LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Deeplearning

Chapter 5: Model-Based Optimization

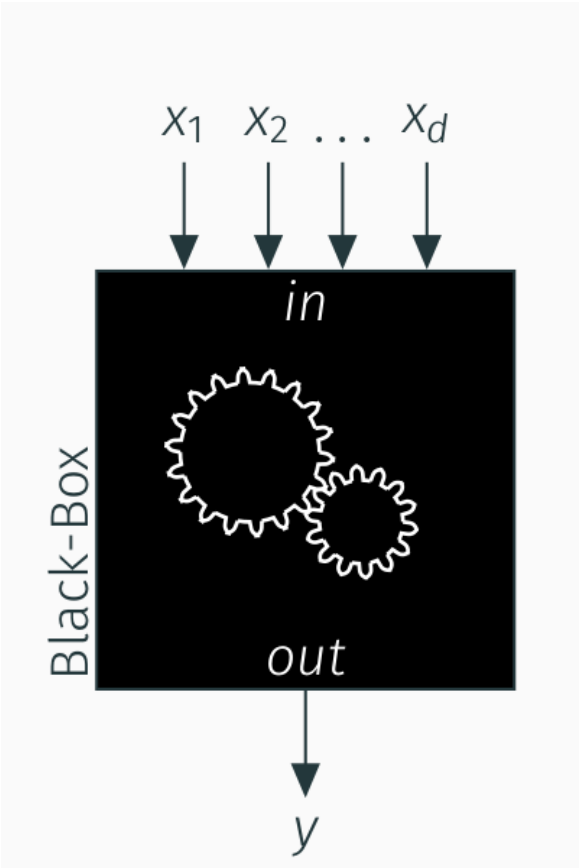
Bernd Bischl

Department of Statistics – LMU Munich

Winter term 2018

Sequential model-based optimization

EXPENSIVE BLACK-BOX OPTIMIZATION



$$y = f(\boldsymbol{x}) \,, \quad f : \mathbb{X} \to \mathbb{R} \tag{1}$$

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in \mathbb{X}} f(\boldsymbol{x}) \tag{2}$$

- y, target value
- $\boldsymbol{x} \in \mathbb{X} \subset \mathbb{R}^d$, domain
- $f(\boldsymbol{x})$ function with considerably long runtime
- Goal: Find optimum $\boldsymbol{x}^*$

SEQUENTIAL MODEL-BASED OPTIMIZATION

- Setting: Expensive black-box problem $f : x \rightarrow \mathbb{R} = min$!
- Classical problem: Computer simulation with a bunch of control parameters and performance output; or algorithmic performance on 1 or more problem instances; we often optimize ML pipelines
- Idea: Let's approximate f via regression!

SEQUENTIAL MODEL-BASED OPTIMIZATION

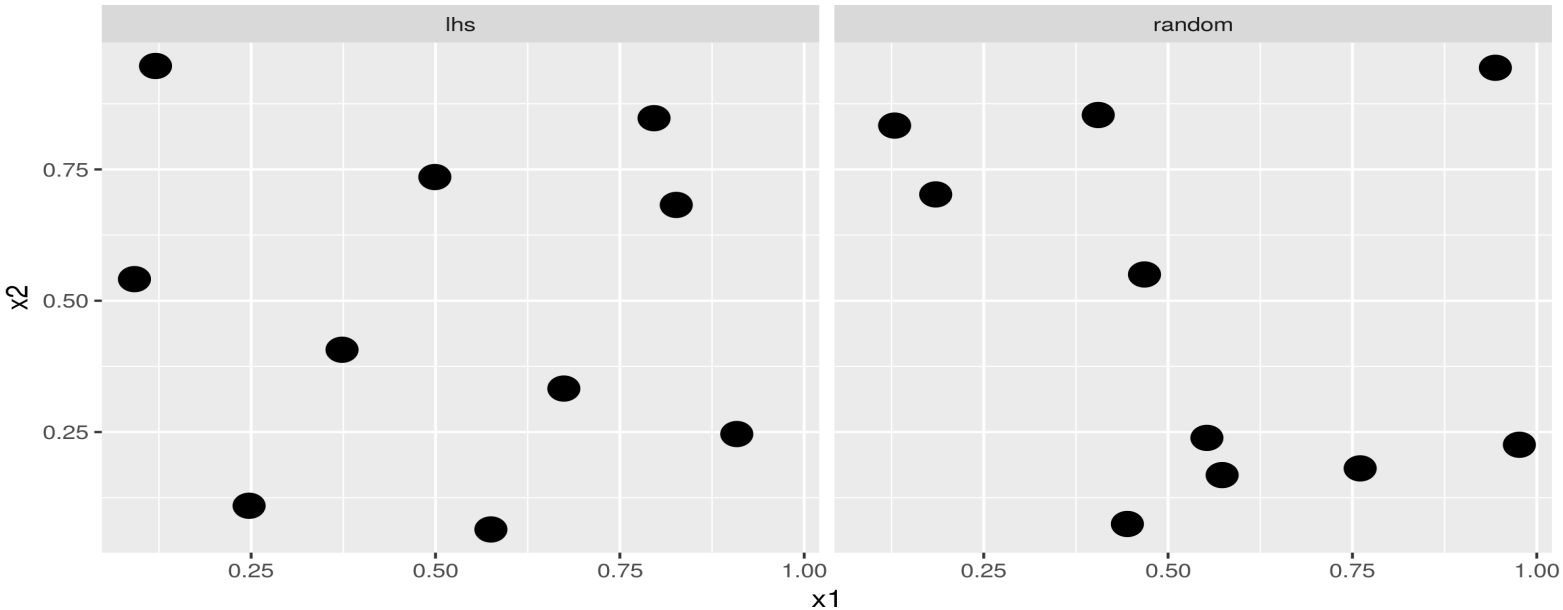
Generic MBO Pseudo Code

- Create initial space filling design and evaluate with f
- In each iteration:
  - Fit regression model on all evaluated points to predict $\hat{f}x$ and uncertainty $\hat{s}(x)$
  - Propose point via infill criterion

$$EI(x) \uparrow \iff \hat{f}(x) \downarrow \ \wedge \ \hat{s}(x) \uparrow$$

  - Evaluate proposed point and add to design
  - EGO proposes kriging (aka Gaussian Process) and EI *Jones 1998, Efficient Global Opt. of Exp. Black-Box Functions*

LATIN HYPERCUBE DESIGNS

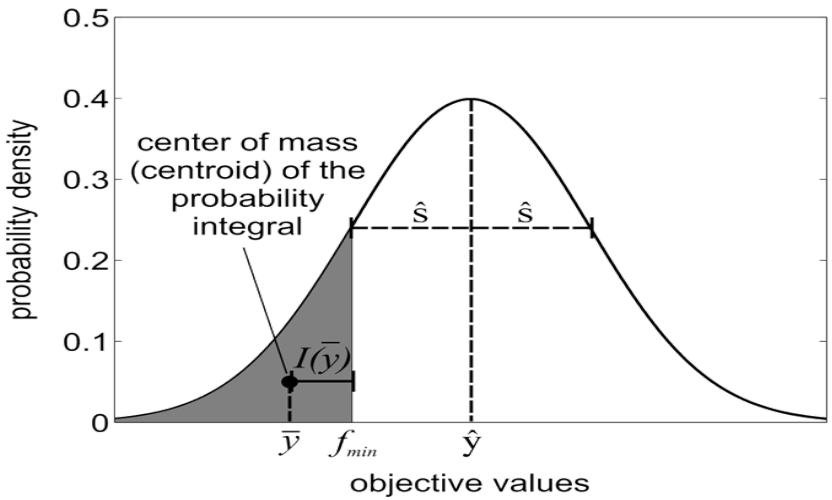


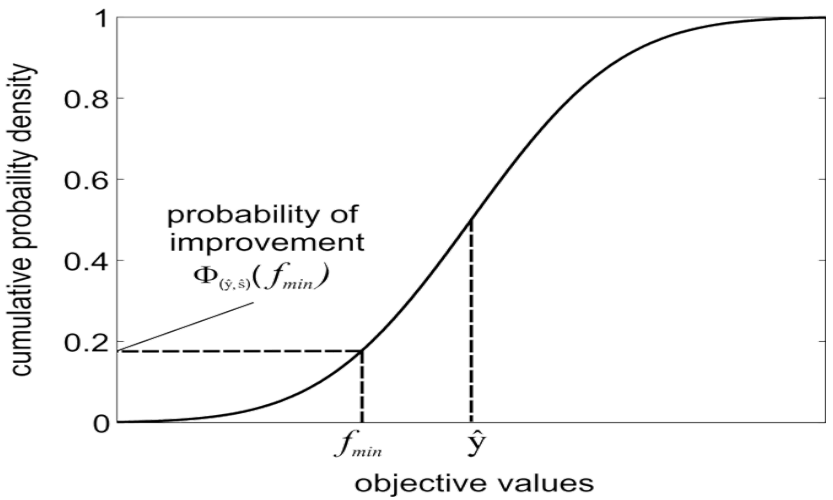
- Initial design to train first regression model
- Not too small, not too large
- LHS / maximin designs: Min dist between points is maximized
- But: Type of design usually has not the largest effect on MBO, and unequal distances between points could even be beneficial

INFILL CRITERIA: EXPECTED IMPROVEMENT

- Define improvement at x over best visited point with $y = f_{min}$ as random variable $I(x) = |f_{min} - Y(x)|^+$
- For kriging $Y(x) \sim N(\hat{f}(x), \hat{s}^2(x))$ (given $x = x$)
- Now define $EI(x) = E[I(x)|x = x]$
- Expectation is integral over normal density starting at f_{min}
- Alternative: Lower confidence bound (LCB) $\hat{f}(x) - \lambda\hat{s}(x)$

