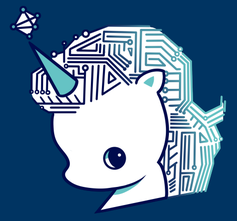
Hyperparameter Optimization

HPO 를 위한 이론, 알고리즘 및 응용 사례

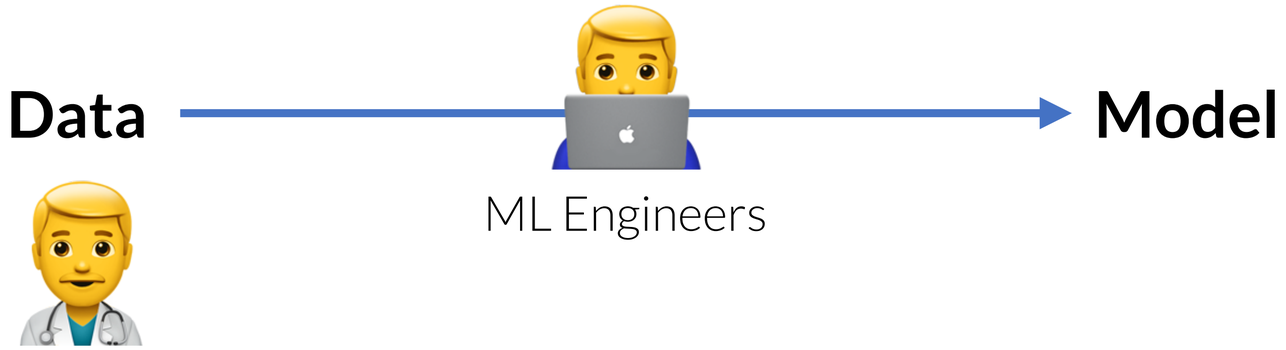


Sungbin Lim (삼성전자 종합기술원, 2021. 11. 09)

Automated Machine Learning

- Millions of organizations worldwide has ML problems

Hand-crafted
Machine Learning

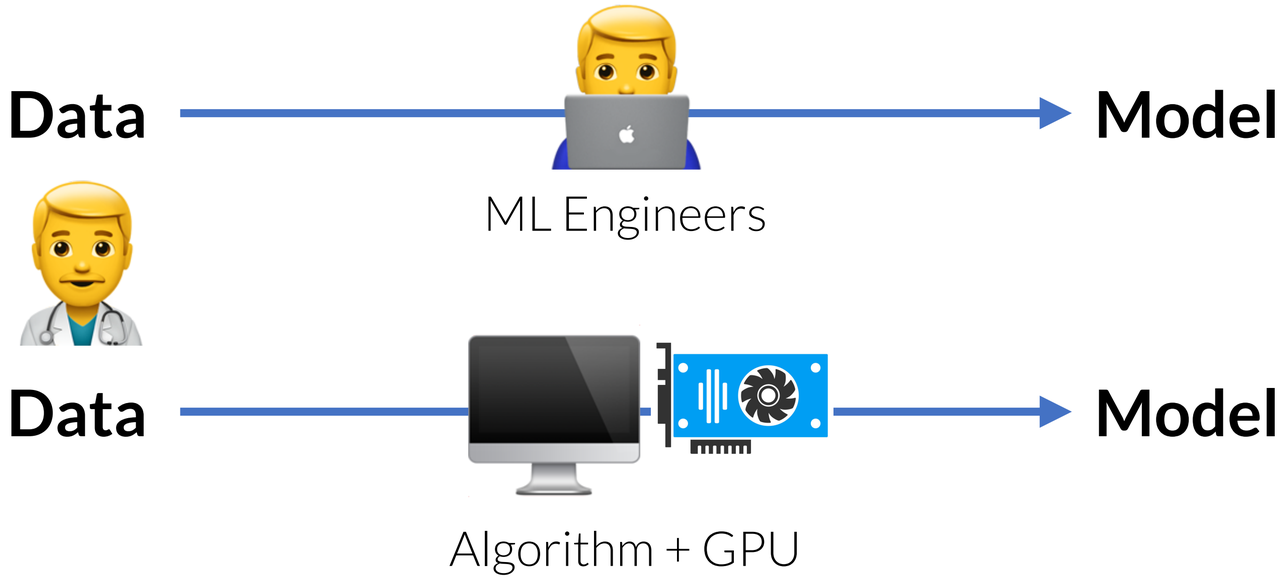


Automated Machine Learning

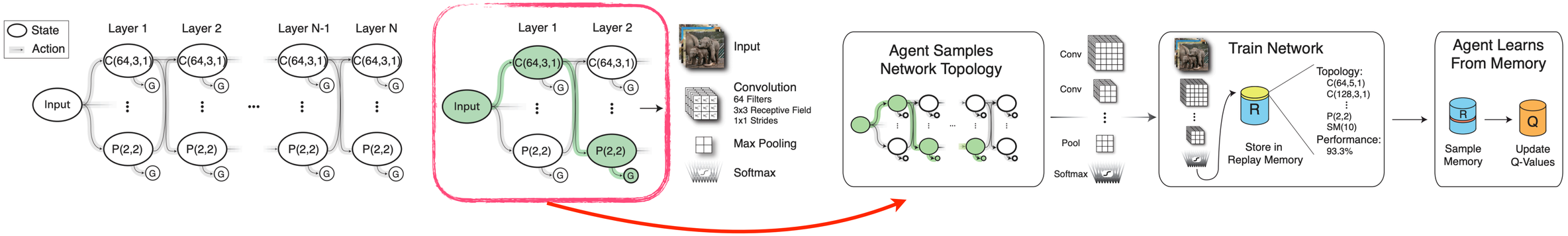
- Millions of organizations worldwide has ML problems
- AutoML allows **non-experts** to solve ML problems

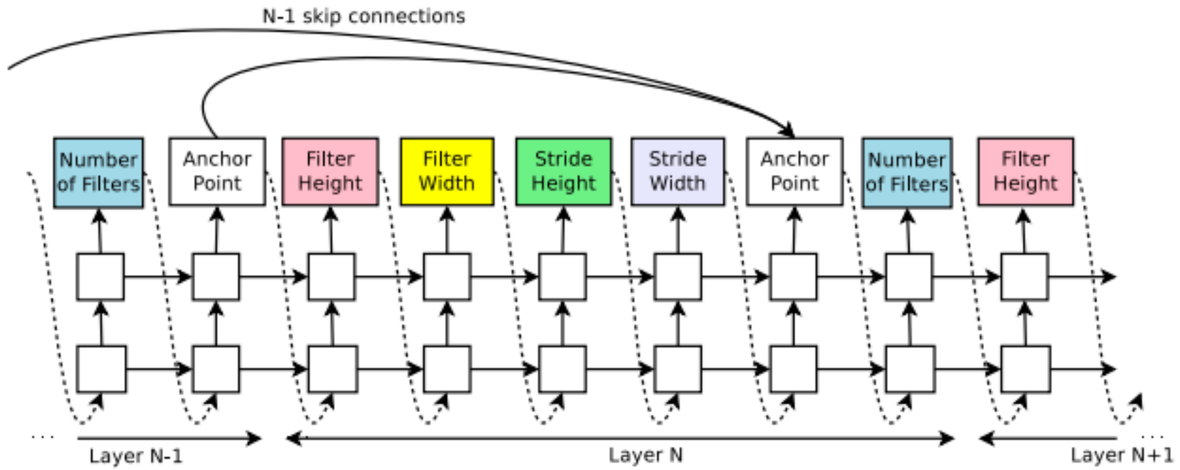
Hand-crafted
Machine Learning

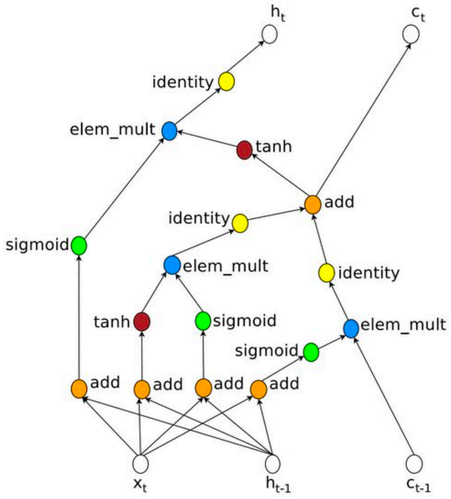
Automated
Machine Learning

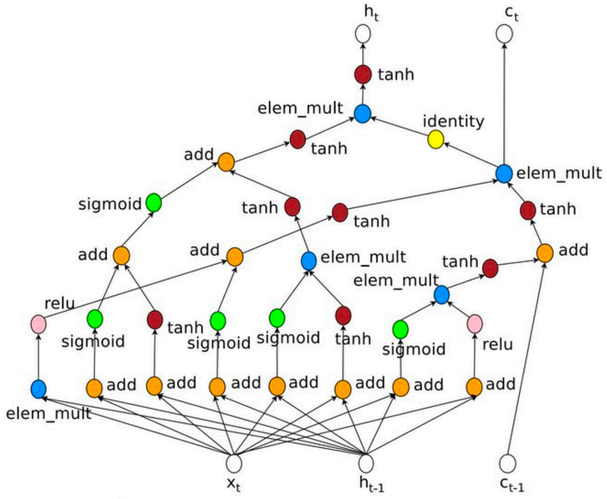


Neural Architecture Search

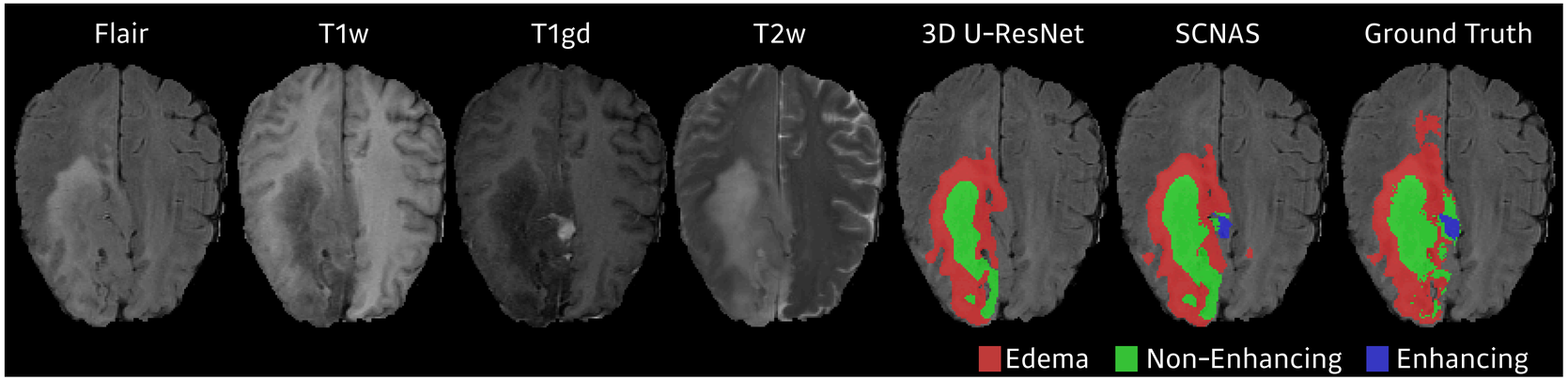


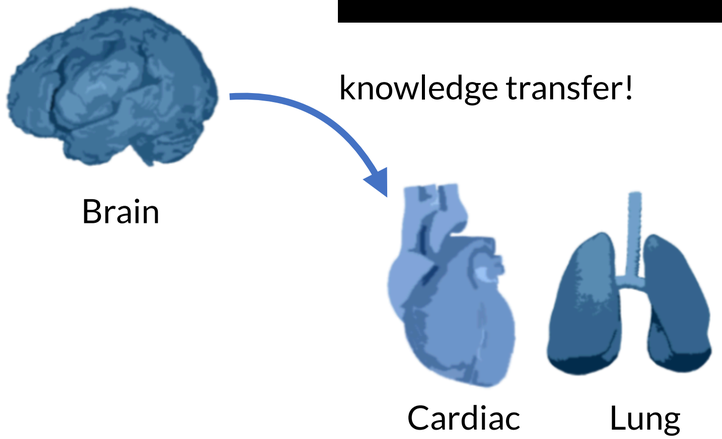






NAS for 3D Medical Images



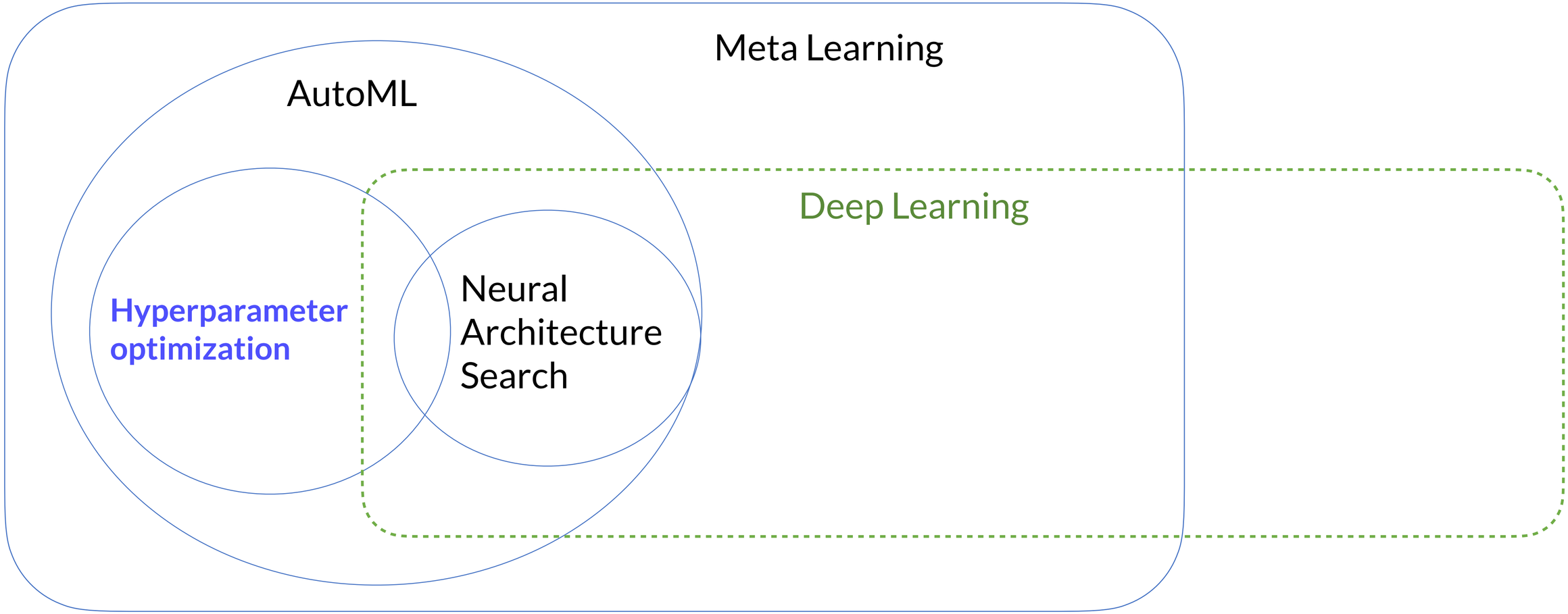


	Brain Tumor (MRI)				Heart (MRI)	Lung (CT)
Label	Edema	Non-Enhancing	Enhancing	Average	Left Atrium	Tumor
3D nnU-Net [2]	80.71	62.22	79.07	74.00	92.45	55.87
3D U-ResNet	70.74	56.69	73.23	66.89	91.48	63.28
SCNAS	80.41	59.85	78.50	72.92	91.29	64.82
SCNAS(transfer)	-	-	-	-	91.91	68.62

