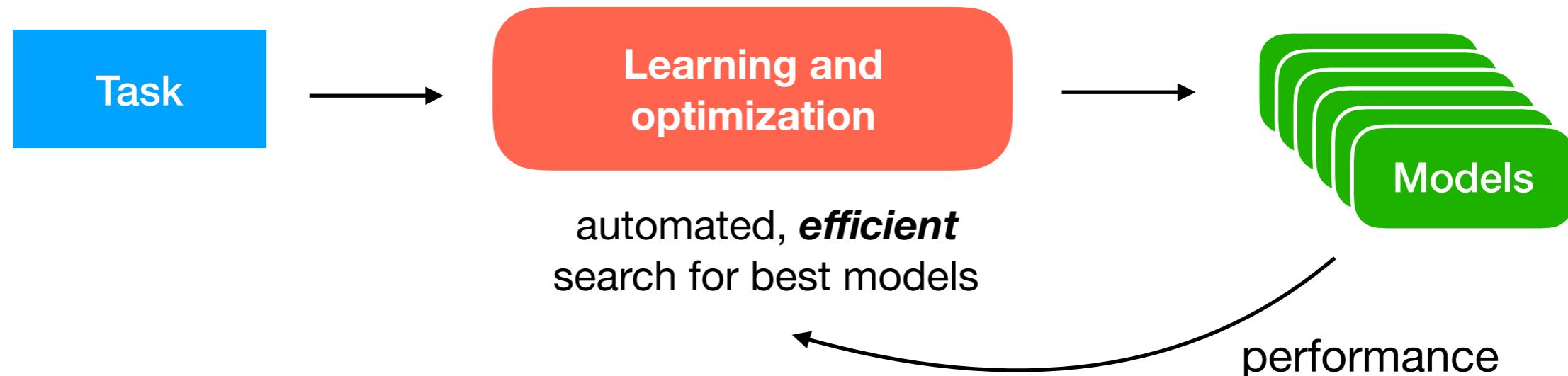
Open access book
(free PDF)
www.automl.org/book
www.amazon.de/dp/3030053172

Automated machine learning

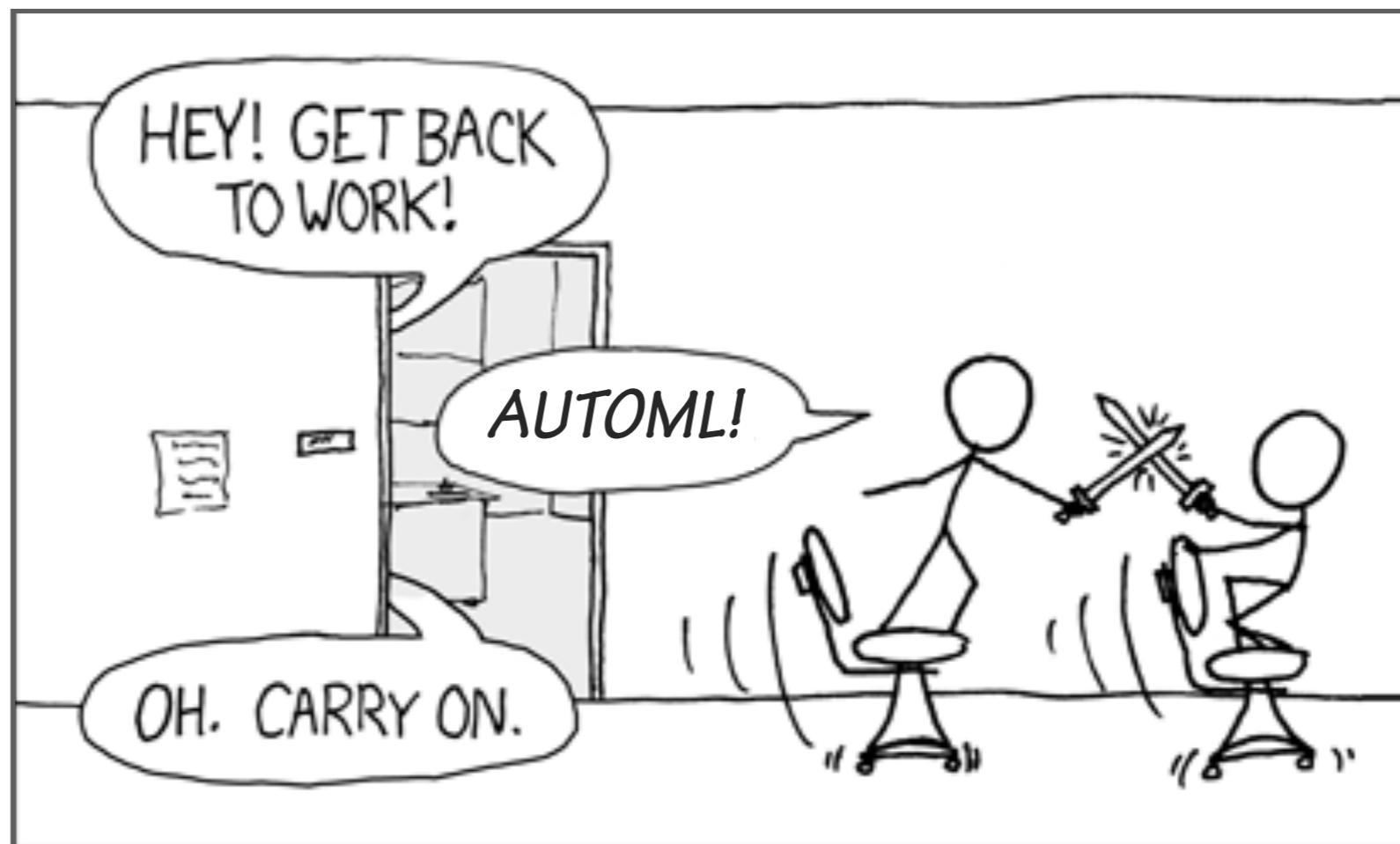
Manual machine learning



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

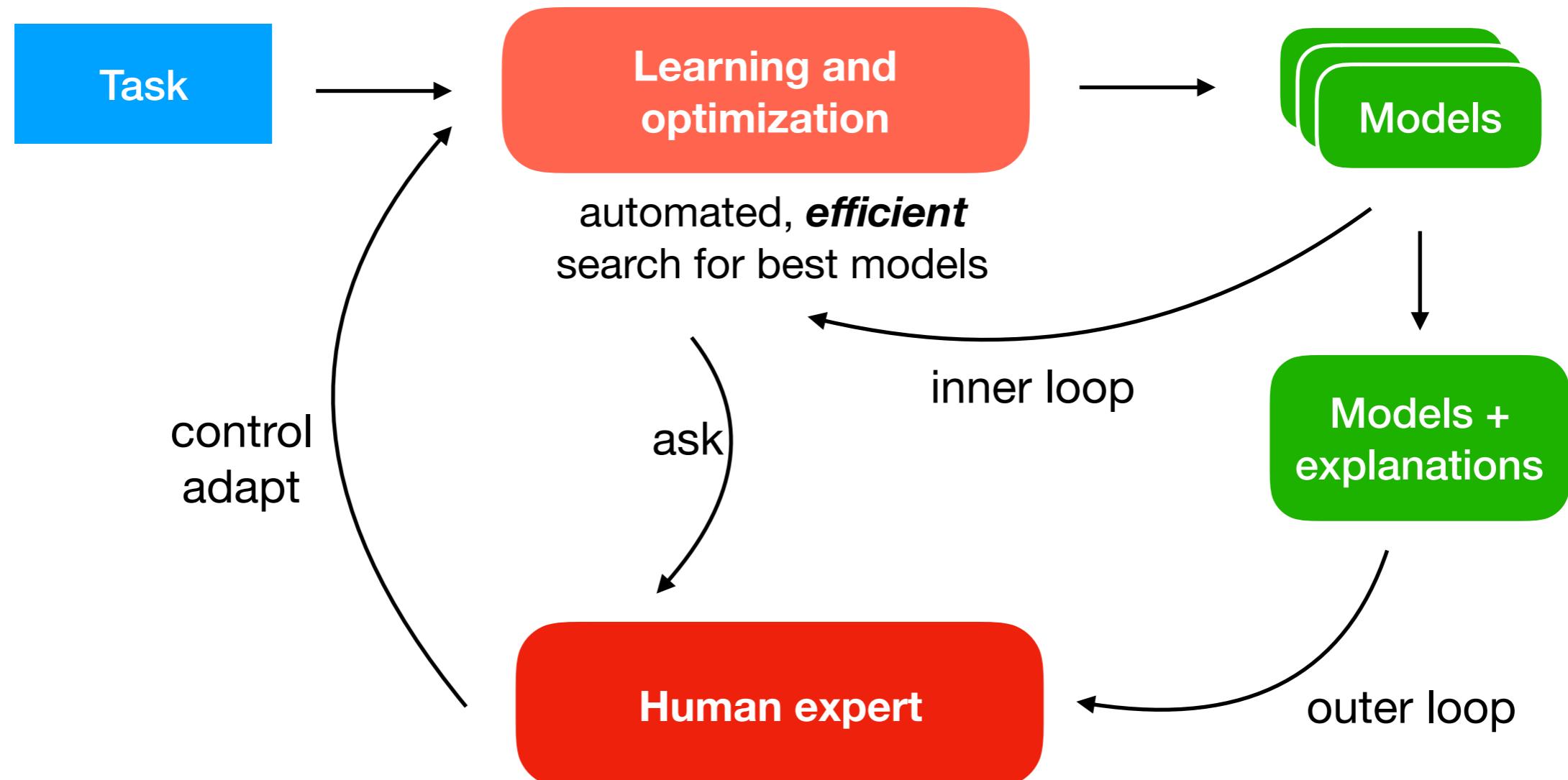


*THE DATA SCIENTIST'S #1 EXCUSE FOR
LEGITIMATELY SLACKING OFF:
“THE AUTOML TOOL IS OPTIMIZING MY MODELS!”*



Semi-Automated machine learning

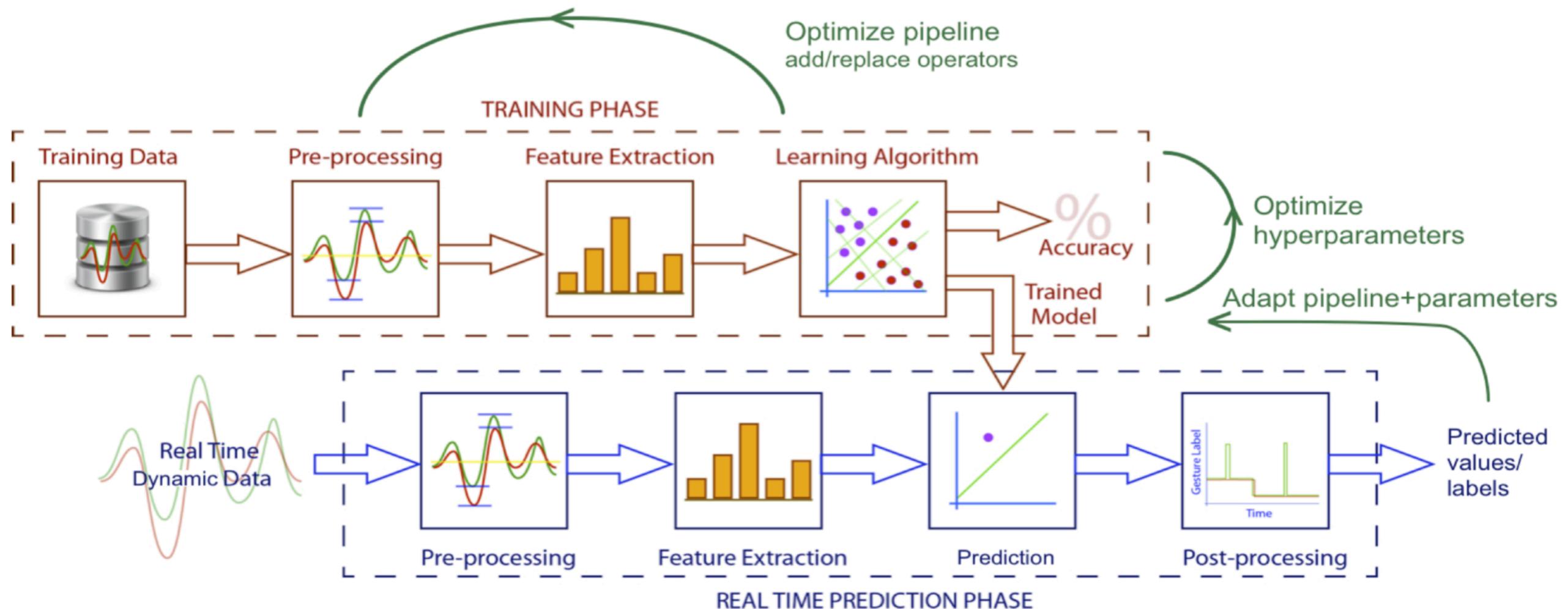
Human-in-the-loop



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

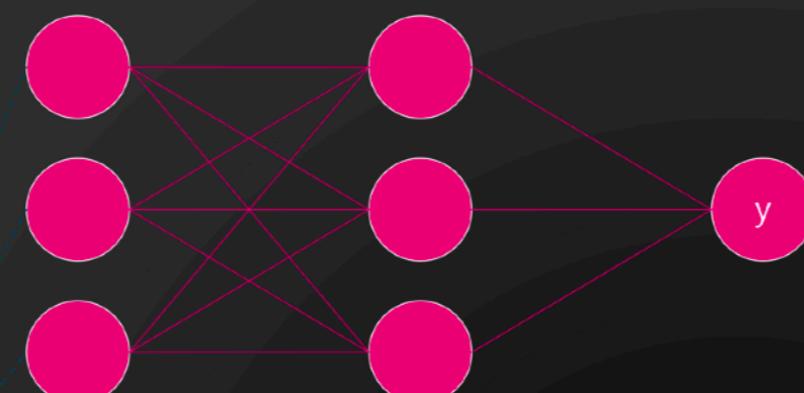


- Excellent tools (e.g. scikit-learn), but little guidance
- Cleaning, preprocessing, feature selection/engineering features, model selection, hyperparameter tuning, adapting to concept drift,...

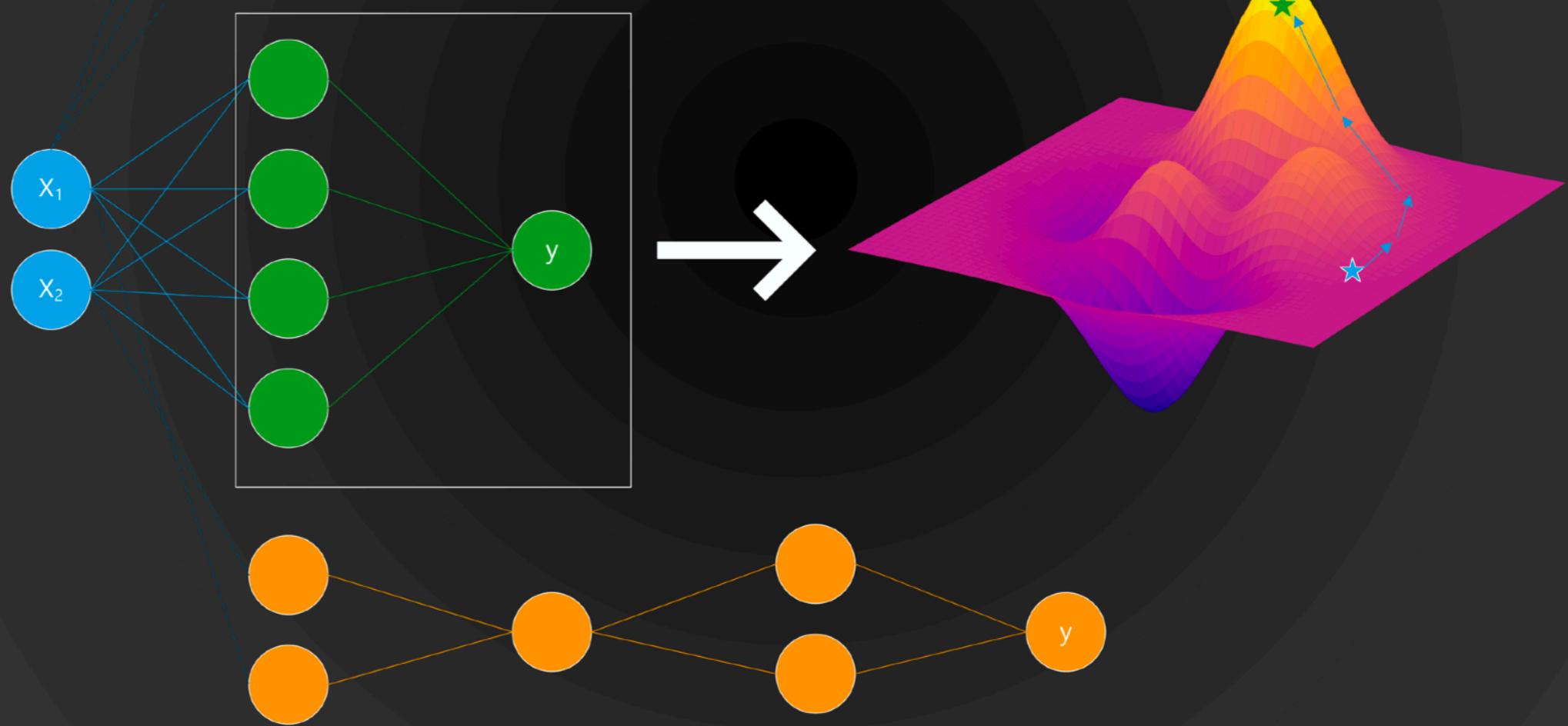
Doing machine learning requires a lot of expertise

Neural Architecture Search (NAS)

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



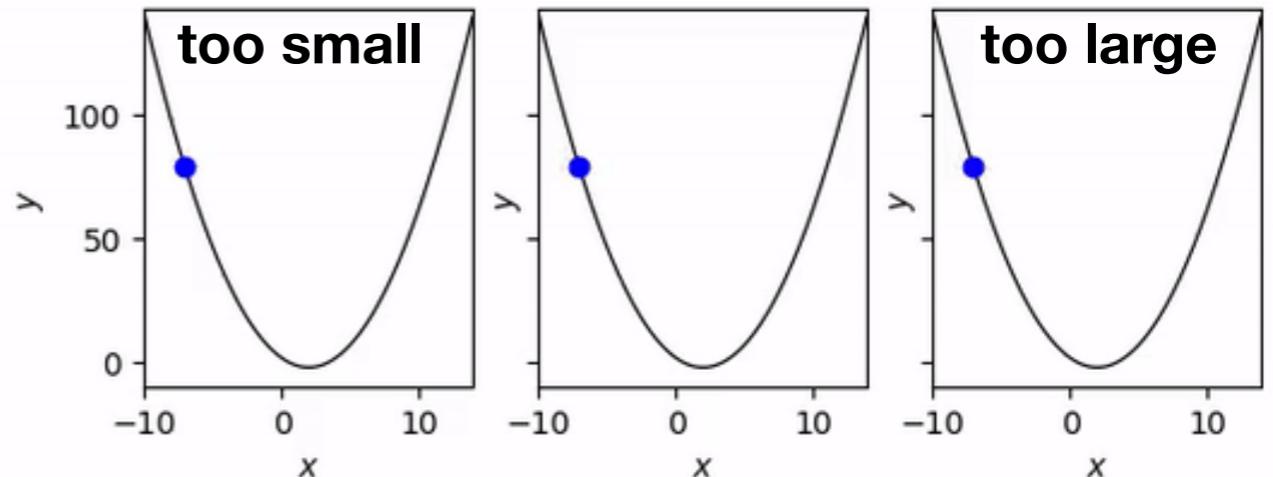
- Gradient descent hyperparameters
- Regularization
- ...



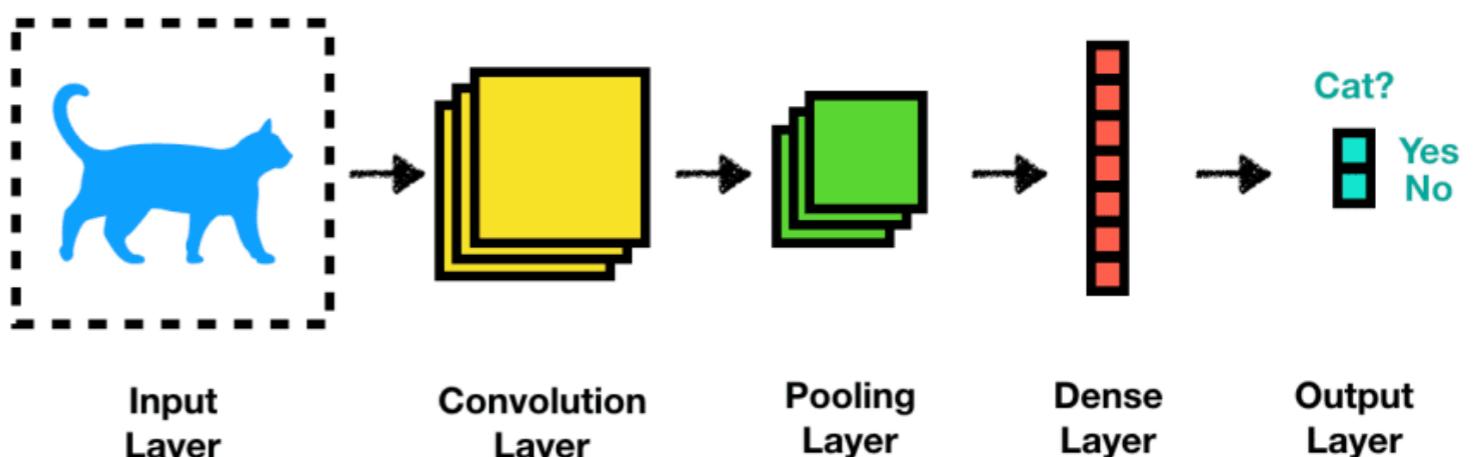
Hyperparameters

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

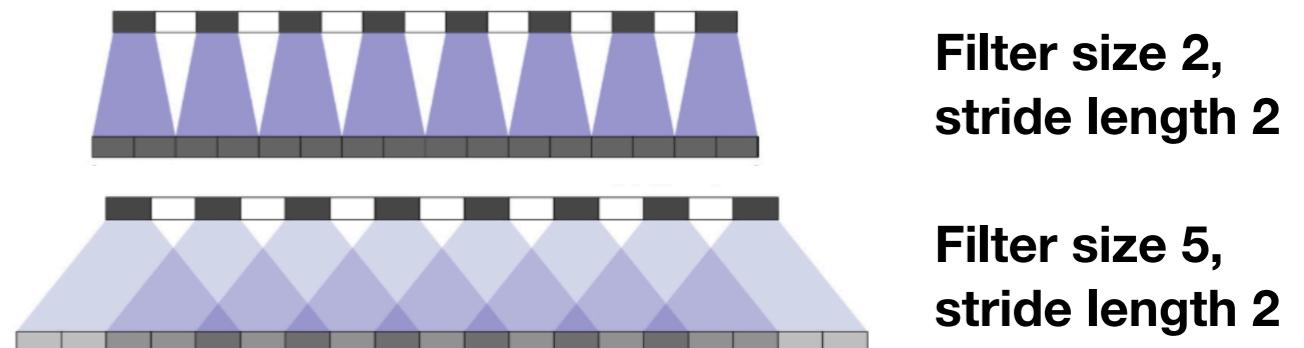
- Numeric
 - e.g. learning rate



- Categorical
 - e.g. layer type



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

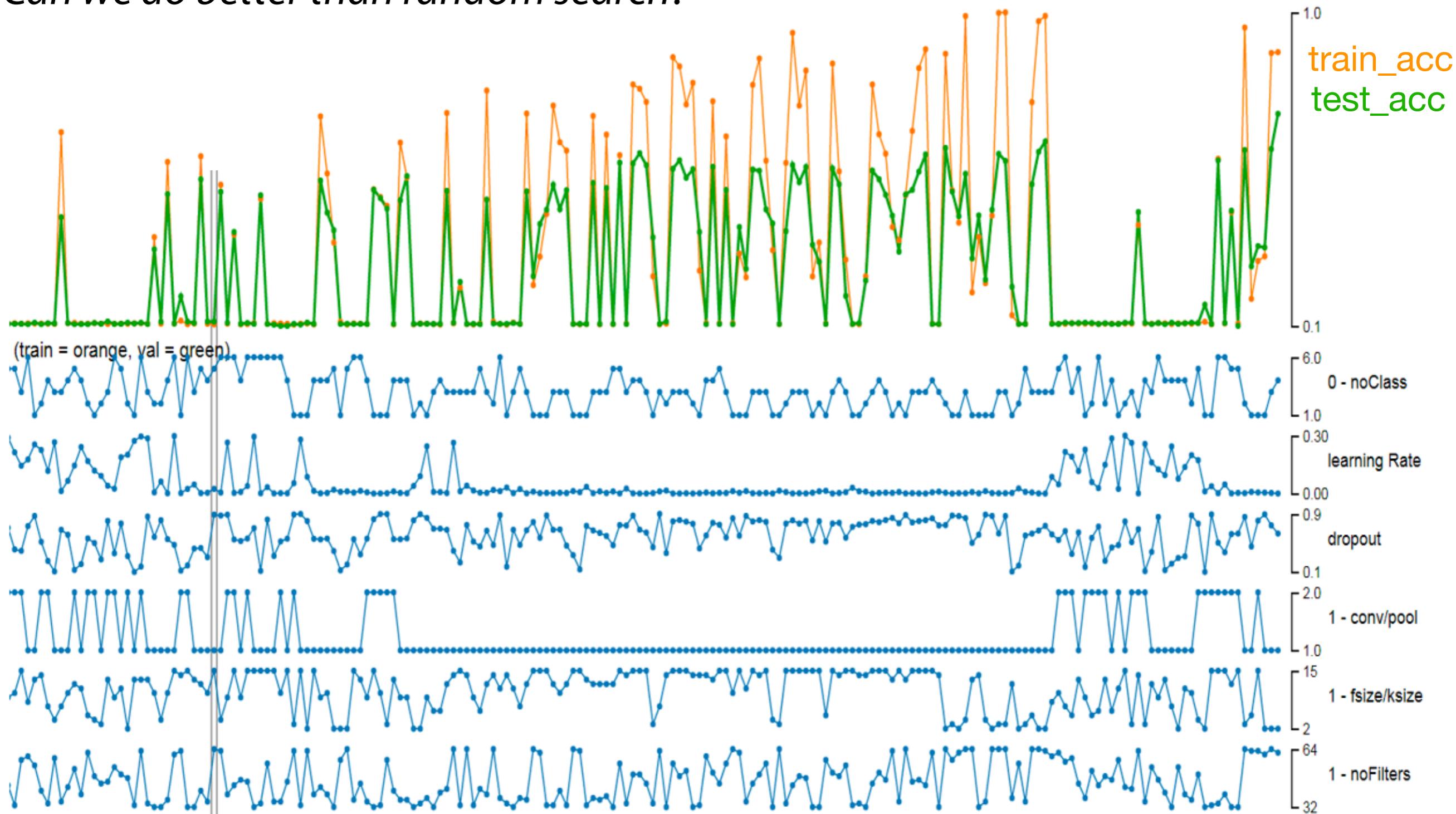


Hyperparameters

Some are very sensitive, others not. Many interact.

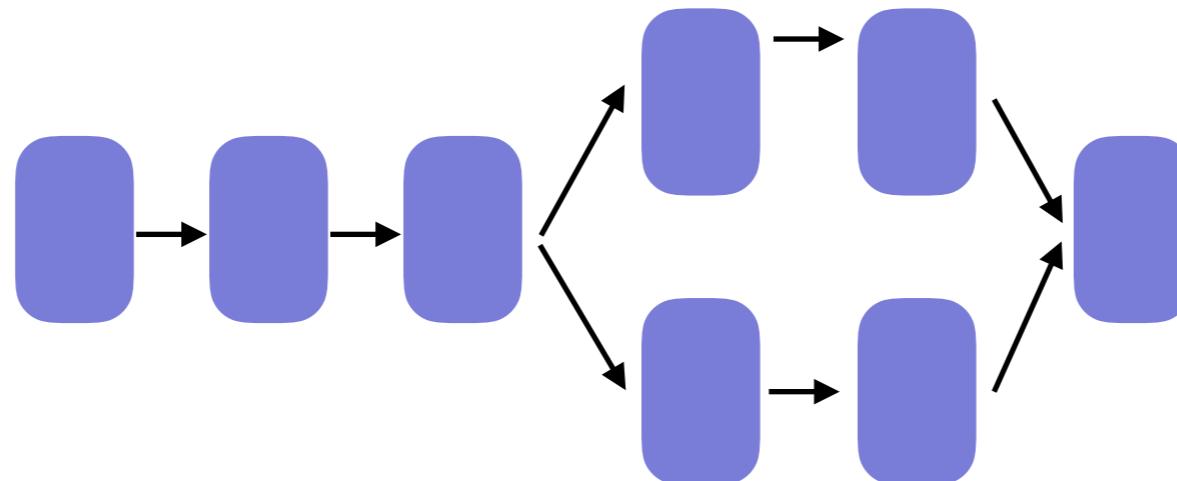
What is the right strategy to find the right **configuration**?

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



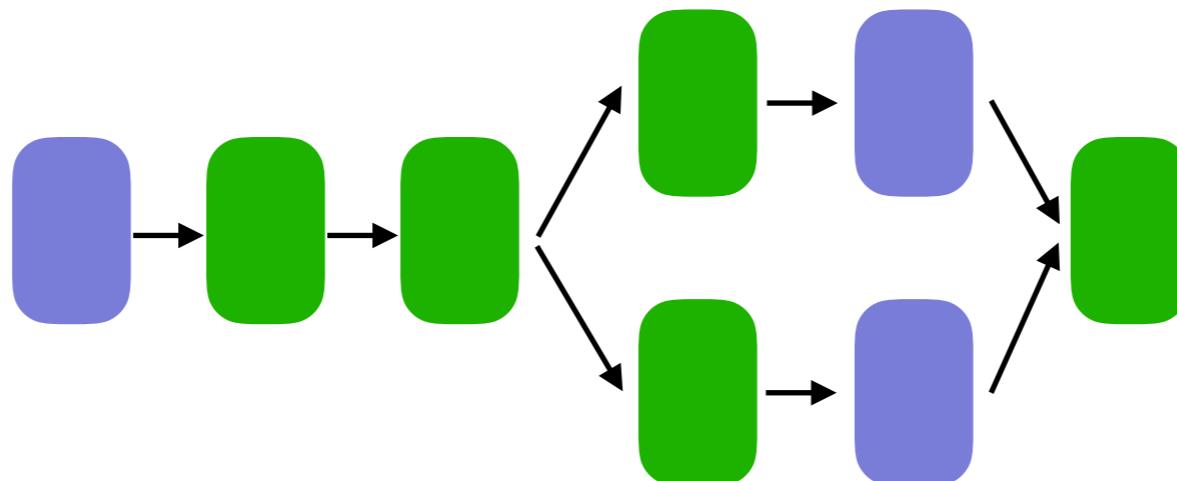
```
make_pipeline(  
    OneHotEncoder(),  
    Imputer(),  
    StandardScaler(),  
    SVC())
```



```
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

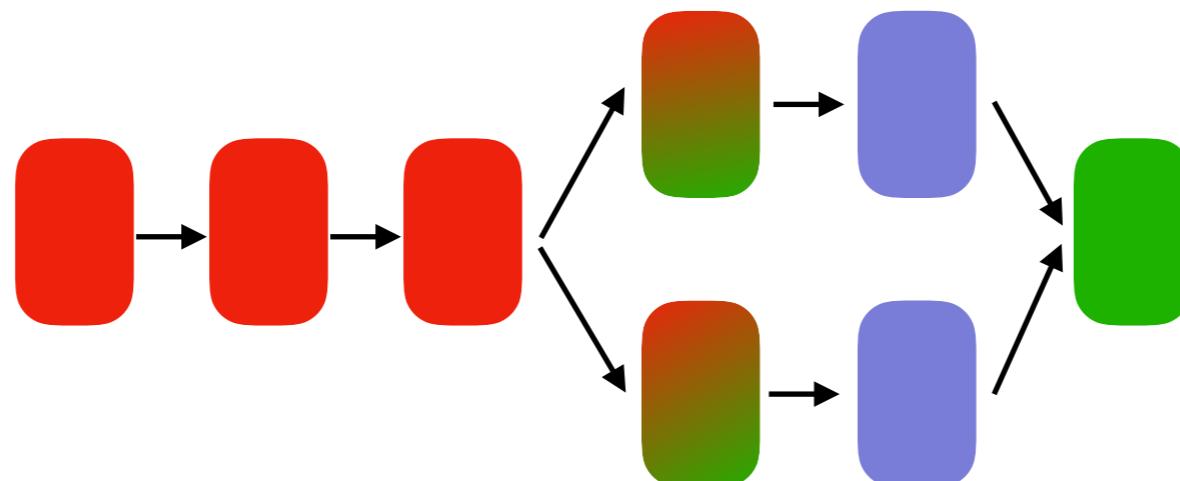
- **Architecture search:** represent and search all possible architectures
- **Hyperparameter optimization:**
 - Which hyperparameters are important? How to optimize them?
 - What is the (multi-)objective function?



```
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)