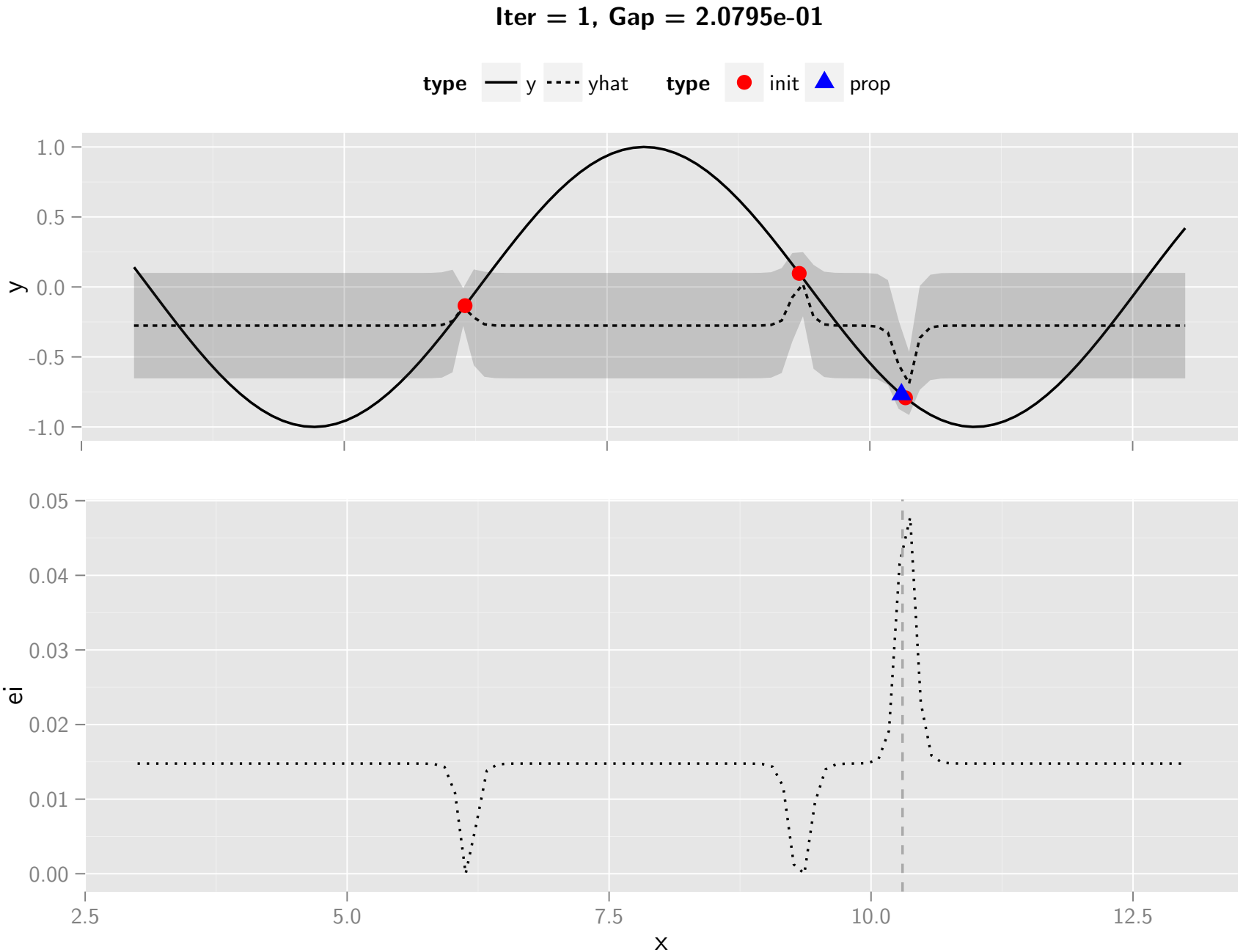
Result: $EI(x) = \left(f_{min} - \hat{f}(x)\right) \Phi\left(\frac{f_{min} - \hat{f}(x))}{\hat{s}(x)}\right) + \hat{s}(x)\phi\left(\frac{f_{min} - \hat{f}(x)}{\hat{s}(x)}\right)$