Scalable Neural Architecture Search for 3D Medical Image Segmentation, ***MICCAI*** (2019)

Joint work with S. Kim (Kakao Brain), I. Kim (Kakao Brain), W. Baek (Kakao Brain), C. Kim (Kakao Brain), H. Cho (SNU), B. Yoon (Kakao Brain), T. Kim (MILA)

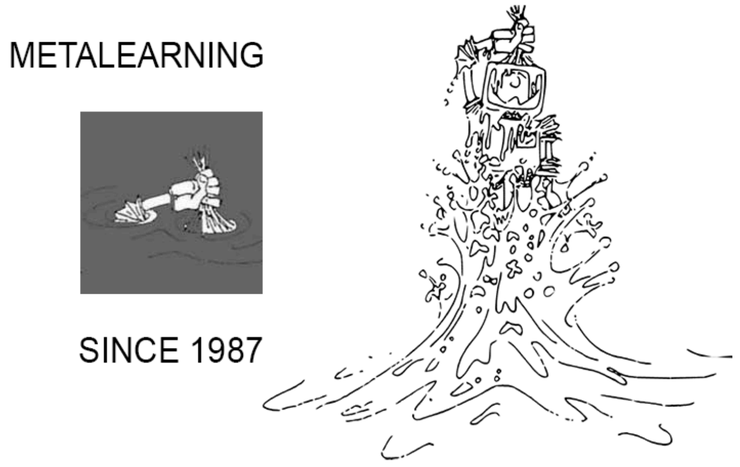
Vénn Diagram for AutoML



Theory & Algorithm
Hyperparameter Optimization

Learning vs Meta Learning

- Learning Algorithm A
  - input: $\mathcal{D}_{\texttt{train}} = \{(x_i, y_i)\}$
  - output: M
  - objective: $\mathcal{D}_{\texttt{test}} = \{(\tilde{x}_i, \tilde{y}_i)\}$
- Meta Learning Algorithm $\mathfrak{A}$
  - input: $\mathfrak{D}_{\texttt{meta-train}} = \{(\mathcal{D}^{(t)}_{\texttt{train}}, \mathcal{D}^{(t)}_{\texttt{test}})\}_{t=1}^{T}$
  - output: A
  - objective: $\mathfrak{D}_{\texttt{meta-test}} = \{(\tilde{\mathcal{D}}^{(t)}_{\texttt{train}}, \tilde{\mathcal{D}}^{(t)}_{\texttt{test}})\}_{t=1}^{\tilde{T}}$



Jürgen Schmidhuber (1987)

Definition of HPO

- Bilevel formulation for HPO

model $g_w : \mathcal{X} \rightarrow \mathcal{Y}$

HPO 대상인 기계학습 모델 외
최적화 solver 등을 포함한다

Bilevel optimization framework

$$\min\{f(\lambda) : \lambda \in \Lambda\}, \tag{1}$$

where function $f : \Lambda \rightarrow \mathbb{R}$ is defined at $\lambda \in \Lambda$ as

$$f(\lambda) = \inf\{E(w_\lambda, \lambda) : w_\lambda \in \arg\min_{u \in \mathbb{R}^d} L_\lambda(u)\}. \tag{2}$$

Bilevel programming	Hyperparameter optimization	Meta-learning
Inner variables	Parameters	Parameters of Ground models
Outer variables	Hyperparameters	Parameters of Meta-learner
Inner objective	Training error	Training errors on tasks (Eq. 3)
Outer objective	Validation error	Meta-training error (Eq. 4)

Bilevel Programming for Hyperparameter Optimization and Meta-Learning, Franceschi et al., ***ICML*** 2018

Definition of HPO

- Bilevel formulation for HPO

model $g_w : \mathcal{X} \rightarrow \mathcal{Y}$

inner objective $L_\lambda(w) = \sum_{(x,y)\in D_{\text{tr}}} \ell(g_w(x), y) + \Omega_\lambda(w)$

일반적인 기계학습에서 사용되는
학습데이터 상 loss 함수를 의미한다

Bilevel optimization framework

$$\min\{f(\lambda) : \lambda \in \Lambda\}, \quad (1)$$

where function $f : \Lambda \rightarrow \mathbb{R}$ is defined at $\lambda \in \Lambda$ as

$$f(\lambda) = \inf\{E(w_\lambda, \lambda) : w_\lambda \in \arg\min_{u\in\mathbb{R}^d} L_\lambda(u)\}. \quad (2)$$

Bilevel programming	Hyperparameter optimization	Meta-learning
Inner variables	Parameters	Parameters of Ground models
Outer variables	Hyperparameters	Parameters of Meta-learner
Inner objective	Training error	Training errors on tasks (Eq. 3)
Outer objective	Validation error	Meta-training error (Eq. 4)

Bilevel Programming for Hyperparameter Optimization and Meta-Learning, Franceschi et al., ***ICML*** 2018

Definition of HPO

- Bilevel formulation for HPO

model $g_w : \mathcal{X} \to \mathcal{Y}$

inner objective $L_\lambda(w) = \sum_{(x,y)\in D_{\text{tr}}} \ell(g_w(x), y) + \Omega_\lambda(w)$

outer objective $E(w, \lambda) = \sum_{(x,y)\in D_{\text{val}}} \ell(g_w(x), y).$

Bilevel optimization framework

$$\min\{f(\lambda) : \lambda \in \Lambda\}, \quad (1)$$

where function $f : \Lambda \to \mathbb{R}$ is defined at $\lambda \in \Lambda$ as

$$f(\lambda) = \inf\{E(w_\lambda, \lambda) : w_\lambda \in \arg\min_{u\in\mathbb{R}^d} L_\lambda(u)\}. \quad (2)$$

Bilevel programming	Hyperparameter optimization	Meta-learning
Inner variables	Parameters	Parameters of Ground models
Outer variables	Hyperparameters	Parameters of Meta-learner
Inner objective	Training error	Training errors on tasks (Eq. 3)
Outer objective	Validation error	Meta-training error (Eq. 4)

최적화된 model weight 을 가지고 validation loss 를 측정

Bilevel Programming for Hyperparameter Optimization and Meta-Learning, Franceschi et al., ***ICML*** 2018

Definition of HPO

- Bilevel formulation for HPO

model $g_w : \mathcal{X} \to \mathcal{Y}$

inner objective $L_\lambda(w) = \sum_{(x,y)\in D_{\text{tr}}} \ell(g_w(x), y) + \Omega_\lambda(w)$

outer objective $E(w, \lambda) = \sum_{(x,y)\in D_{\text{val}}} \ell(g_w(x), y).$

Bilevel optimization framework

$$\min\{f(\lambda) : \lambda \in \Lambda\}, \quad (1)$$

where function $f : \Lambda \to \mathbb{R}$ is defined at $\lambda \in \Lambda$ as

$$f(\lambda) = \inf\{E(w_\lambda, \lambda) : w_\lambda \in \arg\min_{u\in\mathbb{R}^d} L_\lambda(u)\}. \quad (2)$$

Bilevel programming	Hyperparameter optimization	Meta-learning
Inner variables	Parameters	Parameters of Ground models
Outer variables	Hyperparameters	Parameters of Meta-learner
Inner objective	Training error	Training errors on tasks (Eq. 3)
Outer objective	Validation error	Meta-training error (Eq. 4)

이 때 두 데이터셋의 분포가 일치해야 HPO 가 가능하다

Bilevel Programming for Hyperparameter Optimization and Meta-Learning, Franceschi et al., ***ICML*** 2018

Why is HPO difficult?

- Traditional optimization techniques are usually unsuitable for HPO
  - usually employed to optimize expensive to evaluate functions
  - objective function is non-convex and has multiple local optima
  - limited information of the gradient i.e. black-box function
  - mixture of variable types
    - conditionality
- Limited time and resources
  - credit allocation

Why is HPO difficult?

- Optimization target lacking smoothness makes derivative-free methods perform poorly (Sparks et al. 2016)
- ML models or Solvers include various types of hyperparameters
  - e.g. learning rate, layer normalization, number of operations
- HPO techniques sometimes use data sampling to approximate values of the objective function
  - function evaluation time is often ignored in black-box optimization algorithms

How to categorize HPO algorithm?

- Estimator with its Objective function
- Search space
- Search strategy
- Performance estimation strategy

How to categorize HPO algorithm?

- Estimator with its Objective function
- Search space
- Search strategy
- Performance estimation strategy

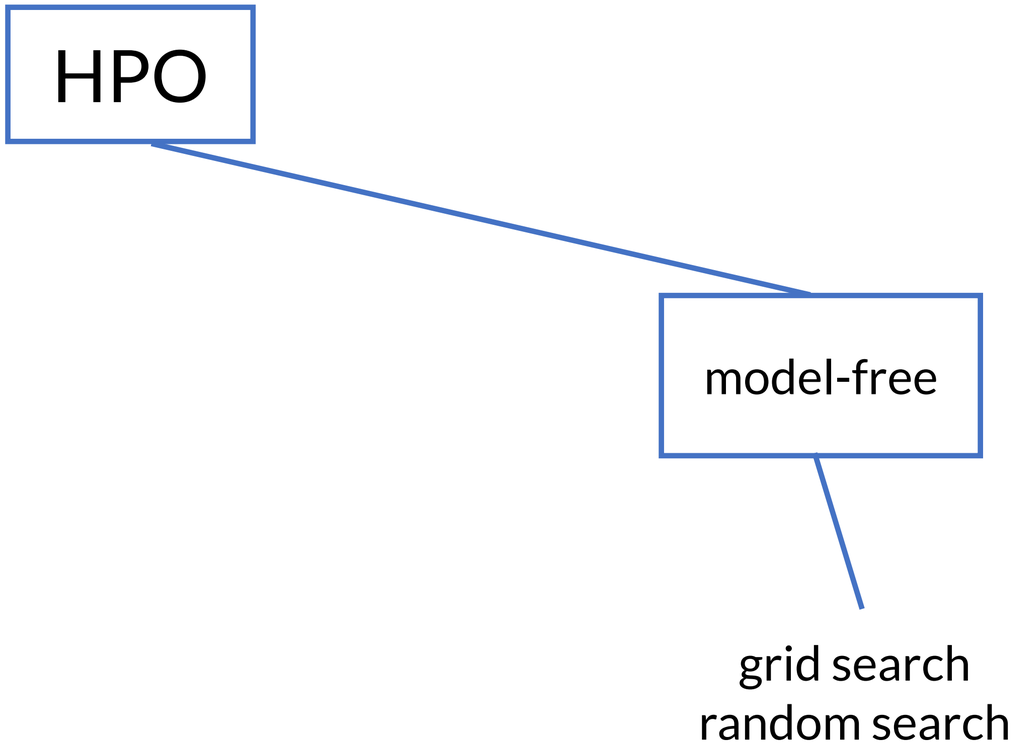
HPO Algorithms Taxonomy

HPO

On Hyperparameter Optimization of Machine Learning Algorithms: Theory & Practice, Yang & Shami, ***Neurocomputing*** 2020

HPO Algorithms Taxonomy

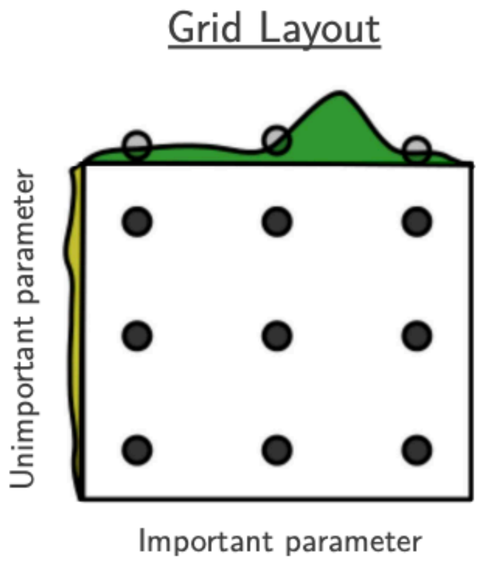
- Model-free methods
  - trial & error
  - grid & random search



On Hyperparameter Optimization of Machine Learning Algorithms: Theory & Practice, Yang & Shami, ***Neurocomputing*** 2020

Grid & Random Search

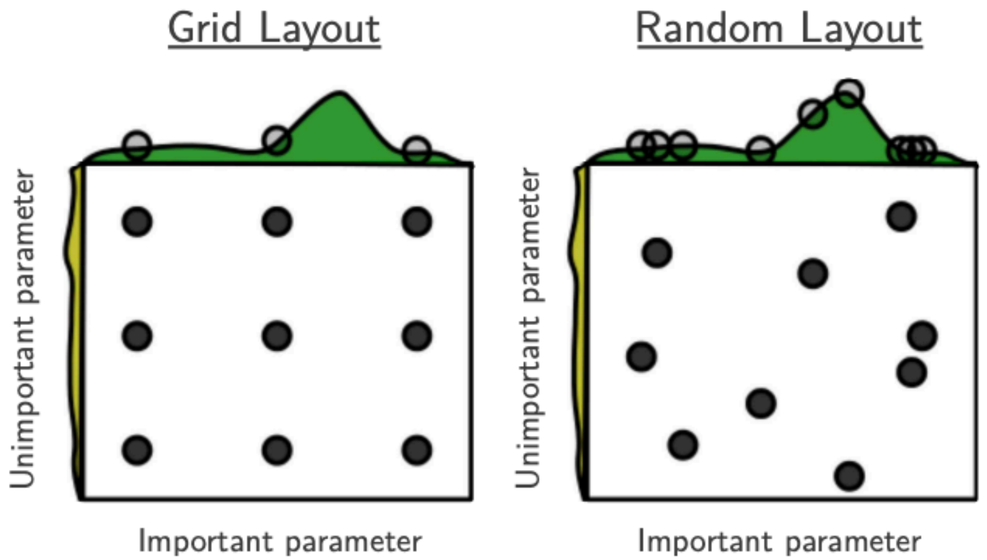
- ***Grid search*** is one of the most commonly used methods
  - it is easy to implement and parallelize
  - curse of dimension $O(n^k)$ → k parameters, n values



Random Search for Hyperparameter Optimization, Bergstra & Bengio, ***JMLR*** 2012

Grid & Random Search

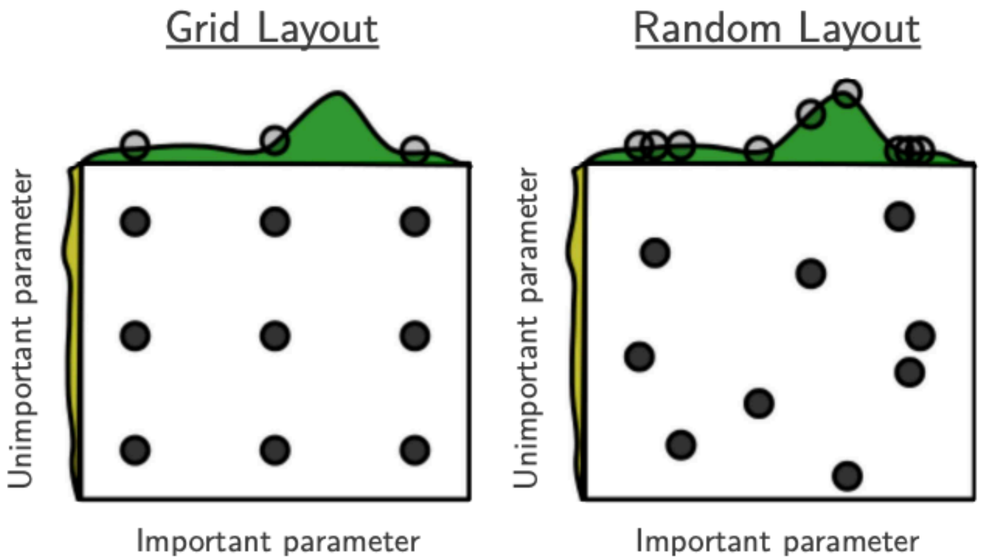
- ***Grid search*** is one of the most commonly used methods
  - it is easy to implement and parallelize
  - curse of dimension $O(n^k)$ → k parameters, n values
- ***Random search*** is similar to grid search, but it is able to explore a larger space