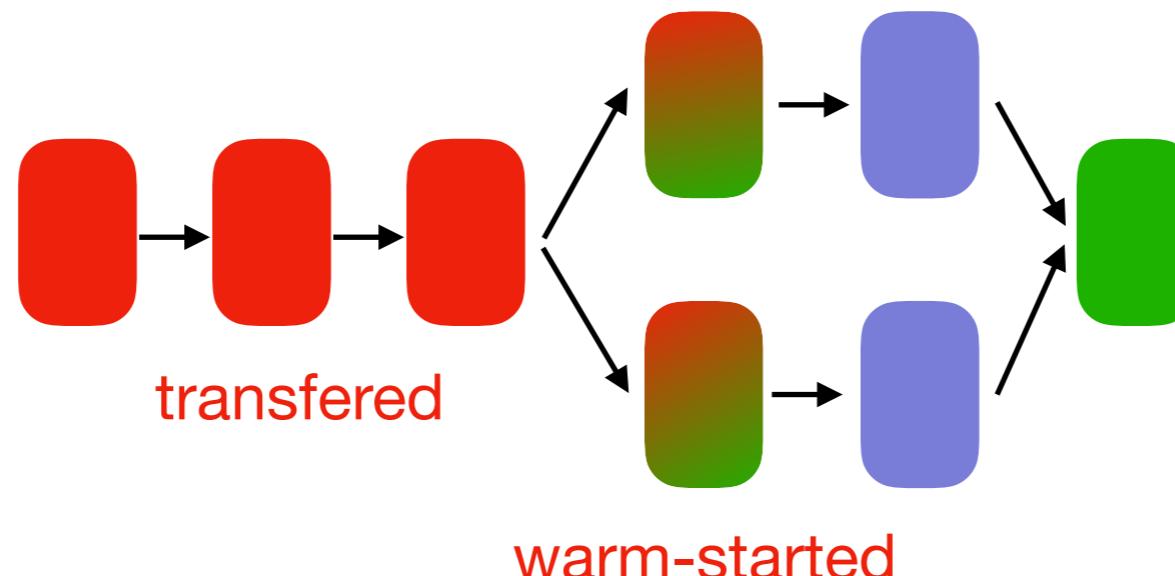


Goal:

```
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?



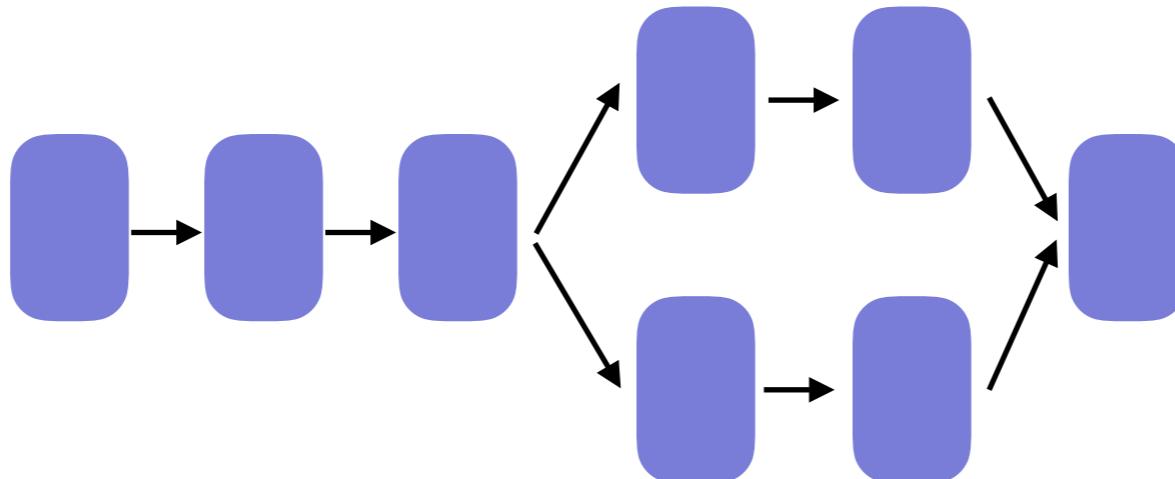
Many combinations are possible!

They can be done *consecutively, simultaneously or interleaved*

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

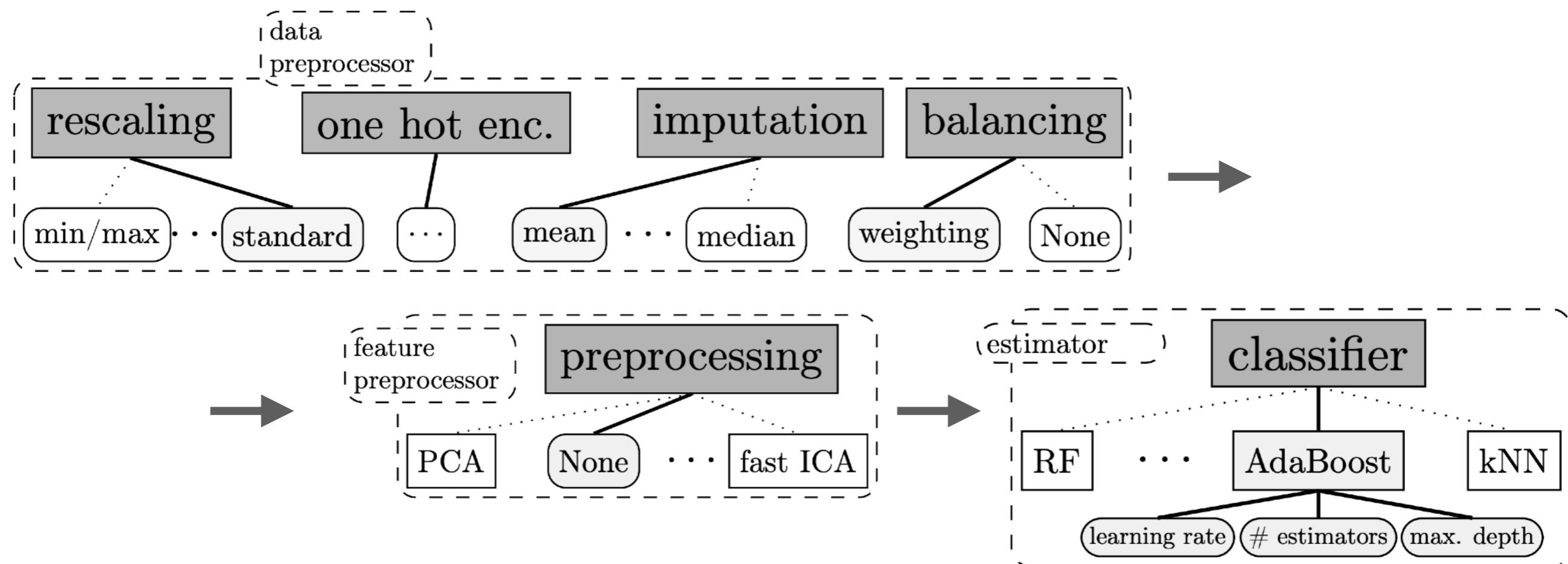
AutoML: subproblems

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



Parameterized architecture

- 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

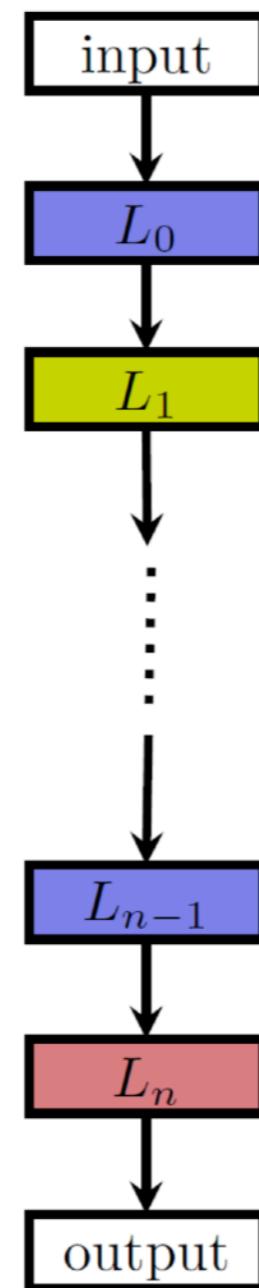
Parameterized neural architectures

Parameterized Sequential

Choose:

- number of layers
- type of layers
 - dense
 - convolutional
 - max-pooling
 - ...
- hyperparameters of layers

+ easier to search
- sometimes too simple

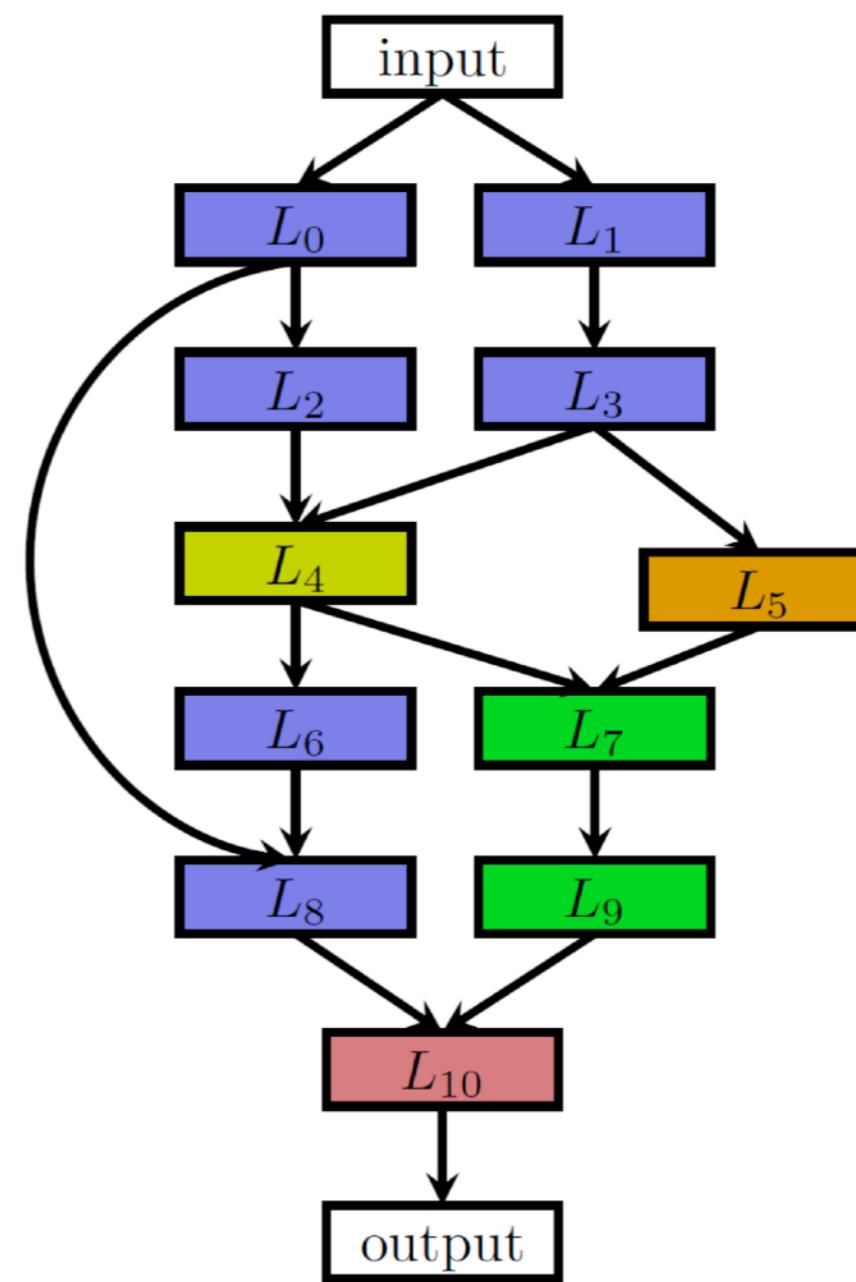


Parameterized Graph

Choose:

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

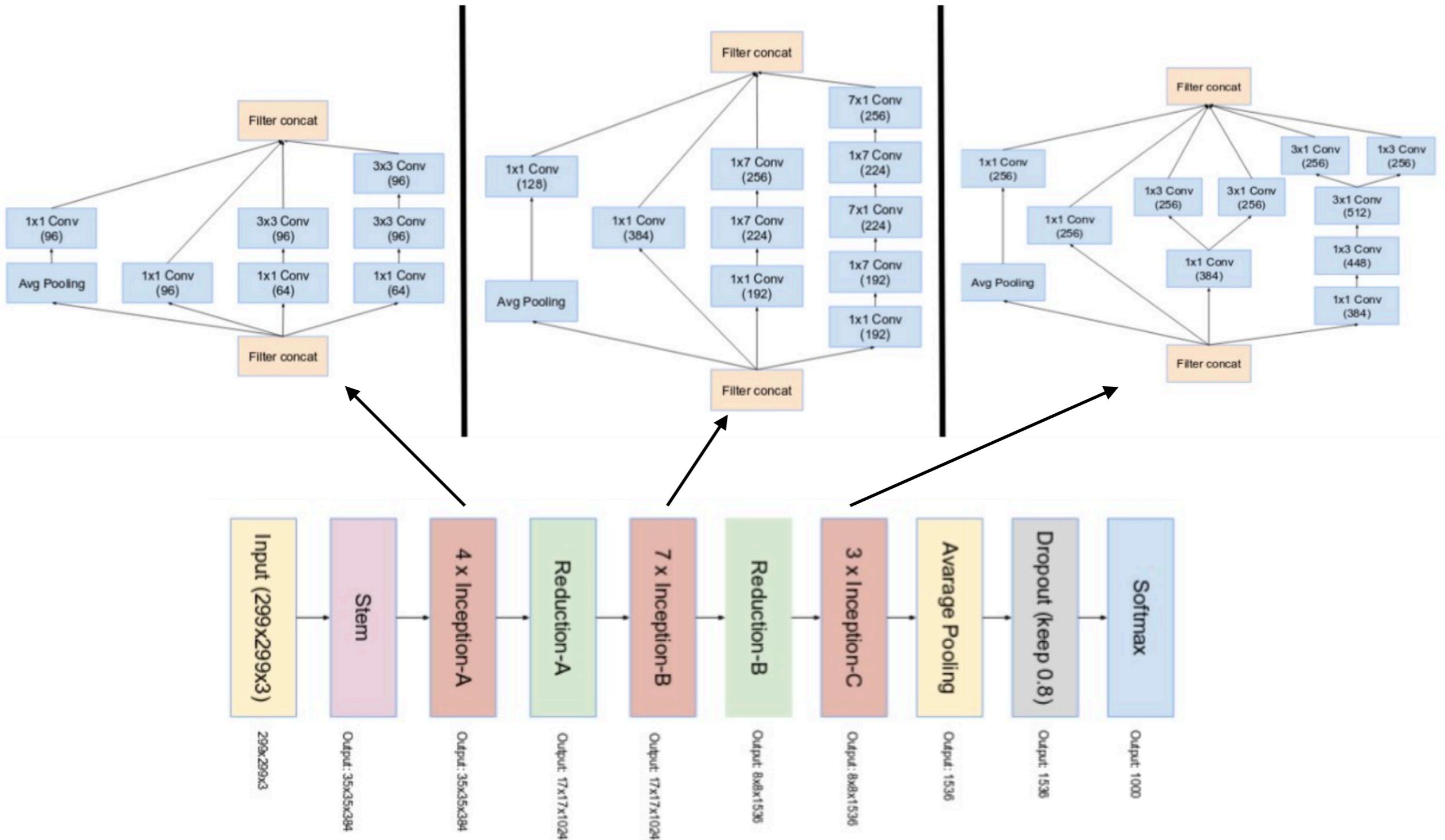
+ more flexible
- much harder to search



Neural Architecture Search

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

e.g. Inception v4:



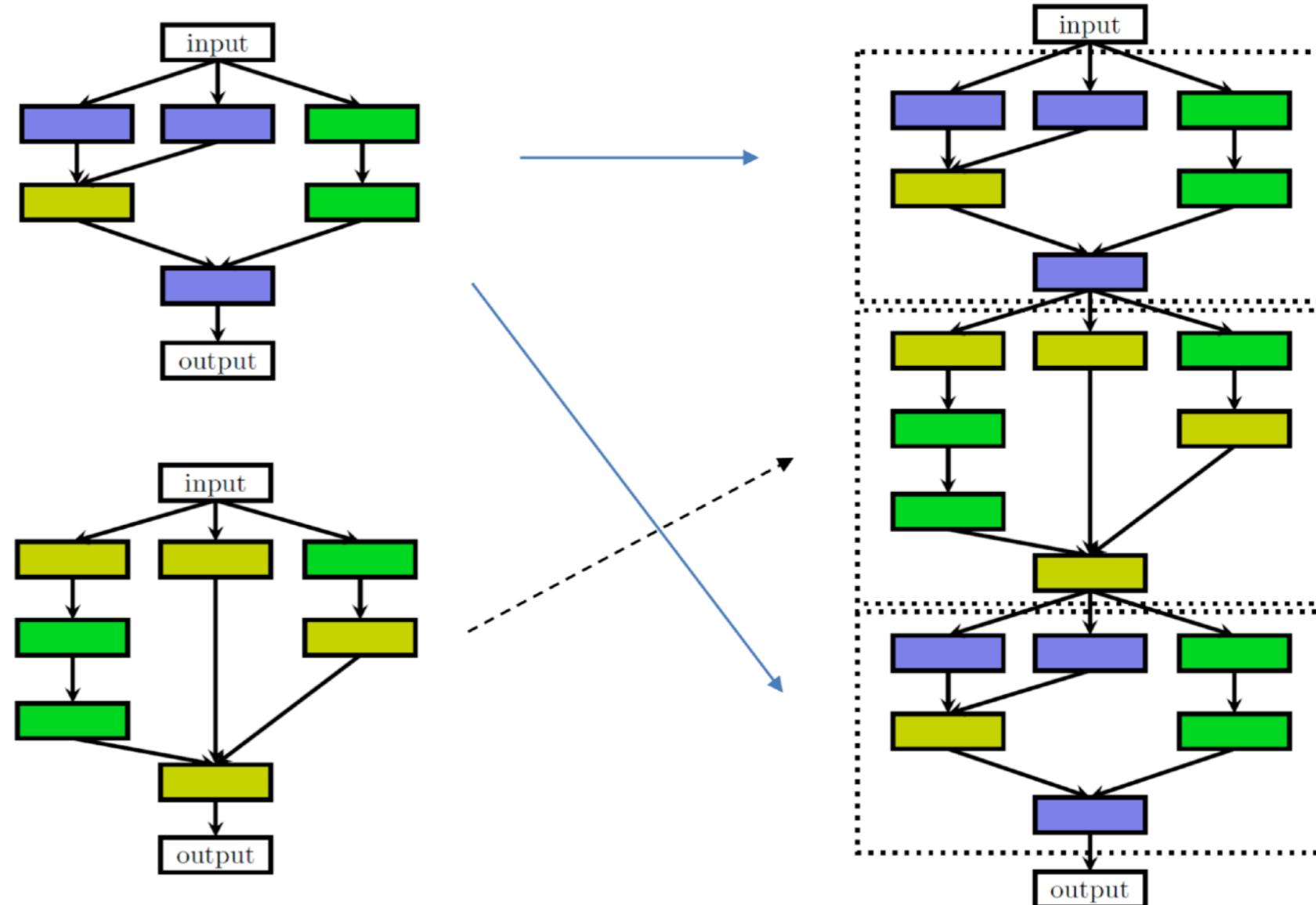
Meta-architectures

Compositionality: learn hierarchical building blocks to simplify the task

Cell search space

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

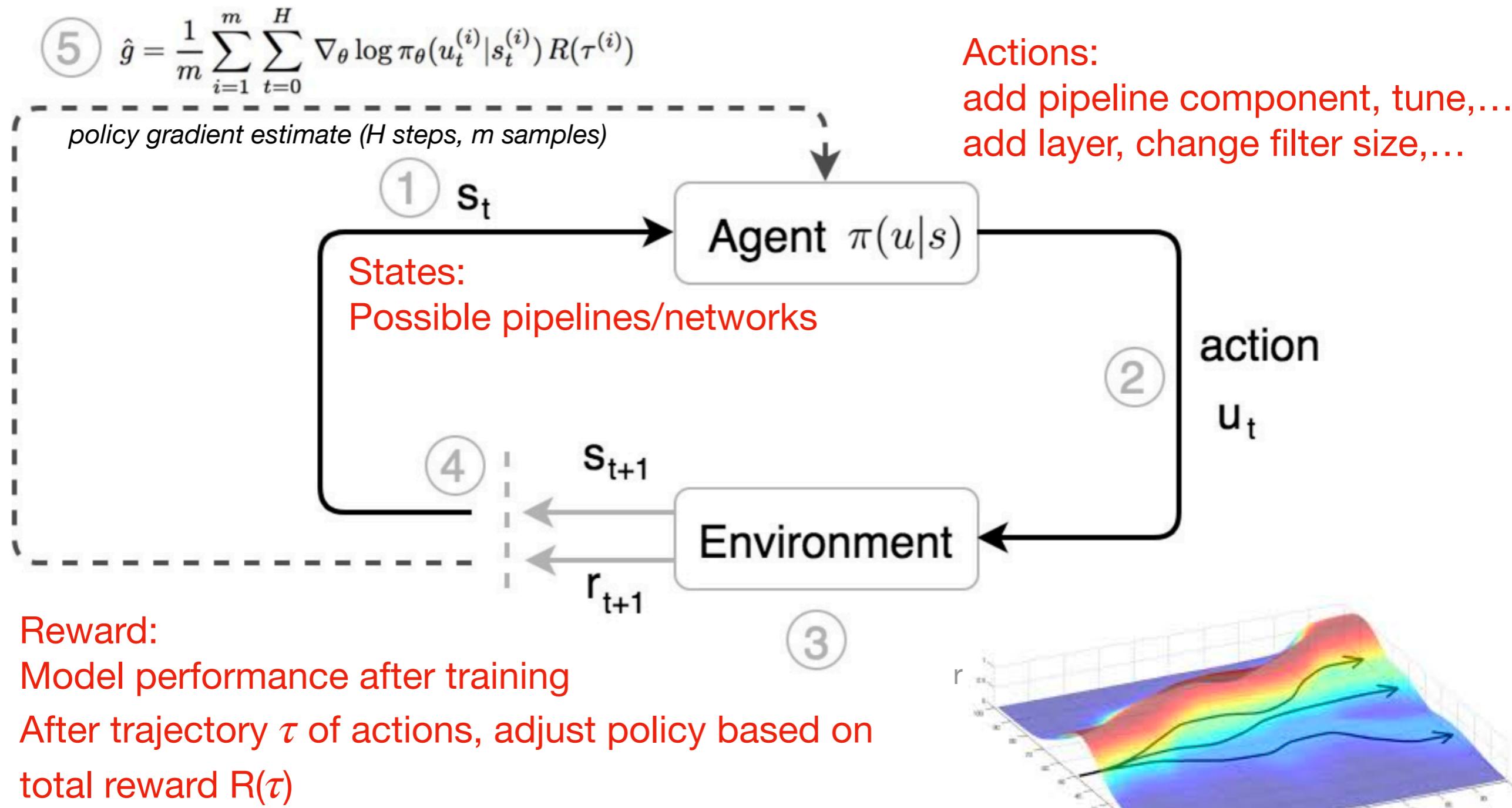
- + smaller search space
- + cells can be learned on a small dataset & transferred to a larger dataset
- strong domain priors, doesn't generalize well



Idea: pipeline search could also use compositionality/hierarchies?

Reinforcement learning

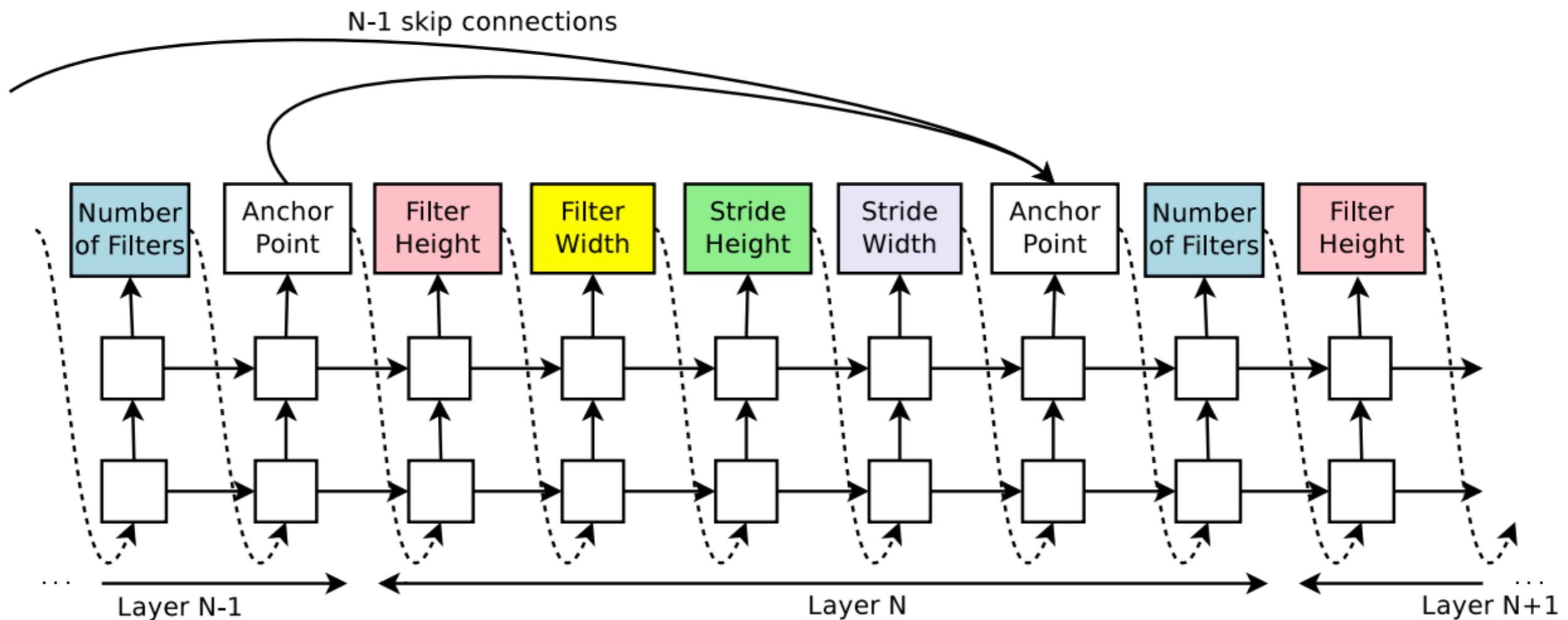
Build pipeline or network step-by-step, learn general strategy (policy)



NAS with Reinforcement learning

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

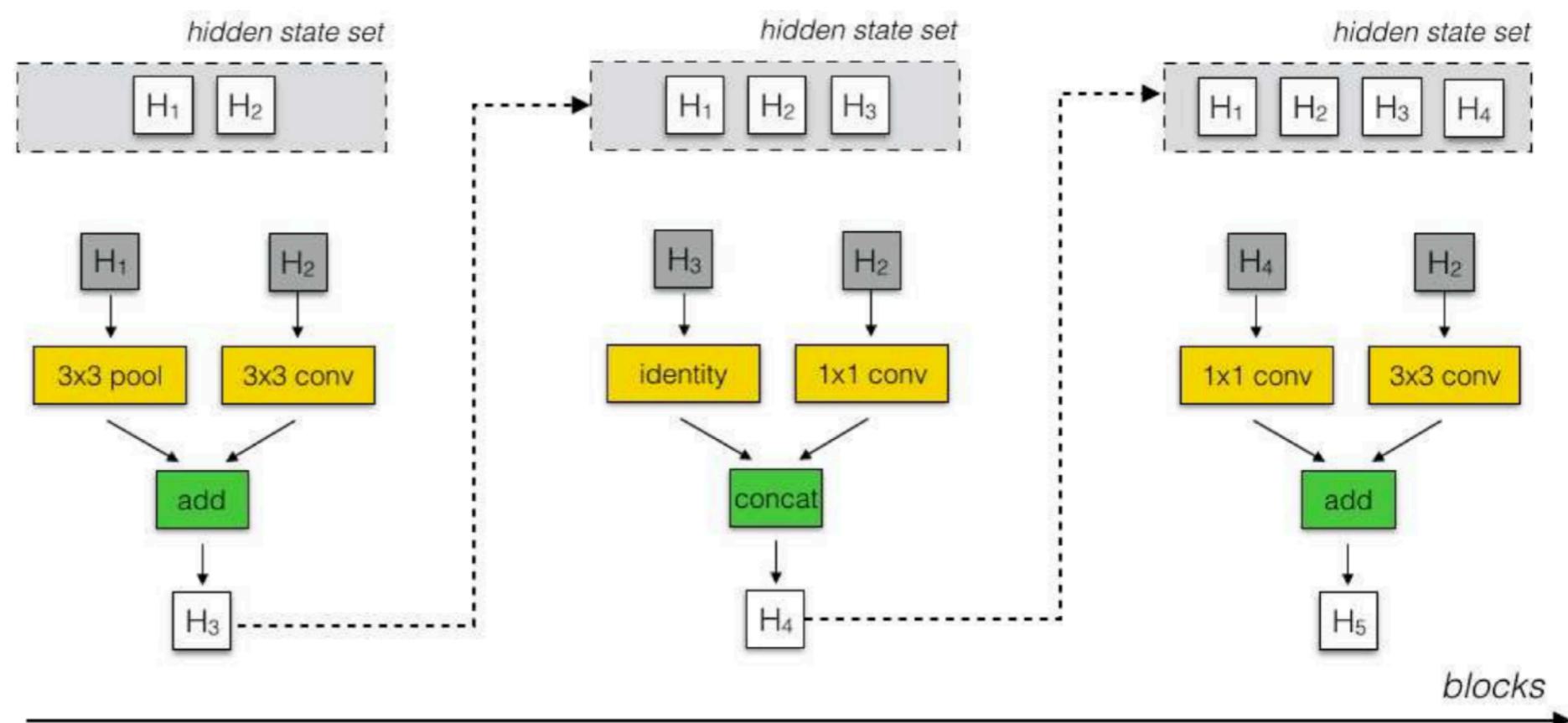
- State of the art on CIFAR-10, Penn Treebank
- **800 GPUs for 3-4 weeks**, 12800 architectures



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

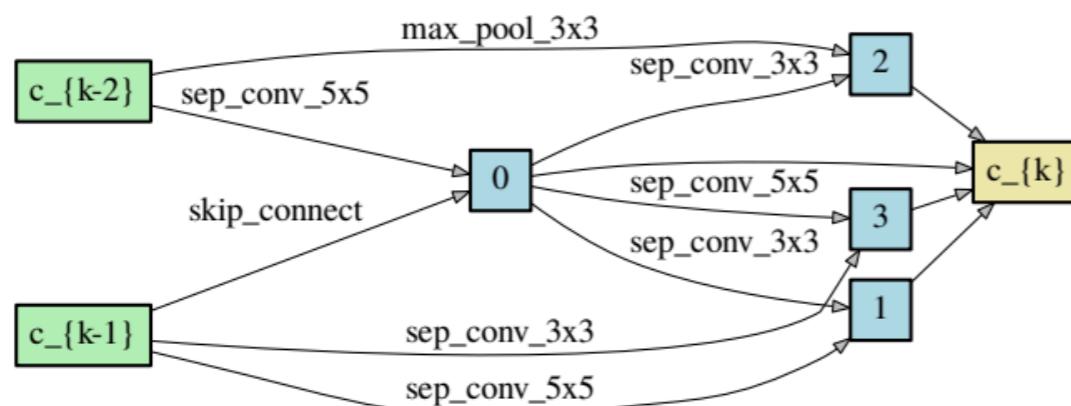


NAS with random search?

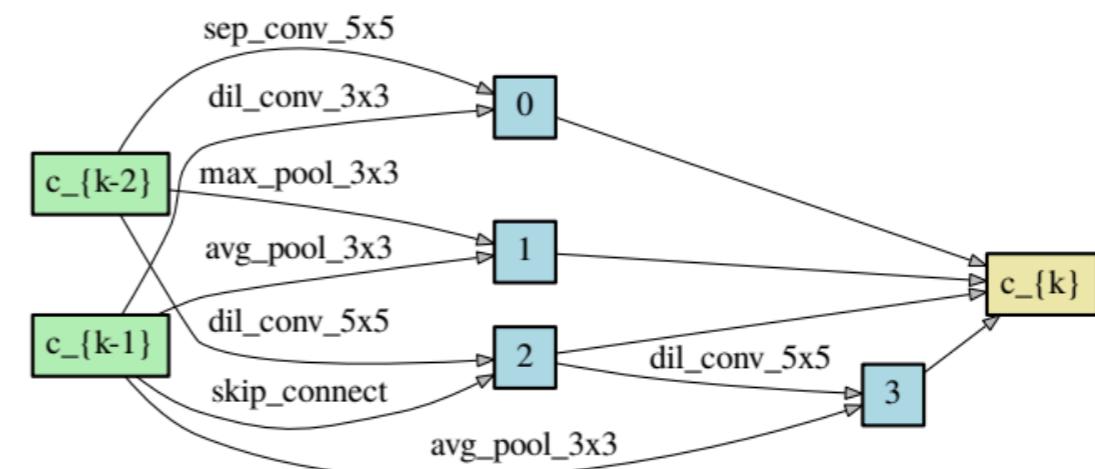
Yu et al. 2019

Real et al. 2019

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



(a) Normal Cell

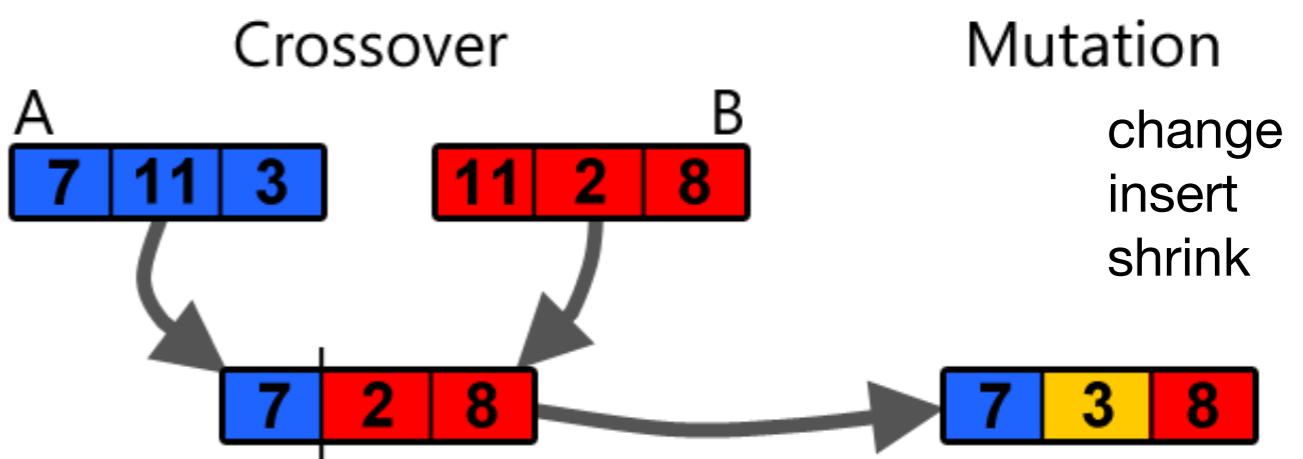


(b) Reduction Cell

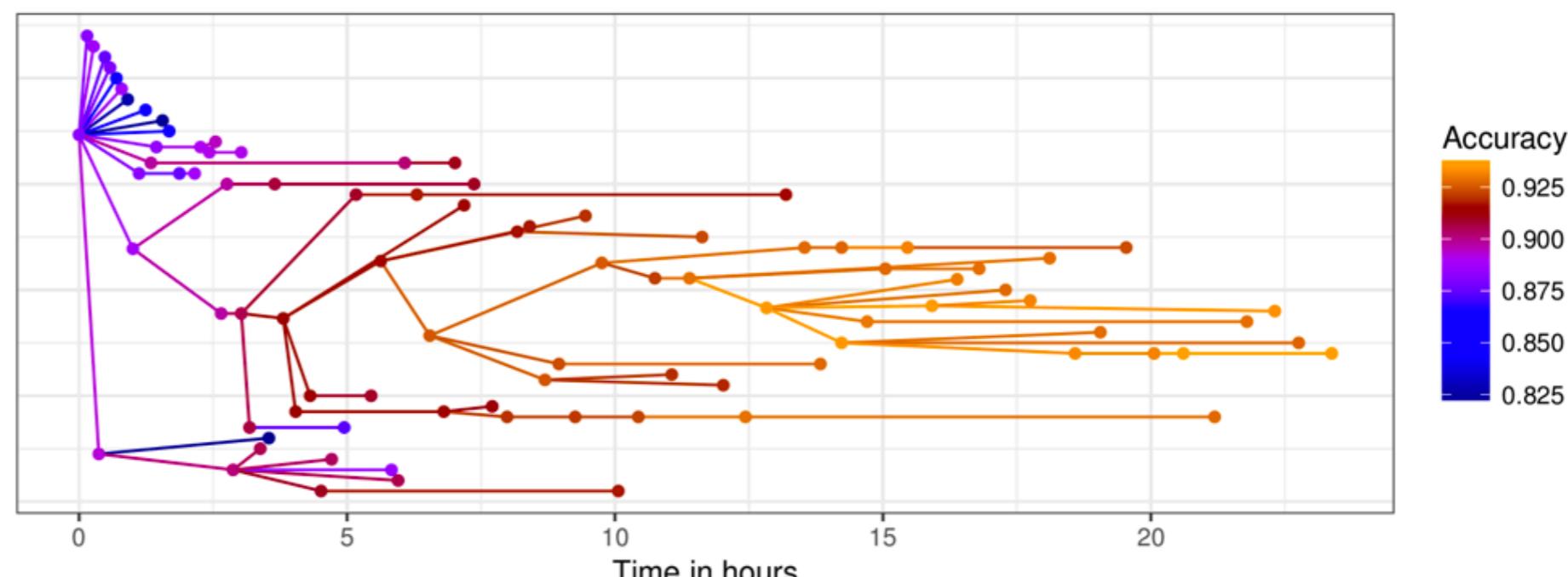
Convolutional Cells on CIFAR-10 Benchmark: Best architecture found by random search with weight-sharing.

Evolving pipelines

- 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

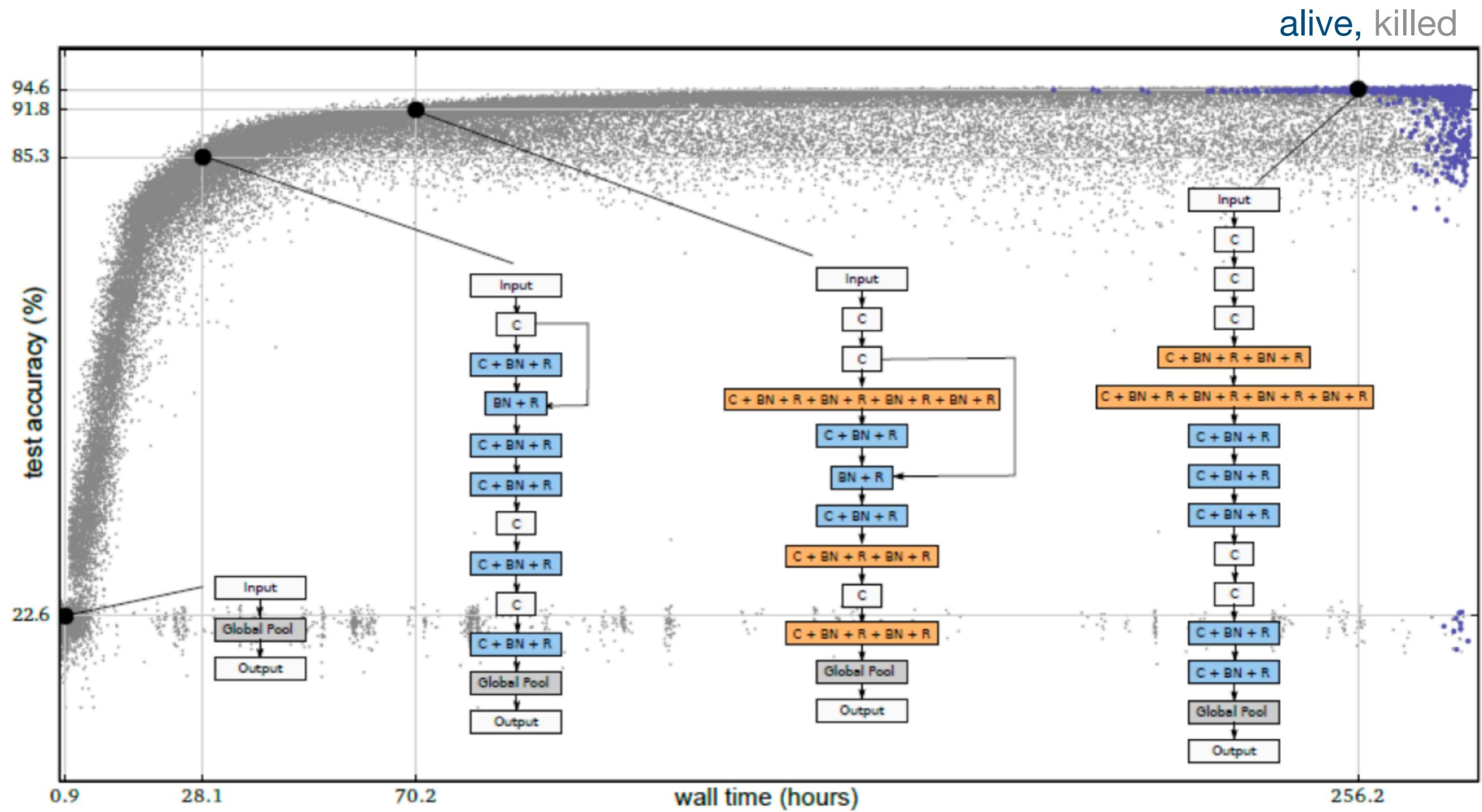


+ more flexible
- larger search space,
can be slower



