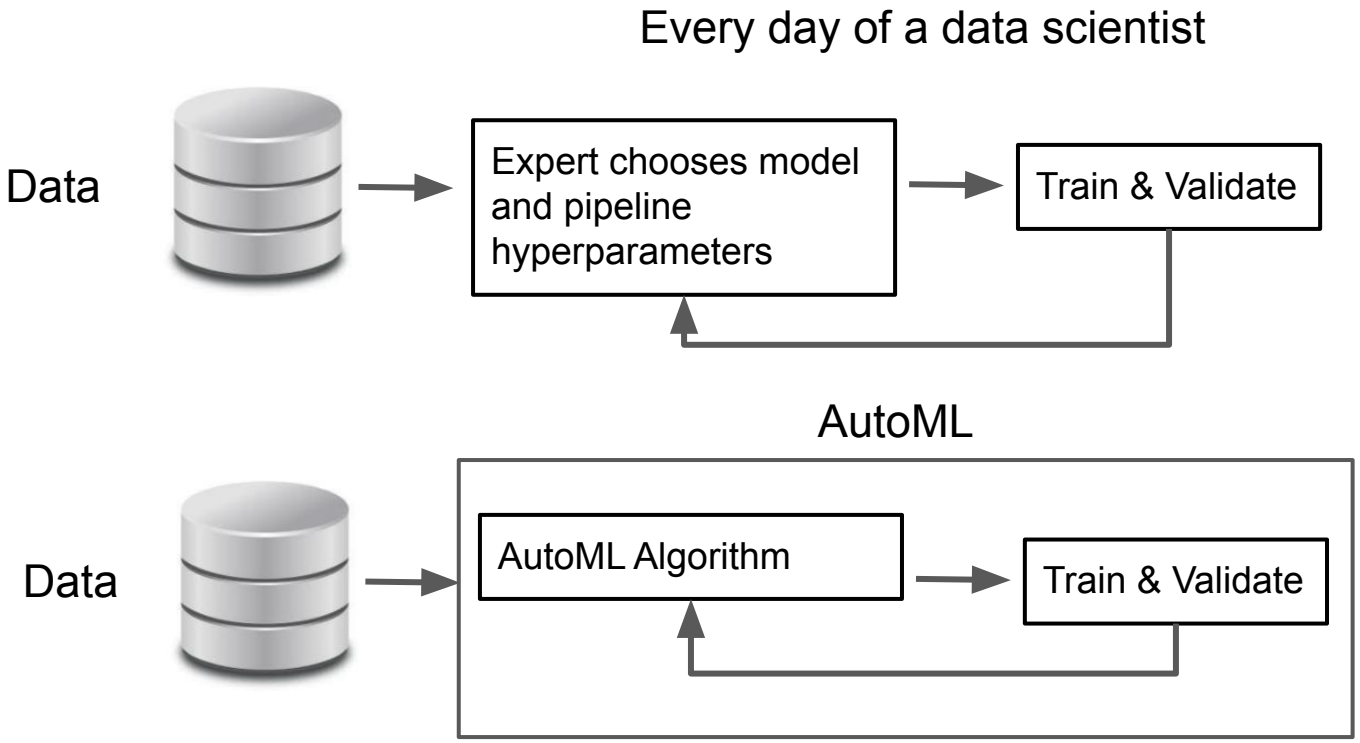
AutoML

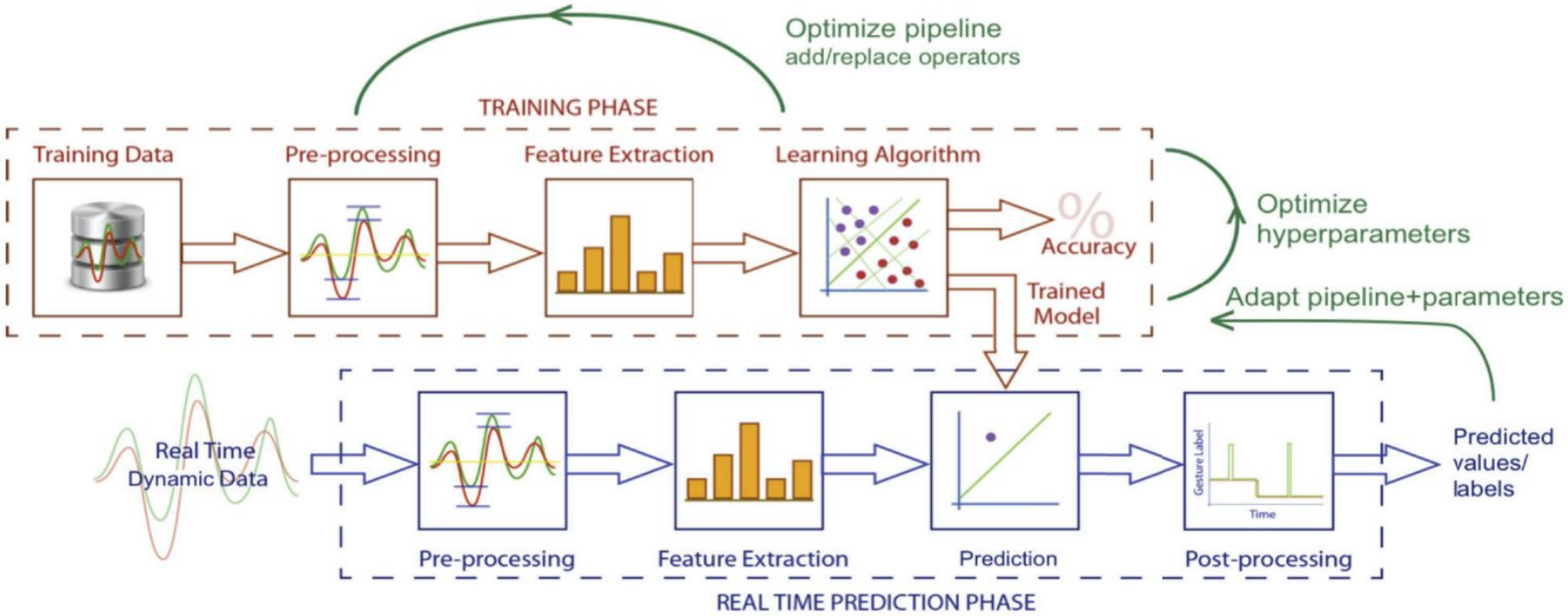
Ilya Trofimov
ilya.trofimov@skoltech.ru

16.02.26

AutoML: end-to-end learning



Training/validation pipeline



https://www.automl.org

Traditional ML pipeline

- Clean & preprocess the data;
- Feature selection, feature engineering;
- Select a model family;
- Set the hyperparameters;
- Construct ensembles of models.

