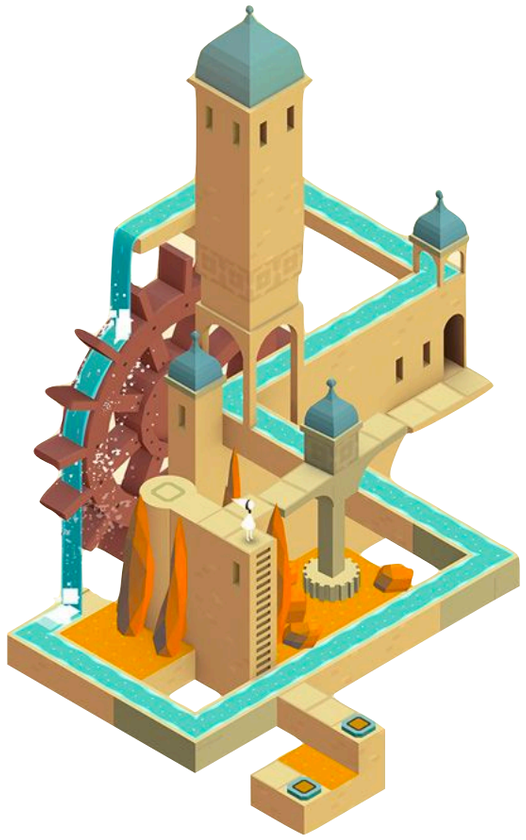
Advanced Course on Data Science & Machine Learning
Siena, 2019



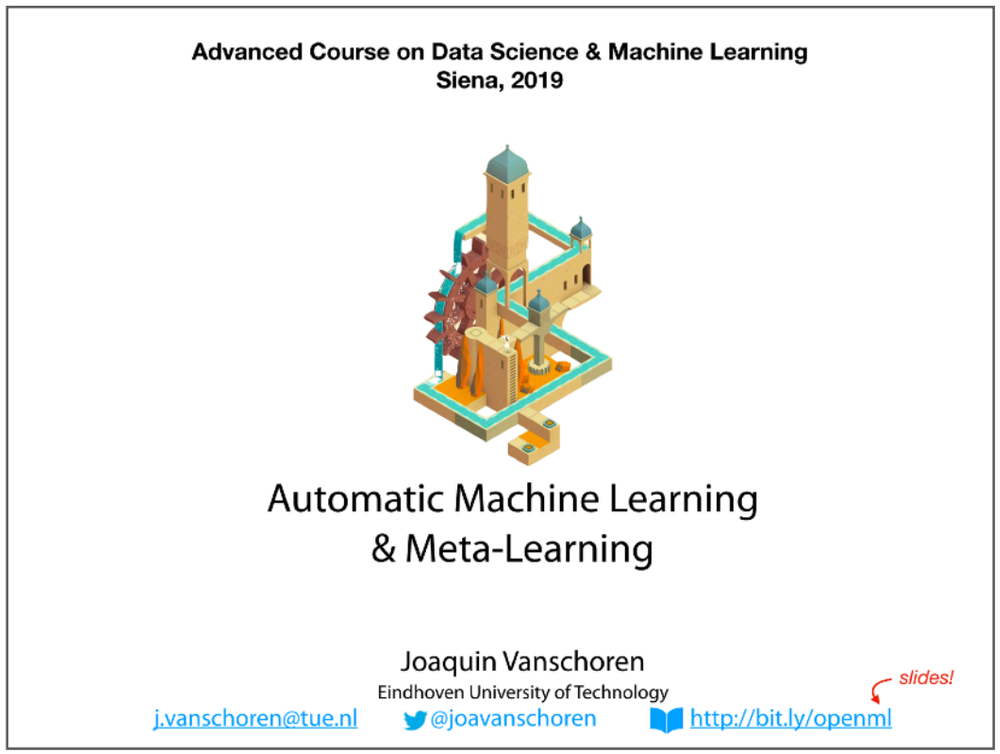
Automatic Machine Learning & Meta-Learning

part 1

Joaquin Vanschoren
Eindhoven University of Technology
j.vanschoren@tue.nl @joavanschoren http://bit.ly/openml

slides!

Slides / Book



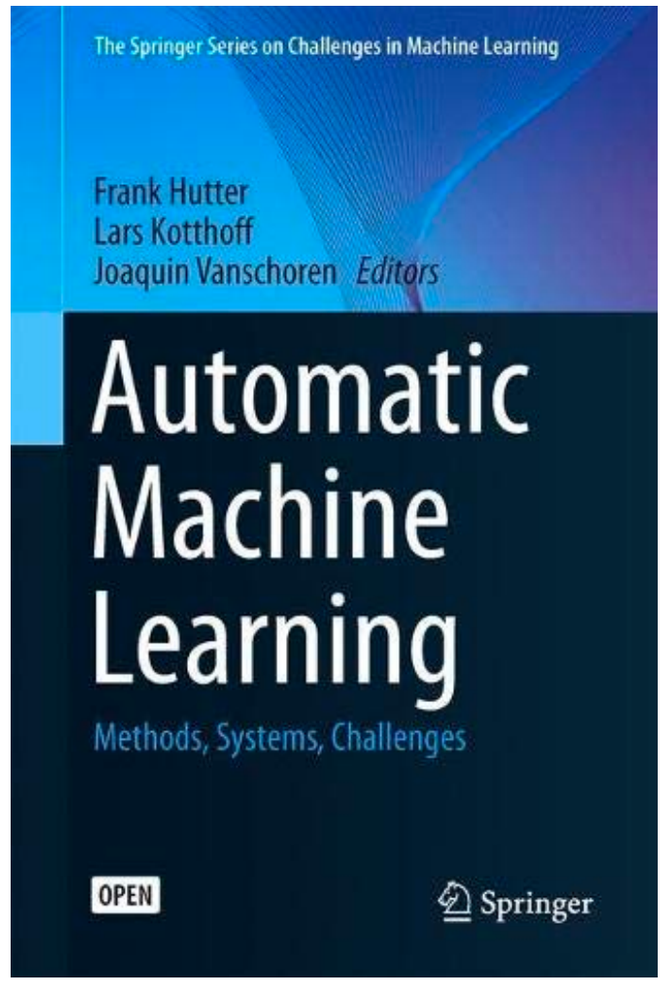
http://bit.ly/openml

More slides:

www.automl.org/events ->
AutoML Tutorial NeurIPS 2018

Video:

www.youtube.com/watch?v=0eBR8a4MQ30

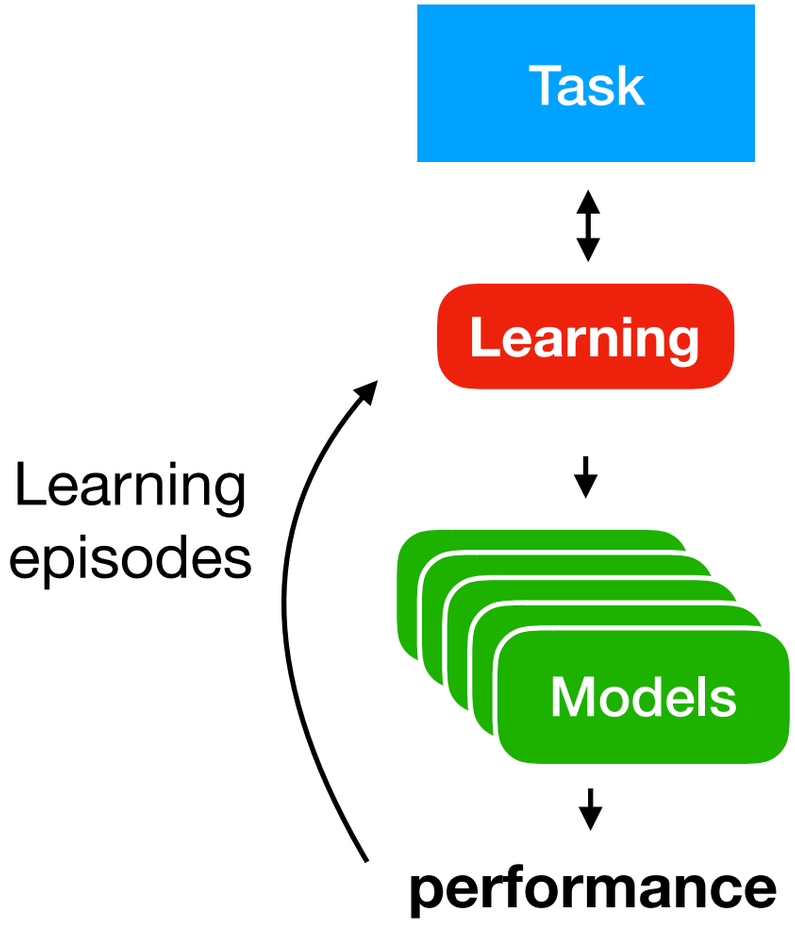


Open access book

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

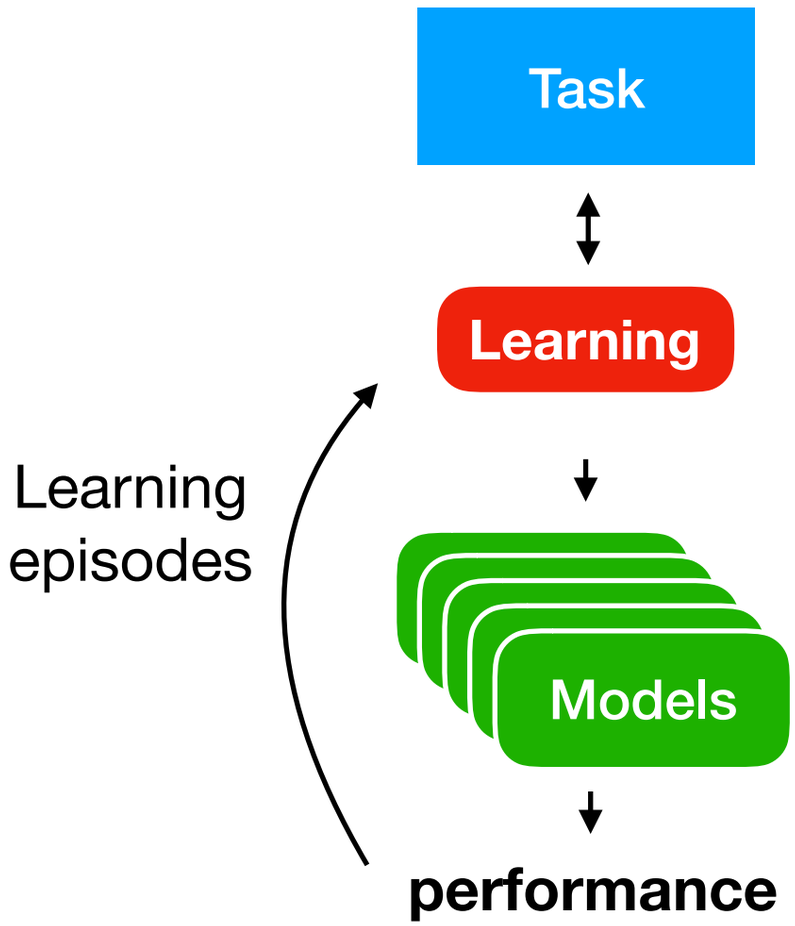
Learning takes experimentation

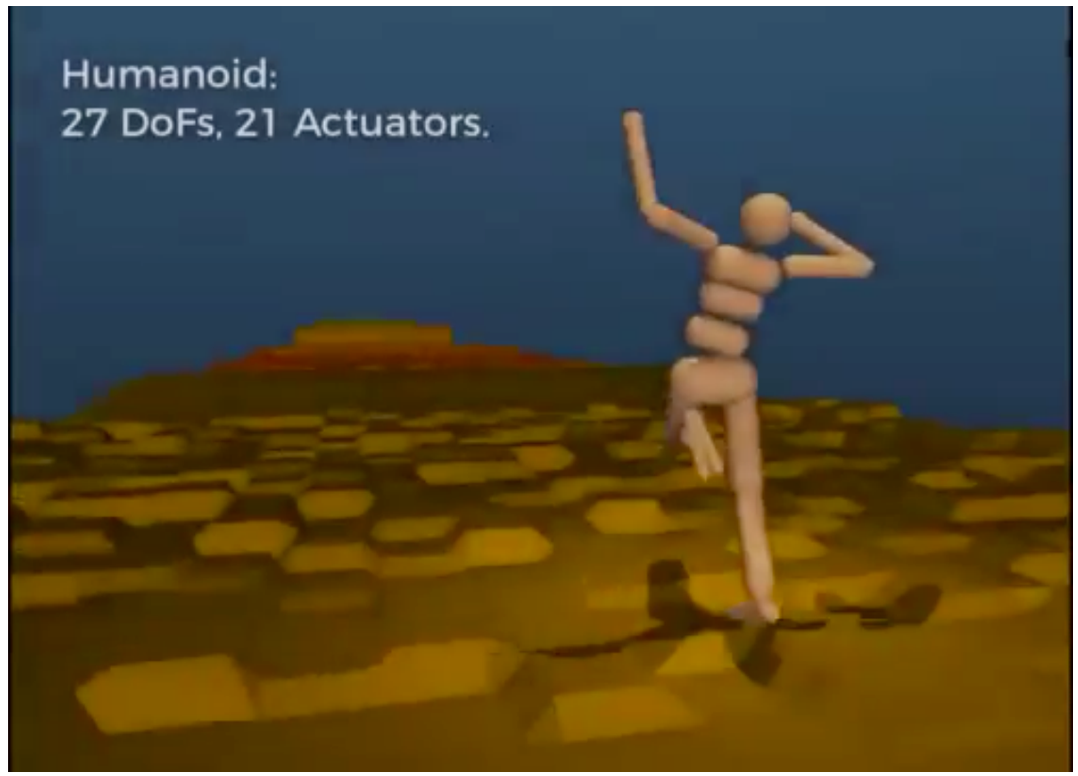
It can take a *lot* of trial and error



Learning takes experimentation

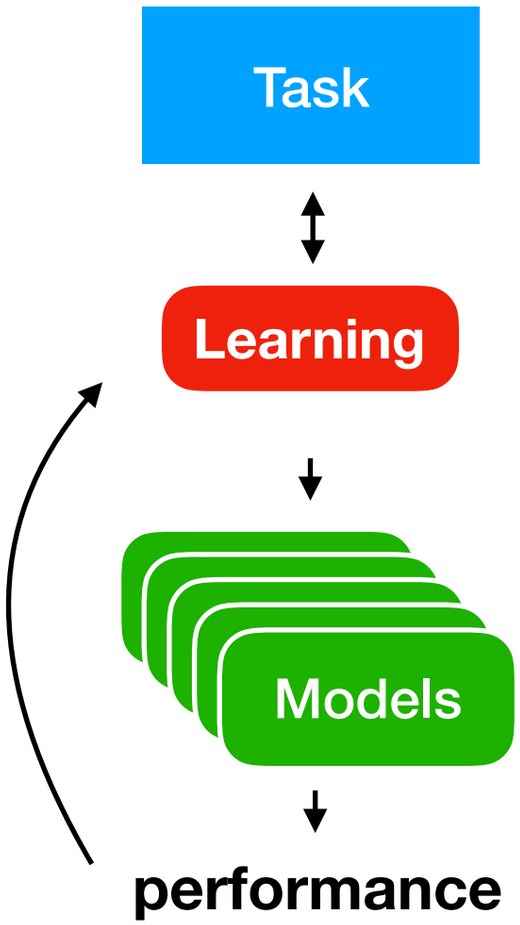
It can take a *lot* of trial and error



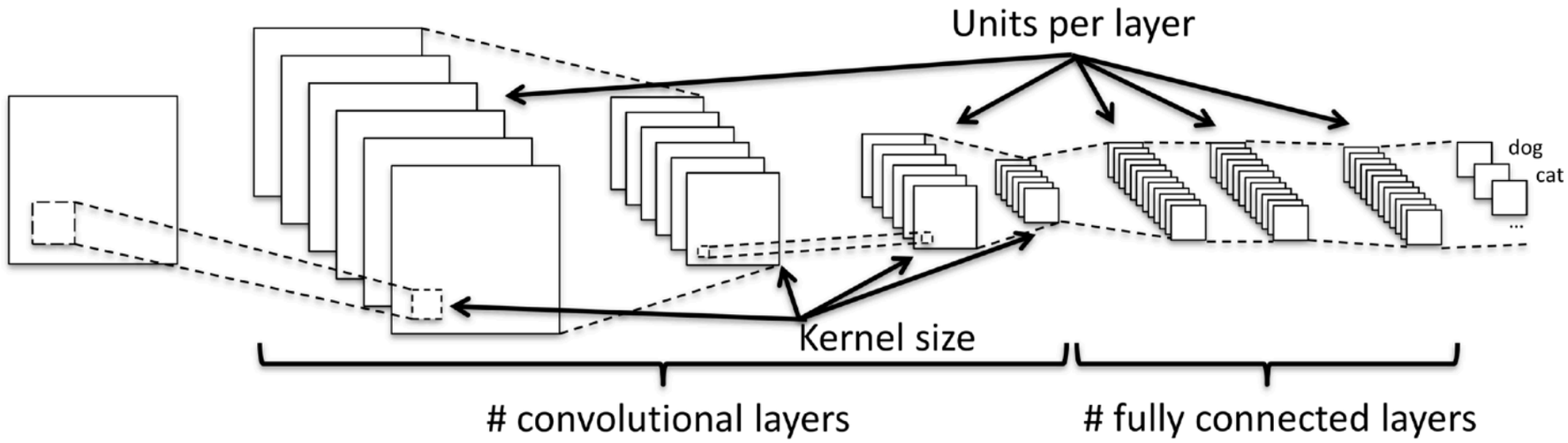


Learning to learn takes experimentation

Building machine learning systems requires a *lot* of expertise and trials



Neural networks



- design architecture (operators/layers)
- many sensitive hyperparameters
  - optimization algorithm, learning rate,…
  - regularization, dropout,…
  - batch size, epochs, early stopping, …
  - data augmentation, …

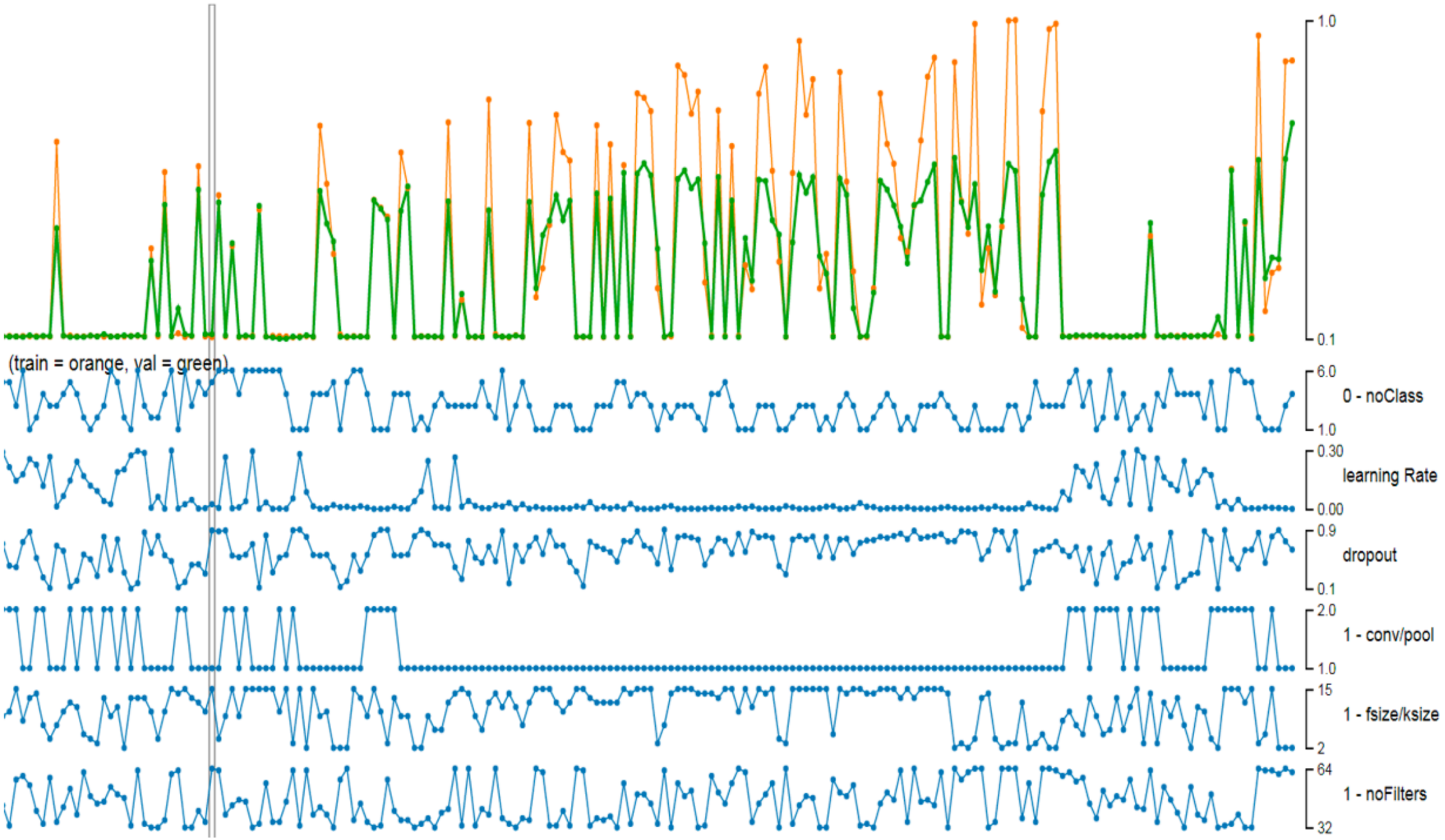
Hyperparameters

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

	Name	Range	Default	log scale	Type	Conditional
Network hyperparameters	batch size	$[32, 4096]$	32	✓	float	-
	number of updates	$[50, 2500]$	200	✓	int	-
	number of layers	$[1, 6]$	1	-	int	-
	learning rate	$[10^{-6}, 1.0]$	10^{-2}	✓	float	-
	L_2 regularization	$[10^{-7}, 10^{-2}]$	10^{-4}	✓	float	-
	dropout output layer	$[0.0, 0.99]$	0.5	✓	float	-
	solver type	{SGD, Momentum, Adam, Adadelta, Adagrad, smorm, Nesterov }	smorm3s	-	cat	-
	lr-policy	{Fixed, Inv, Exp, Step}	fixed	-	cat	-
Conditioned on solver type	β_1	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	β_2	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	ρ	$[0.05, 0.99]$	0.95	✓	float	✓
	momentum	$[0.3, 0.999]$	0.9	✓	float	✓
Conditioned on lr-policy	γ	$[10^{-3}, 10^{-1}]$	10^{-2}	✓	float	✓
	k	$[0.0, 1.0]$	0.5	-	float	✓
	s	$[2, 20]$	2	-	int	✓
Per-layer hyperparameters	activation-type	{Sigmoid, TanH, ScaledTanH, ELU, ReLU, Leaky, Linear}	ReLU	-	cat	✓
	number of units	$[64, 4096]$	128	✓	int	✓
	dropout in layer	$[0.0, 0.99]$	0.5	-	float	✓
	weight initialization	{Constant, Normal, Uniform, Glorot-Uniform, Glorot-Normal, He-Normal, He-Uniform, Orthogonal, Sparse}	He-Normal	-	cat	✓
	std. normal init.	$[10^{-7}, 0.1]$	0.0005	-	float	✓
	leakiness	$[0.01, 0.99]$	$\frac{1}{3}$	-	float	✓
	tanh scale in	$[0.5, 1.0]$	$2/3$	-	float	✓
	tanh scale out	$[1.1, 3.0]$	1.7159	✓	float	✓