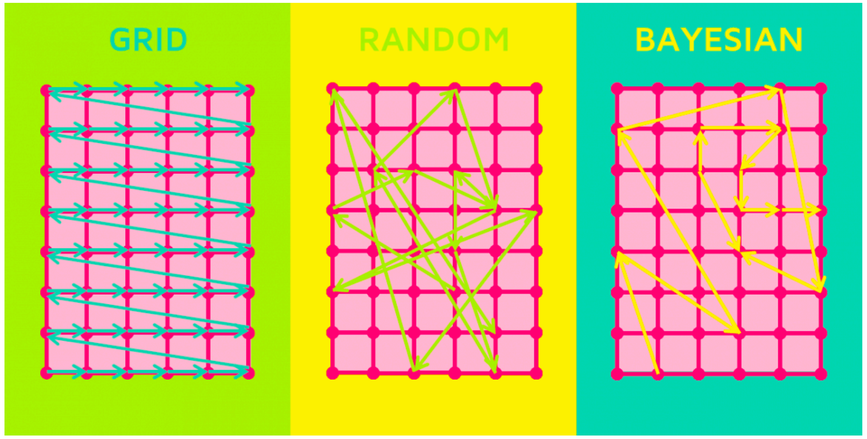


Random Search for Hyperparameter Optimization, Bergstra & Bengio, ***JMLR*** 2012

Grid & Random Search

- ***Grid search*** is one of the most commonly used methods
  - it is easy to implement and parallelize
  - curse of dimension $O(n^k)$ → k parameters, n values
- ***Random search*** is similar to grid search, but it is able to explore a larger space
- Both methods have limitation since every evaluation in their iterations is independent of previous trials

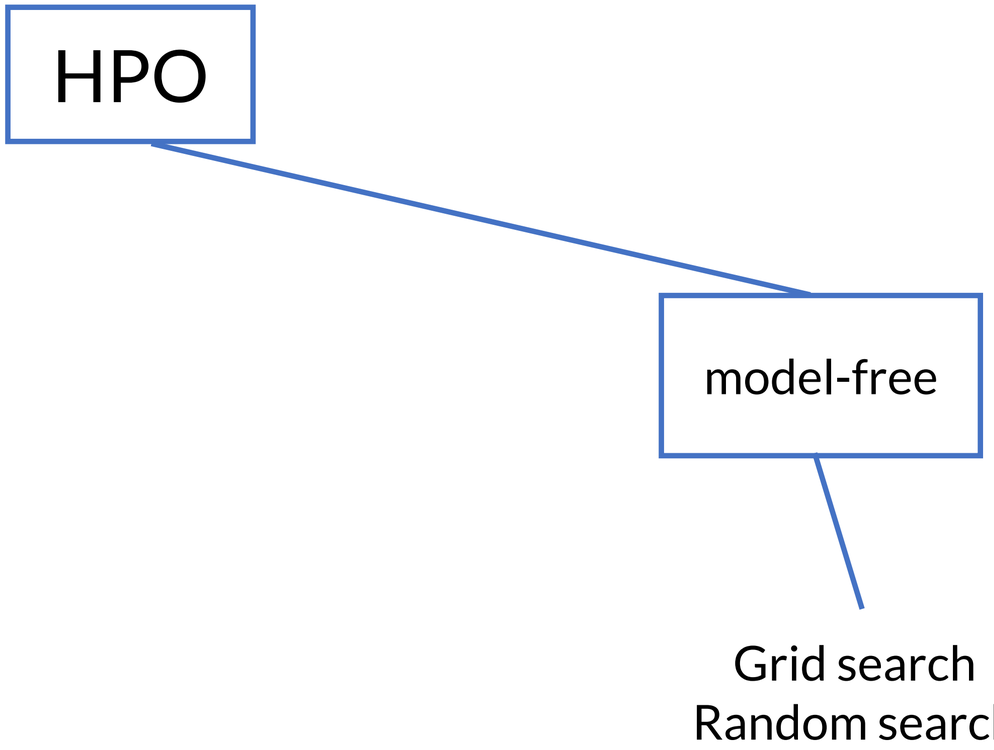




Random Search for Hyperparameter Optimization, Bergstra & Bengio, ***JMLR*** 2012

HPO Algorithms Taxonomy

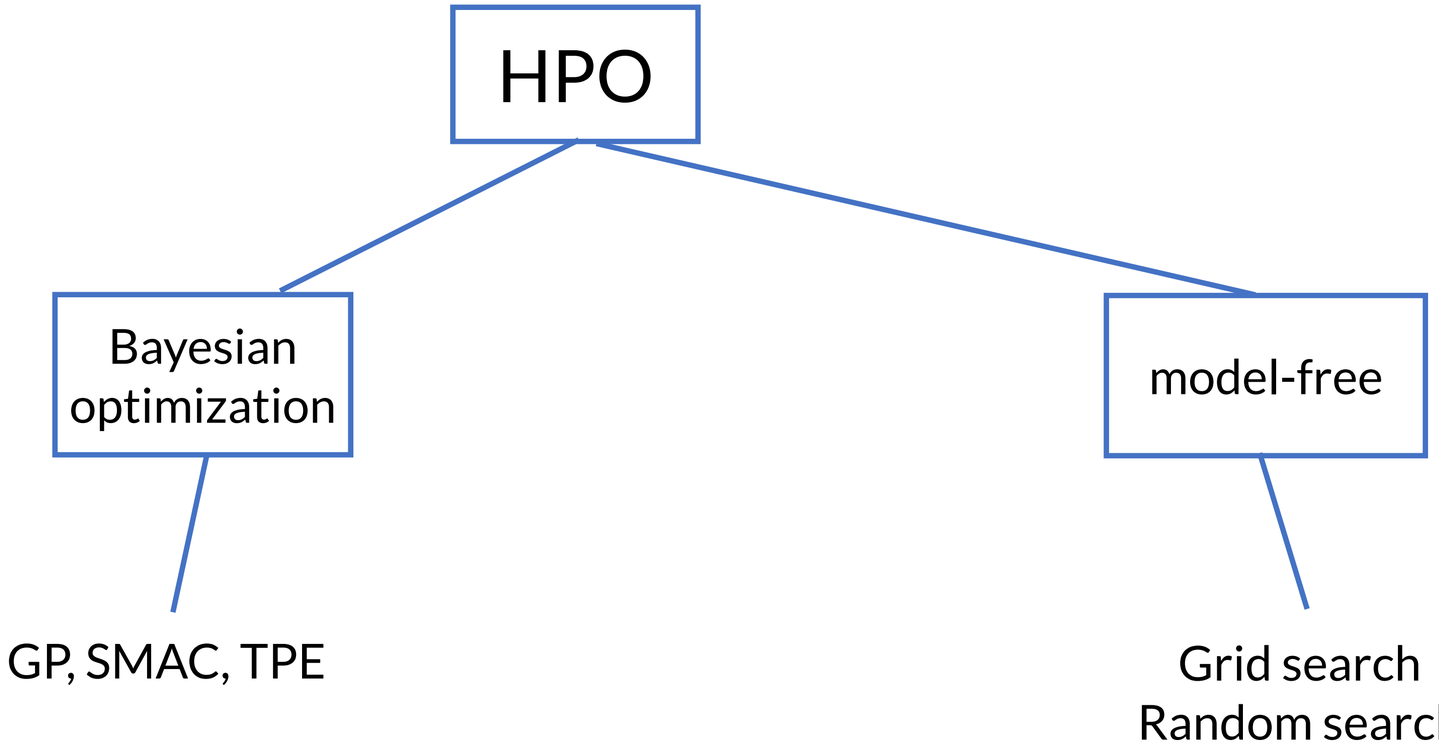
- Model-free methods
  - trial & error
  - grid & random search
- Gradient-based optimization



On Hyperparameter Optimization of Machine Learning Algorithms: Theory & Practice, Yang & Shami, ***Neurocomputing*** 2020

HPO Algorithms Taxonomy

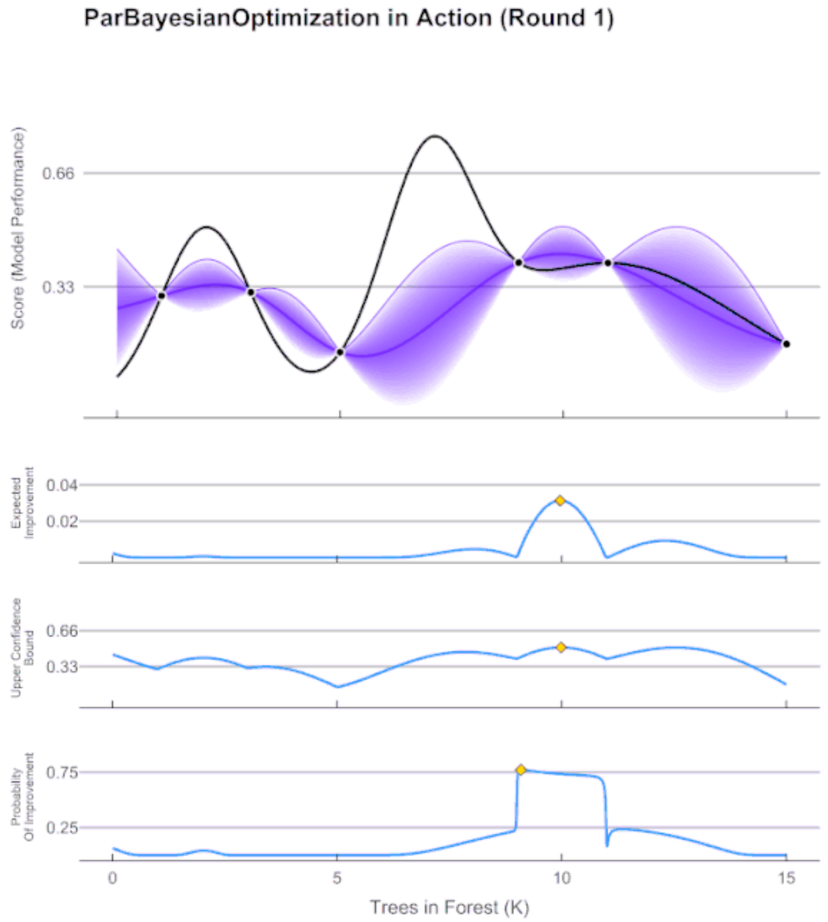
- Model-free methods
  - trial & error
  - grid & random search
- Gradient-based optimization
- Bayesian optimization



On Hyperparameter Optimization of Machine Learning Algorithms: Theory & Practice, Yang & Shami, ***Neurocomputing*** 2020

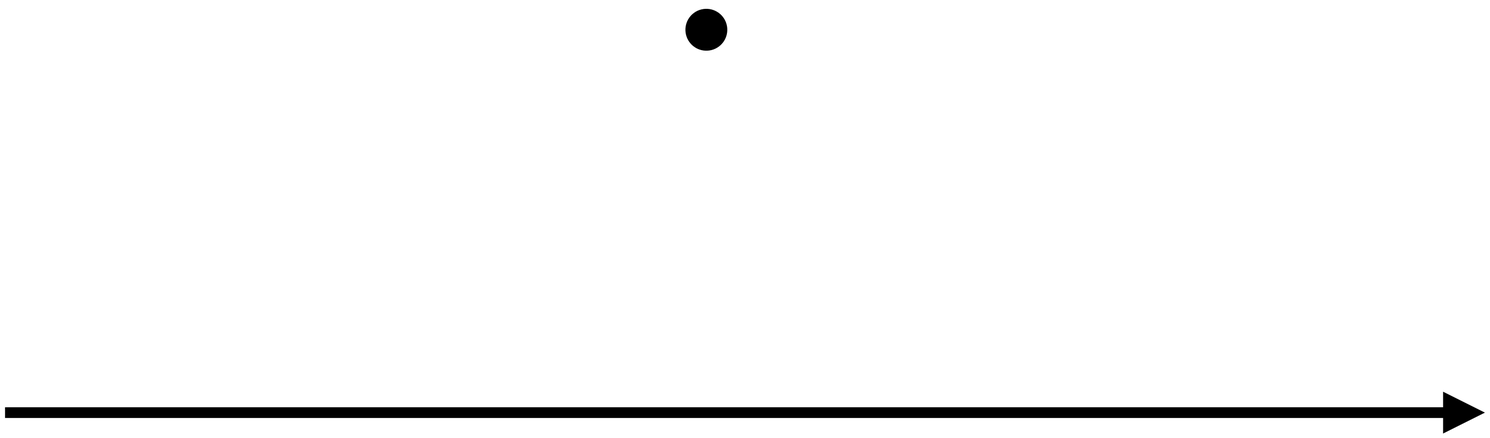
Bayesian Optimization

- BO is an iterative algorithm that is popularly used in HPO problems
  - BO determines the future evaluation points based on the previous results
- ***Surrogate model***
  - fit the observed points into the objective
- ***Acquisition function***
  - determines the usage of points by balancing between exploration and exploitation



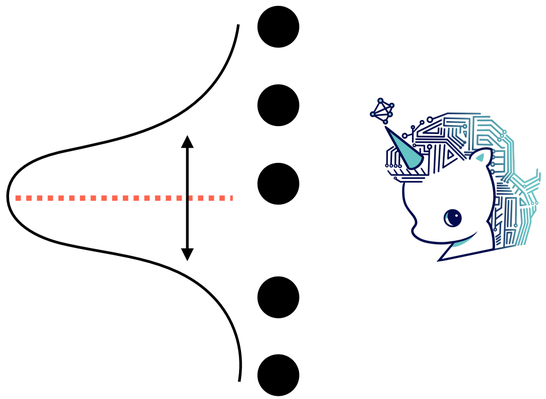
Functional Uncertainty

- Distribution over the space of trajectories or functions



Functional Uncertainty

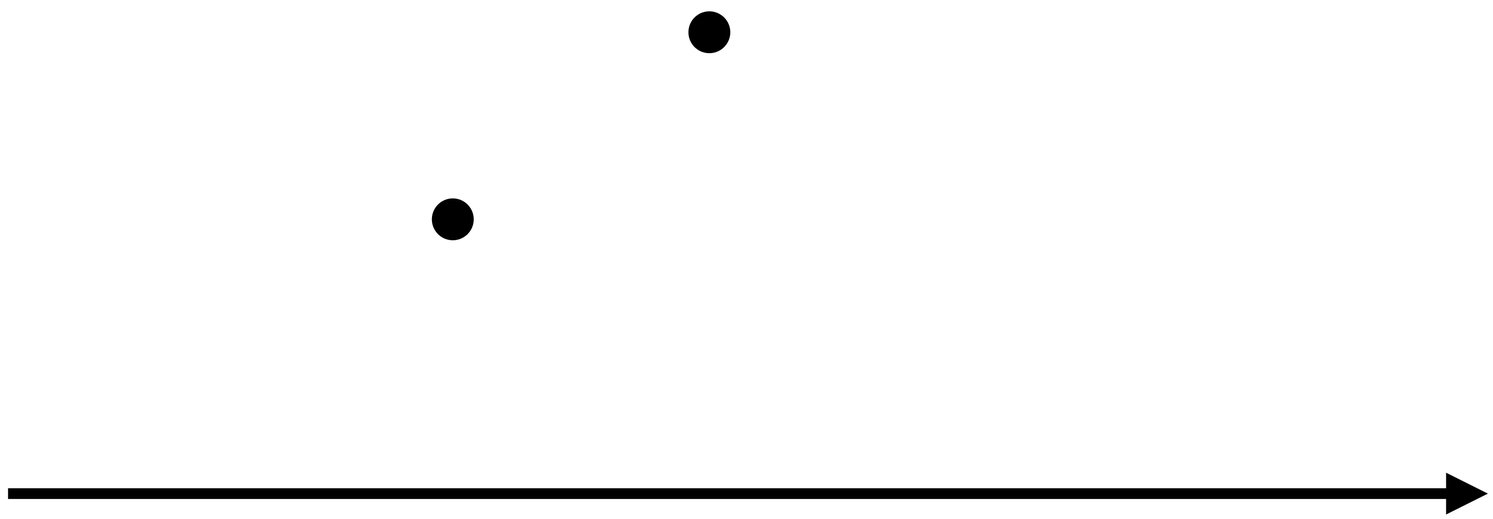
- Distribution over the space of trajectories or functions



