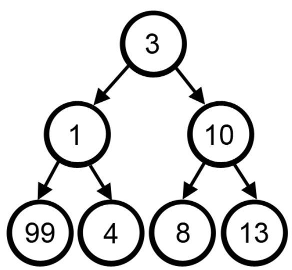
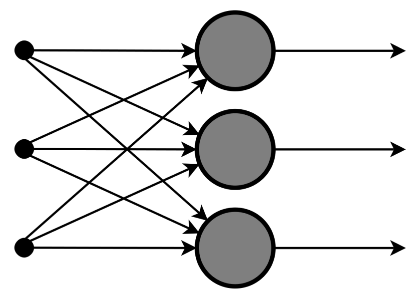
CSE 40171: Artificial Intelligence





Neural Network Model Search: Hyperparameter Optimization Strategies

Homework #4 has been released

It is due at 11:59PM on 10/18

Quiz #1 is scheduled for 10/30

Project proposal instructions have been released. Proposals are Due 11/4.

(Let me know if you need a group)

Q1: What properties would an approach that is better than random search have?

Q2: What are some possible alternatives to random search?

Generic Sequential Model-based Optimization

SMBO(f, M_0, T, S)

$\mathcal{H} \leftarrow \emptyset$,
For $t \leftarrow 1$ **to** T,
  $x^* \leftarrow \text{argmin}_x \; S(x, M_{t-1})$,
  Evaluate $f(x^*)$, ▷ *Expensive step*
  $\mathcal{H} \leftarrow \mathcal{H} \cup (x^*, f(x^*))$,
  Fit a new model M_t to $\mathcal{H}$.
return $\mathcal{H}$

Tree-Structured Parzen Estimator (TPE)

There is more than one way to search via hyperopt:

```
# define an objective function
def objective(args):
    case, val = args
    if case == 'case 1':
        return val
    else:
        return val ** 2

# define a search space
from hyperopt import hp
space = hp.choice('a',
    [
        ('case 1', 1 + hp.lognormal('c1', 0, 1)),
        ('case 2', hp.uniform('c2', -10, 10))
    ])

# minimize the objective over the space
from hyperopt import fmin, tpe, space_eval
best = fmin(objective, space, algo=tpe.suggest, max_evals=100)

print(best)
# -> {'a': 1, 'c2': 0.01420615366247227}
print(space_eval(space, best))
# -> ('case 2', 0.01420615366247227}
```

http://hyperopt.github.io/hyperopt/

Step 1: Sample reference sets

Bergstra et al. NIPS 2011

Assumption: “good” and “bad” hyperparameter sets can be modeled by different distributions

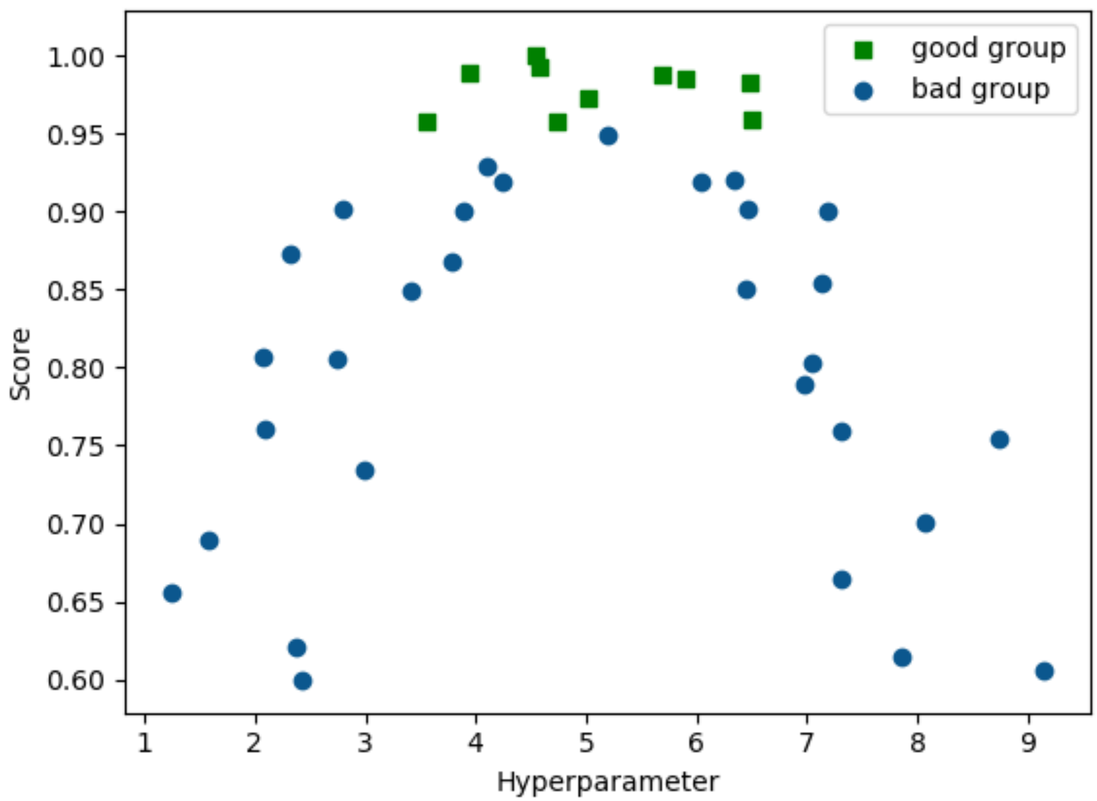


Image Credit: http://dkopczyk.quantee.co.uk/hyperparameter-optimization/

Ex. 25% of parameter sets go to the “good” group after random search

Step 2: Kernel Density Estimation

- Each sample defines a Gaussian distribution
- Mean equal to a hyperparameter value; specified standard deviation
- Distributions are stacked together and normalized to get a valid prob. distribution