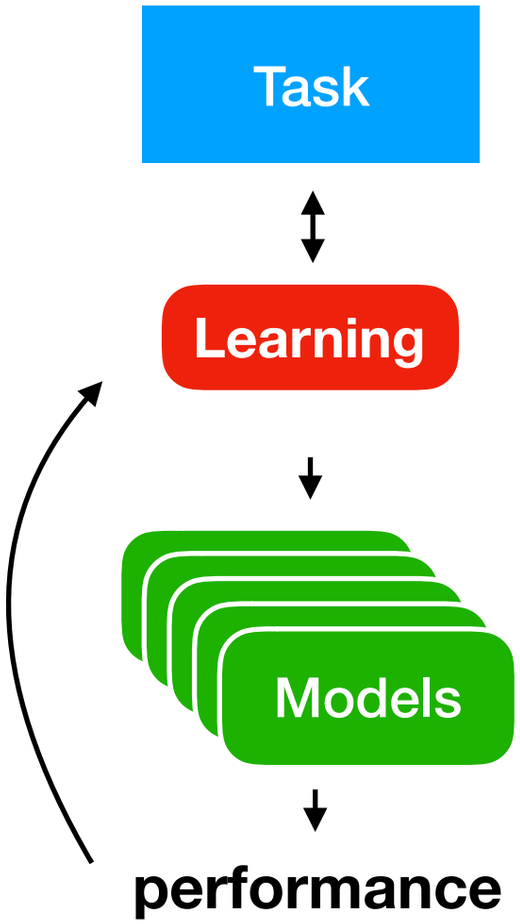
Hyperparameters

Can be very sensitive

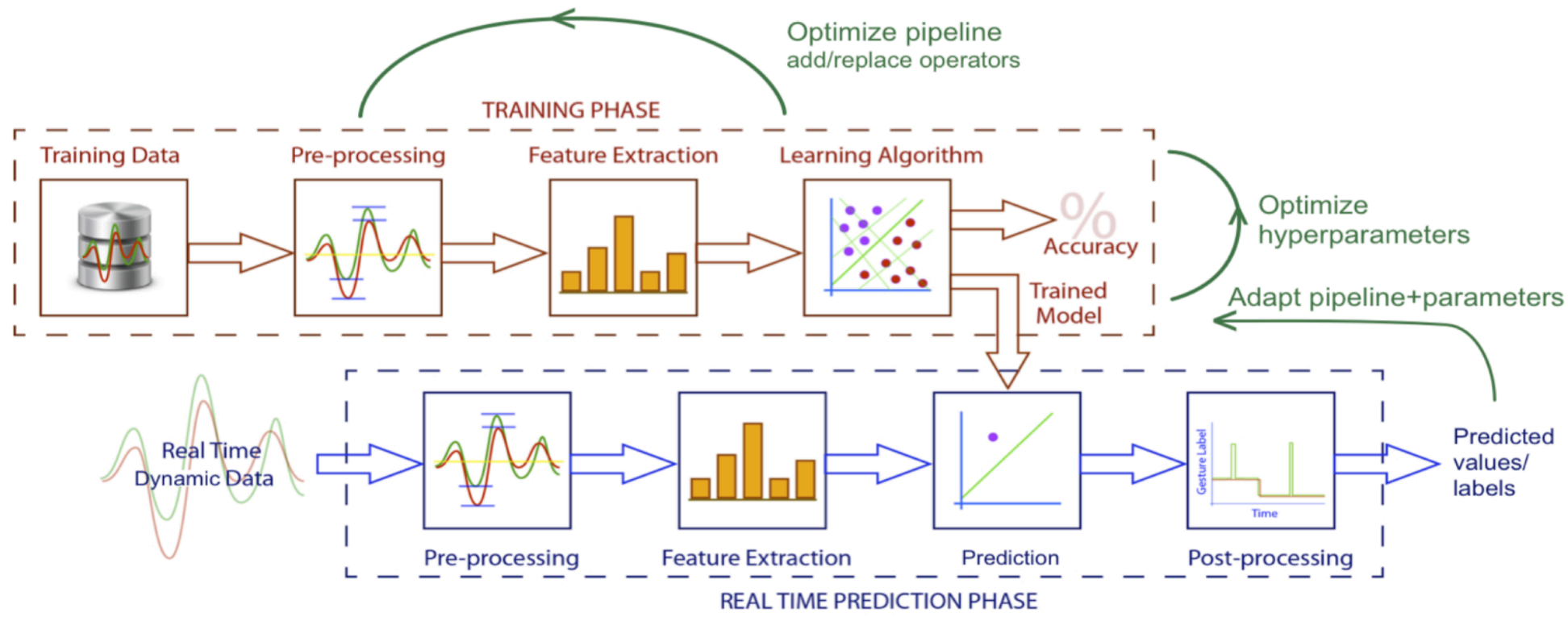


Learning to learn takes experimentation

Building machine learning systems requires a lot of expertise and trials



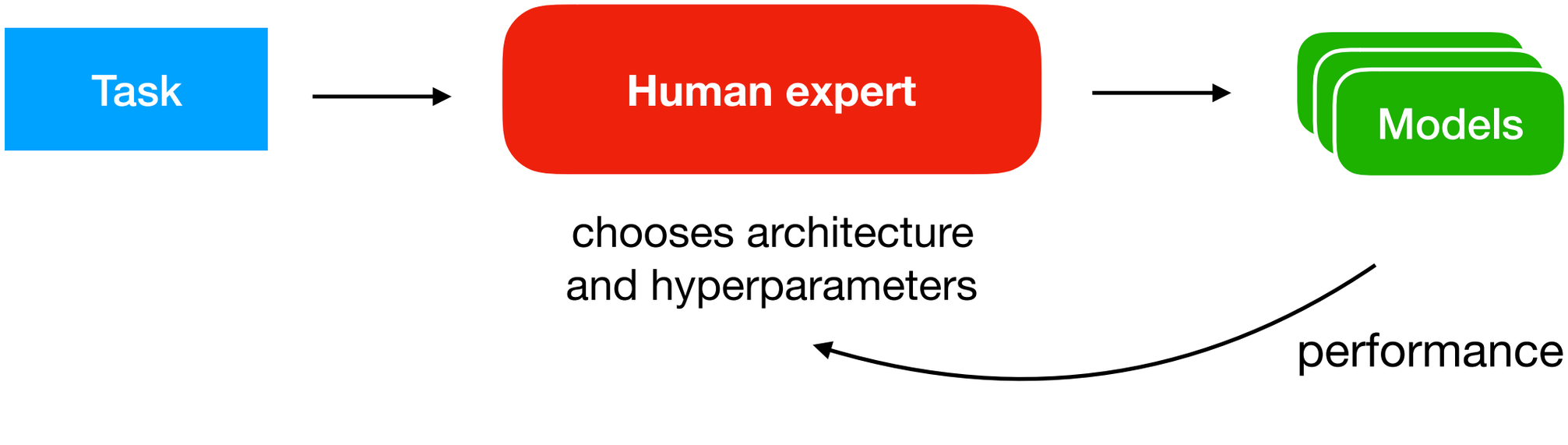
Machine learning pipelines



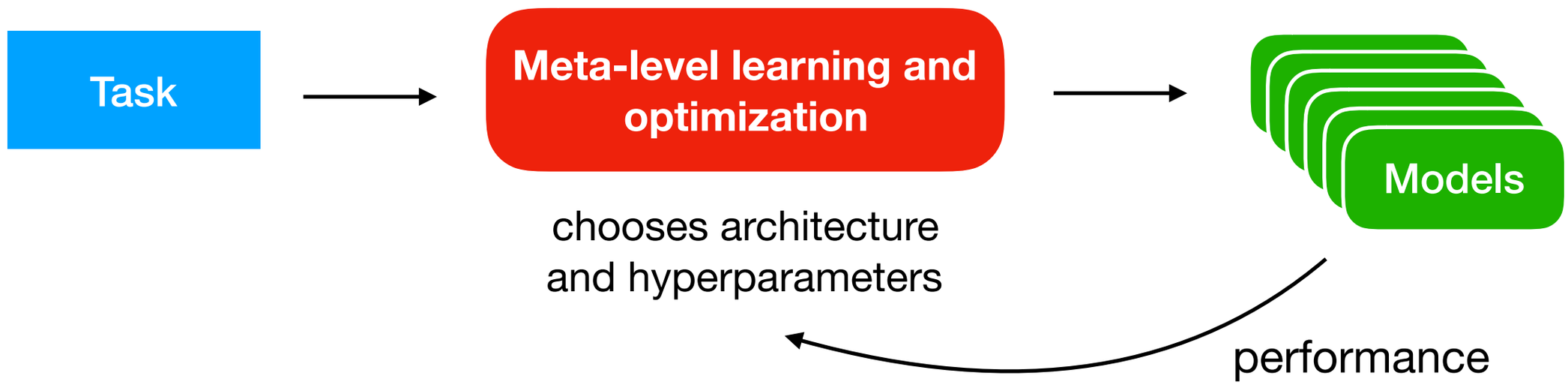
- design pipeline architecture (operators)
- clean, preprocess data (!)
- select and/or engineer features
- select and tune models
- adjust to evolving input data (concept drift),…

Automated machine learning

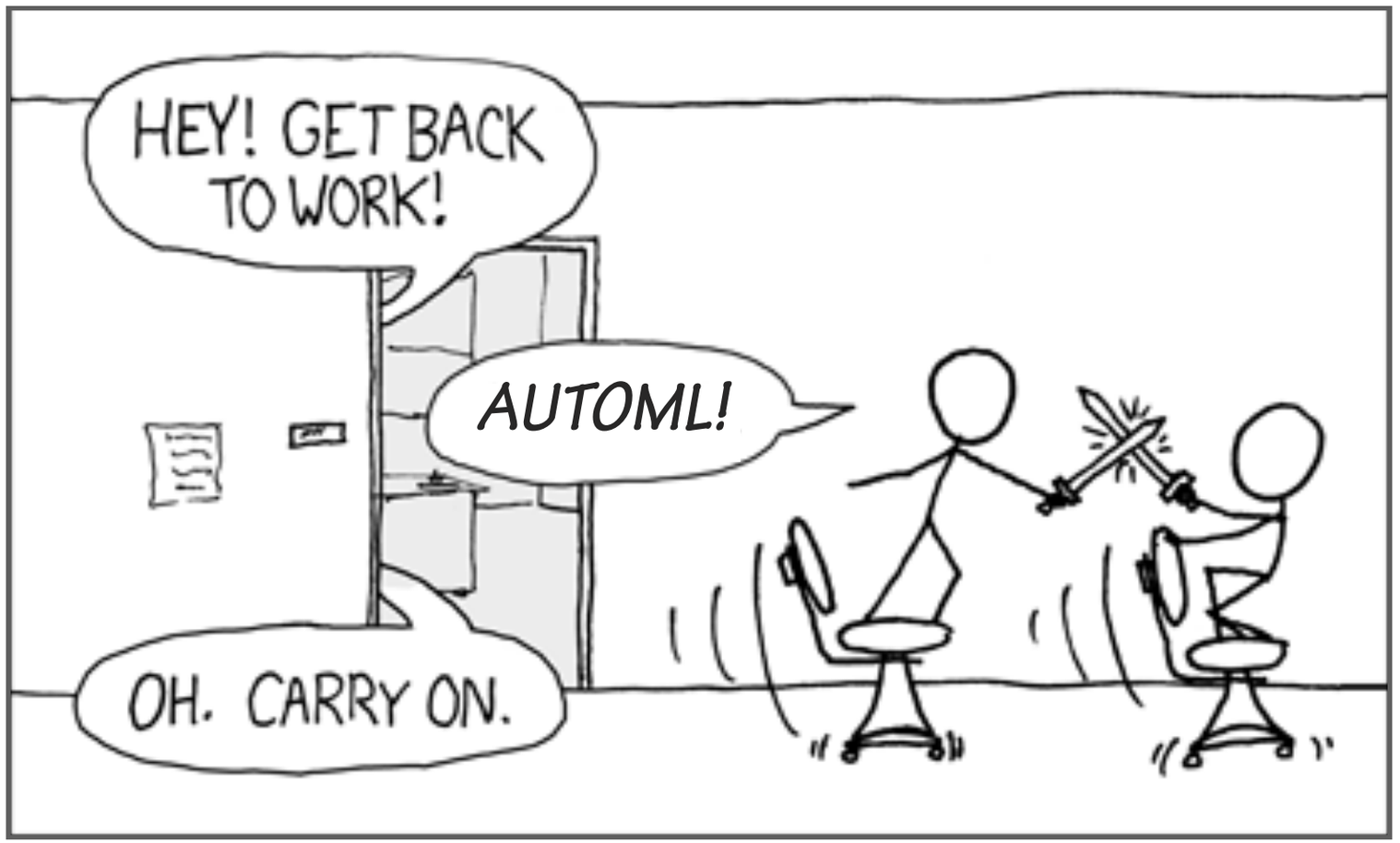
Current practice



Replace manual model building by automation



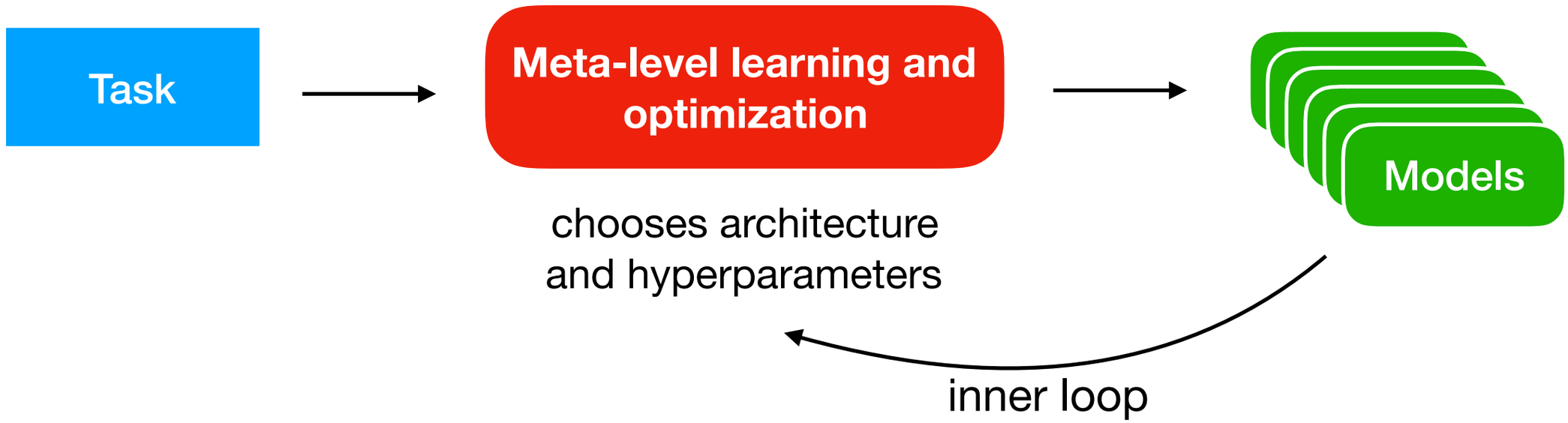
THE DATA SCIENTIST'S #1 EXCUSE FOR LEGITIMATELY SLACKING OFF: "THE AUTOML TOOL IS OPTIMIZING MY MODELS!"



Sutton 2019

Human-in-the-loop AutoML (semi-AutoML)

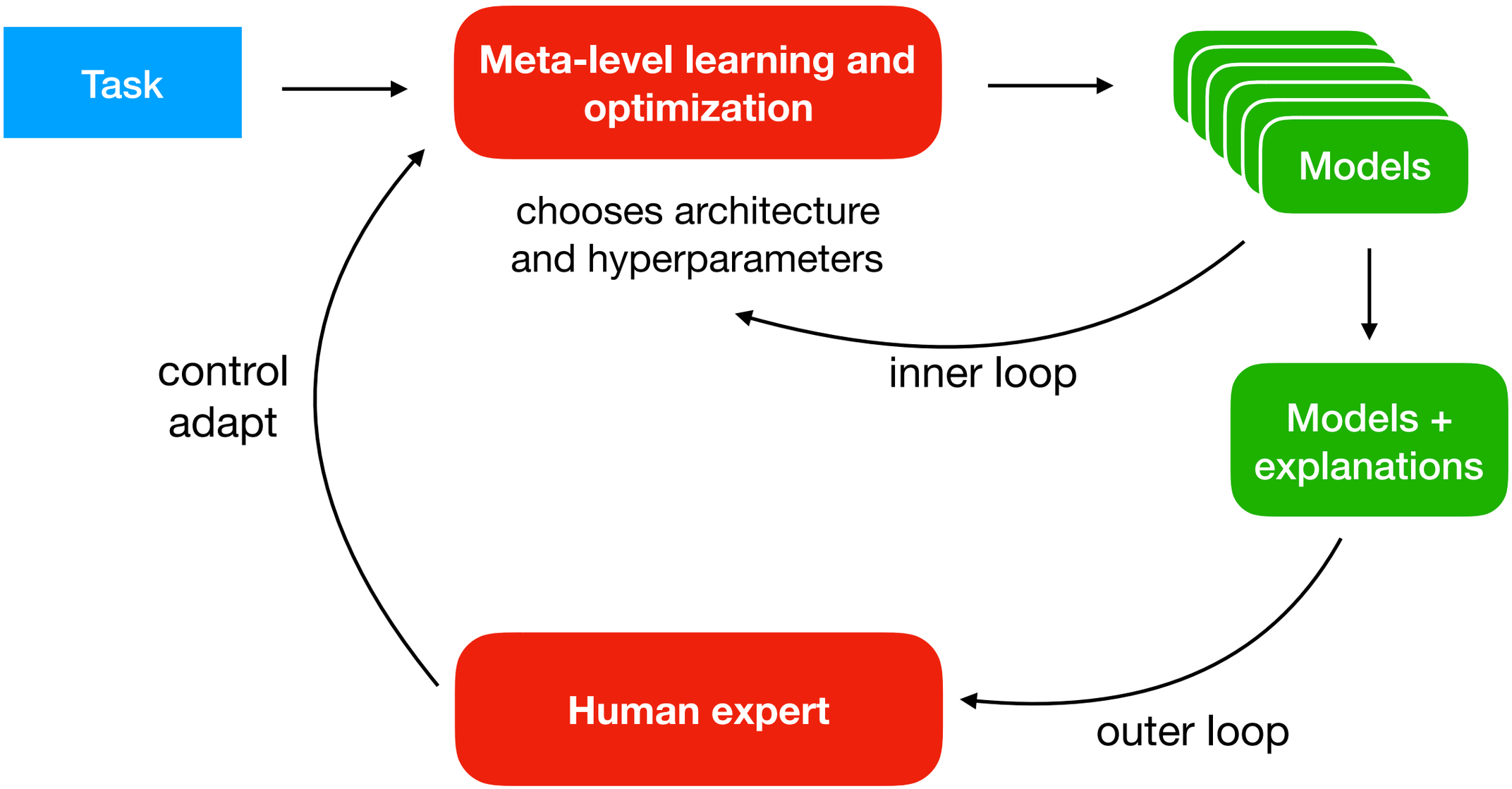
Domain knowledge and human expertise are very valuable
e.g. unknown unknowns, preference learning,…



Sutton 2019

Human-in-the-loop AutoML (semi-AutoML)

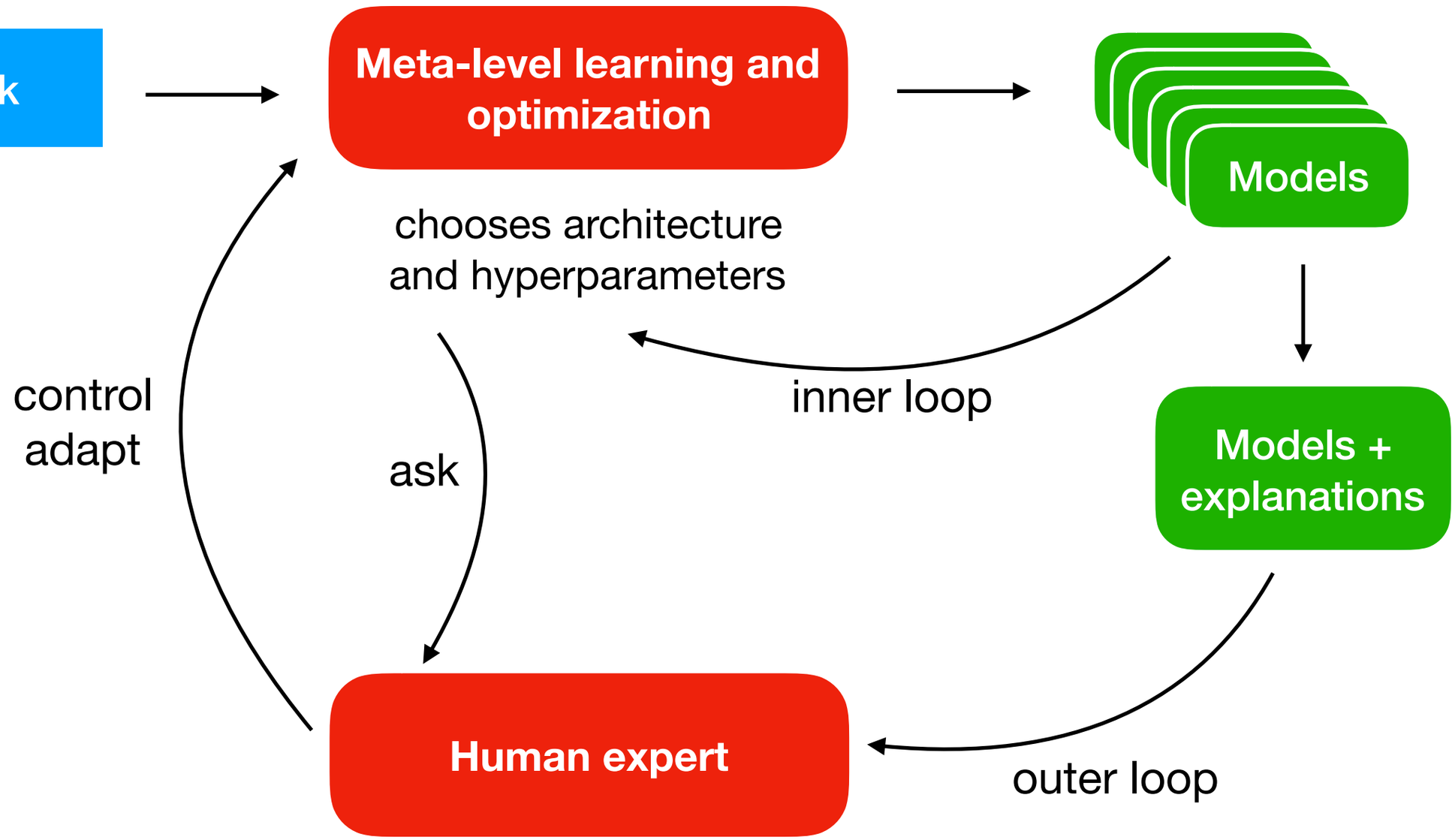
Domain knowledge and human expertise are very valuable
e.g. unknown unknowns, preference learning,…