Neuro-evolution

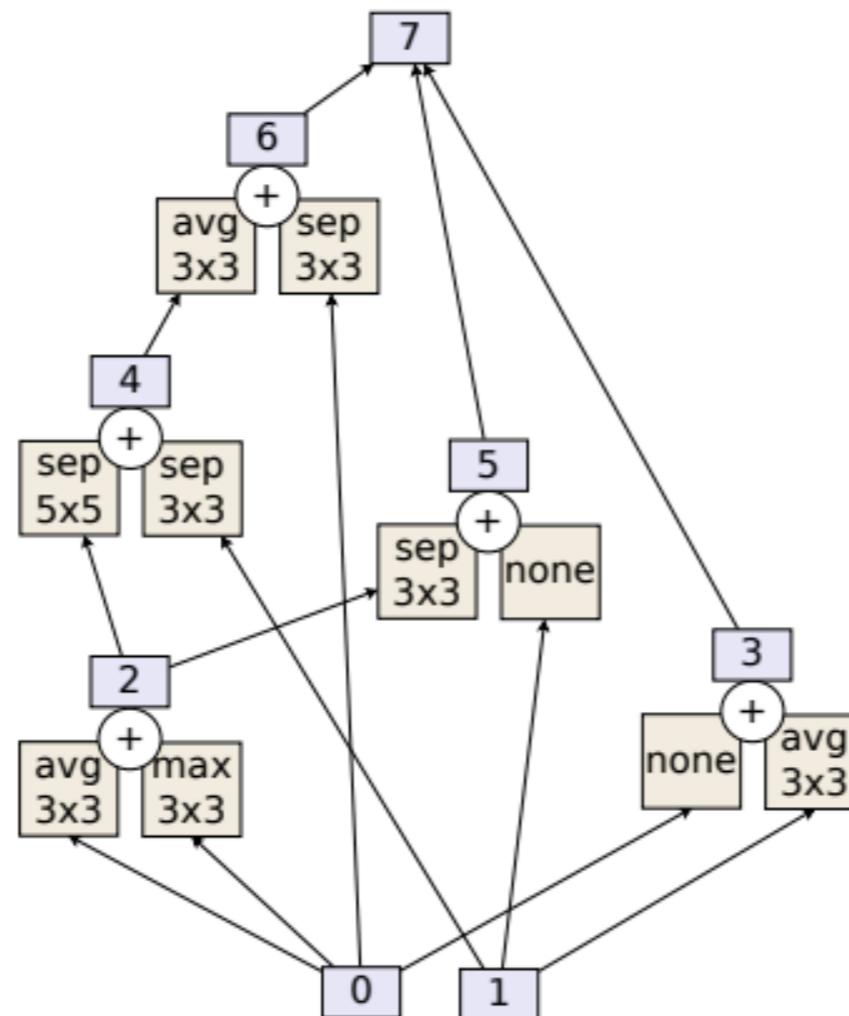
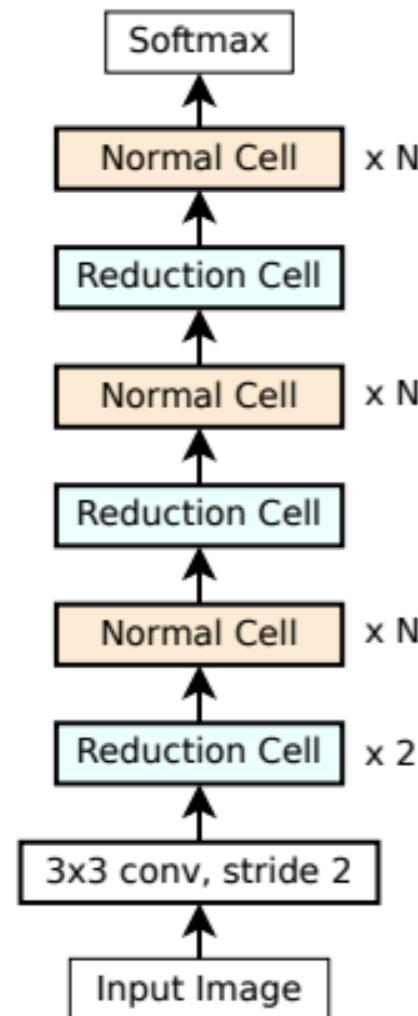
- same idea: learn the neural architecture through evolution
- mutations: add, change, remove layer



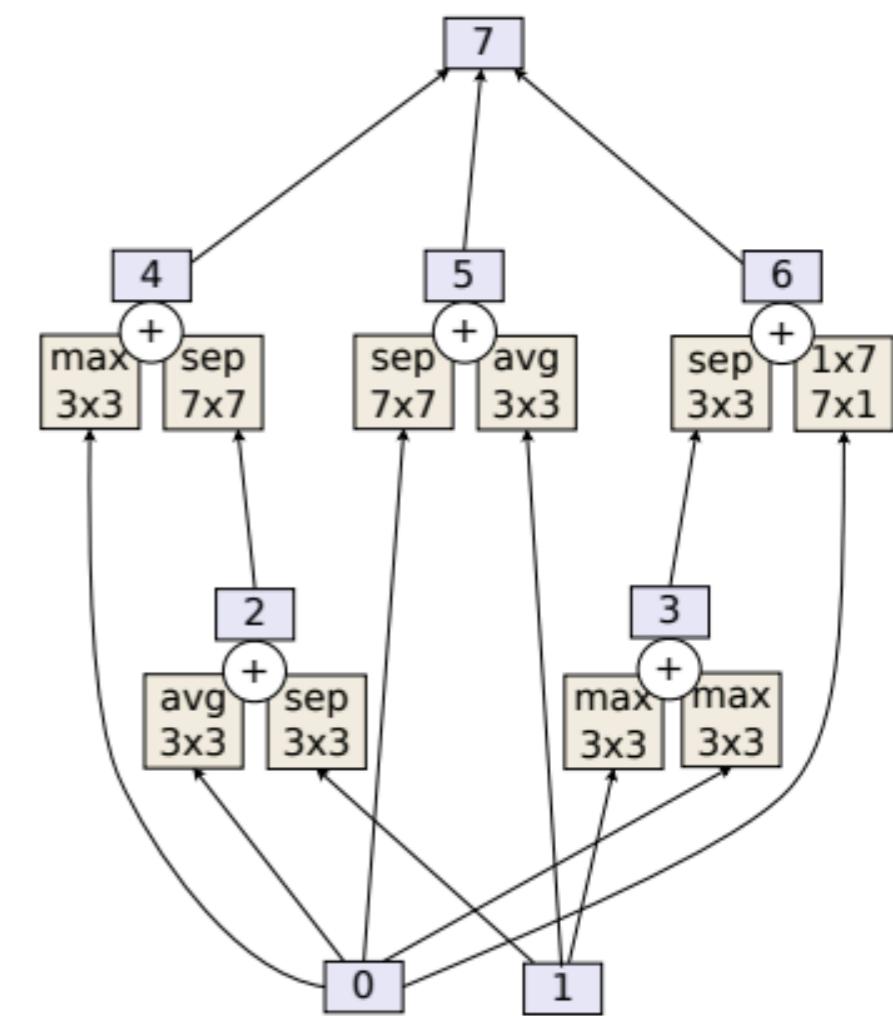
Neuro-evolution

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

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



normal cell

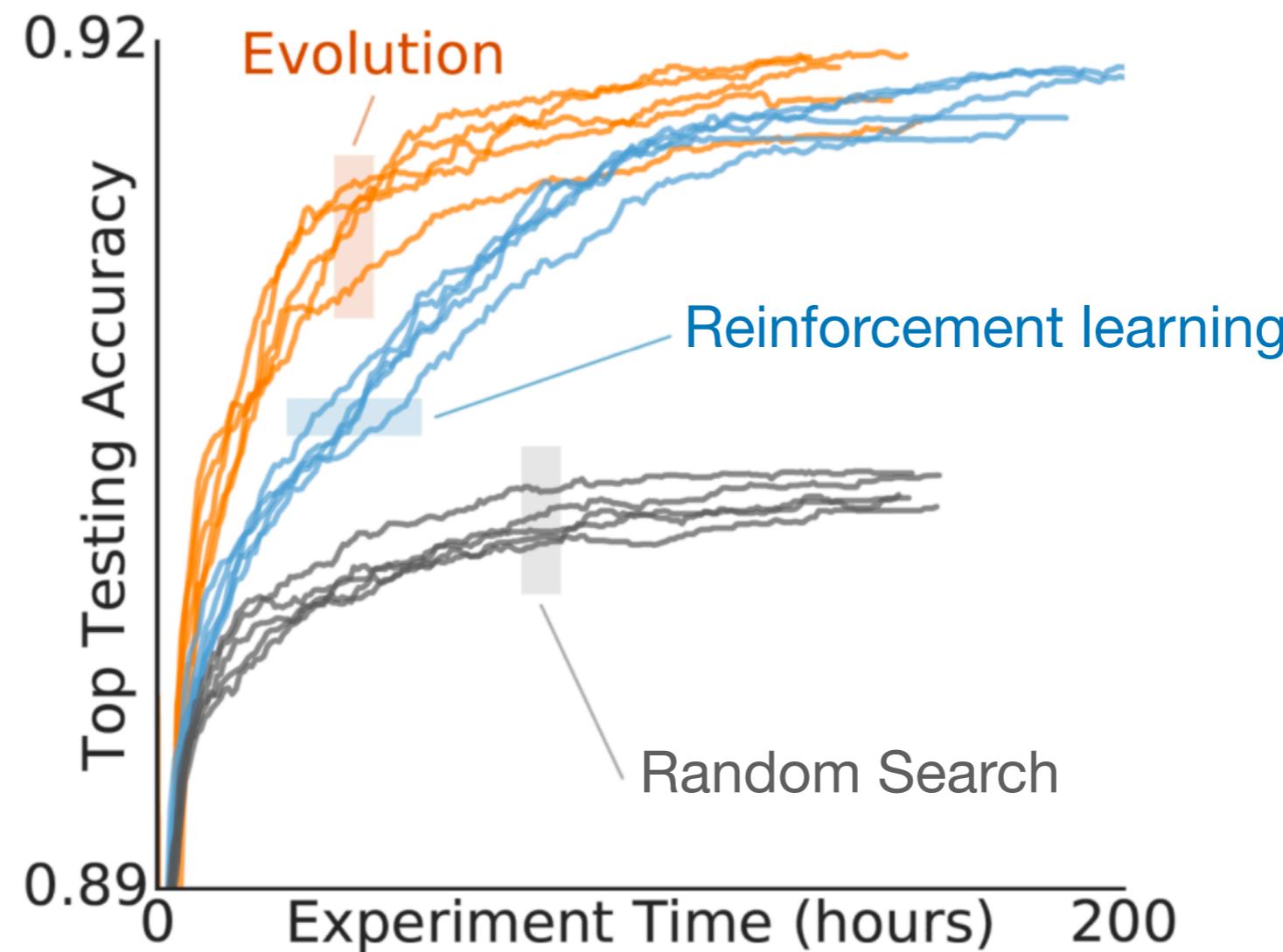


reduction cell

Neuro-evolution

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

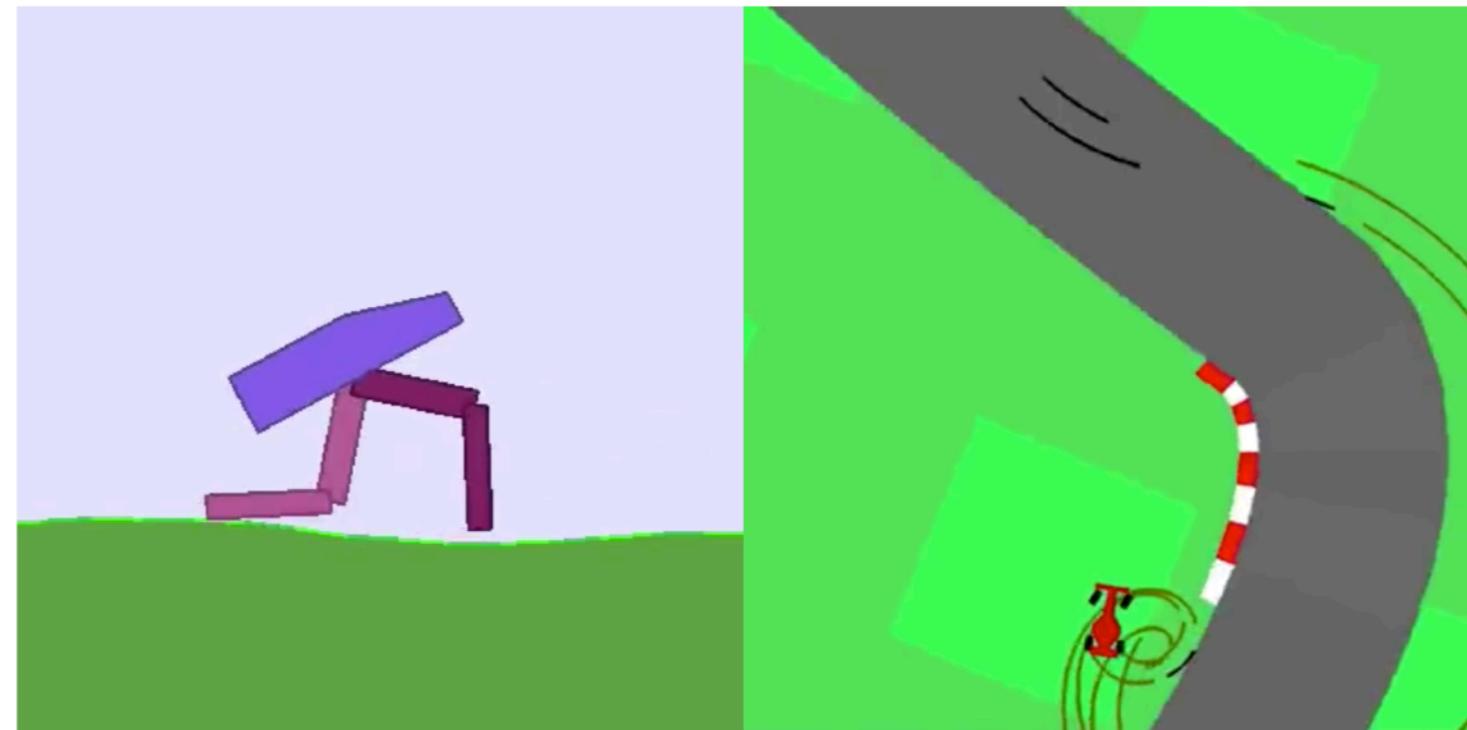
- Cell search space + aging evolution (kill oldest networks)
- More efficient than reinforcement learning



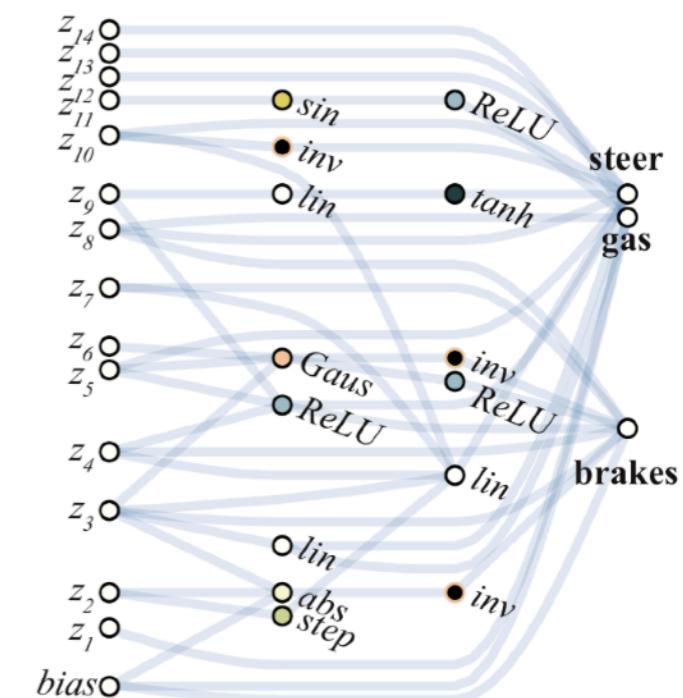
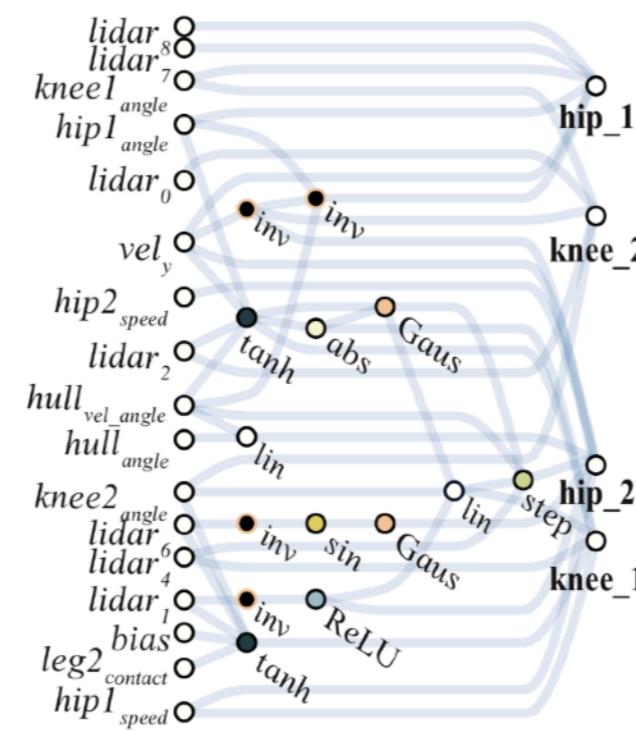
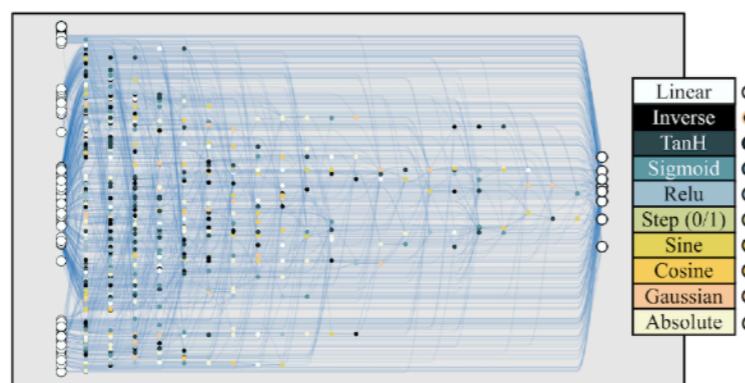
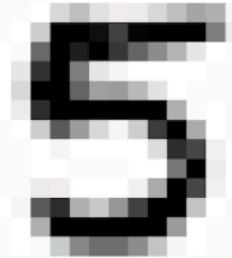
Weight-agnostic neural networks

Evolving the networks while training the model weights is expensive

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

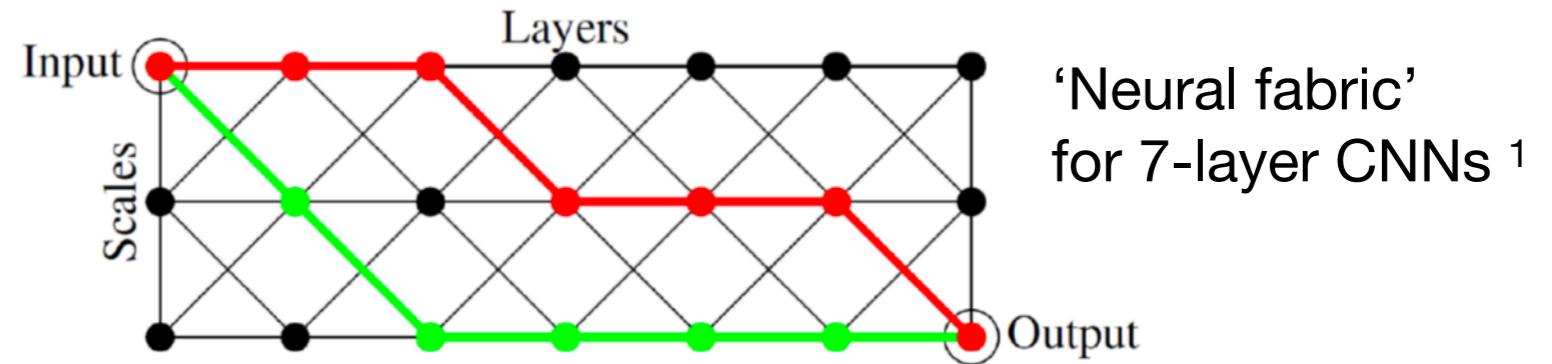
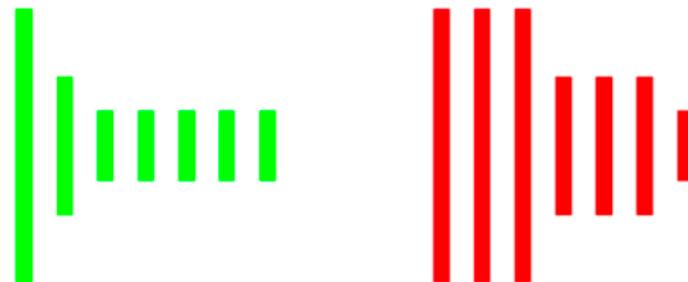


MNIST digit

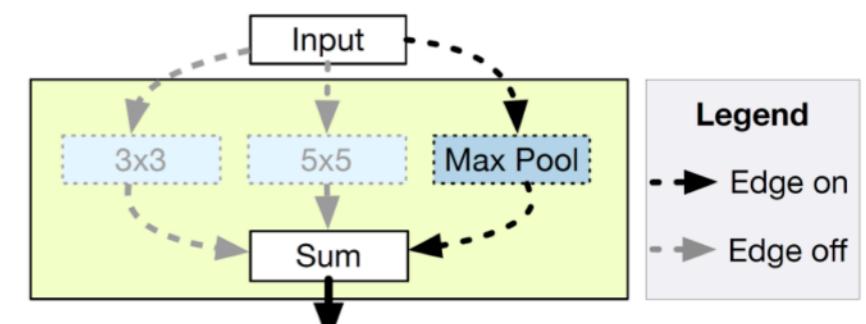


Hypernetworks

- Every candidate network belongs to a *hypernetwork*



- 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

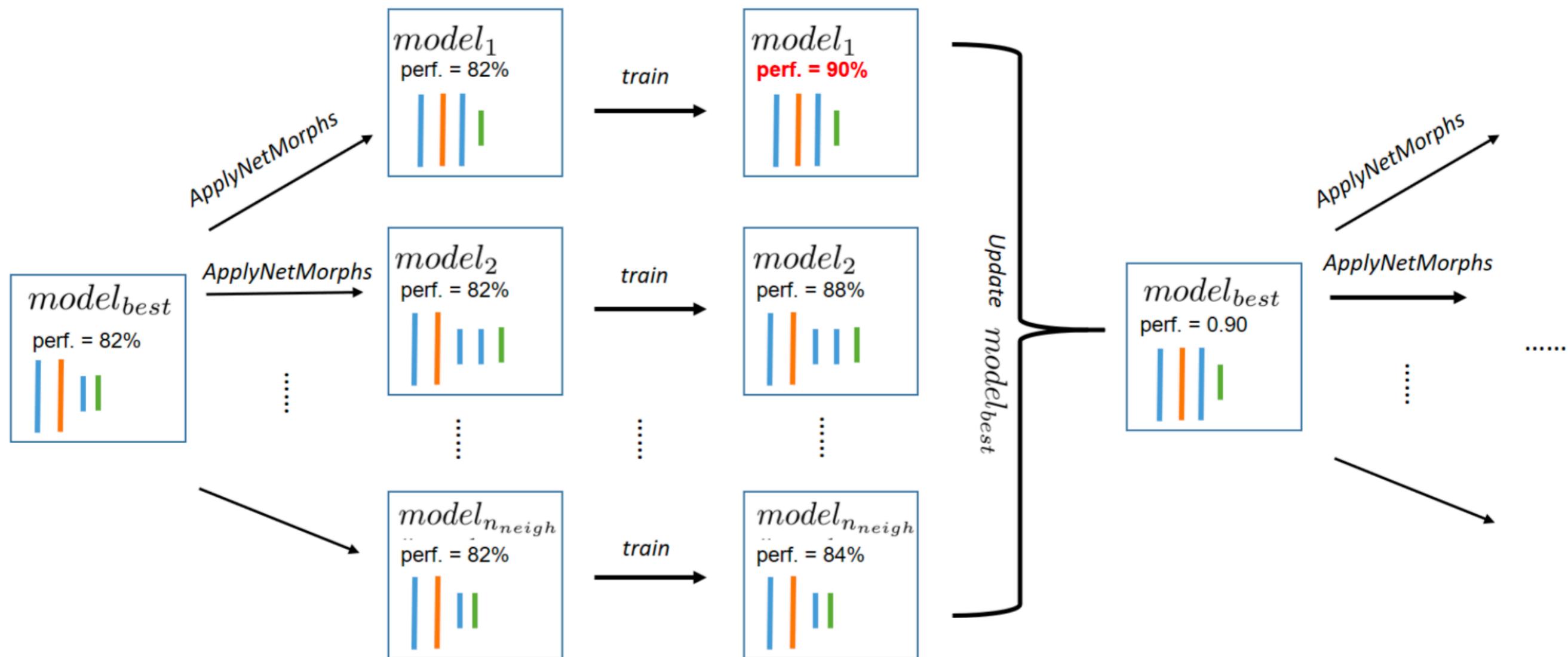


Network Morphisms

[AutoKeras]

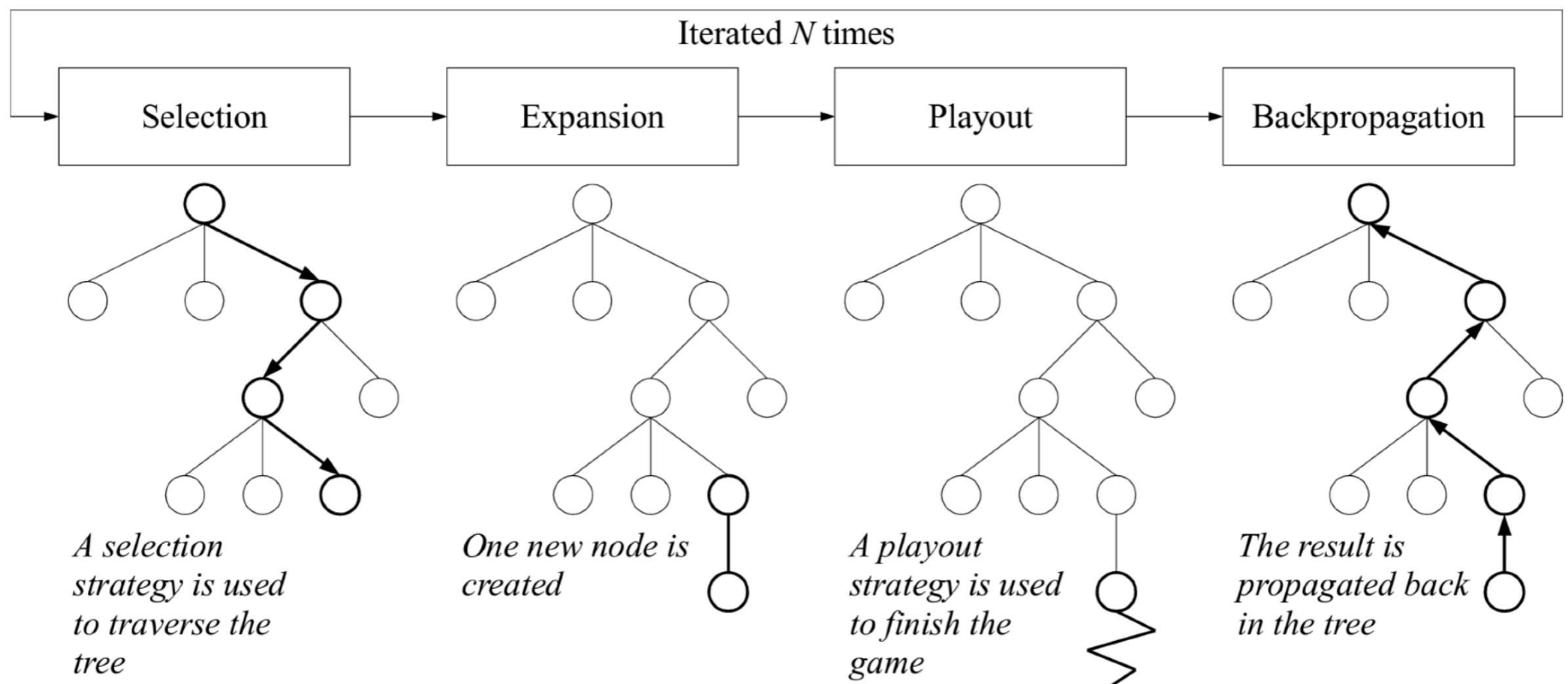


- Change architecture, but not modelled function
- Keep the initial weights, only update the derived architectures



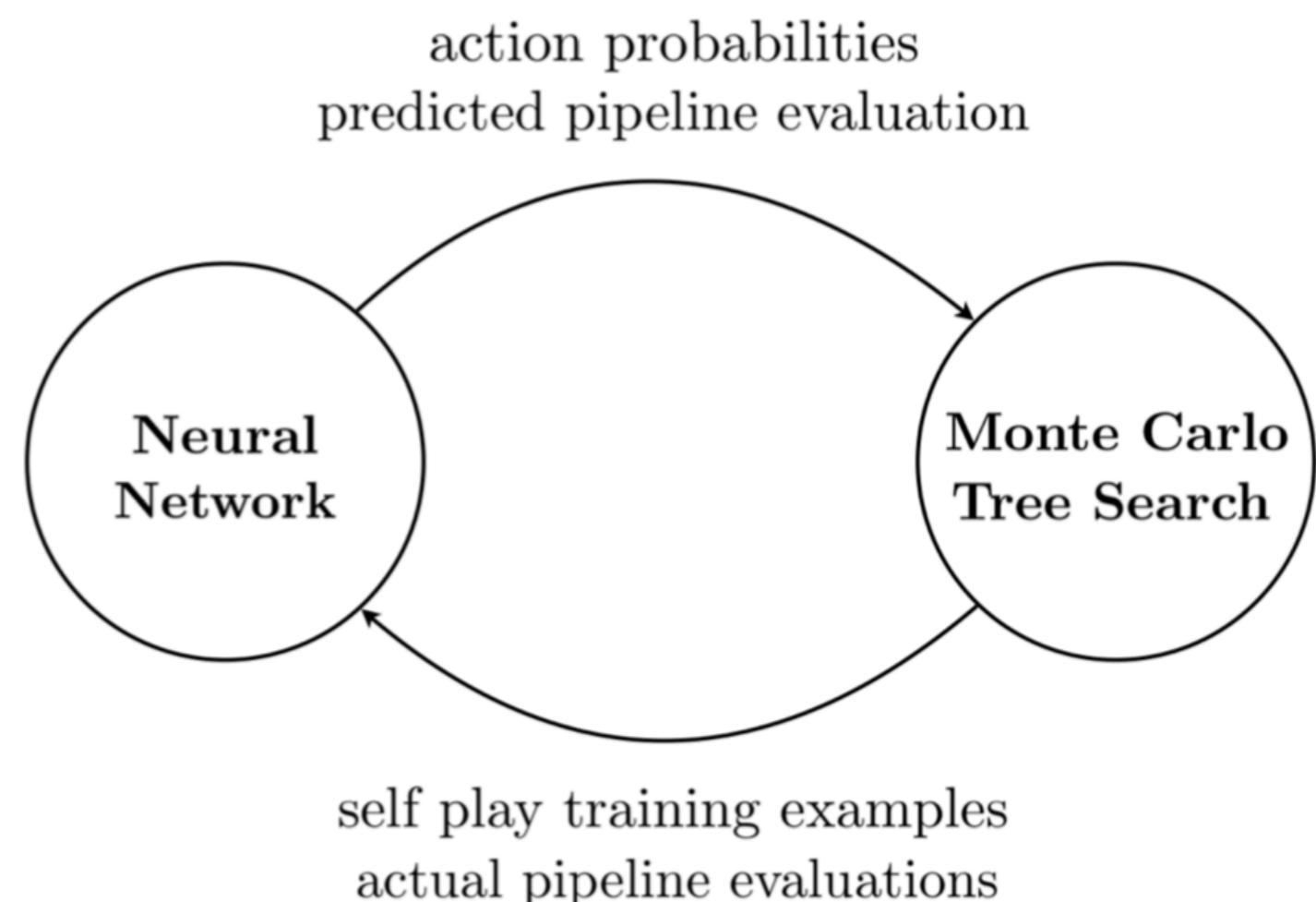
Monte Carlo Tree Search (MCTS)

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



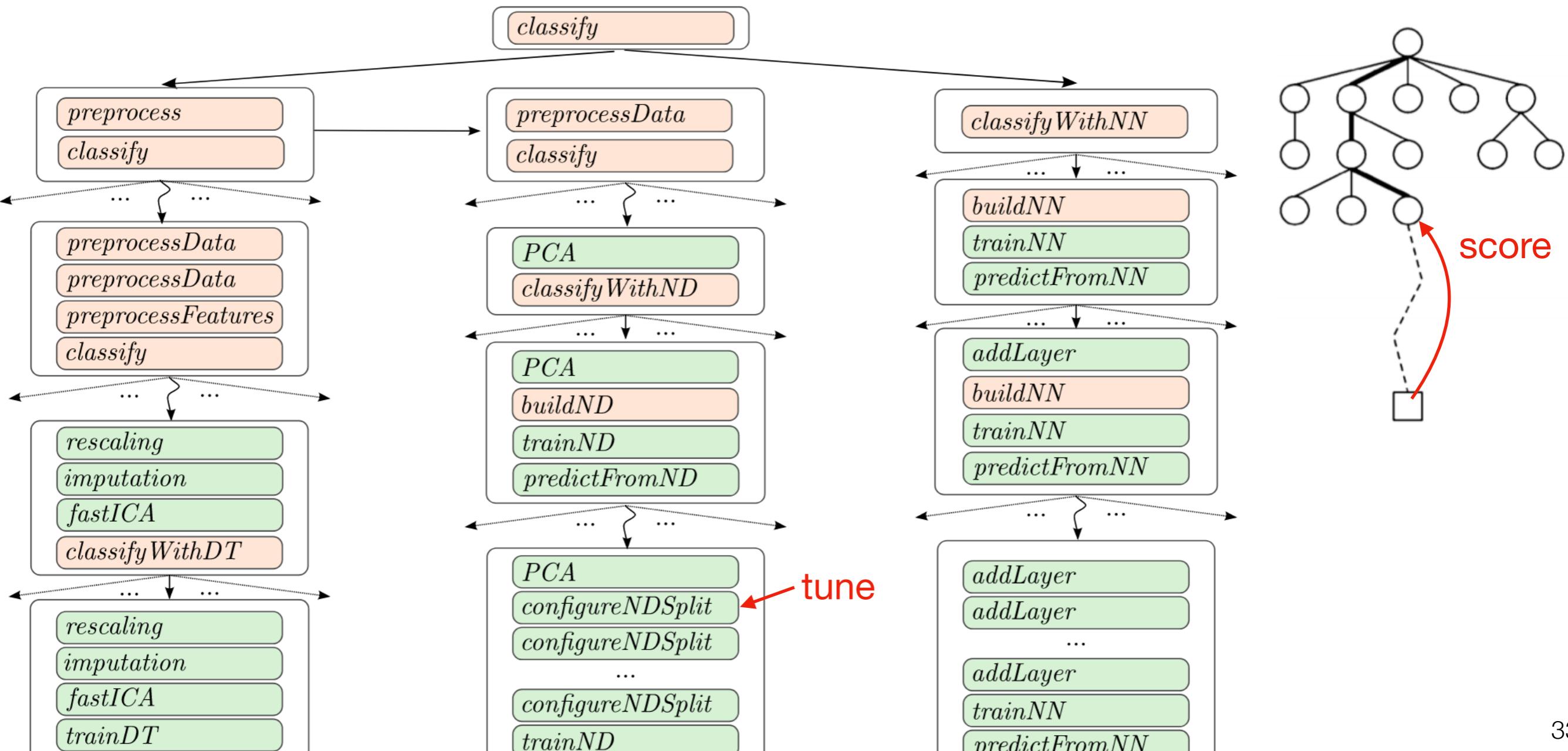
MCTS + reinforcement learning

- 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



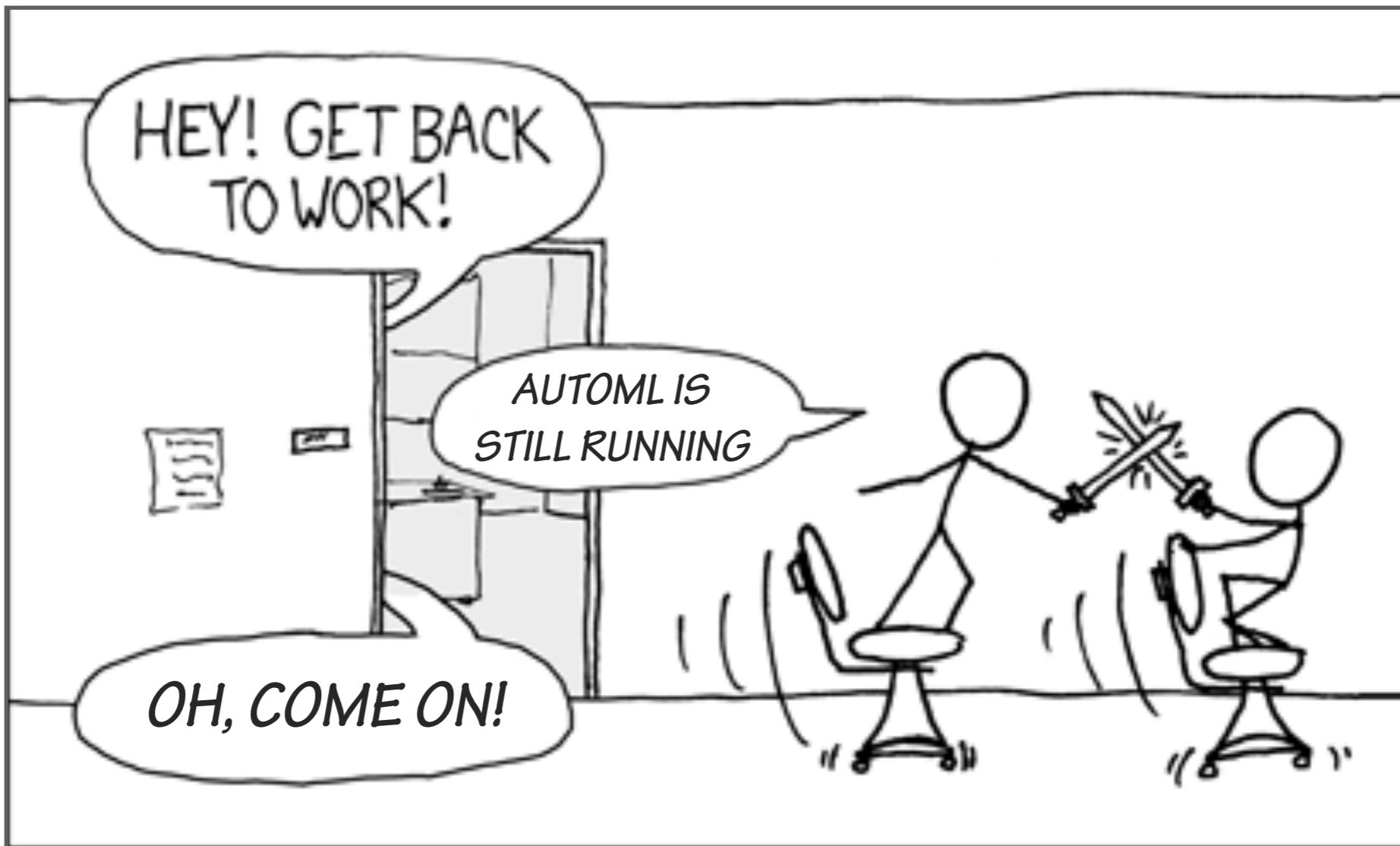
Hierarchical planning

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



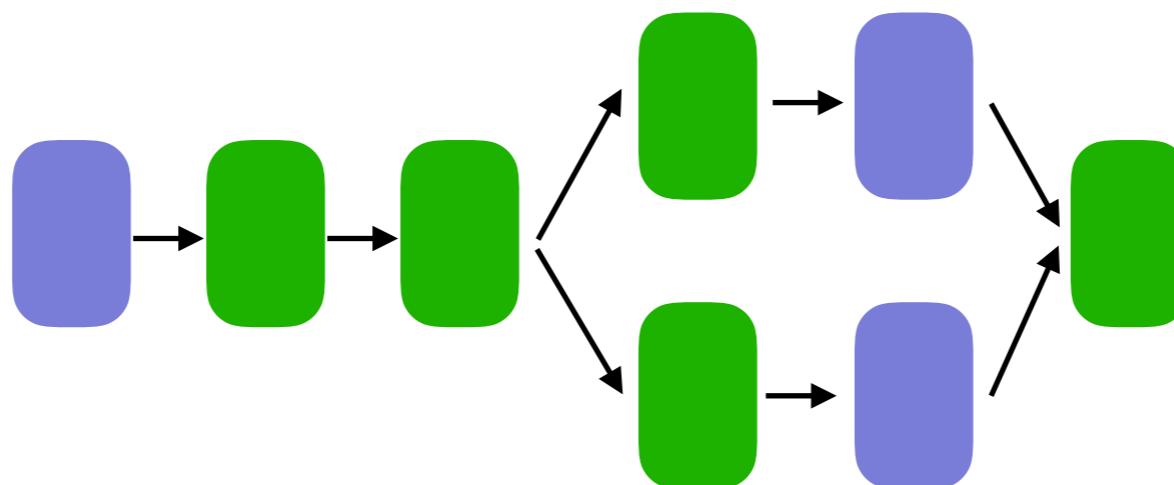
Hyperparameter Optimization

Searching the hyperparameter space **efficiently**



AutoML: subproblems

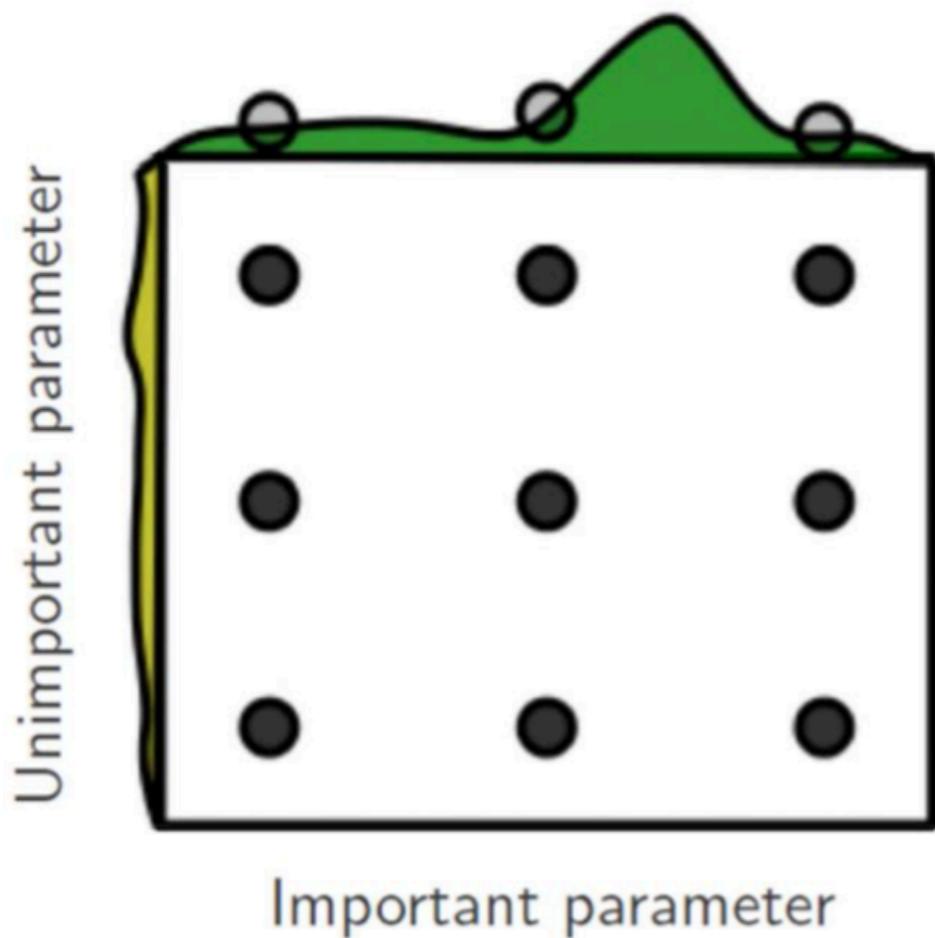
- **Hyperparameter optimization:** *many different approaches*
 - Random search
 - Bayesian optimization
 - Population-based methods (evolution)
 - Multi-fidelity optimization
 - Differentiable optimization
 - ...



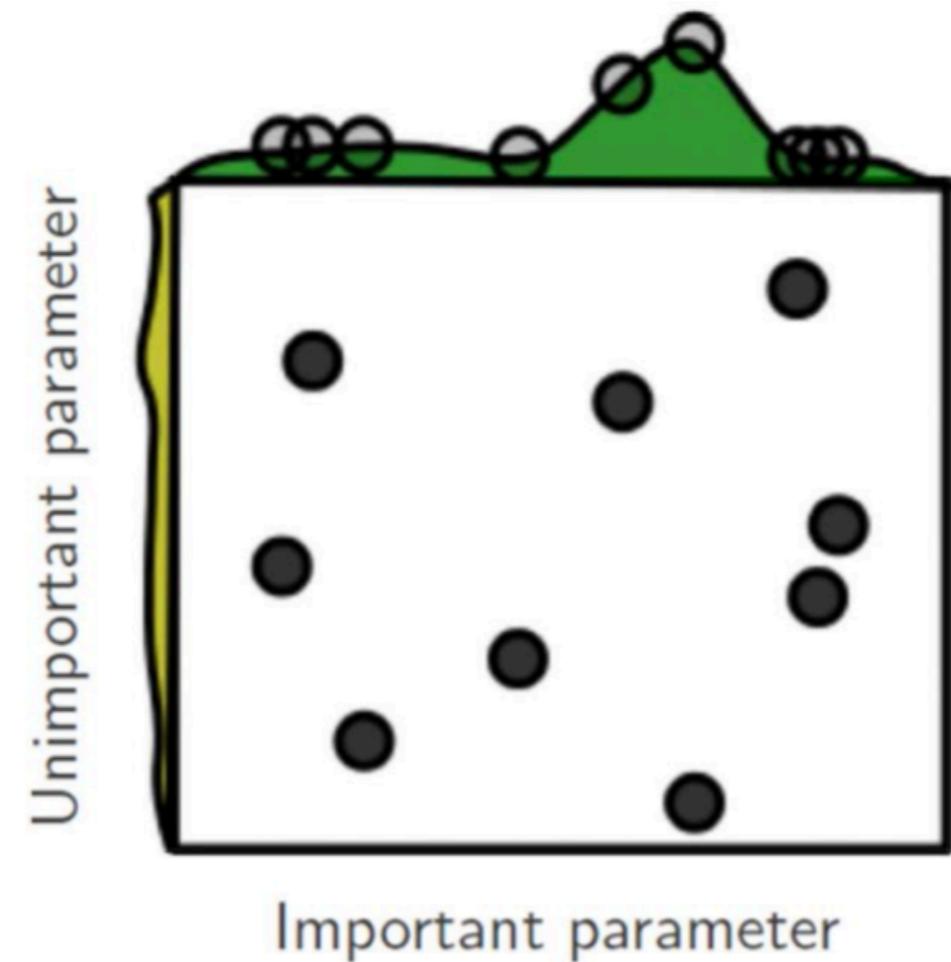
Random search

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

Grid Layout

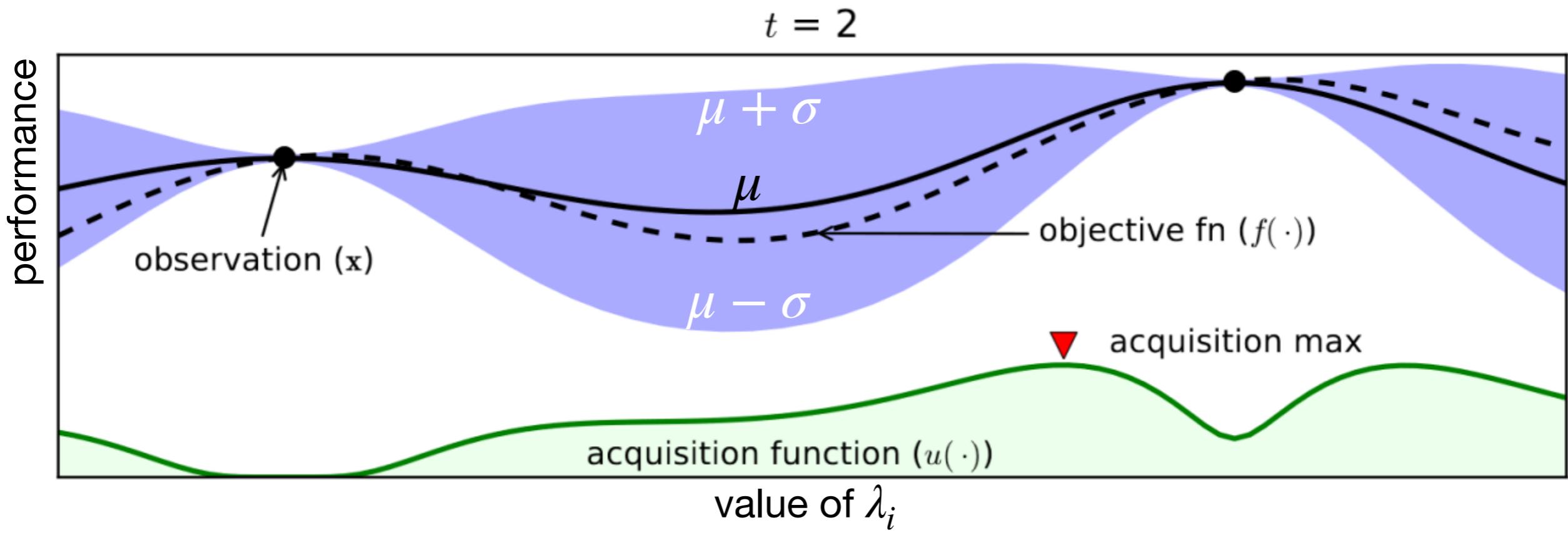


Random Layout



Bayesian Optimization

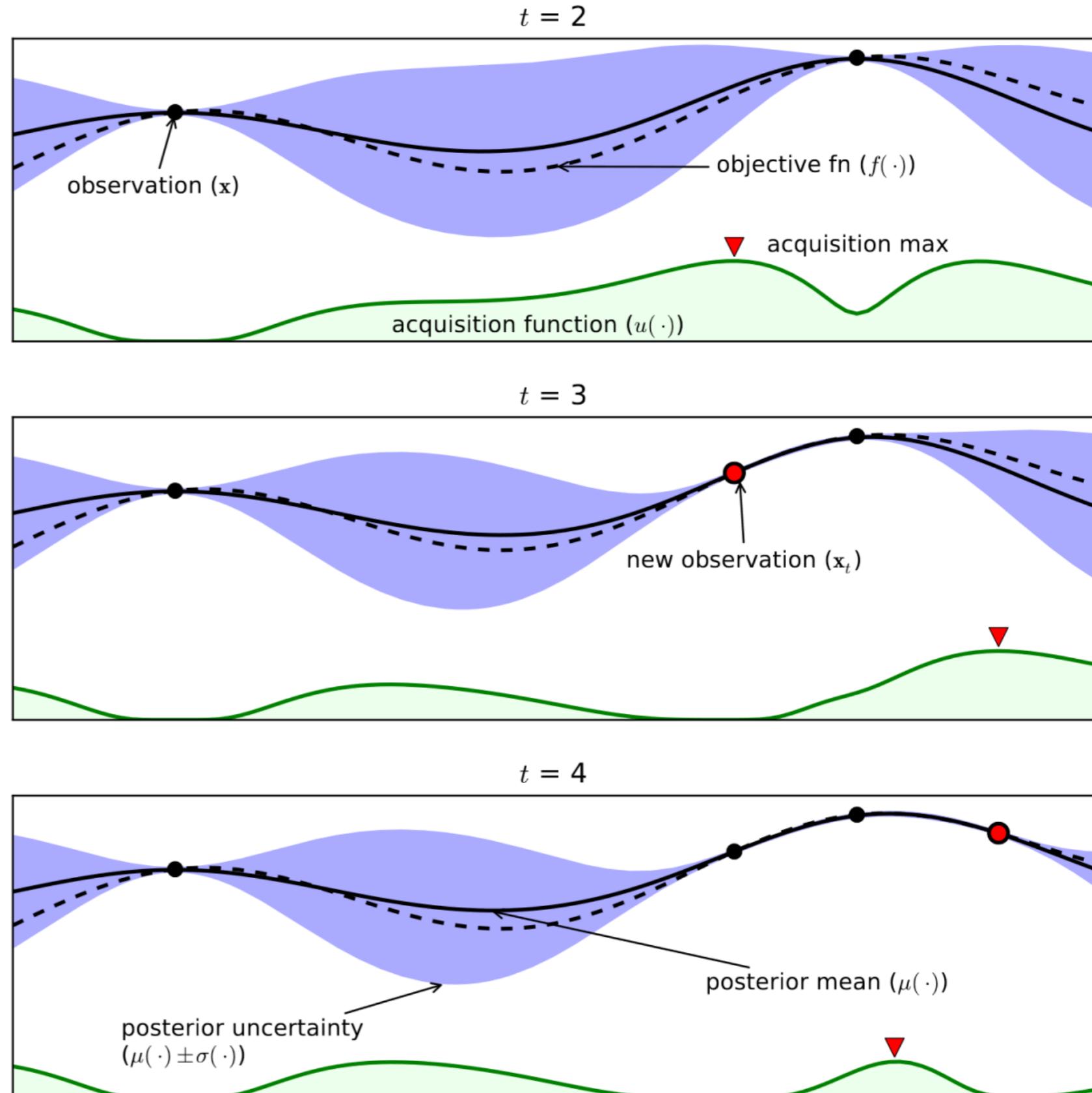
- Start with a few (random) hyperparameter configurations