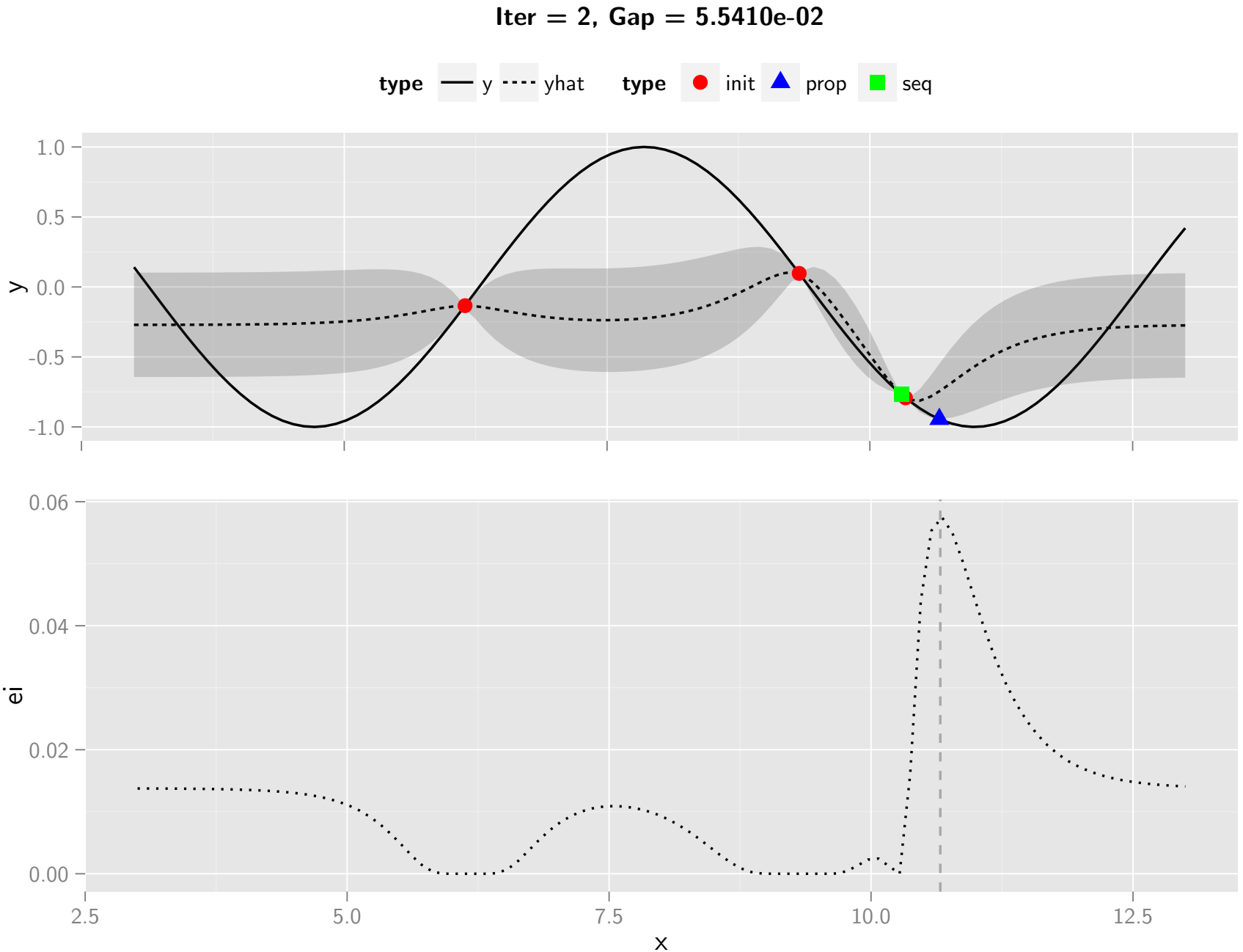


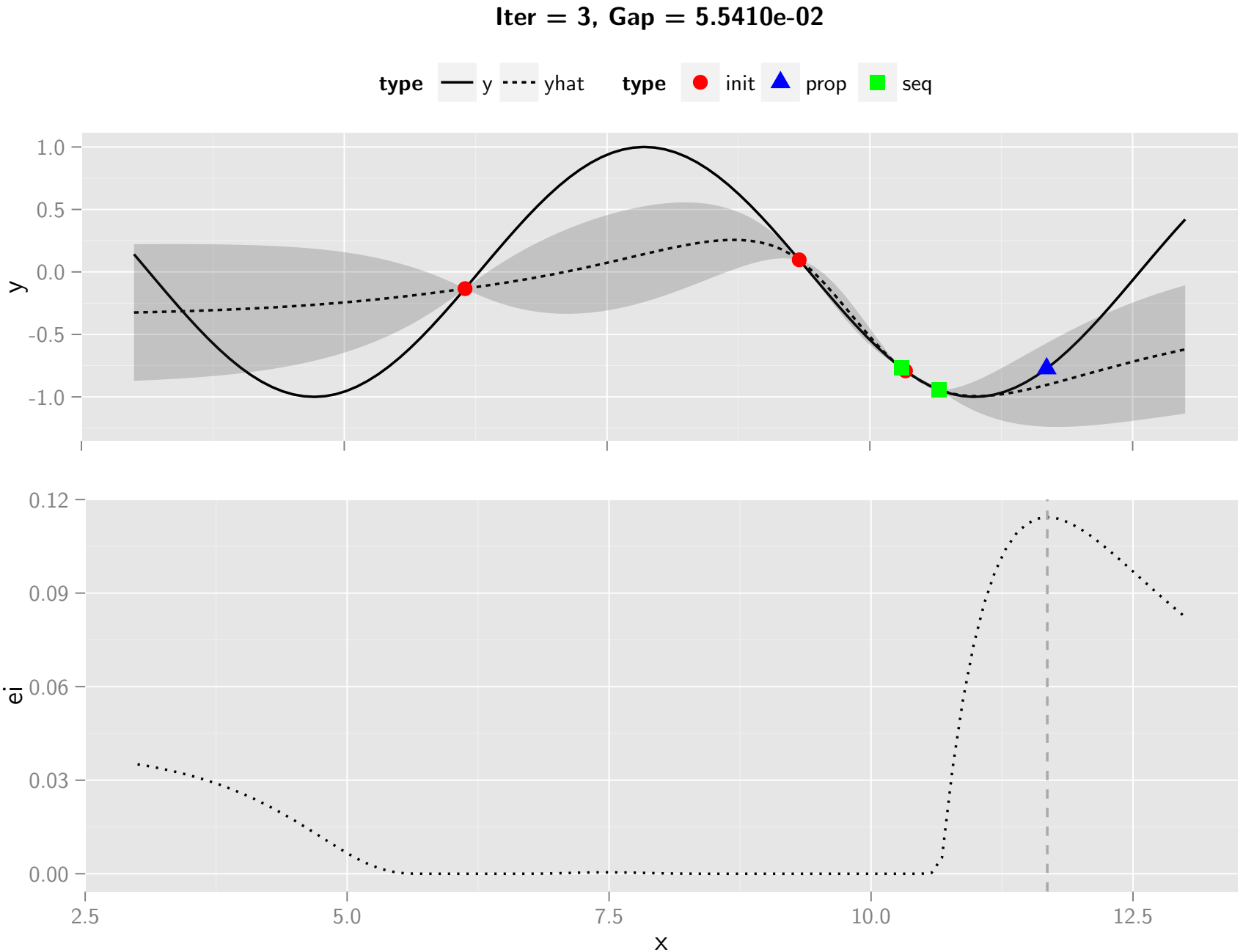


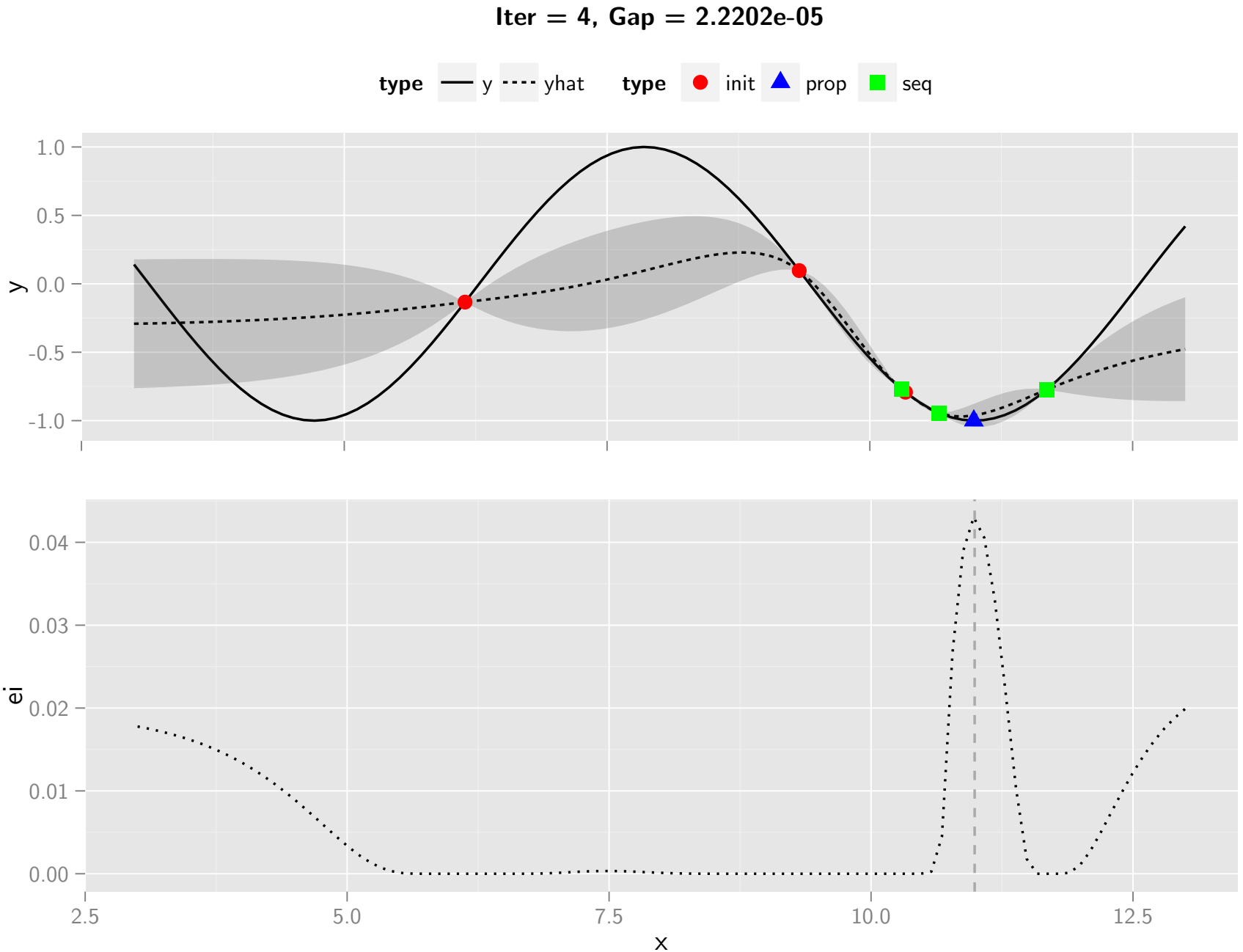
FOCUSSEARCH

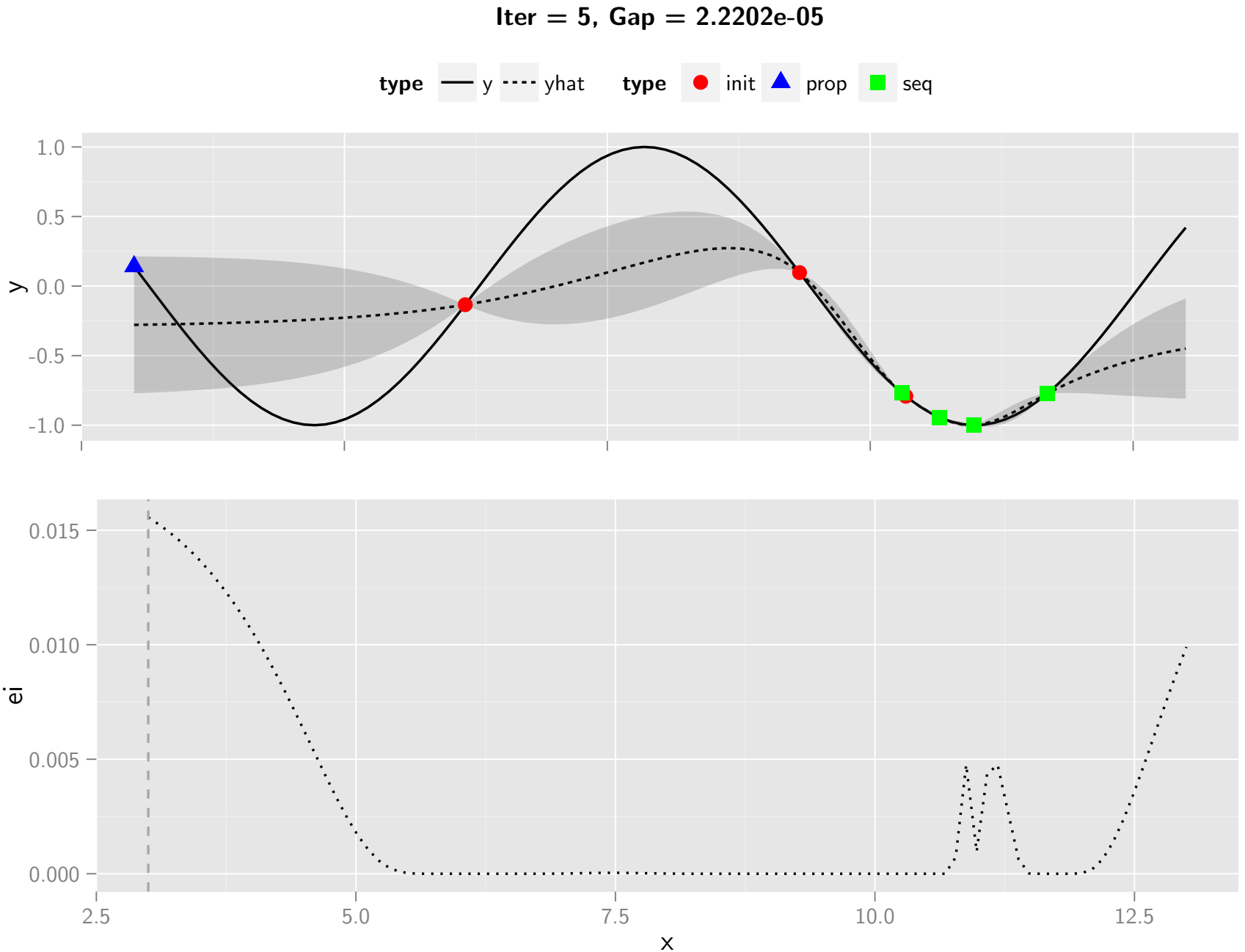
- EI optimization is multimodal and not that simple
- But objective is now cheap to evaluate
- Many different algorithms exist, from gradient-based methods with restarts to evolutionary algorithms
- We use an iterated, focusing random search coined “focus search”
- In each iteration a random search is performed
- We then shrink the constraints of the feasible region towards the best point in the current iteration (focusing) and iterate, to enforce local convergence
- Whole process is restarted a few times
- Works also for categorical and hierarchical params