this is pretty easy

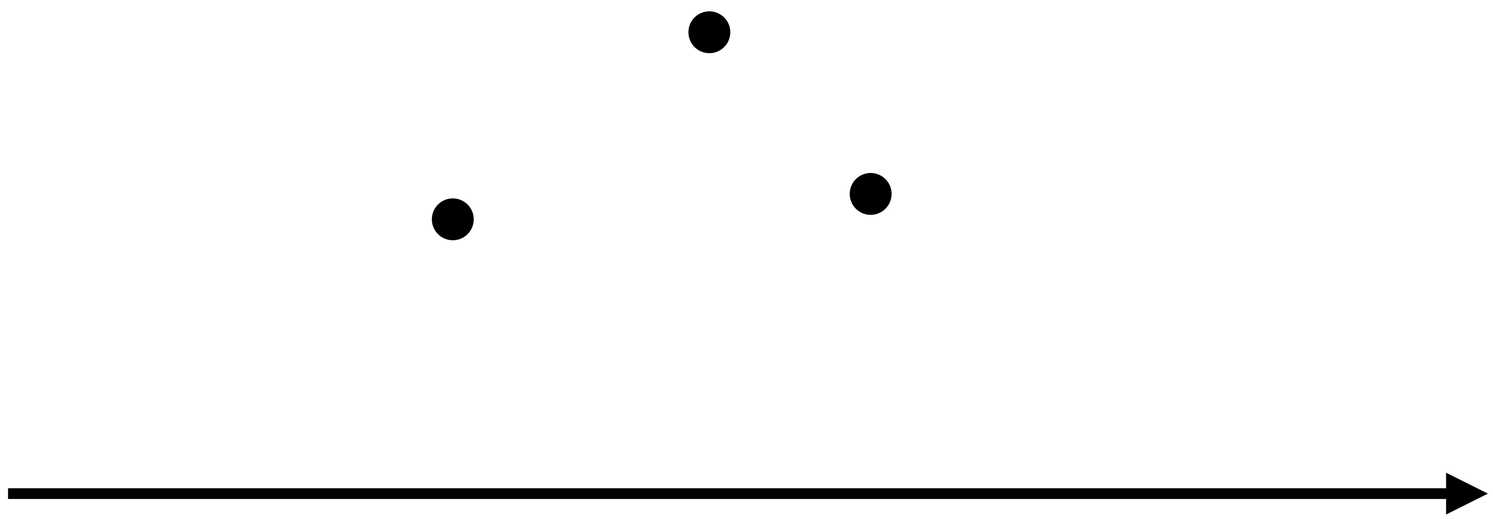
Functional Uncertainty

- Distribution over the space of trajectories or functions



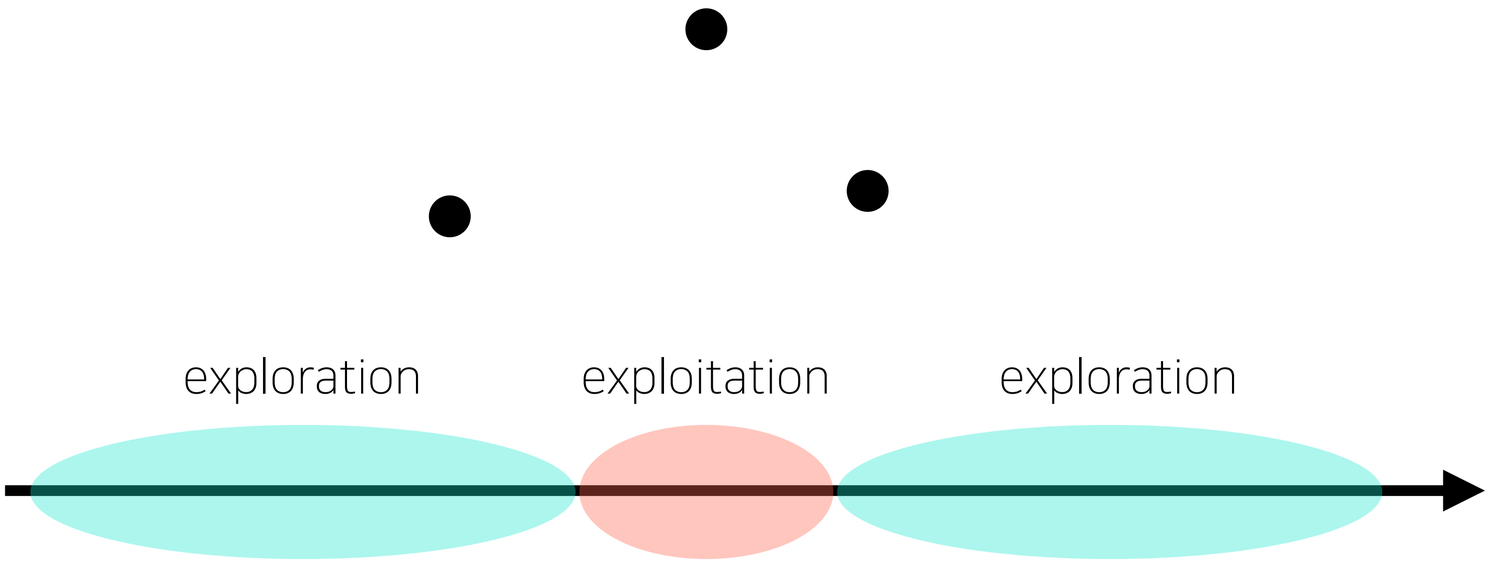
Functional Uncertainty

- Distribution over the space of trajectories or functions



Functional Uncertainty

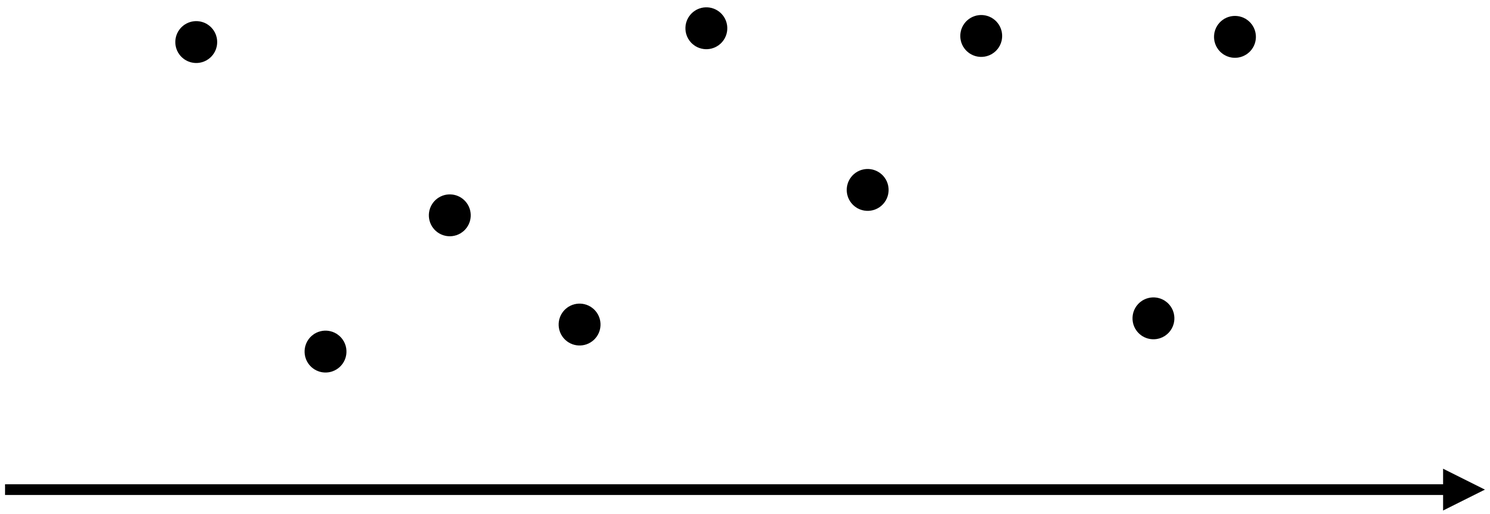
- Distribution over the space of trajectories or functions



Functional Uncertainty

- Distribution over the space of trajectories or functions

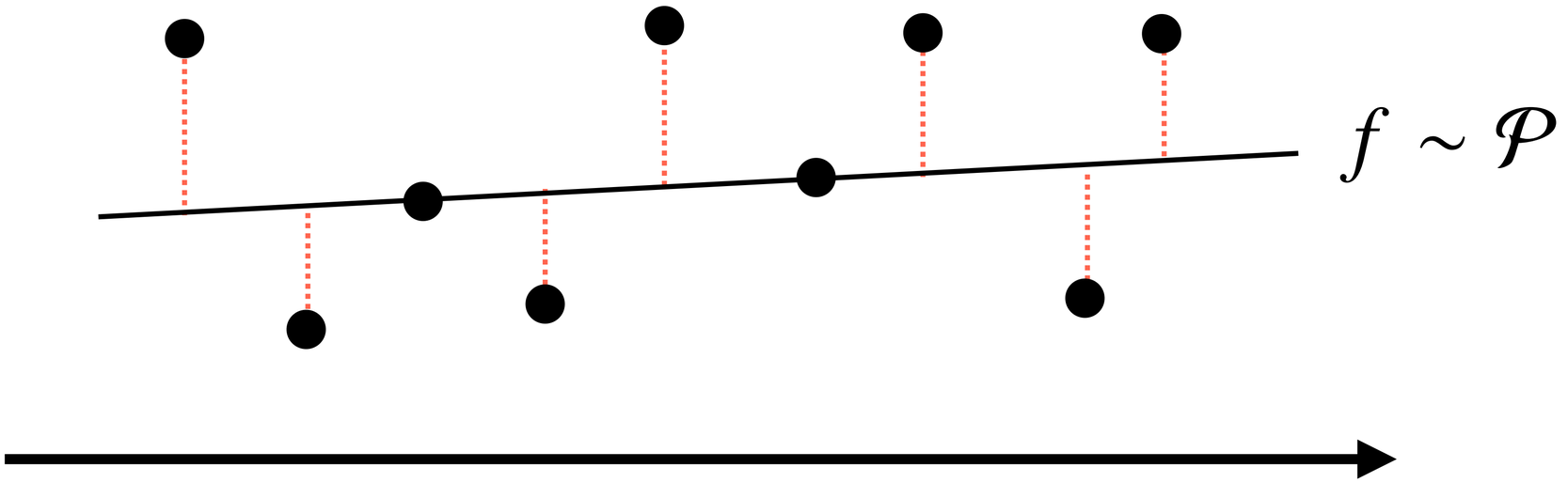
can you infer the shape of the function through this data?



Functional Uncertainty

- Distribution over the space of trajectories or functions

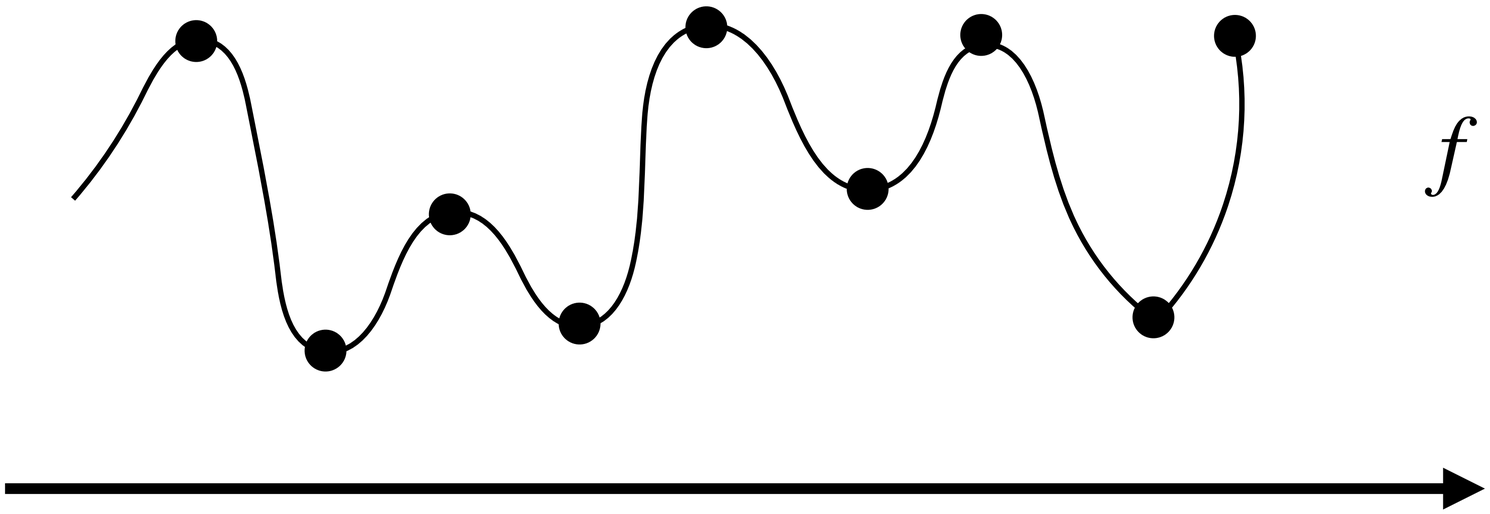
can you infer the shape of the function through this data?



Functional Uncertainty

- Distribution over the space of trajectories or functions

can you infer the shape of the function through this data?



$$f \sim \mathcal{P}'$$

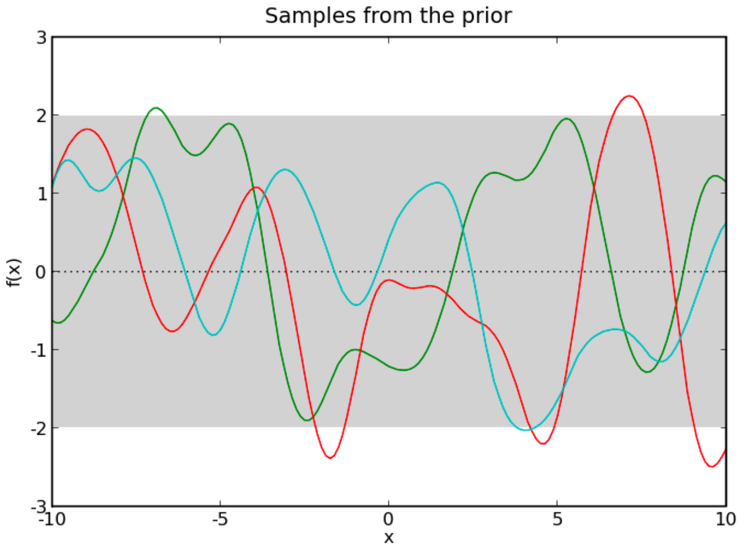
Gaussian Process

- We can assume that the function is drawn from a Gaussian Distribution
  - formally, $P_{x_{1:n}} = \mathcal{N}(\mu, \Sigma)$

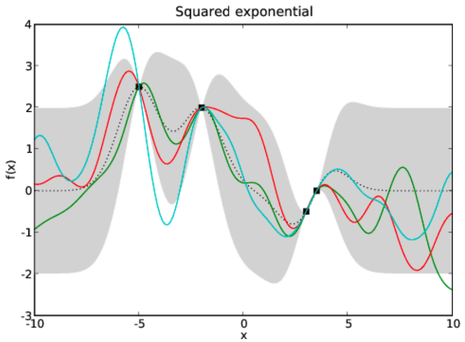
mean vector

covariance matrix

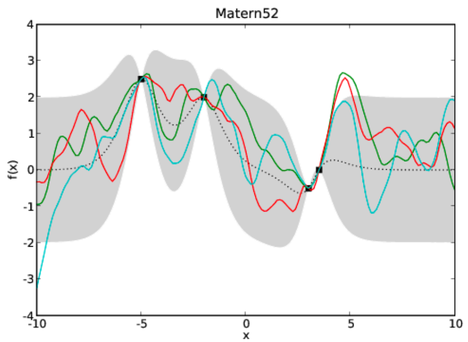
this is not data-driven!