- Fit a *surrogate model* to predict other configurations
- Probabilistic regression: mean μ and standard deviation σ (blue band)
- Use an *acquisition function* to trade off exploration and exploitation, e.g. Expected Improvement (EI)
- Sample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

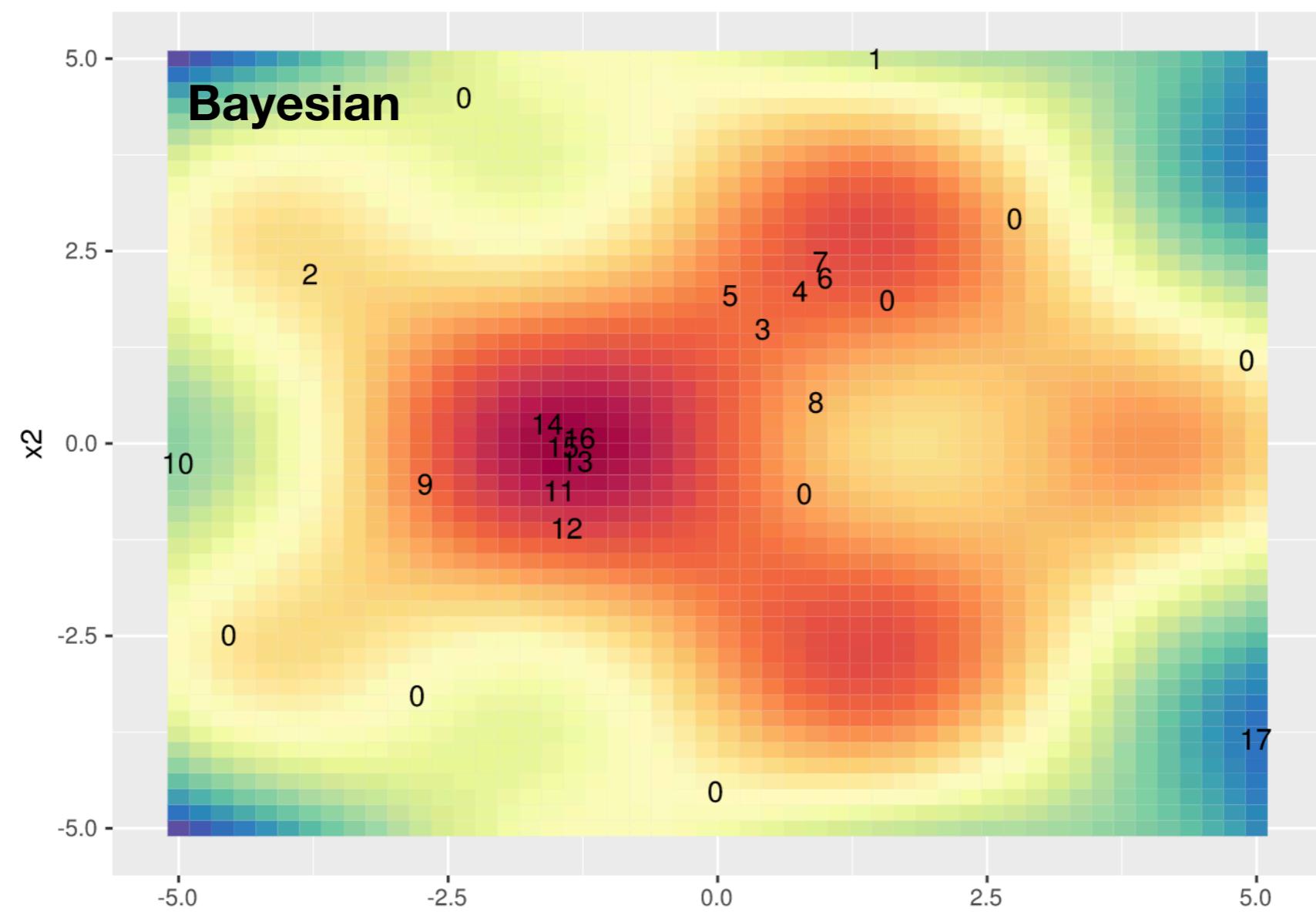
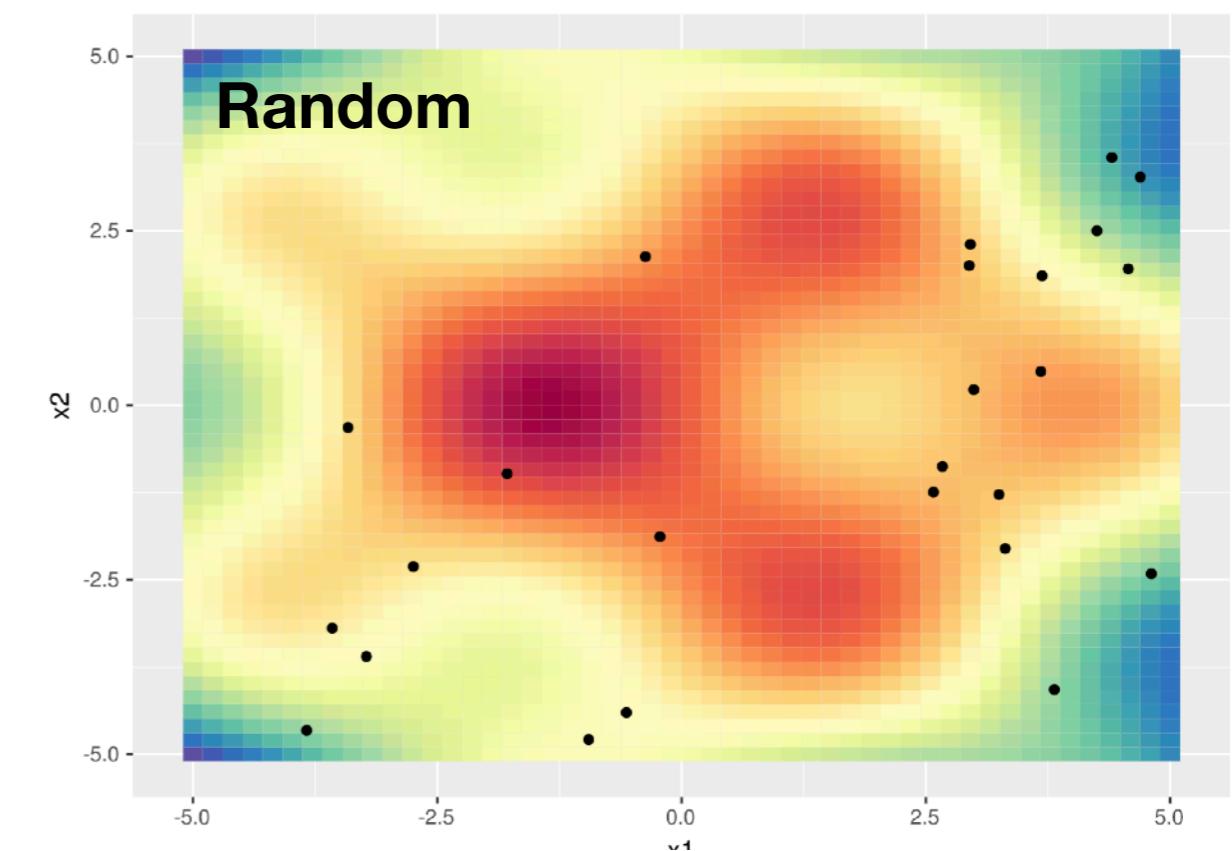
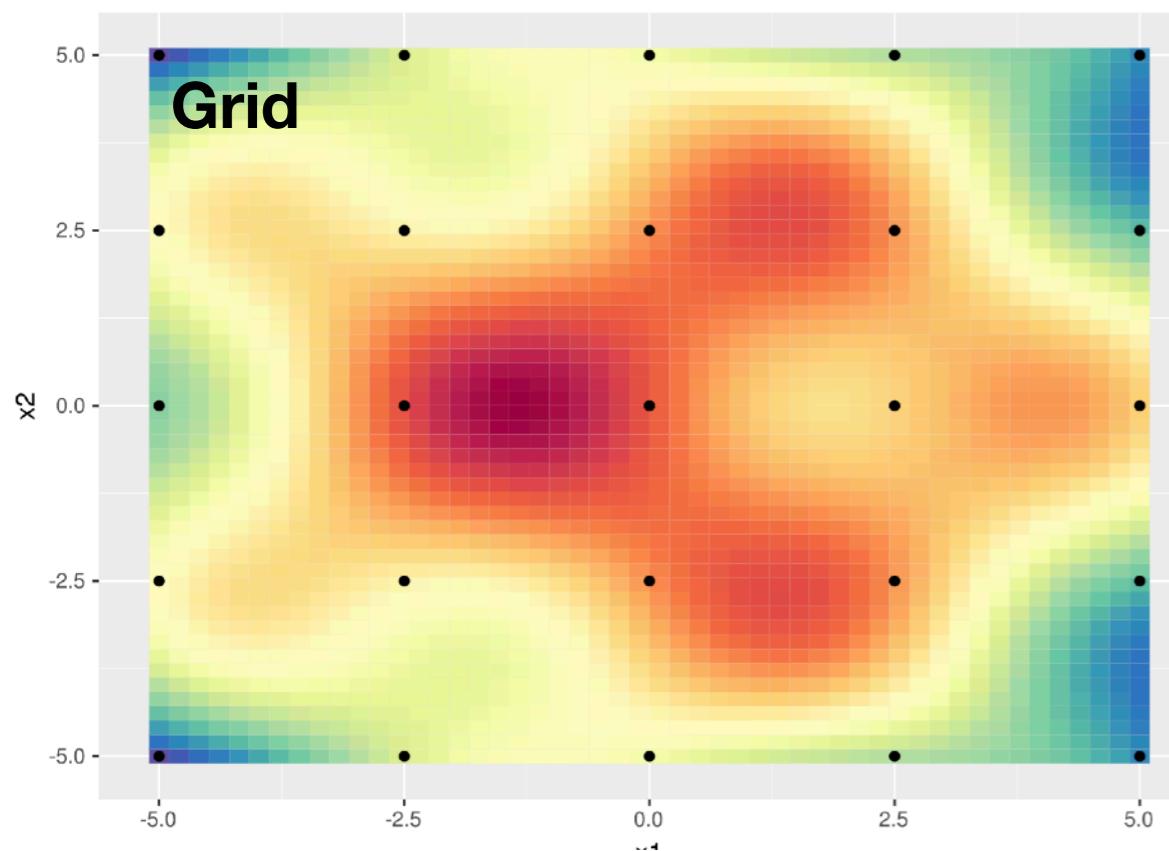


Hyperparameter optimization

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



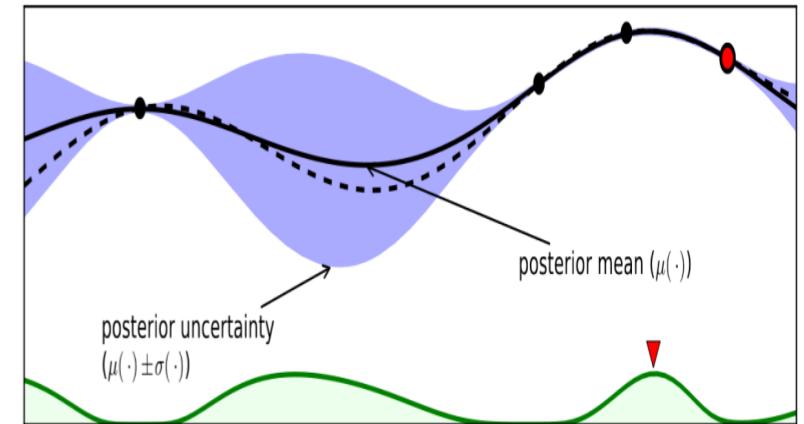


Hyperparameter optimization

Bayesian Optimization: which surrogate?

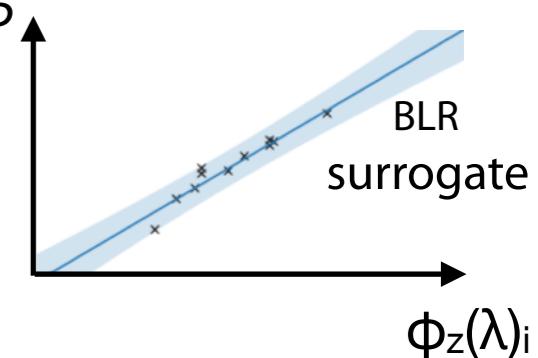
Gaussian processes [\[skopt\]](#)

- + 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



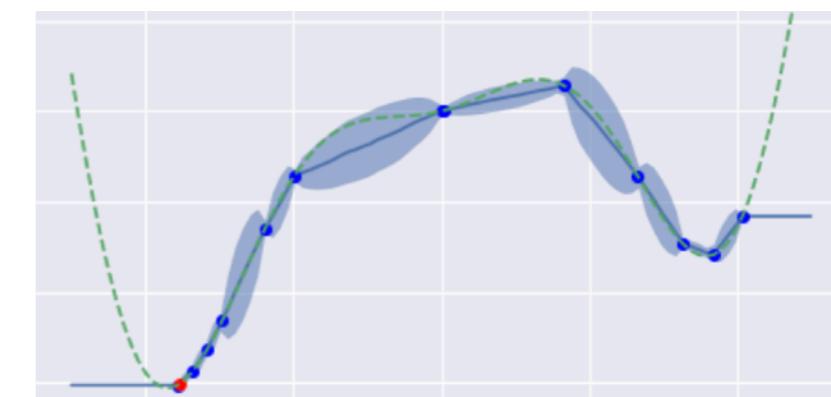
Bayesian neural networks [\[Springenberg et al. 2016\]](#)

- + Bayesian - Scalability?

[\[SMAC\]](#)

Random Forests [\[Hutter et al. 2011, Feurer et al. 2015\]](#)

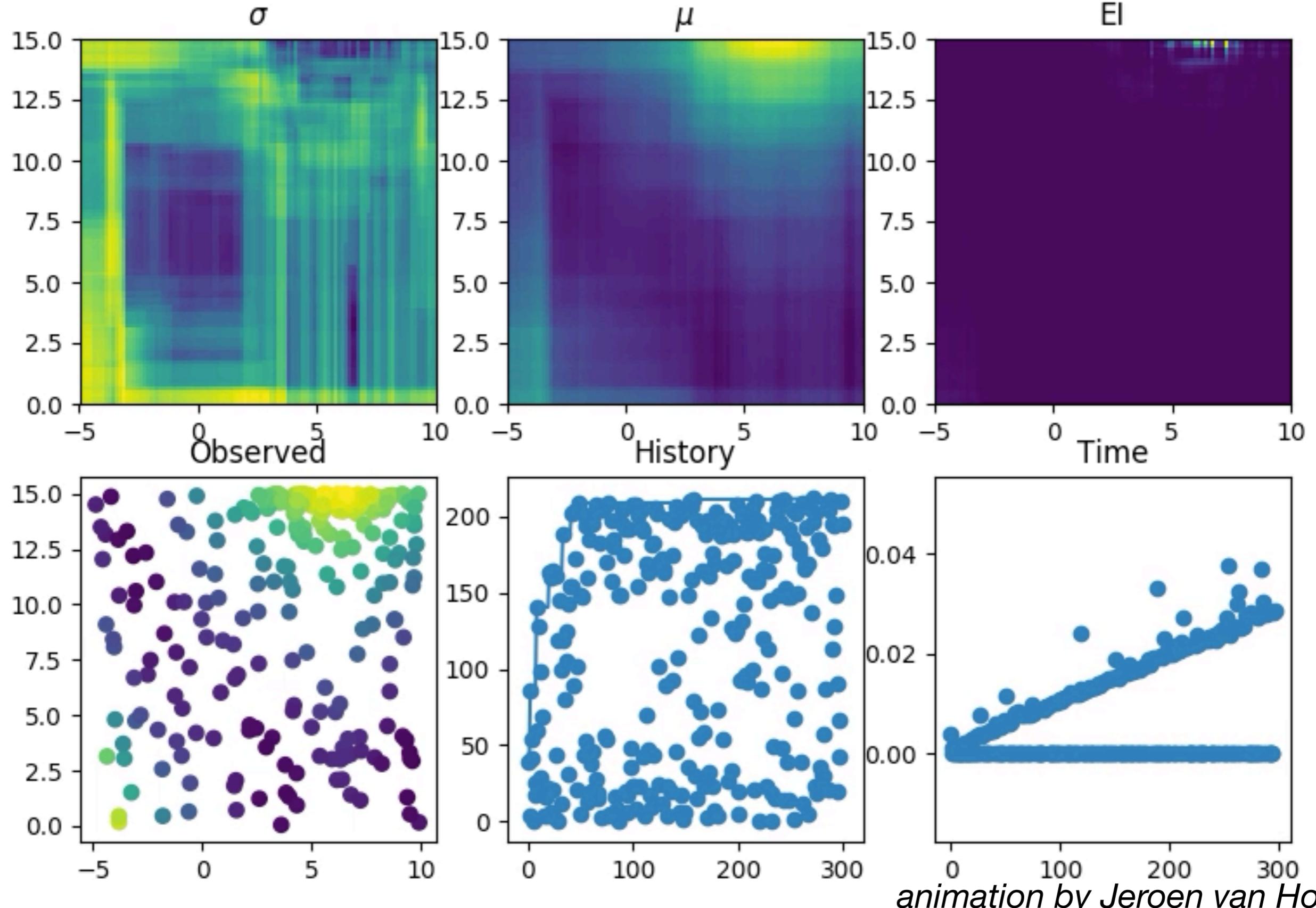
- + scalable, handles conditional hyperparams
- bad uncertainty estimates, extrapolation



Boosting + quantile regression [\[van Hoof, Vanschoren 2019\]](#) [\[HyperBoost\]](#)

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

Random Forest Surrogate



Tree of Parzen Estimators

1. Test some hyperparameters

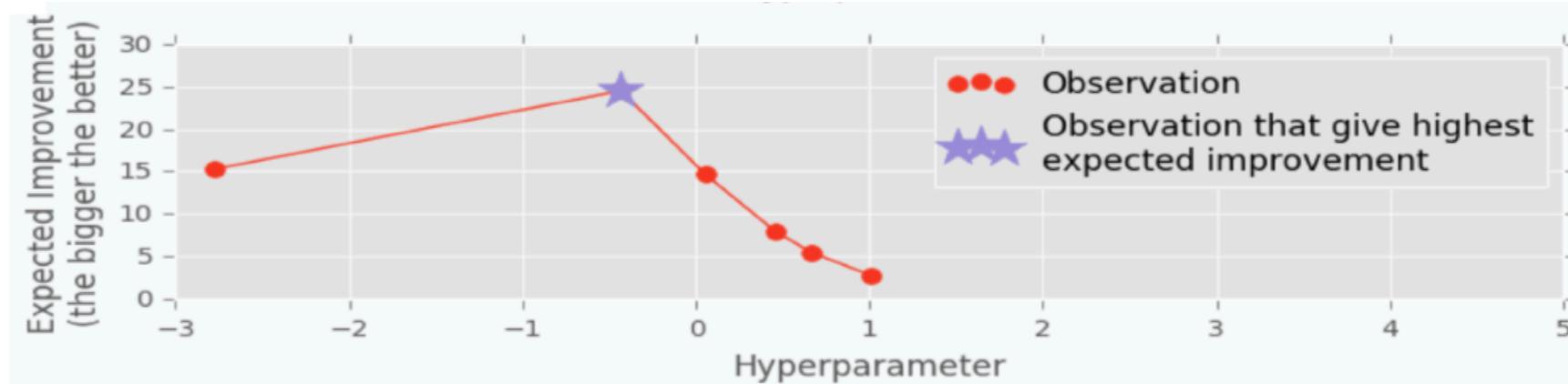
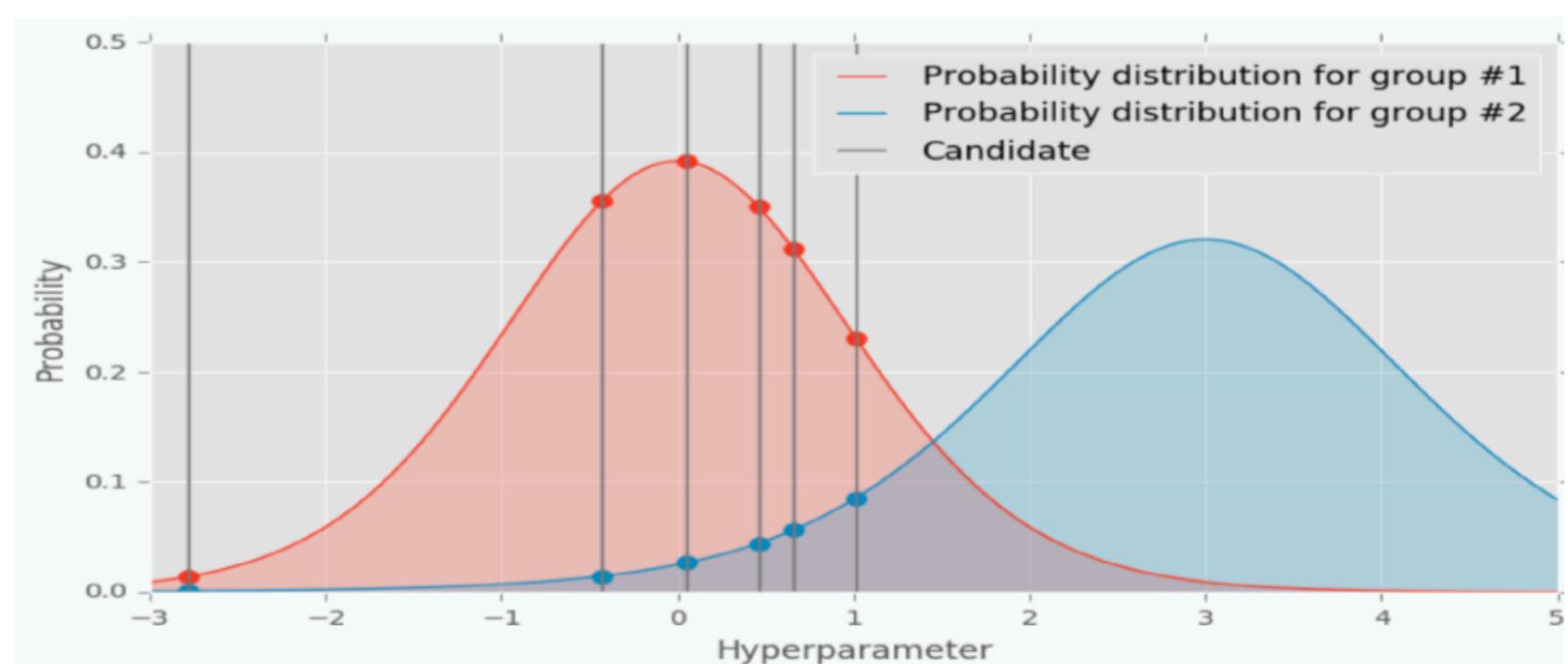
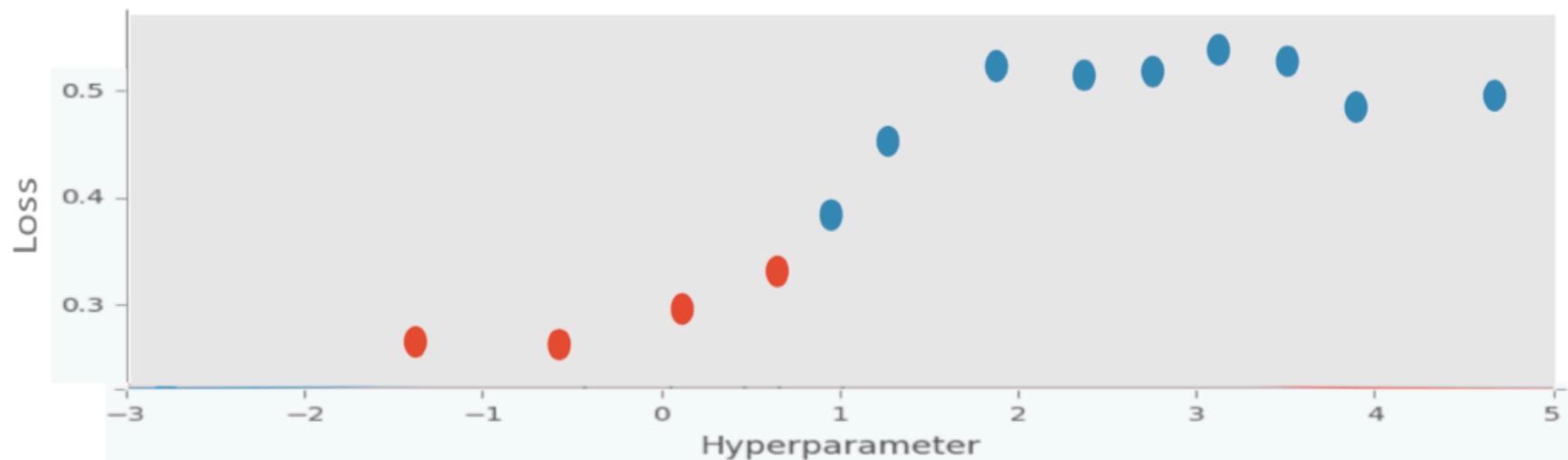
2. Separate into **good** and **bad** hyperparameters (with some quantile)

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

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

Shown to be equivalent to EI!

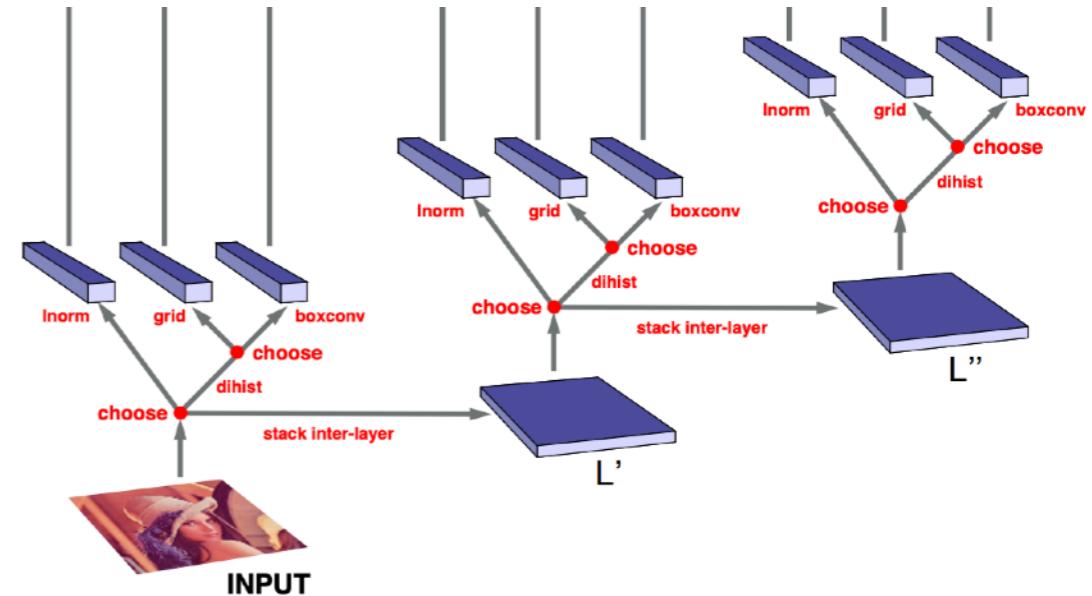
Efficient, **parallelizable**, robust, but less sample efficient than GPs



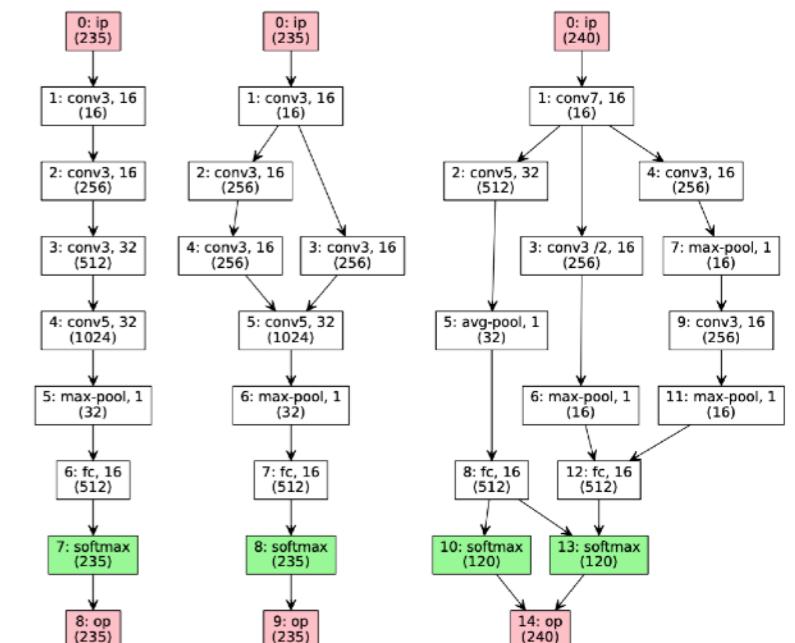
Bayesian Optimization

Examples:

- Image classification pipelines¹
 - 238 hyperparameters, tuned with TPE

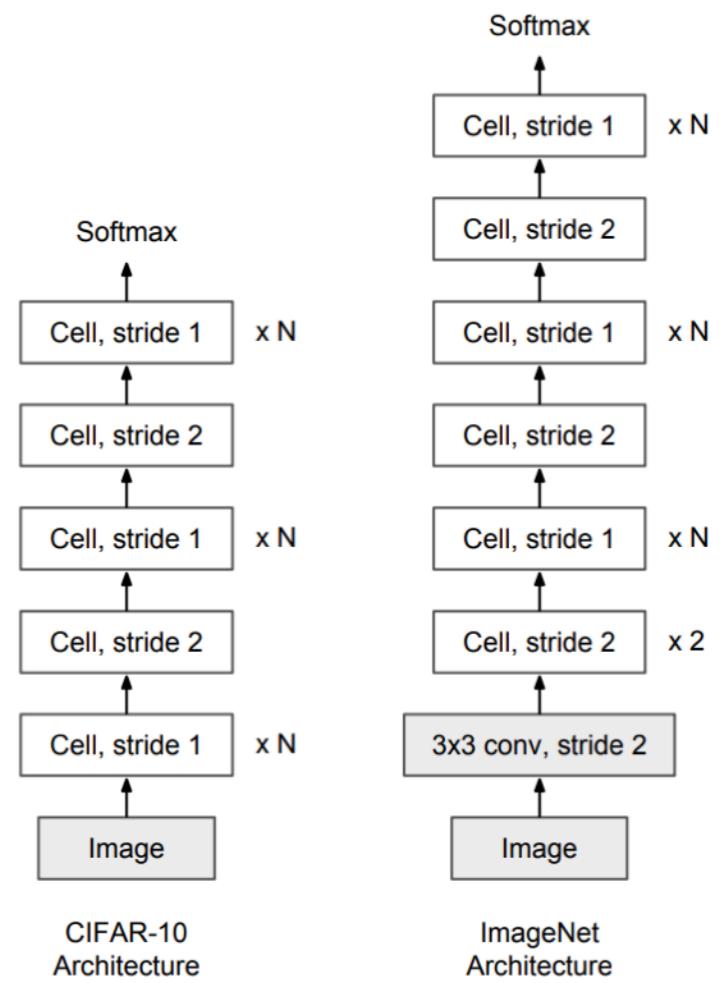
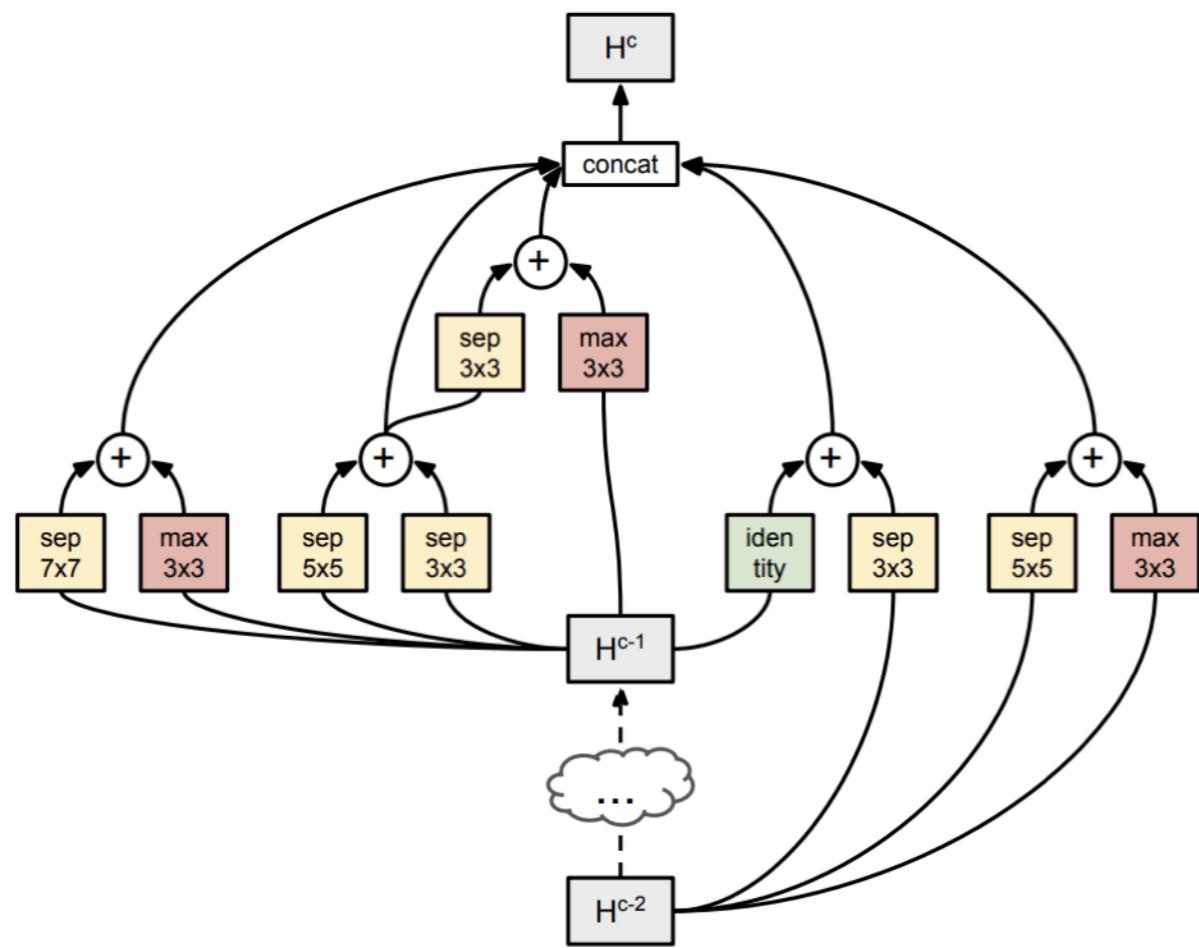


- Kernels to optimize neural nets with GP-based Bayesian optimization
 - ArcNet²: DNNs, 23 hyperparameters, 6 kernels
 - NASBOT³: DNNs and CNNs



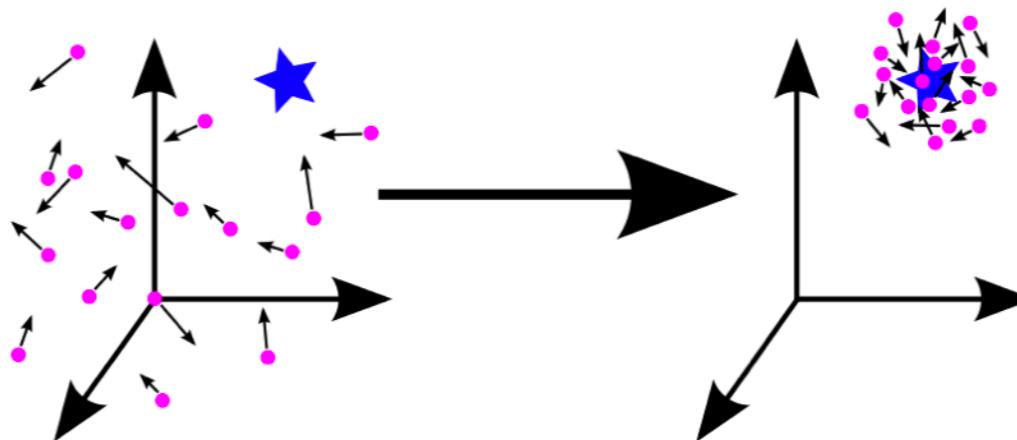
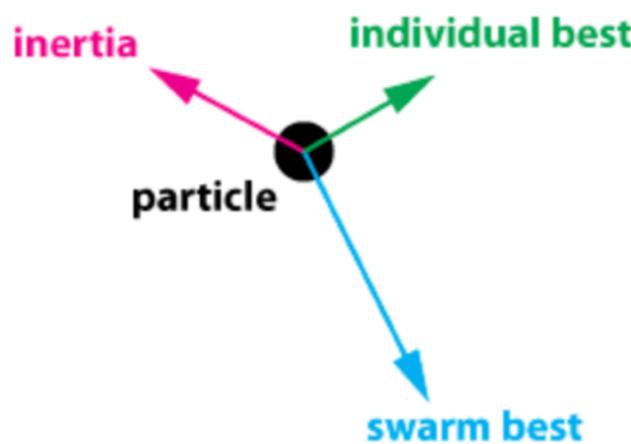
Bayesian Optimization

- AutoNet¹ : DNNs, 63 hyperparameters, tuned with SMAC
- Joint NAS + HPO²: ResNets, tuned with BO-HB
- PNAS (Progressive NAS)³
 - Cell search space, optimized with SMAC, HPO afterwards
 - SotA on ImageNet, CIFAR



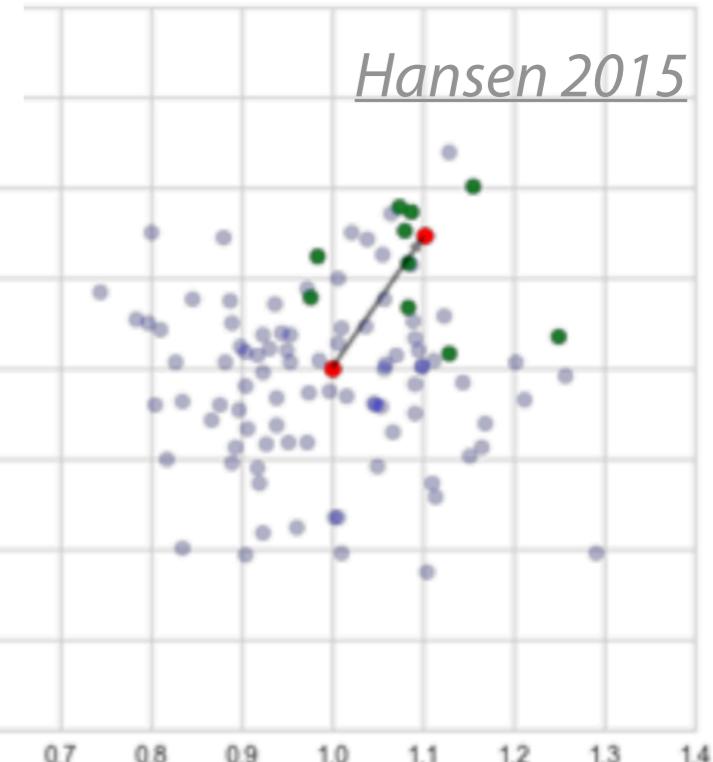
Population-based methods

- 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