Hyperparameter optimization

Definition: Hyperparameter Optimization (HPO)

Let

- $\boldsymbol{\lambda}$ be the hyperparameters of a ML algorithm A with domain $\boldsymbol{\Lambda}$,
- $\mathcal{L}(A_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$ denote the loss of A, using hyperparameters $\boldsymbol{\lambda}$ trained on D_{train} and evaluated on D_{valid}.

The hyperparameter optimization (HPO) problem is to find a hyperparameter configuration $\boldsymbol{\lambda}^*$ that minimizes this loss:

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \mathcal{L}(A_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$$

Combined algorithm selection and hyperparameter optimization (CASH)

Definition: Combined Algorithm Selection and Hyperparameter Optimization (CASH)

Let

- $\mathcal{A} = \{A^{(1)}, \ldots, A^{(n)}\}$ be a set of algorithms
- $\mathbf{\Lambda}^{(i)}$ denote the hyperparameter space of $A^{(i)}$, for $i = 1, \ldots, n$
- $\mathcal{L}(A^{(i)}_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$ denote the loss of $A^{(i)}$, using $\lambda \in \mathbf{\Lambda}^{(i)}$ trained on D_{train} and evaluated on D_{valid}.

The Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem is to find a combination of algorithm $A^* = A^{(i)}$ and hyperparameter configuration $\boldsymbol{\lambda}^* \in \mathbf{\Lambda}^{(i)}$ that minimizes this loss:

$$A^*_{\boldsymbol{\lambda}^*} \in \underset{A^{(i)} \in \mathcal{A}, \boldsymbol{\lambda} \in \mathbf{\Lambda}^{(i)}}{\arg\min} \mathcal{L}(A^{(i)}_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$$

Type of hyperparameters

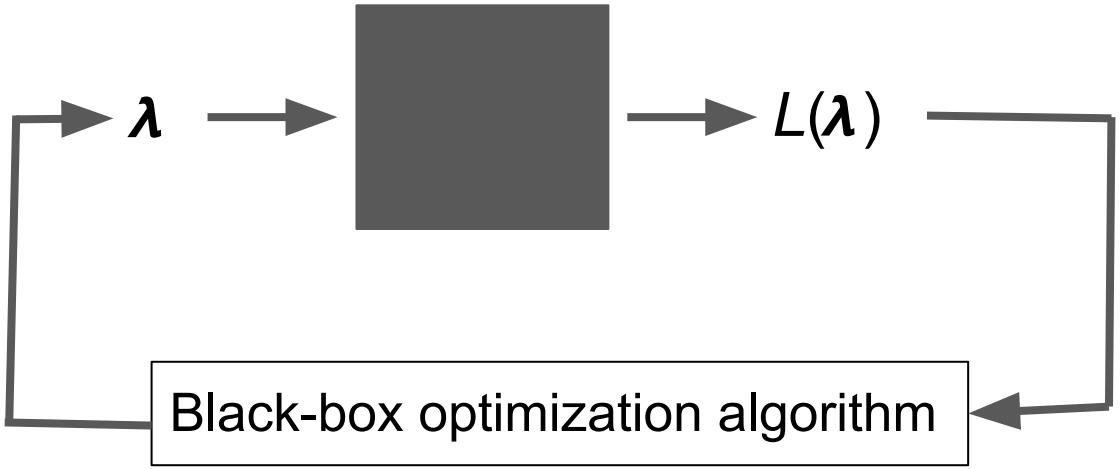
- **Real-valued**: SVM C-parameter, learning rate;
- **Integer**: number of trees in XGBoost, kernel size;
- **Categorial** (finite unordered set):
  - Example 1: Algorithm = SVM, NN, Random Forest
  - Example 2: Activation function = RELU, Leaky RELU, Tanh, GELU
  - Example 3: operator = {conv3x3, separable conv3x3, max pool, …}

Conditional hyperparameters

Hyperparameter B are only active if other hyperparameters A are set a certain way

- **A** = choice of optimizer (Adam or SGD); **B** = Adam‘s second momentum hyperparameter (only active if A=Adam)
- **A** = type of layer *k* (convolution, max pooling, fully connected, ...); **B** = conv. kernel size of that layer (only active if A = convolution)
- **A** = choice of classifier (RF or SVM); **B** = SVM‘s kernel parameter (only active if A = SVM)