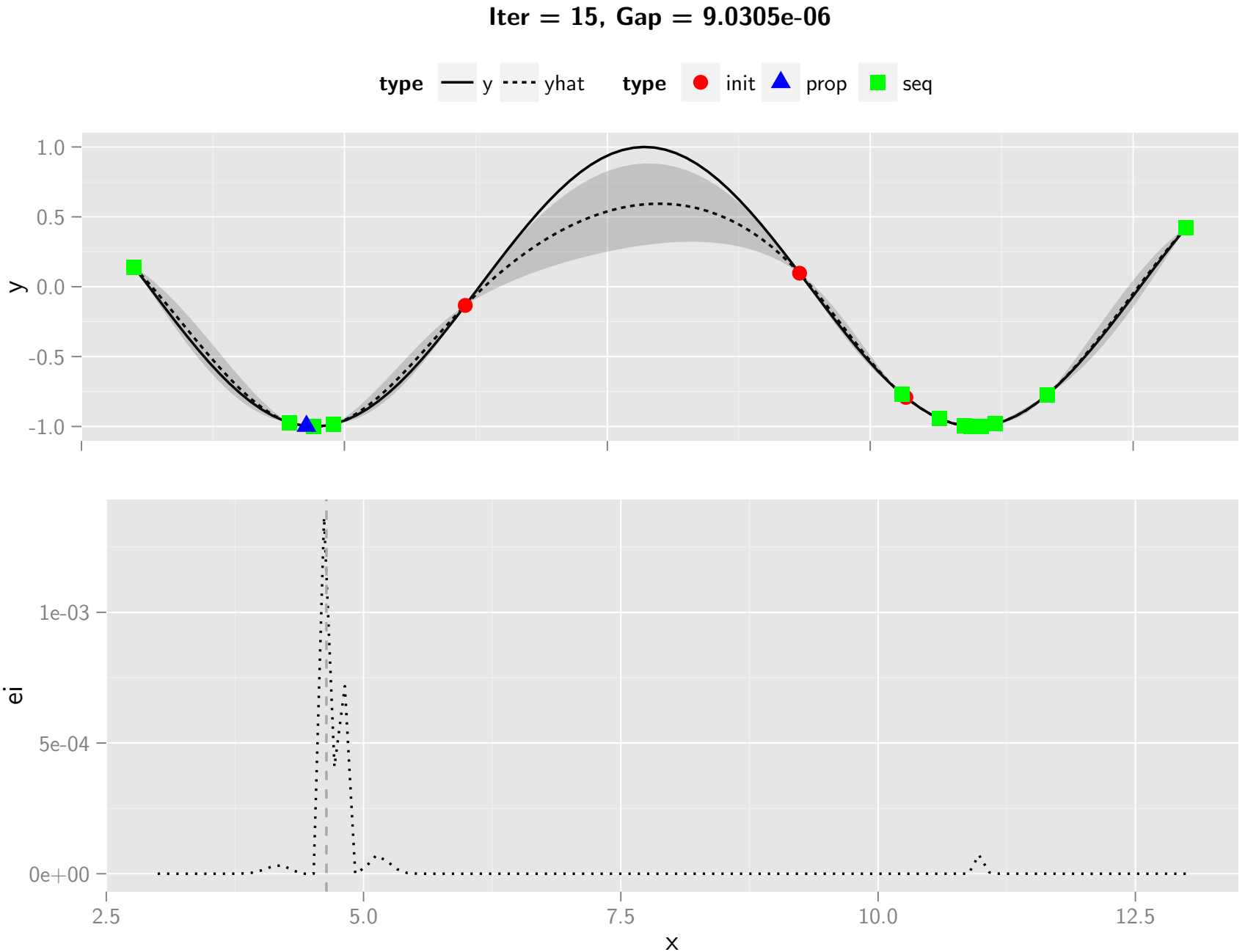

Iter = 1, Gap = 2.0795e-01
type
y
yhat
type
init
prop
1.0
0.5
0.0
-0.5
-1.0
y
0.05
0.04
0.03
0.02
0.01
0.00
ei
2.5
5.0
7.5
10.0
12.5
x


Iter = 2, Gap = 5.5410e-02
type
y
yhat
type
init
prop
seq
1.0
0.5
0.0
-0.5
-1.0
y
0.06
0.04
0.02
0.00
ei
2.5
5.0
7.5
10.0
12.5
x


Iter = 3, Gap = 5.5410e-02
type
y
yhat
type
init
prop
seq
1.0
0.5
0.0
-0.5
-1.0
y
0.12
0.09
0.06
0.03
0.00
ei
2.5
5.0
7.5
10.0
12.5
x

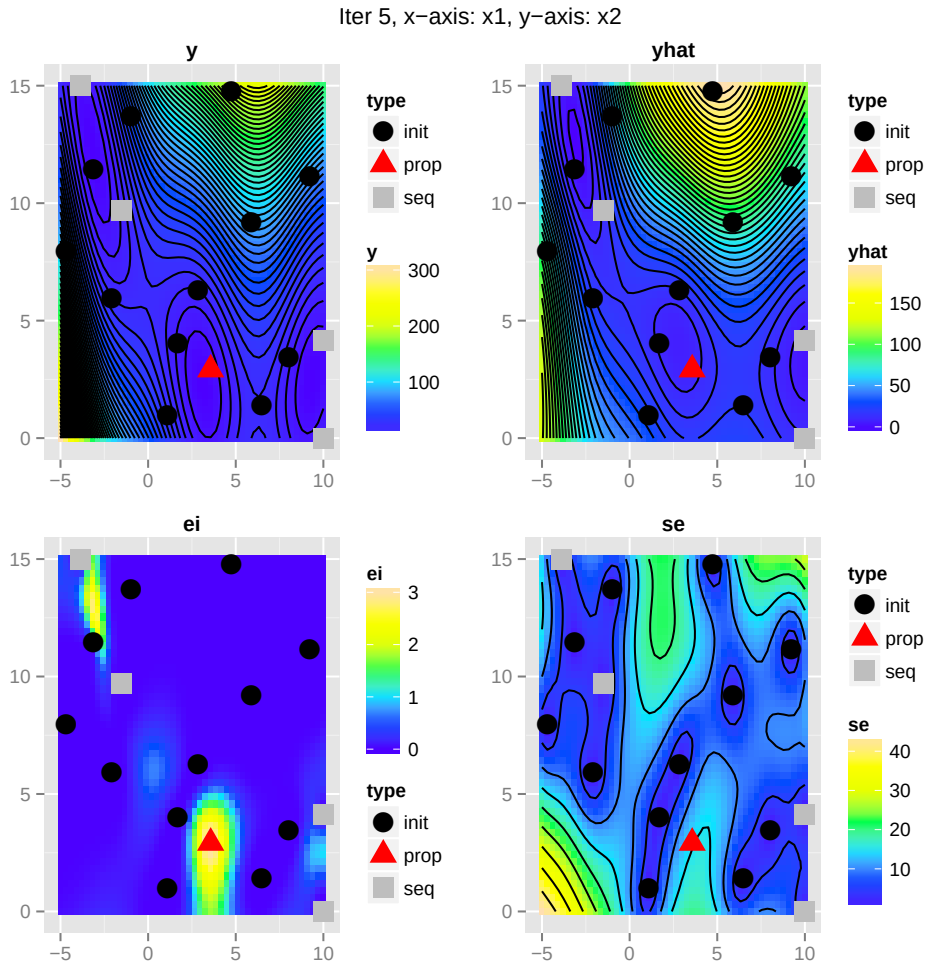

Iter = 4, Gap = 2.2202e-05
type
y
yhat
type
init
prop
seq
1.0
0.5
0.0
-0.5
-1.0
y
0.04
0.03
0.02
0.01
0.00
ei
2.5
5.0
7.5
10.0
12.5
x


Iter = 5, Gap = 2.2202e-05
type
y
yhat
type
init
prop
seq
1.0
0.5
0.0
-0.5
-1.0
y
0.015
0.010
0.005
0.000
ei
2.5
5.0
7.5
10.0
12.5
x


Iter = 15, Gap = 9.0305e-06
type
y
yhat
type
init
prop
seq
1.0
0.5
0.0
-0.5
-1.0
y
1e-03
5e-04
0e+00
ei
2.5
5.0
7.5
10.0
12.5
x

MLRMBO: MODEL-BASED OPTIMIZATION TOOLBOX

- Any regression from mlr
- Arbtritrary infill
- Single - or multi-crit
- Multi-point proposal
- Via parallelMap and batchtools runs on many parallel backends and clusters
- Algorithm configuration
- Active research