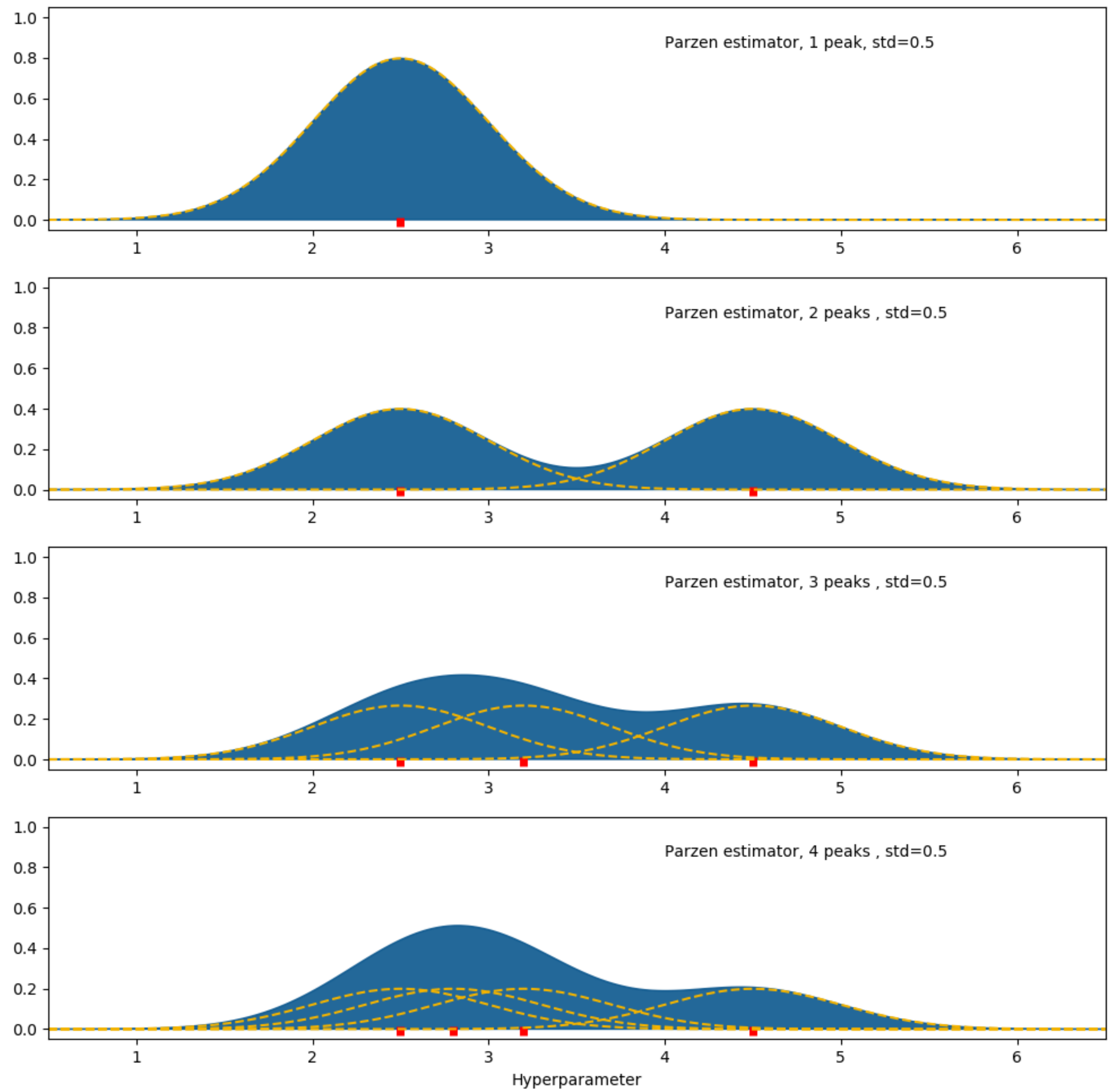


Image Credit: http://dkopczyk.quantee.co.uk/hyperparameter-optimization/

Step 3: Find candidate with best expected improvement

Sampling Problem: find a hyperparameter combination that more likely belongs to a “good” group and less likely to “bad” group.

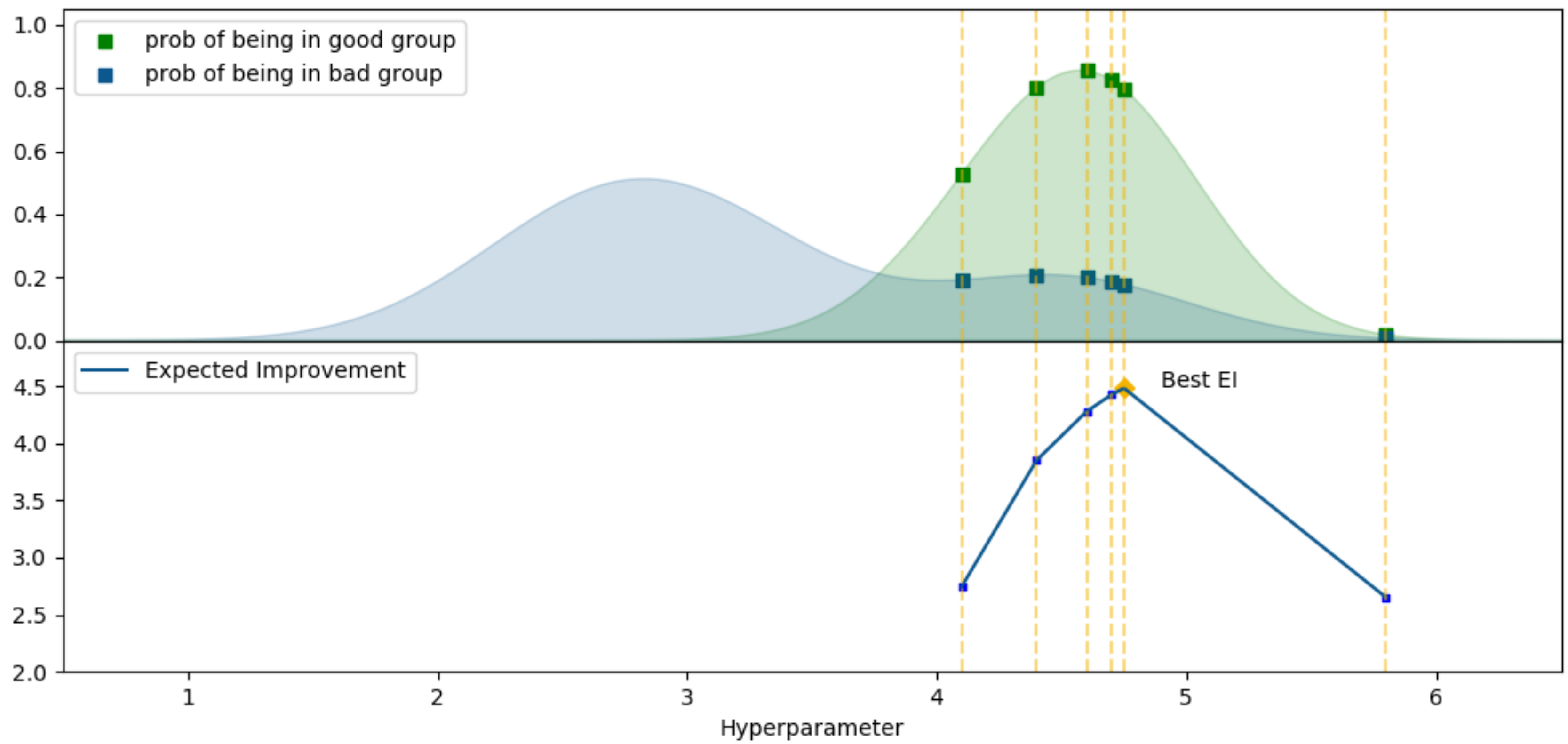


Image Credit: http://dkopczyk.quantee.co.uk/hyperparameter-optimization/

How effective is TPE?

Bergstra et al. NIPS 2011

	convex	MRBI
TPE	**14.13** ± 0.30 %	**44.55** $\pm 0.44\%$
GP	$16.70 \pm 0.32\%$	$47.08 \pm 0.44\%$
Manual	$18.63 \pm 0.34\%$	$47.39 \pm 0.44\%$
Random	18.97 ± 0.34 %	$50.52 \pm 0.44\%$

Algorithms were allowed up to 200 trials. The manual searches used 82 trials for convex and 27 trials MRBI.

Boston Housing Price Problem
(http://dkopczyk.quantee.co.uk/hyperparameter-optimization/)

Random

TIME (minutes)	BEST CV SCORE (%)	TEST SCORE (%)
3.8	86.10	89.58

TPE

TIME (minutes)	BEST CV SCORE (%)	TEST SCORE (%)
4.5	86.37	90.42

Pros of TPE

+ Conceptually Simple
+ Implementations in multiple Python packages
    - hyperopt, optunity
+ At least as effective as random search in some settings