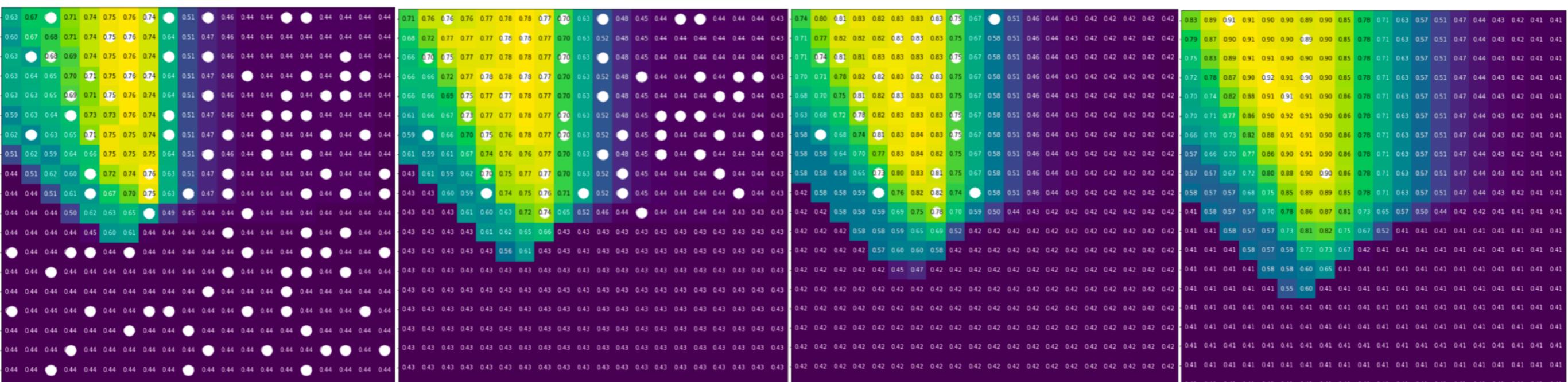
[Loshilov, Hutter 2016]



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



1/16

1/8

1/4

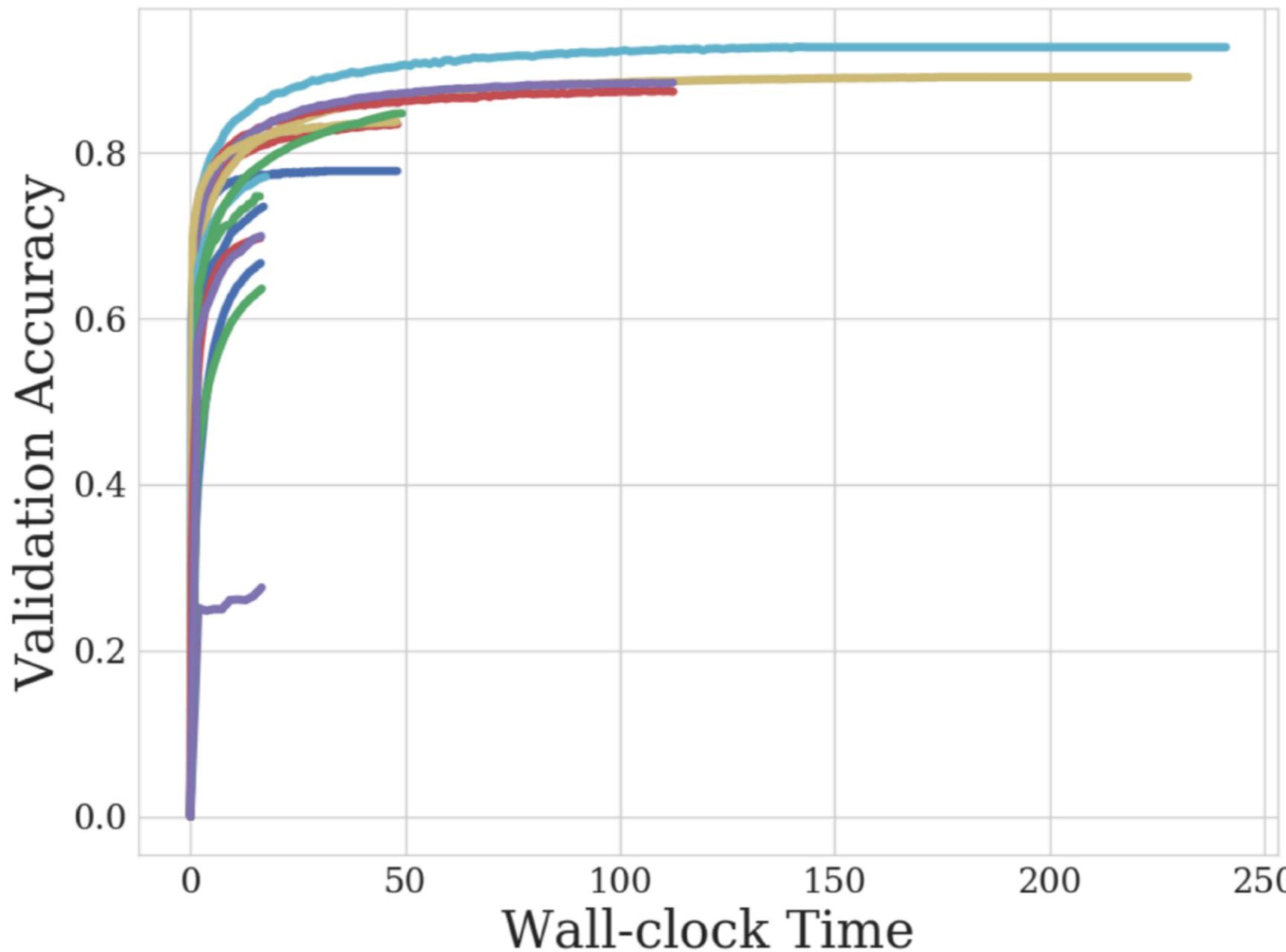
1/2

sample size

Multi-fidelity optimization

Successive halving:

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

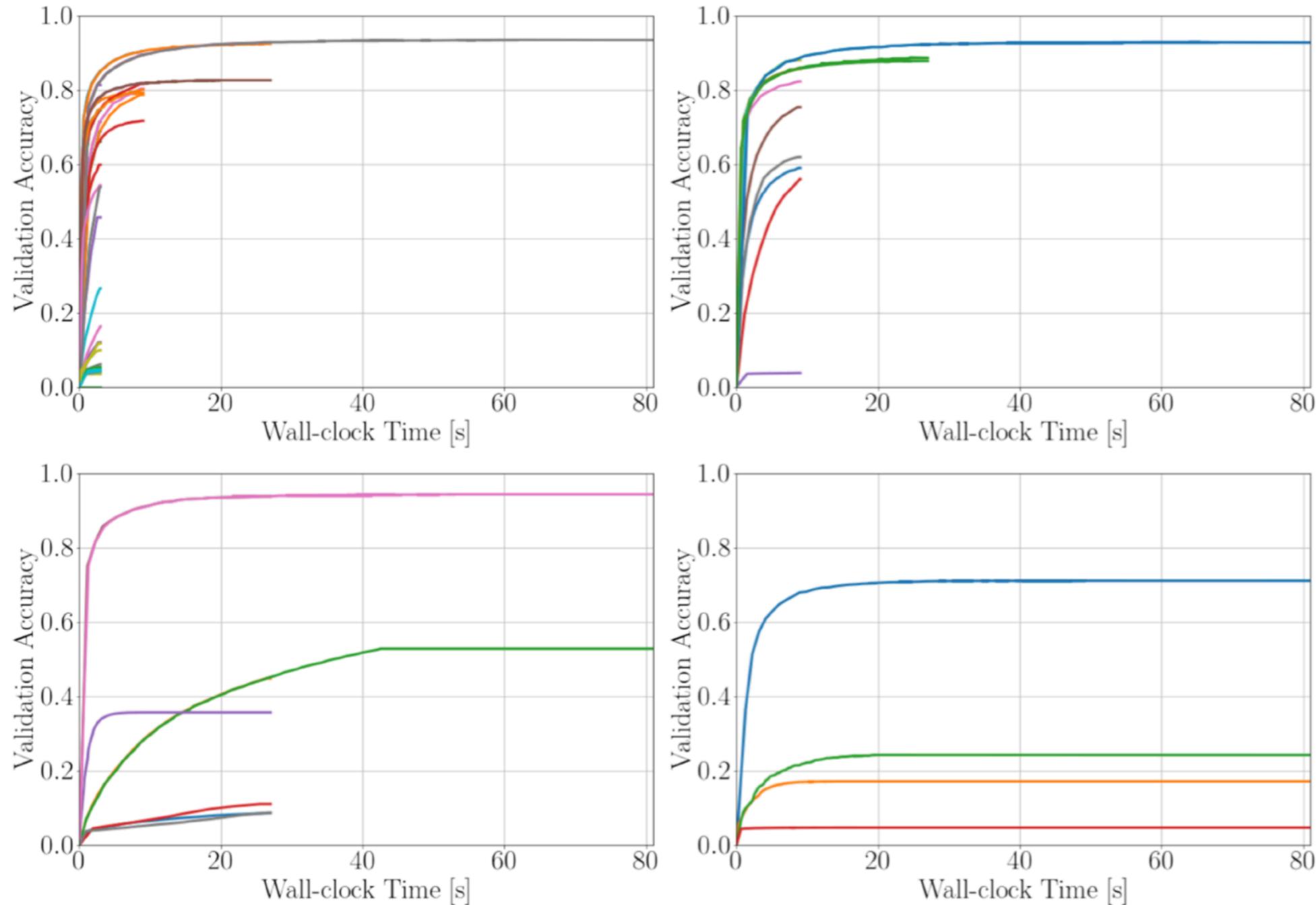


Hyperparameter optimization

Multi-fidelity optimization

Hyperband (HB): Repeated, decreasingly aggressive successive halving

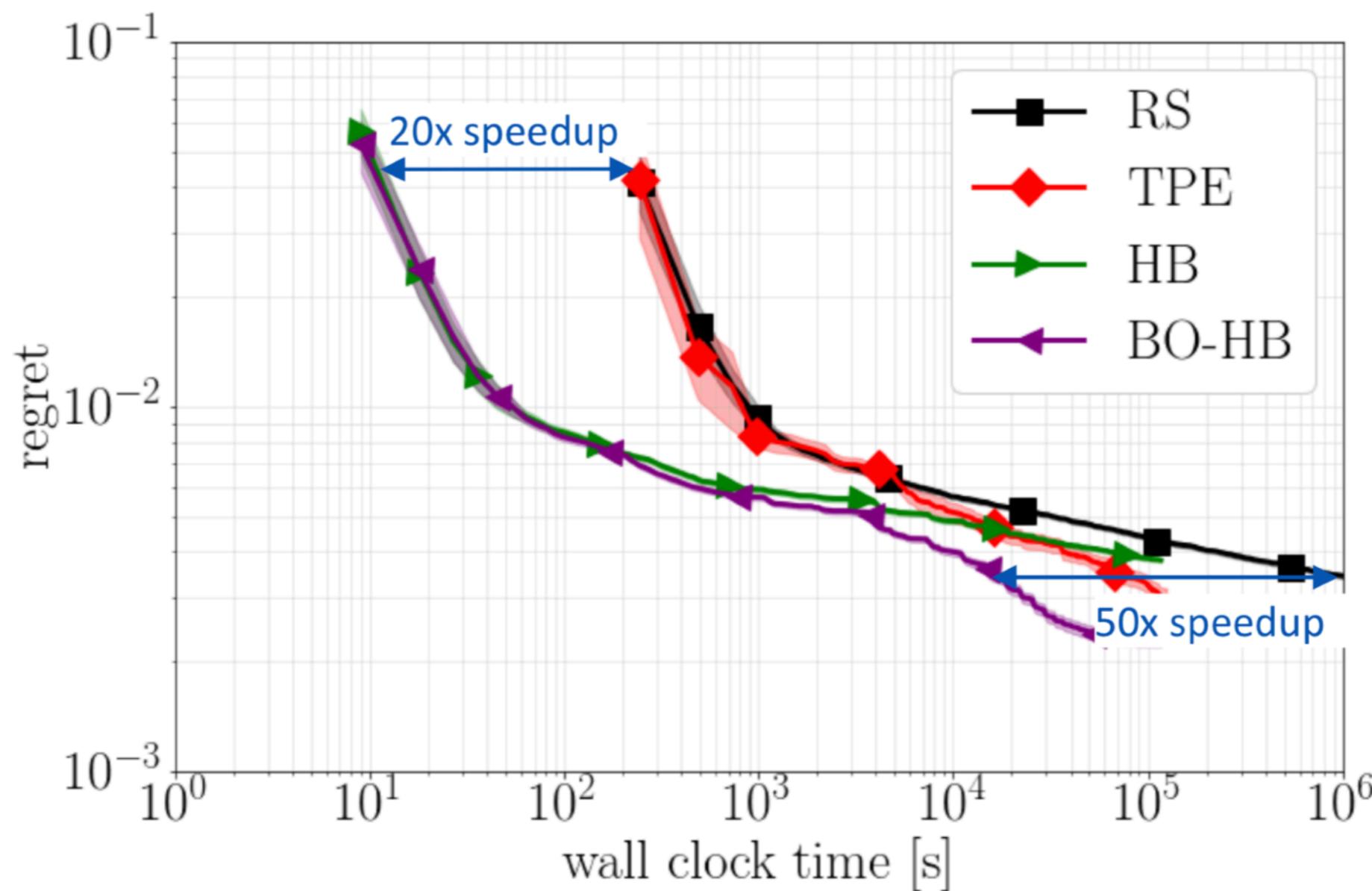
- Minimizes (doesn't eliminate) chance that candidate was pruned too early
- Strong anytime performance, easy to implement, scalable, parallelizable



Multi-fidelity optimization

Combined Bayesian Optimization and Hyperband (BO-HB)

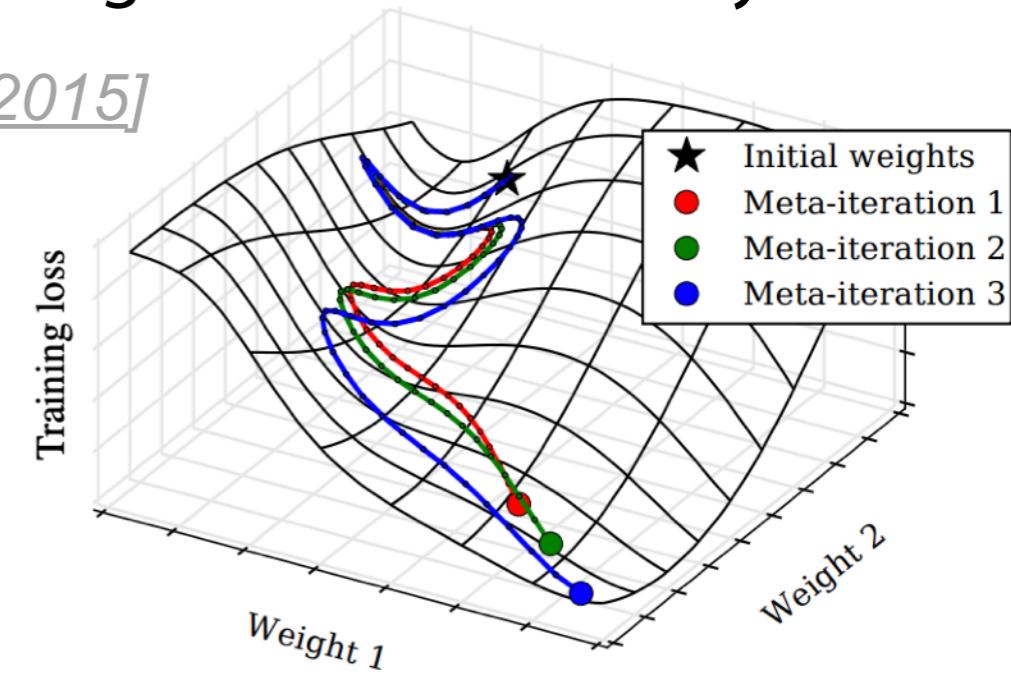
- Choose which configurations to evaluate (with TPE)
- Hyperband: allocate budgets more efficiently
- Strong anytime and final performance



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

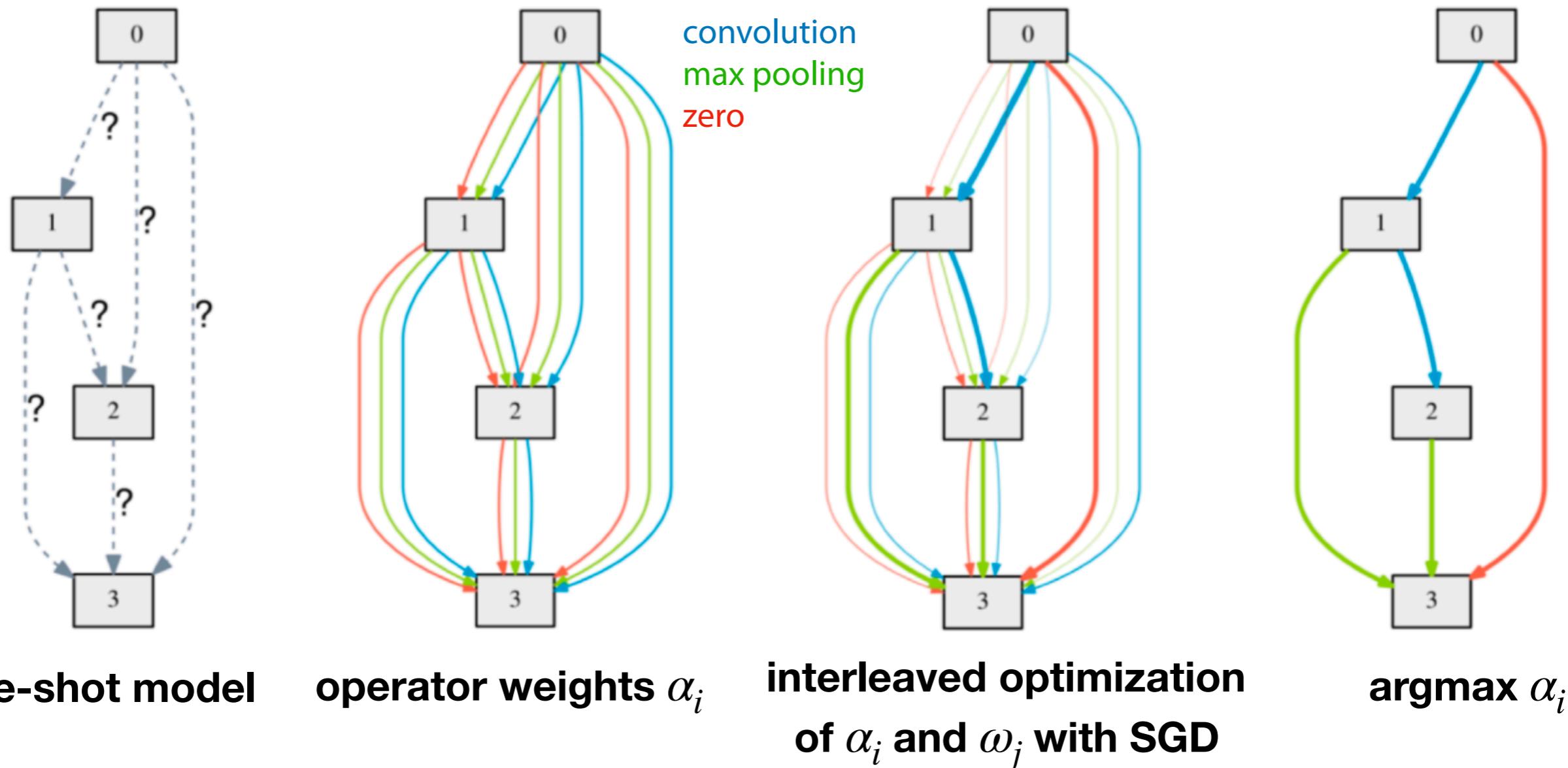


$$\begin{aligned} & \min_{\lambda} \mathcal{L}_{val}(w^*(\lambda), \lambda) \\ s.t. \quad & w^*(\lambda) = \operatorname{argmin}_w \mathcal{L}_{train}(w, \lambda) \end{aligned}$$

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

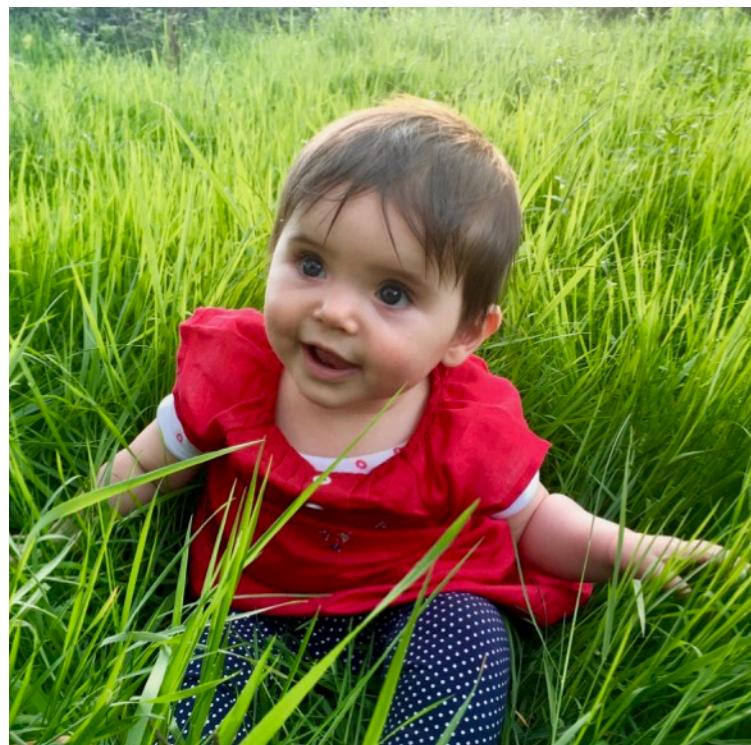
DARTS: Differentiable NAS

- 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



Learning is a never-ending process

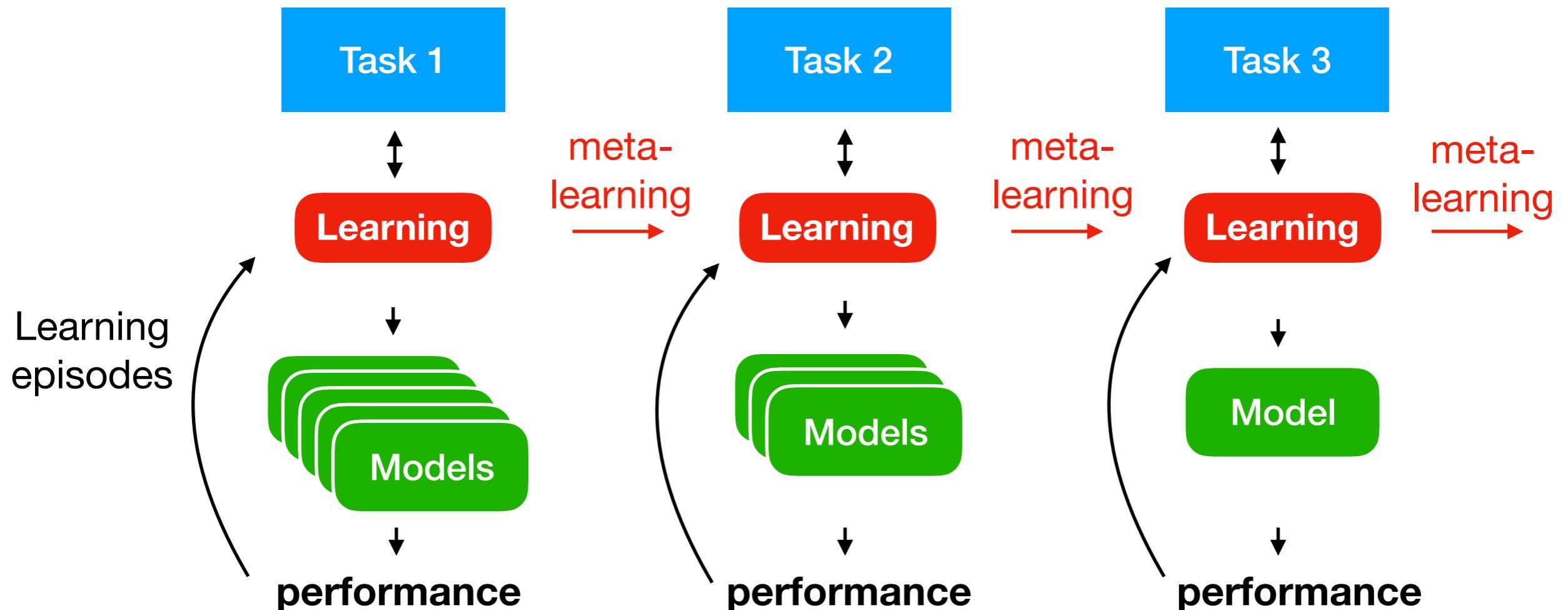
Humans don't learn from scratch



Learning is a never-ending process

Humans learn *across* tasks

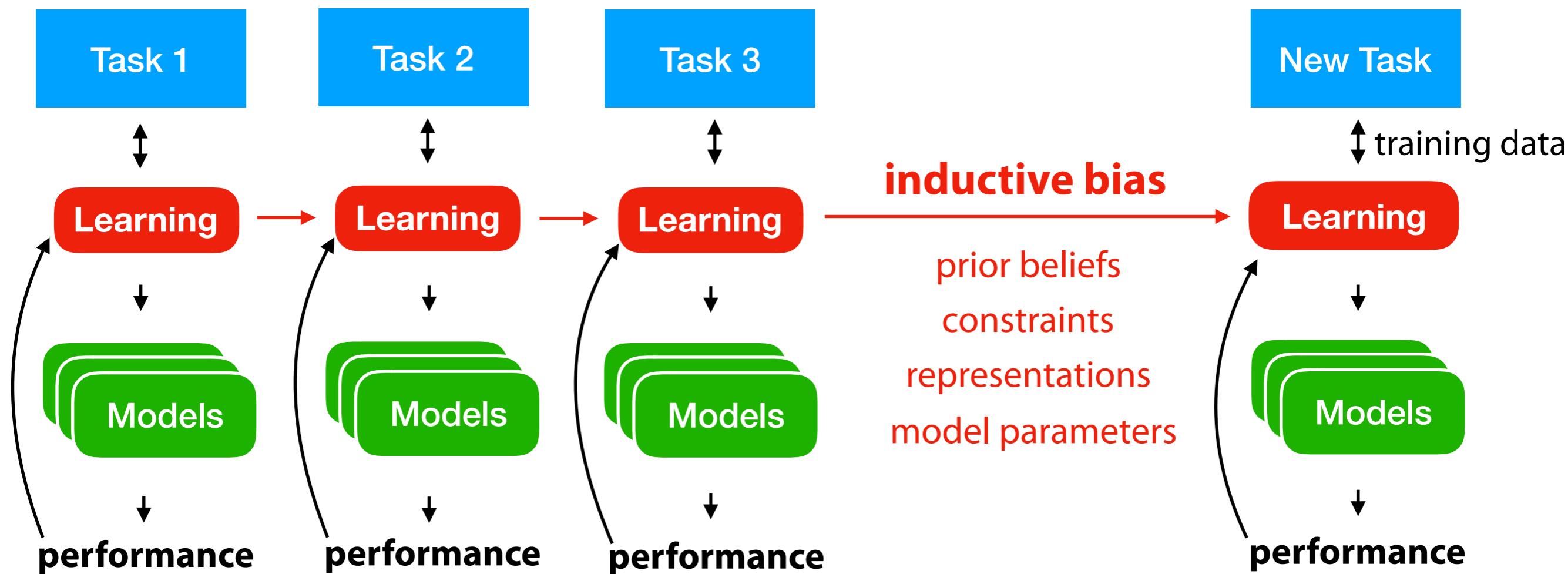
Why? Requires less trial-and-error, less data



Learning to learn

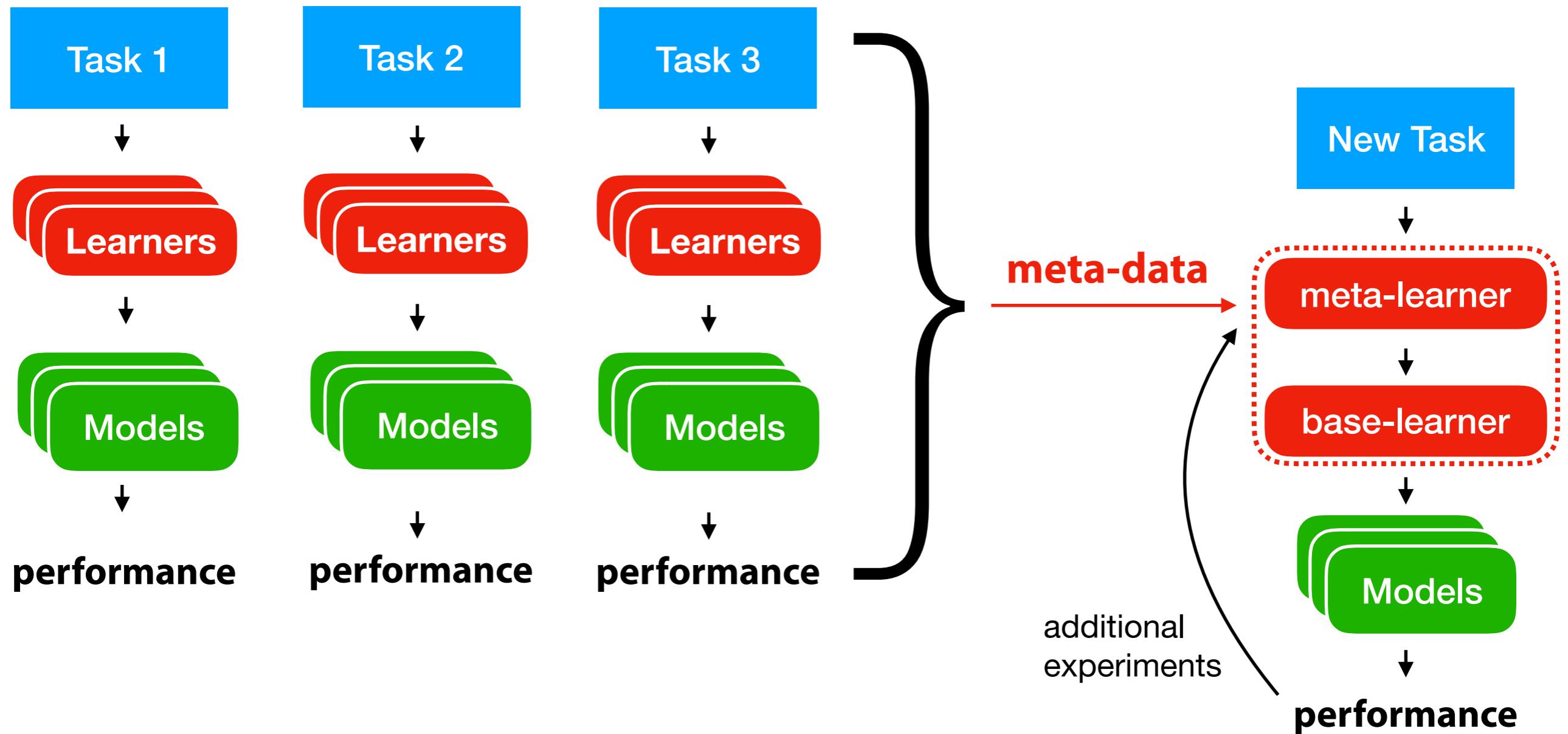
If prior tasks are *similar*, we can **transfer** prior knowledge to new tasks

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

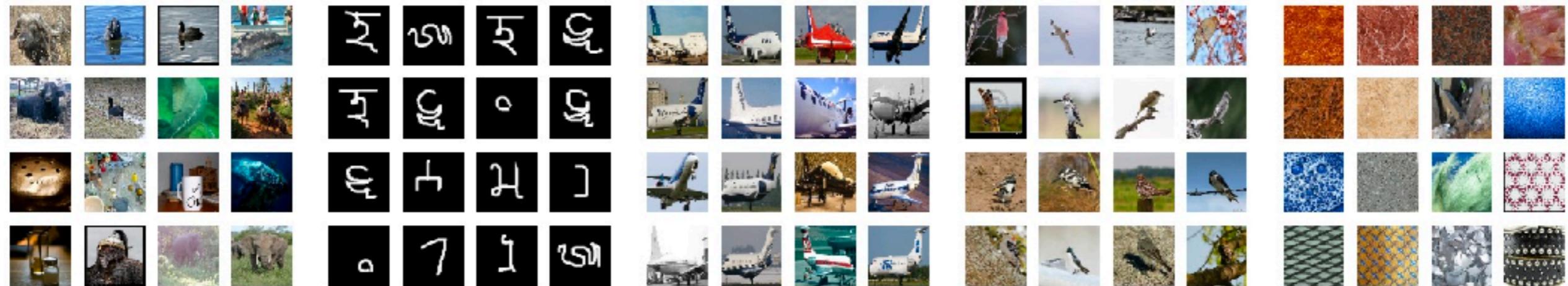


Meta-learning

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



Example: meta-dataset



(a) ImageNet

(b) Omniglot

(c) Aircraft

(d) Birds

(e) DTD



(f) Quick Draw

(g) Fungi

(h) VGG Flower

(i) Traffic Signs

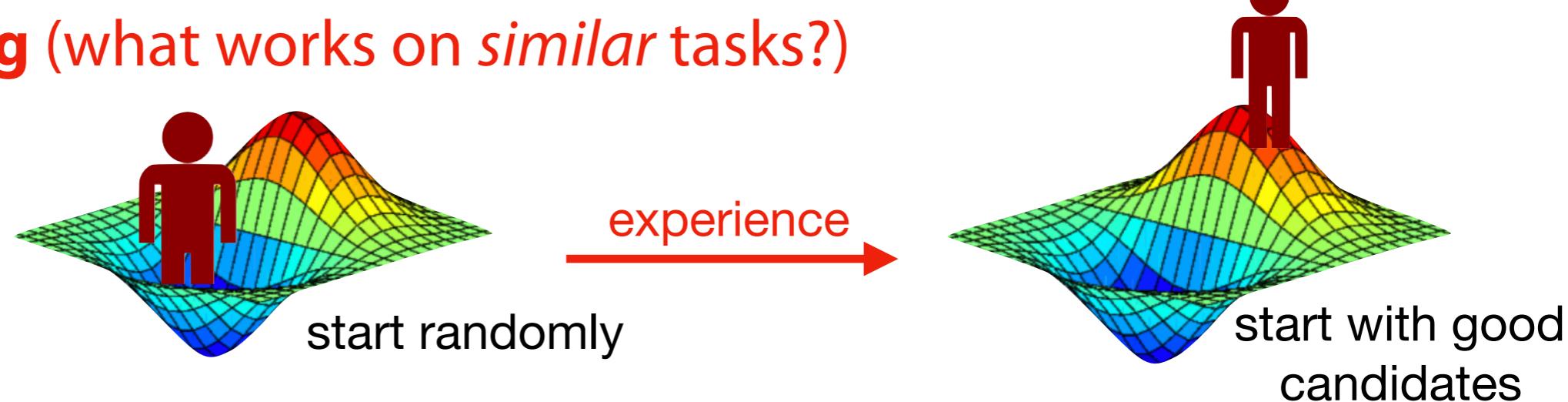
(j) MSCOCO

How?

Search space design (what works in general?)



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

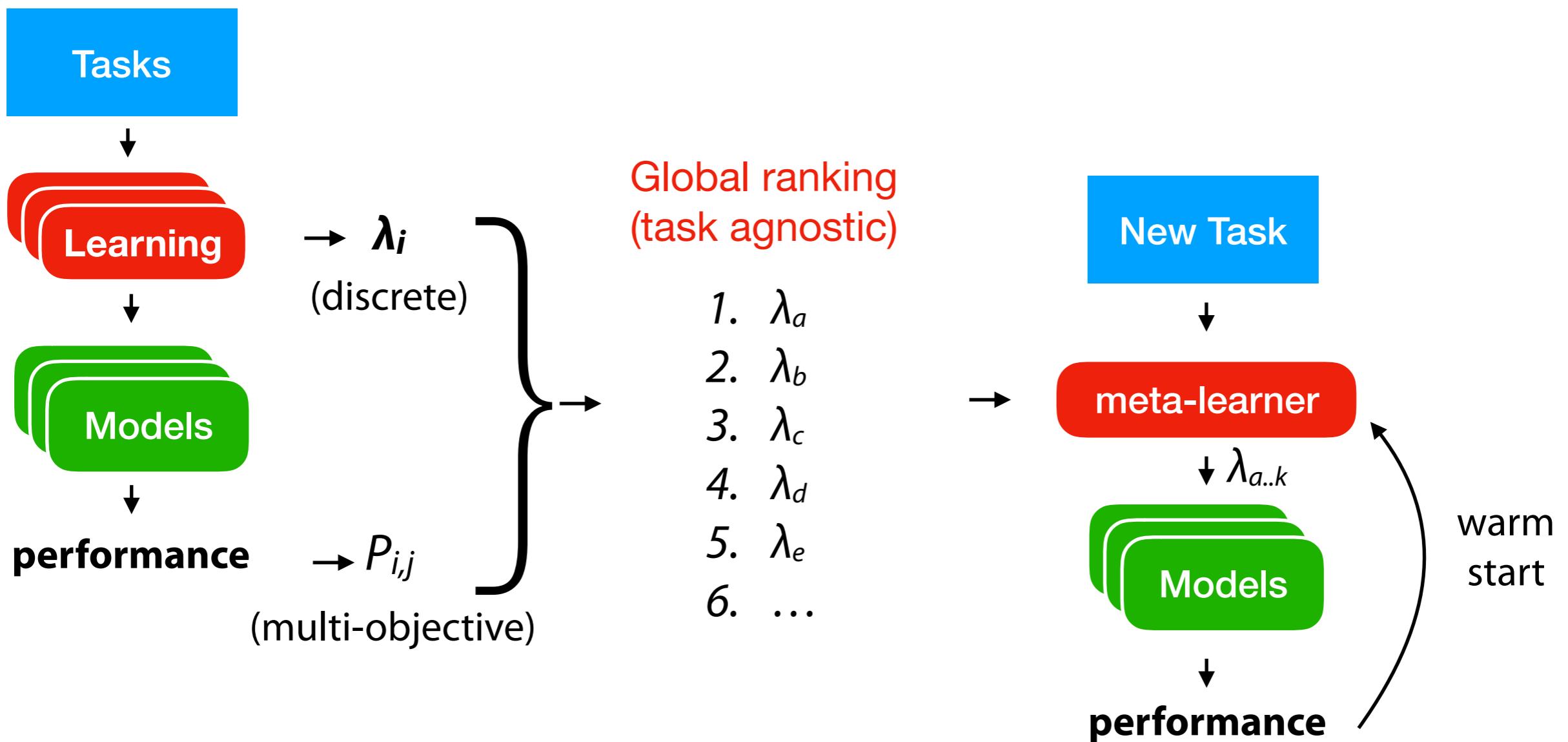


Model transfer (reuse on very similar tasks)



Rankings

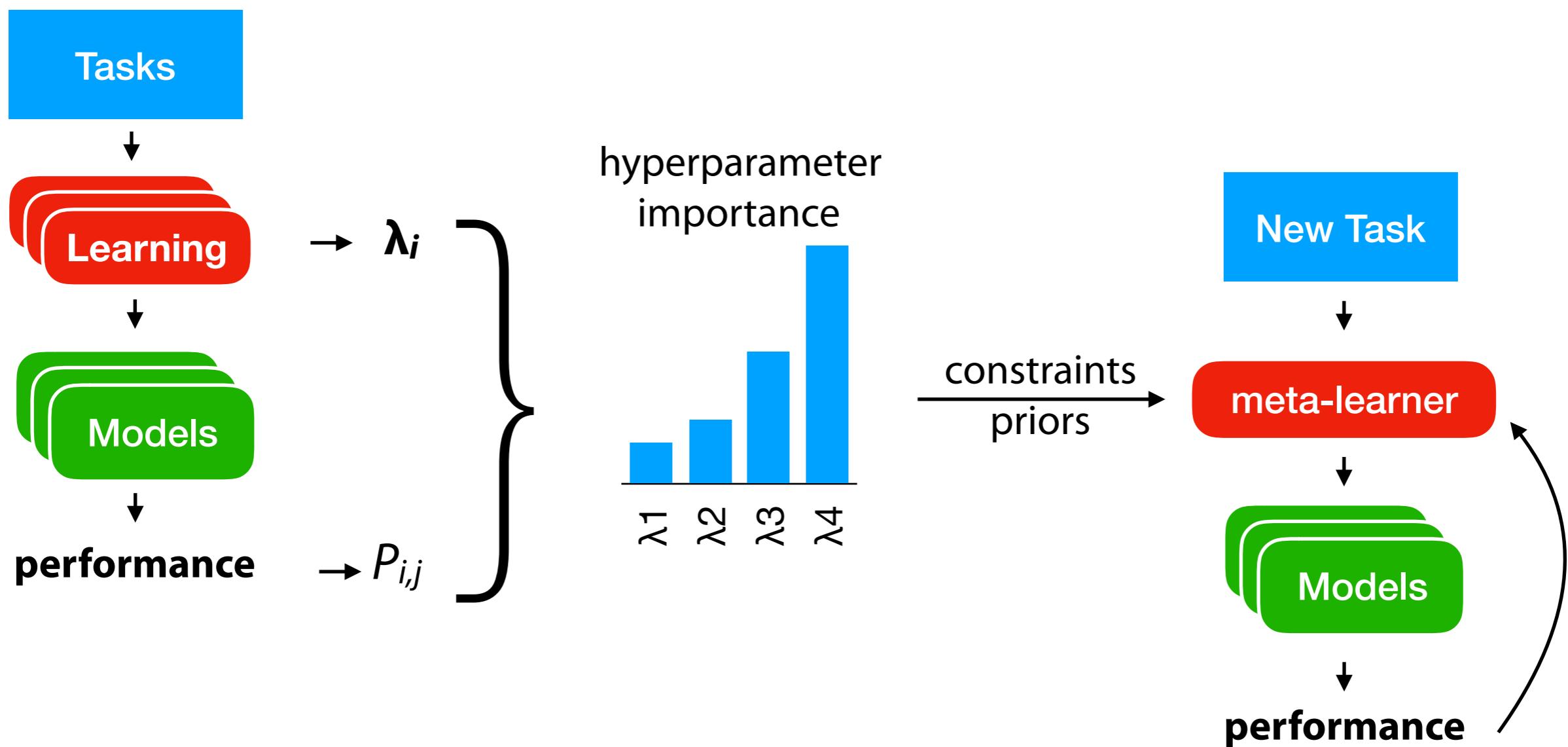
- Build a *global (multi-objective) ranking*, recommend the top-K
- Can be used as a *warm start* for optimization techniques
 - E.g. Bayesian optimization, evolutionary techniques,...