Sutton 2019

Human-in-the-loop AutoML (semi-AutoML)

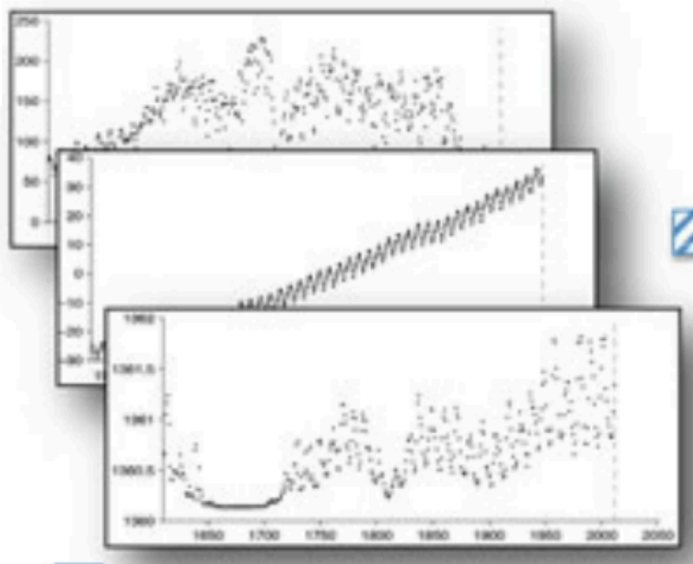
Domain knowledge and human expertise are very valuable
e.g. unknown unknowns, preference learning,…



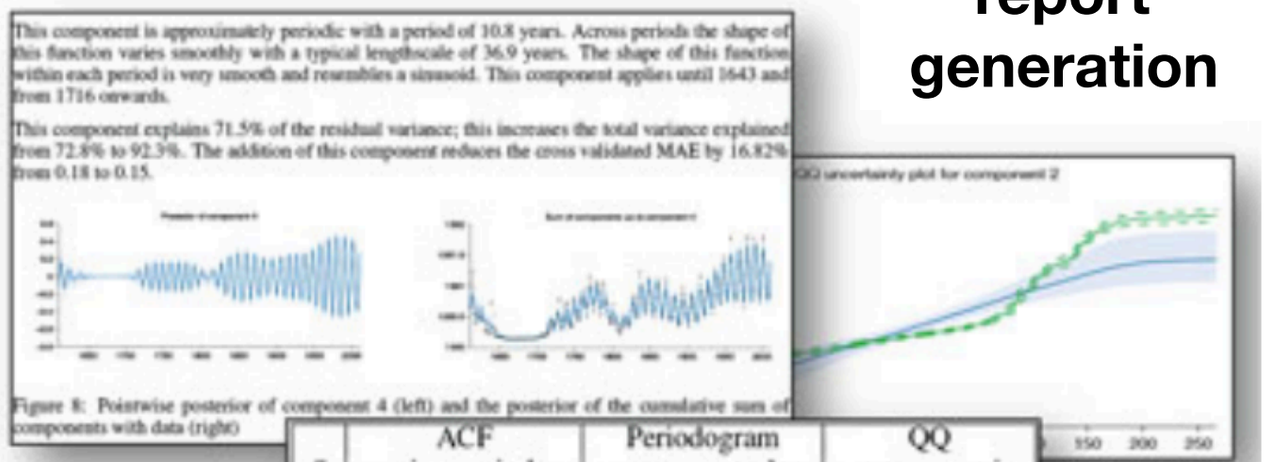
Human-in-the-loop AutoML (semi-AutoML)

Explain model in human language: *automatic statistician*

data input



report generation

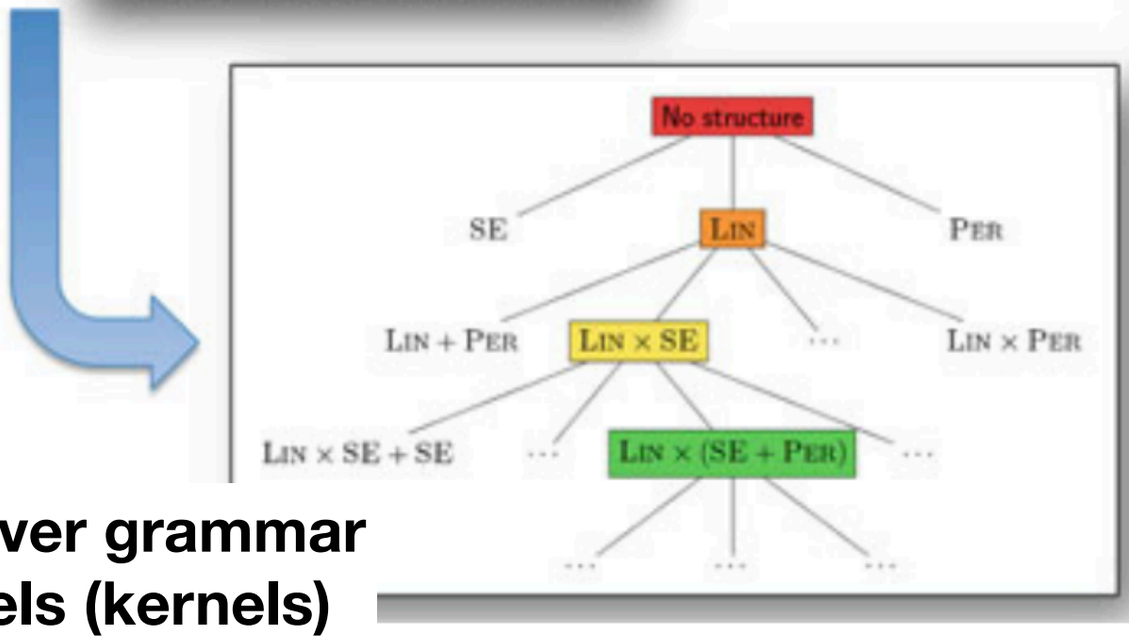


This component is approximately periodic with a period of 10.8 years. Across periods the shape of this function varies smoothly with a typical lengthscale of 36.9 years. The shape of this function within each period is very smooth and resembles a sinusoid. This component applies until 1643 and from 1716 onwards.

This component explains 71.5% of the residual variance; this increases the total variance explained from 72.8% to 92.3%. The addition of this component reduces the cross validated MAE by 16.82% from 0.18 to 0.15.

Figure 8: Pointwise posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

#	ACF min	ACF min loc	Periodogram max	Periodogram max loc	QQ max	QQ min
1	0.502	0.582	0.341	0.413	0.341	0.679
2	0.802	0.199	0.558	0.630	**0.049**	0.785
3	0.251	0.475	0.799	0.447	0.534	0.769
4	0.527	0.503	0.504	0.481	0.430	0.616
5	0.493	0.477	0.503	0.487	0.518	0.381



search over grammar of models (kernels)

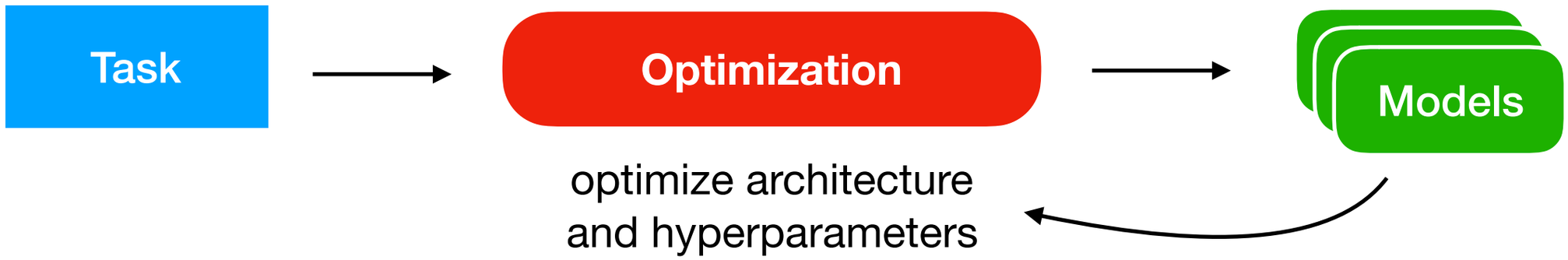
Lin
with linearly growing amplitude

SE × Per
approximately periodic function

model component to English translation

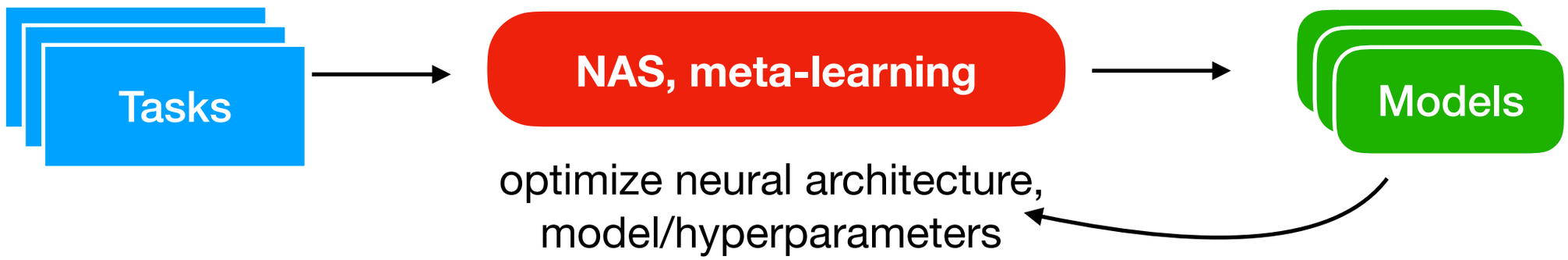
Overview

part 1 AutoML introduction, promising optimization techniques



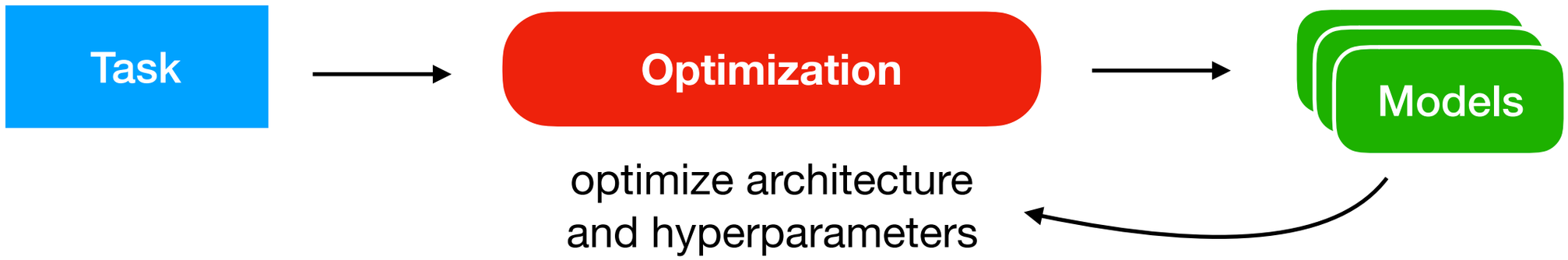
part 2 Meta-learning

part 3 Neural architecture search, meta-learning on neural nets



Overview

part I AutoML introduction, optimization techniques



1. Problem definition
2. (Pipeline) architecture search
3. Optimization techniques
4. Performance improvements
5. Benchmarks

AutoML Definition

Let:

$\{A^{(1)}, A^{(2)}, \ldots, A^{(n)}\}$ be a set of *algorithms* (operators)

$\Lambda^{(i)} = \lambda_1 \times \lambda_2 \times \ldots \times \lambda_m$ be the *hyperparameter space* for $A^{(i)}$

$\mathscr{A}$ be the space of possible *architectures* of one or more algorithms

$\Lambda_{\mathscr{A}} = \Lambda^{(1)} \times \Lambda^{(2)} \times \ldots \times \Lambda^{(m)}$ its combined *configuration space*

$\boldsymbol{\lambda} \in \Lambda_{\mathscr{A}}$ a specific *configuration* (of architecture and hyperparameters)

$\mathscr{L}(\boldsymbol{\lambda}, D_{train}, D_{valid})$ the loss of the model created by $\boldsymbol{\lambda}$, trained on data D_{train}, and validated on data D_{valid}

Find the configuration that minimizes the expected loss on a dataset $\mathscr{D}$:

$$\boldsymbol{\lambda}^* = \underset{\lambda \in \Lambda_{\mathscr{A}}}{\operatorname{argmin}} \mathbb{E}_{(D_{train}, D_{valid}) \sim \mathscr{D}} \mathscr{L}(\boldsymbol{\lambda}, D_{train}, D_{test})$$

Types of hyperparameters

- Continuous (e.g. learning rate, SVM_C,…)
- Integer (e.g. number of hidden units, number of boosting iterations,…)
- Categorical
  - e.g. choice of algorithm (SVM, RandomForest, Neural Net,…)
  - e.g. choose of operator (Convolution, MaxPooling, DropOut,…)
  - e.g. activation function (ReLU, Leaky ReLU, tanh,…)
- Conditional
  - e.g. SVM kernel if SVM is selected, kernel width if RBF kernel is selected
  - e.g. Convolution kernel size if Convolution layer is selected

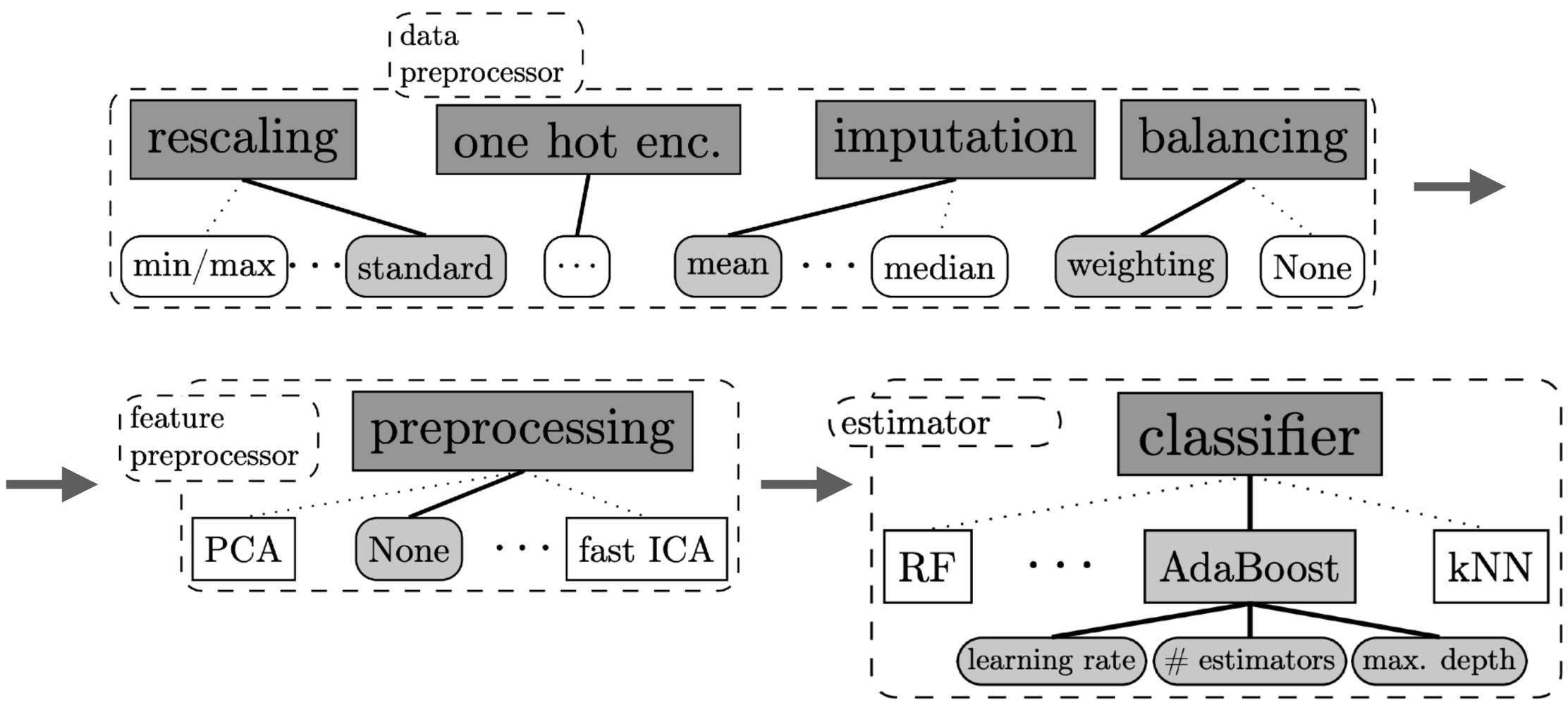
Architecture vs hyperparameters

- We can identify two subproblems:
  - **Architecture search**: search space of all possible architectures
    - ***Pipelines***: Fixed predefined pipeline, grammars, genetic programming, planning, Monte-Carlo Tree Search
    - ***Neural architectures***: See part 3
  - **Hyperparameter optimization**: optimize remaining hyperparameters
    - ***Optimization***: grid/random search, Bayesian optimization, evolution, multi-armed bandits, gradient descent (only NNs)
    - ***Meta-learning:*** See part 2
- Can be solved *consecutively*, *simultaneously* or *interleaved*
- ***Compositionality***: breaking down the learning process into smaller reusable tasks makes it easier, more transferable, more robust

Pipeline search: fixed pipelines

- Parameterize best-practice (linear) pipelines
  - Introduce conditional hyperparameters $\lambda_r \in \{A^{(1)}, \ldots, A^{(k)}, \varnothing\}$
  - Combined Algorithm Selection and Hyperparameter optimization (CASH):

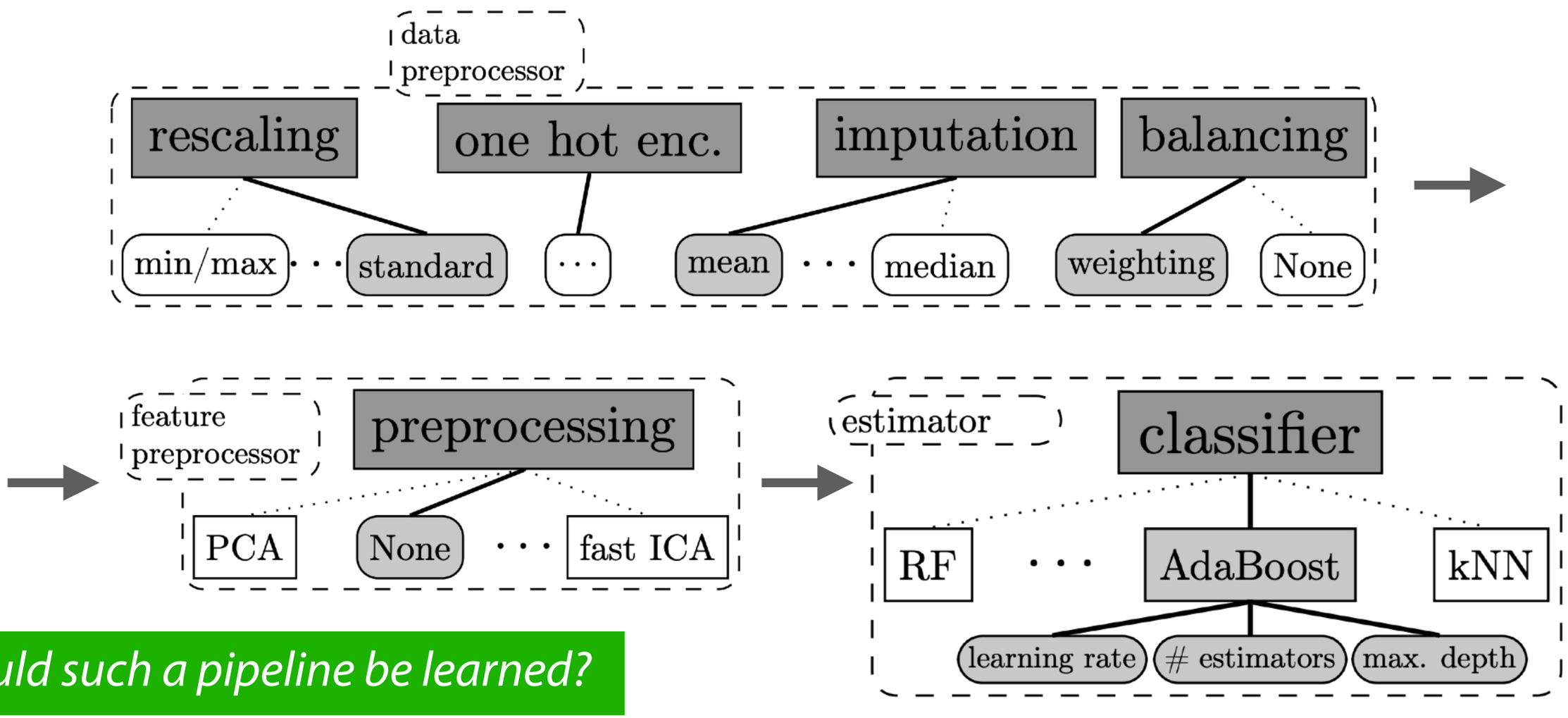
$$\Lambda_{\mathscr{A}} = \Lambda^{(1)} \times \Lambda^{(2)} \times \ldots \times \Lambda^{(m)} \times \lambda_r$$



Pipeline search: fixed pipelines

- Parameterize best-practice (linear) pipelines
  - Introduce conditional hyperparameters $\lambda_r \in \{A^{(1)}, \ldots, A^{(k)}, \varnothing\}$
  - Combined Algorithm Selection and Hyperparameter optimization (CASH):