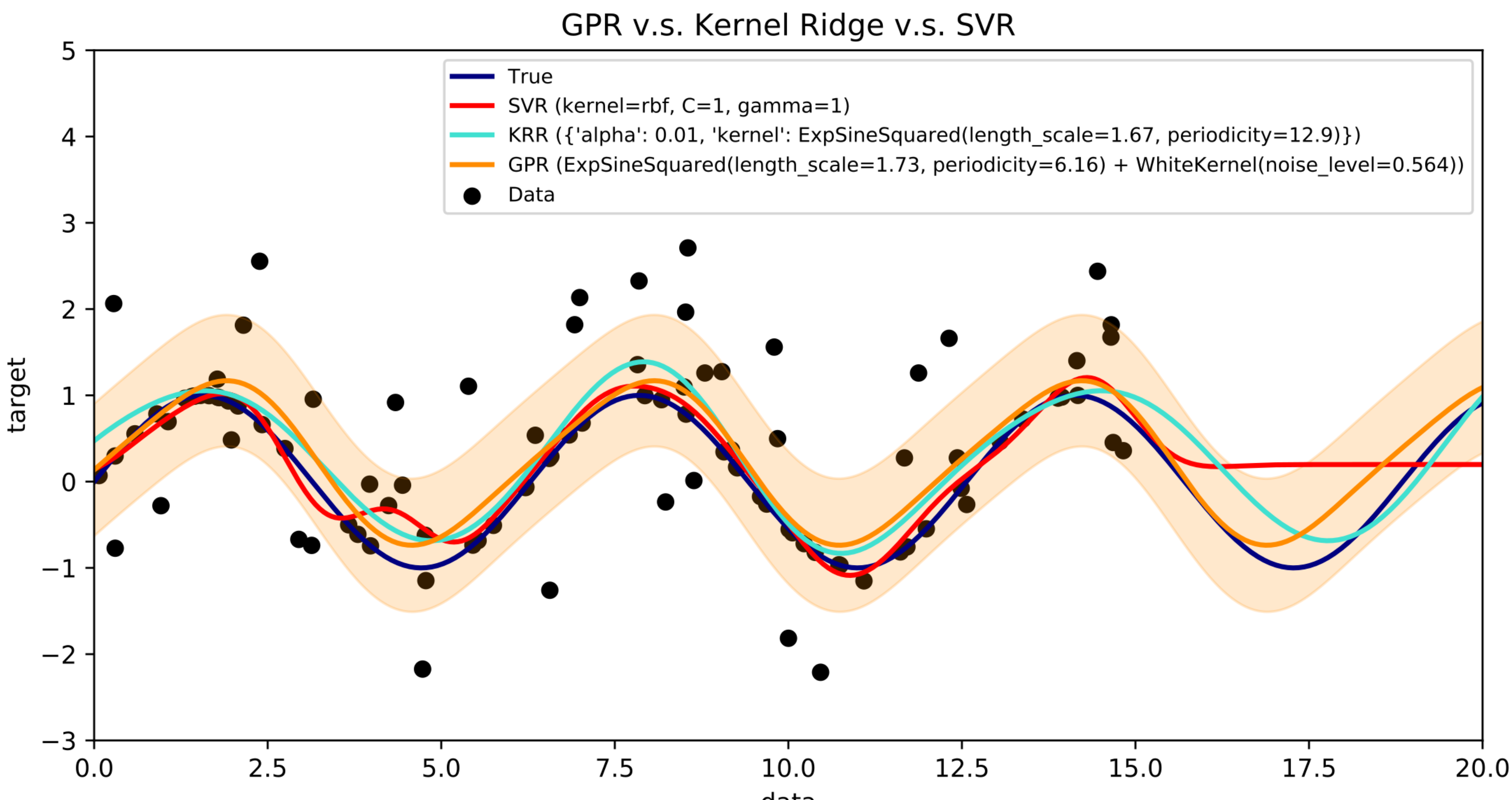
Cons of TPE

- Limited by the structure learned from data
- It is possible for TPE to be arbitrarily bad with a bad choice of $P(y \mid x)$
- Possible to be slower than random sampling at finding a global optimum with an apparently good $P(y \mid x)$
- Realistically, will only find a local optimum

Gaussian Process (GP)

Bergstra et al. NIPS 2011

Long recognized as a good method for modeling loss functions in model-based optimization literature

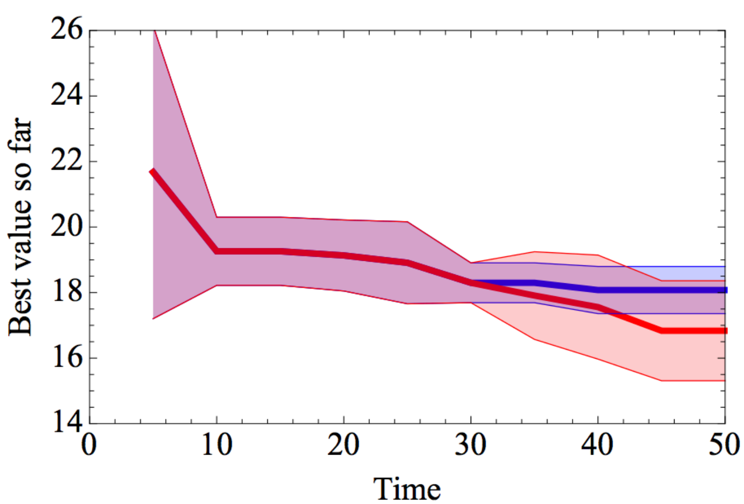


How effective is GP?

Bergstra et al. NIPS 2011

	convex	**MRBI**
TPE	**14.13** ± 0.30 %	**44.55** $\pm 0.44\%$
GP	$16.70 \pm 0.32\%$	$47.08 \pm 0.44\%$
Manual	$18.63 \pm 0.34\%$	$47.39 \pm 0.44\%$
Random	18.97 ± 0.34 %	$50.52 \pm 0.44\%$

Algorithms were allowed up to 200 trials. The manual searches used 82 trials for convex and 27 trials MRBI.



Boston Housing Price Problem

Red = GP, Blue = Random. Shaded areas = one-sigma error bars

Pros and cons of GP

+ Priors over functions that are *closed under sampling*

+ Provide an assessment of prediction uncertainty incorporating the effect of data scarcity

+ At least as effective as random search in some settings

- Has its own hyperparameters, which must be tuned

- Limited by the structure learned from data

- Realistically, will only find a local optimum

- Empirically worse than TPE

More on Bayesian Optimization

Let's revisit the idea of Gaussian Process…

Snoek et al. NIPS 2012

- A learning algorithm's generalization performance is modeled as a sample from a GP
- Type of kernel and the treatment of its hyperparameters, can play a crucial role in obtaining a good optimizer

But need to factor in variable cost

Thinking about this problem from a systems perspective

Machine learning problems are different from other black-box optimization problems

- each function evaluation can require a variable amount of time

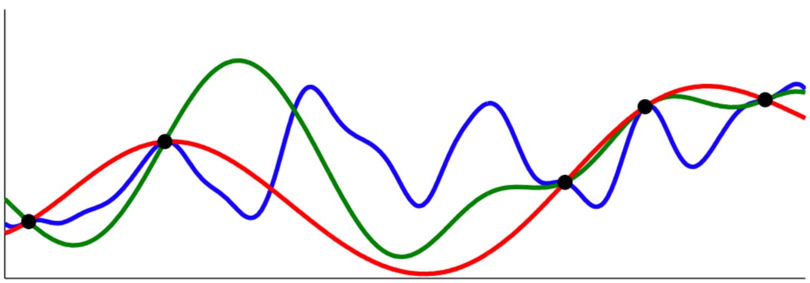
Machine learning experiments are often run in parallel, on multiple cores or machines.

Problem: in both cases the standard sequential approach of GP optimization can be suboptimal

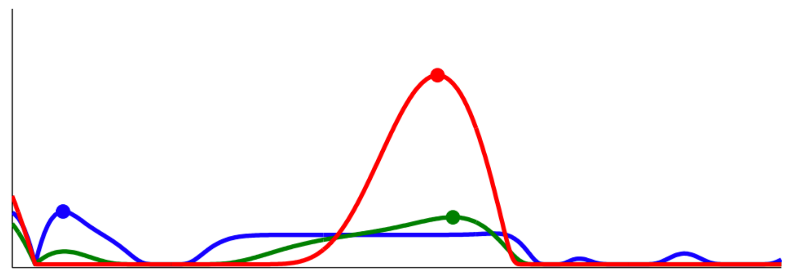
Integrated acquisition function

$$\hat{a}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}) = \int a(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)\, p(\theta \,|\, \{\mathbf{x}_n, y_n\}_{n=1}^{N})\, \mathrm{d}\theta,$$

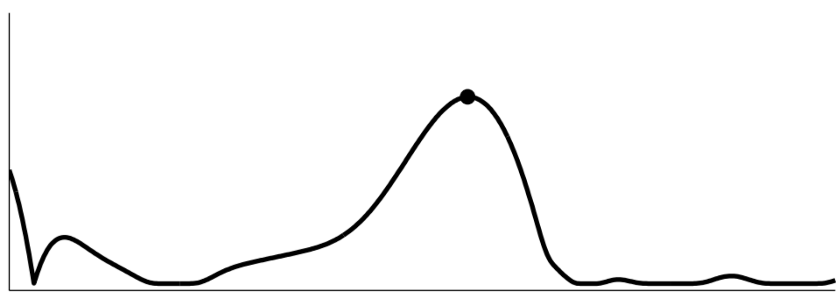
Depends on the parameters and all of the observations