Black-box optimization



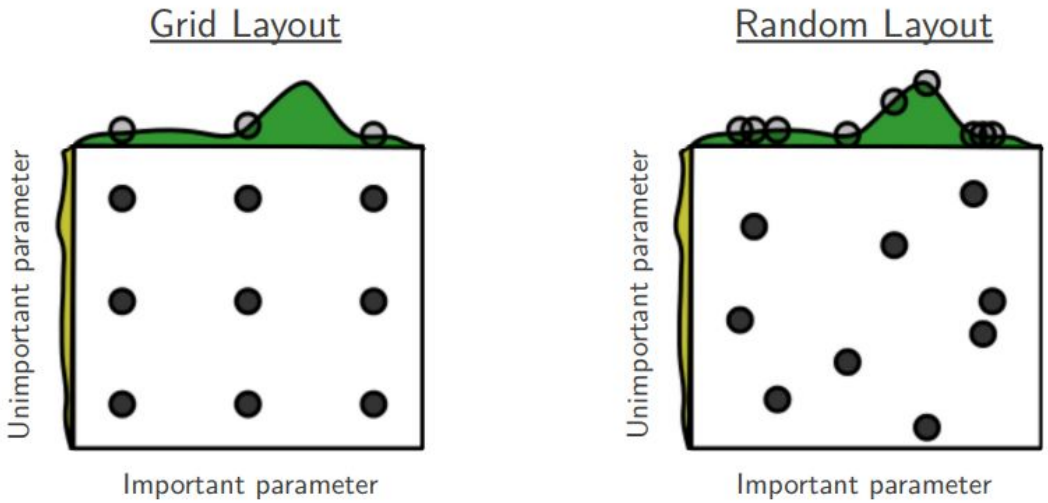
- No derivatives are available
- Black-box function is expensive to evaluate - sample efficiency is required

Black-box optimization

1. Grid search / Random Search
2. Bayesian optimization
3. Multi-fidelity optimization
4. Hyperparameter gradient descent

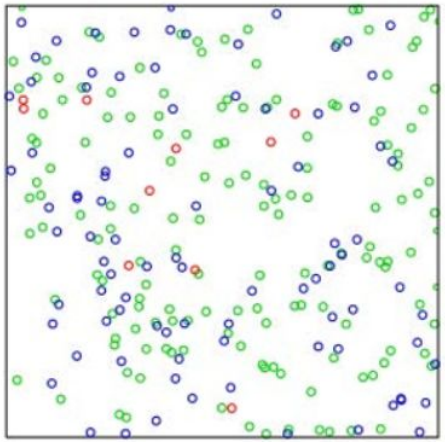
Random Search

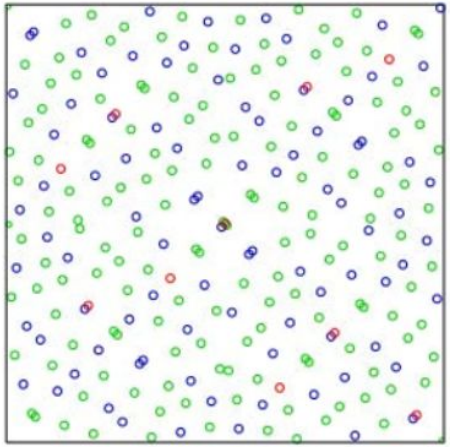
Random search is a useful, model-free baseline.



Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, *13*(2).

Sobol sequence (low-discrepancy sequence)





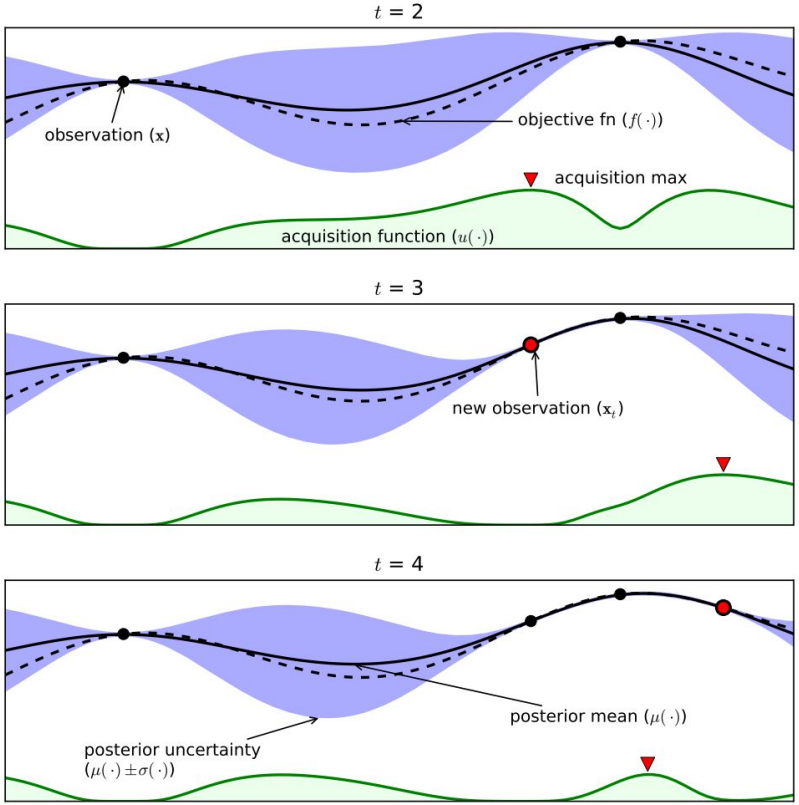
red=1,..,10, blue=11,..,100, green=101,..,256

https://en.wikipedia.org/wiki/Sobol_sequence

Bayesian Optimization

- Find a probabilistic model for function evaluations
- Use that model to trade-off exploration/exploitation

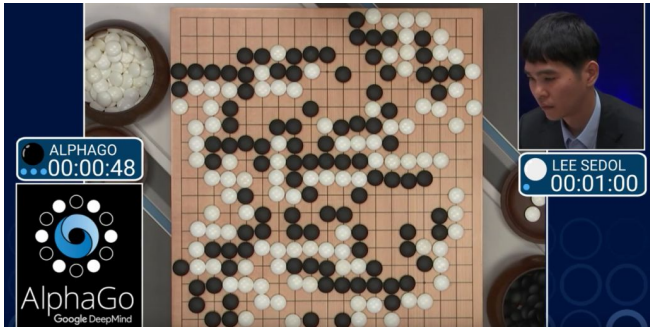
$$P(f|\mathcal{D}_{1:t}) \propto P(\mathcal{D}_{1:t}|f)P(f).$$



Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, *25*.