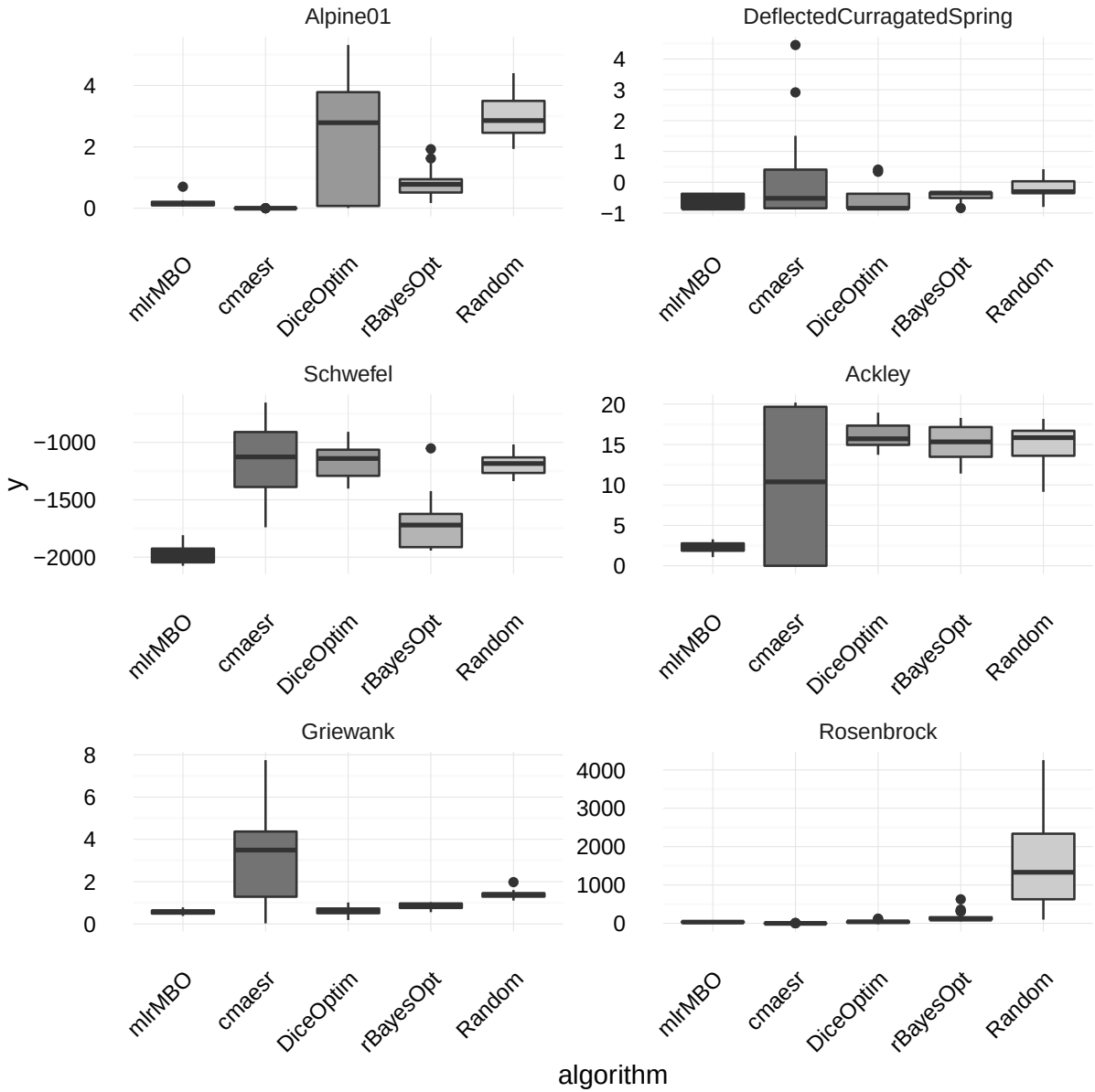


- mlr: `https://github.com/mlr-org/mlr`
- mlrMBO: `https://github.com/mlr-org/mlrMBO`
- mlrMBO Paper on arXiv (under review) `https://arxiv.org/abs/1703.03373`

BENCHMARK MBO ON ARTIFICIAL TEST FUNCTIONS

- Comparison of mlrMBO on multiple different test functions
  - Multimodal
  - Smooth
  - Fully numeric
  - *Well known*
- We use GPs with
  - LCB with $\lambda = 1$
  - Focussearch
  - 200 iterations
  - 25 point initial design, created by LHS sampling
- Comparison with
  - Random search
  - CMAES
  - other MBO implementations in R

MBO GP VS. COMPETITORS IN 5D



Interesting Challenges

CHALLENGE: THE CORRECT SURROGATE?

- GPs are very much tailored to what we want to do, due to their spatial structure in the kernel and the uncertainty estimator.
- But GPs are rather slow. And (fortunately) due to parallization (or speed-up tricks like subsampling) we have more design points to train on.
- Categorical features are also a problem in GPs (although methods exist, usually by changing the kernel)
- Random Forests handle categorical features nicely, are much faster. But they don't rely on a spatial kernel and the uncertainty estimation is much more heuristic / may not represent what we want.

CHALLENGE: TIME HETEROGENEITY

- Complex configuration spaces across many algorithms results in vastly different runtimes in design points.
- Actually just the RBF-SVM tuning can result in very different runtimes.
- We don't care how many points we evaluate, we care about total walltime of the configuration.
- The option to subsample further complicates things.
- Parallelization further complicates things.
- Option: Estimate runtime as well with a surrogate, integrate it into acquisition function.

ML Model Selection and Hyperparameter Optimization

AUTOMATIC MODEL SELECTION

Prior approaches:

- Looking for the silver bullet model
  $\rightsquigarrow$ Failure
- Exhaustive benchmarking / search
  $\rightsquigarrow$ Very expensive, often contradicting results
- Meta-Learning:
  $\rightsquigarrow$ Good meta-features are hard to construct
  $\rightsquigarrow$ IMHO: Gets more interesting when combined with SMBO

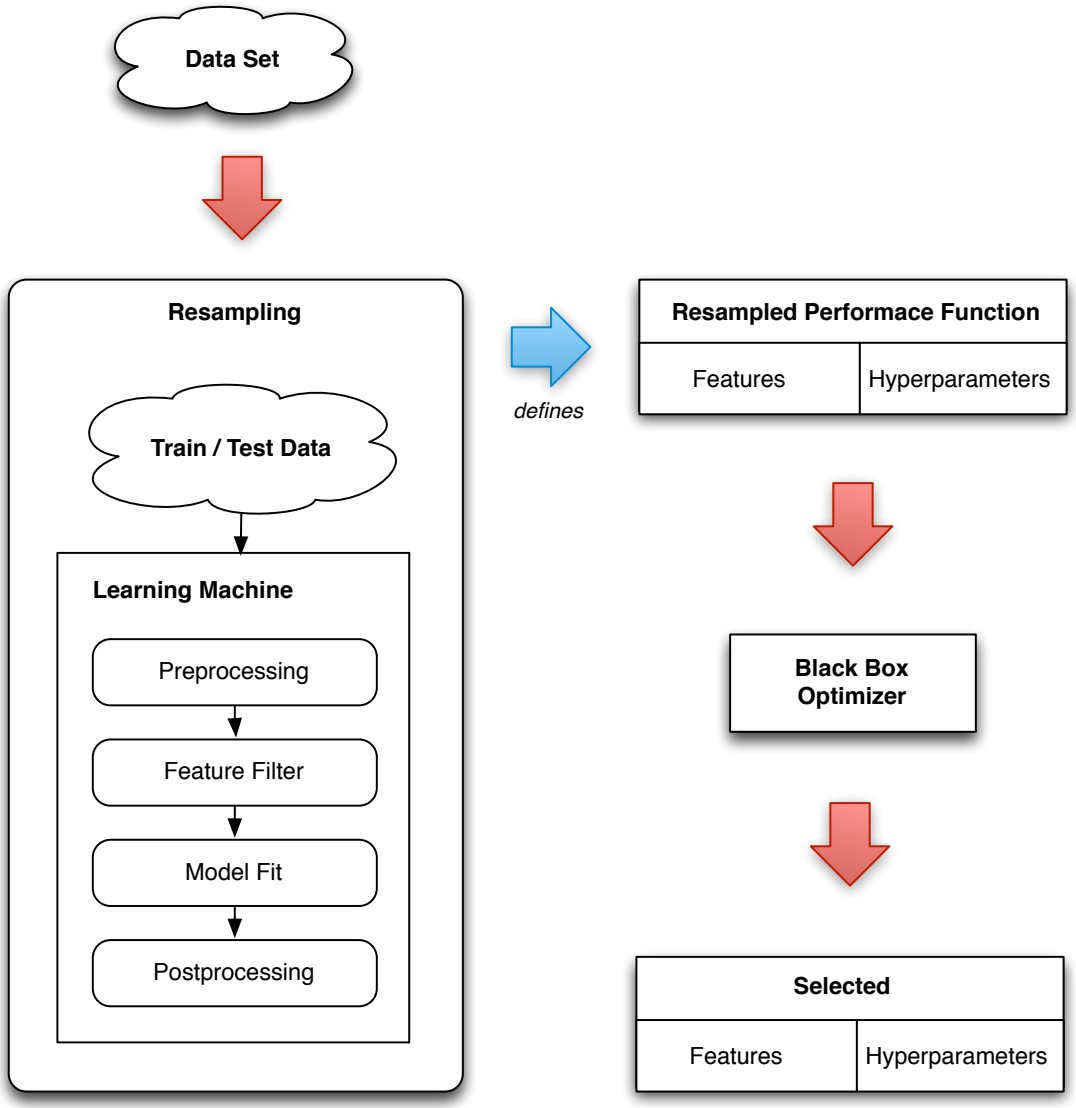
Goal for AutoML:

- Data dependent
- Automatic
- Include every relevant modeling decision
- Efficient
- Learn on the model-settings level!