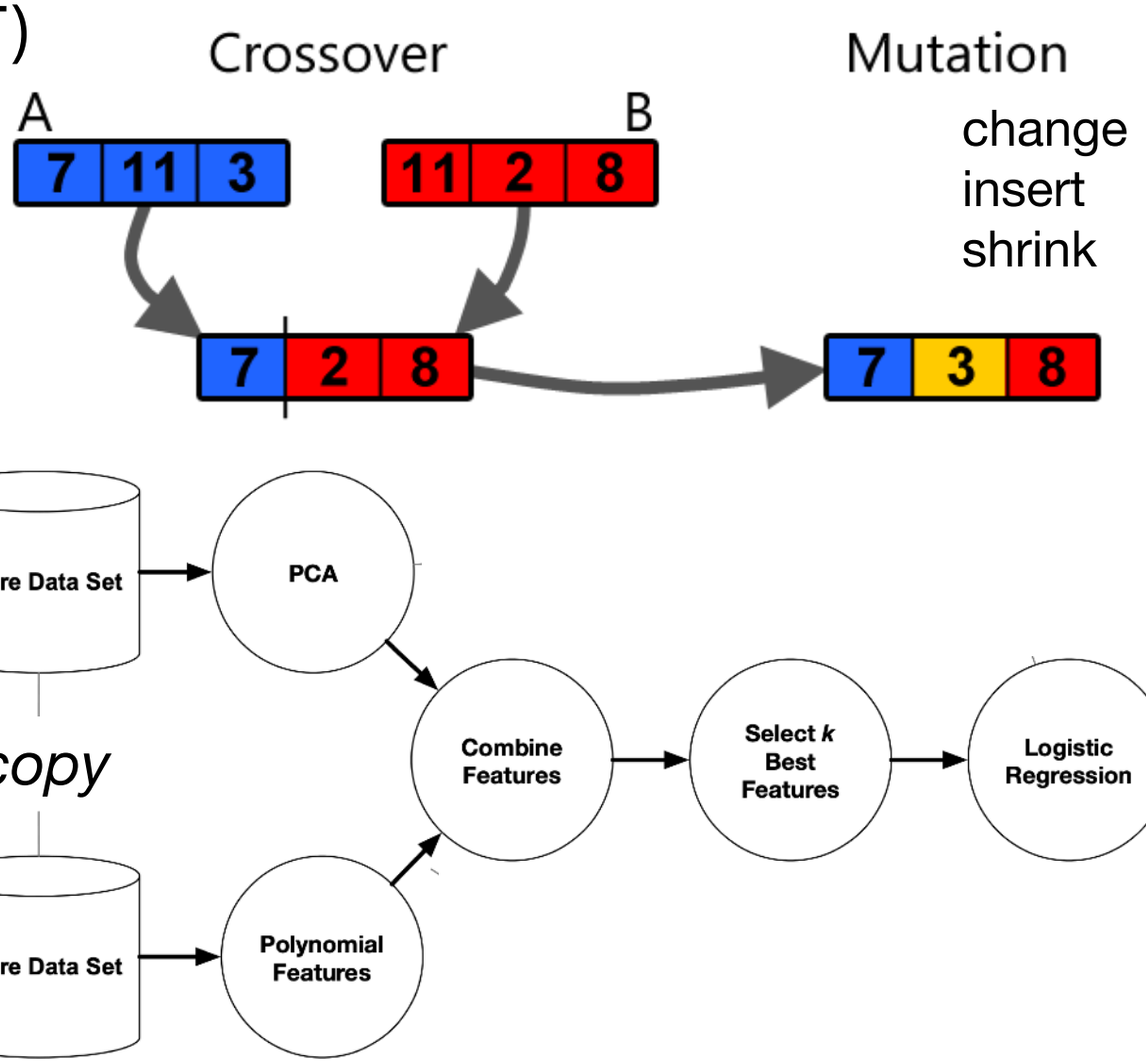
$$\Lambda_{\mathscr{A}} = \Lambda^{(1)} \times \Lambda^{(2)} \times \ldots \times \Lambda^{(m)} \times \lambda_r$$

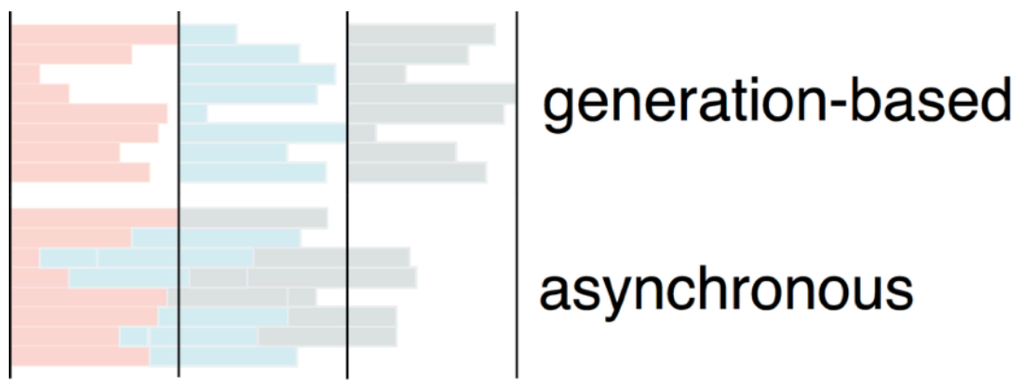


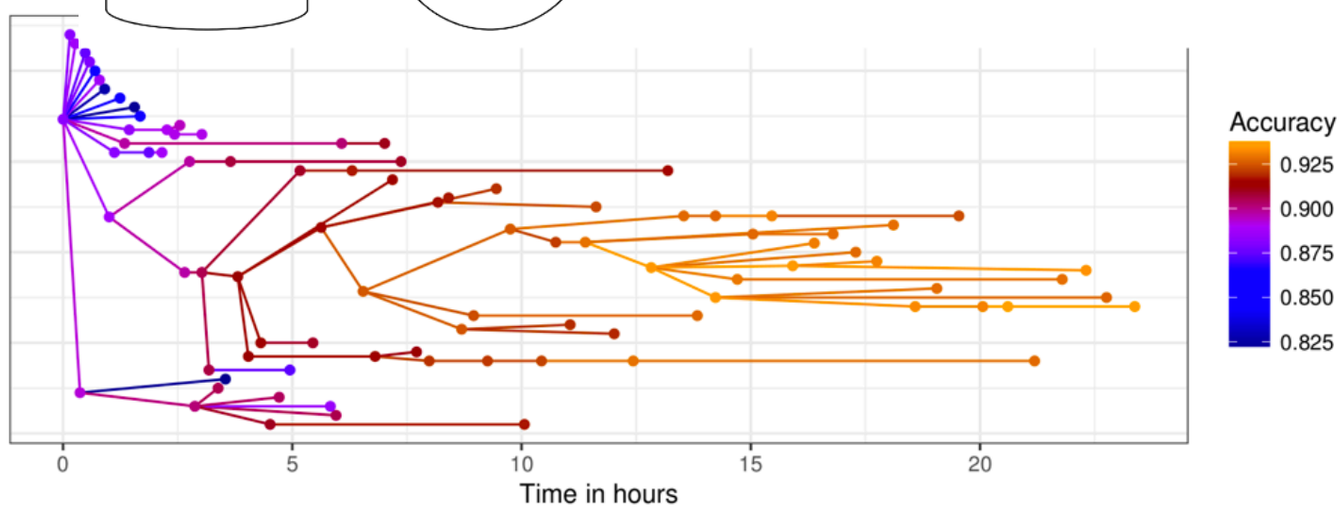
Could such a pipeline be learned?

Pipeline search: genetic programming

- Tree-based pipeline optimization (TPOT)
  - Start with random pipelines, best of every generation will cross-over or mutate
  - GP primitives include data copy and feature joins: trees
  - Multi-objective optimization: accurate but short
  - Easy to parallelize
- Asynchronous evolution (GAMA)







Pipeline search: TPOT demo

```
from tpot import TPOTClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,
                                                    train_size=0.75, test_size=0.25)

tpot = TPOTClassifier(generations=5, population_size=50, verbosity=2, n_jobs=-1)
tpot.fit(X_train, y_train)
```

```
Optimization Progress:   0%|          | 0/300 [00:00<?, ?pipeline/s]
```

```
print(tpot.score(X_test, y_test))
```

Pipeline search: TPOT demo

```
from tpot import TPOTClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,
                                                    train_size=0.75, test_size=0.25)

tpot = TPOTClassifier(generations=5, population_size=50, verbosity=2, n_jobs=-1)
tpot.fit(X_train, y_train)
```

```
Optimization Progress:   0%|          | 0/300 [00:00<?, ?pipeline/s]
```

```
print(tpot.score(X_test, y_test))
```

Pipeline search: genetic programming

- Grammar-based genetic programming (RECIPE) *[de Sa et al. 2017]*
  - Encodes background knowledge, avoids invalid pipelines
  - Cross-over and mutation respect the grammar

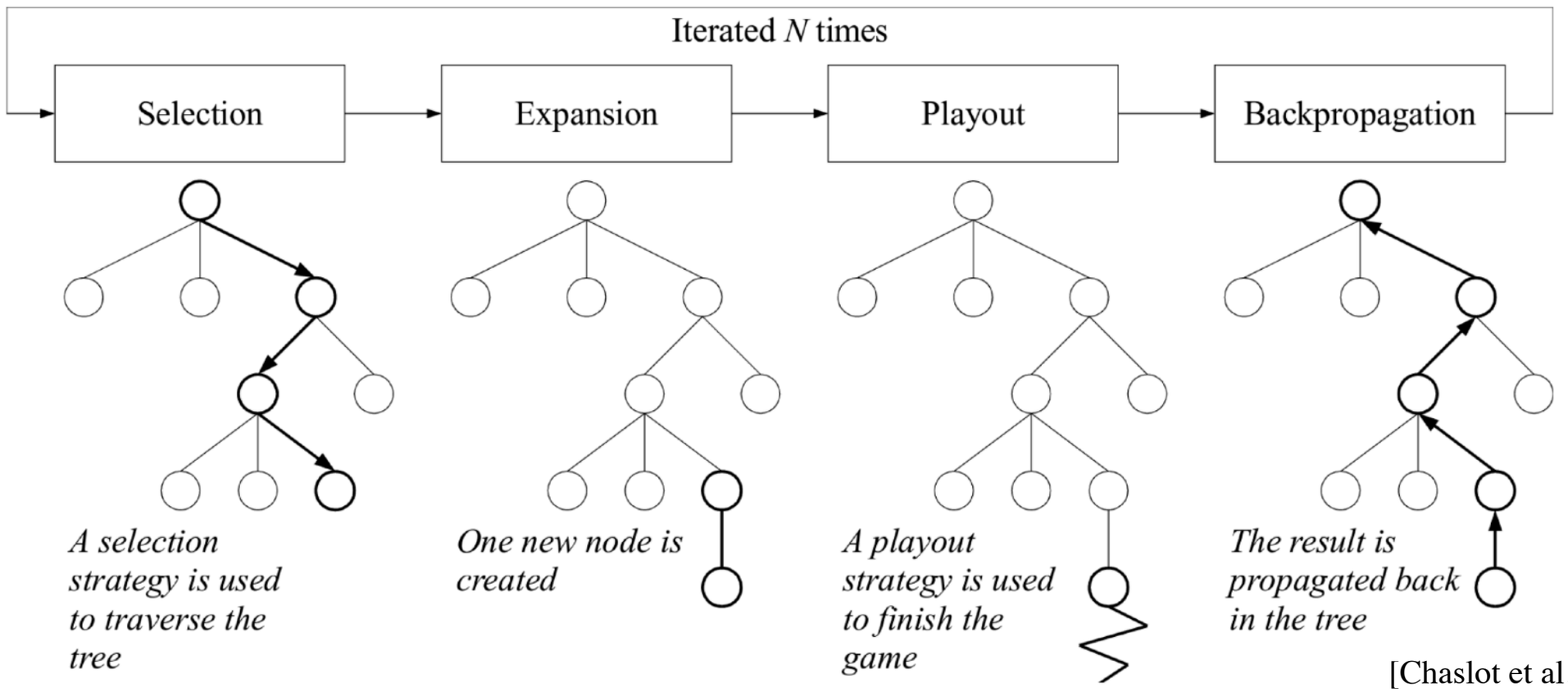
production rule *optional* *non-terminal*

```
<Start> ::= [<Pre-processing>] <Algorithm>
<Pre-processing> ::= [<Imputation>] <DimensionalityDefinition>
<Imputation> ::= Mean | Median | Max
<DimensionalityDefinition> ::= <FeatureSelection>   [<FeatureConstruction>]
                               [<FeatureSelection>] <FeatureConstruction>
<FeatureSelection> ::= <Supervised> | <Unsupervised>
<Supervised> ::= SelectKBest <K> <score> | VarianceThreshold | [...]
<score> ::= f-classification | chi2
<K> ::= 1 | 2 | 3 | [...] | NumberOfFeatures - 1
<perc> ::= 1 | 2 | 3 | [...] | 99
<Unsupervised> ::= PCA | FeatureAgglomeration <affinity> | [...]
<affinity> ::= Euclidian | L1 | L2 | Manhattan | Cosine
<FeatureConstruction> ::= PolynomialFeatures
<Algorithm> ::=  <NaiveBayes> | <Trees> | [...]
<NaiveBayes> ::= GaussianNB | MultinomialNB | BernoulliNB
```

terminal

Pipeline search: Monte Carlo Tree Search

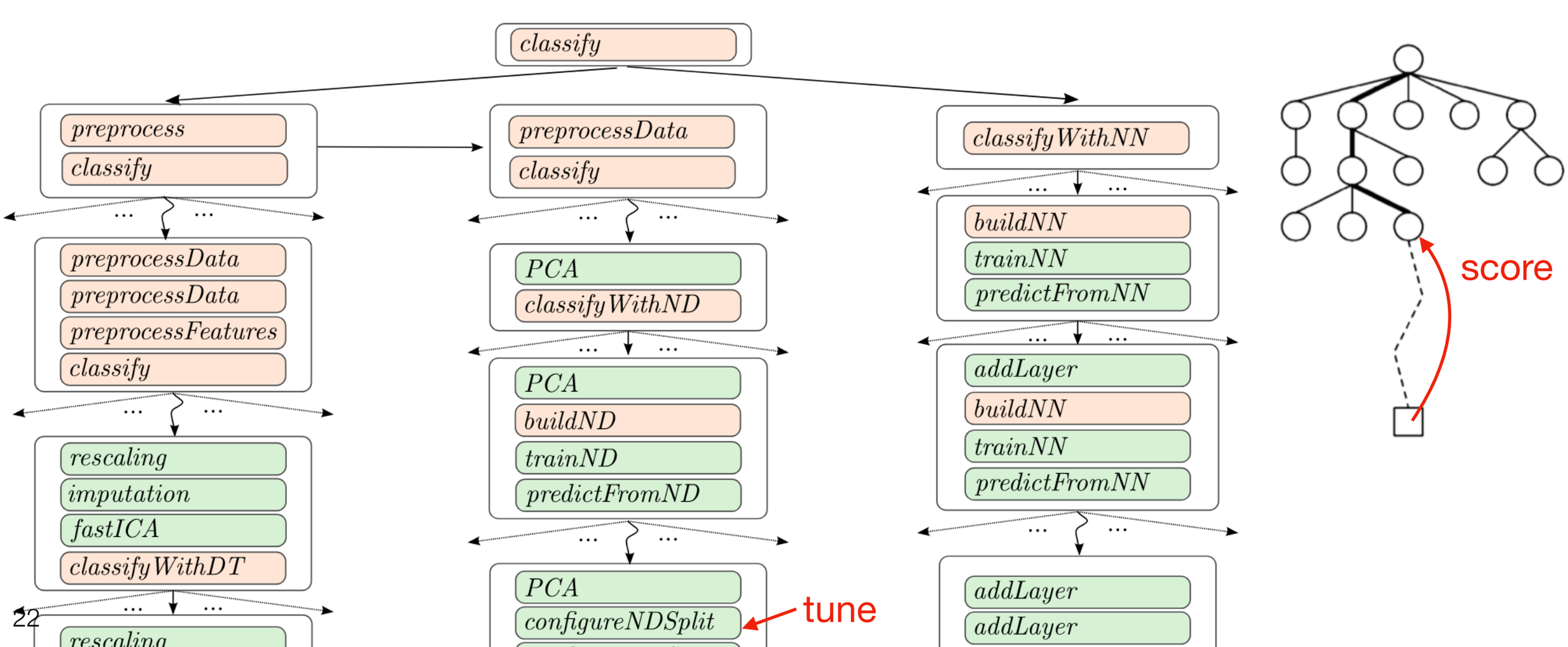
- Use MCTS to search for optimal pipelines
- Optimize the structure and hyperparameters simultaneously by building a surrogate model to predict configuration performance
  - Bayesian surrogate model: MOSAIC *[Rakotoarison et al. 2019]*
  - Neural network: AlphaD3M *[Drori et al. 2018]*



[Chaslot et al., 2008]

Pipeline search: planning

- Hierarchical planners: ML-Plan *[Mohr et al. 2018]*, *P4ML* *[Gil et al. 2018]*
  - Graph search problem, can be solved with best first search
  - Use random path completion (as in MCTS) to evaluate each node
    - Hyperparameter optimization interleaved as possible actions



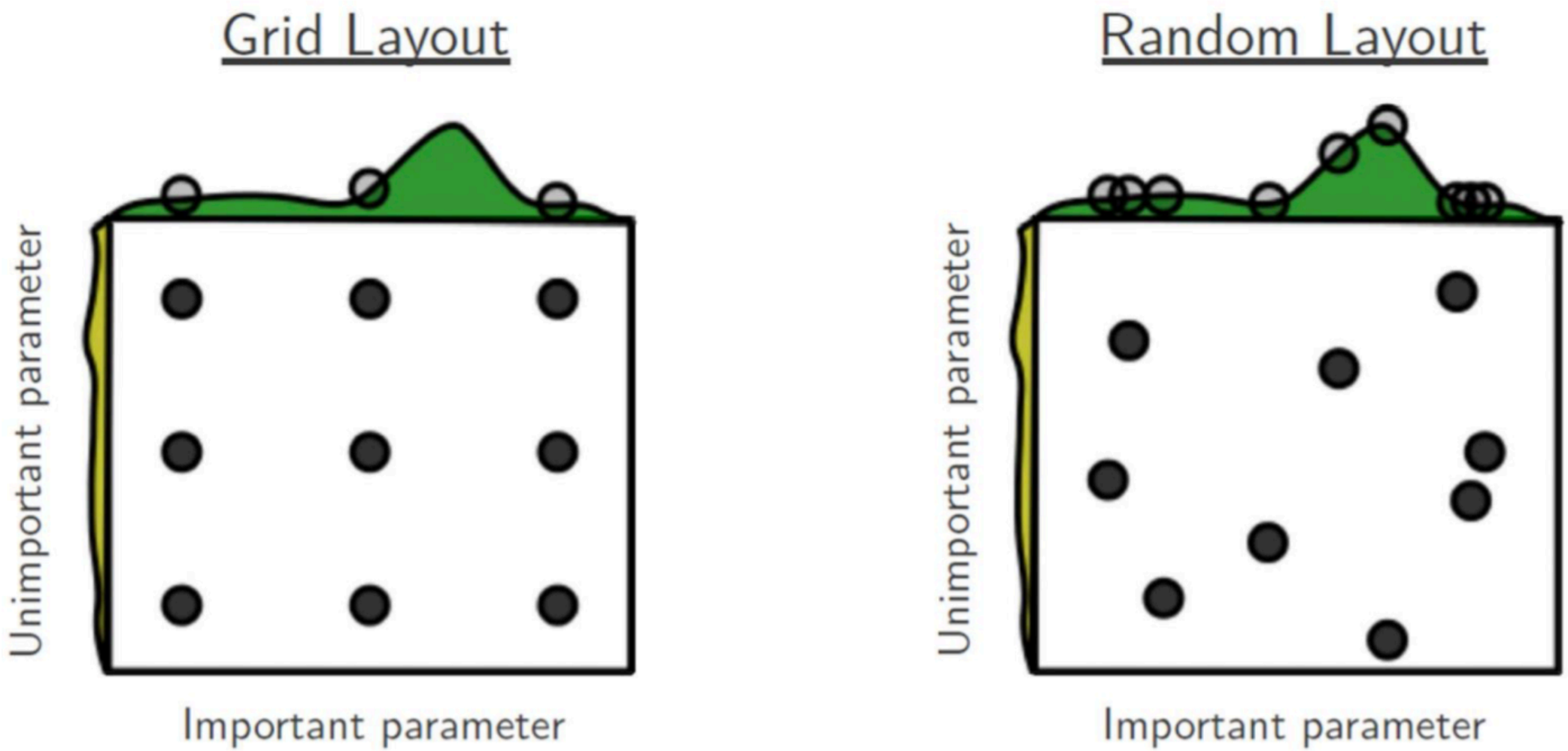
Hyperparameter optimization (HPO)

Black box optimization: expensive for machine learning.
Sample efficiency is important!

$$\boldsymbol{\lambda}^* = \underset{\lambda \in \Lambda_{\mathscr{A}}}{\operatorname{argmin}} \, \mathbb{E}_{(D_{train}, D_{valid}) \sim \mathscr{D}} \mathscr{L}(\boldsymbol{\lambda}, D_{train}, D_{test})$$

HPO: grid and random search

- Random search handles unimportant dimensions better
- Easily parallelizable, but uninformed (no learning)

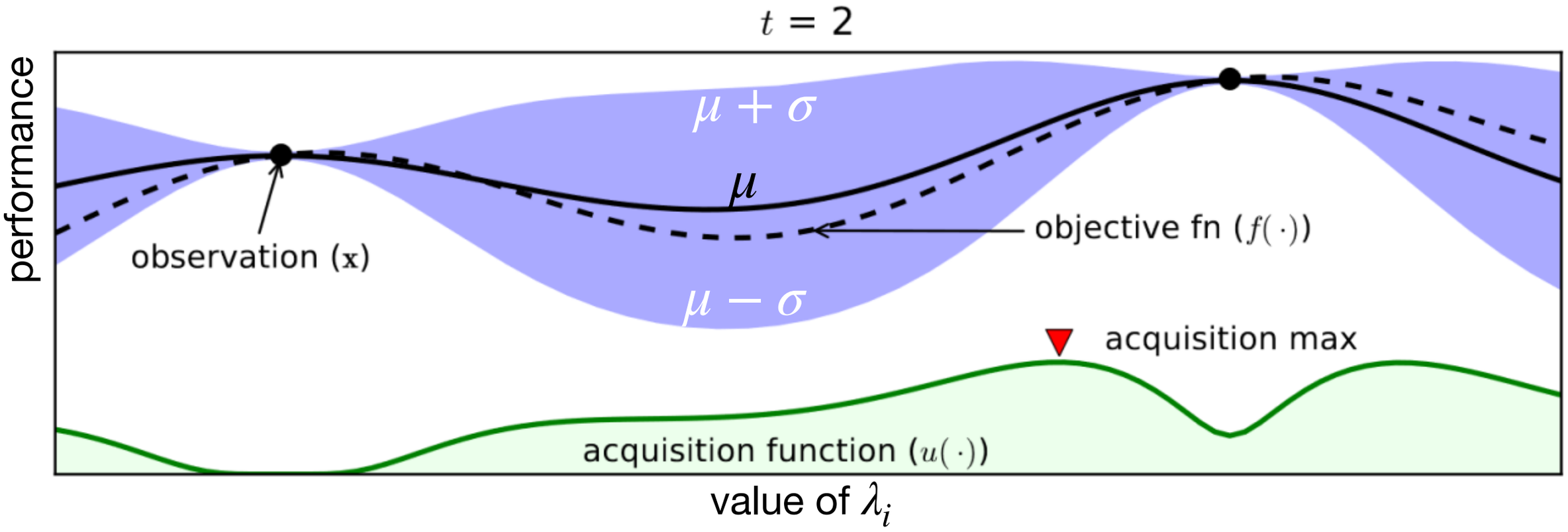


Bergstra & Bengio, JMLR 2012

Mockus [1974]