Brochu, E., Cora, V. M., & De Freitas, N. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Application to AlphaGO



- During the development of AlphaGo, its many hyperparameters were tuned with **Bayesian optimization** multiple times. This automatic tuning process resulted in substantial improvements in playing strength;
- For example, prior to the match with Lee Sedol, we tuned the latest AlphaGo agent and this improved its **win-rate from 50% to 66.5%** in self-play games. This tuned version was deployed in the final match. Of course, since we tuned AlphaGo many times during its development cycle, the compounded contribution was even higher than this percentage.

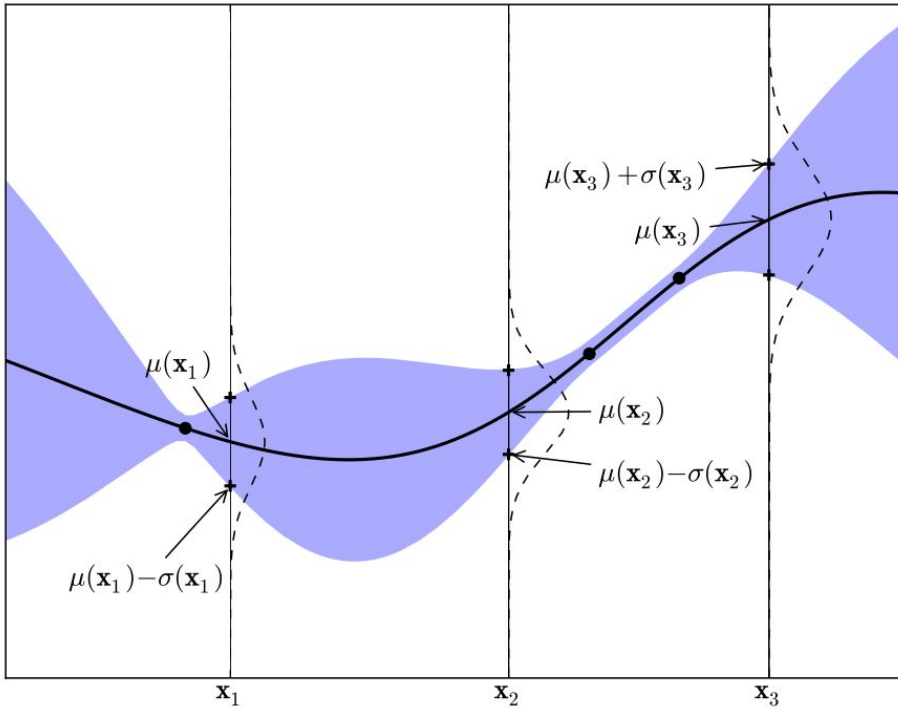
Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D., & de Freitas, N. (2018). Bayesian optimization in alphago. *arXiv preprint arXiv:1812.06855*.

Gaussian Process prior

Function evaluation have join

multivariate normal distribution

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')).$$



Gaussian Process prior, kernels

Squared exponential: $k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$

ARD: $k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T \operatorname{diag}(\boldsymbol{\theta})^{-2}(\mathbf{x} - \mathbf{x}')\right)$

Automatic Relevance Determination

Matern: $k(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{2^{\varsigma-1}\Gamma(\varsigma)}\left(2\sqrt{\varsigma}\,\|\mathbf{x}_i - \mathbf{x}_j\|\right)^{\varsigma} H_{\varsigma}\left(2\sqrt{\varsigma}\,\|\mathbf{x}_i - \mathbf{x}_j\|\right)$

it is generalization of the absolute and squared exponential kernels.

H(.) - is a Bessel function, *Γ(.)* - gamma function, ζ controls smoothness, becomes abs. exp. for ζ=0.5.

Gaussian Process prior

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_t) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_t, \mathbf{x}_1) & \dots & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{f}_{1:t} \\ f_{t+1} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^T & k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \end{bmatrix}\right),$$

$$\mathbf{k} = \begin{bmatrix} k(\mathbf{x}_{t+1}, \mathbf{x}_1) & k(\mathbf{x}_{t+1}, \mathbf{x}_2) & \cdots & k(\mathbf{x}_{t+1}, \mathbf{x}_t) \end{bmatrix}$$

Posterior distribution:

$$\begin{aligned} \mu_t(\mathbf{x}_{t+1}) &= \mathbf{k}^T \mathbf{K}^{-1} \mathbf{f}_{1:t} \\ \sigma_t^2(\mathbf{x}_{t+1}) &= k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k} \end{aligned}$$

Acquisition functions

Probability of improvement: $$\begin{aligned} \mathrm{PI}(\mathbf{x}) &= P(f(\mathbf{x}) \geq f(\mathbf{x}^+)) \\ &= \Phi\left(\frac{\mu(\mathbf{x}) - f(\mathbf{x}^+)}{\sigma(\mathbf{x})}\right) \end{aligned}$$

Expected improvement: $$\mathbb{E}(\max\{0, f_{t+1}(\mathbf{x}) - f(\mathbf{x}^+)\} \,|\mathcal{D}_t)$$

Confidence bound: $$\mathrm{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(x)$$

Note: for function maximization

Bayesian Optimization algorithm

Algorithm 1 Bayesian Optimization

1: **for** $t = 1, 2, \ldots$ **do**
2: Find $\mathbf{x}_t$ by optimizing the acquisition function over the GP: $\mathbf{x}_t = \text{argmax}_{\mathbf{x}}\, u(\mathbf{x}|\mathcal{D}_{1:t-1})$.
3: Sample the objective function: $y_t = f(\mathbf{x}_t) + \varepsilon_t$.
4: Augment the data $\mathcal{D}_{1:t} = \{\mathcal{D}_{1:t-1}, (\mathbf{x}_t, y_t)\}$ and update the GP.
5: **end for**