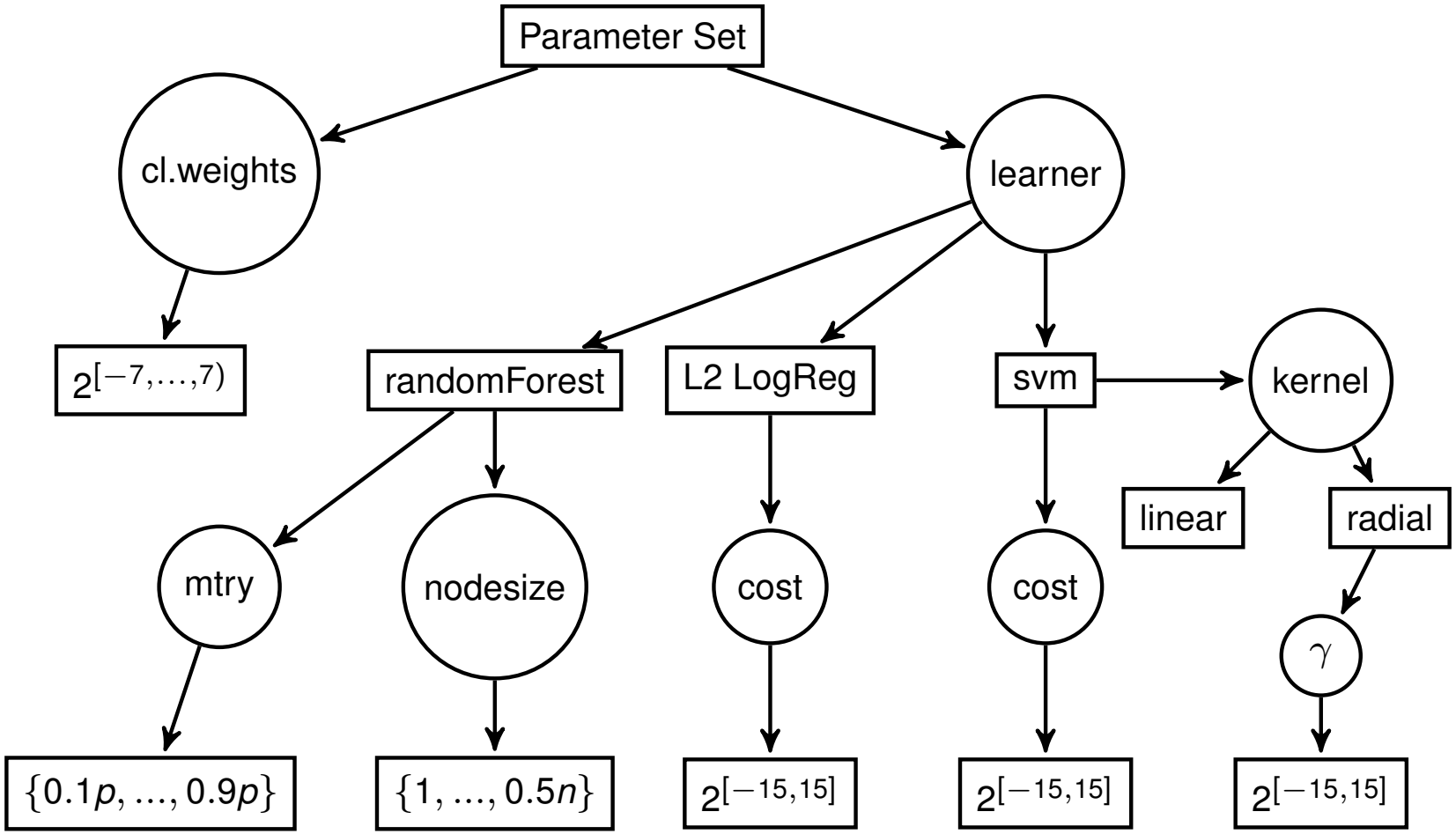
FROM NORMAL SMBO TO HYPERARAMETER TUNING

- Objective function is resampled performance measure
- Parameter space $\theta \in \Theta$
  might be discrete and dependent / hierarchical
- No derivative for $f(\cdot, \theta)$, black-box
- Objective is stochastic / noisy
- Objective is expensive to evaluate
- In general we face a problem of algorithm configuration:
- $\rightsquigarrow$ Usual approaches: racing or model-based / bayesian optimization

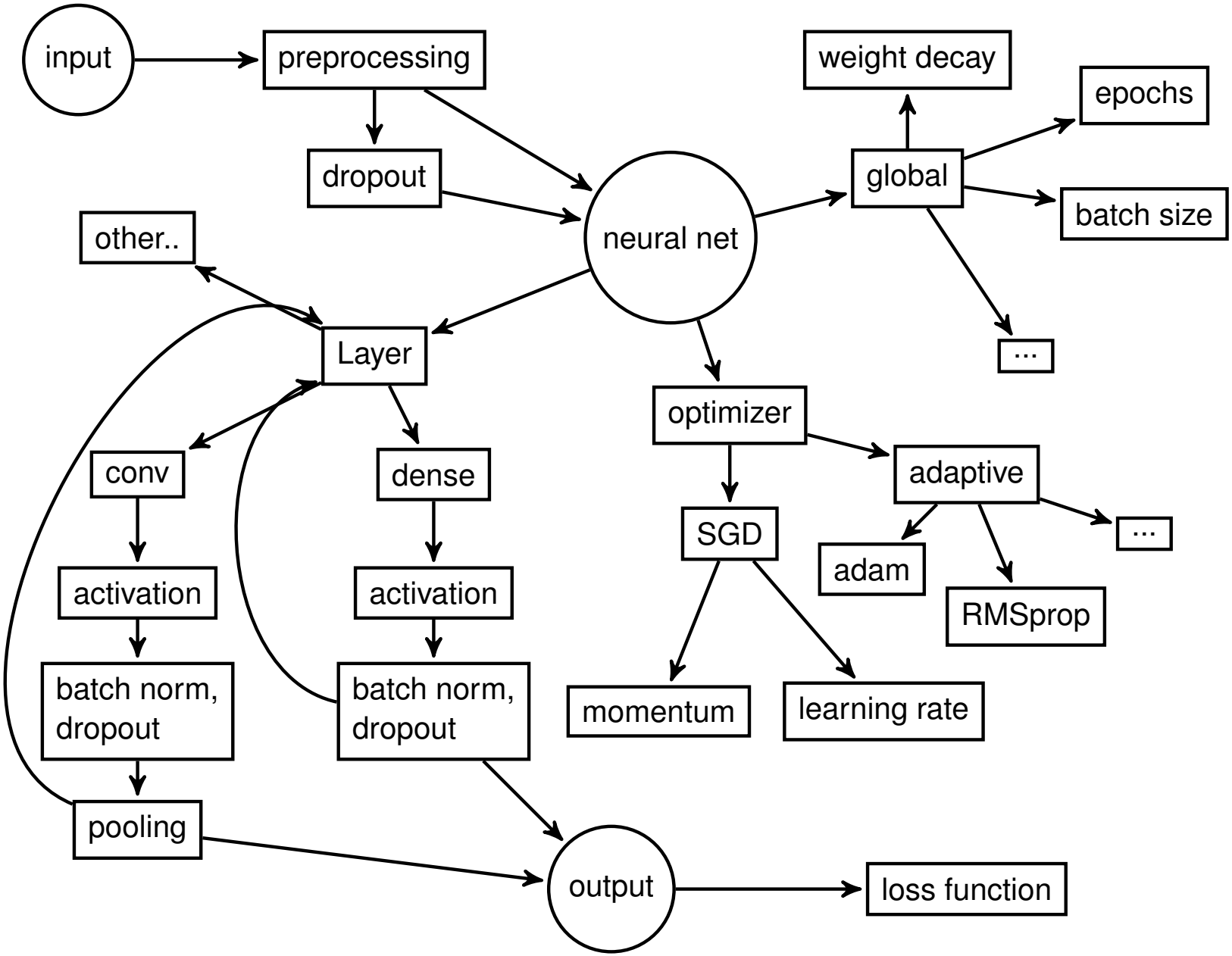
FROM NORMAL SMBO TO HYPERARAMETER TUNING



COMPLEX PARAMETER SPACE



COMPLEX PARAMETER SPACE



FROM NORMAL SMBO TO HYPERARAMETER TUNING

- Initial design: LHS principle can be extended, or just use random
- Focus search: Can be (easily) extended, as it is based on random search. To zoom in for categorical parameters we randomly drop a category for each param which is not present in the currently best configuration.

- Few approaches for GPs with categorical params exist (usually with new covar kernels), not very established
- Alternative: Random regression forest (mlrMBO, SMAC)
- Estimate uncertainty / confidence interval for mean response by efficient bootstrap technique[1], or jackknife, so we can define $EI(x)$ for the RF
- Dependent params in mlrMBO: Imputation
- Many of the current techniques to handle these problems are (from a theoretical standpoint) somewhat crude

HYPERPARAMETER TUNING

- Still common practice: grid search
  Simple example for a neural network with one conv and two dense layers (mnist):
  - dropout(input) $\in (0.1, 0.9)$
  - momentum $\in (0, 0.99)$
  - Evaluate a grid of $10^2 = 100$ combinations
- Bad beacause:
  - optimum might be "off the grid"
  - lots of evaluations in bad areas
  - lots of costy evaluations
- How bad?