HPO: Bayesian Optimization

- Start with a few (random) hyperparameter configurations
- Build a ***surrogate model*** to predict how well other configurations will work: mean μ and standard deviation σ (blue band)
  - Any probabilistic regression model: e.g. Gaussian processes
- To avoid a greedy search, use an ***acquisition function*** to trade off exploration and exploitation, e.g. Expected Improvement (EI)
- Sample for the best configuration under that function

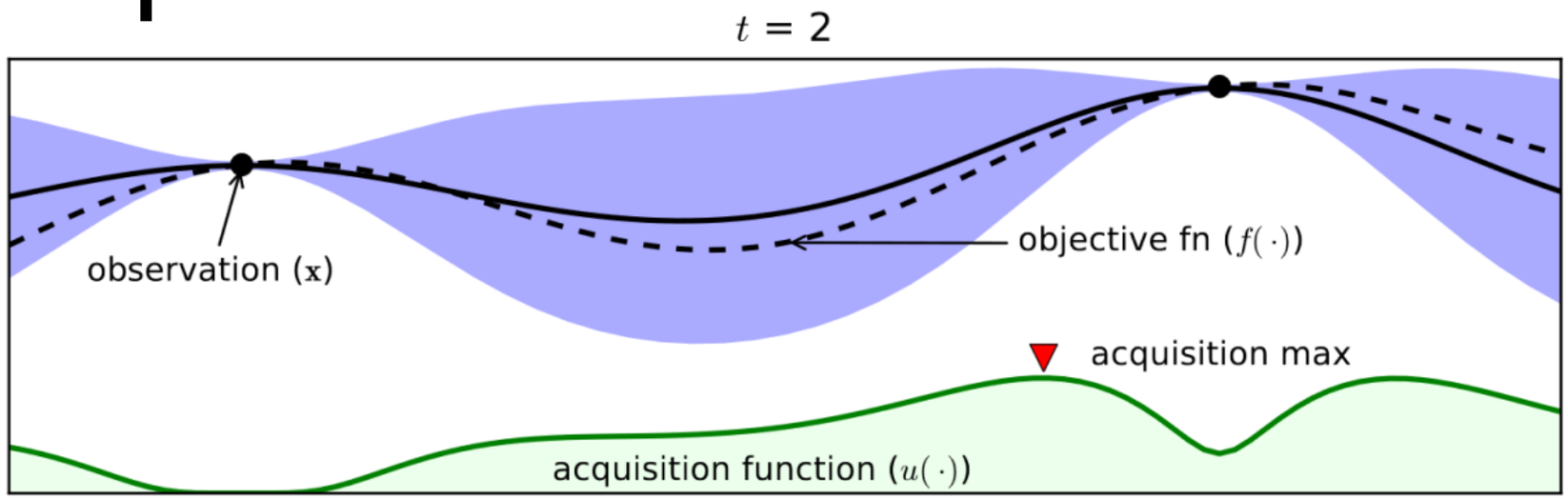


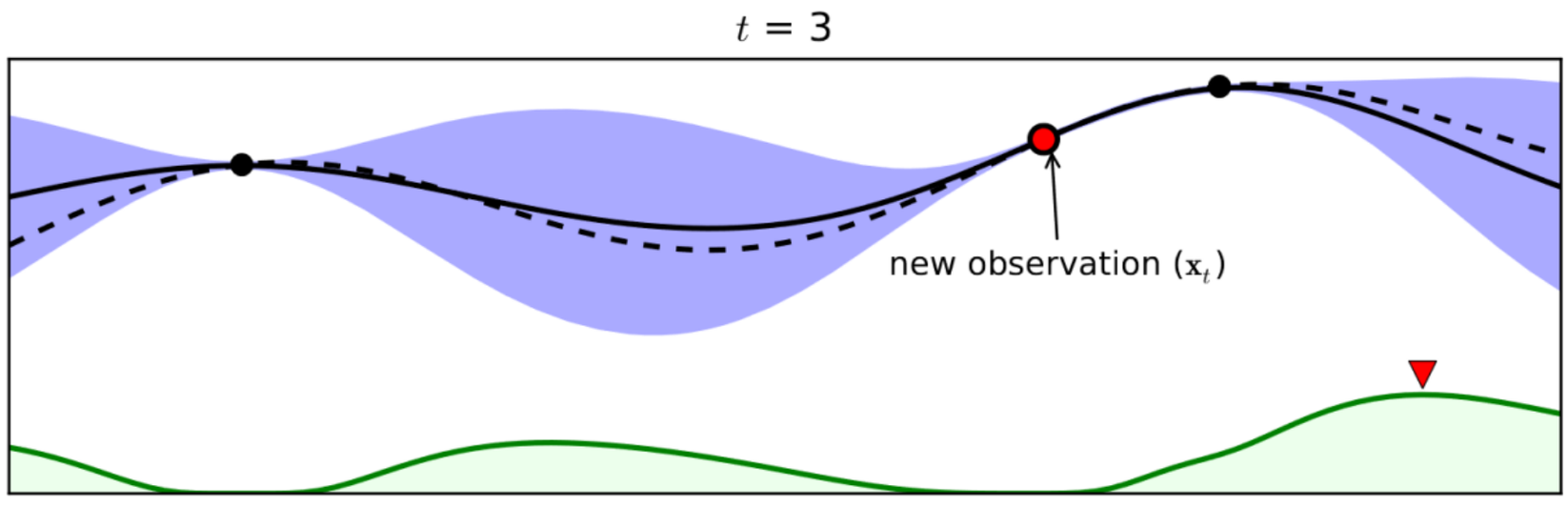
HPO: Bayesian Optimization

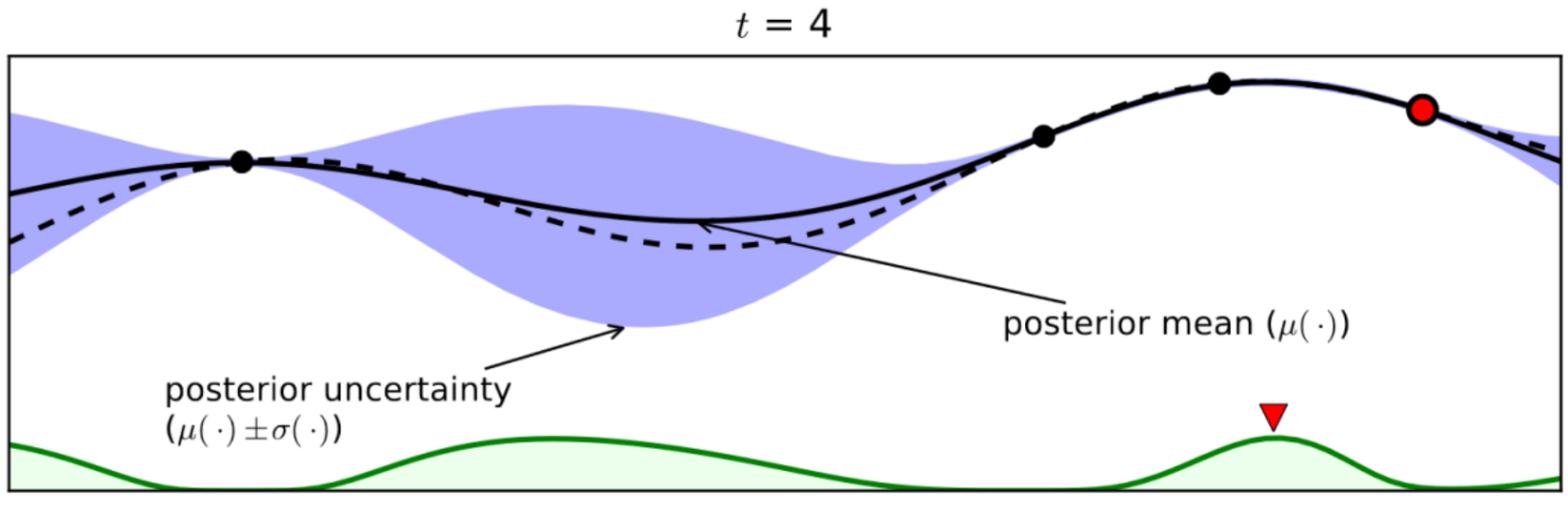
- Repeat
- Stopping criterion:
  - Fixed budget (time, evaluations)
  - Min. distance between configs
  - Threshold for acquisition function
  - Still overfits easily
- Convergence results

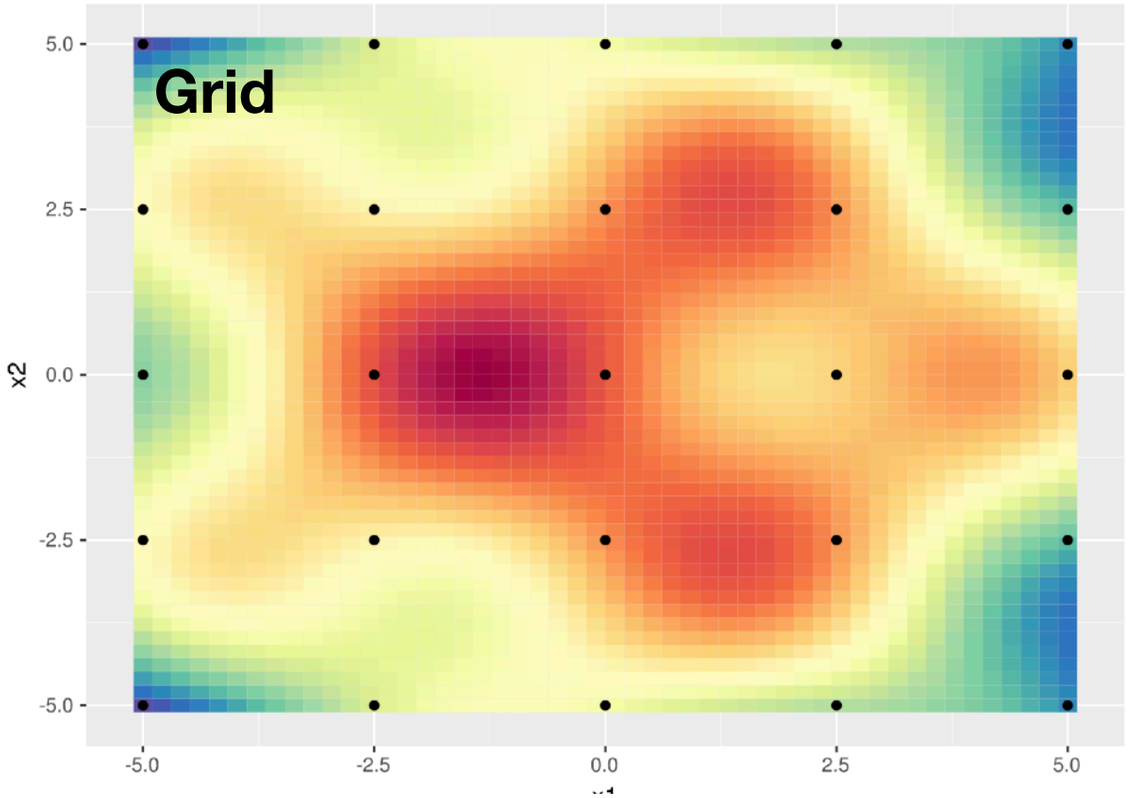
Srinivas et al. 2010, Freitas et al. 2012, Kawaguchi et al. 2016

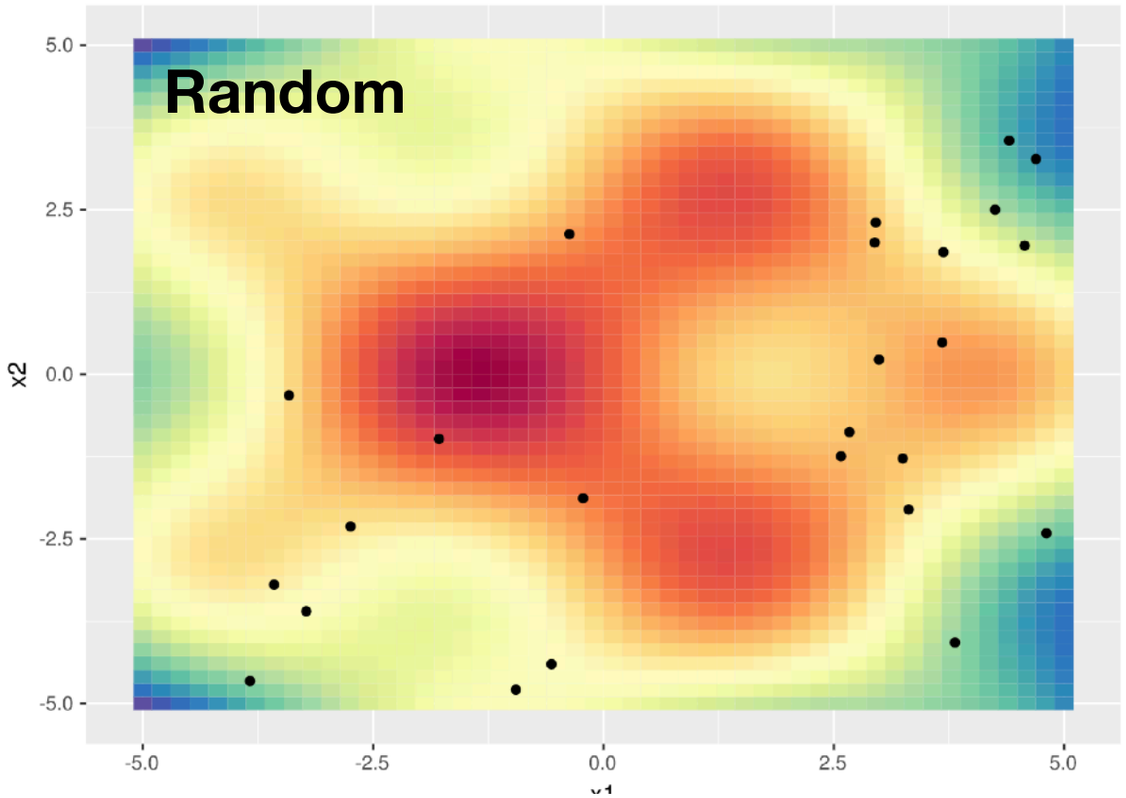
- Good for non-convex, noisy objectives
- Used in AlphaGo

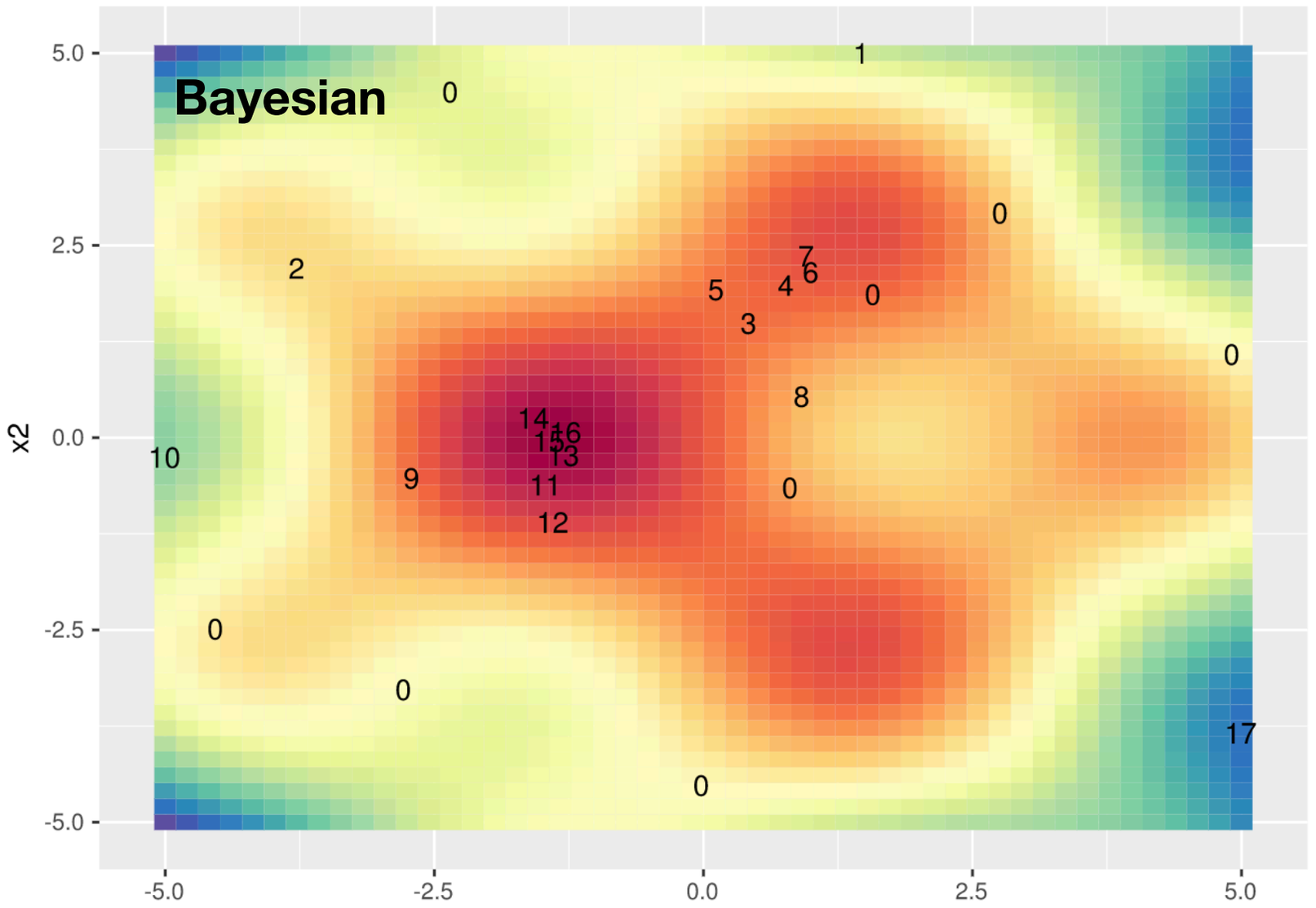







Grid
5.0
2.5
0.0
-2.5
-5.0
x2
-5.0
-2.5
0.0
2.5
5.0


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


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

Bayesian Optimization: surrogate models

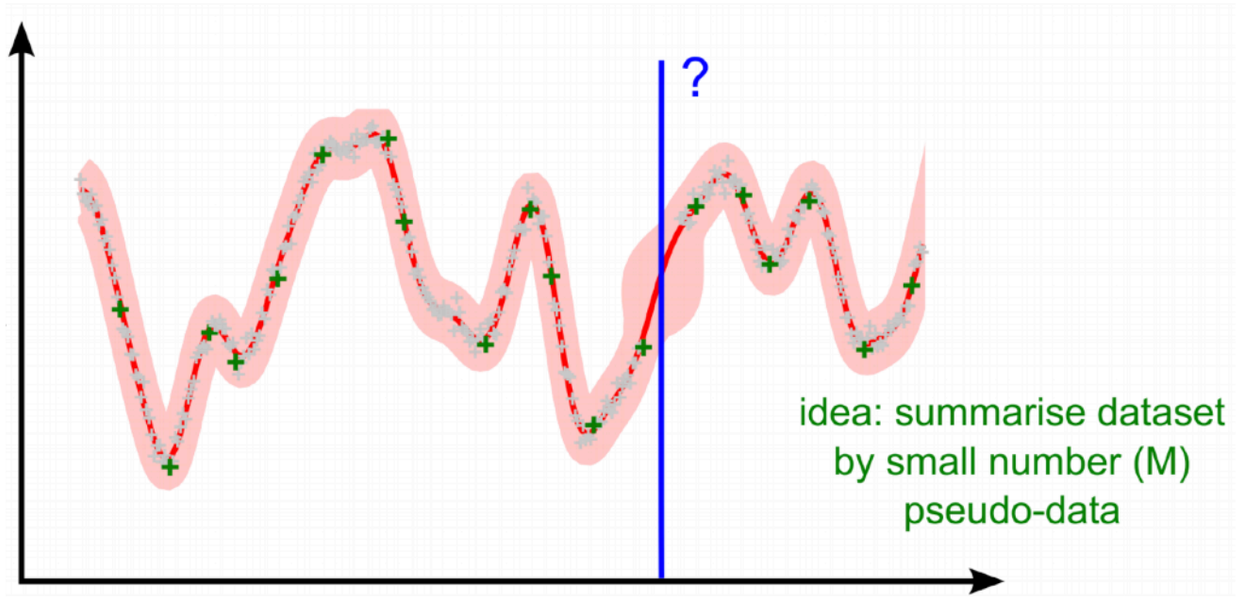
Gaussian processes

+ uncertainty, extrapolation

- Scalability (cubic)

  Sparse GPs *[Lawrence et al. 2003]*

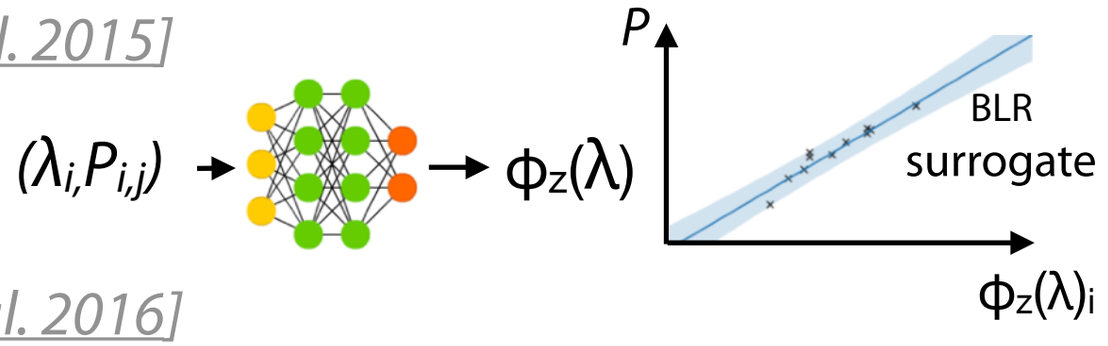
  Random embeddings *[Wang et al. 2013]*



- Robustness? Meta-BayesOpt to optimize kernel *[Malkomes et al. 2016]*

Neural networks + Bayesian LR *[Snoek et al. 2015]*

+ Scalable, Learns basis expansion Φ_z



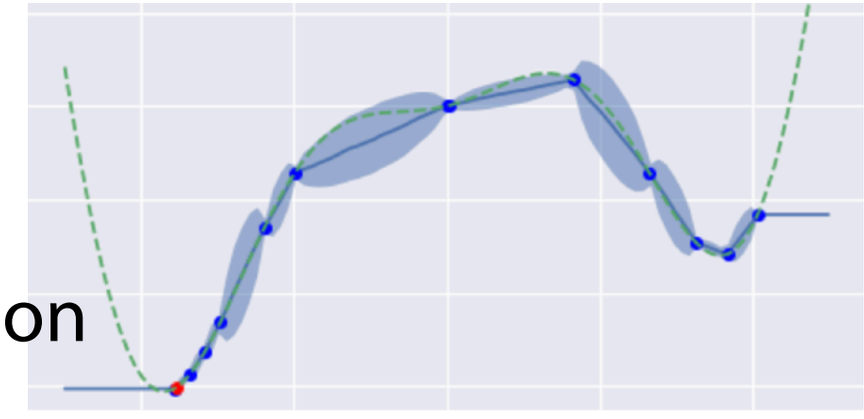
Bayesian neural networks *[Springenberg et al. 2016]*