Acquisition functions

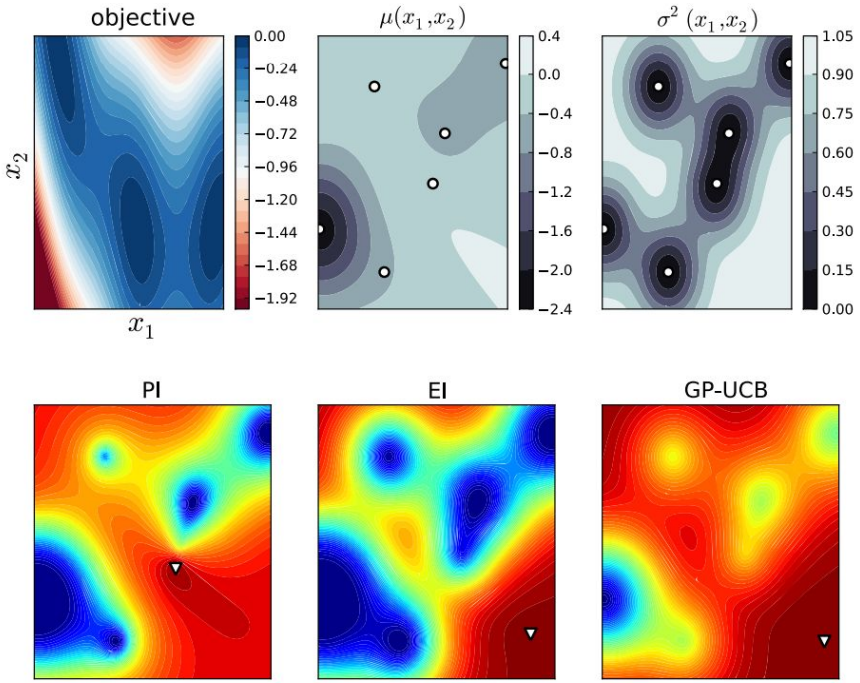


Figure 6: *Examples of acquisition functions and their settings in 2 dimensions. The top row shows the objective function (which is the Branin function here), and the posterior mean and variance estimates $\mu(\cdot)$ and $\sigma^2(\cdot)$. The samples used to train the GP are shows with white dots. The second row shows the acquisition functions for the GP. From left to right: probability of improvement (Eqn (2)), expected improvement (Eqn (4)) and upper confidence bound (Eqn (5)). The maximum of each function is shown with a triangle marker.*

Acquisition functions

Maximum entropy search (MES). The acquisition function is the gain in mutual information between the maximum y_* and the next point ***{x, y}*** to query:

$$\alpha_t(x) = I(\{\boldsymbol{x}, y\}; y_* \mid D_t)$$
$$= H(p(y \mid D_t, \boldsymbol{x})) - \mathbb{E}[H(p(y \mid D_t, \boldsymbol{x}, y_*))]$$

Wang, Z., & Jegelka, S. (2017). Max-value entropy search for efficient Bayesian optimization. In *International Conference on Machine Learning* (pp. 3627-3635). PMLR.

Problems with Gaussian Processes

- Poor performance in high-dimensional spaces;
- Computationally hard for large samples (cubic complexity);
- Hard to apply to structured discrete spaces.

Alternatives:

- SMAC, uncertainty estimation with Random Forests;
- Tree-structured Pazen estimators;
- Bayesian NN’s, work well on small-dimensional data.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011, January). Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization* (pp. 507-523). Springer, Berlin, Heidelberg.

Tree-structured Parzen estimator (TPE)

$$p(x|y) = \begin{cases} \ell(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases} \qquad p(y < y^*) = \gamma$$

fraction of “good” points, typically 0.1-0.2

l(x) - pdf of “good” points

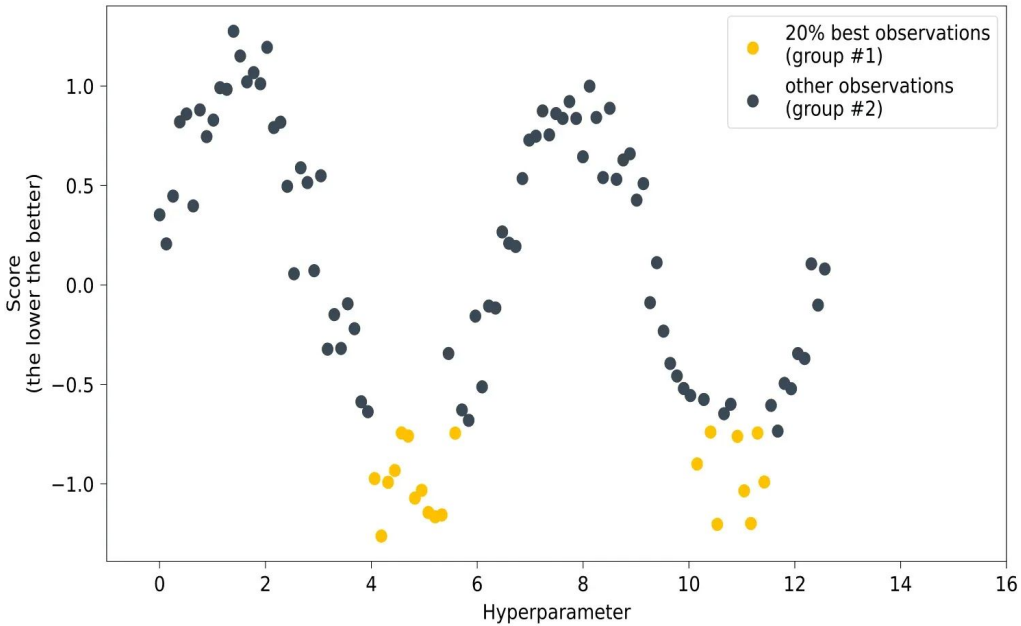
g(x) - pdf of “bad” points

l(x) and *g(x)* are estimated by non-parametric KDE.

$$\widehat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *NIPS*.
Watanabe, S. (2023). Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance.