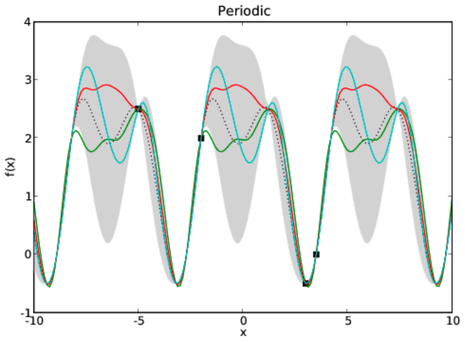
pythonhostd.org



squared exponential



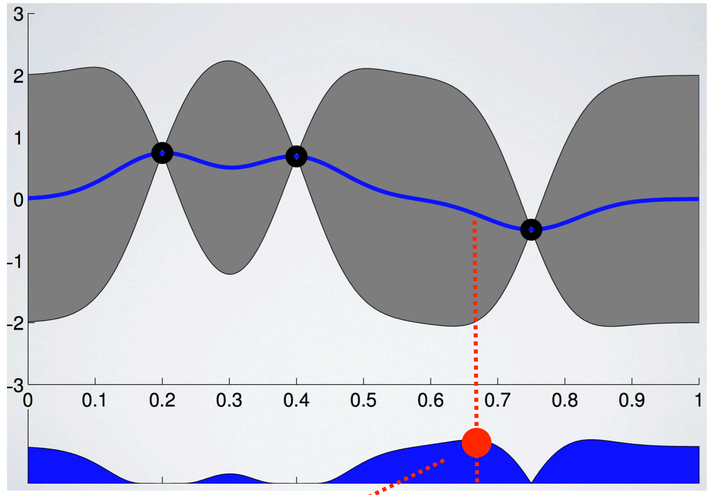
Matern



Periodic

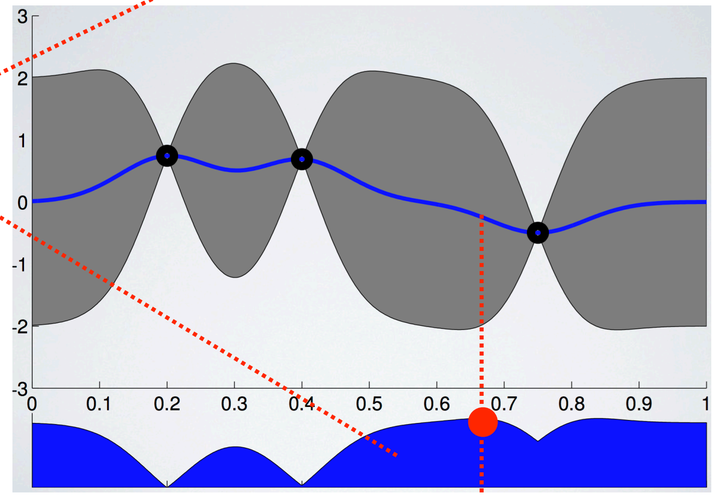
Acquisition Function

- Balance between exploration & exploitation
  - Probability Improvement
  - Expected Improvement
  - Upper Confidence Bound (UCB)
  - Entropy Search
  - Thompson Sampling



EI (expected improvement)

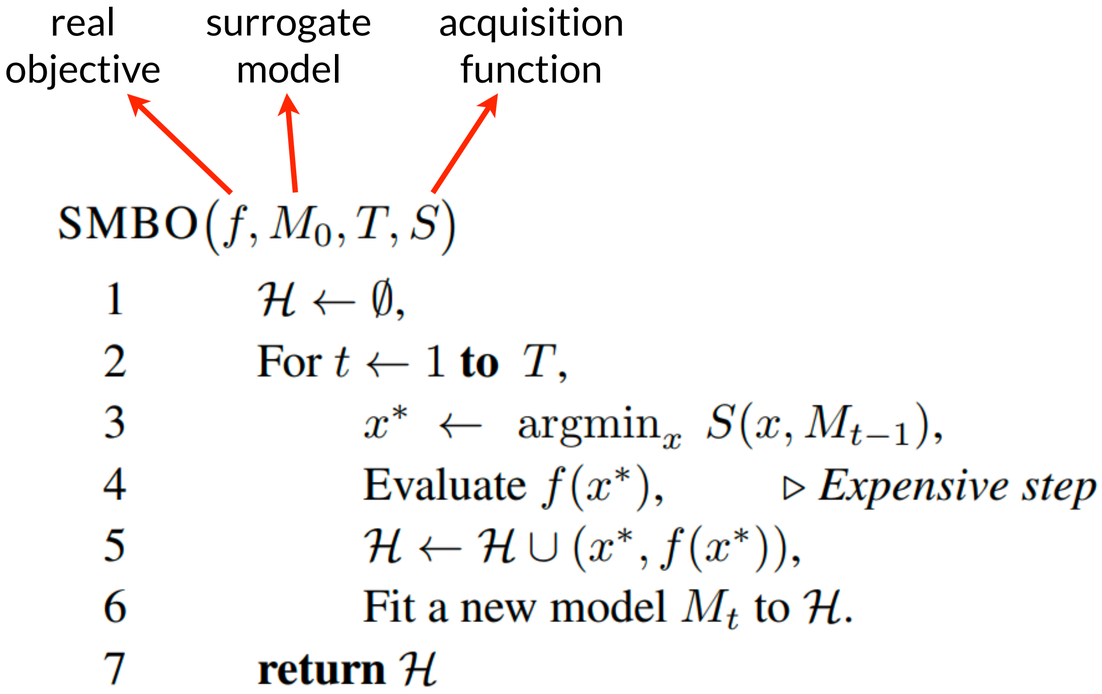
acquisition function



UCB (upper confidence bound)

Basic procedure of BO

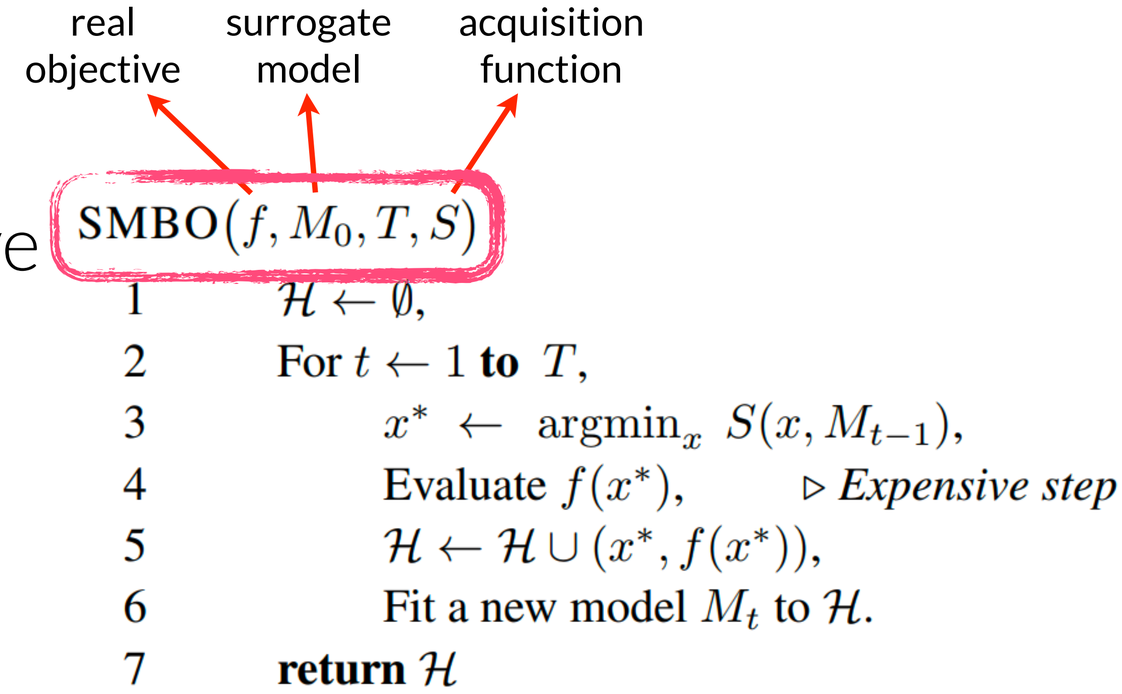
- **S**equential **M**odel **B**ased **O**ptimization



Sequential Model-Based Optimization for General Algorithm Configuration, Hutter et al., 2011

Basic procedure of BO

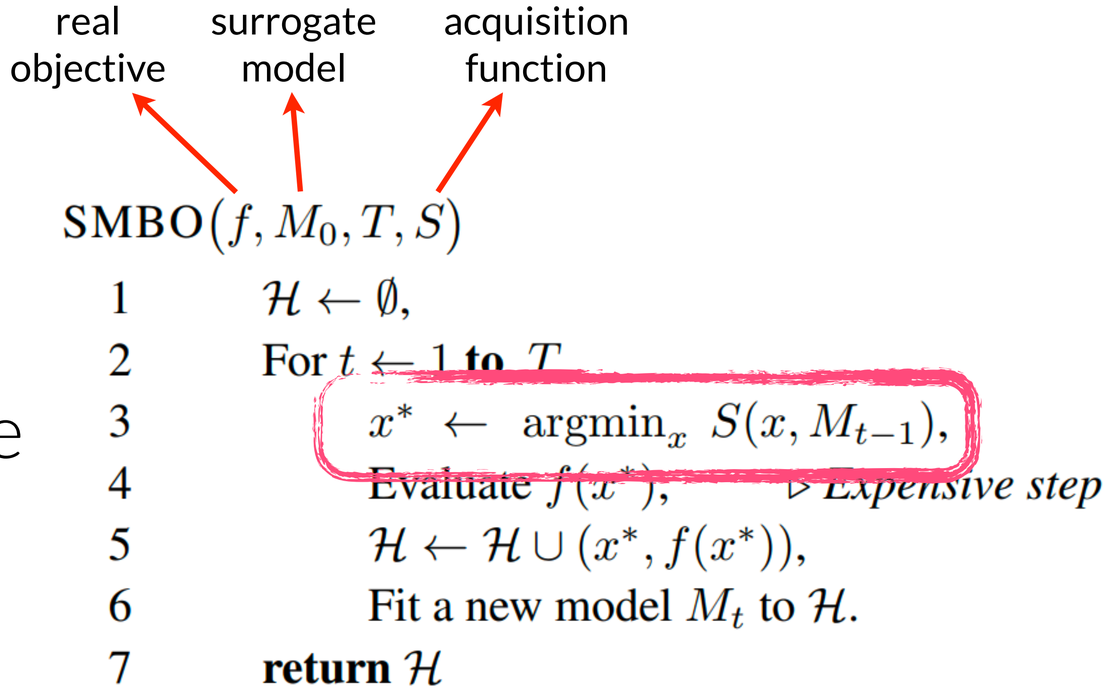
- Sequential Model Based Optimization
- Build a **surrogate model** of the objective functions → GP, Random Forest, TPE



Sequential Model-Based Optimization for General Algorithm Configuration, Hutter et al., 2011

Basic procedure of BO

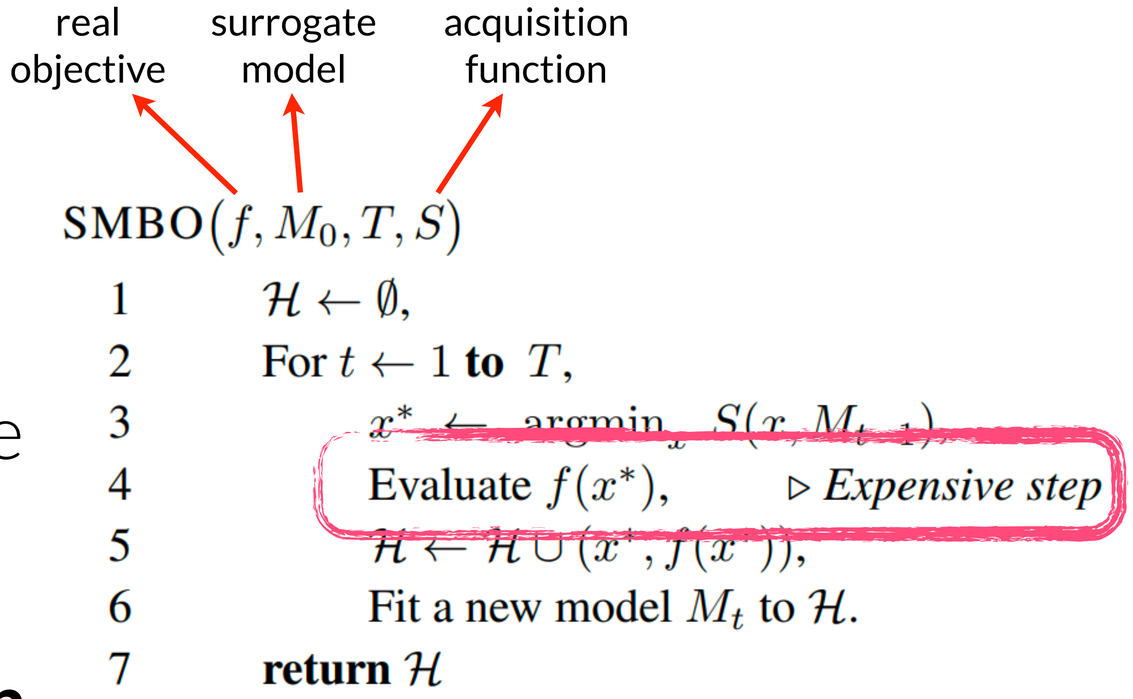
- Sequential Model Based Optimization
- Build a surrogate model of the objective functions → GP, Random Forest, TPE
- Detect the optimal configurations on the surrogate model → **optimization**



Sequential Model-Based Optimization for General Algorithm Configuration, Hutter et al., 2011