(a) Posterior samples under varying hyperparameters



(b) Expected improvement under varying hyperparameters



(c) Integrated expected improvement

Image Credit: Snoek et al. NIPS 2012

Monte Carlo acquisition for parallelizing Bayesian optimization

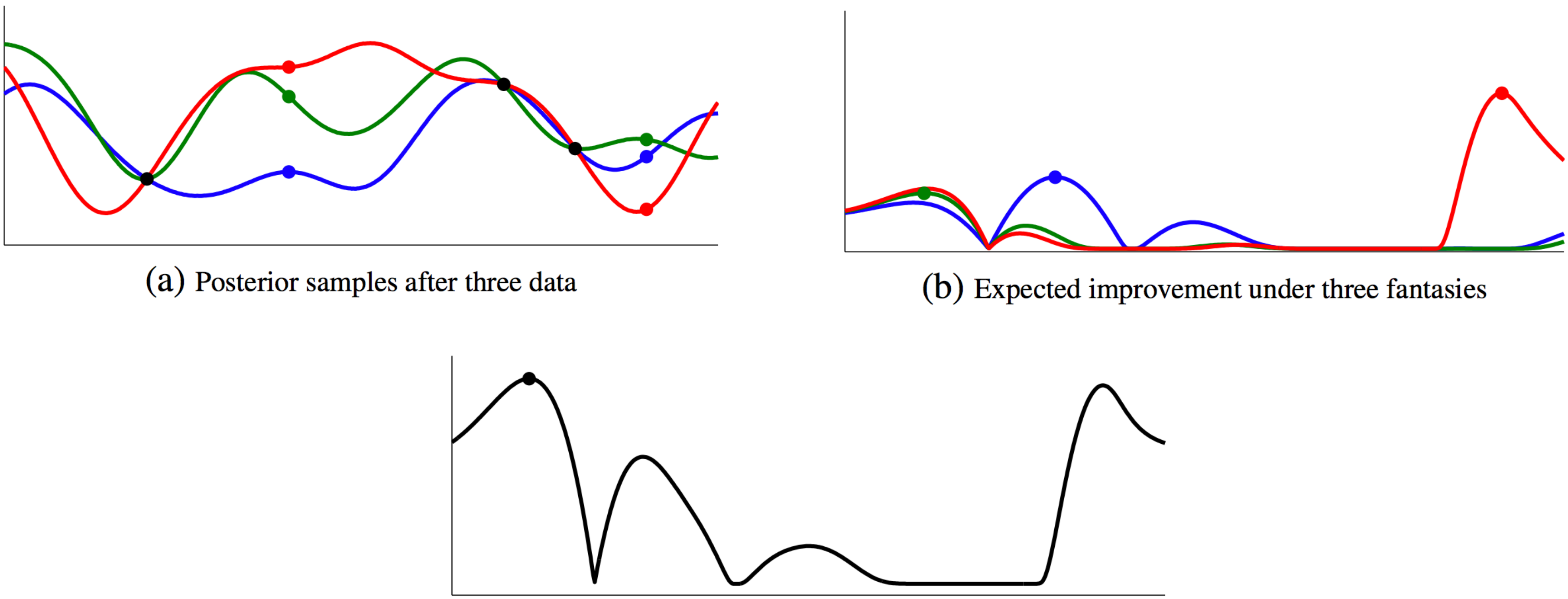
Compute Monte Carlo estimates of the acquisition function under different possible results from pending function evaluations.

Scenario: N evaluations have completed, yielding data $\{\mathbf{x}_n, y_n\}_{n=1}^N$, and in which J evaluations are pending at locations $\{\mathbf{x}_j\}_{j=1}^J$

Choose a new point based on the expected acquisition function under all possible outcomes of these pending evaluations:

$$\hat{a}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta, \{\mathbf{x}_j\}) = \int_{\mathbb{R}^J} a(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta, \{\mathbf{x}_j, y_j\})\, p(\{y_j\}_{j=1}^J \,|\, \{\mathbf{x}_j\}_{j=1}^J, \{\mathbf{x}_n, y_n\}_{n=1}^N)\, \mathrm{d}y_1 \cdots \mathrm{d}y_J.$$

Monte Carlo acquisition for parallelizing Bayesian optimization



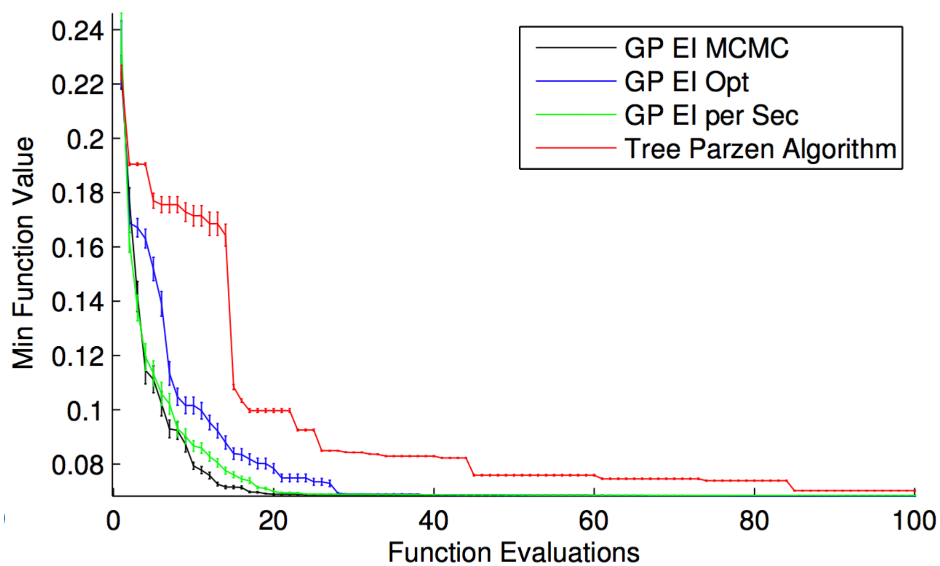
(a) Posterior samples after three data

(b) Expected improvement under three fantasies

(c) Expected improvement across fantasies

Image Credit: Snoek et al. NIPS 2012

Logistic regression on MNIST



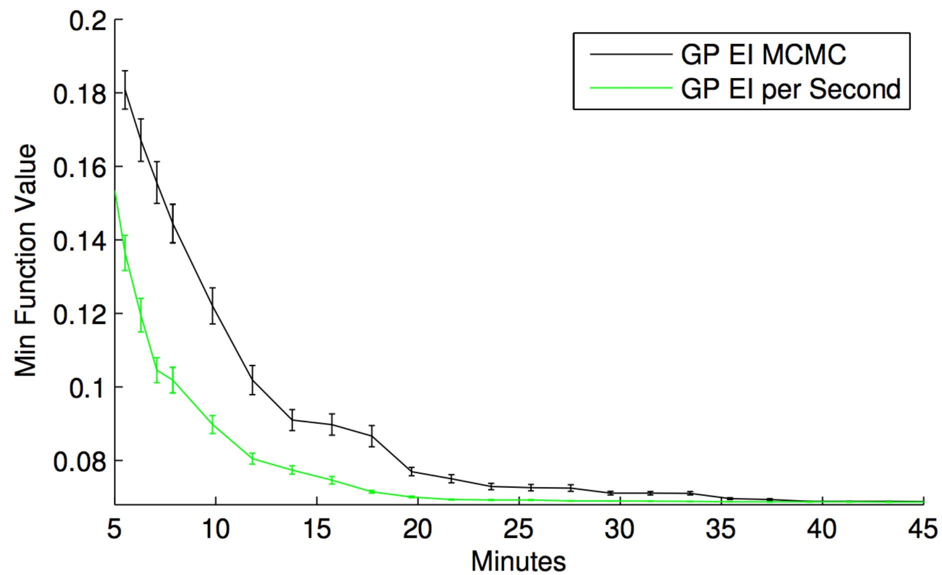
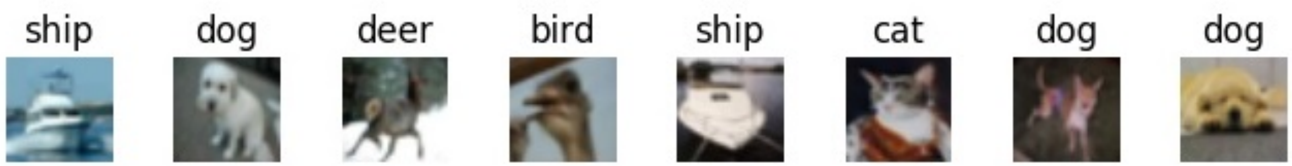
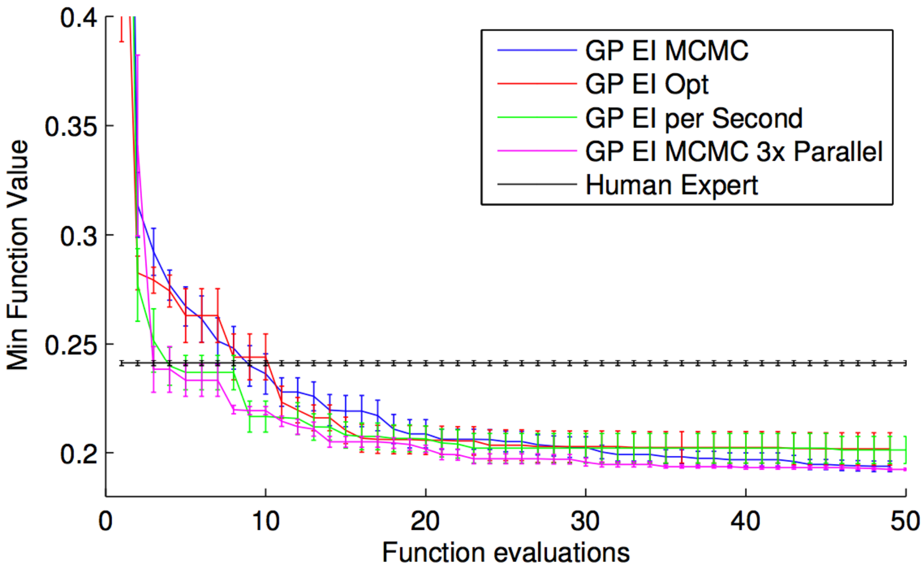