Hyperparameter importance

- **Functional ANOVA** ¹

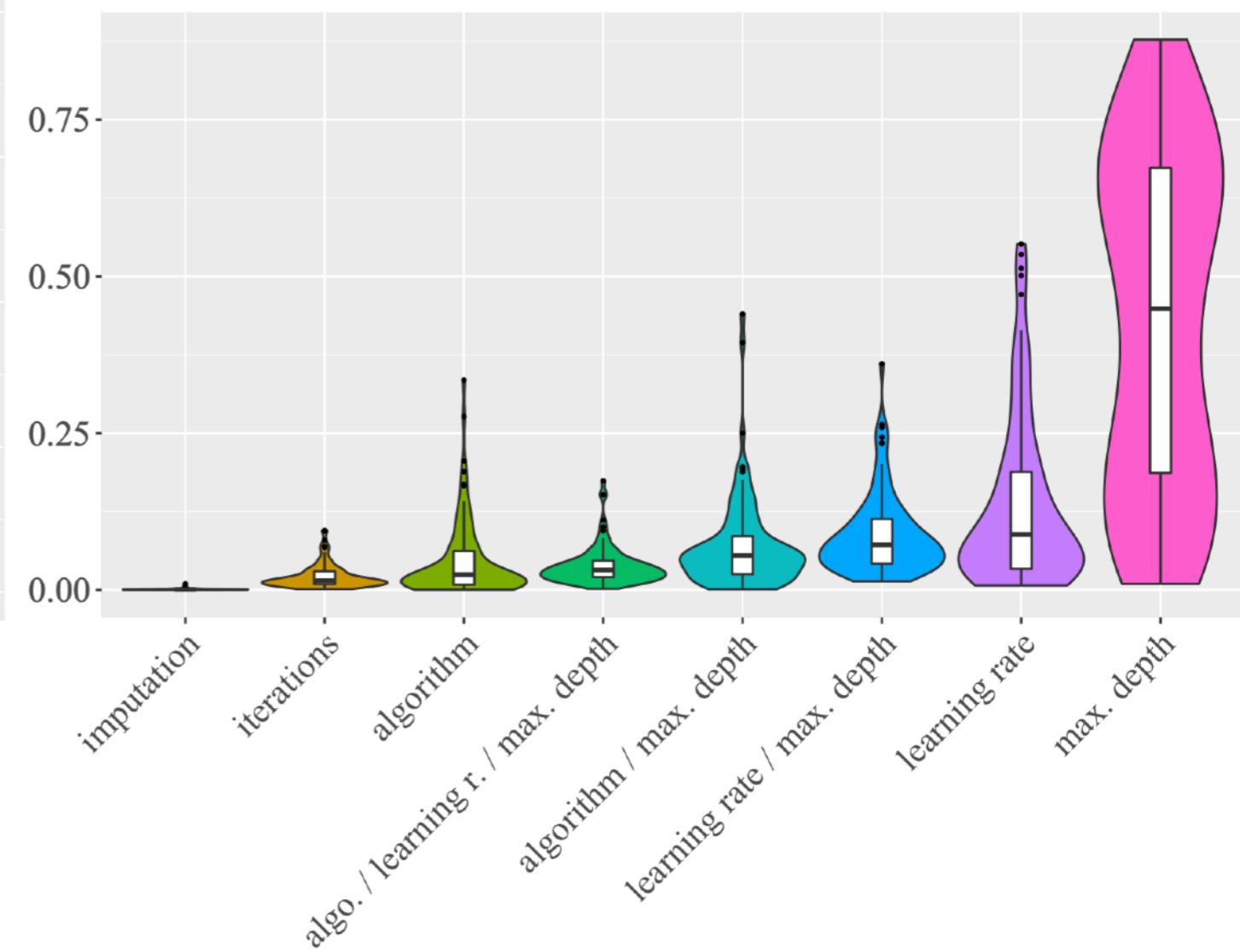
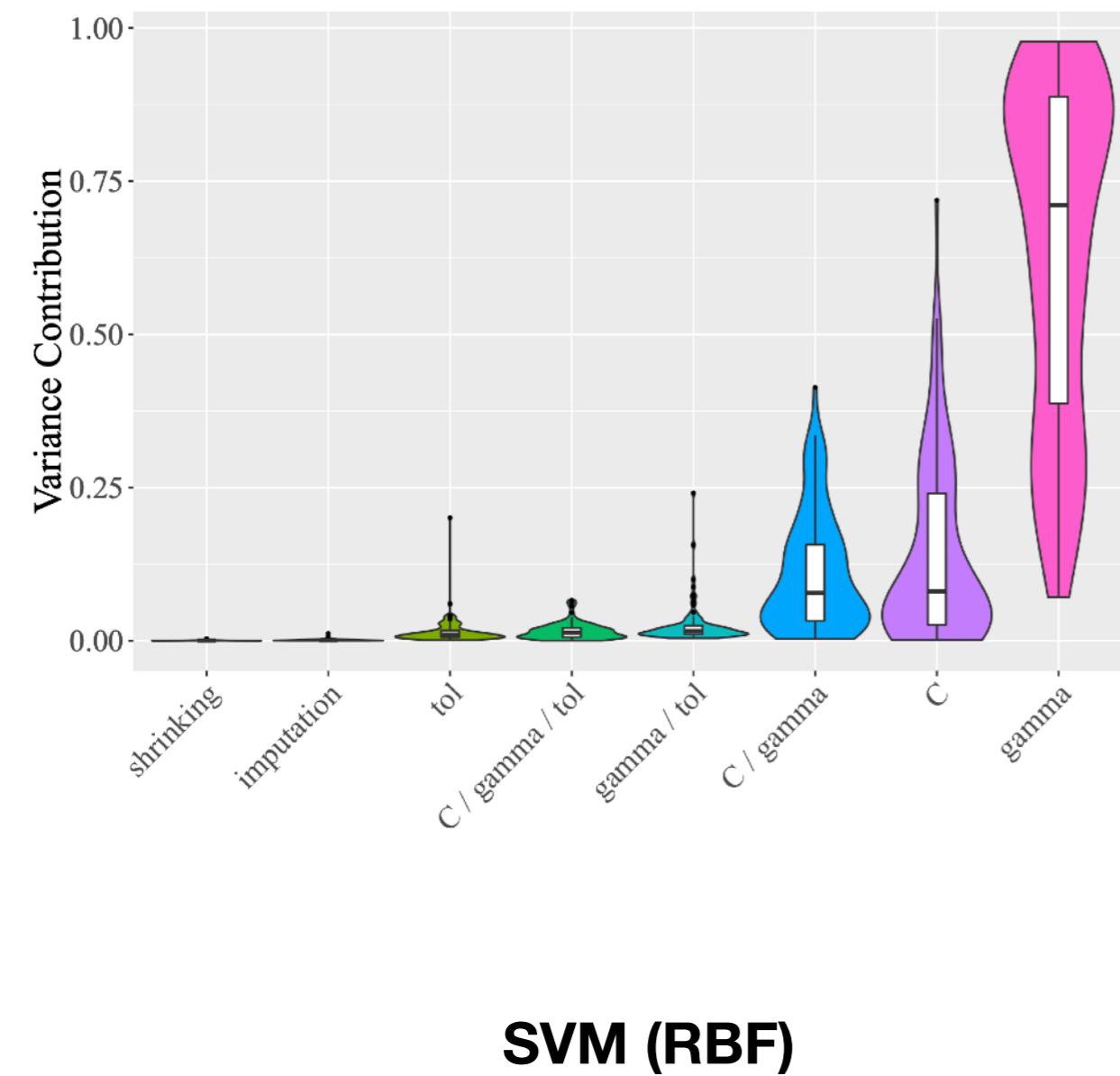
Select hyperparameters that cause variance in the evaluations.



Hyperparameter importance

- **Functional ANOVA** ¹

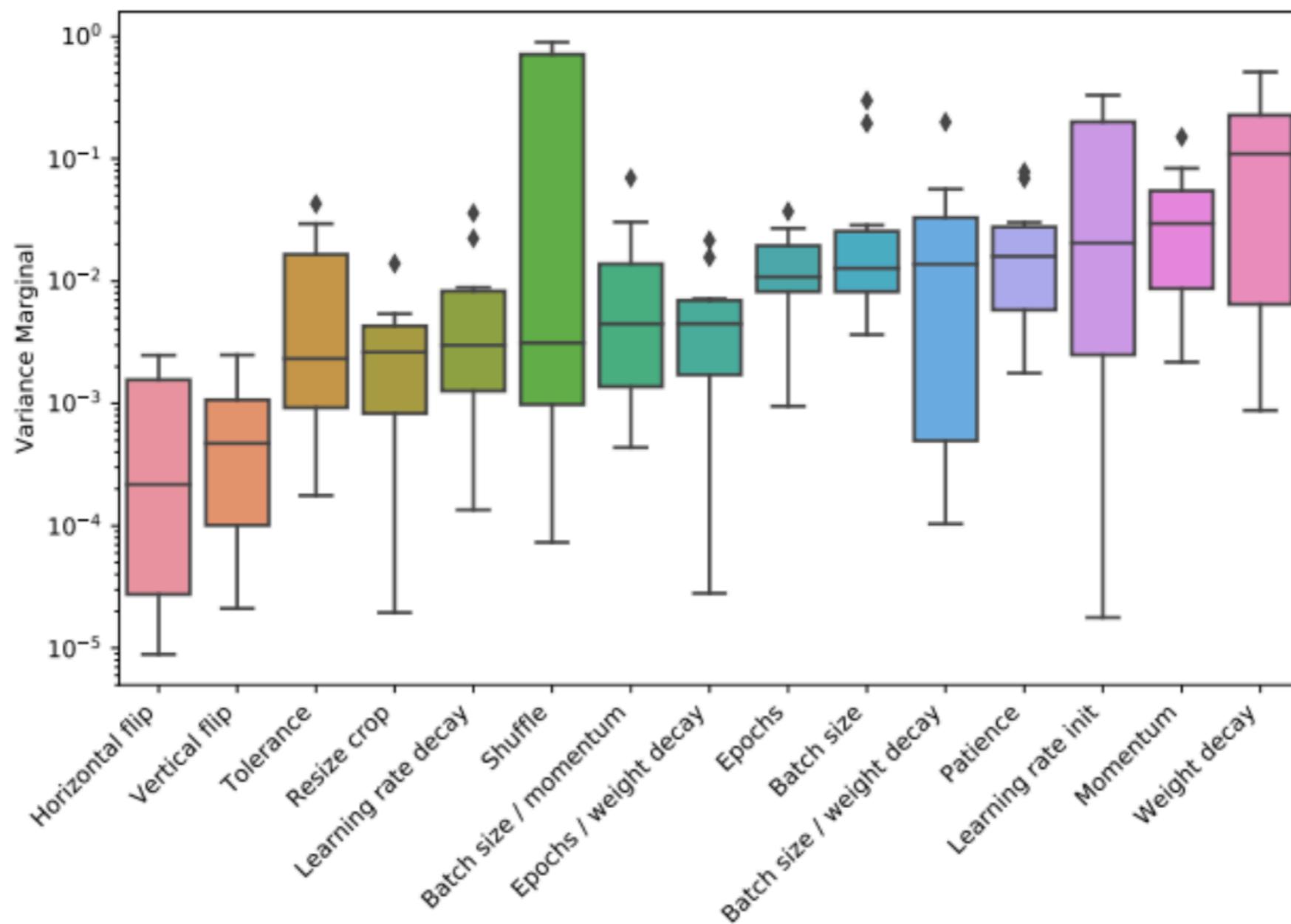
Select hyperparameters that cause variance in the evaluations.



Hyperparameter importance

- **Functional ANOVA** ¹

Select hyperparameters that cause variance in the evaluations.



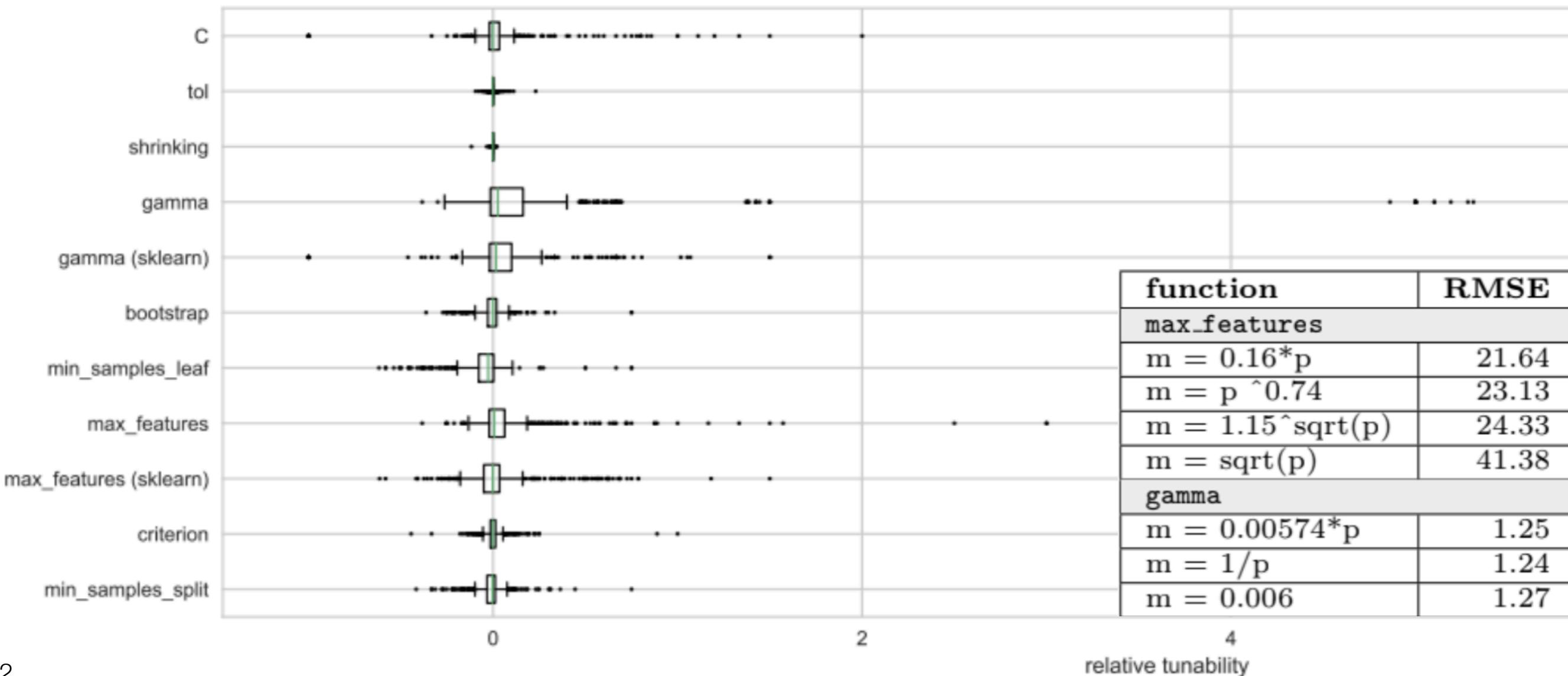
Hyperparameter importance

- **Functional ANOVA** ¹

Select hyperparameters that cause variance in the evaluations.

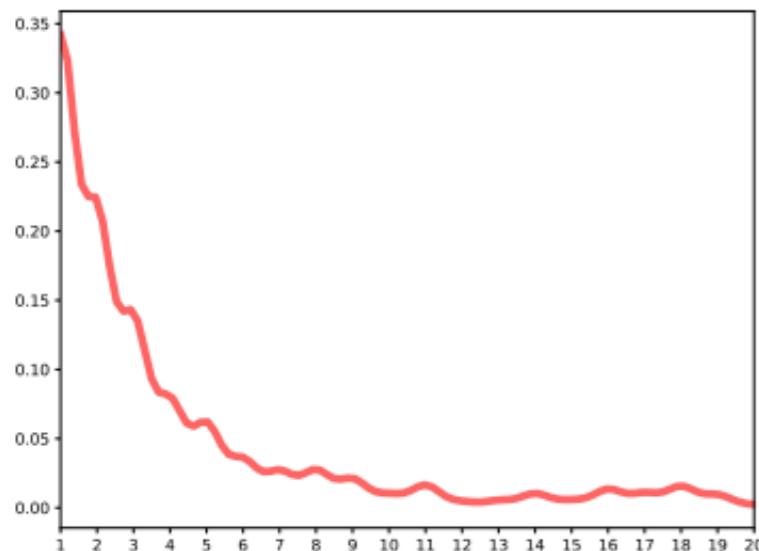
- **Tunability** ^{2,3,4}

Learn good defaults, measure improvement from tuning over defaults

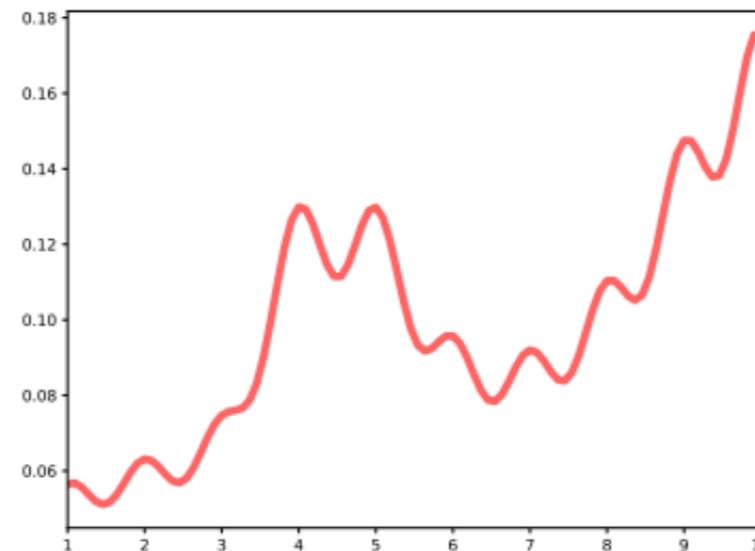


Hyperparameter priors

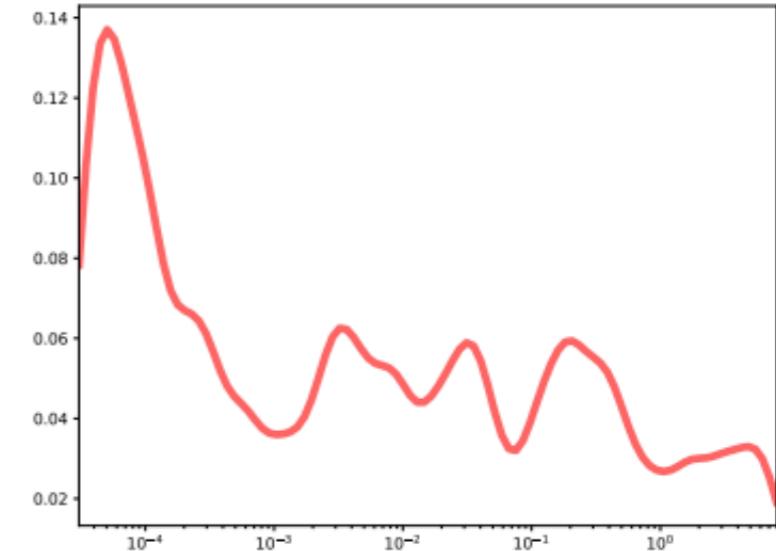
- Select hyperparameters that cause variance in the evaluations.
Learn priors for hyperparameter tuning (e.g. Bayesian optimization)



(a) RF: min. samples per leaf



(b) Adaboost: max. depth of tree

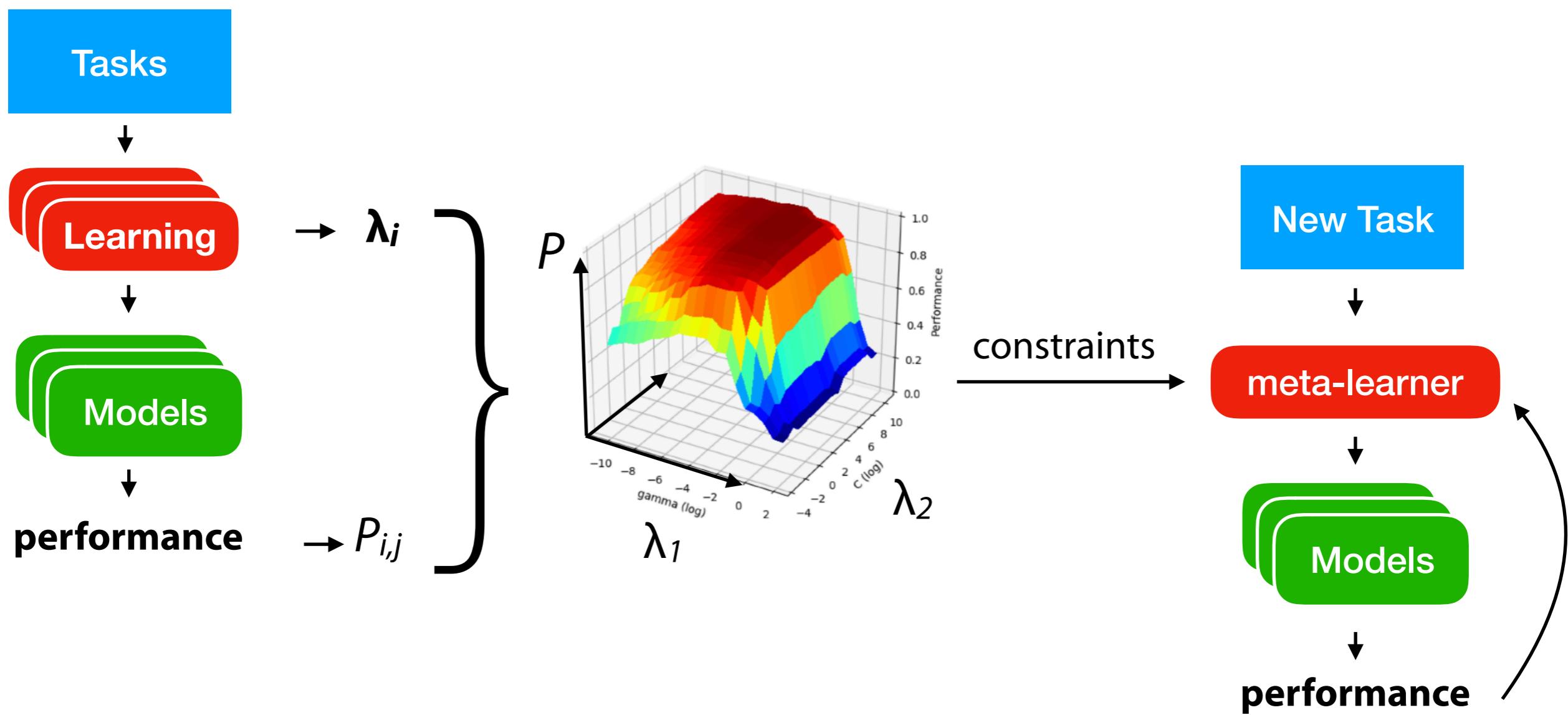


(c) SVM (RBF kernel): gamma

Priors (KDE) for the most important hyperparameters

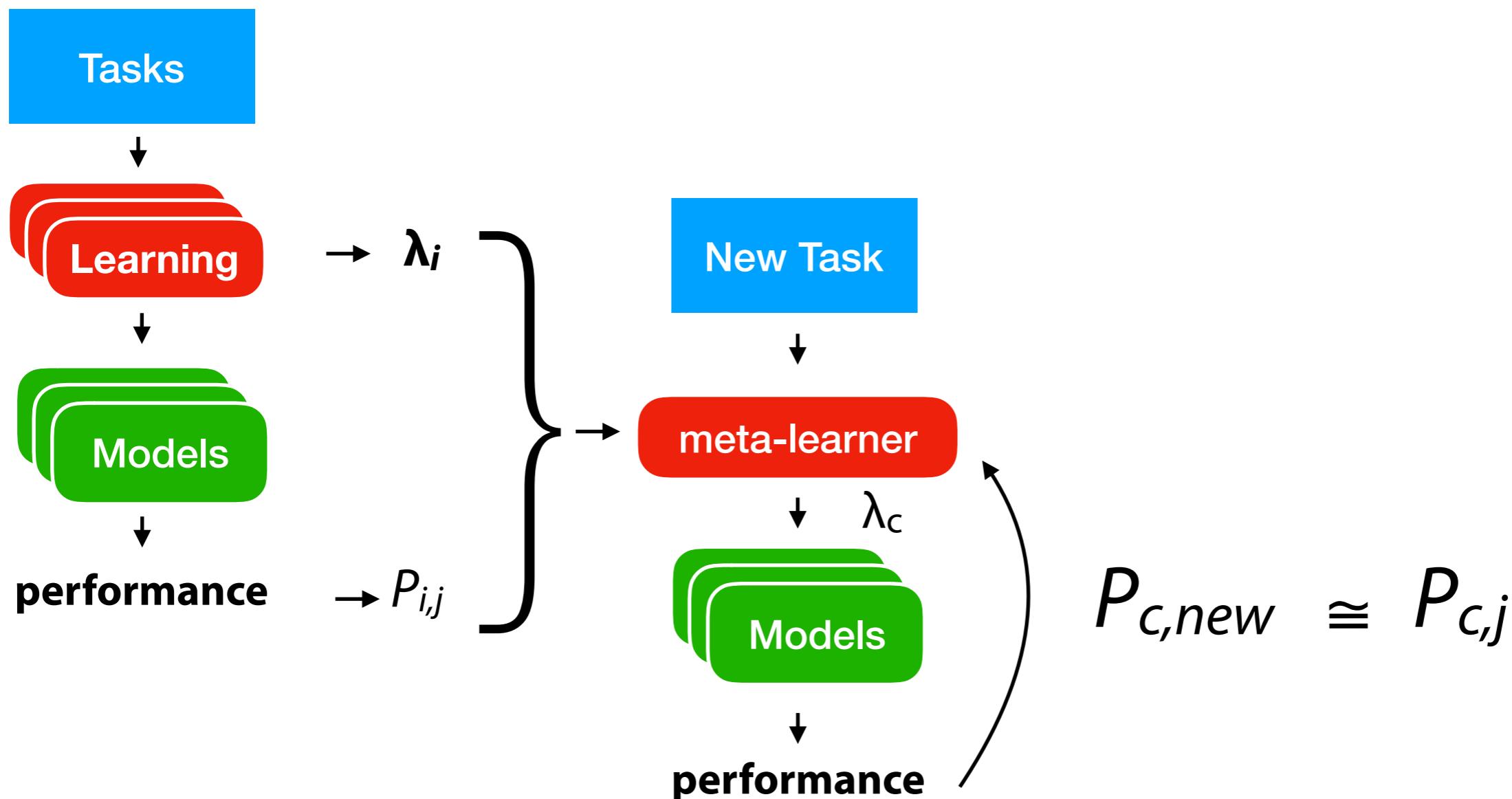
Hyperparameter priors

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



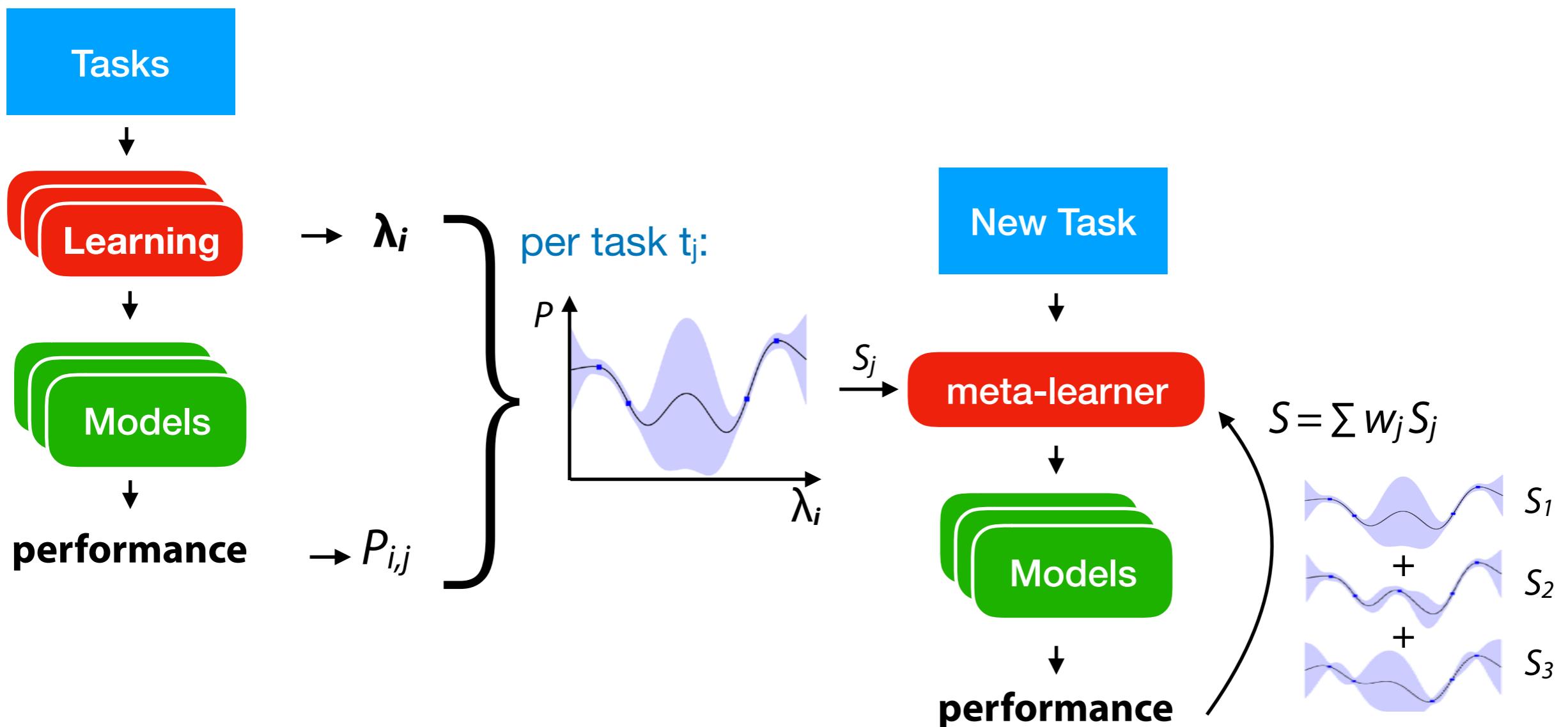
Learning task similarity

- 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*)



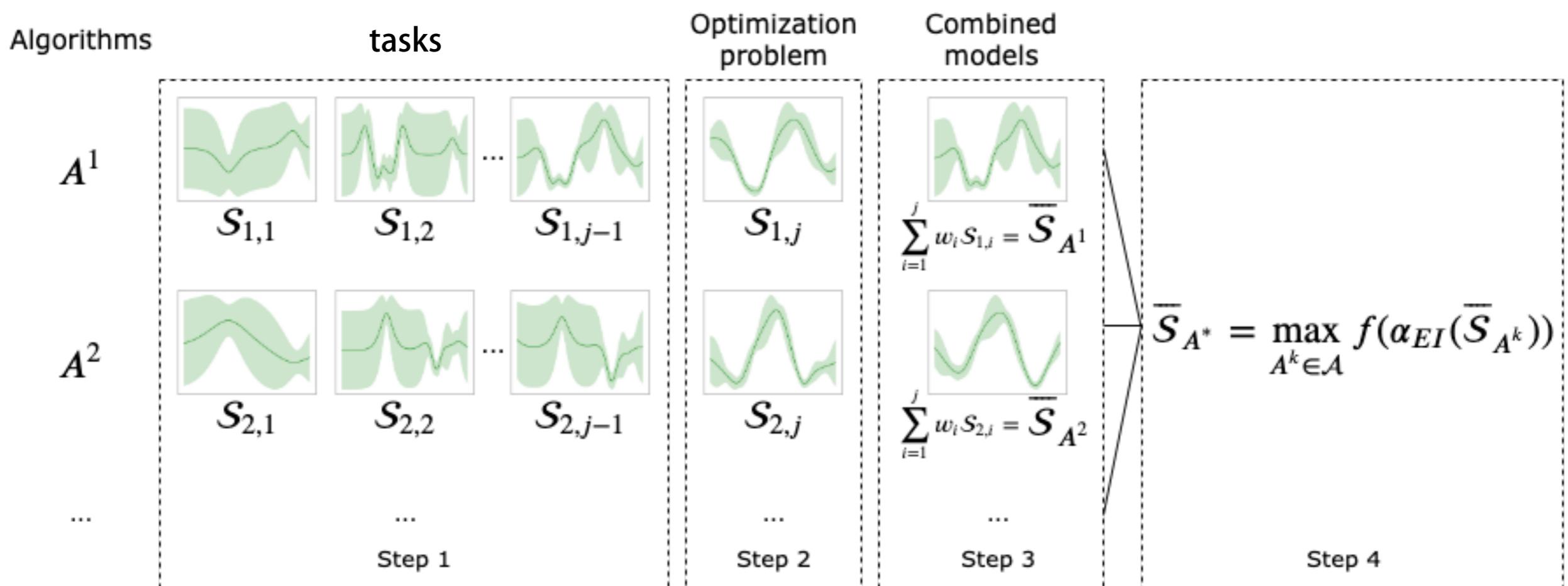
Surrogate model transfer

- 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 task²



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

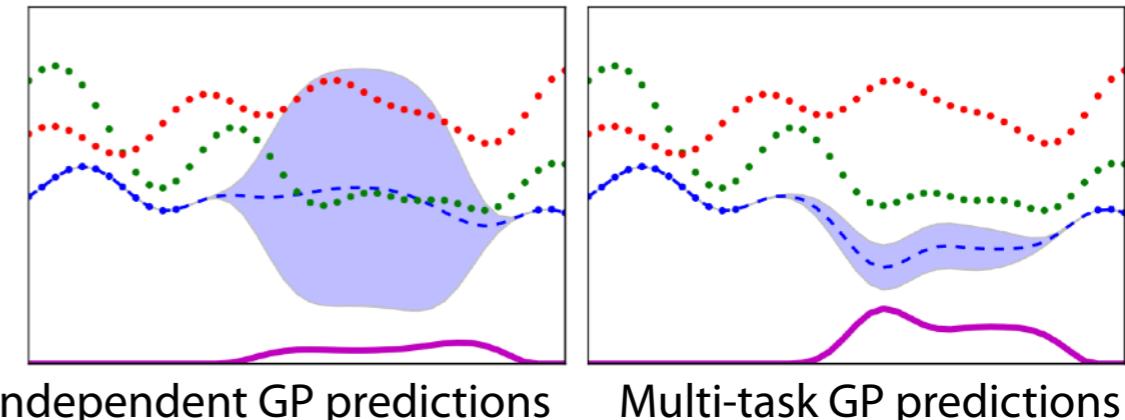


Meta-Learning

Multi-task Bayesian optimization

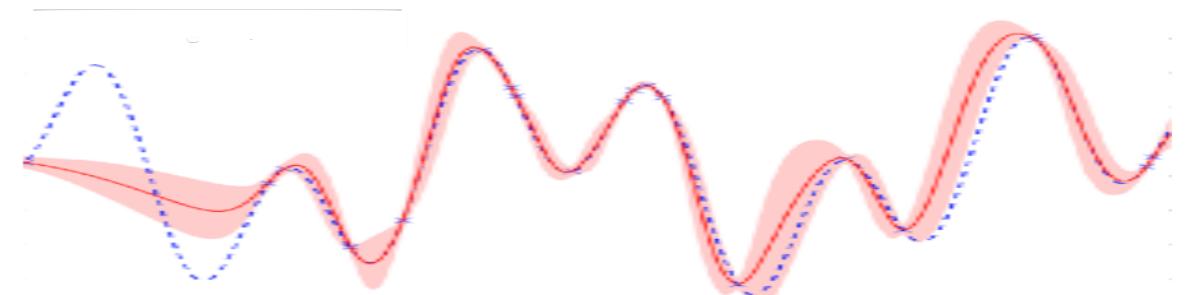
- **Multi-task Gaussian processes:** train surrogate model on t tasks simultaneously¹

- If tasks are similar: transfers useful info
- Not very scalable



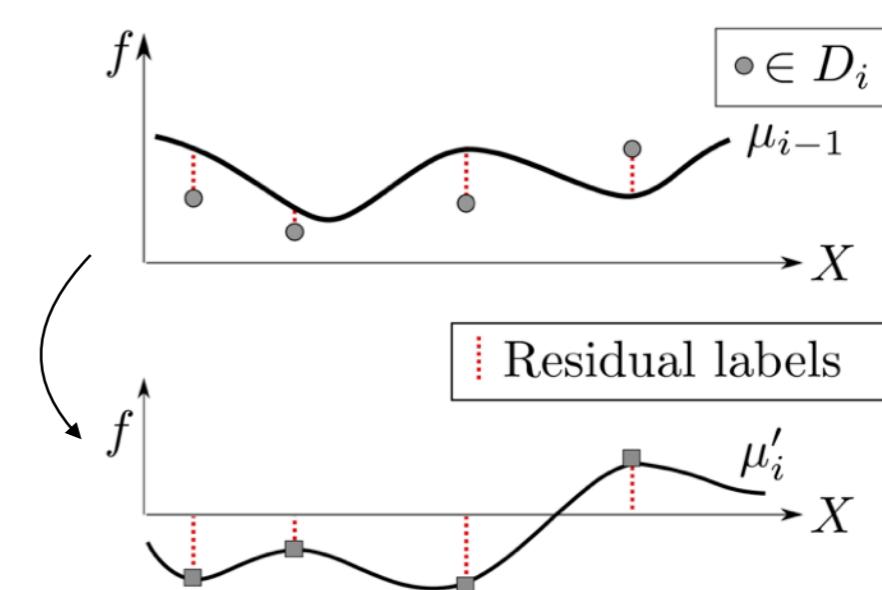
- **Bayesian Neural Networks** as surrogate model²

- Multi-task, more scalable



- **Stacking Gaussian Process regressors** (Google Vizier)³

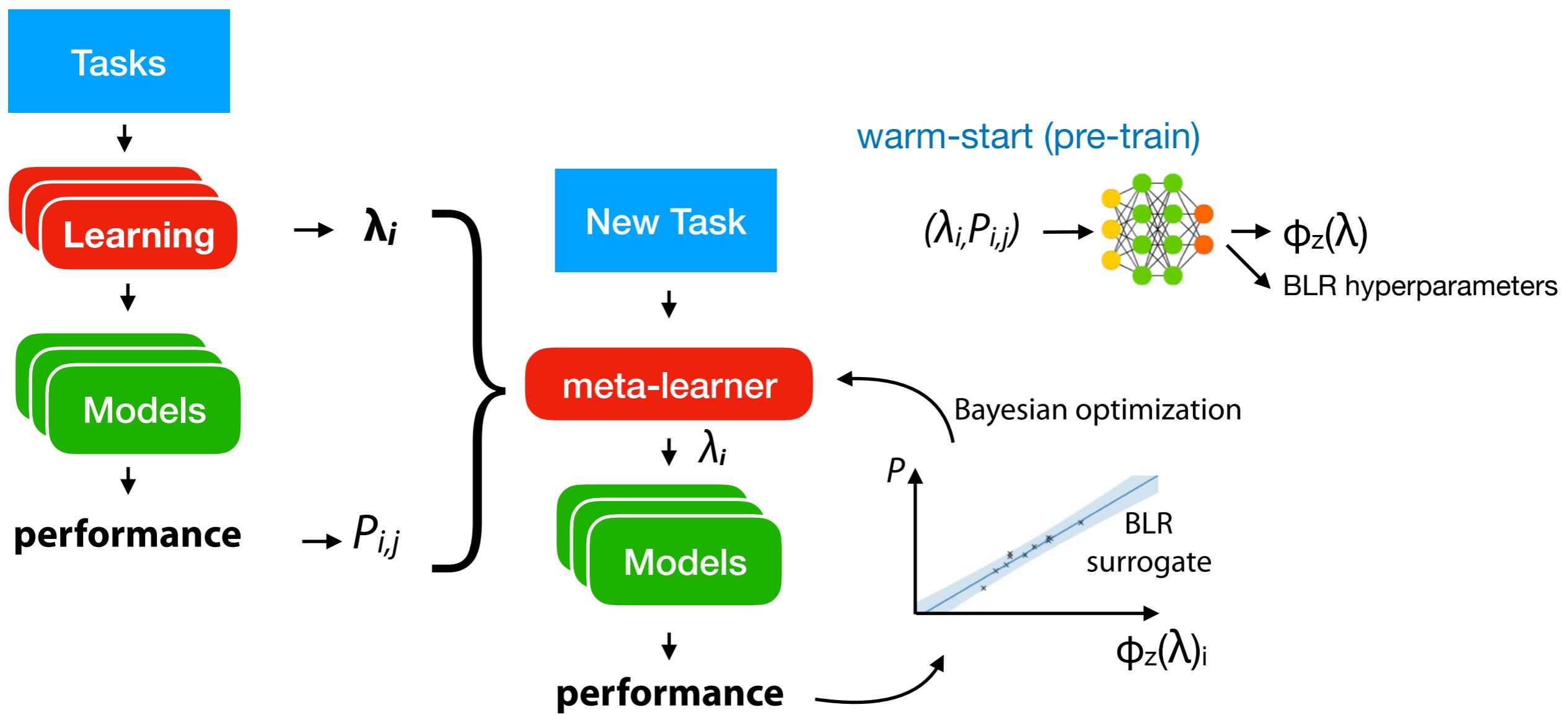
- Continual learning (sequential similar tasks)
- Transfers a prior based on residuals of previous GP



Meta-Learning

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



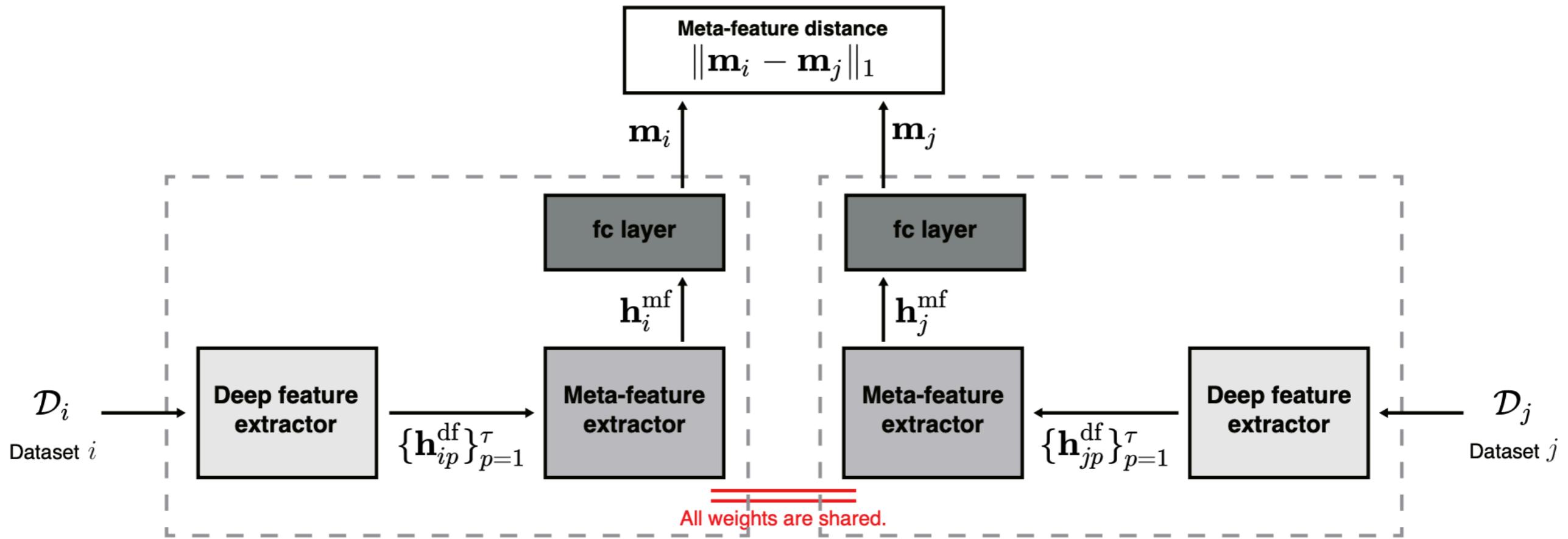
Meta-features

- Task descriptions, allow to compute a distance between task
- **Hand-crafted (interpretable) meta-features¹**
 - **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