MXNET IN MLR

```
# libs (currently mxnet is only available in the mxnet branch)
devtools::install_github("mlr-org/mlr", force = TRUE, ref = "mxnet")
require(mlr)
require(mlrMBO)
require(gridExtra)

# learner (note that mlr generates the output layer automatically)
lrn = makeLearner("classif.mxff",
  layers = 2, num.layer1 = 30, num.layer2 = 128,
  act1 = "relu", act2 = "relu", act.out = "softmax",
  conv.layer1 = TRUE, conv.data.shape = c(28,28),
  conv.kernel1 = c(2,2), conv.stride1 = c(2,2),
  pool.kernel1 = c(2,2), pool.stride1 = c(2,2),
  begin.round = 1, num.round = 100, array.batch.size = 128)
```

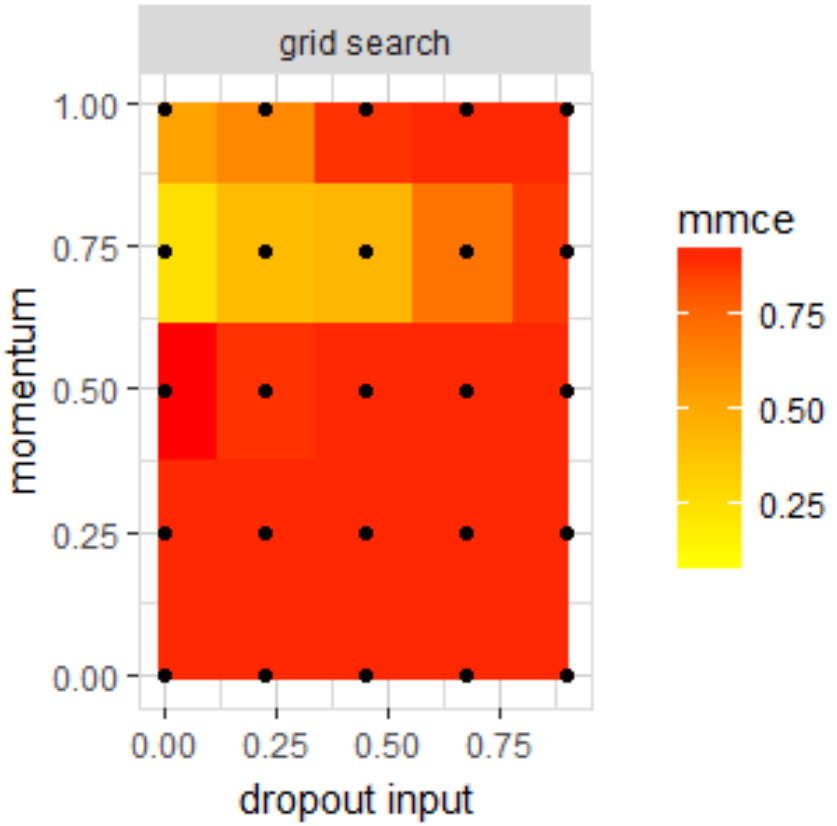
MXNET IN MLR

```
# task
task = makeClassifTask(data = mnist, target = "label")

# param set
ps = makeParamSet(
  makeNumericParam(id = "dropout.input", lower = 0.1, upper = 0.9),
  makeNumericParam(id = "momentum", lower = 0, upper = 0.99))

# make a grid search
grid.ctrl = makeTuneControlGrid(resolution = 5)
rdesc = makeResampleDesc("CV", iters = 3)
res.grid = tuneParams(lrn, task, rdesc, par.set = ps,
  control = grid.ctrl, show.info = FALSE)
op.df.gs = as.data.frame(res.grid$opt.path)
```

HYPERPARAMETER TUNING



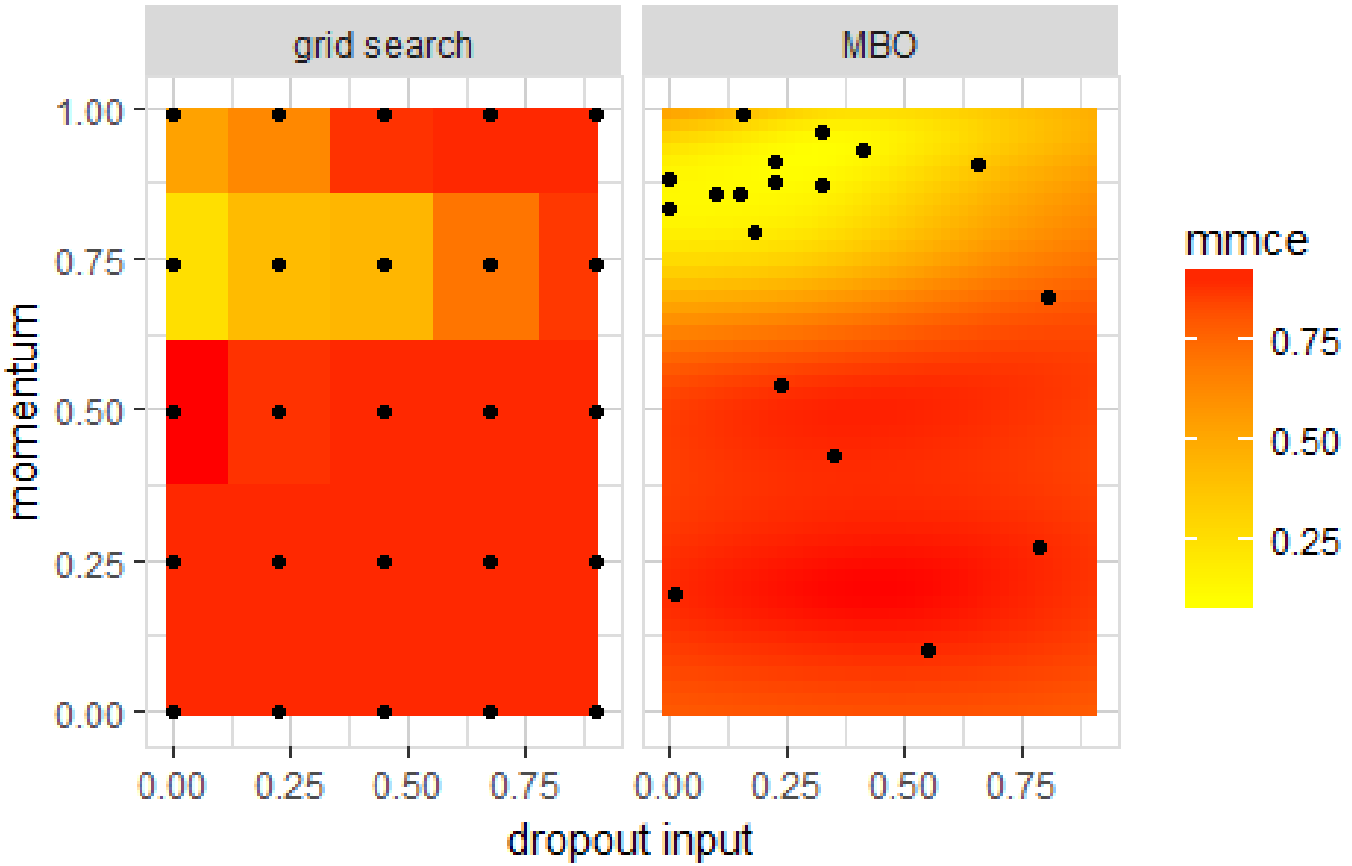
- Because of budget restrictions grid might even be smaller!
- Unpromising area quite big!
- Lots of costly evaluations!

MXNET IN MLR

- With **mlrMBO** it is not hard to do it better!

```
# make MBO
mbo.ctrl = makeMBOControl()
# iters = 10 results in 4d + 10 evaluations (here 18)
mbo.ctrl = setMBOControlTermination(mbo.ctrl, iters = 10)
surrogate.lrn = makeLearner("regr.km", predict.type = "se")
ctrl = mlr:::makeTuneControlMBO(learner = surrogate.lrn,
  mbo.control = mbo.ctrl, same.resampling.instance = FALSE)
rdesc = makeResampleDesc("CV", iters = 3)
res.mbo = tuneParams(lrn, task, rdesc, par.set = ps,
  control = ctrl, show.info = FALSE)
```