+ Bayesian - Scalability not studied yet

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

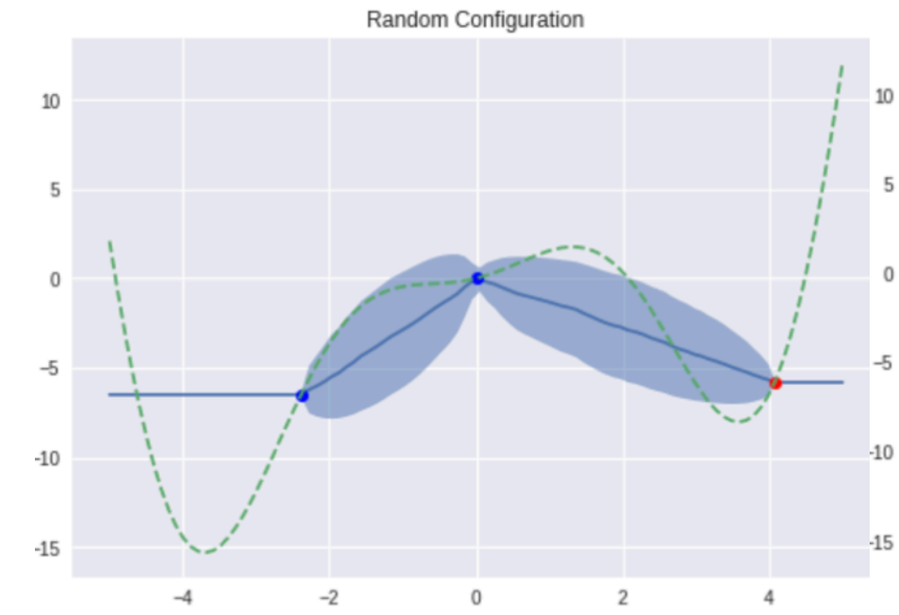
Frequentist uncertainty estimate (variance)

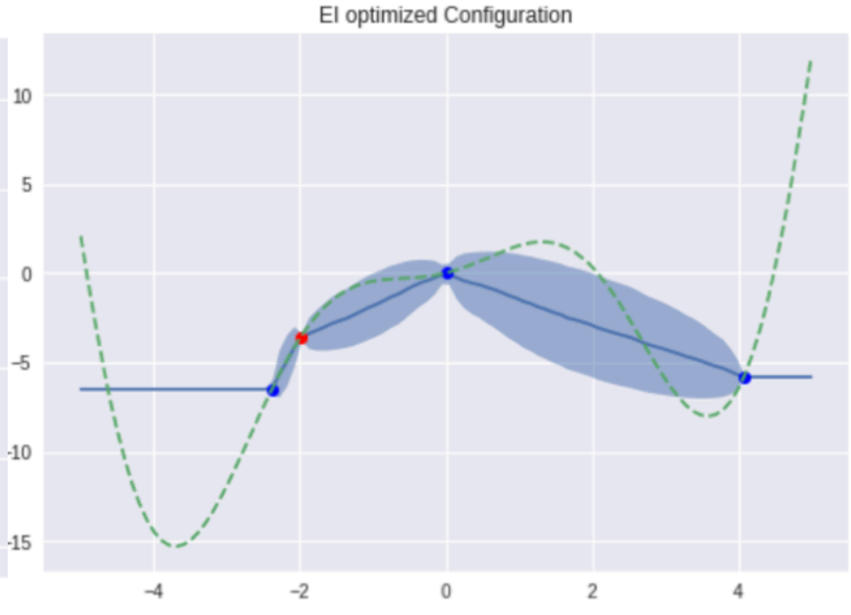
+ scalable, complex HPs - uncertainty, extrapolation

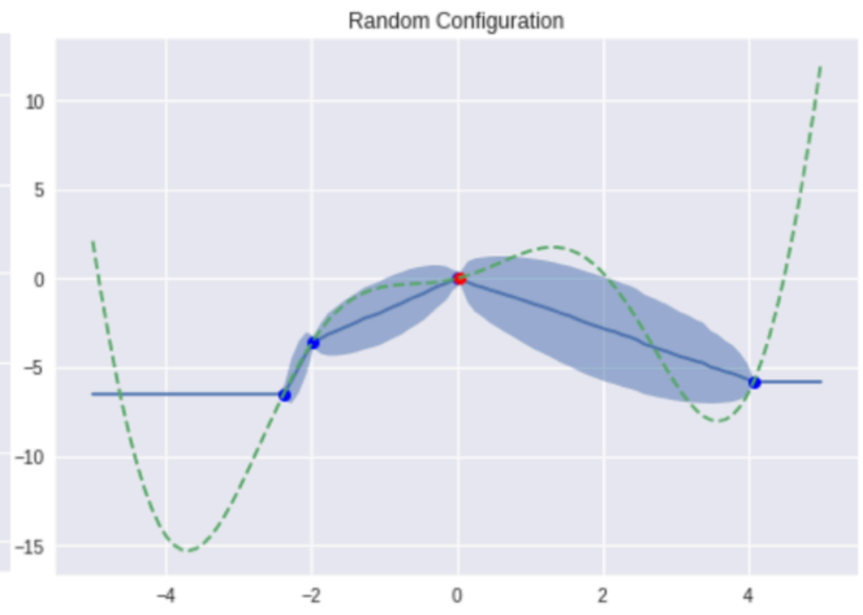


Bayesian Optimization: surrogate models

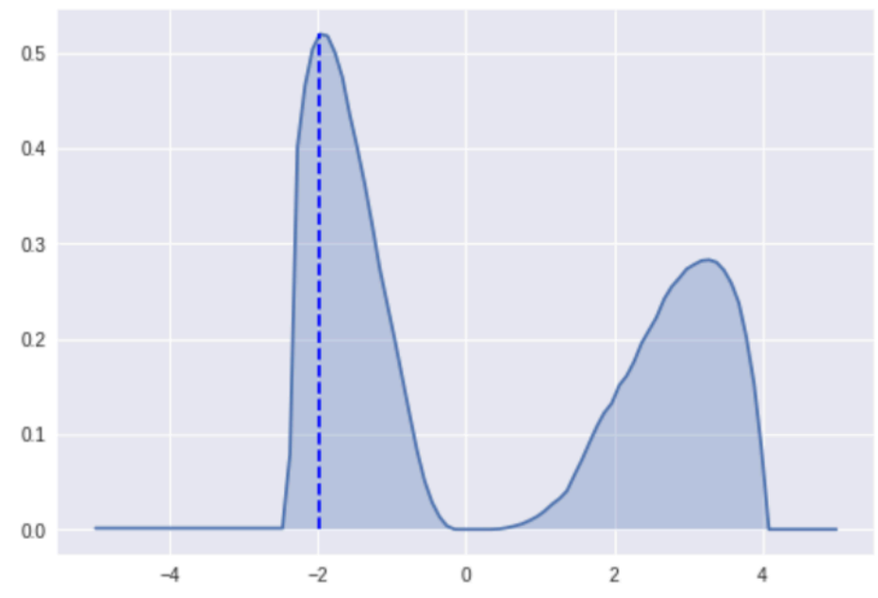
Random Forests (SMAC) example:



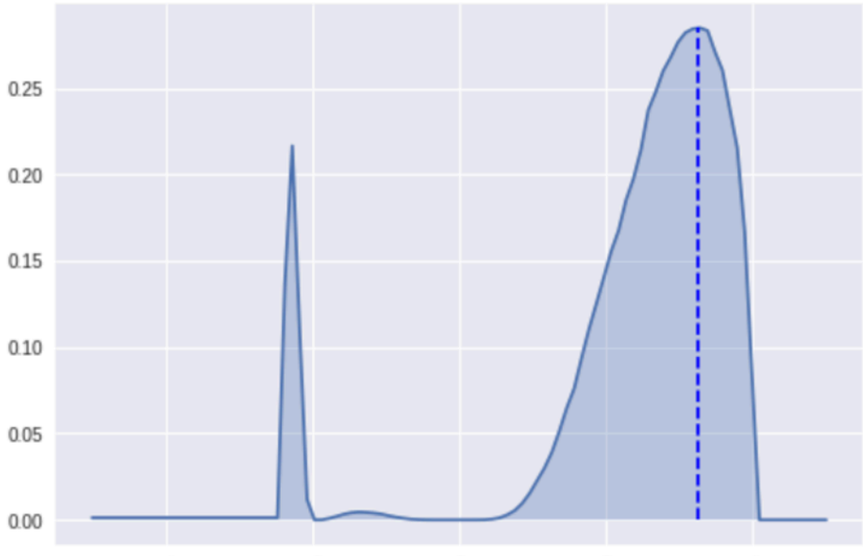




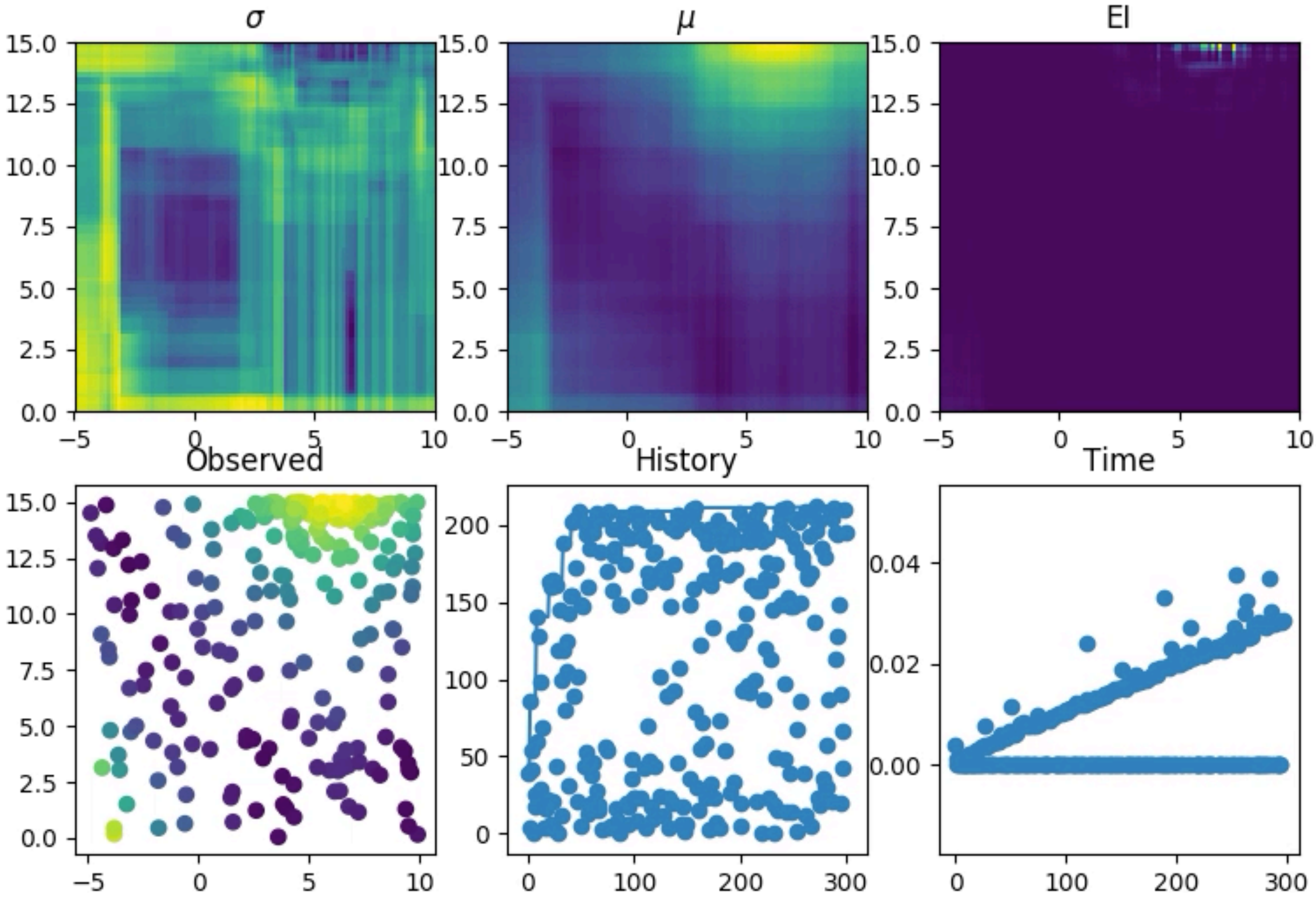
interleaved with random search



(minimizing)

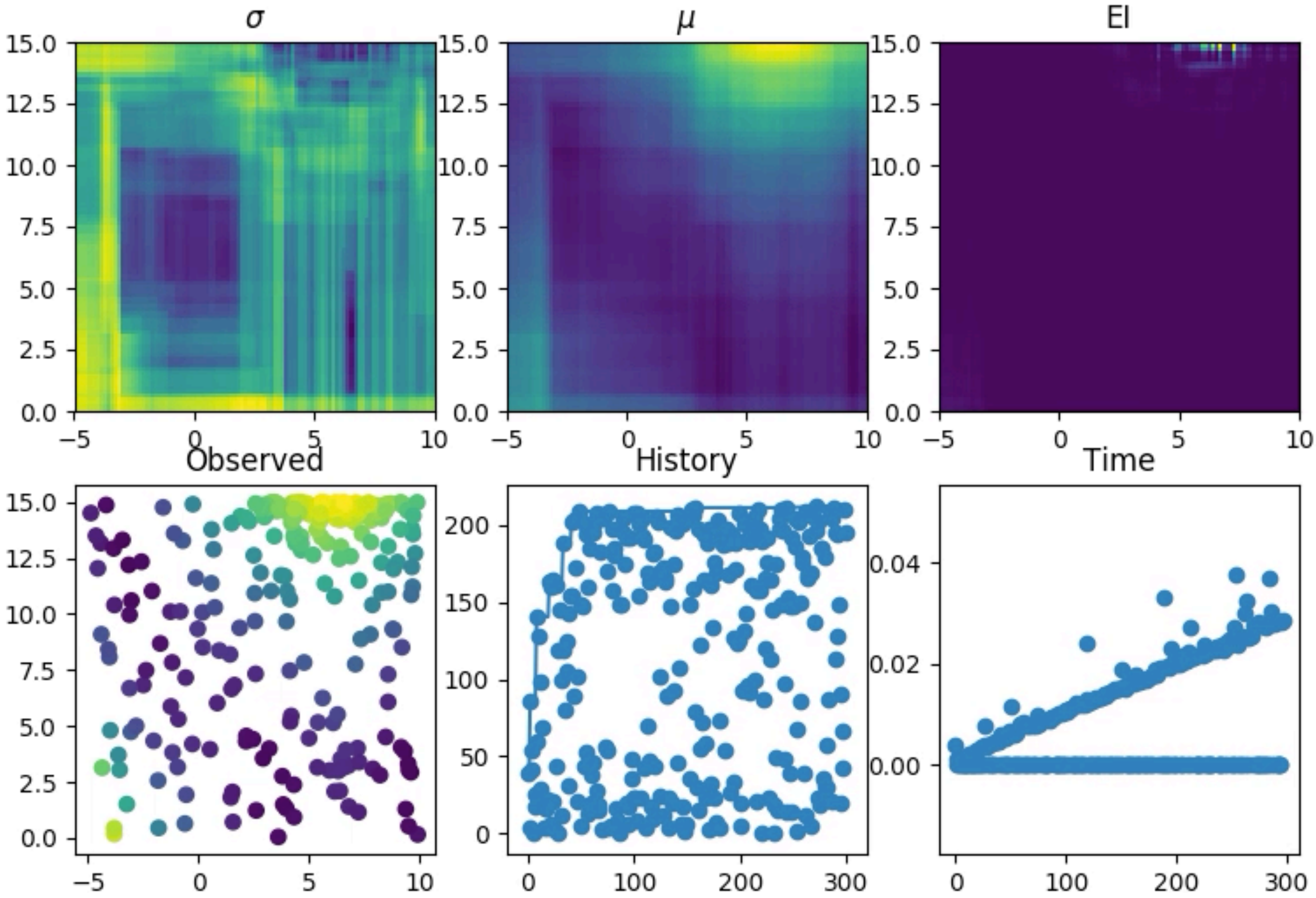


Random Forest Surrogate



animation by Jeroen van Hoof

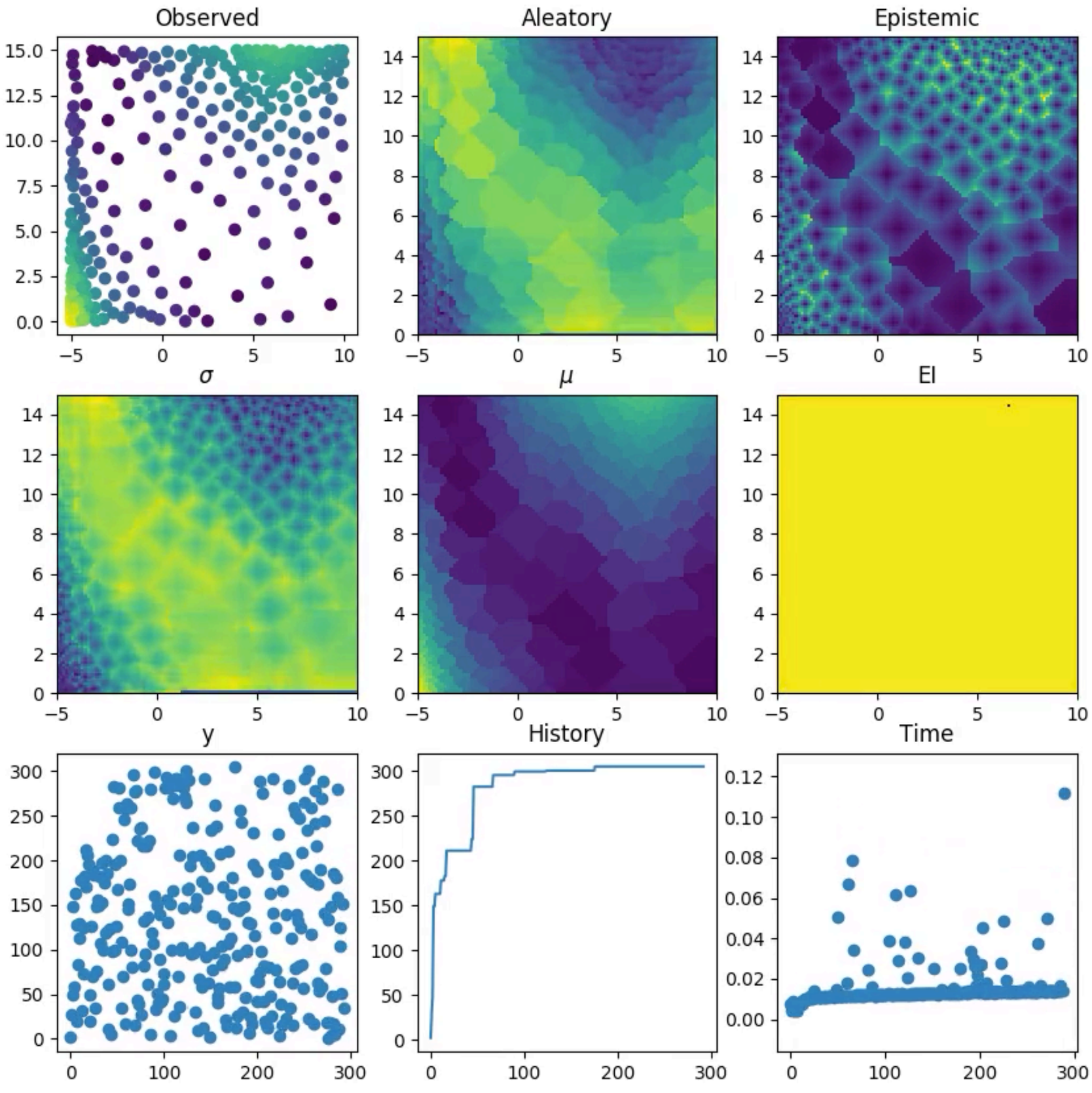
Random Forest Surrogate



animation by Jeroen van Hoof

Gradient Boosting Surrogate

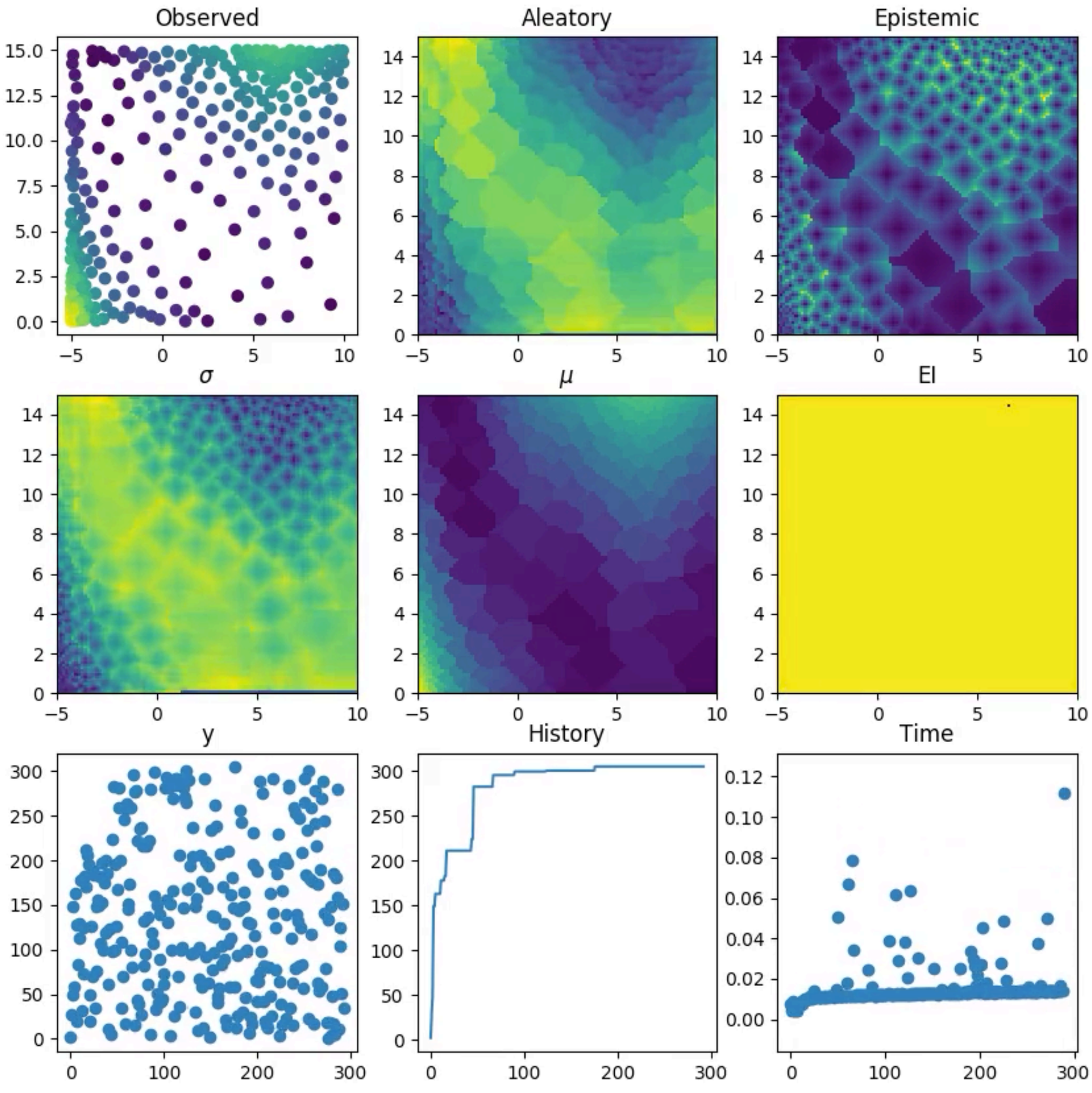
Estimate uncertainty with quantile regression