- **Learning a joint task representation**

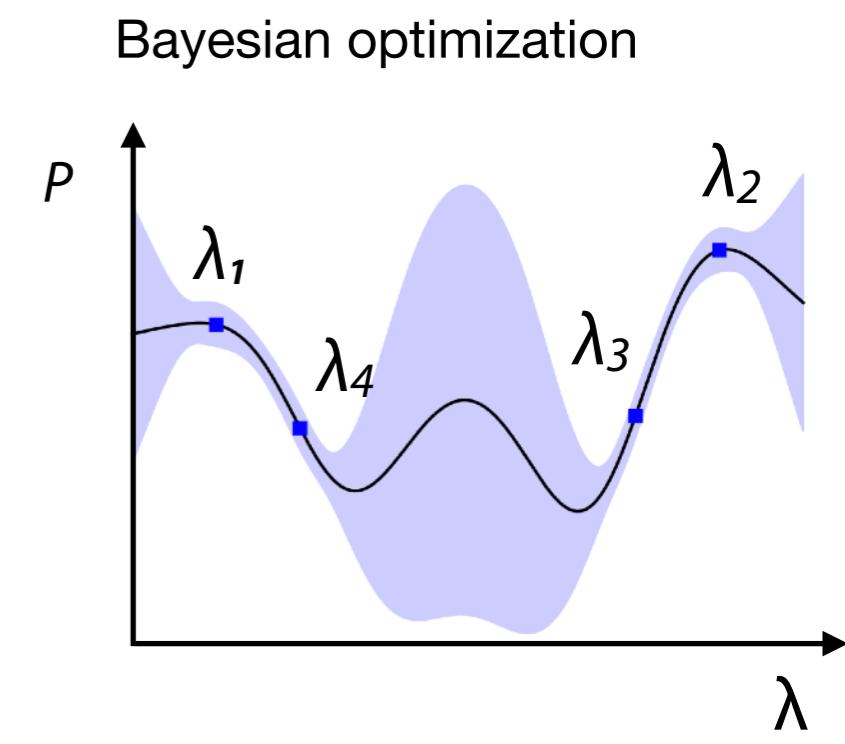
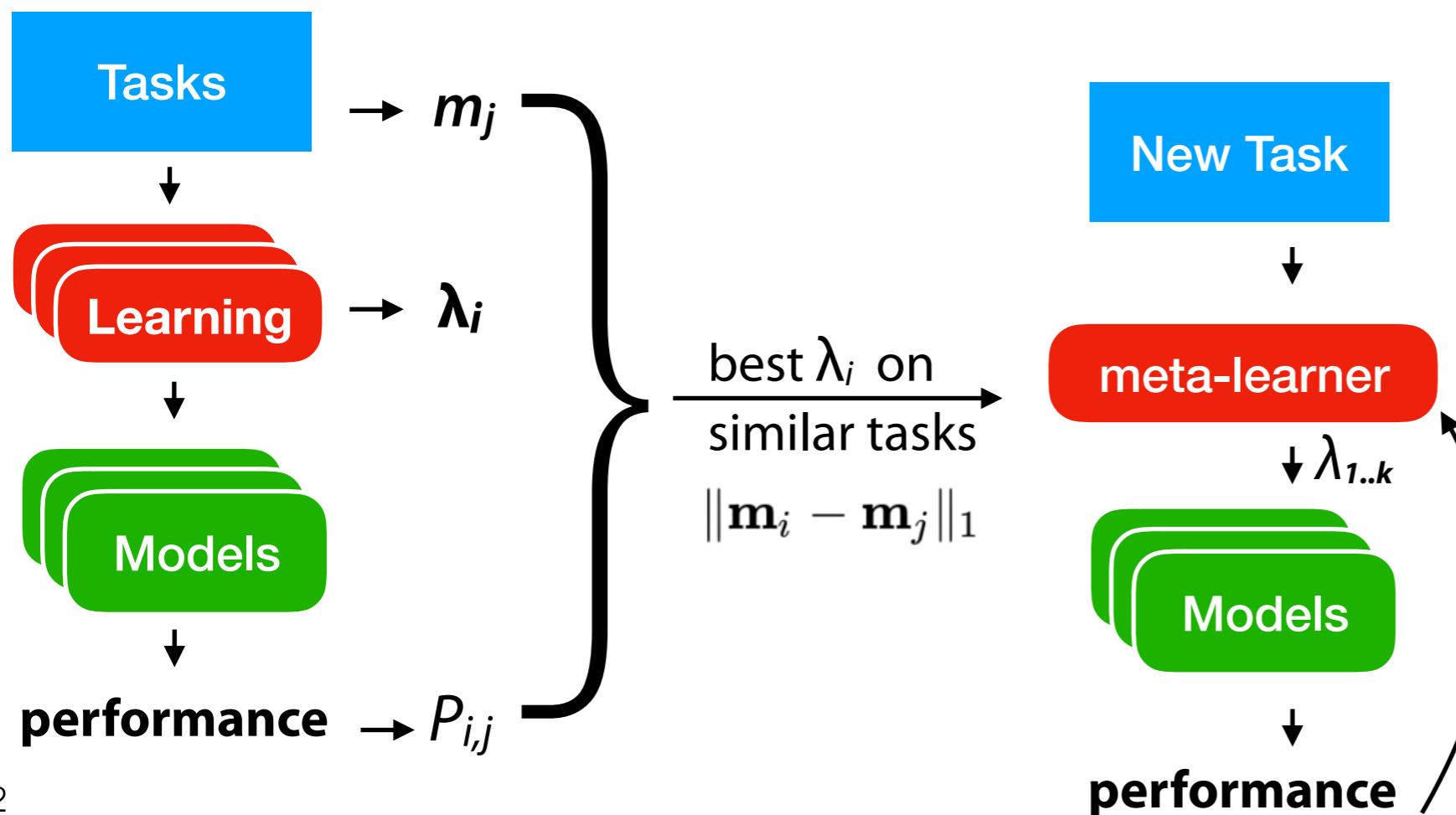
- Deep metric learning: learn a representation h^{mf} using ground truth distance
 - Siamese Network: similar task, similar representation ¹
 - Feed through pretrained CNN, extract vector from weights ²
- Recommend neural architectures, or warm-start neural architecture search
- Ground truth can be obtained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



Meta-Learning

Warm-starting from similar tasks

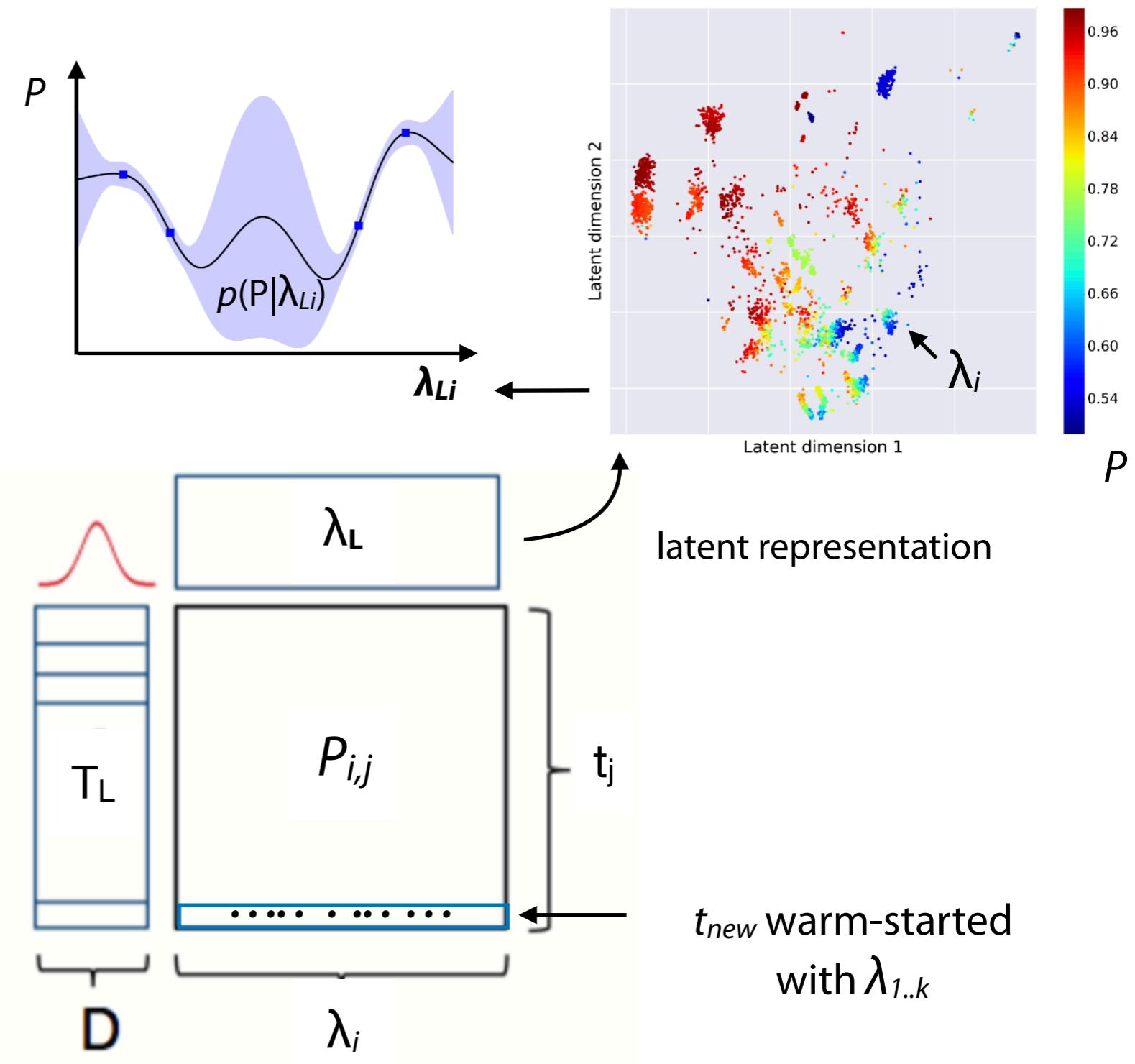
- Find k most similar tasks, warm-start search with best λ_i
 - Auto-sklearn: Bayesian optimization (SMAC)
 - Meta-learning yield better models, faster
 - Winner of AutoML Challenges



Meta-Learning

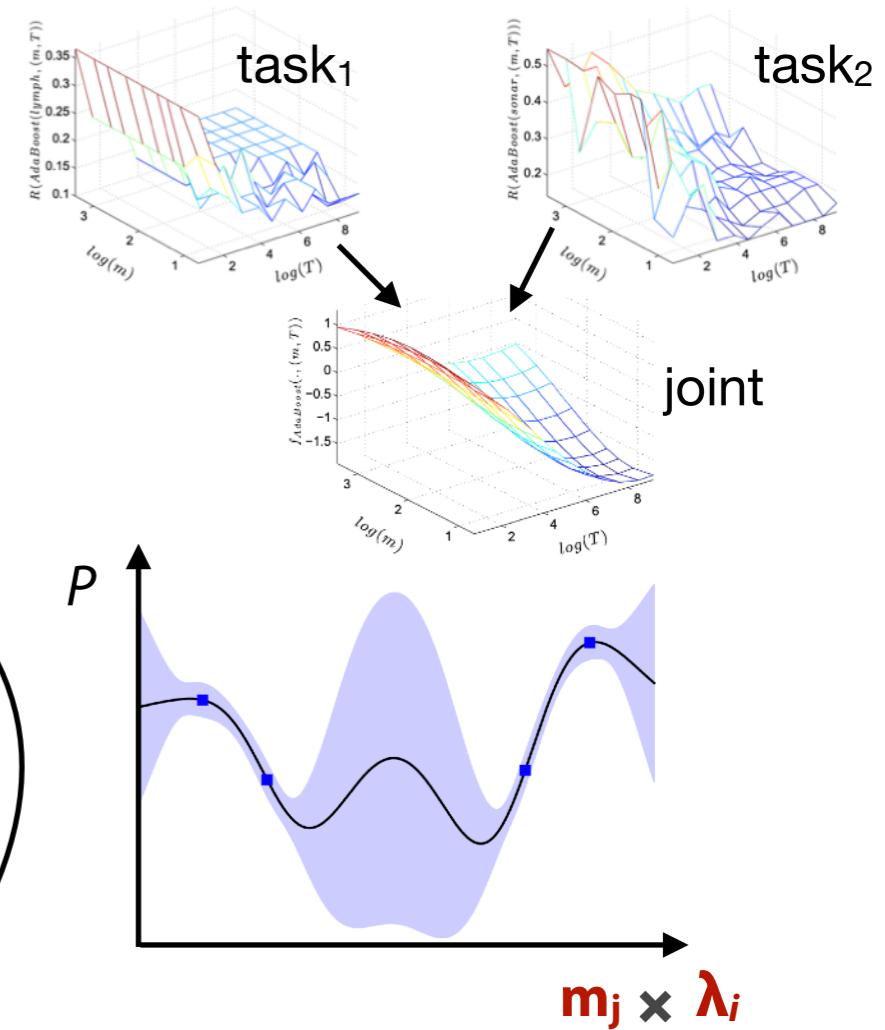
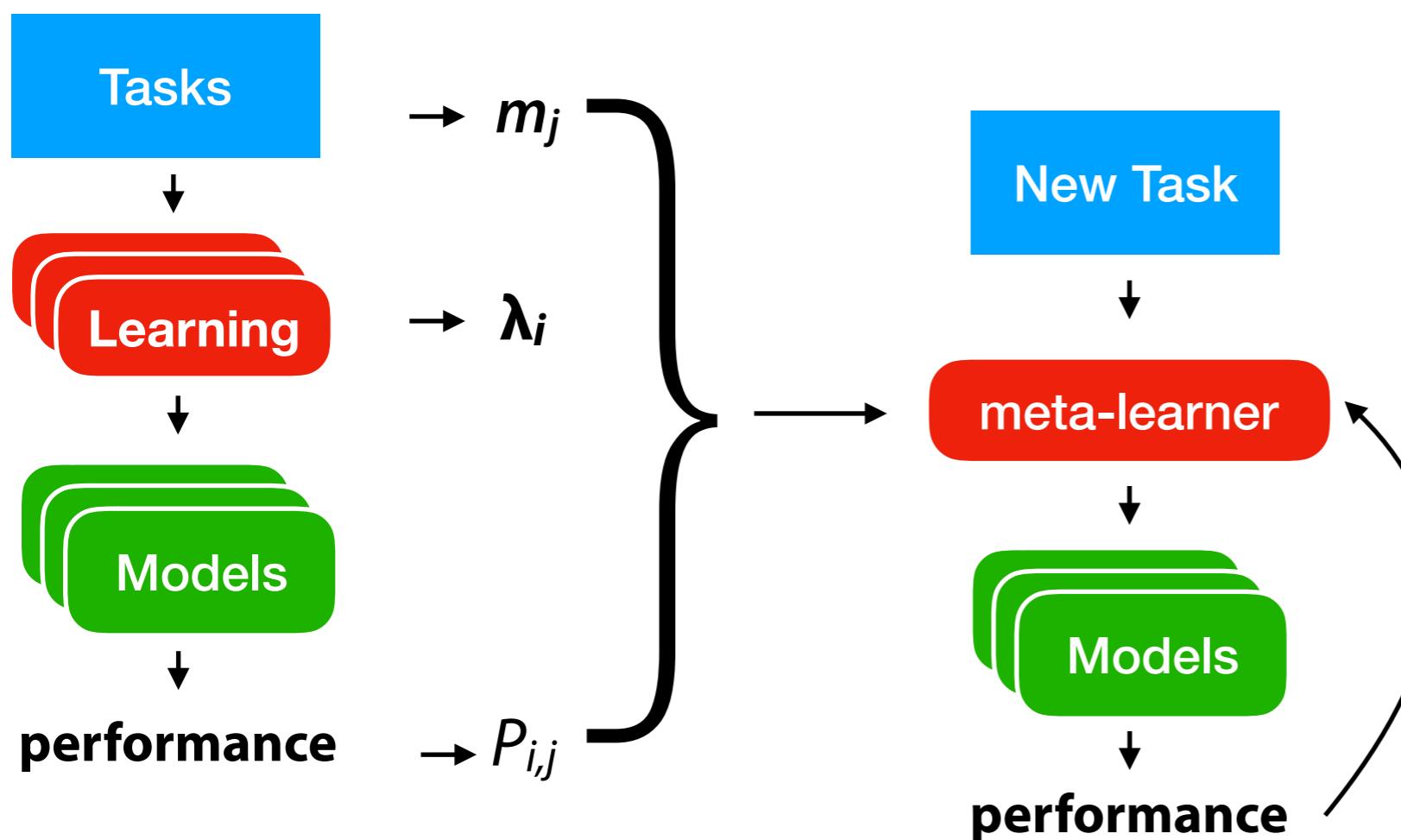
Warm-starting from similar tasks

- 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



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**



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



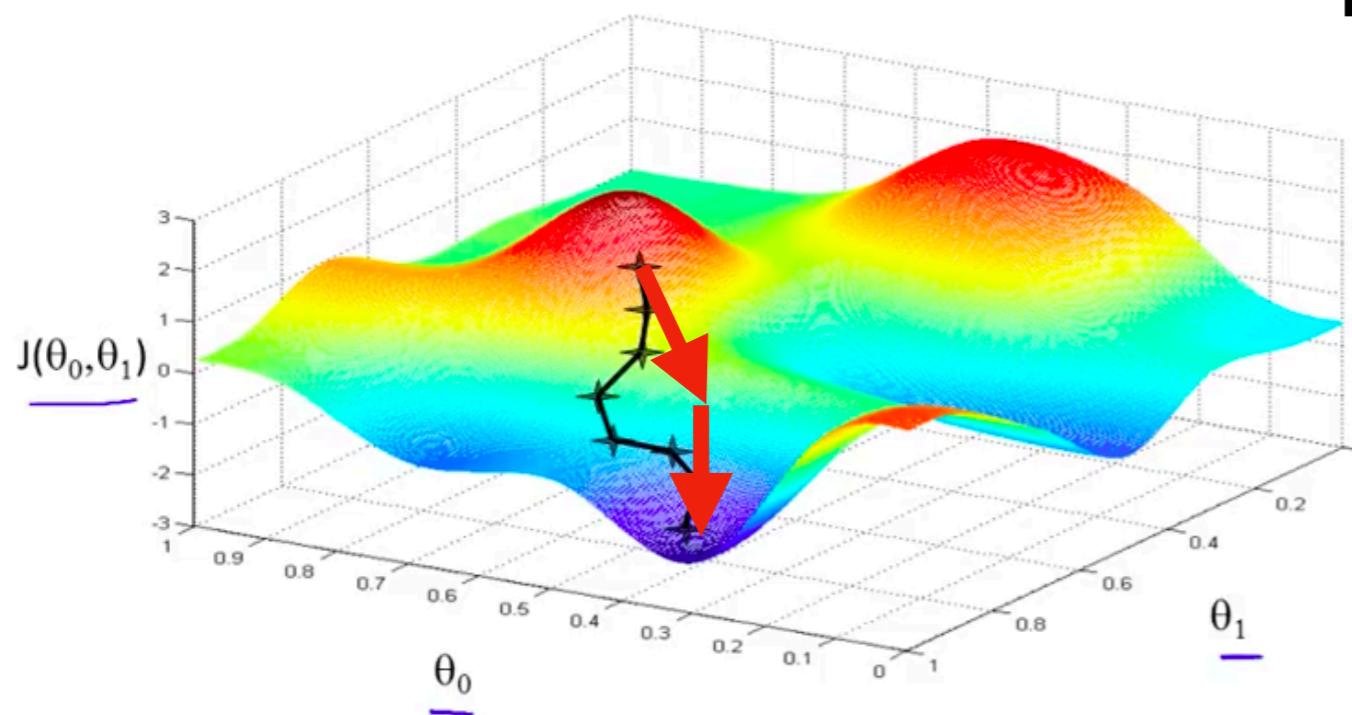
- Predict performance / runtime for given Θ_i and task ⁴



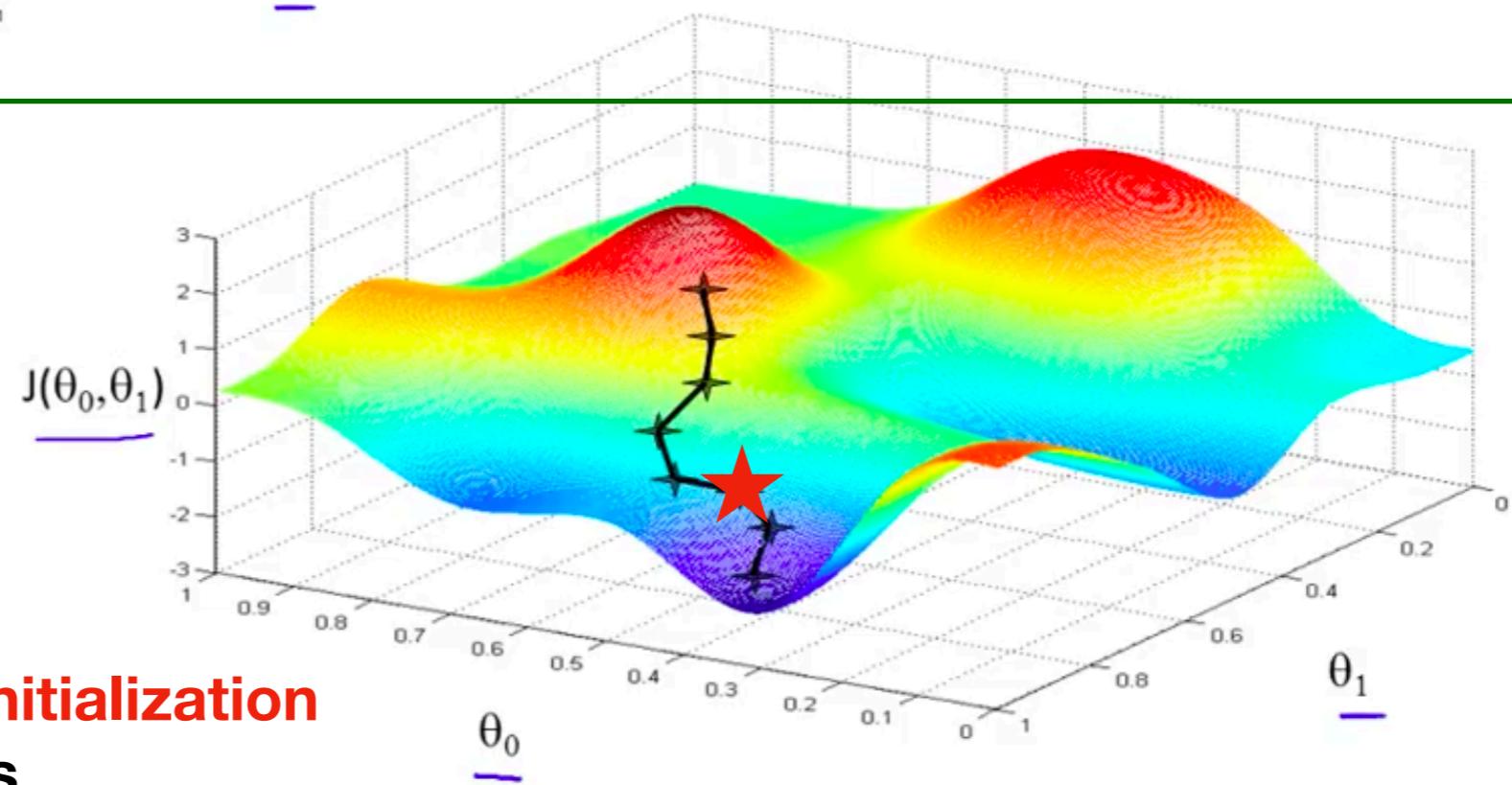
- Can be integrated in larger AutoML systems: warm start, guide search,...

Learning to learn

Transfer knowledge between very similar tasks



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

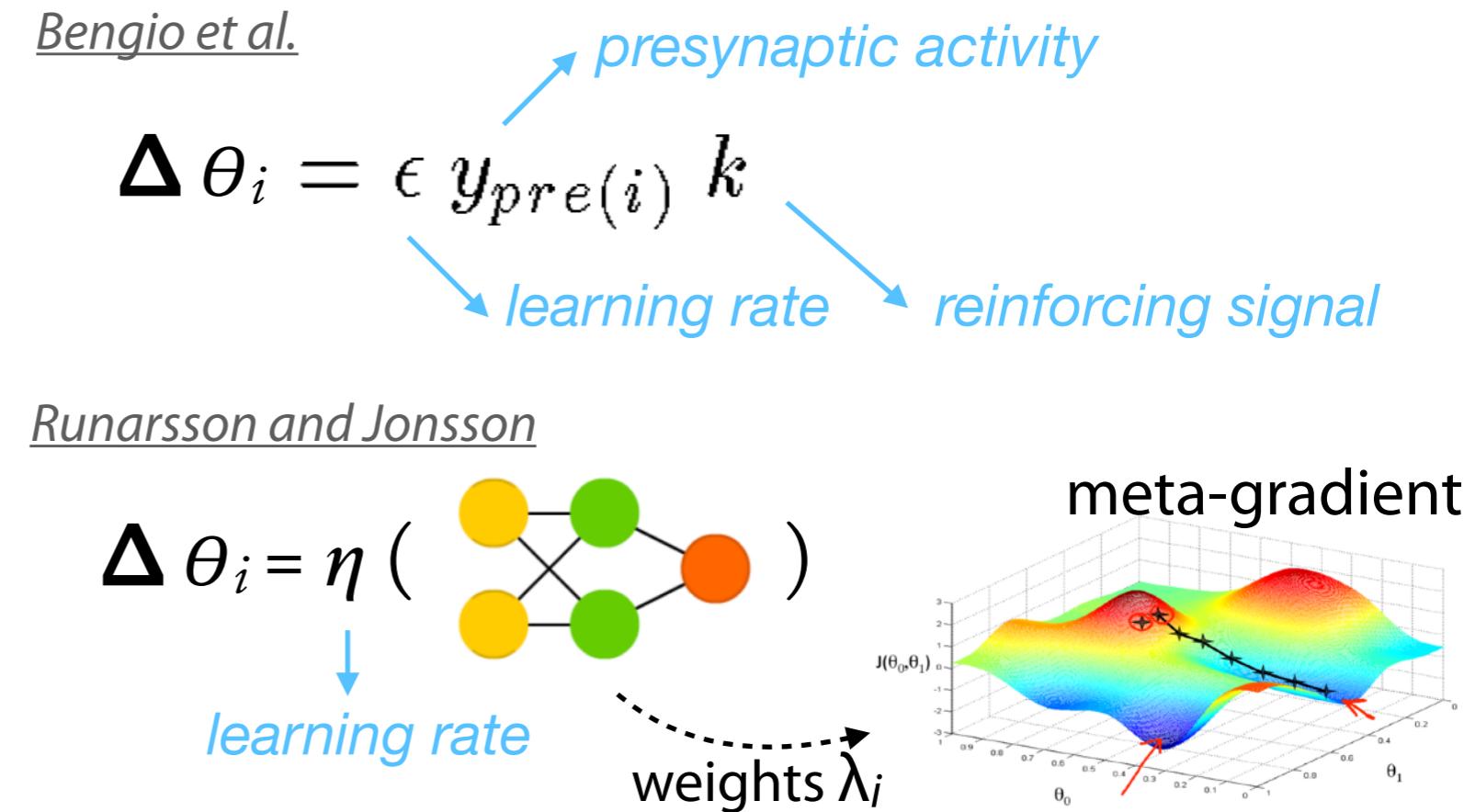
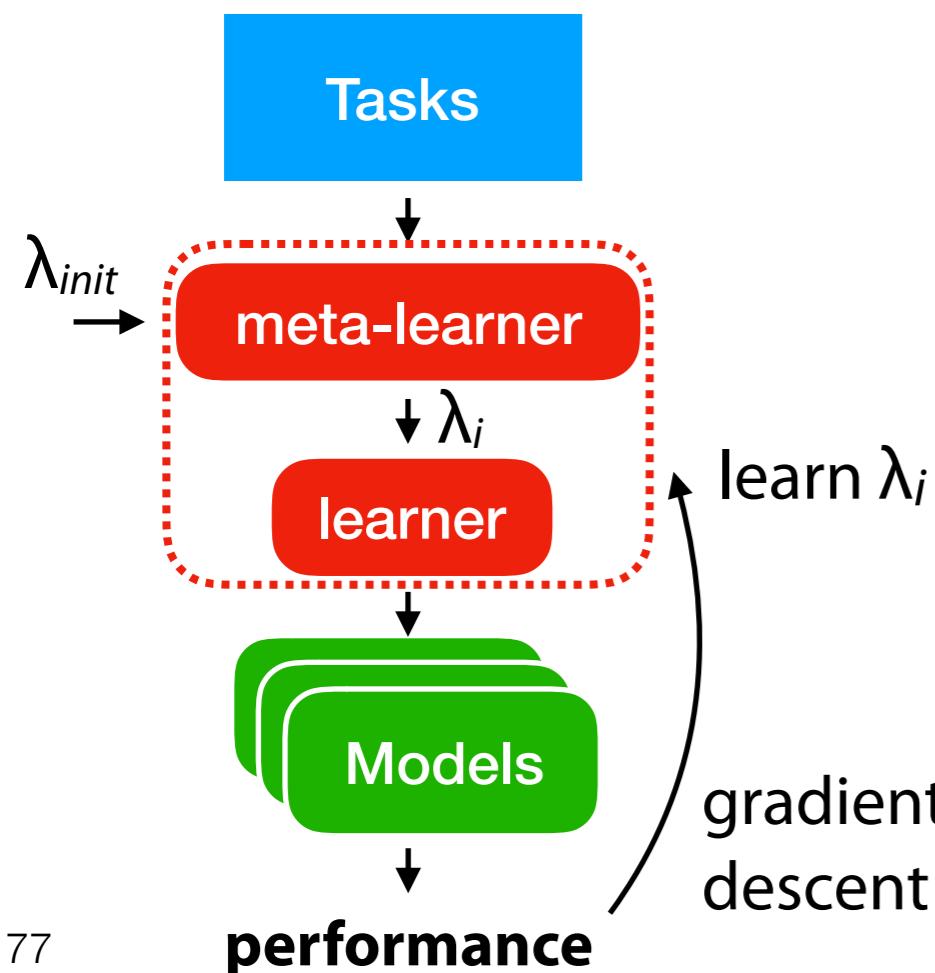


Learn and transfer a weight initialization for a group of tasks

Learning to learn by gradient descent

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

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

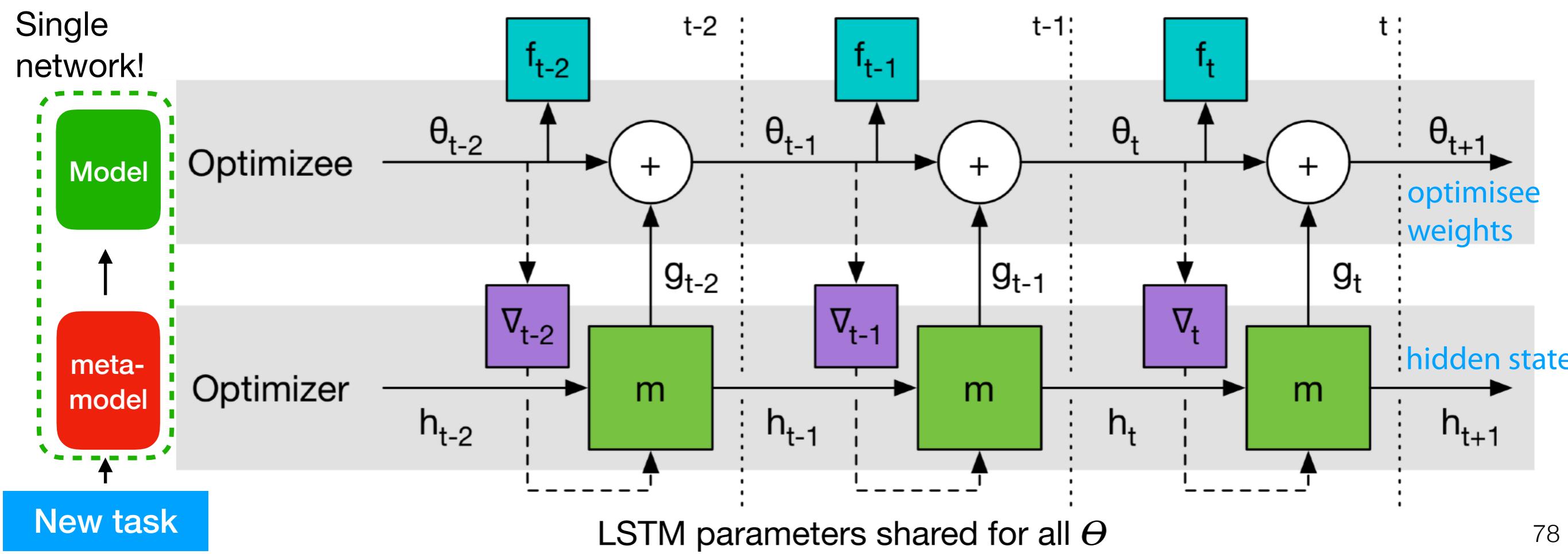


Meta-Learning

Learning to learn gradient descent

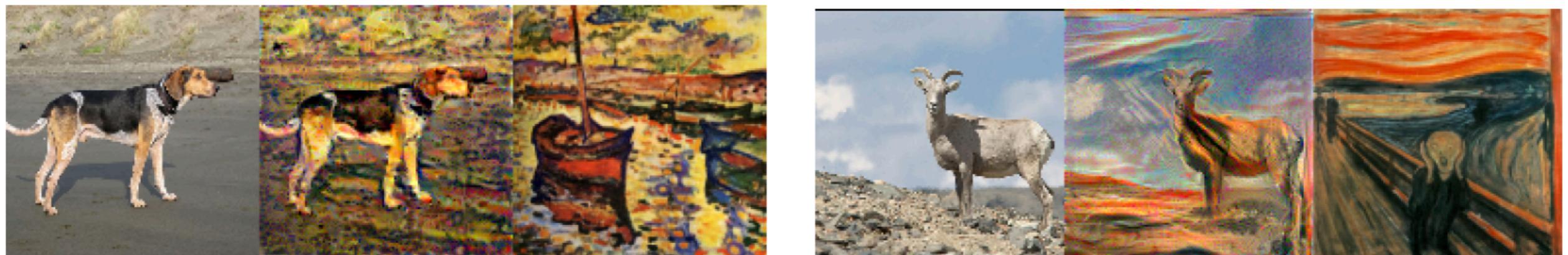
by gradient descent

- 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



Learning to learn gradient descent

by gradient descent

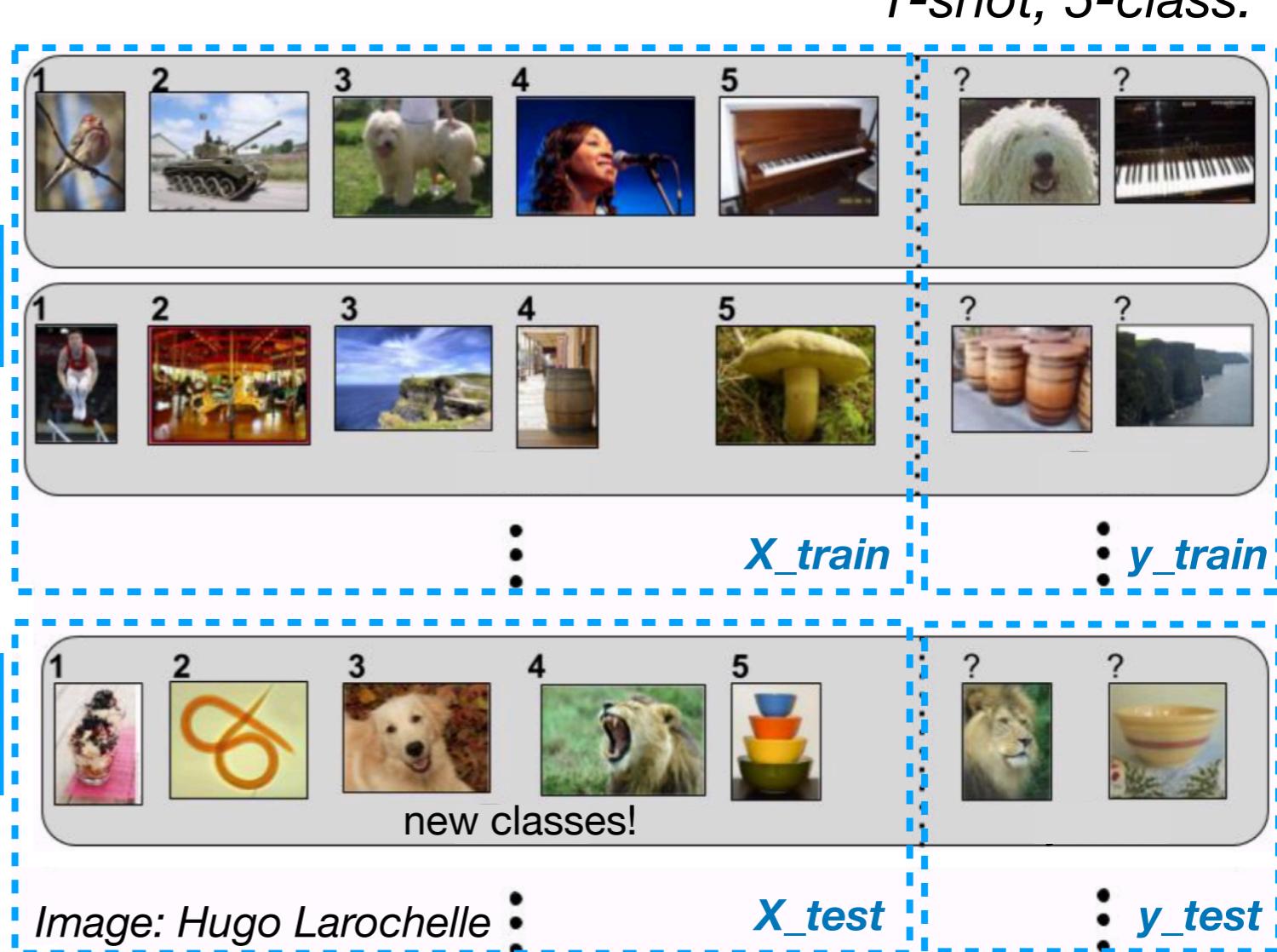
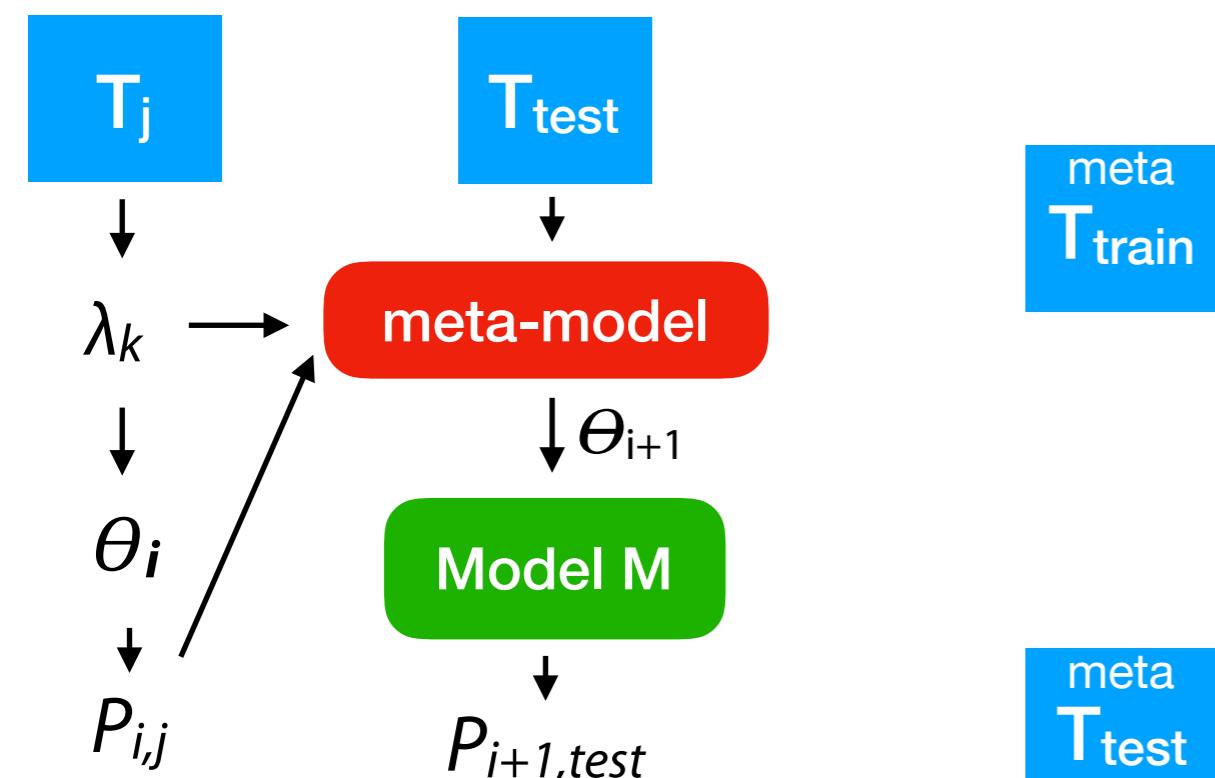


- Left: optimizer and optimizee trained to do style transfer
- Right: optimizer solves similar tasks (different style, content and resolution) without any more training data

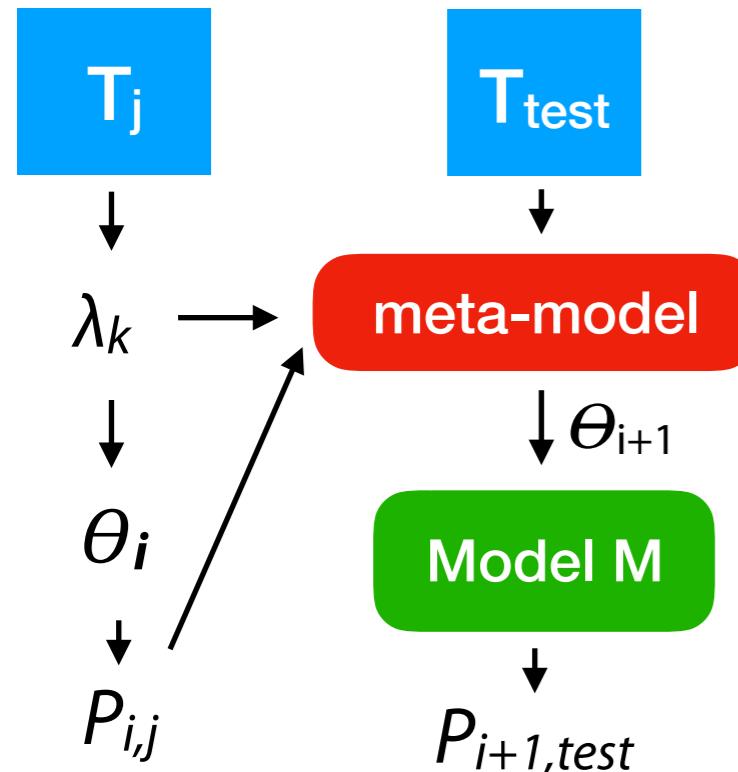
Application: Few-shot learning

- 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

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



Few-shot learning: approaches



- 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*
 - ...

Meta-Learning

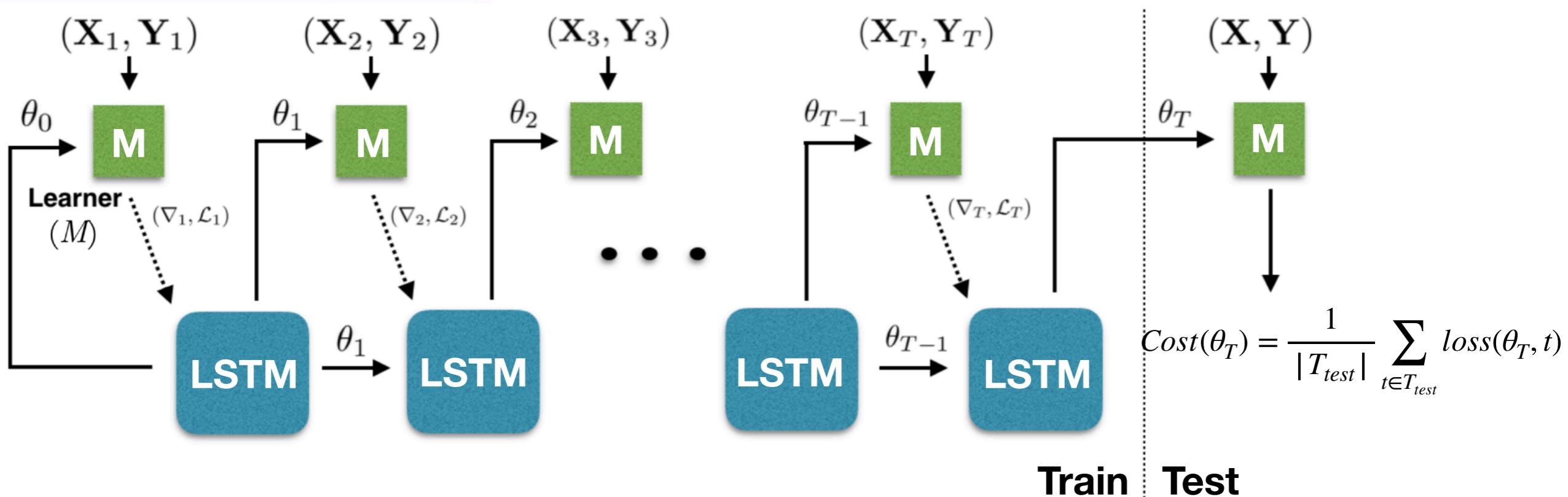
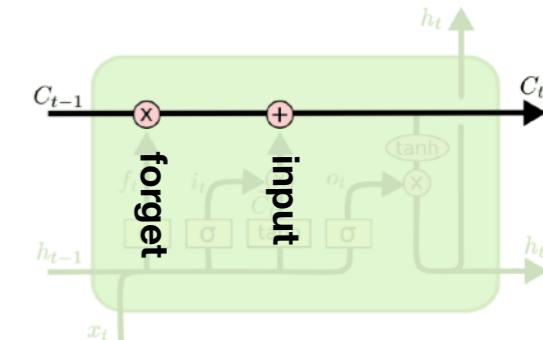