Tree-structured Parzen estimator (TPE)



Tree-structured Parzen estimator (TPE)

The parametrization of $p(x, y)$ as $p(y)p(x|y)$ in the TPE algorithm was chosen to facilitate the optimization of EI.

$$\mathrm{EI}_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)p(y|x)dy = \int_{-\infty}^{y^*} (y^* - y)\frac{p(x|y)p(y)}{p(x)}dy$$

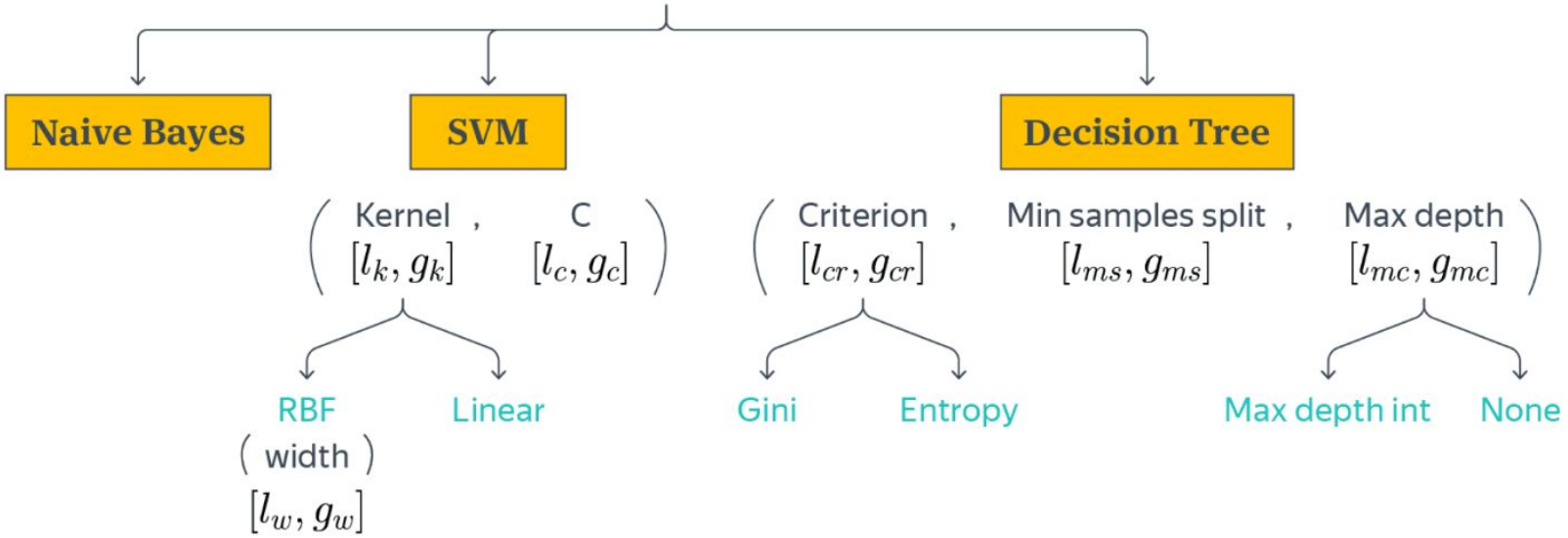
By construction, $\gamma = p(y < y^*)$ and $p(x) = \int_{\mathbb{R}} p(x|y)p(y)dy = \gamma\ell(x) + (1-\gamma)g(x)$. Therefore

$$\int_{-\infty}^{y^*} (y^* - y)p(x|y)p(y)dy \quad = \quad \ell(x)\int_{-\infty}^{y^*} (y^* - y)p(y)dy = \gamma y^*\ell(x) - \ell(x)\int_{-\infty}^{y^*} p(y)dy,$$

so that finally $EI_{y^*}(x) = \frac{\gamma y^*\ell(x) - \ell(x)\int_{-\infty}^{y^*} p(y)dy}{\gamma\ell(x) + (1-\gamma)g(x)} \propto \left(\gamma + \frac{g(x)}{\ell(x)}(1-\gamma)\right)^{-1}$.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, *24*.

Tree-structured Parzen estimator (TPE)



Optimization of EI is done sequentially at each level.

https://academy.yandex.ru/handbook/ml/article/podbor-giperparametrov

Tree-structured Parzen estimator (TPE)

At each step, evaluate point with minimum $g(x)/\ell(x)$

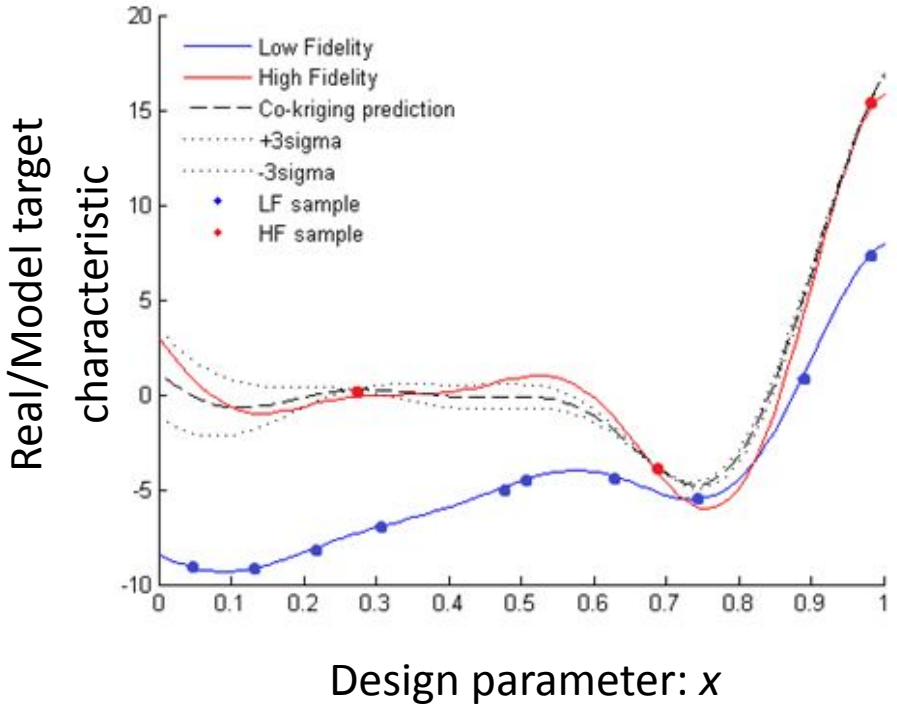
Pros: scalable, robust, works with conditional subspaces;

Cons: no so sample-efficient as BayesOpt with GP.