Jeroen van Hoof (under review)

Gradient Boosting Surrogate

Estimate uncertainty with quantile regression



Jeroen van Hoof (under review)

HPO: Tree of Parzen Estimators

1.Test some hyperparameters

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

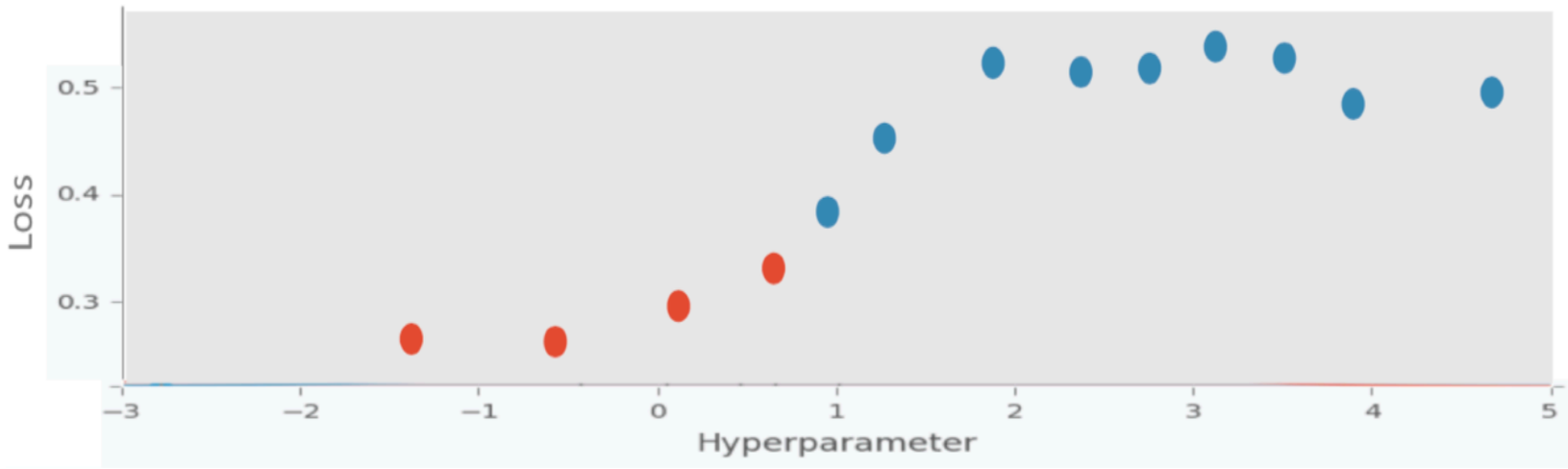
3. Fit non-parametric KDE for $p(\lambda = \text{good})$ and $p(\lambda = \text{bad})$

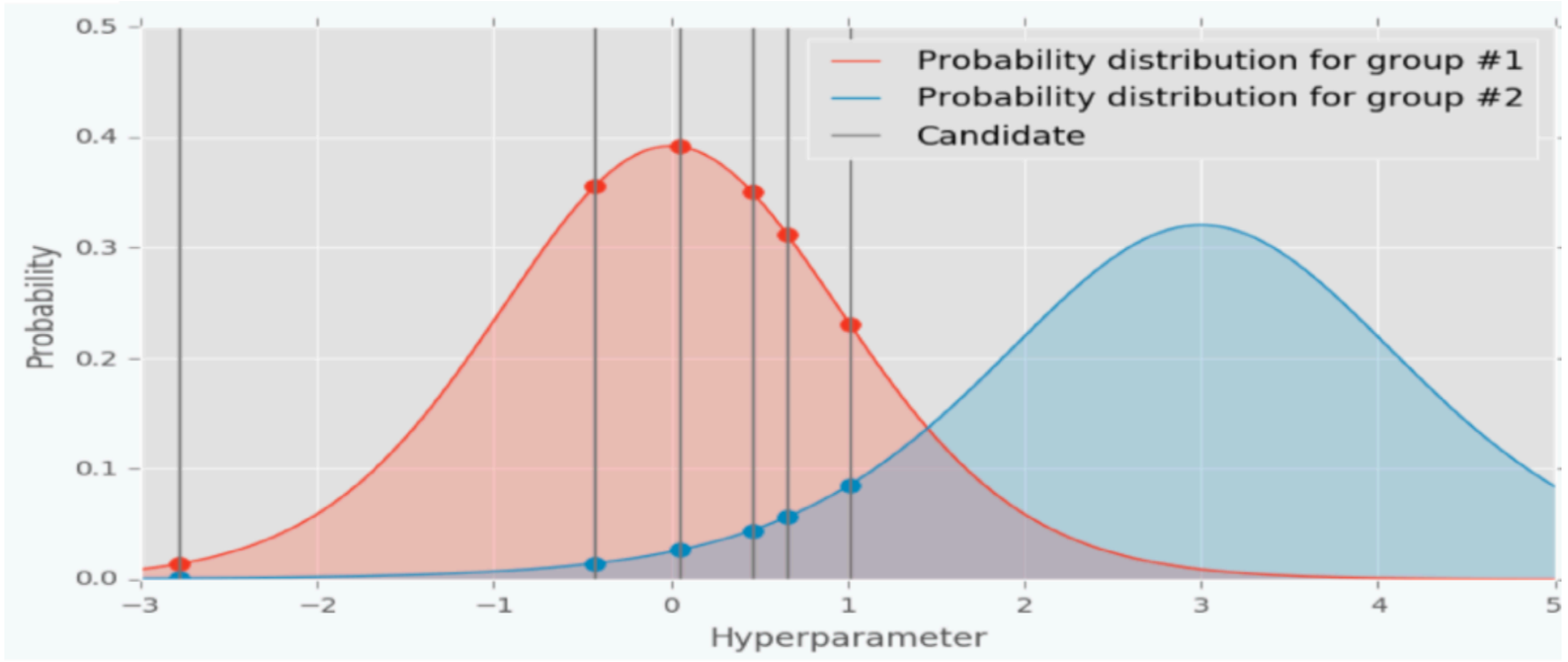
4. For a few samples, evaluate $\dfrac{p(\lambda = \text{good})}{p(\lambda = \text{bad})}$

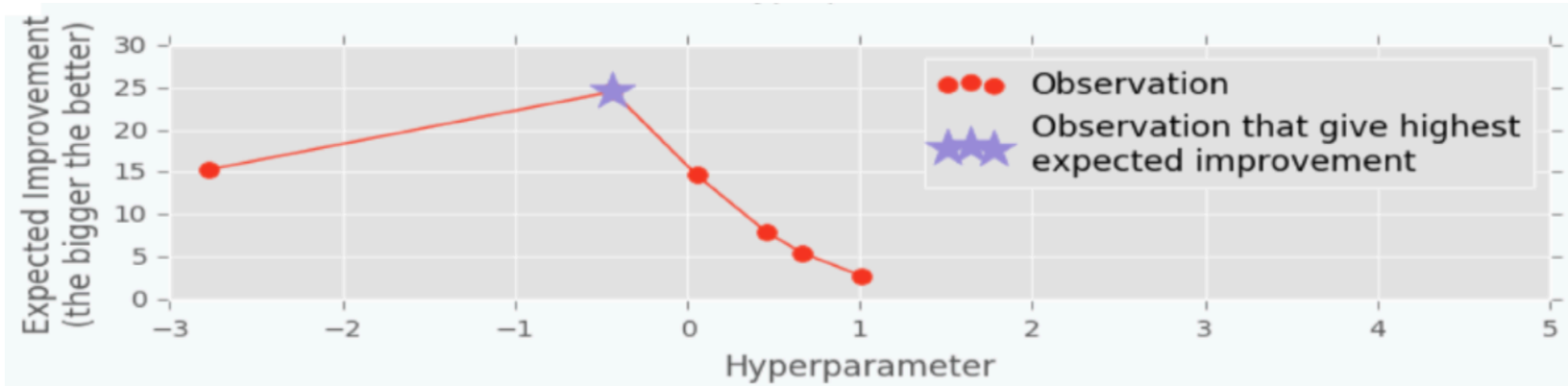
Shown to be equivalent to EI!

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

Used in HyperOpt-sklearn [Komer et al. 2019]



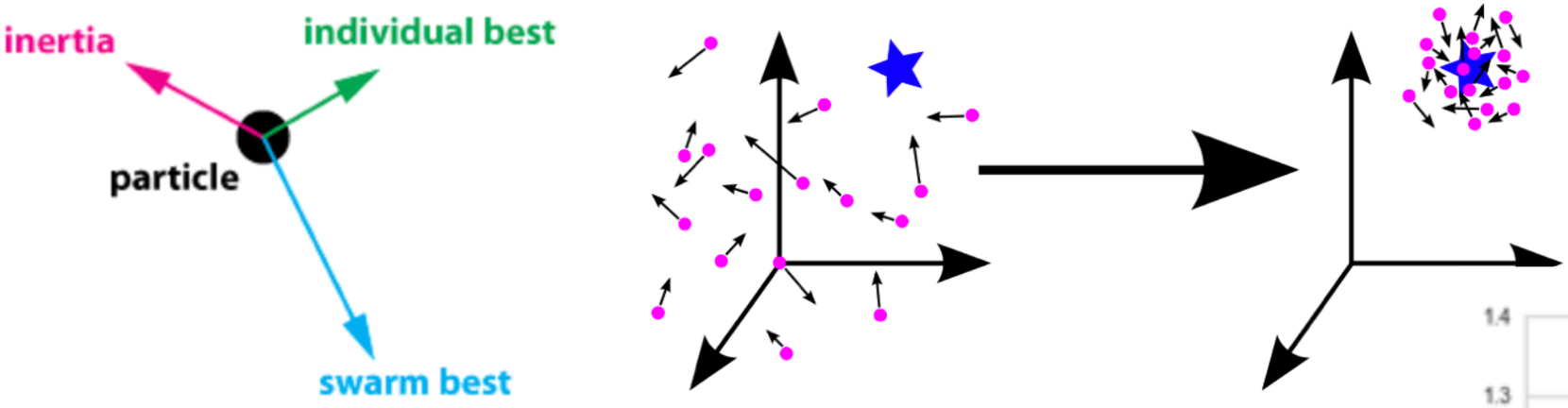




HPO: population-based methods

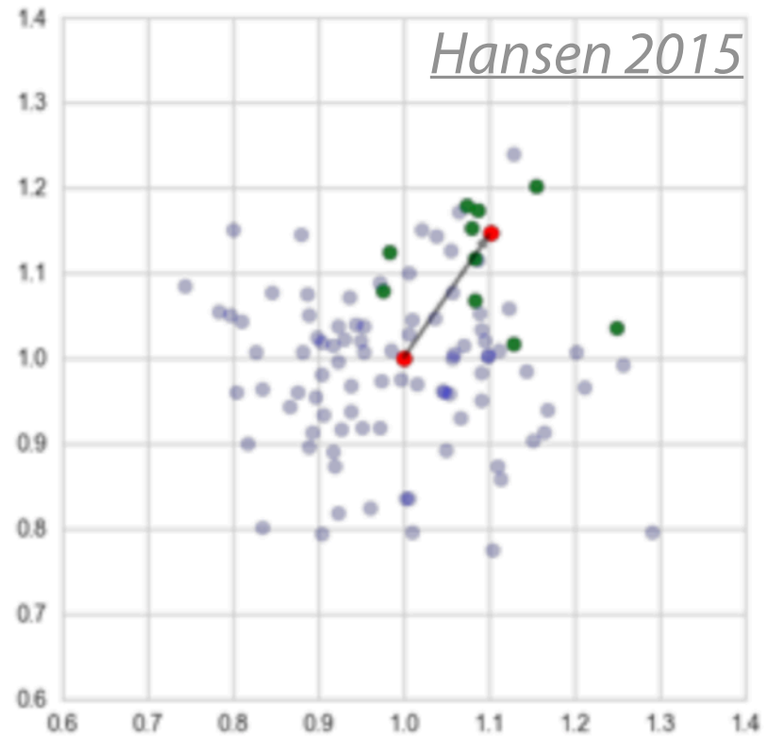
- Less sample efficient, but easy to parallelize, and adapts to changes
- Genetic programming *Olson, Moore 2016, 2019*
  - Mutations: add, mutate/tune, remove HP

- 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]

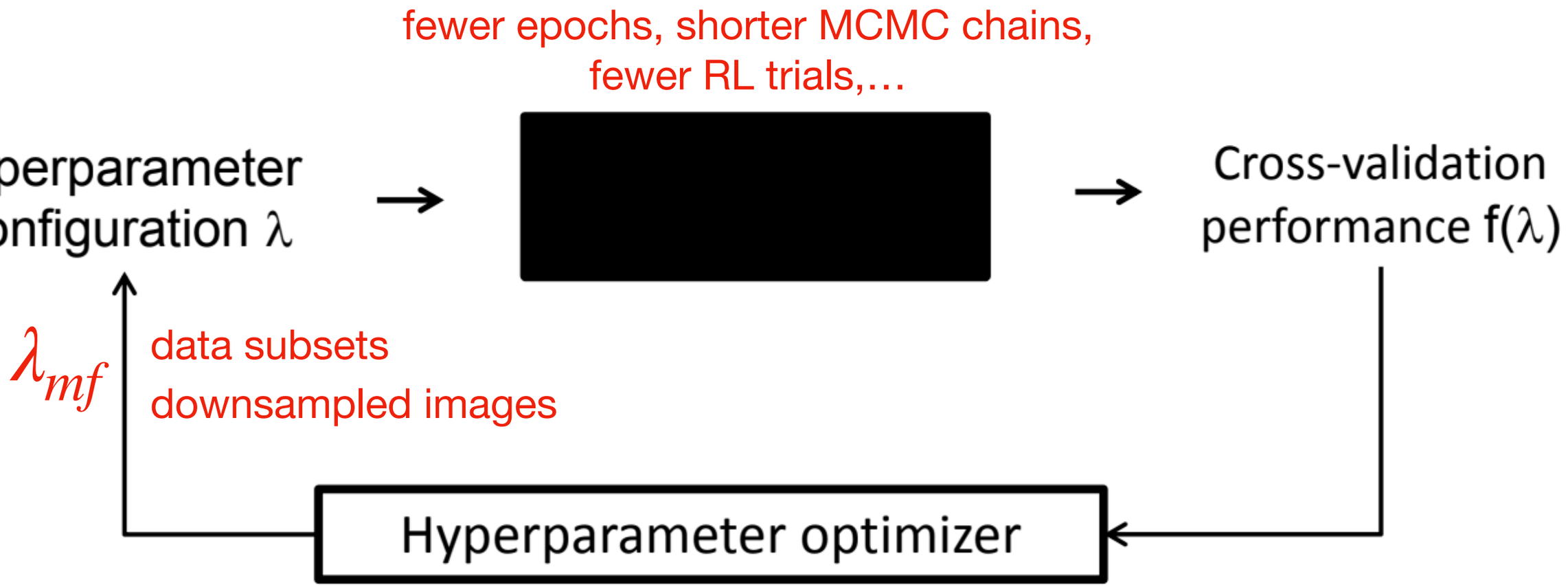


AutoML performance improvements

Making it practically useful (inspired by humans)

Multi-fidelity approaches

General techniques for cheap approximations of black-box

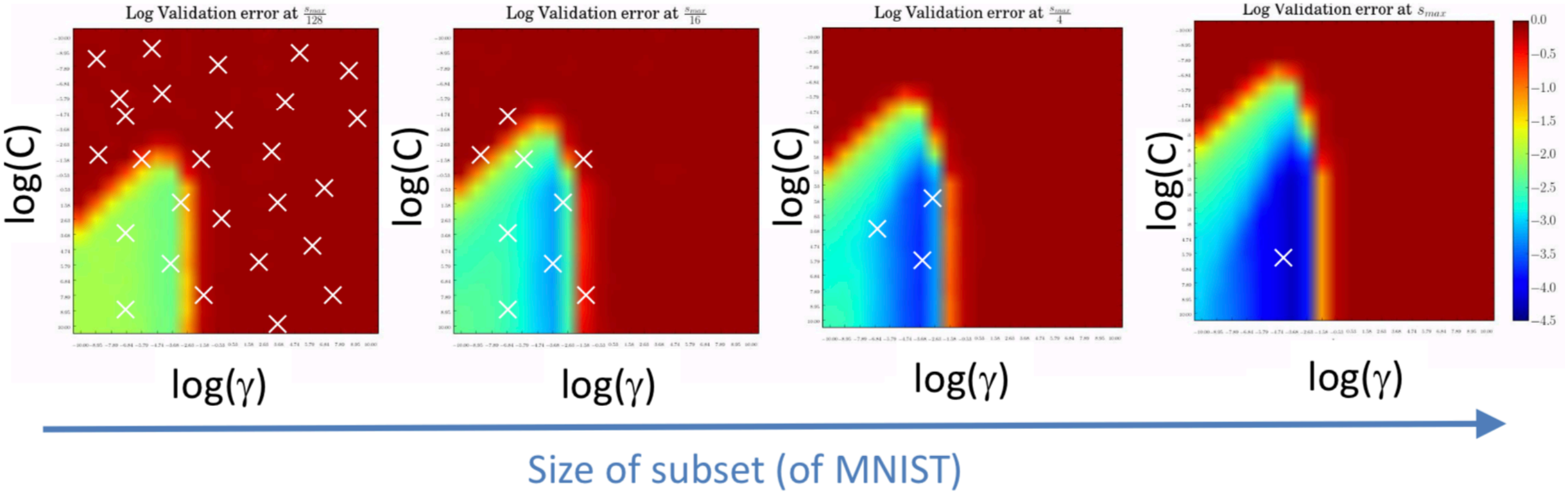


$$\lambda^* = \underset{\lambda \in \Lambda_{\mathcal{A}}}{\operatorname{argmin}} \, \mathbb{E}_{(D_{train}, D_{valid}) \sim \mathcal{D}} \mathcal{L}(\lambda, D_{train}, D_{test})$$

Multi-fidelity approaches

Cheap low-fidelity surrogates: train on small subsets, infer which regions may be interesting to evaluate in more depth

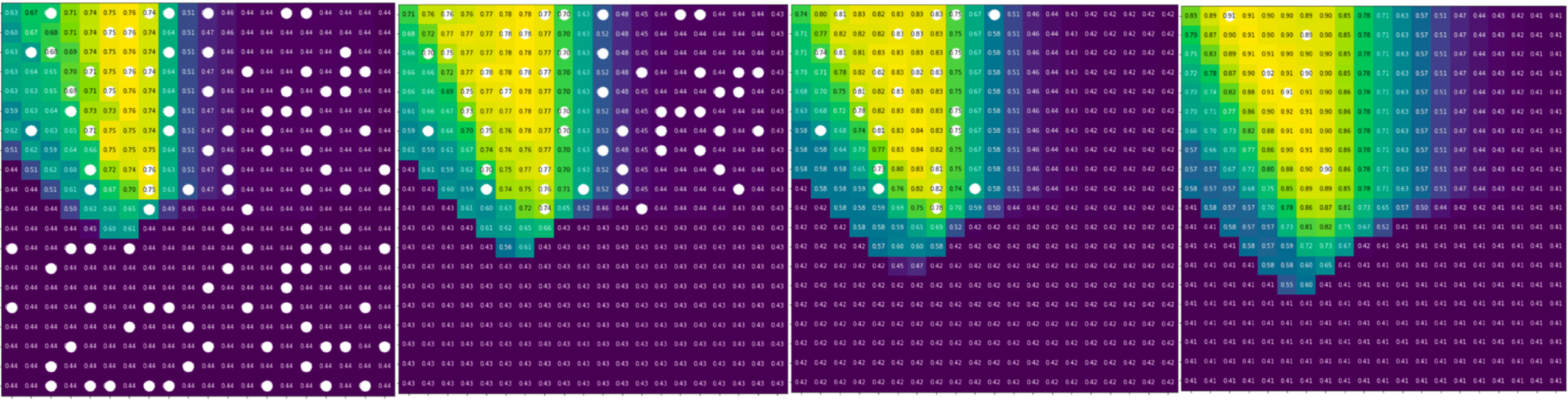
- Evaluate random samples on smallest set
- Update a Bayesian surrogate model
- Select fewer points based on surrogate, evaluate on more data, repeat



Multi-fidelity approaches

Successive halving:

- Randomly sample candidates and evaluate on a small data sample
- retrain the 50% best candidates on twice the data

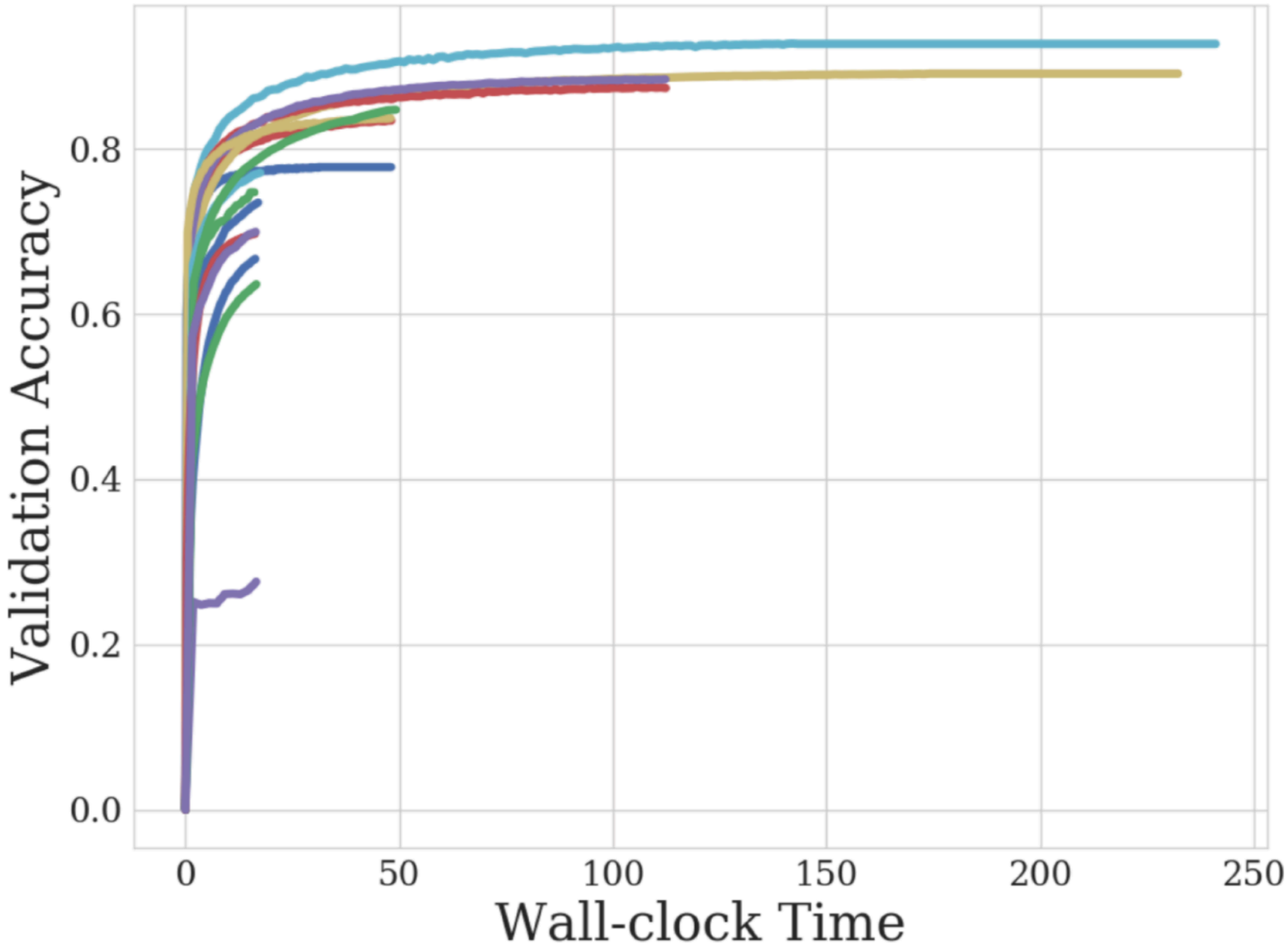


1/16 1/8 1/4 1/2

Multi-fidelity approaches

Successive halving:

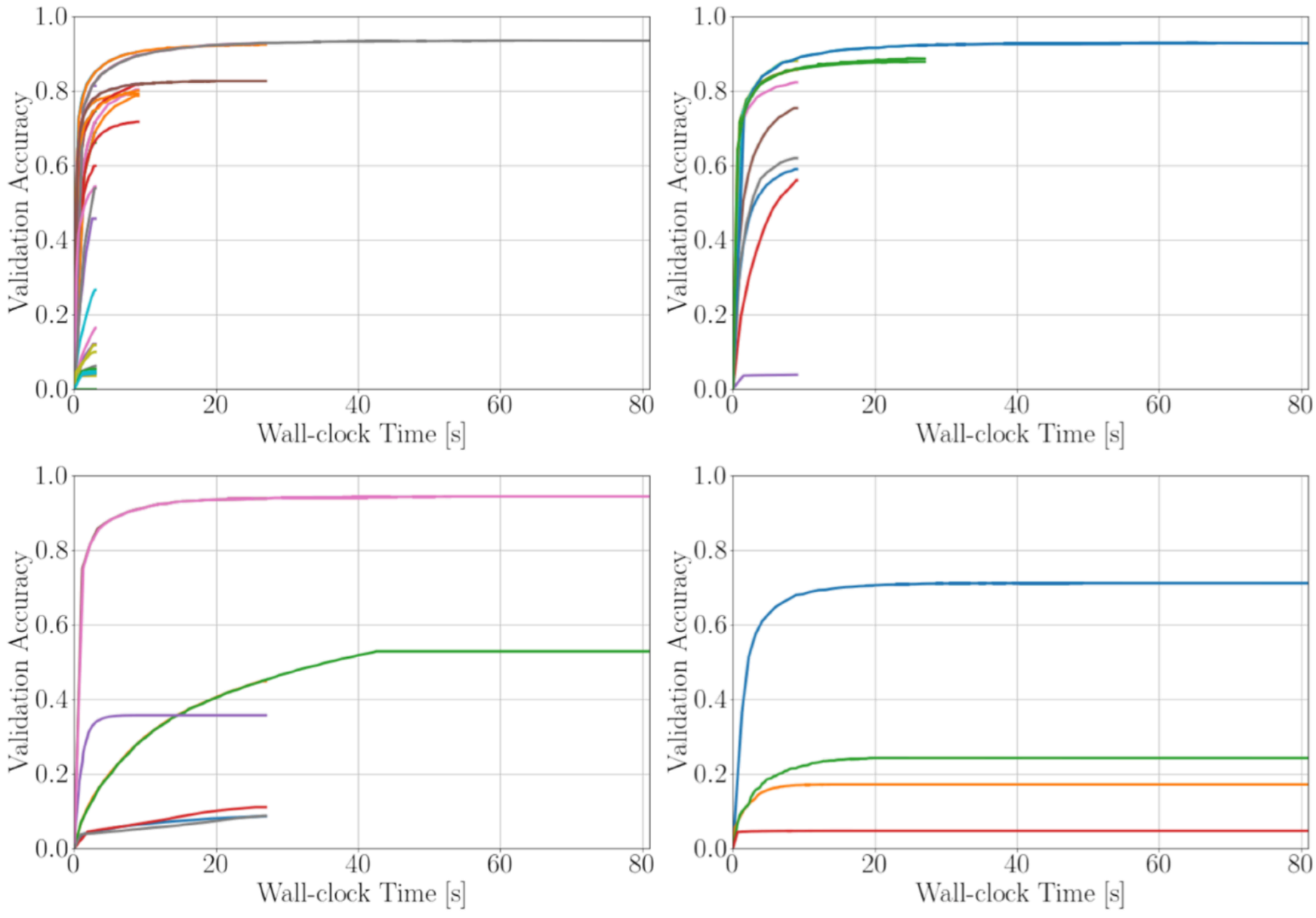
- Randomly sample candidates and evaluate on a small data sample
- retrain the best half candidates on twice the data



Multi-fidelity approaches

Hyperband: Repeated, decreasingly aggressive successive halving

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

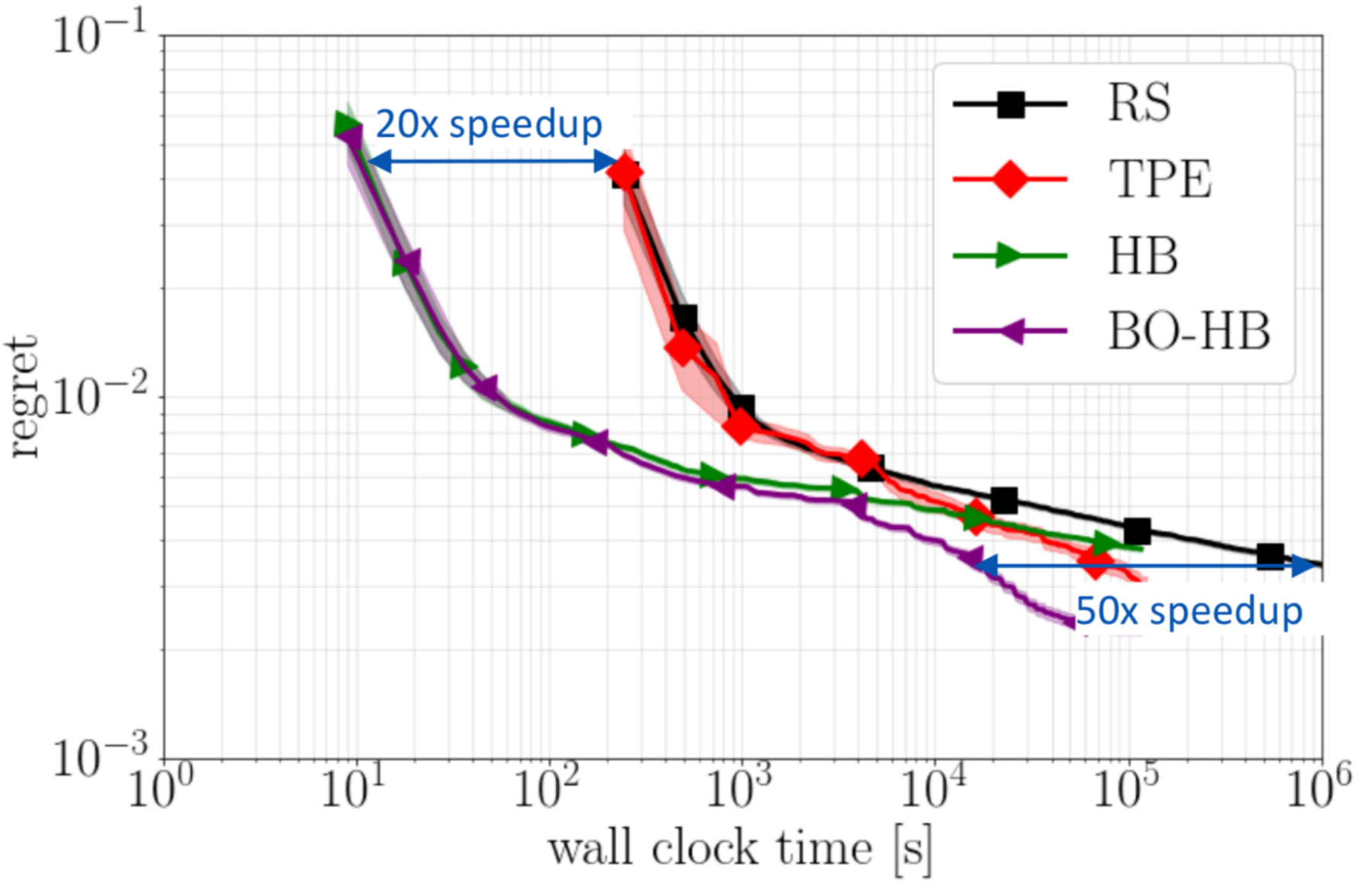


Falkner et al. 2018

Multi-fidelity approaches

Combined Bayesian Optimization and Hyperband (BO-HB)

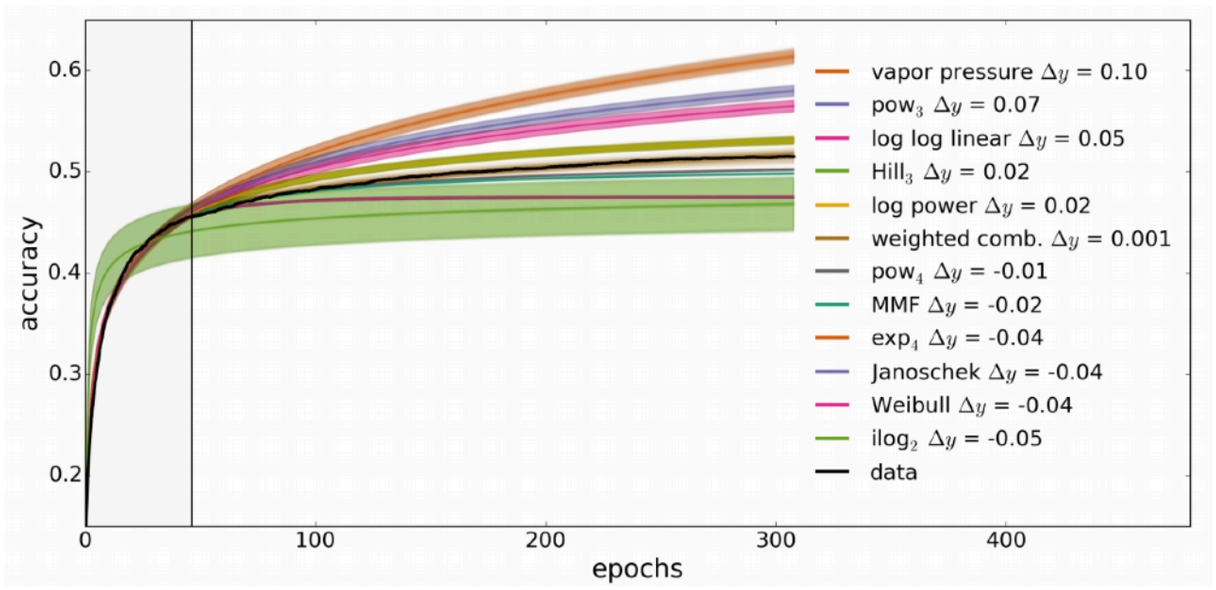
- BayesOpt: choose which configurations to evaluate
- Hyperband: allocate budgets more efficiently
- Strong anytime and final performance

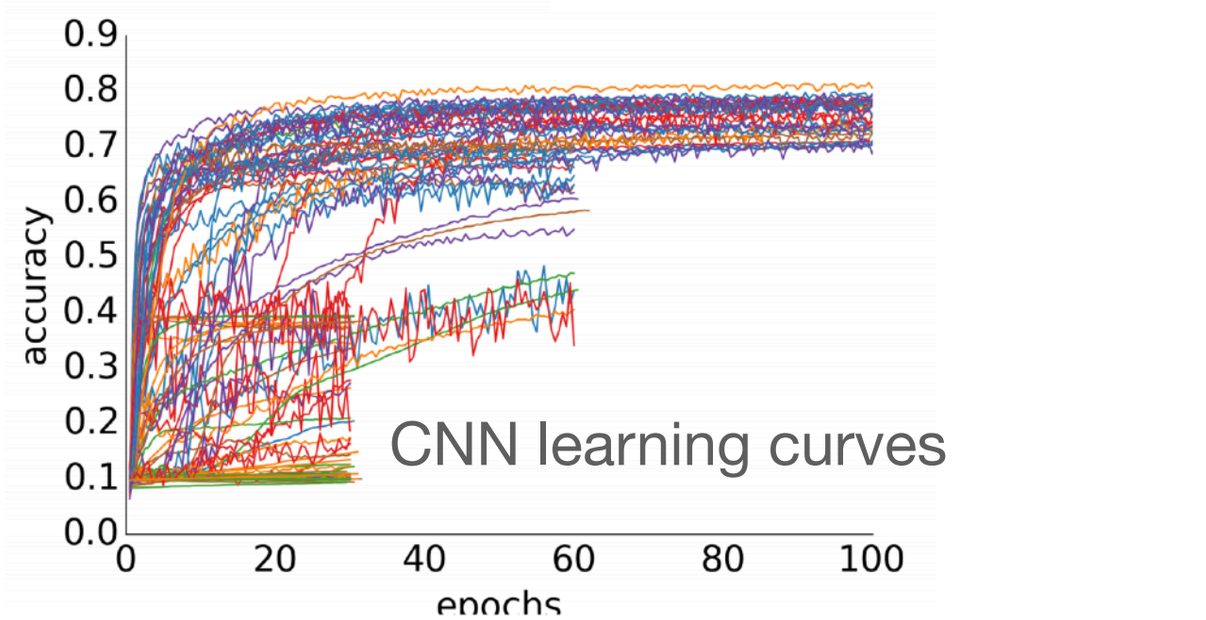


Early stopping

Learning curve prediction

- Model learning curves as parametric functions, fit *[Domhan et al 2015]*
- Train a Bayesian Neural Net to predict learning curves *[Klein et al 2017]*

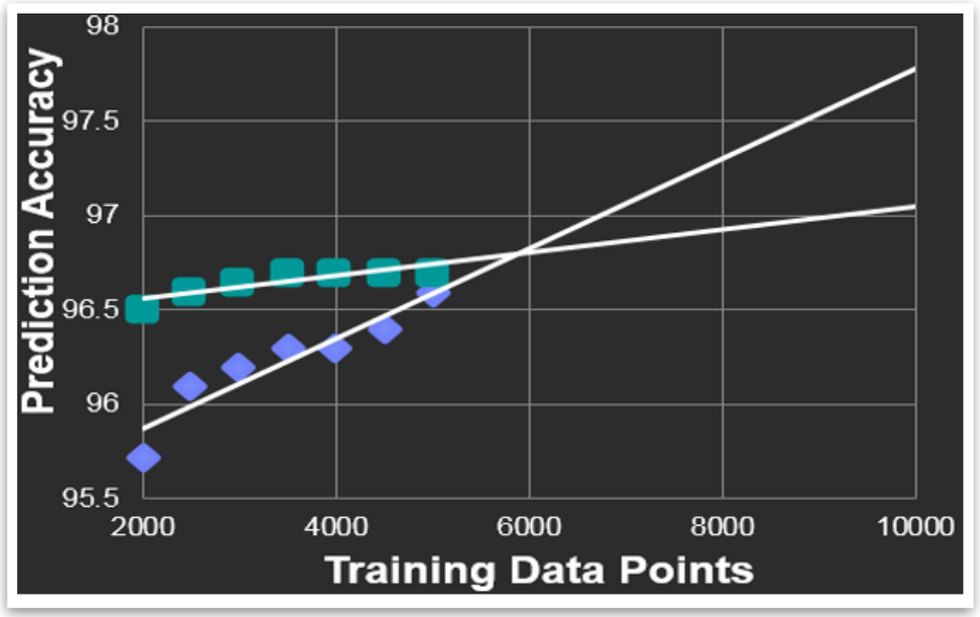




Data allocation using upper bounds (DAUB)

- Fit linear function through latest estimates
- Compute upper performance bound

Sabharwal et al. 2015



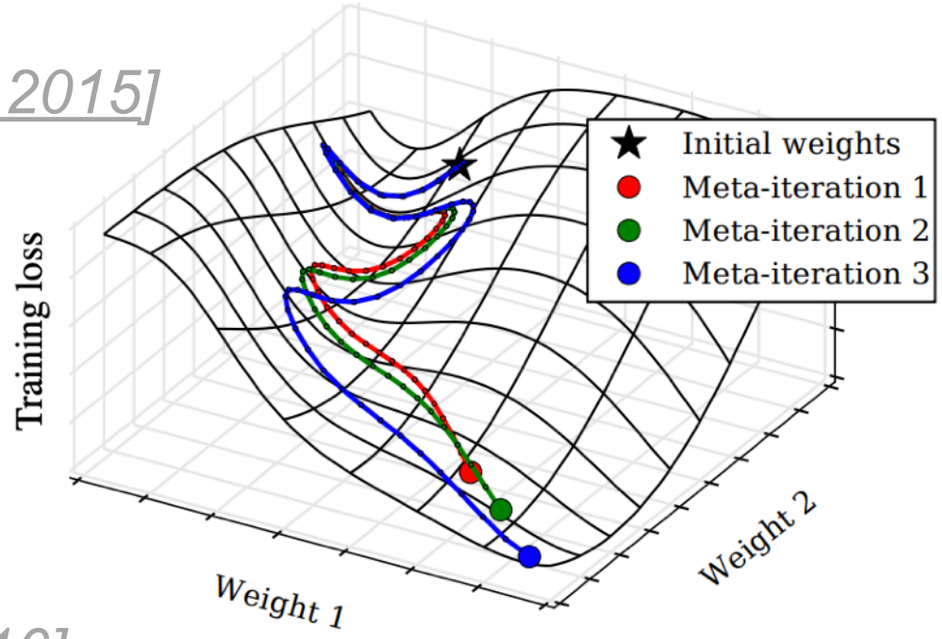
Hyperparameter gradient descent

Optimize neural network hyperparameters and weights simultaneously

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

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

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



- 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. $\nabla_{w}\mathcal{L}_{train}$

Meta-learning

Meta-learning can drastically speed up the architecture and hyperparameter search, in combination with the optimization techniques we just discussed

- Learn which hyperparameters are really important
- Learn which hyperparameter values should be tried first
- Learn which architectures will most likely work
- Learn how to clean and annotate data
- Learn which feature representations to use
- …

Thank you! To be continued…

Part 2: meta-learning

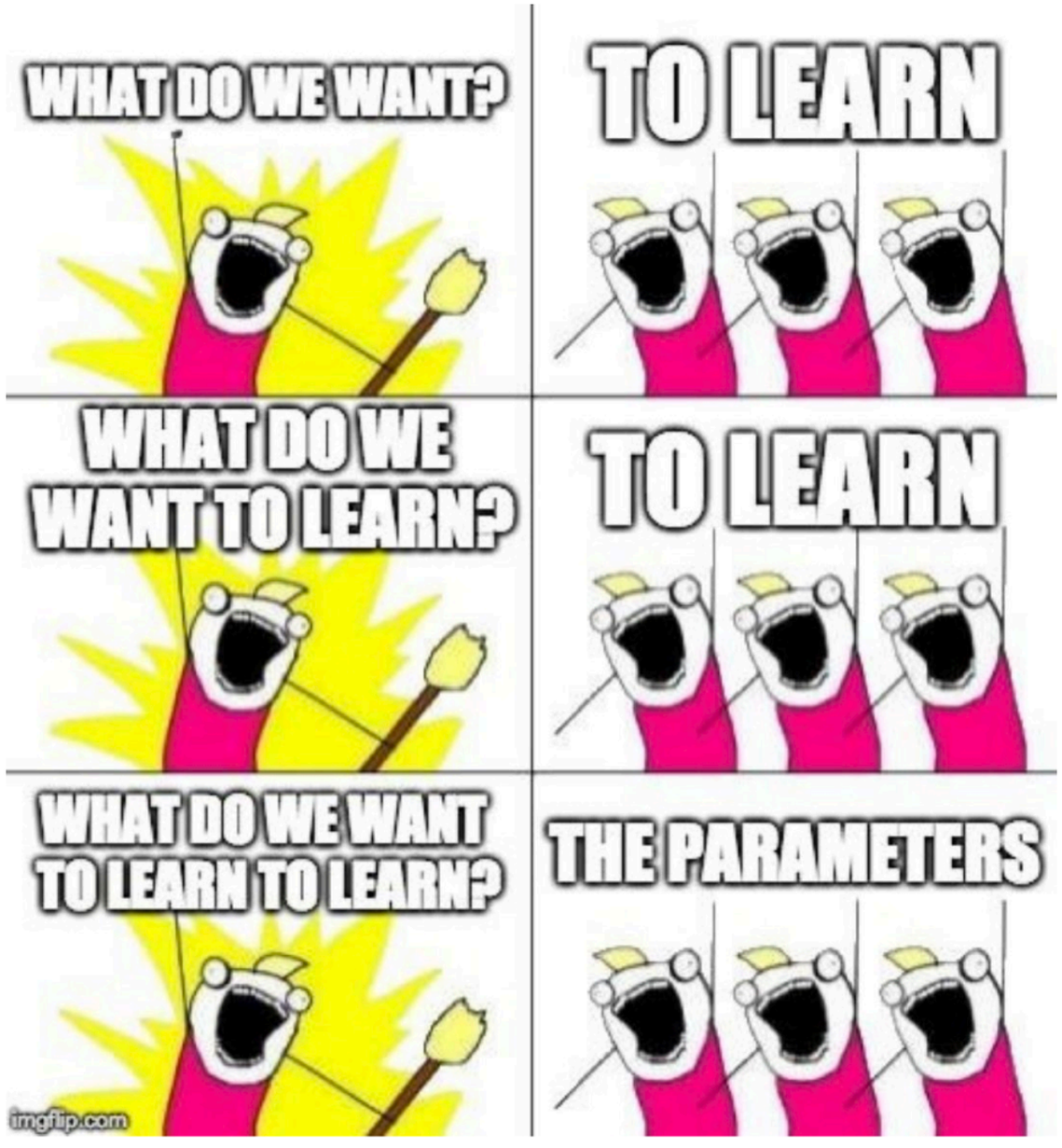


Image: Pesah et al. 2018