LSTM meta-learner + gradient descent

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

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

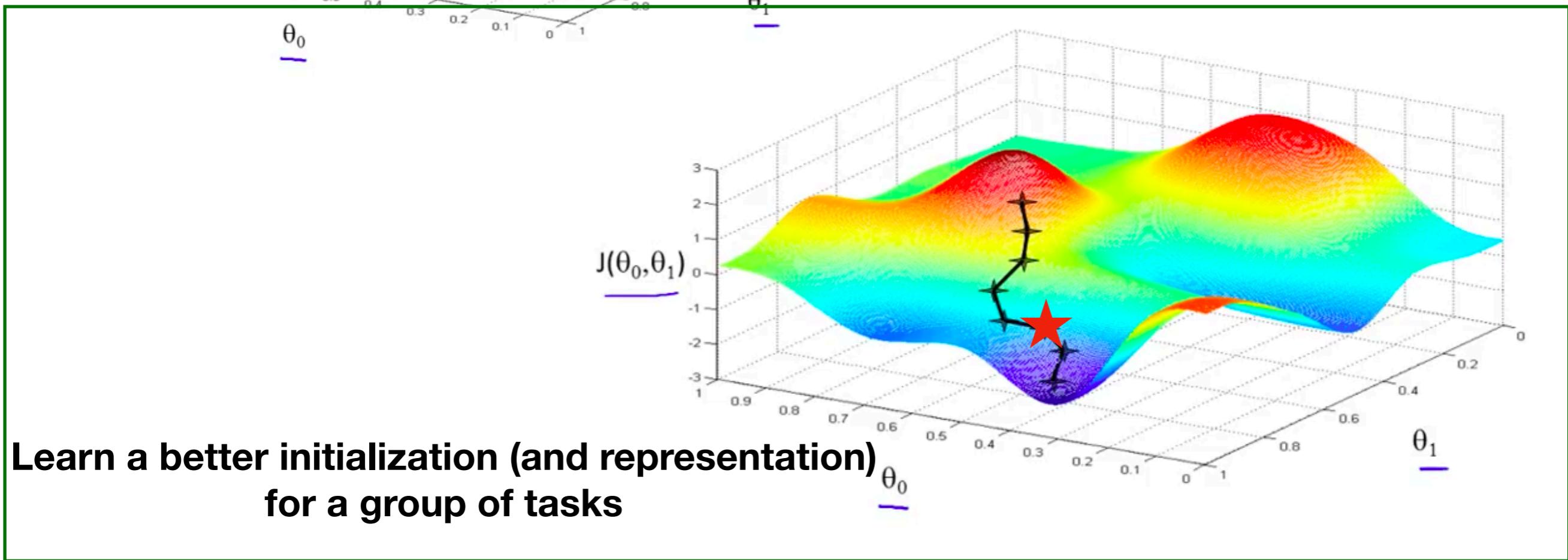
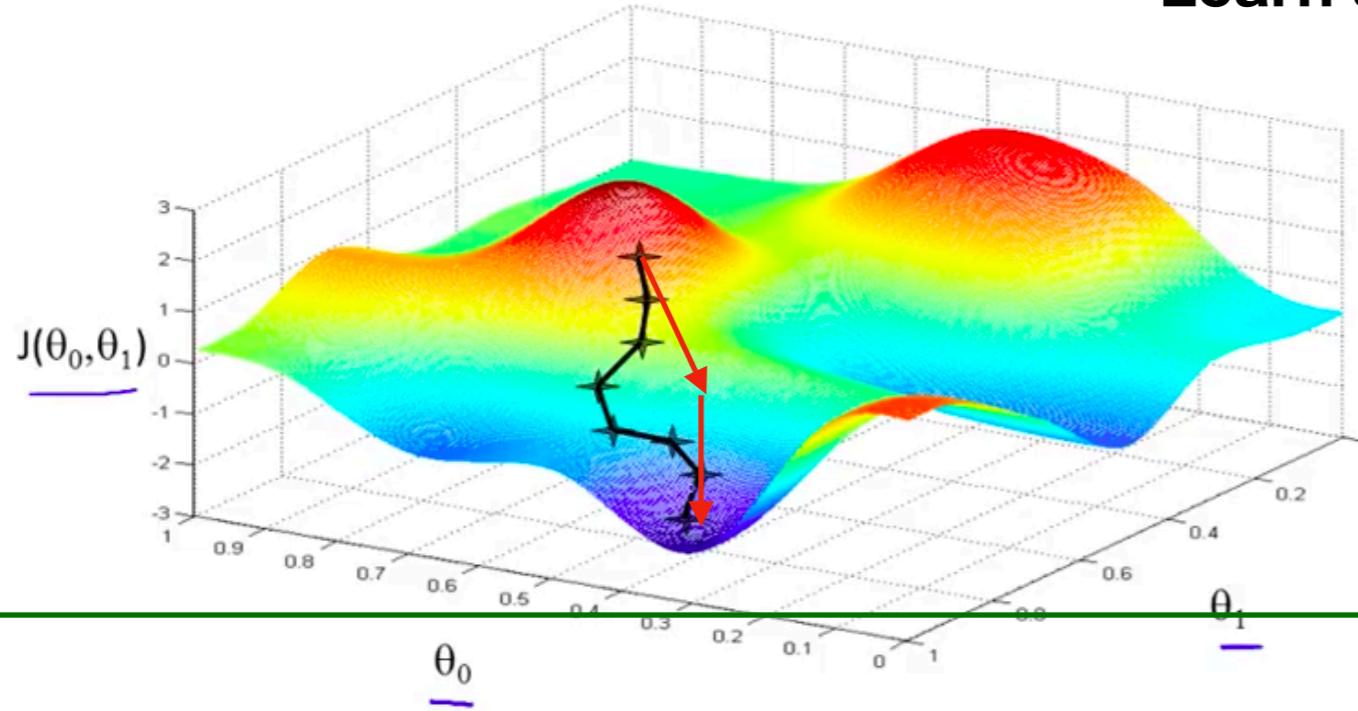
- Hence, training a meta-learner LSTM yields an update rule for training M

- 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



Model transfer (Learning to learn)

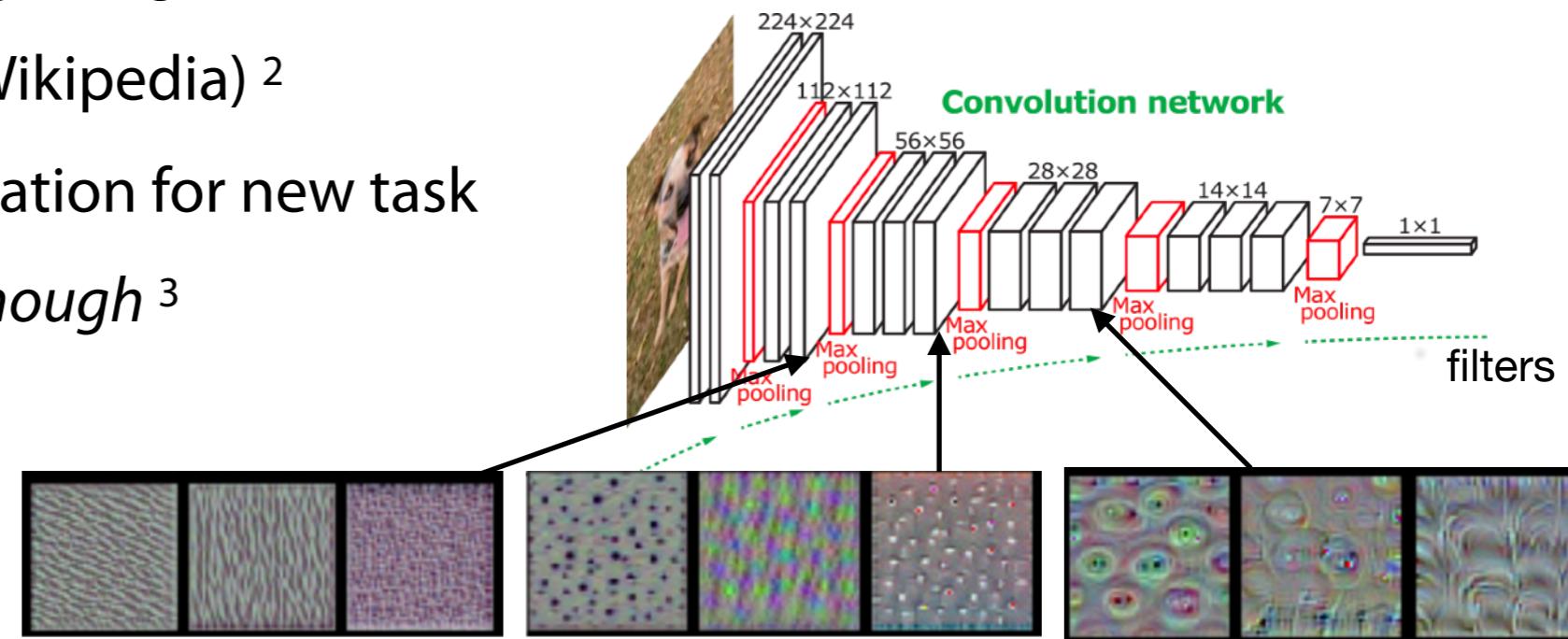
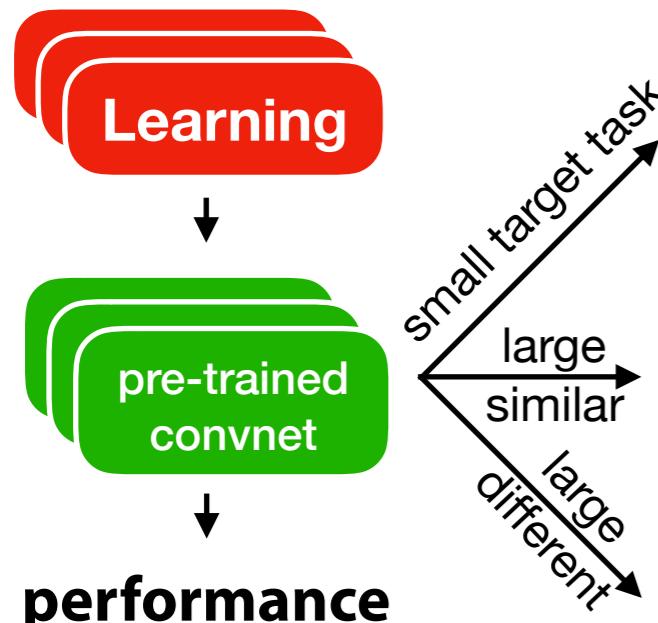
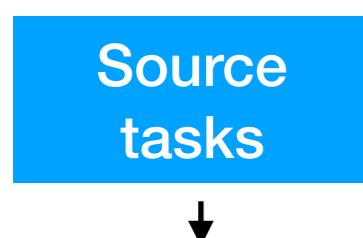
Learn a better gradient update rule
for a group of tasks



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

Transfer learning

- 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



frozen new

Feature extraction:

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

pre-trained new

End-to-end tuning:

train from initialized weights

frozen new

Fine-tuning:

unfreeze last layers, tune on new task

Meta-Learning

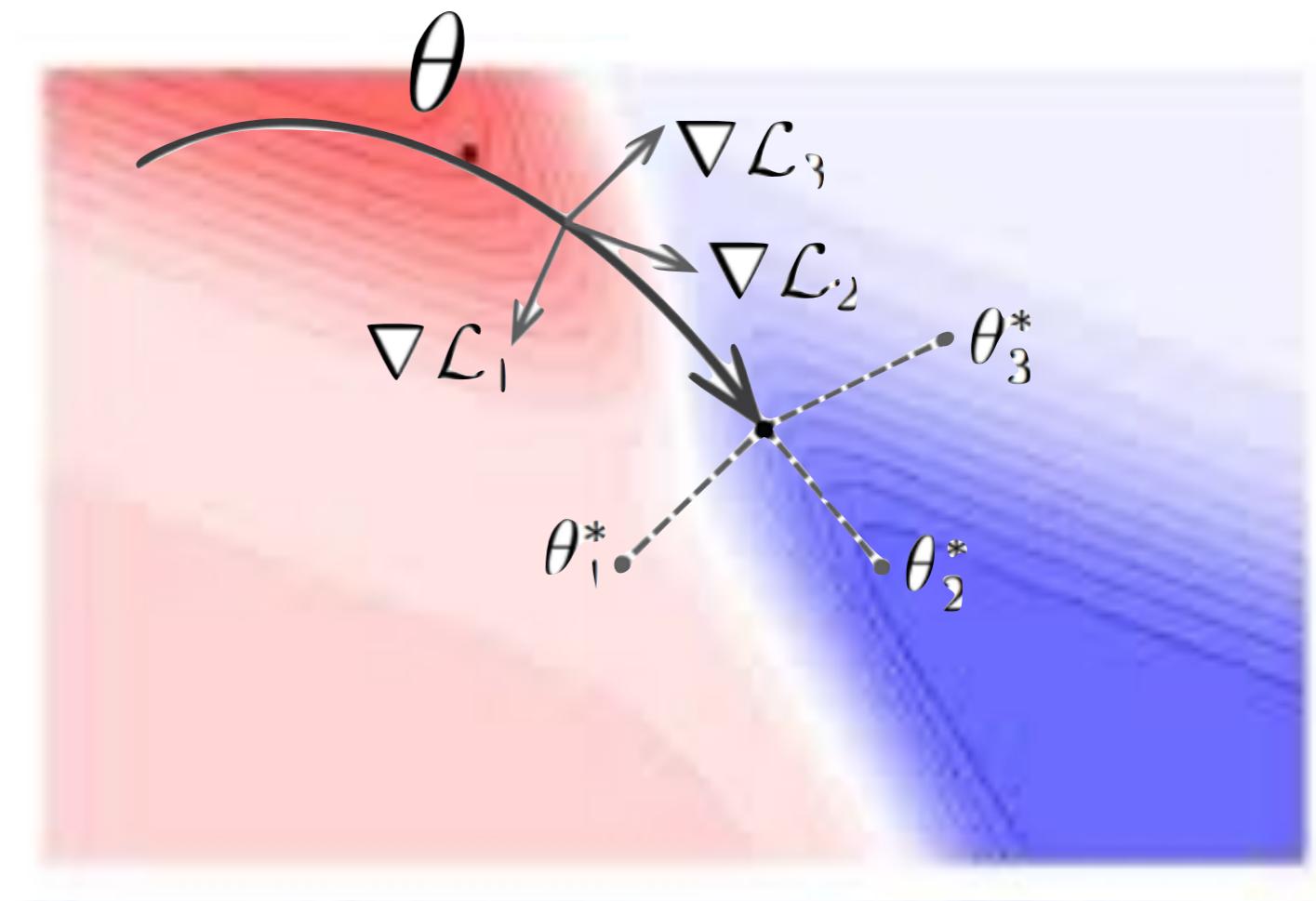
Model-agnostic meta-learning

- 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

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

- Repeat

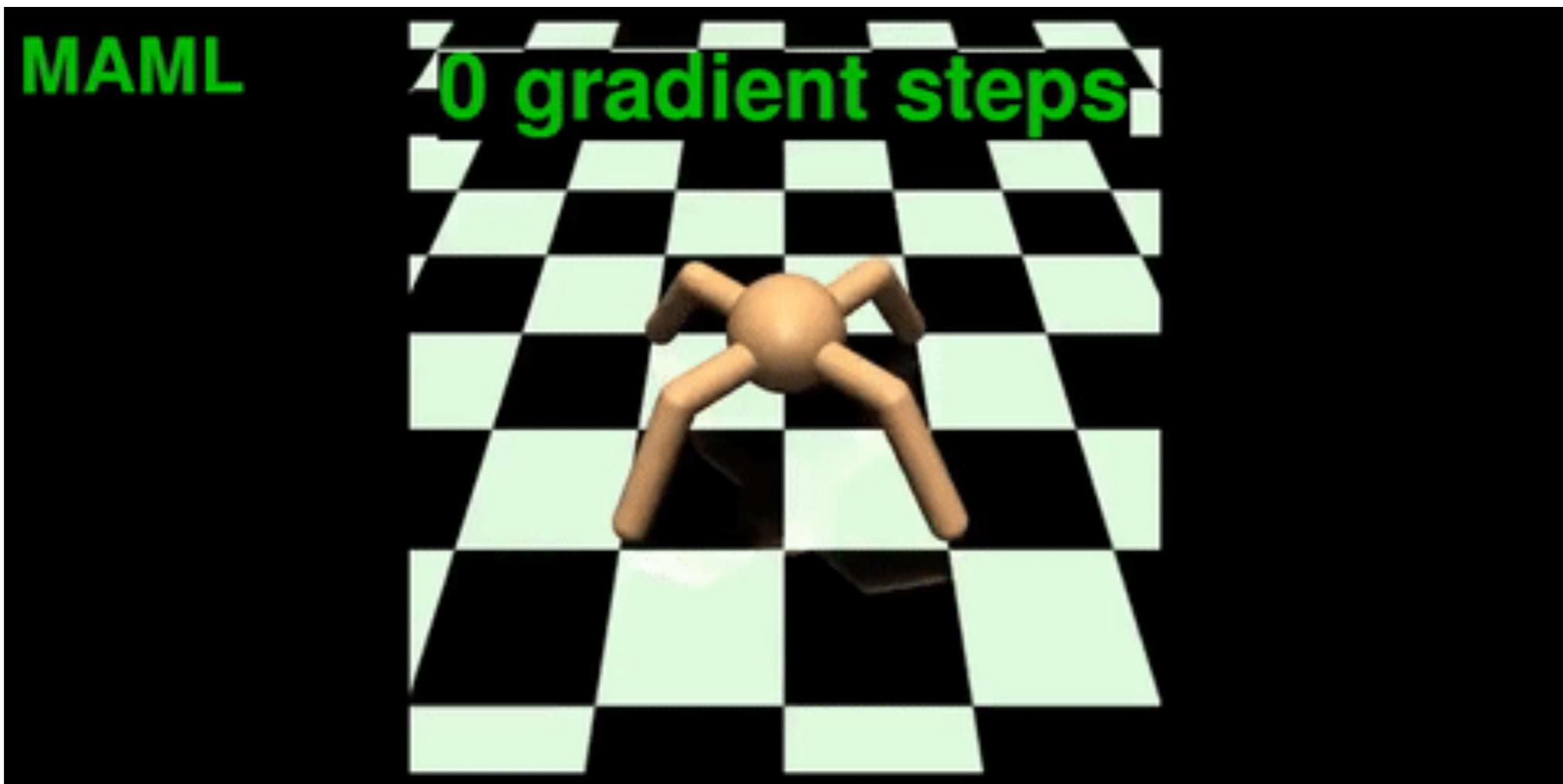


Model-agnostic meta-learning

- 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. weights²
 - 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

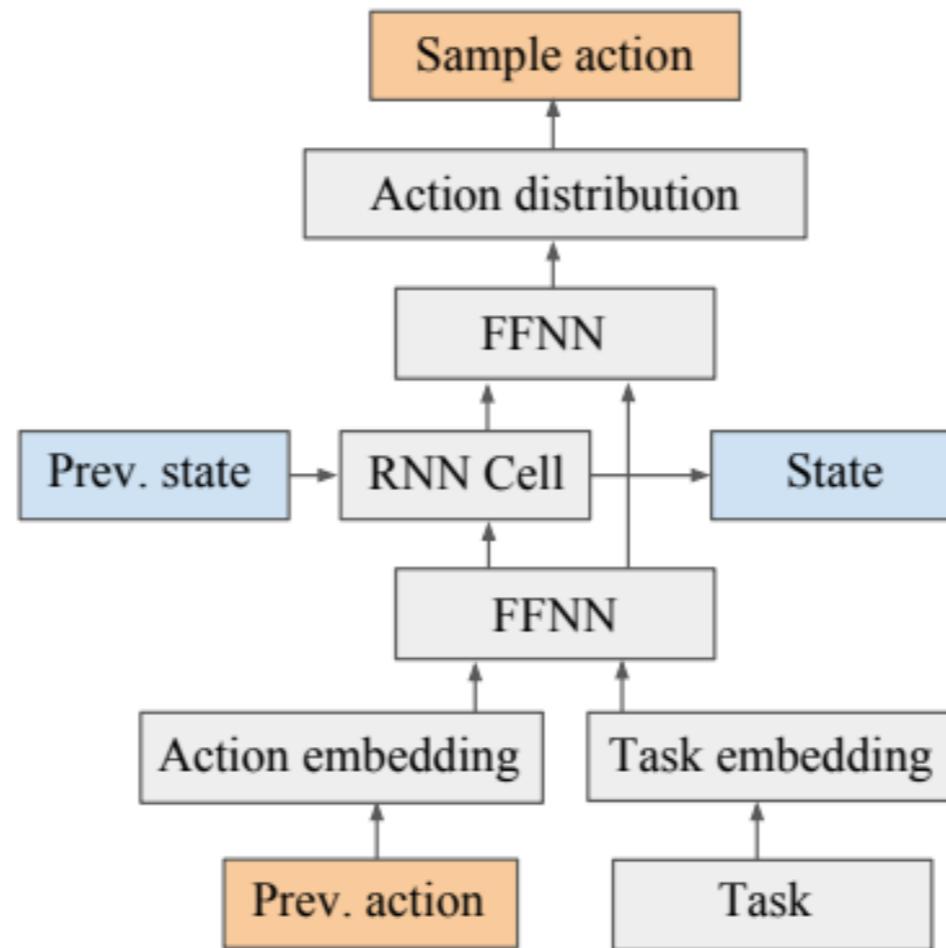
- For reinforcement learning:



Neural Architecture Meta-Learning

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

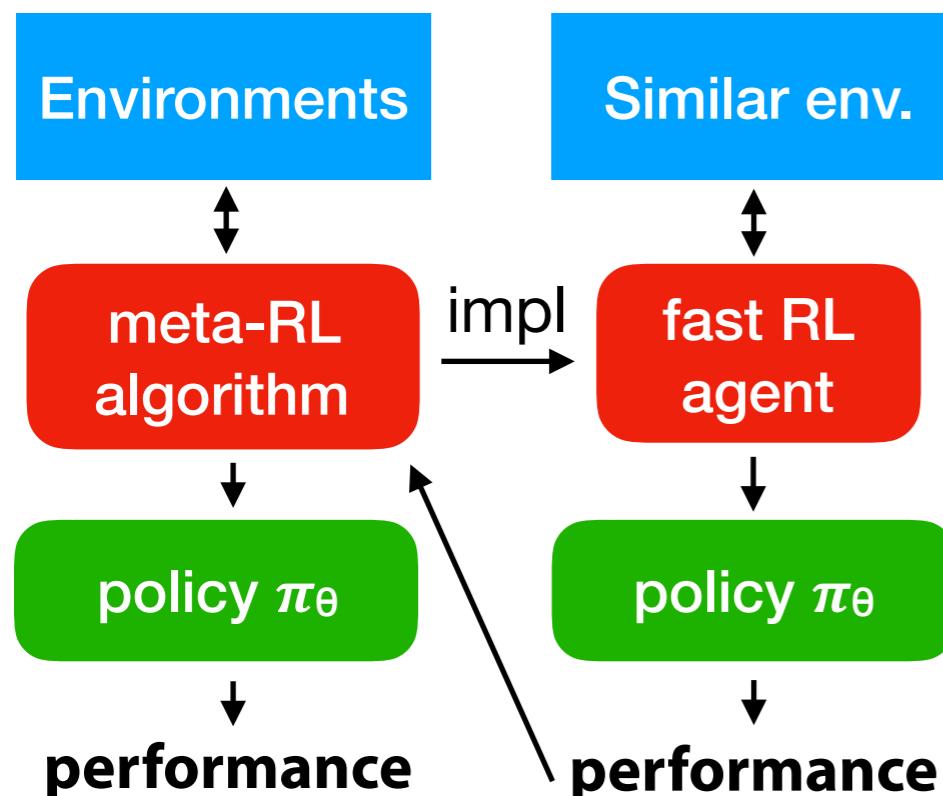
Wong et al. 2018



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

Learning to reinforcement learn

- 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



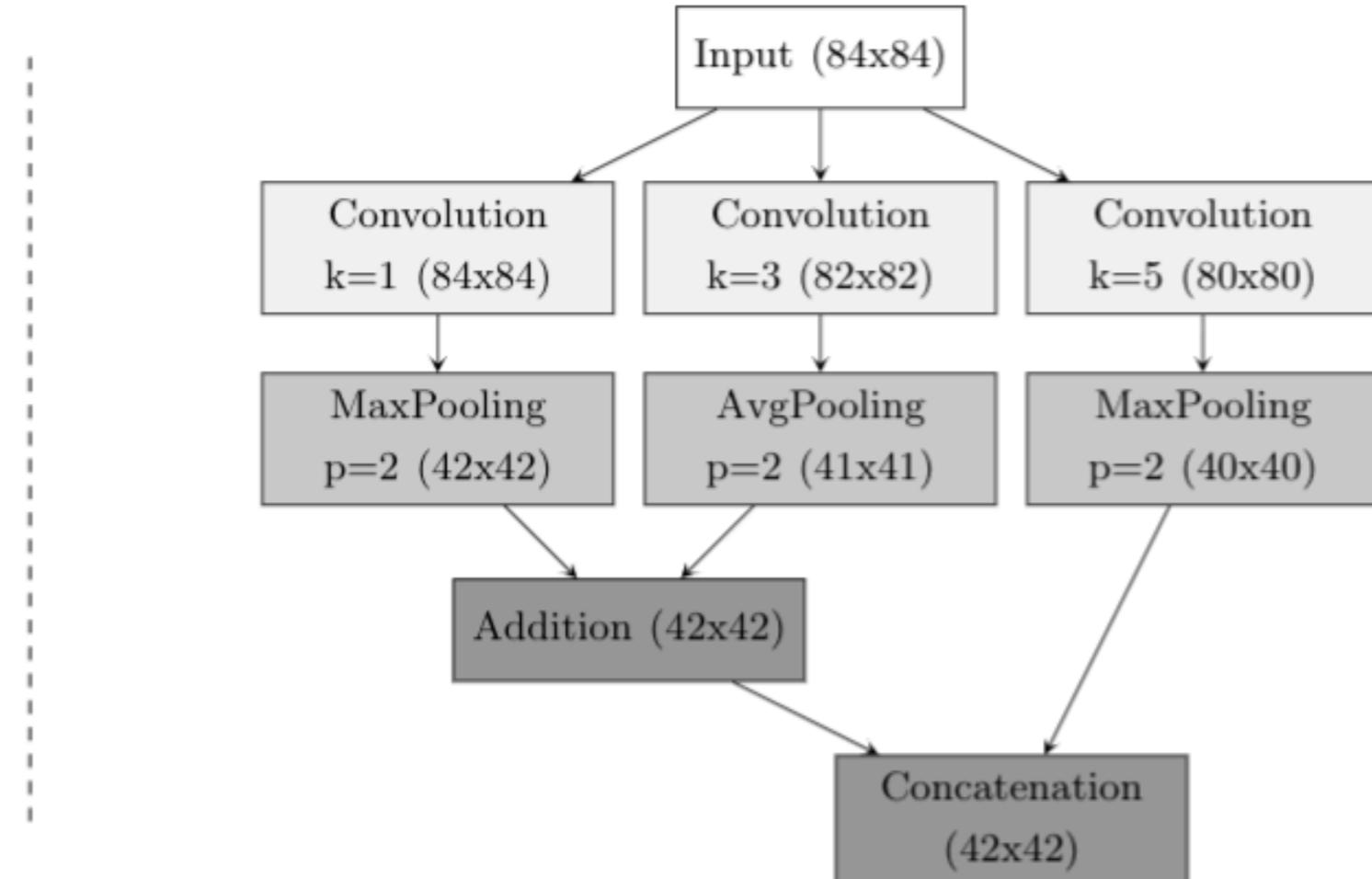
- Learning to reinforcement learn ^{1,2}
 - Use RNN-based deep RL to train a recurrent network on many tasks
 - Learns to implement a 'fast' RL agent, encoded in its weights

Meta-Learning

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]



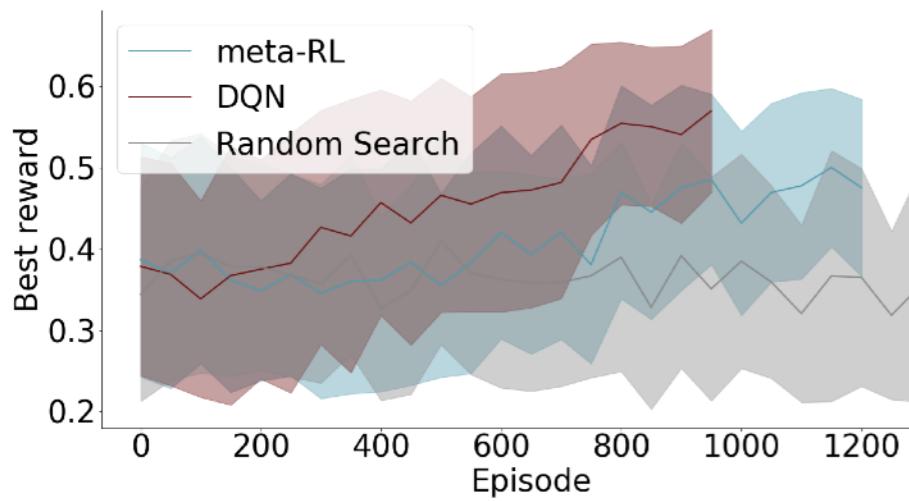
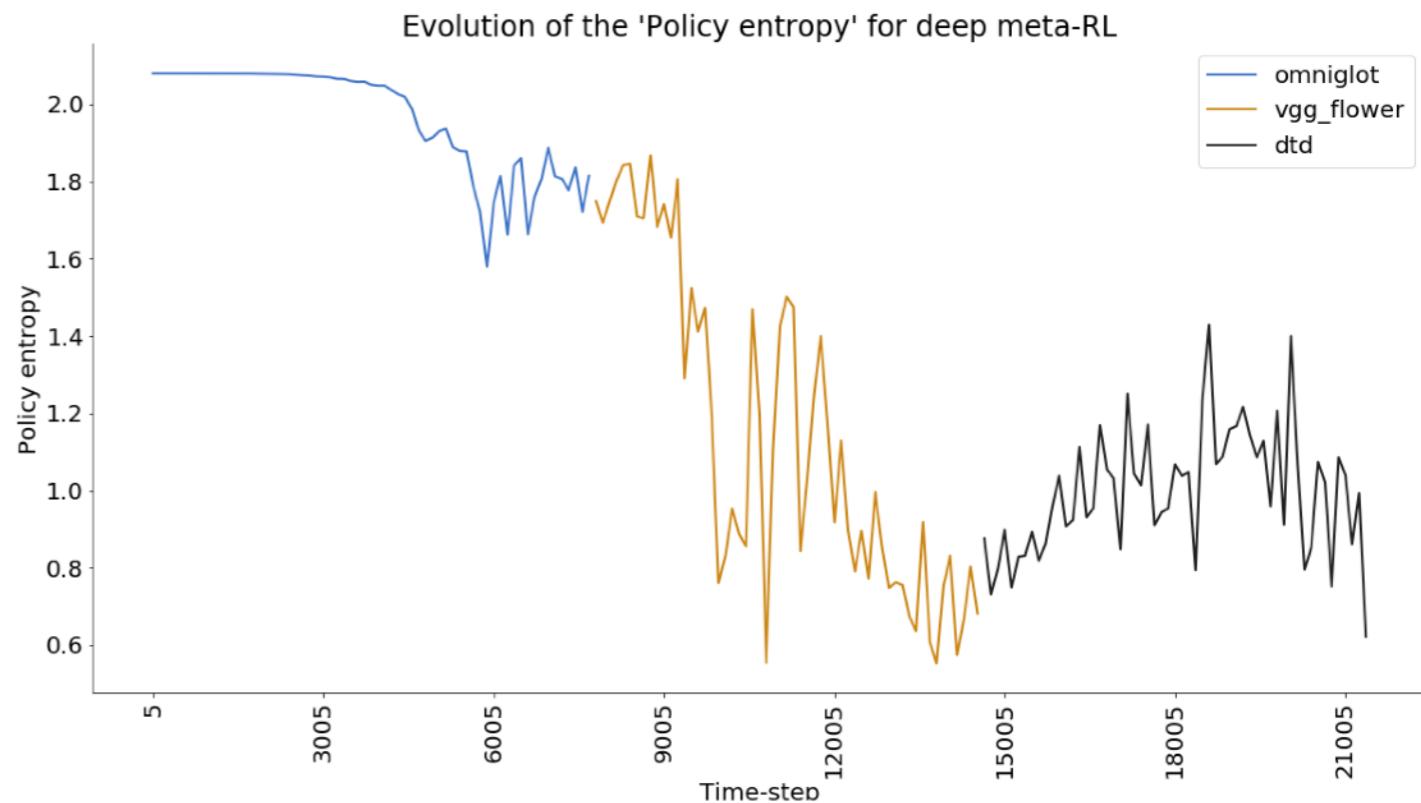
*Actions: add certain layers
in certain locations*

Meta-Learning

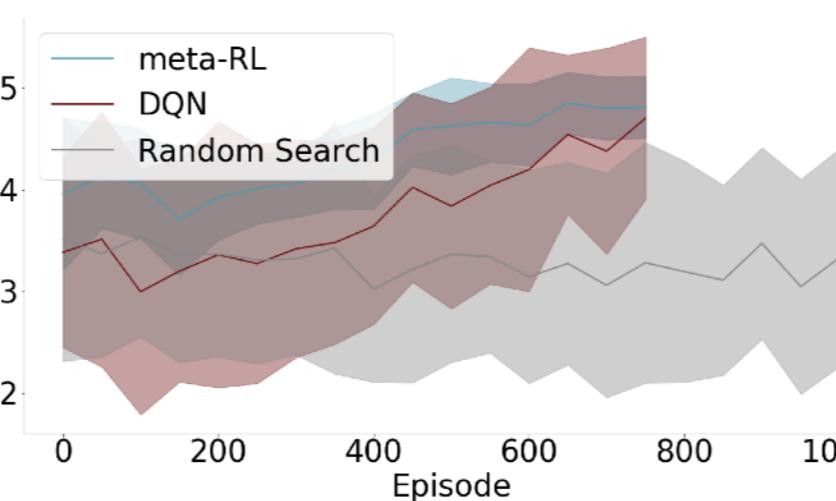
Meta-Reinforcement Learning for NAS

Results on increasingly difficult tasks:

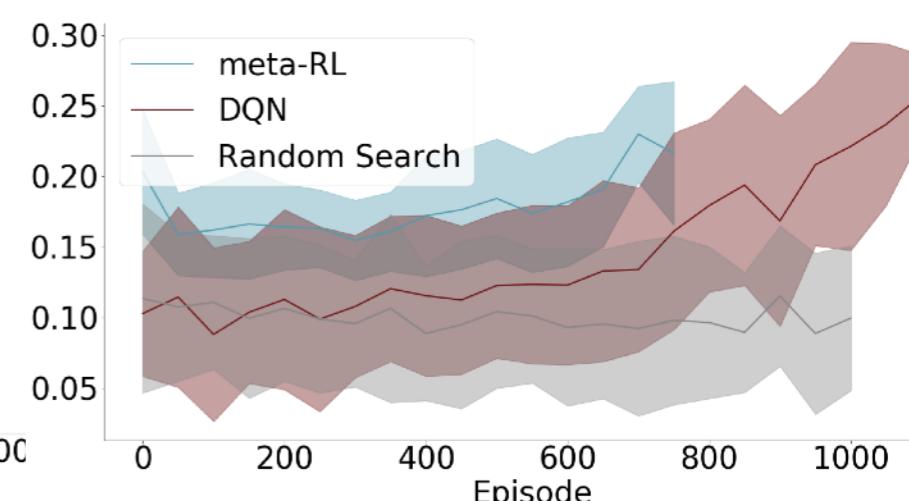
- omniglot
- vgg_flower
- dtd



omniglot



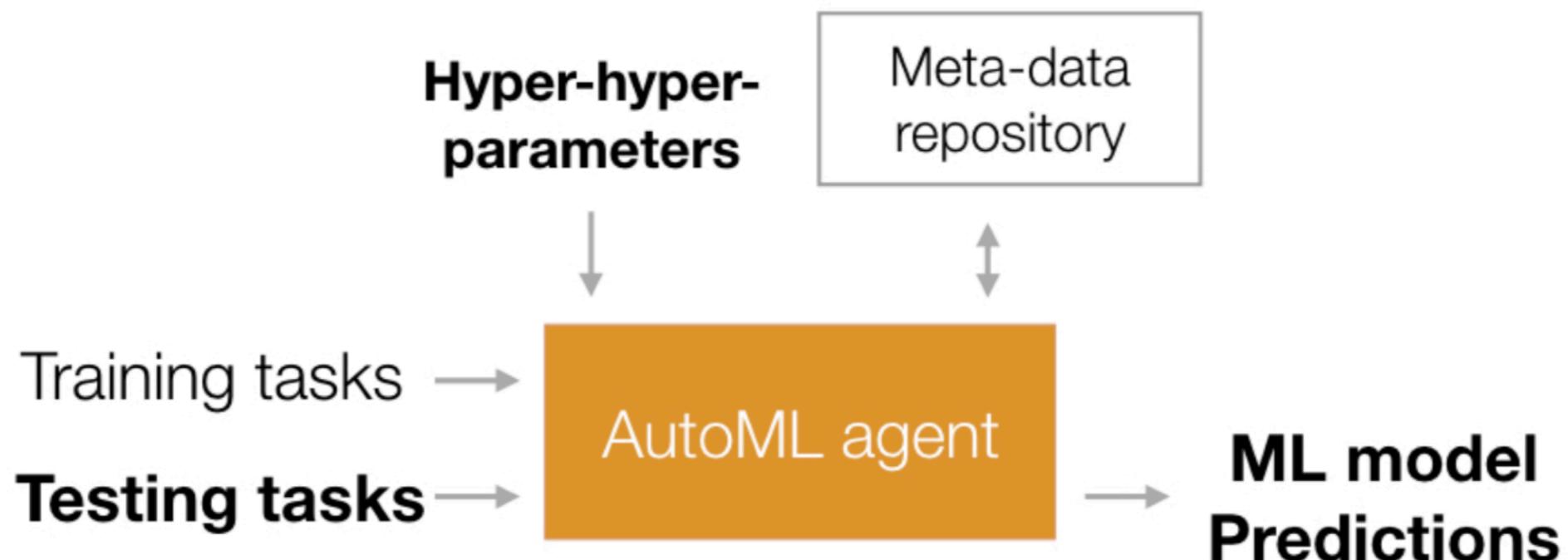
vgg_flower



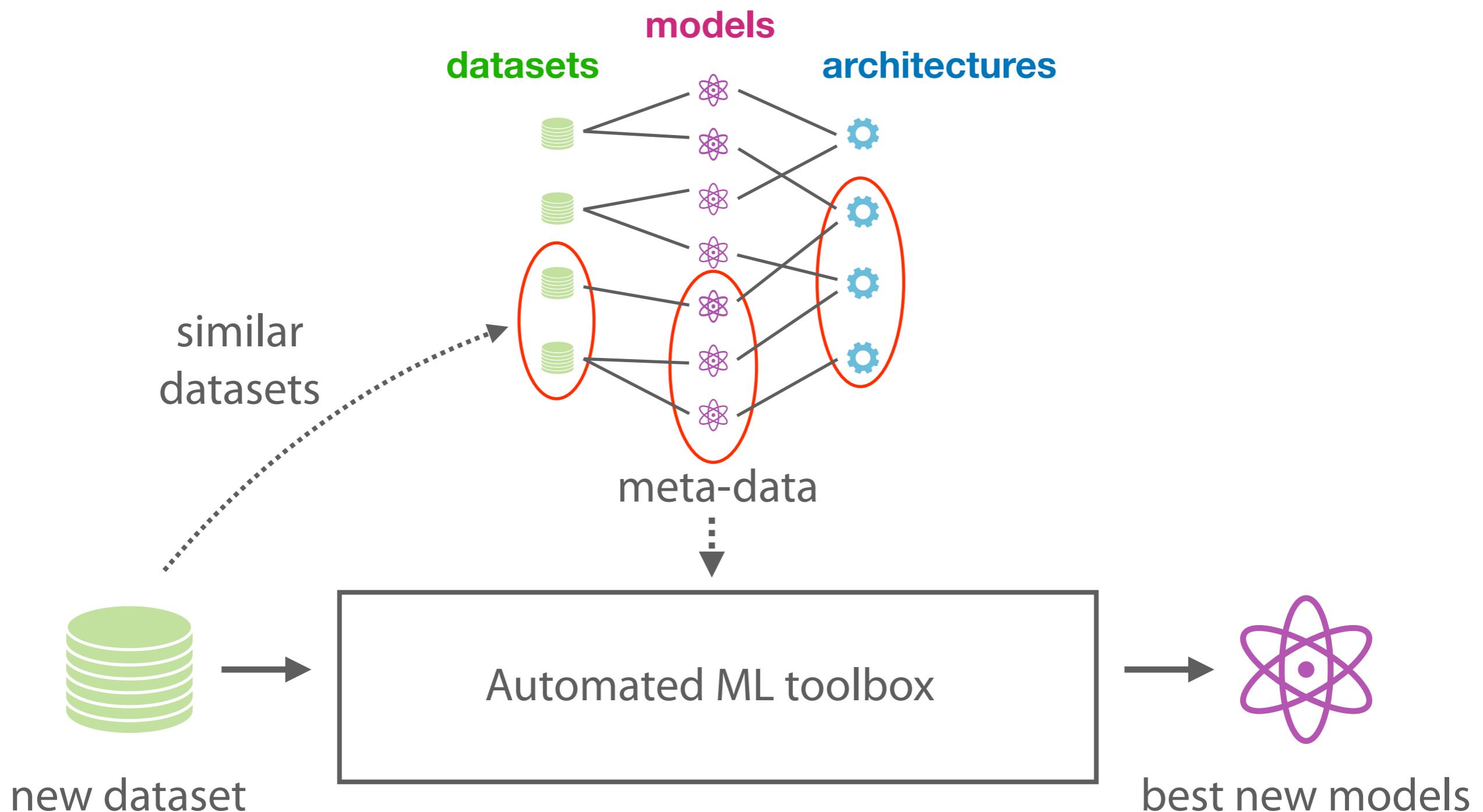
dtd

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

Many other tools for hyperparameter optimization alone

Thank you!