HYPERPARAMETER TUNING

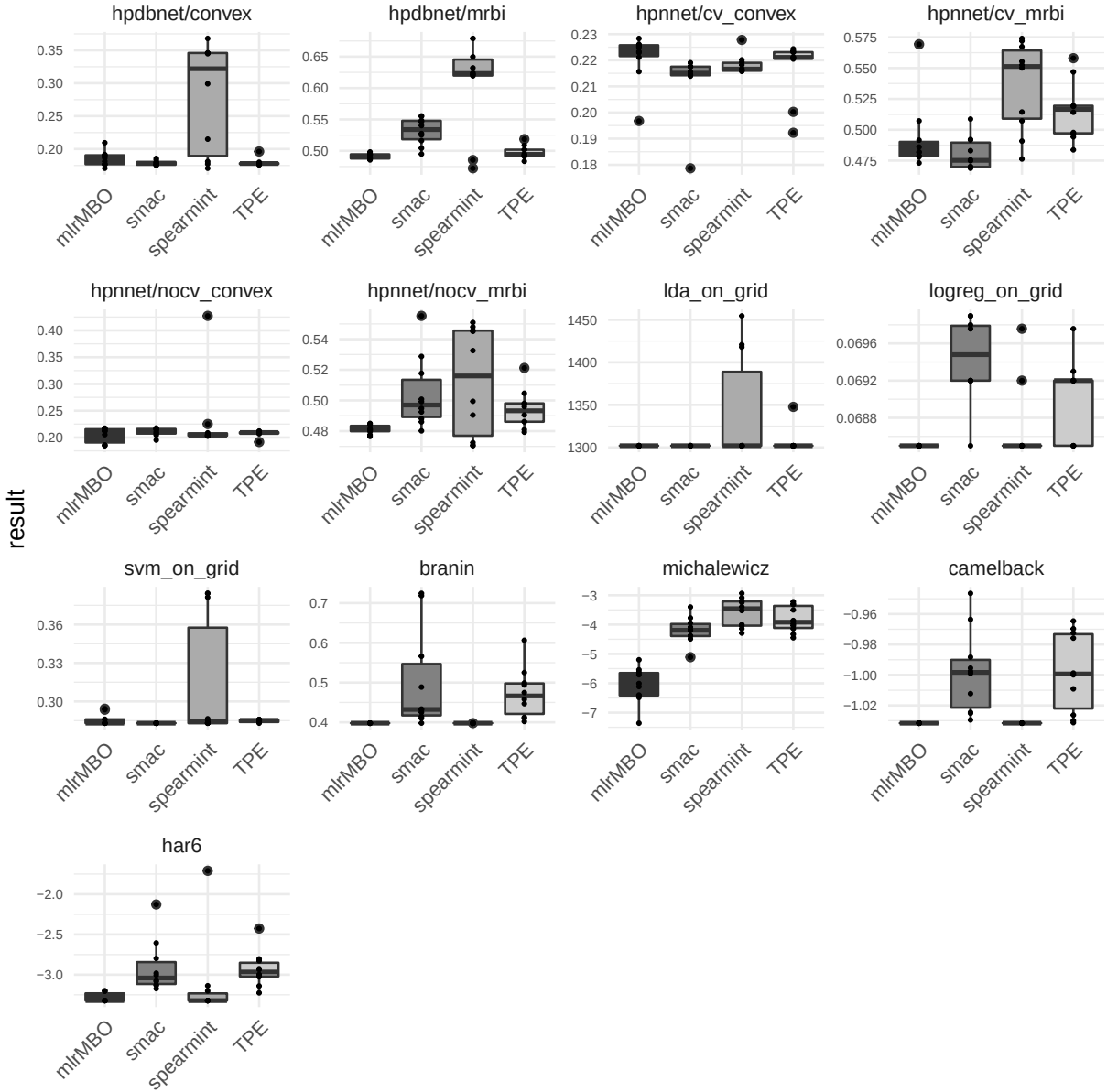


- The best grid search obtains an mmce.test.mean of 0.275.
- MBO manages to get an mmce.test.mean of 0.183

HPOLIB

- HPOlib is a set of standard benchmarks for hyperparameter optimizer
- Allows comparison with
  - Spearmint
  - SMAC
  - Hyperopt (TPE)
- Benchmarks:
  - Numeric test functions (similar to the ones we've seen bevor)
  - Numeric machine learning problems (lda, SVM, logistic regression)
  - Deep neural networks and deep belief networks with 15 and 35 parameters.
- For benchmarks with discrete and dependent parameters (hpnnet, hpdbnet) a random forest with standard error estimation is used.

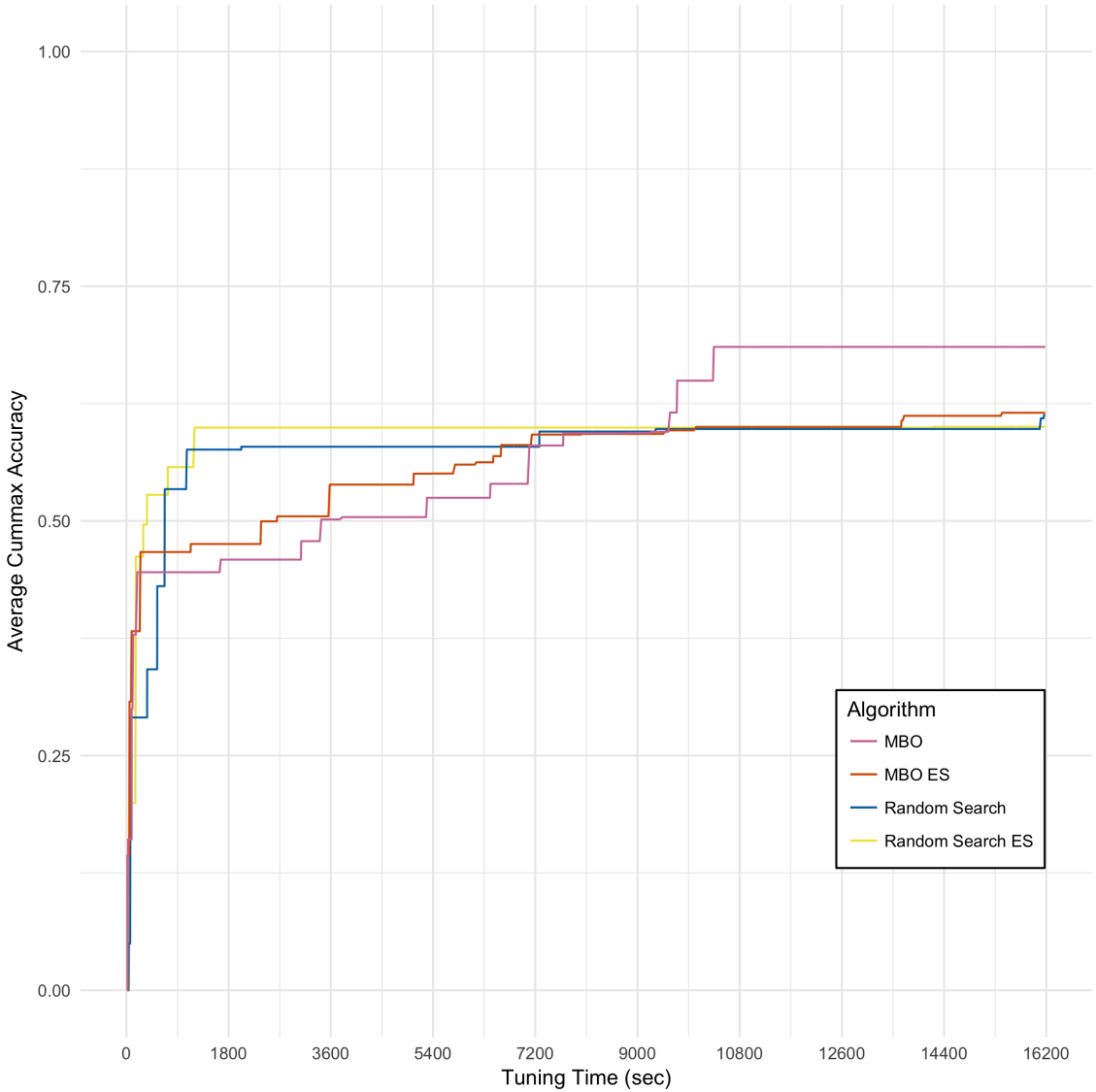
MBO: HPOLIB


hpdbnet/convex
hpdbnet/mrbi
hpnnet/cv_convex
hpnnet/cv_mrbi
hpnnet/nocv_convex
hpnnet/nocv_mrbi
lda_on_grid
logreg_on_grid
svm_on_grid
branin
michalewicz
camelback
har6
result
mlrMBO
smac
spearmint
TPE

DEEP LEARNING CONFIGURATION EXAMPLE

- Dataset: CIFAR-10 (60000 32x32 images with 3 color channels; 10 classes)
- Configuration of a deep neural network (mxnet)
- Size of parameter set: 30, including number of hidden layers, activation functions, regularization, convolution layer setting, etc.
- Split: 2/3 training set, 1/6 test set, 1/6 validation set
- Time budget per tuning run: 4.5h (16200 sec)
- Surrogate: Random forest
- Acquisition: LCB with $\lambda = 2$

DEEP LEARNING CONFIGURATION EXAMPLE



HYPERBAND

- Hyperband tries to tackle the B to n problem (“budget and amount of configurations to compare”). Particularly interesting for neural networks.
- Suppose we compare 81 random architectures: how many epochs until we eliminate “bad” architectures?
- Naturally, if we eliminate after a few epochs, architectures with high learning rate will dominate.
- Hyperband tries to solve the issue by considering different brackets with different B and n.

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

Figure: Hyperband with 5 brackets: n: number of configurations, r: budget/epochs for each config. For default values, only the best third of the configurations are kept and retrained until the next elimination phase. In the end only one config for each bracket remains (Lisha Li et al. (2016)).

ARCHITECTURE SEARCH

- Despite the success of neural networks, they are still hard to design.
- Barret Zoph & Quoc V. Le (2016) propose neural architecture search with reinforcement learning

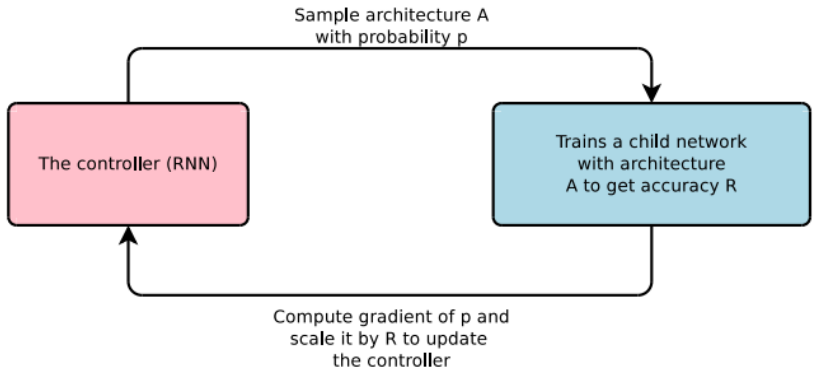


Figure: Overview of architecture search (Barret Zoph & Quoc V. Le (2016)).

- Another noteworthy approach is “Designing Neural Network Architectures using Reinforcement Learning” by Bowen Baker et al. (2017).

REFERENCES

Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh and Ameet Talwalkar (2016)

Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization

https://arxiv.org/abs/1603.06560

Barret Zoph and Quoc V. Le (2016)

Neural Architecture Search with Reinforcement Learning

https://arxiv.org/abs/1611.01578

Bowen Baker, Otkrist Gupta Nikhil Naik and Ramesh Raskar(2017)

Designing Neural Network Architectures using Reinforcement Learning

https://arxiv.org/abs/1611.02167