Image Credit: Snoek et al. NIPS 2012

CNN validation error on CFAR-10





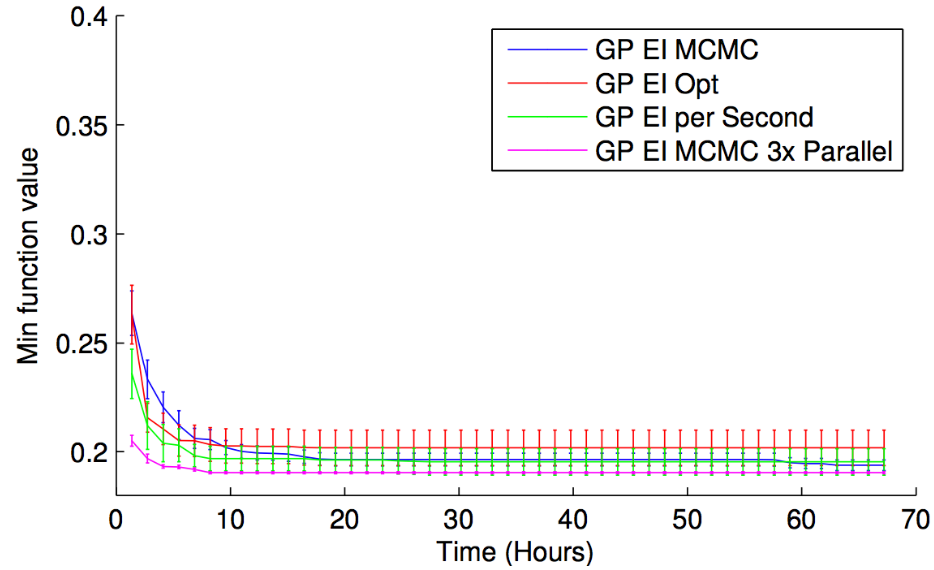
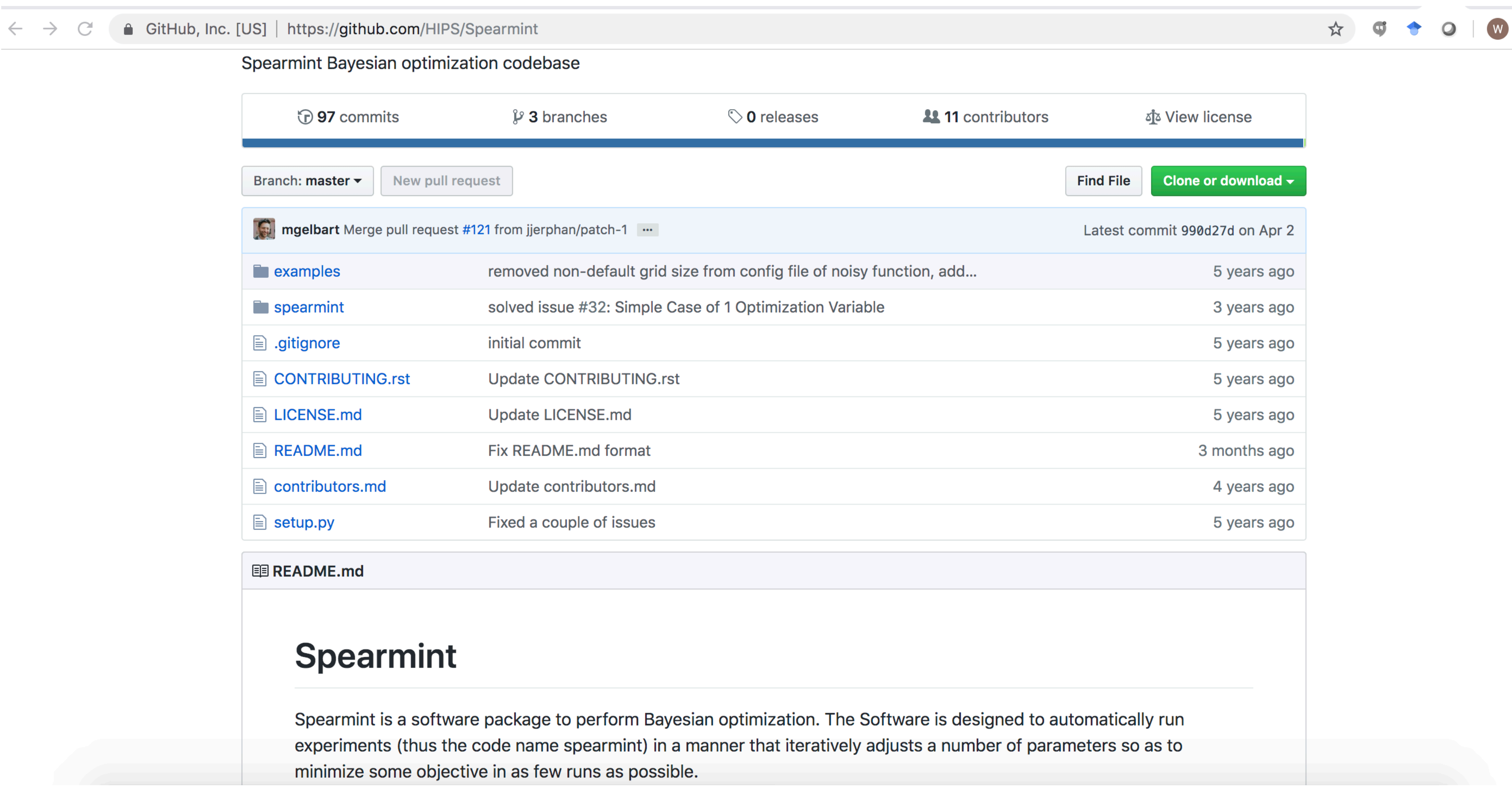


Image Credit: Snoek et al. NIPS 2012

Spearmint


GitHub, Inc. [US] | https://github.com/HIPS/Spearmint
Spearmint Bayesian optimization codebase
97 commits
3 branches
0 releases
11 contributors
View license
Branch: master
New pull request
Find File
Clone or download
mgelbart Merge pull request #121 from jjerphan/patch-1
Latest commit 990d27d on Apr 2
examples
removed non-default grid size from config file of noisy function, add…
5 years ago
spearmint
solved issue #32: Simple Case of 1 Optimization Variable
3 years ago
.gitignore
initial commit
5 years ago
CONTRIBUTING.rst
Update CONTRIBUTING.rst
5 years ago
LICENSE.md
Update LICENSE.md
5 years ago
README.md
Fix README.md format
3 months ago
contributors.md
Update contributors.md
4 years ago
setup.py
Fixed a couple of issues
5 years ago
README.md
Spearmint
Spearmint is a software package to perform Bayesian optimization. The Software is designed to automatically run experiments (thus the code name spearmint) in a manner that iteratively adjusts a number of parameters so as to minimize some objective in as few runs as possible.