Basic procedure of BO

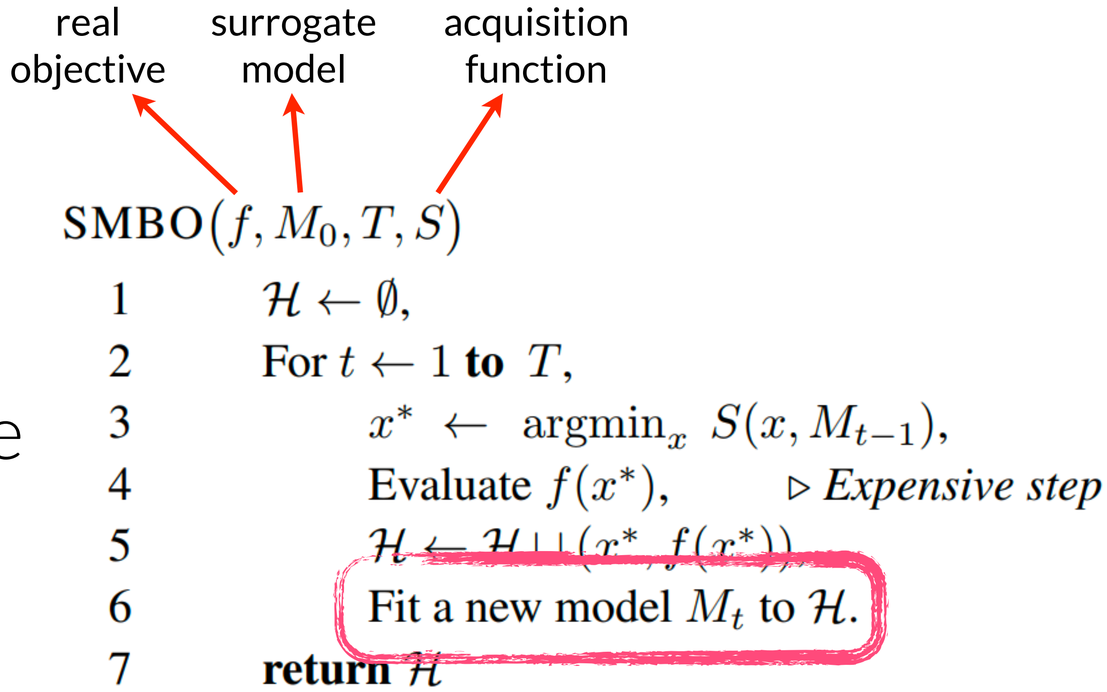
- Sequential Model Based Optimization
- Build a surrogate model of the objective functions → GP, Random Forest, TPE
- Detect the optimal configurations on the surrogate model → optimization
- Apply these hyperparameter values to the real objective function → **evaluation**



Sequential Model-Based Optimization for General Algorithm Configuration, Hutter et al., 2011

Basic procedure of BO

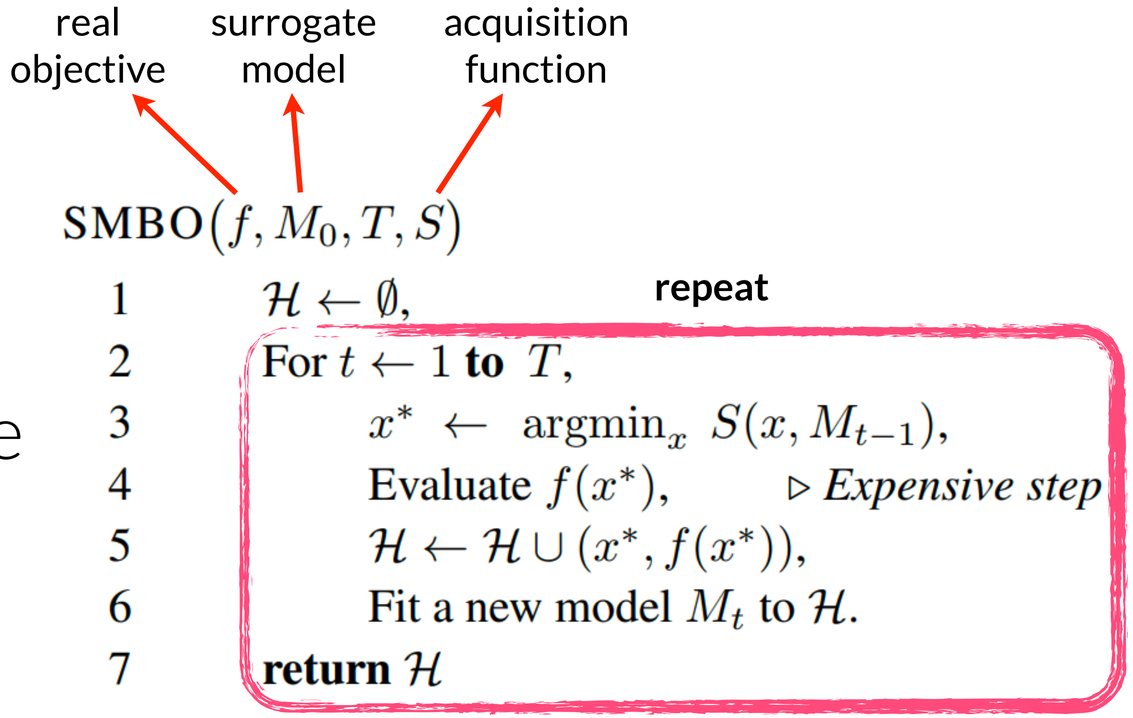
- Sequential Model Based Optimization
- Build a surrogate model of the objective functions → GP, Random Forest, TPE
- Detect the optimal configurations on the surrogate model → optimization
- Apply these hyperparameter values to the real objective function → evaluation
- Update the surrogate model with new results → **learning**



Sequential Model-Based Optimization for General Algorithm Configuration, Hutter et al., 2011

Basic procedure of BO

- Sequential Model Based Optimization
- Build a surrogate model of the objective functions → GP, Random Forest, TPE
- Detect the optimal configurations on the surrogate model → optimization
- Apply these hyperparameter values to the real objective function → evaluation
- Update the surrogate model with new results → learning



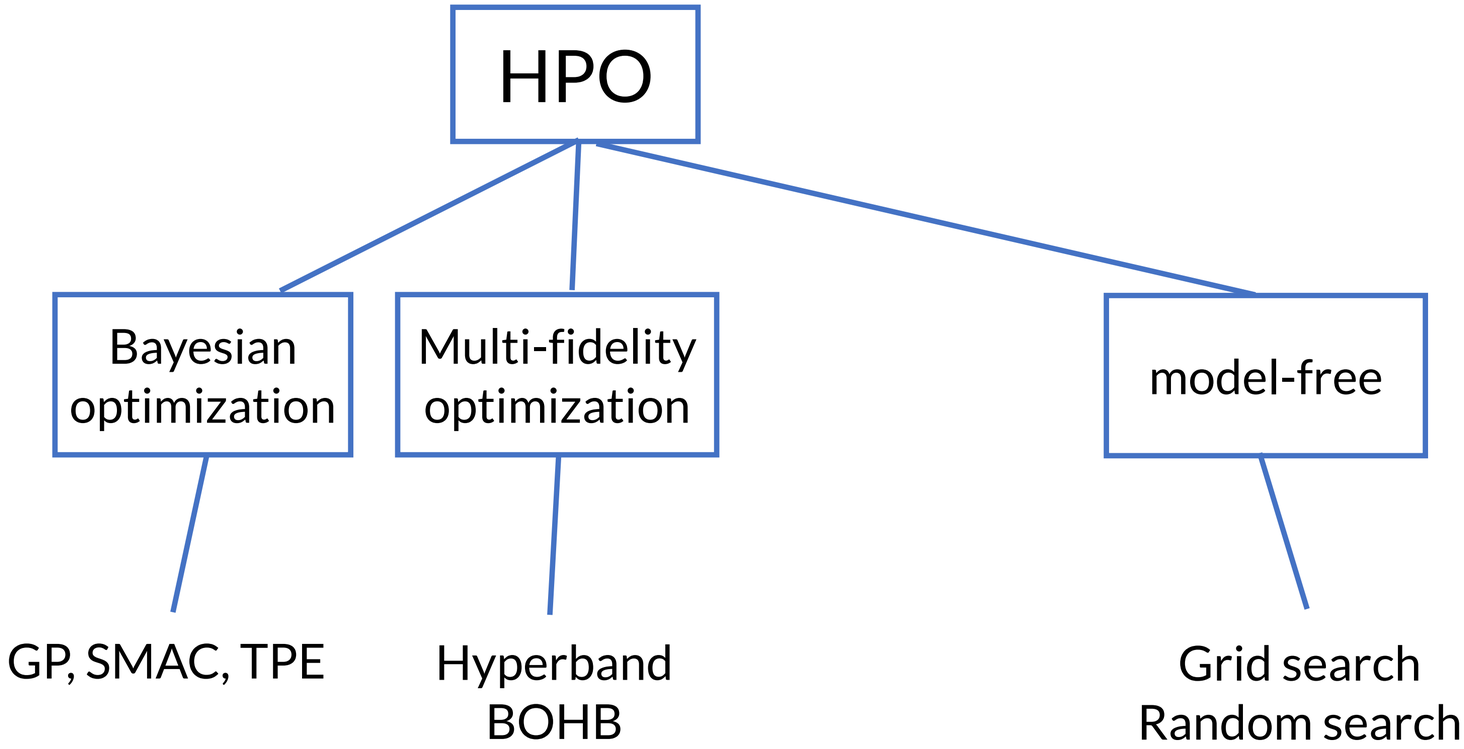
Sequential Model-Based Optimization for General Algorithm Configuration, Hutter et al., 2011

Pros and Cons of BO Algorithms

- **Pros**
  - more efficient than GS and RS
  - can detect near-optimal configuration combinations within a few iterations
  - require less computational resources
- **Cons**
  - since BO models are executed based on the preciously-tested values, they are difficult to parallelize
  - another configurations for BO procedures

HPO Algorithms Taxonomy

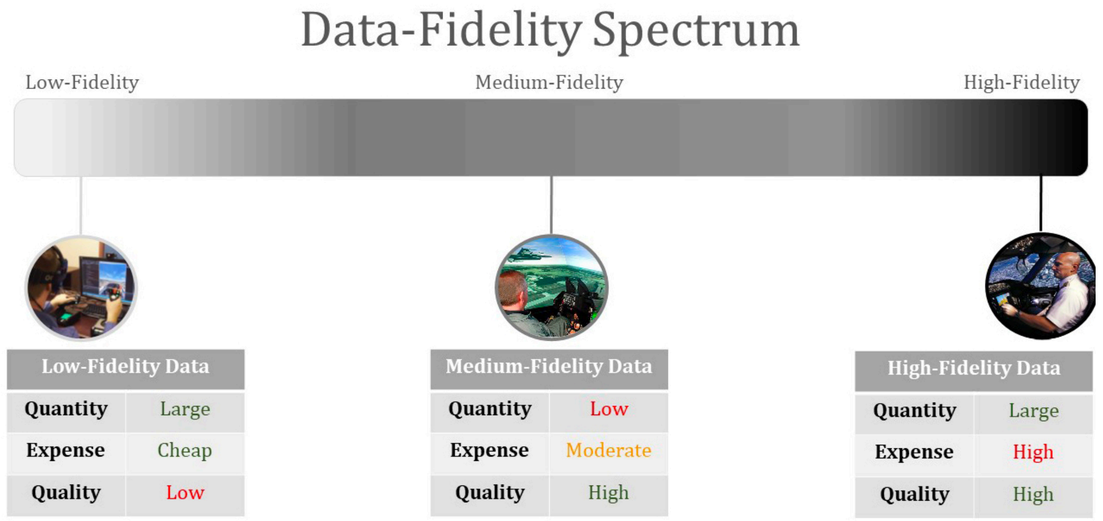
- Model-free methods
  - trial & error
  - grid & random search
- Gradient-based optimization
- Bayesian optimization
- Multi-fidelity optimization



On Hyperparameter Optimization of Machine Learning Algorithms: Theory & Practice, Yang & Shami, ***Neurocomputing*** 2020

Multi-Fidelity Optimization

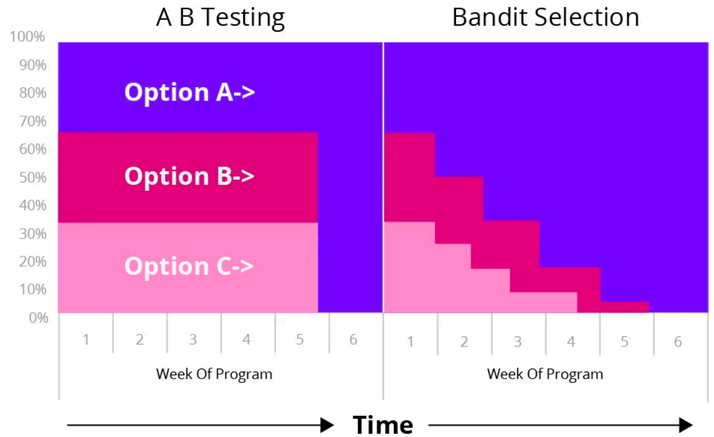
- ***Low-fidelity***
  - low cost, poor generalization
- ***High-fidelity***
  - higher cost, better generalization
- One can use a subset of the original dataset or a subset of the features → multi-fidelity
  - Poorly performing configurations are discarded after each round of hyperparameter evaluation on generated subsets
  - Well-performing configurations are evaluated on the entire train data



Schlicht (2017)

Successive Halving Algorithm

- Idea of SHA
  - uniformly allocate the budget to a set of arms for iterations
  - evaluate and throw out the half with the double budgets
- The main concern of SHA is how to allocate the budget $b = B/n$ and how to balance the trade-off between the evaluation and budget



Successive Halving Algorithm

input: Budget B, n arms where $\ell_{i,k}$ denotes the kth loss from the ith arm

Initialize: $S_0 = [n]$.

For $k = 0, 1, \ldots, \lceil \log_2(n) \rceil - 1$

Pull each arm in S_k for $r_k = \lfloor \frac{B}{|S_k| \lceil \log_2(n) \rceil} \rfloor$ additional times and set $R_k = \sum_{j=0}^{k} r_j$.