Multi-fidelity optimization

Low-fidelity

High-fidelity



Multi-fidelity optimization

In many situations, one has several approximations of target with variable fidelity and computational cost:

$$y^{(k)}(x),\ y^{(k-1)}(x),\ldots,\ y^{(1)}(x)$$

Cheap approximations of the black-box, performance on which correlates with the black-box:

- train of subsets of the data;
- train for fewer epochs of iterative training algorithms (e.g., SGD);
- shorter MCMC chains in Bayesian deep learning;
- fewer trials in deep reinforcement learning;
- downsampled images in object recognition.

Also applicable in different domains, e.g., fluid simulations:

- Less particles;
- Shorter simulations;
- Coarse grid.

Hyperband

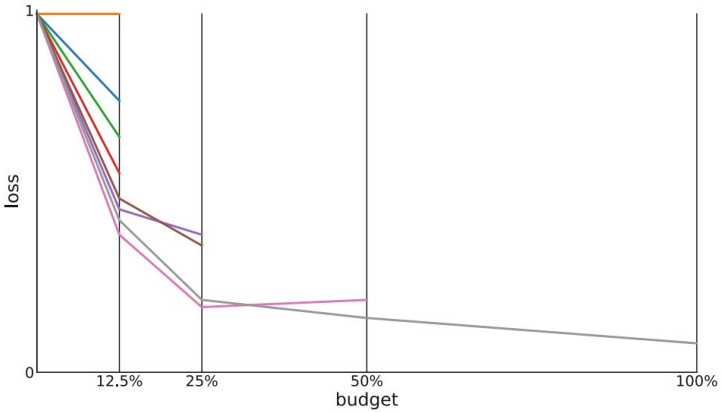


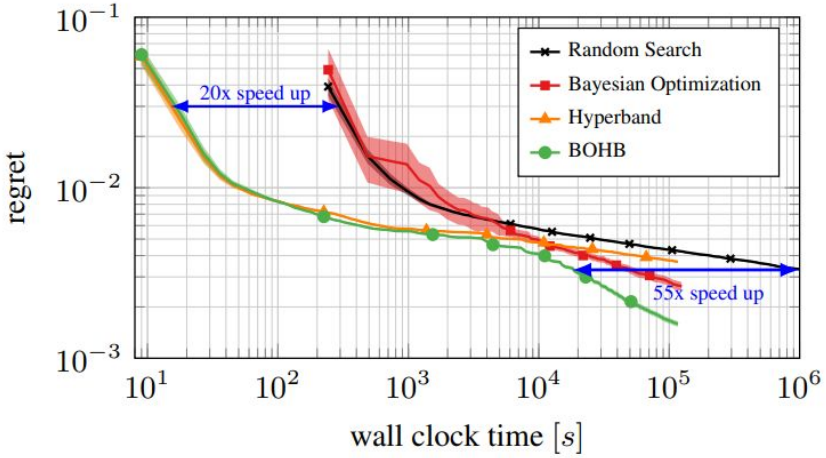
Fig. 1.3 Illustration of successive halving for eight algorithms/configurations. After evaluating all algorithms on $\frac{1}{8}$ of the total budget, half of them are dropped and the budget given to the remaining algorithms is doubled

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
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								

n_i - number of configurations, r_i - resources

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, *18*(1), 6765-6816.

BOHB: Robust and Efficient Hyperparameter Optimization at Scale



In opposite to Hyperband, configurations are not sampled randomly but with probabilities proportional to Expected Improvement from TPE

$$\mathrm{EI}_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y) p(y|x) dy$$

$$EI_{y^*}(x) = \frac{\gamma y^* \ell(x) - \ell(x) \int_{-\infty}^{y^*} p(y) dy}{\gamma \ell(x) + (1-\gamma) g(x)} \propto \left(\gamma + \frac{g(x)}{\ell(x)}(1-\gamma)\right)^{-1}$$

$$l(\boldsymbol{x}) = p(y < \alpha | \boldsymbol{x}, D)$$
$$g(\boldsymbol{x}) = p(y > \alpha | \boldsymbol{x}, D)$$

Falkner, S., Klein, A., Hutter, F.: BOHB: Robust and Efficient Hyperparameter Optimization at Scale. ICML' 18.

Co-kriging fusion modeling

High-fidelity data Low-fidelity data

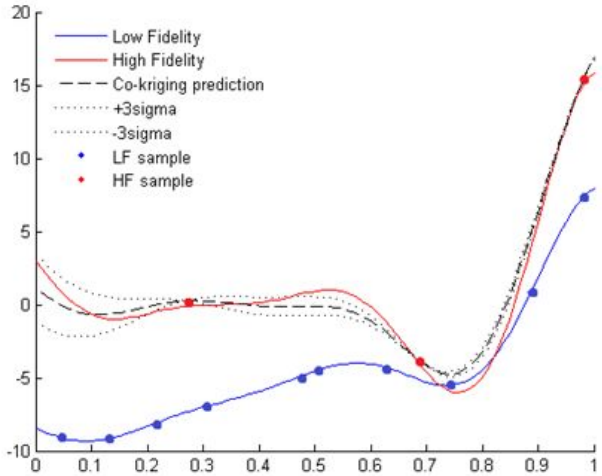
$$y^{(2)}(x) = \rho y^{(1)}(x) + \delta(x)$$

- Fit co-kriging fusion model
- $y^{(2)}(x), y^{(1)}(x), \delta(x)$ are approximated with Gaussian Processes Regression

  $f \sim GP(\mu(x), k_\theta(x, x'))$

  where $\mu(x)$ is a mean function,

  $k_\theta(x, x')$ is a covariance function.