Optunity

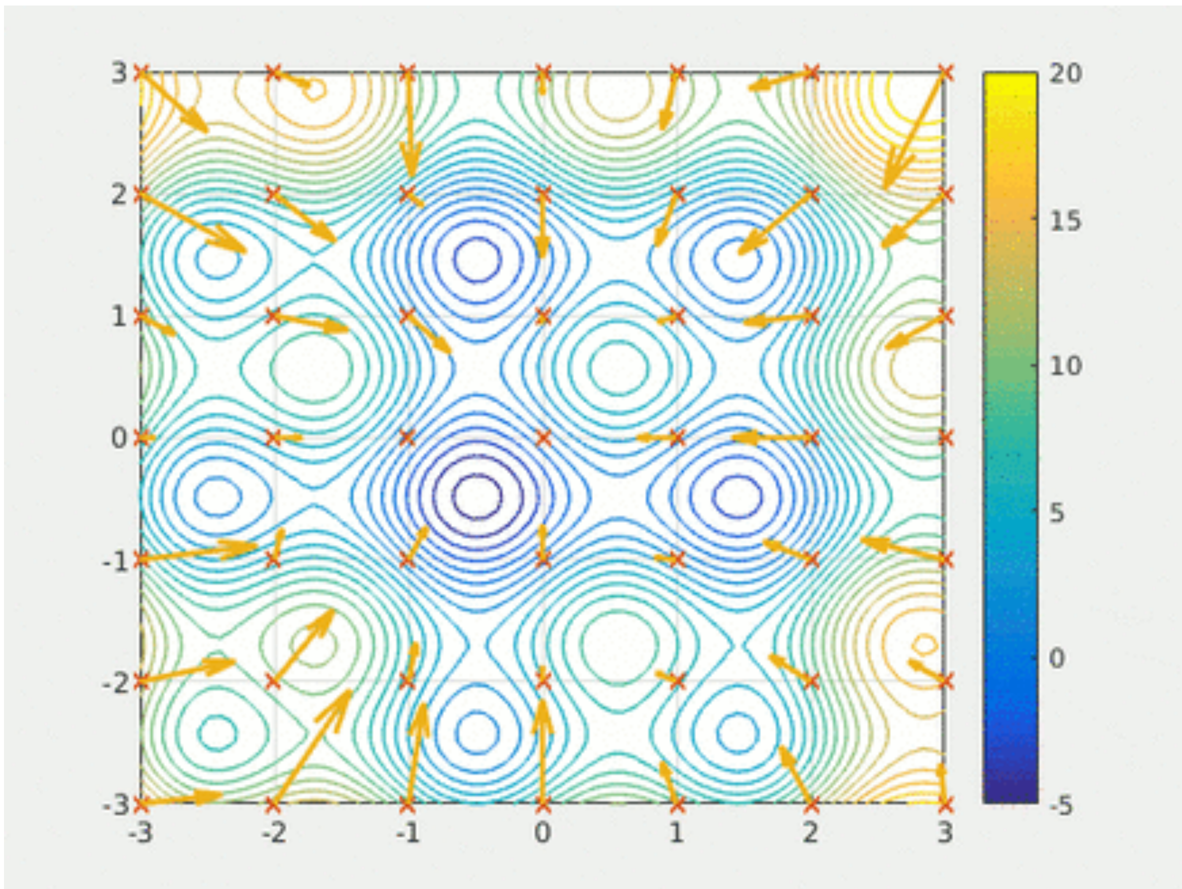
https://github.com/claesenm/optunity.git

pip install optunity

Particle Swarm Optimization

Position of a particle represents a set of hyperparameters

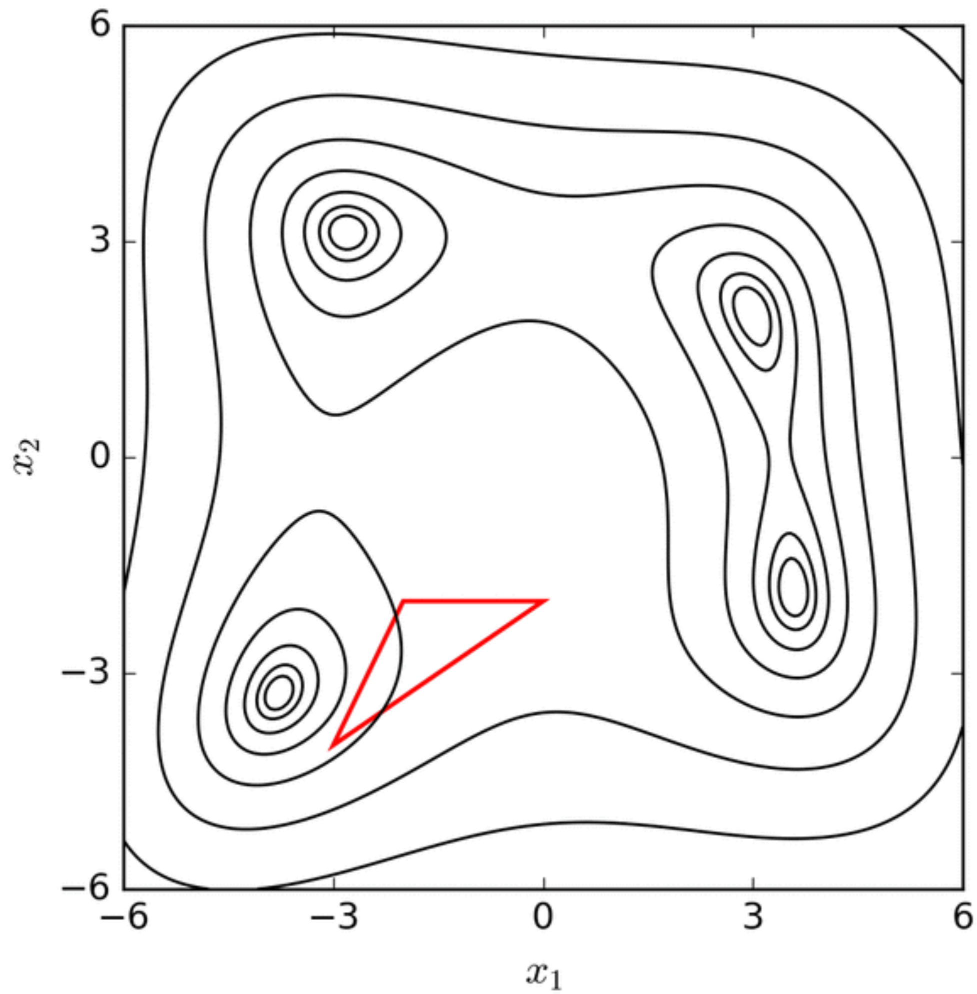
Movement is influenced by the goodness of the objective function value