Let σ_k be a bijection on S_k such that $\ell_{\sigma_k(1),R_k} \leq \ell_{\sigma_k(2),R_k} \leq \cdots \leq \ell_{\sigma_k(|S_k|),R_k}$

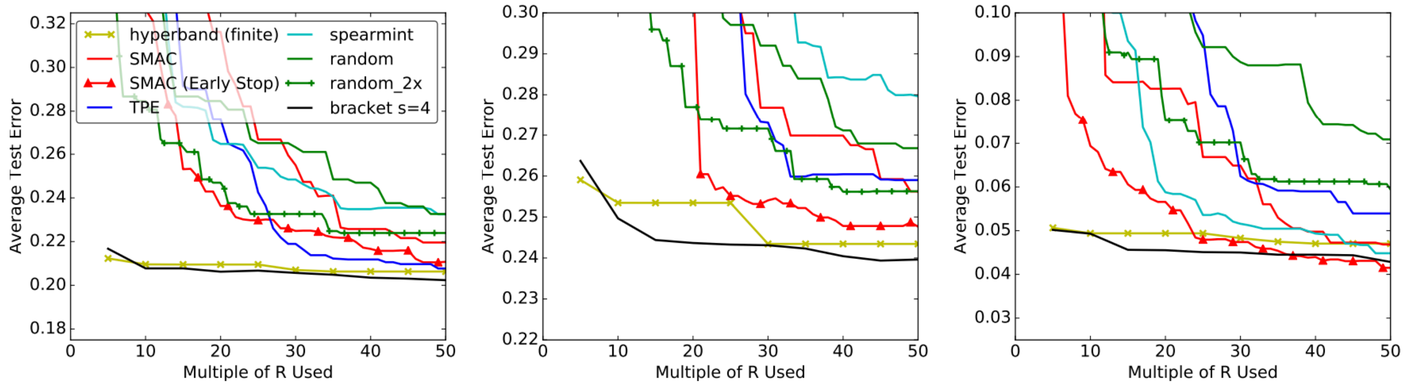
$S_{k+1} = \left\{ i \in S_k : \ell_{\sigma_k(i),R_k} \leq \ell_{\sigma_k(\lfloor |S_k|/2 \rfloor),R_k} \right\}$.

output : Singleton element of $S_{\lceil \log_2(n) \rceil}$

Non-stochastic Best Arm Identification and Hyperparameter Optimization, Jamieson & Talwalkar, ***AISTATS*** 2015

Hyperband

- Idea of Hyperband
  - adaptive resource allocation
  - early stopping
- Solve the dilemma of SHA by dynamically choosing a reasonable number of configurations
  - use random search to get n number of configurations



(a) CIFAR-10 (b) MRBI (c) SVHN

Successive Halving Algorithm

input: Budget B, n arms where $\ell_{i,k}$ denotes the kth loss from

Algorithm 1: HYPERBAND algorithm for hyperparameter optimization.

input : R, η (default $\eta = 3$)
initialization : $s_{\max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\max} + 1)R$
for $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ **do**
  $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$, $r = R\eta^{-s}$
  // begin SUCCESSIVEHALVING with (n, r) inner loop
  T =get_hyperparameter_configuration(n)
  for $i \in \{0, \ldots, s\}$ **do**
    $n_i = \lfloor n\eta^{-i} \rfloor$
    $r_i = r\eta^i$
    $L = \{\text{run\_then\_return\_val\_loss}(t, r_i) : t \in T\}$
    T =top_k($T, L, \lfloor n_i/\eta \rfloor$)
  end
end
return *Configuration with the smallest intermediate loss seen so far.*

Hyperband: A novel bandit-based approach to hyperparameter optimization, Li et al, ***JMLR*** 2018

BOHB: BO + Hyperband

- State-of-the art HPO technique that combines BO and Hyperband
- Hyperband uses a random search to get the configuration space
  - BOHB replace RS with BO!
- Easy to parallelizable
- Require the evaluations on subsets with small budgets

Algorithm 2: Pseudocode for sampling in BOHB

input : observations D, fraction of random runs ρ, percentile q, number of samples N_s, minimum number of points N_{min} to build a model, and bandwidth factor b_w

output : next configuration to evaluate

if *rand()* $< \rho$ **then return** random configuration

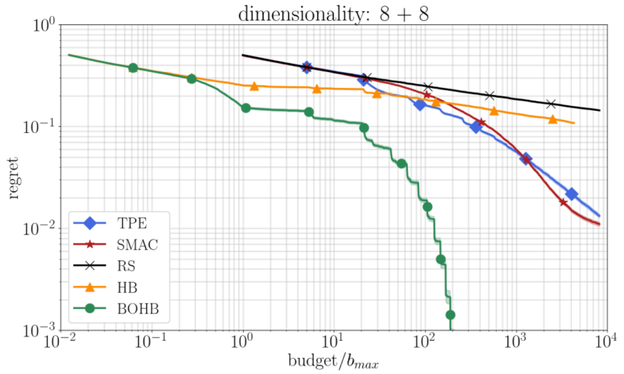
$b = \arg\max \{D_b : |D_b| \geq N_{min} + 2\}$

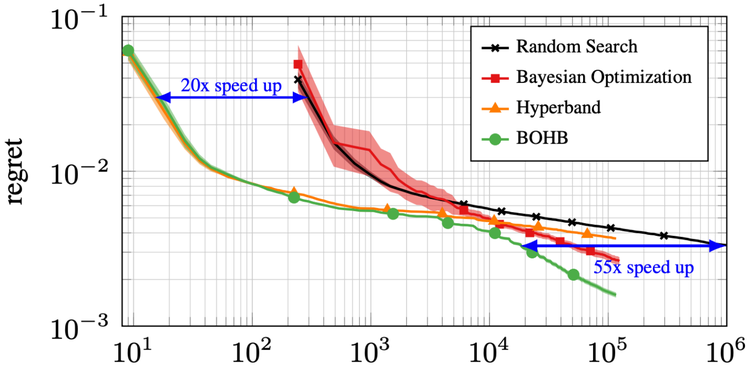
if $b = \emptyset$ **then return** random configuration

fit KDEs according to Eqs. (2) and (3)

draw N_s samples according to $l'(\boldsymbol{x})$ (see text)

return sample with highest ratio $l(\boldsymbol{x})/g(\boldsymbol{x})$





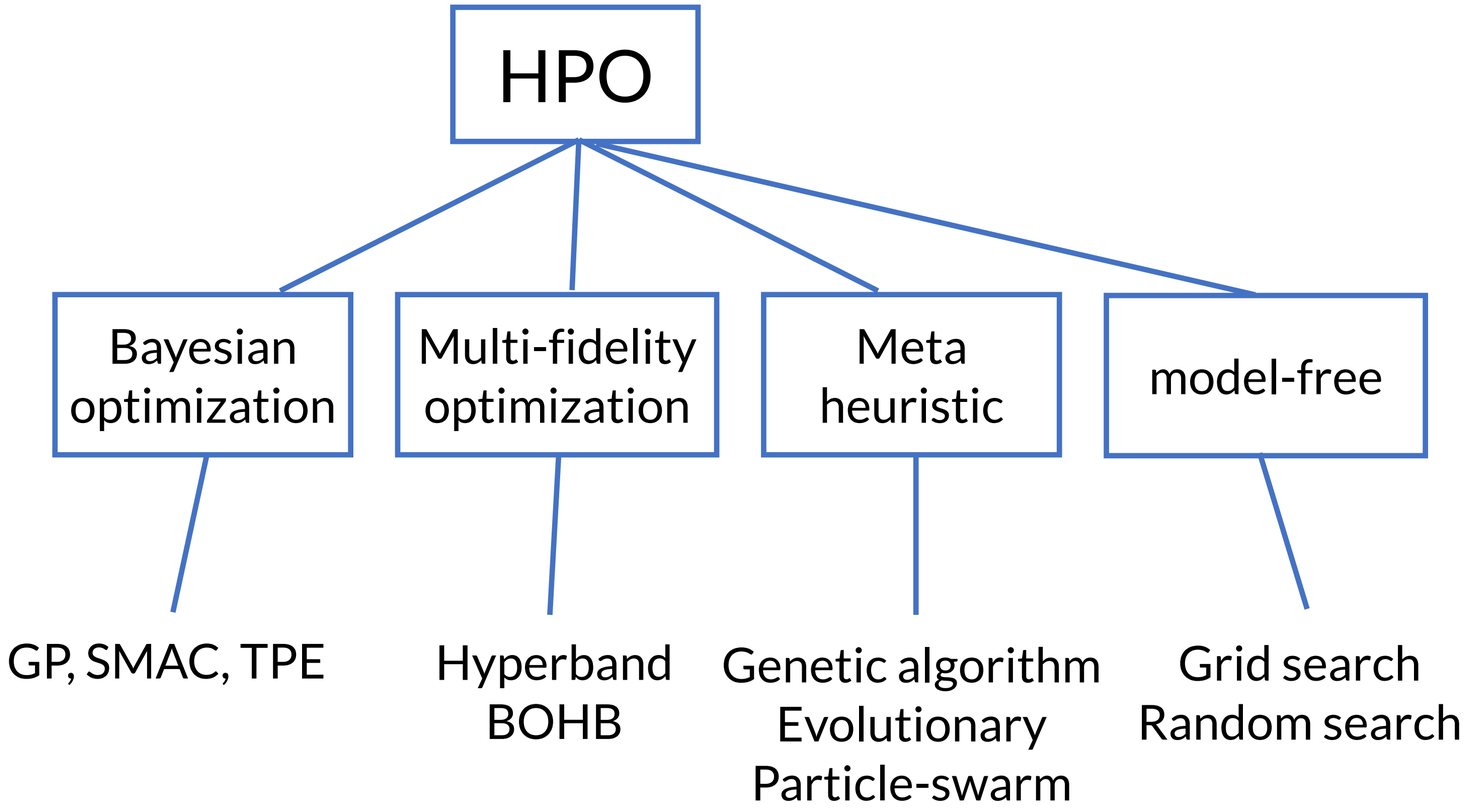
BOHB: Robust and Efficient Hyperparameter Optimization at Scale, Falkner et al, ***ICML*** 2018

Pros and Cons of Multi-fidelity optimization

- **Pros**
  - **parallelizable!**
  - fast convergence
- **Cons**
  - requires budget and resource allocation
  - can be poor at final performance

HPO Algorithms Taxonomy

- Model-free methods
  - trial & error
  - grid & random search
- Gradient-based optimization
- Bayesian optimization
- Multi-fidelity optimization
- Meta-heuristic algorithms



On Hyperparameter Optimization of Machine Learning Algorithms: Theory & Practice, Yang & Shami, ***Neurocomputing*** 2020

Population-Based Approach

- Population based approaches are a major type of meta-heuristic method
  - start by creating and updating a population as each generation
  - each individual in every generation is evaluated until the global optimum is identified
- The main difference between population-based approaches are the methods used to generate and select populations
  - genetic algorithms (GA)
  - particle swarm optimization (PSO)

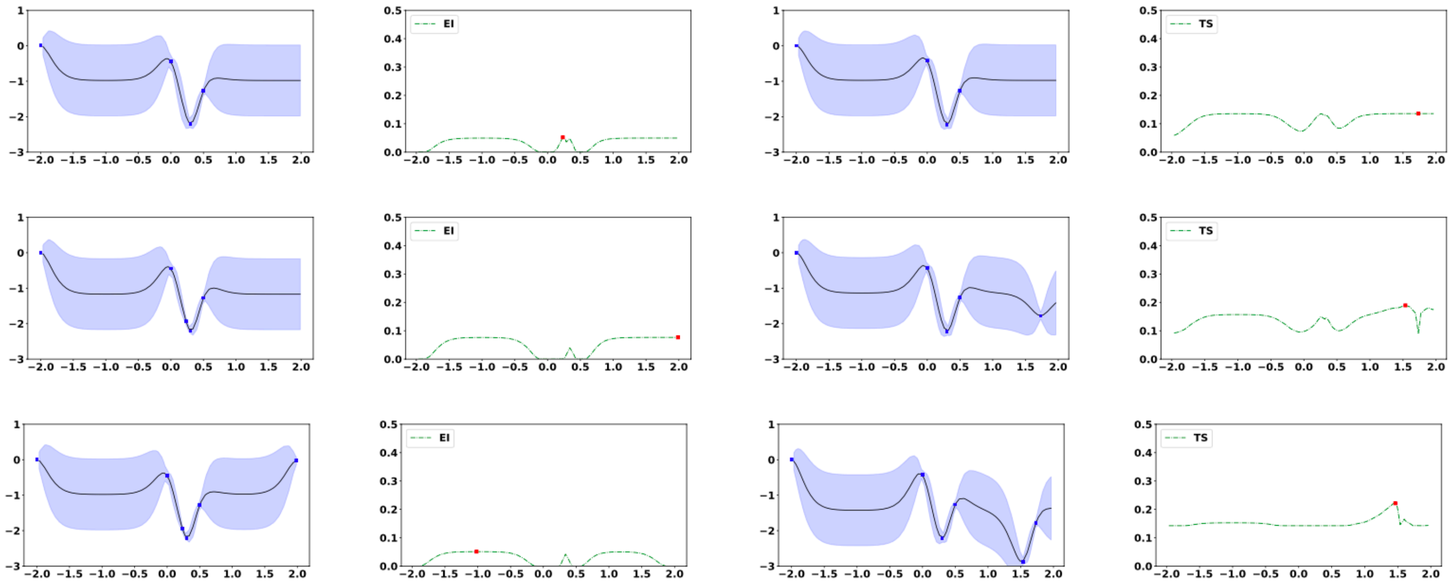
Comparison of HPO algorithms

		Pros	Cons	Time Complexity
model-free	Grid Search	Simple	Inefficient	$O(n^k)$
	Random Search	Easy to implement Parallelizable	Ignore the previous observations	$O(n)$
Bayesian optimization	Gaussian Process	Data efficient	Poor parallelization kernel selection	$O(n^3)$
	SMAC	Efficient with variable types	Poor parallelization Poor performance	$O(n \log n)$
	TPE	Efficient with variable types Efficient with conditional	Poor parallelization	$O(n \log n)$
Multi-fidelity optimization	Hyperband	Parallelizable	Inefficient with conditional Poor final performance	$O(n \log n)$
	BOHB	Efficient with variable types Parallelizable	Require the evaluations on subsets with small budgets	$O(n \log n)$
Meta-heuristic	PSO	Efficient with variable types Parallelizable	Require initialization	$O(n \log n)$

On Hyperparameter Optimization of Machine Learning Algorithms: Theory & Practice, Yang & Shami, ***Neurocomputing*** 2020

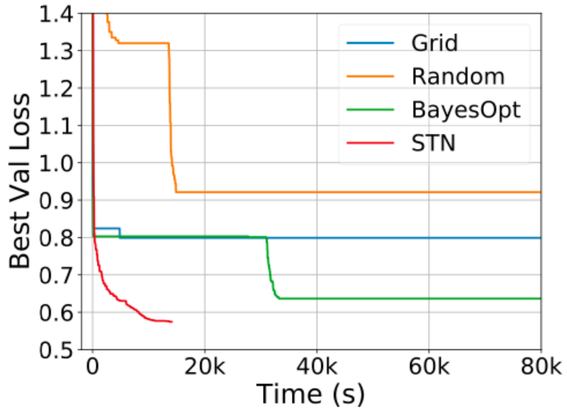
Advances: Two-step look-ahead BO

- Myopic exploration of expected improvement
  - estimate acquisition function via Monte Carlo method

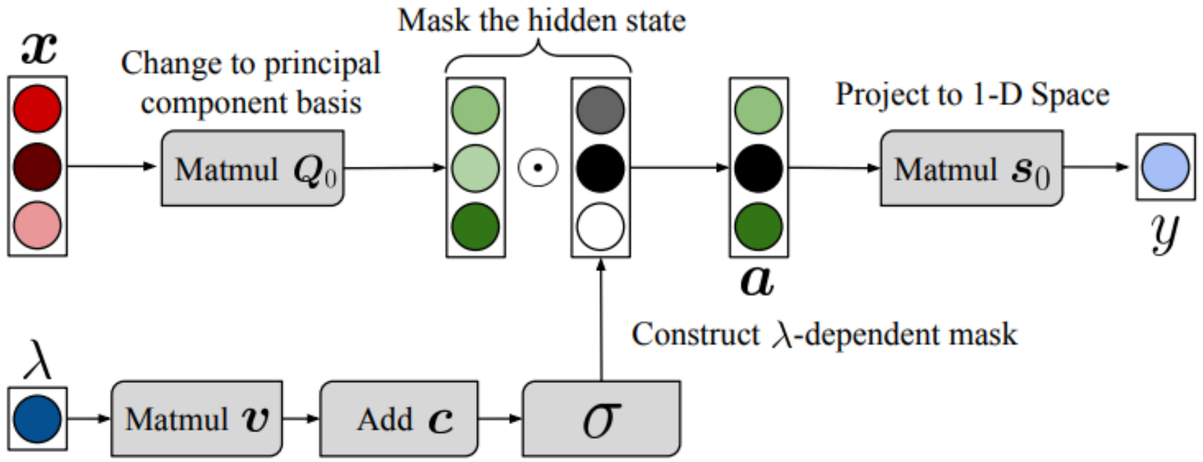


Practical Two-step Look-Ahead Bayesian Optimization, Wu & Frazier., ***NeurIPS*** 2019

Advances: Self-tuning networks



- Hyperparameter scheduling method
  - self-tuning network tunes hyperparameters in the training
  - transforms bilevel optimization into single-level optimization