Klyuchnikov, N., Mottin, D., Koutrika, G., Müller, E., & Karras, P. Figuring out the user in a few steps: Bayesian multifidelity active search with cokriging. KDD'19.

Multi-Fidelity Bayesian Optimization via Deep Neural Networks

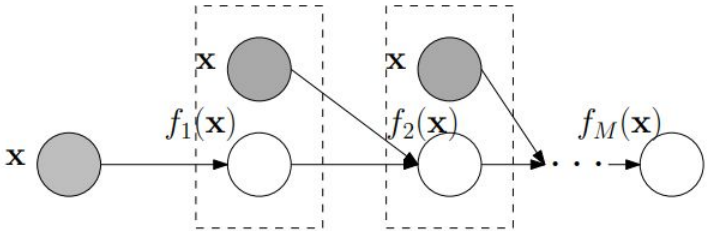


Figure 1: Graphical representation of the DNN based multi-fidelity surrogate model. The output in each fidelity $f_m(\mathbf{x})$ $(1 \leq m \leq M)$ is fulfilled by a (deep) neural network.

$$p(\mathcal{W}, \mathcal{Y}|\mathcal{X}, \Theta, \mathbf{s}) = \prod_{m=1}^{M} \mathcal{N}(\mathbf{w}_m|\mathbf{0}, \mathbf{I}) \prod_{n=1}^{N_m} \mathcal{N}\left(y_{nm}|f_m(\mathbf{x}_{nm}), \sigma_m^2\right),$$

$$\mathbf{x}_m = [\mathbf{x}; f_{m-1}(\mathbf{x})], \quad f_m(\mathbf{x}) = \mathbf{w}_m^\top \boldsymbol{\phi}_{\boldsymbol{\theta}_m}(\mathbf{x}_m), \quad y_m(\mathbf{x}) = f_m(\mathbf{x}) + \epsilon_m,$$

m networks, each of them does regression for the fidelity *m;*

Li, S., Xing, W., Kirby, R., & Zhe, S. (2020). Multi-Fidelity Bayesian Optimization via Deep Neural Networks. *Advances in Neural Information Processing Systems*, *33*

Multi-Fidelity Bayesian Optimization via Deep Neural Networks

Acquisition function (max-value entropy):

$$a(\mathbf{x}, m) = \frac{1}{\lambda_m} I\left(f^*, f_m(\mathbf{x})|\mathcal{D}\right) = \frac{1}{\lambda_m}\left(H\left(f_m(\mathbf{x})|\mathcal{D}\right) - \mathbb{E}_{p(f^*|\mathcal{D})}\left[H\left(f_m(\mathbf{x})|f^*, \mathcal{D}\right)\right]\right)$$

$\lambda_m > 0$ is the cost of querying with fidelity m.

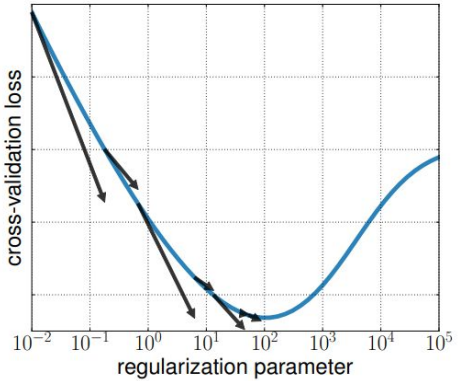
m, x = argmax $f_m(x)$*

Complex math is used to approximate posterior (including stochastic variational inference, reparameterization trick, Gauss-Hermite quadrature, etc.)

Differentiable hyperparameter optimization

$$\min_{\lambda \in \Lambda} \quad \mathcal{L}_{val}(w^*(\lambda), \lambda)$$
$$\text{subject to} \quad w^*(\lambda) \in \arg\min_{w} \mathcal{L}_{train}(w, \lambda)$$

$$\frac{d\mathcal{L}_{val}}{d\lambda} = \underbrace{\frac{\partial \mathcal{L}_{val}}{\partial \lambda}}_{\text{Direct effect}} + \underbrace{\frac{\partial \mathcal{L}_{val}}{\partial w} \cdot \frac{dw^*(\lambda)}{d\lambda}}_{\text{Indirect effect through weights}}$$



Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. In *International conference on machine learning* (pp. 737-746). PMLR.

Differentiable hyperparameter optimization

$$\nabla_w \mathcal{L}_{train}(w^*(\lambda), \lambda) = 0$$

Derivative of the implicit function:

$$\frac{d}{d\lambda}\left(\nabla_w \mathcal{L}_{train}(w^*(\lambda), \lambda)\right) = \nabla^2_{ww}\mathcal{L}_{train} \cdot \frac{dw^*(\lambda)}{d\lambda} + \nabla^2_{w\lambda}\mathcal{L}_{train} = 0$$

$$\boxed{\frac{dw^*(\lambda)}{d\lambda} = -\left[\nabla^2_{ww}\mathcal{L}_{train}\right]^{-1} \cdot \nabla^2_{w\lambda}\mathcal{L}_{train}}$$

$$\boxed{\frac{d\mathcal{L}_{val}}{d\lambda} = \frac{\partial \mathcal{L}_{val}}{\partial \lambda} - \frac{\partial \mathcal{L}_{val}}{\partial w} \cdot \left[\nabla^2_{ww}\mathcal{L}_{train}\right]^{-1} \cdot \nabla^2_{w\lambda}\mathcal{L}_{train}}$$

hypergradient

Hyperparameters optimization software

Scikit-learn: grid search, random search;

Optuna: grid search, random search, TPE. https://optuna.org

BOHB: https://github.com/automl/HpBandSter

GP Bayesian optimization:

Spearmint: https://github.com/JasperSnoek/spearmint

Metric Optimization Engine: https://github.com/Yelp/MOE

https://github.com/fmfn/BayesianOptimization

SIGOPT: https://sigopt.com // Web-API

Thank you for attention!