Particle Swarm Arrows Animation

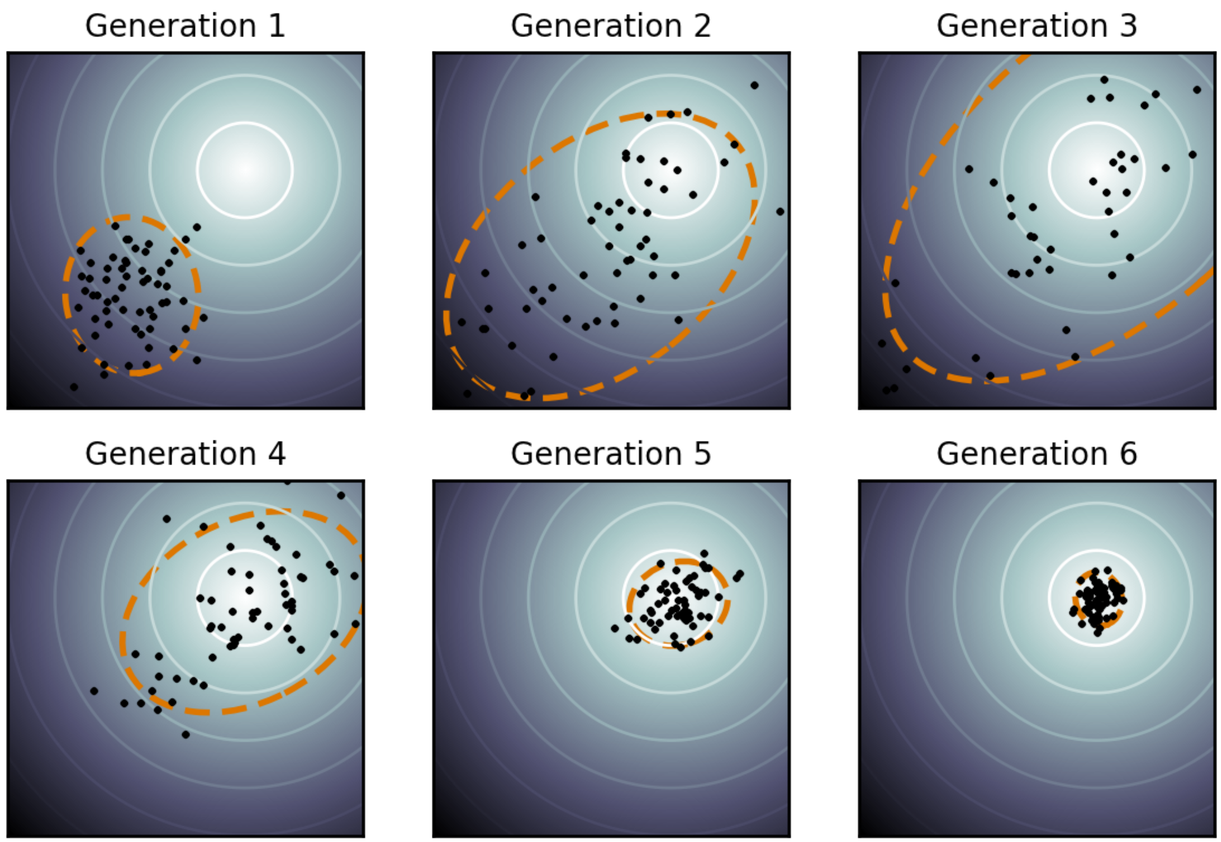
Nelder-Mead Simplex

- Nonlinear optimization method based on the concept of a *simplex*
- Good local search method, but will get stuck in bad regions when a poor starting point is specified



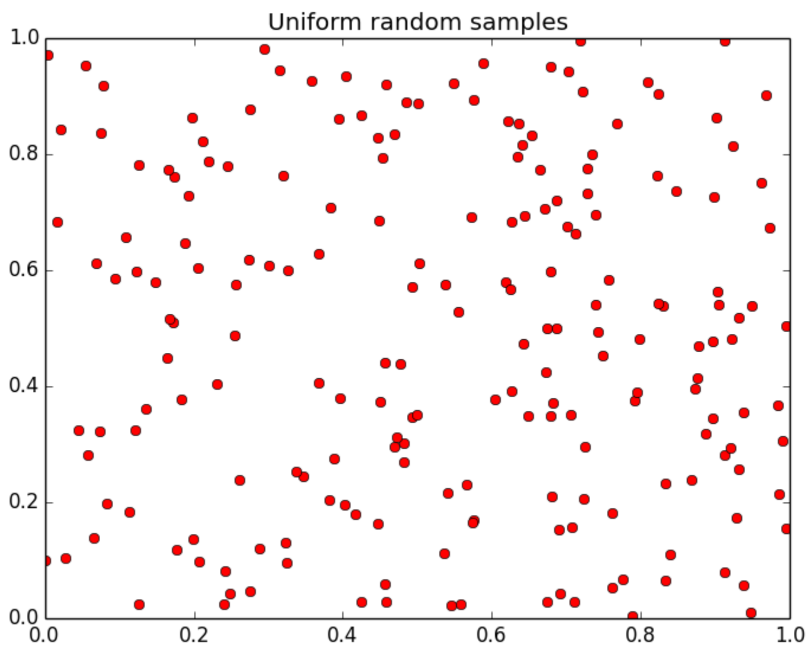
Covariance Matrix Adaptation Evolutionary Strategy

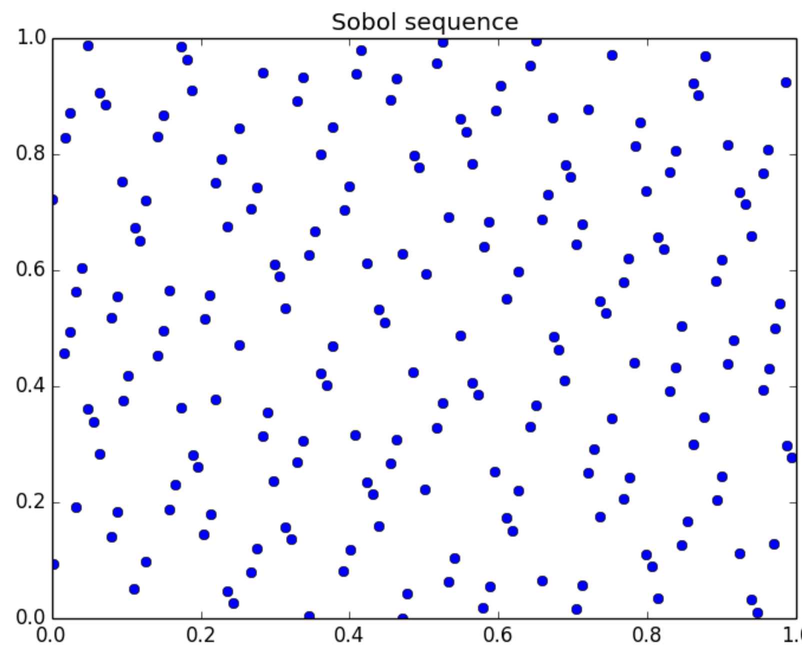
- Evolutionary strategy for continuous function optimization
- Dynamically adapt search resolution per hyperparameter, allowing for efficient searches at different scales



Sobol Sequences

Sobol sequences are designed to cover the unit hypercube with lower discrepancy than completely random sampling

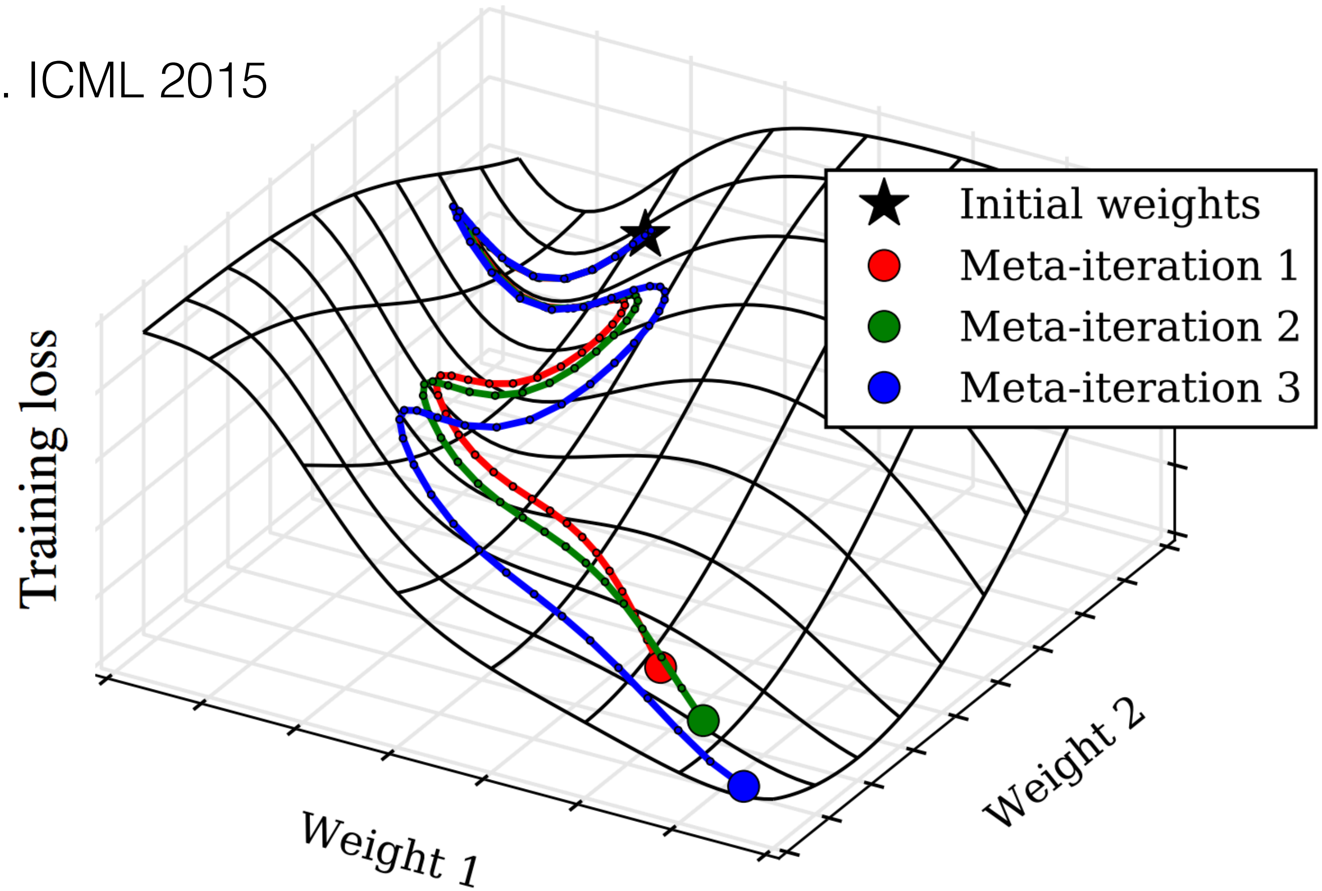




Plots generated via optunity: https://optunity.readthedocs.io

Gradient-based Optimization

Maclaurin et al. ICML 2015



1. Entire training run with SGD to optimize weights
2. Compute gradients of validation loss with respect to hyperparameters via backprop.
3. Update hyperparameters in the direction of this hypergradient

Opens up a “garden of delights”…

- Efficient optimization of thousands of parameters
- Finer-grained hyperparameter optimization, *e.g.,* per layer optimization in a neural network
- Flexibility over:
    - Model classes
    - Regularization
    - Training Methods

Optimization over training error (MNIST)

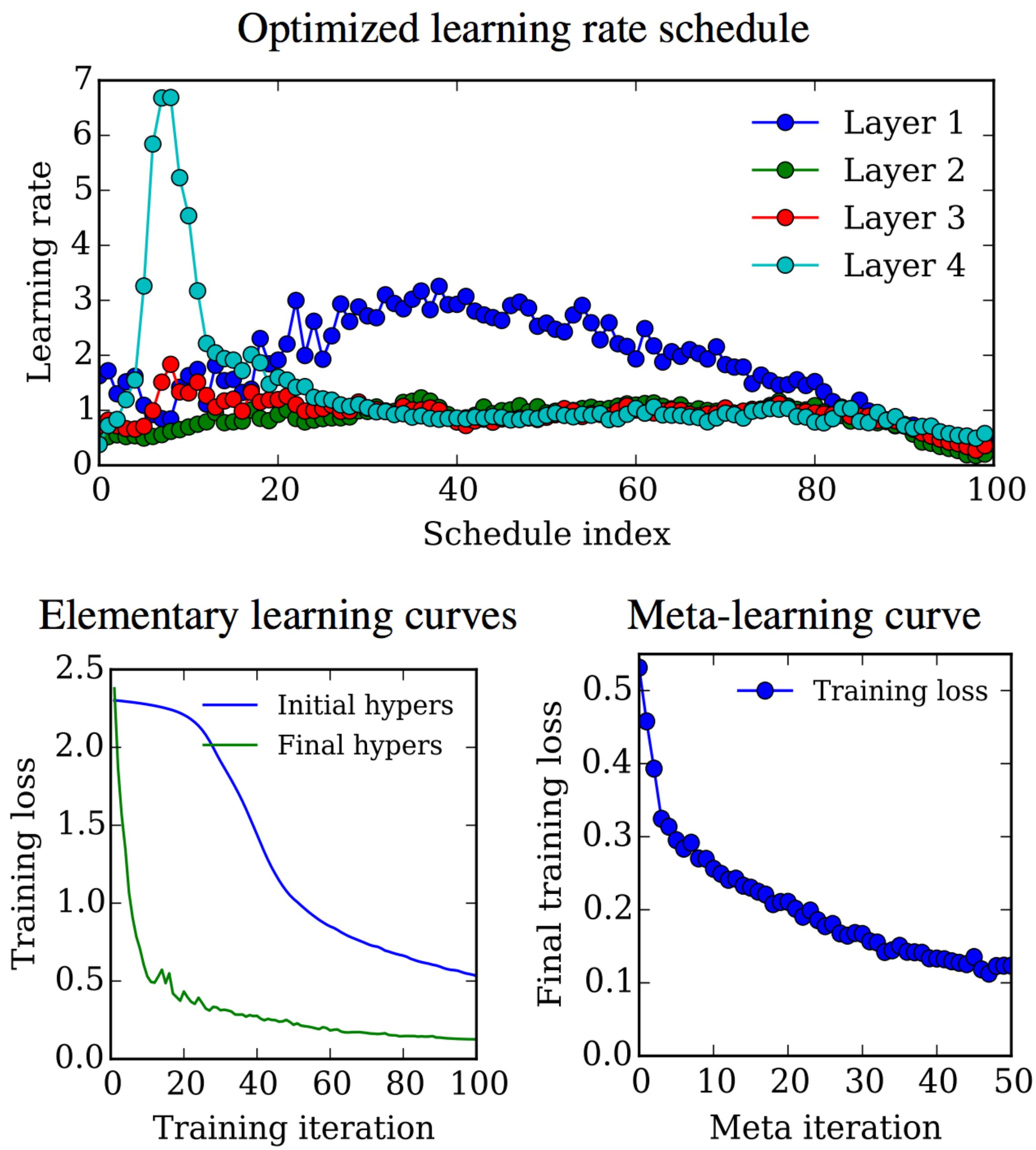


Image Credit: Maclaurin et al. ICML 2015

Limitations

- Learning long-term dependencies with gradient descent is difficult
  - Large learning rates induce chaotic behavior in the learning dynamics
- Overfitting
- Discrete parameters still need to be optimized by hand

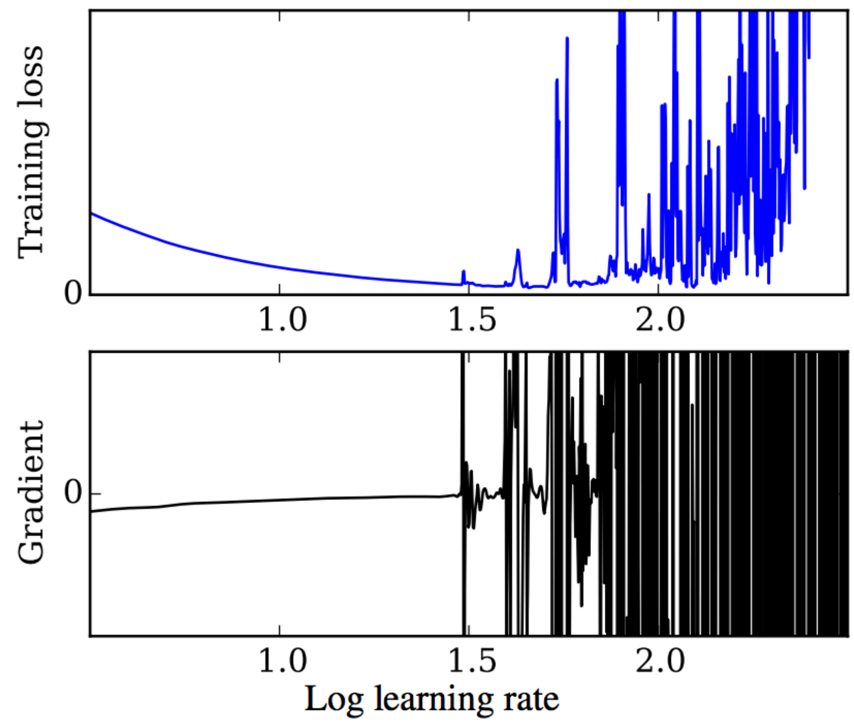


Image Credit: Maclaurin et al. ICML 2015