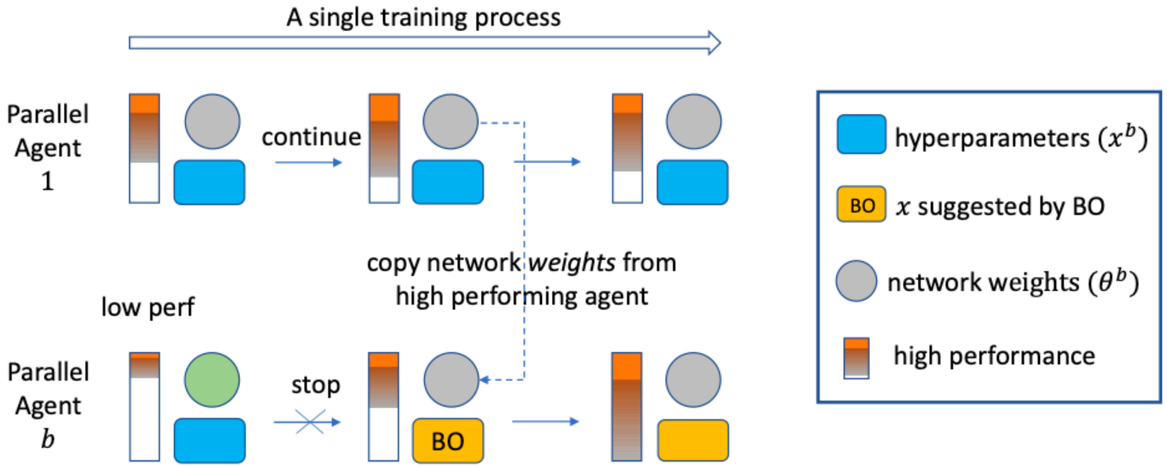
Algorithm 1 STN Training Algorithm

Initialize: Best-response approximation parameters $\boldsymbol{\phi}$, hyperparameters $\boldsymbol{\lambda}$, learning rates $\{\alpha_i\}_{i=1}^3$
while not converged **do**
  for $t = 1, \ldots, T_{train}$ **do**
    $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}|\boldsymbol{\sigma})$
    $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \alpha_1 \frac{\partial}{\partial \boldsymbol{\phi}} f(\boldsymbol{\lambda} + \boldsymbol{\epsilon}, \hat{\mathbf{w}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda} + \boldsymbol{\epsilon}))$
  for $t = 1, \ldots, T_{valid}$ **do**
    $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}|\boldsymbol{\sigma})$
    $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \alpha_2 \frac{\partial}{\partial \boldsymbol{\lambda}} \left(F(\boldsymbol{\lambda} + \boldsymbol{\epsilon}, \hat{\mathbf{w}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda} + \boldsymbol{\epsilon})) - \tau \mathbb{H}[p(\boldsymbol{\epsilon}|\boldsymbol{\sigma})]\right)$
    $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} - \alpha_3 \frac{\partial}{\partial \boldsymbol{\sigma}} \left(F(\boldsymbol{\lambda} + \boldsymbol{\epsilon}, \hat{\mathbf{w}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda} + \boldsymbol{\epsilon})) - \tau \mathbb{H}[p(\boldsymbol{\epsilon}|\boldsymbol{\sigma})]\right)$

Bilevel Optimization of Hyperparameters using Structured Best-Response Functions, MacKay et al., ***ICLR*** 2019

Advances: Population based Bandit

- PBT explores configuration space randomly
  - PB2 explores the space based on the probabilistic model (GP-UCB)
  - proves the theoretical guarantees



Algorithm 1: Population-Based Bandit Optimization (PB2)

Initialize: Network weights $\{\theta_0^b\}_{b=1}^B$, hyperparameters $\{x_0^b\}_{b=1}^B$, dataset $D_0 = \emptyset$
(in parallel) for $t = 1, \ldots, T-1$ **do**

1. **Update Models:** $\theta_t^b \leftarrow \text{step}(\theta_{t-1}^b | x_{t-1}^b)$
2. **Evaluate Models:** $y_t^b = F_t(x_t^b) - F_{t-1}(x_{t-1}^b) + \epsilon_t$ for all b
3. **Record Data:** $D_t = D_{t-1} \cup \{(y_t^b, t, x_t^b)\}_{b=1}^B$
4. If $t \mod t_{\text{ready}} = 0$:
   - **Copy weights:** Rank agents, if θ^b is in the bottom $\lambda\%$ then copy weights θ^j from the top $\lambda\%$.
   - **Select hyperparameters:** Fit a GP model to D_t and select hyper-parameters $x_t^b, \forall b \leq B$ by maximizing Eq. (5).

Return the best trained model θ

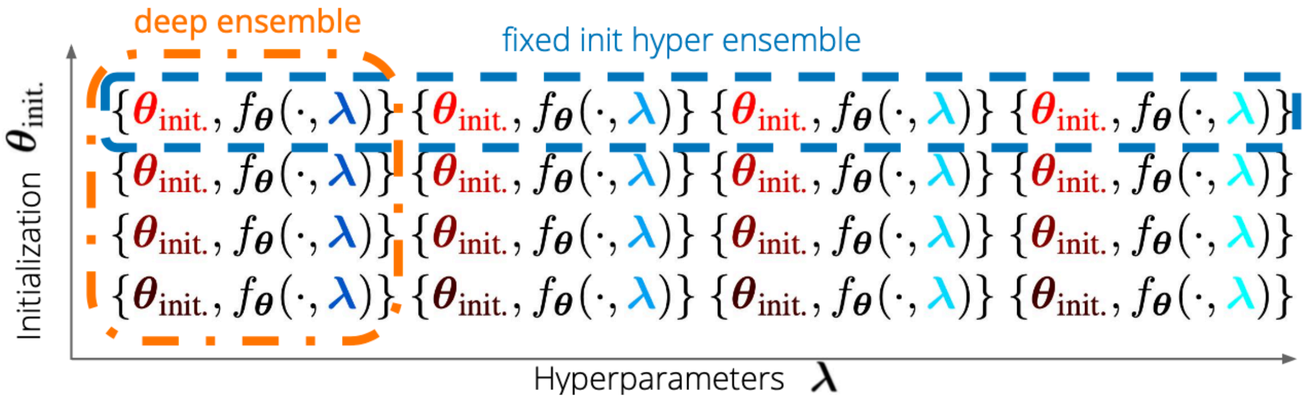
Provably efficient hyperparameter optimization with population-based bandits, Parker-Holder et al., ***NeurIPS*** 2020

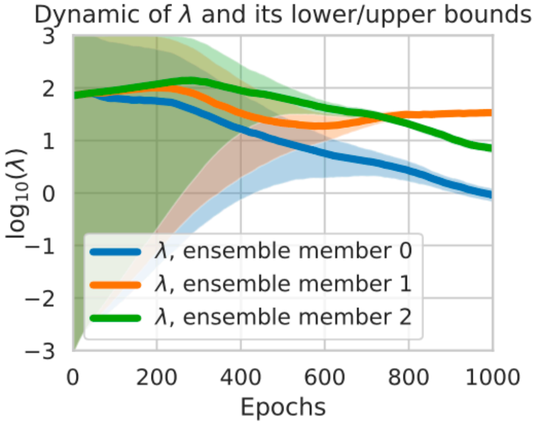
Advances: Hyper Deep Ensemble

- Hyper Deep Ensemble
  - Deep Ensemble
  - Hyper Ensemble
- BatchEnsemble → reduces memory cost

Algorithm 1: `hyper_deep_ens`(K, κ)

$\mathcal{M}_0 = \{f_{\boldsymbol{\theta}_j}(\cdot, \boldsymbol{\lambda}_j)\}_{j=1}^{\kappa} \leftarrow$ `rand_search`(κ);
$\mathcal{E}_0 \leftarrow$ `hyper_ens`$(\mathcal{M}_0,\ K)$ and $\mathcal{E}_{\text{strat.}} = \{\ \}$;
foreach $f_{\boldsymbol{\theta}}(\cdot, \boldsymbol{\lambda}) \in \mathcal{E}_0$.unique() **do**
  foreach $k \in \{1, \ldots, K\}$ **do**
    $\boldsymbol{\theta}' \leftarrow$ random initialization;
    $f_{\boldsymbol{\theta}_k}(\cdot, \boldsymbol{\lambda}) \leftarrow$ train $f_{\boldsymbol{\theta}'}(\cdot, \boldsymbol{\lambda})$;
    $\mathcal{E}_{\text{strat.}} = \mathcal{E}_{\text{strat.}} \cup \{\ f_{\boldsymbol{\theta}_k}(\cdot, \boldsymbol{\lambda})\}$;
  end
end
return `hyper_ens`$(\mathcal{E}_{\text{strat.}},\ K)$;

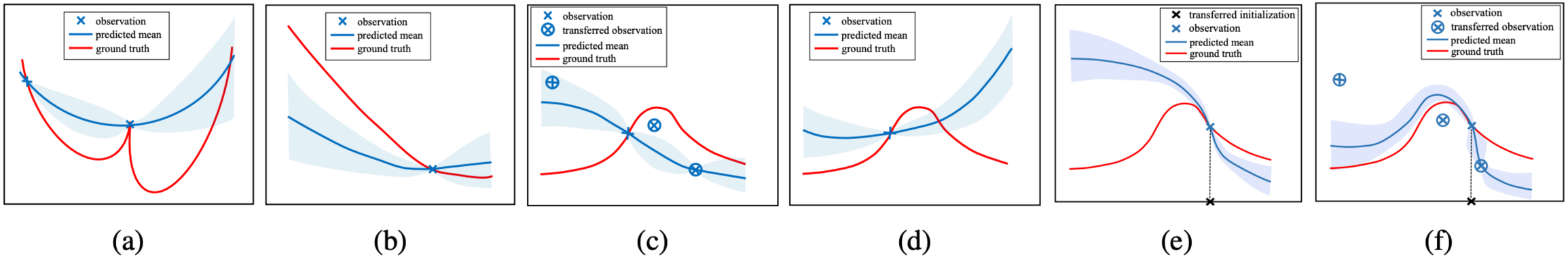




Hyperparameter Ensembles for Robustness and Uncertainty Quantification, Wenzel et al., ***NeurIPS*** 2020

Advances: Transferable HPO

- Transferable Neural Process (TNP)
  - end-to-end surrogate model that learns a comprehensive set of meta-knowledge
- Follows the strategy of MAML



Meta-learning Hyperparameter Performance Prediction with Neural Processes, Wei et al., ***ICML*** 2021

Neural Bootstrapping Attention for Neural Processes

Minsub Lee[1]*, Junhyun Park[1]*, Sojin Jang[1], Chanhui Lee[1],
Hyungjoo Cho[2], Minsuk Shin[3], Sungbin Lim[1]†
Artificial Intelligence Graduate School, UNIST[1]
Department of Transdisciplinary Studies, Seoul National University[2]
Department of Statistics, University of South Carolina[3]
{minsublee, junhyun, sungbin}@unist.ac.kr


Lee, Minsub


Park, Junhyun


Jang, Sojin


Lee, Chanhui

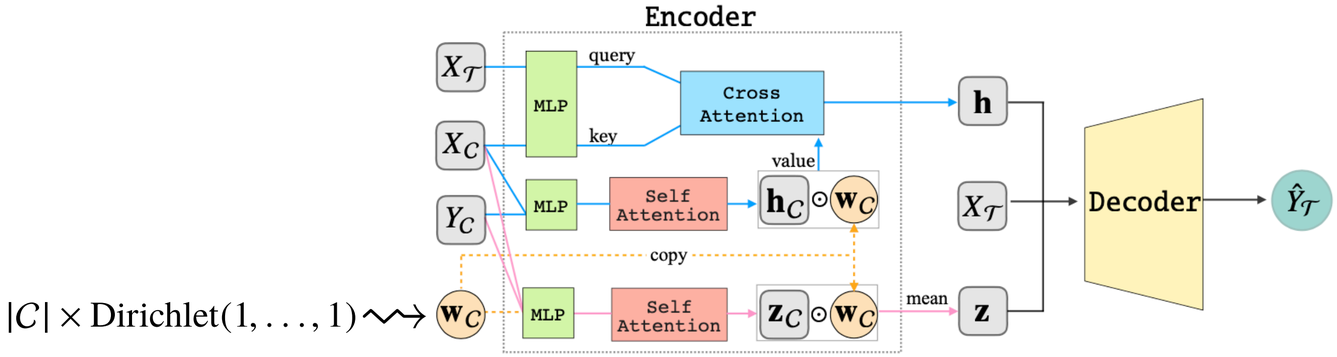

Cho, Hyungjoo


Shin, Minsuk


LIM

Neural Process based Bayesian Optimization

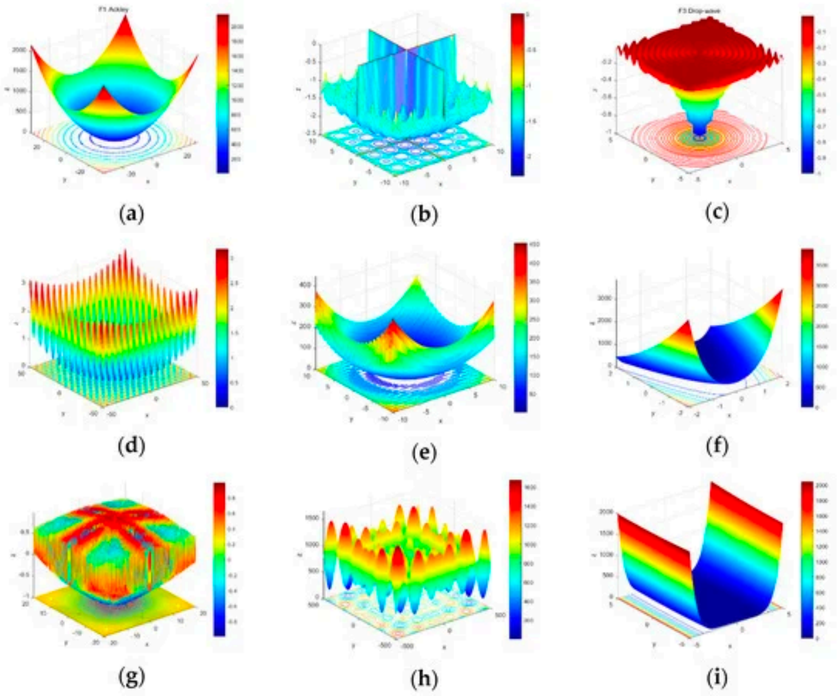
- Neural Bootstrap + Neural Process
  - ☑ 뉴럴 프로세스의 메타학습 과정에서 보지 못한 함수도 최적화 가능한가?
  - ☑ 온라인 학습이 가능한가?
  - ☑ 고차원에서도 학습 가능한가?
  - ☑ 데이터 효율적인가?



Algorithm 1: Neural Process based Bayesian Optimization.

Input : Target function $f^\star$; Acquisition function $\mathcal{U}$; Observed data $\mathcal{D}_0 = \{(x_0, f^\star(x_0))\}$; Maximum evaluation step T.

1 Meta-train a neural process p_θ on $f \sim \mathcal{P}(\mathcal{F})$ by (2.1).
2 **for** $t = 1, \ldots, T$ **do**
3 Find x_t by optimizing acquisition function: $x_t = \underset{x \in \mathcal{X}}{\text{argmin}}\, \mathcal{U}\left(p_\theta(y|x, \mathcal{D}_{t-1})\right)$
4 Evaluate $f^\star(x_t)$ and update the observed data: $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{(x_t, f^\star(x_t))\}$



Neural Bootstrap

Neural Bootstrapper

Minsuk Shin[1]*, **Hyungjoo Cho**[2,3]*, **Hyun-seok Min**[2], **Sungbin Lim**[4†]
Department of Statistics, University of South Carolina[1]
Tomocube Inc.[2]
Department of Transdisciplinary Studies, Seoul National University[3]
Artificial Intelligence Graduate School, UNIST[4]
sungbin@unist.ac.kr

Shin, Minsuk

Cho, Hyungjoo

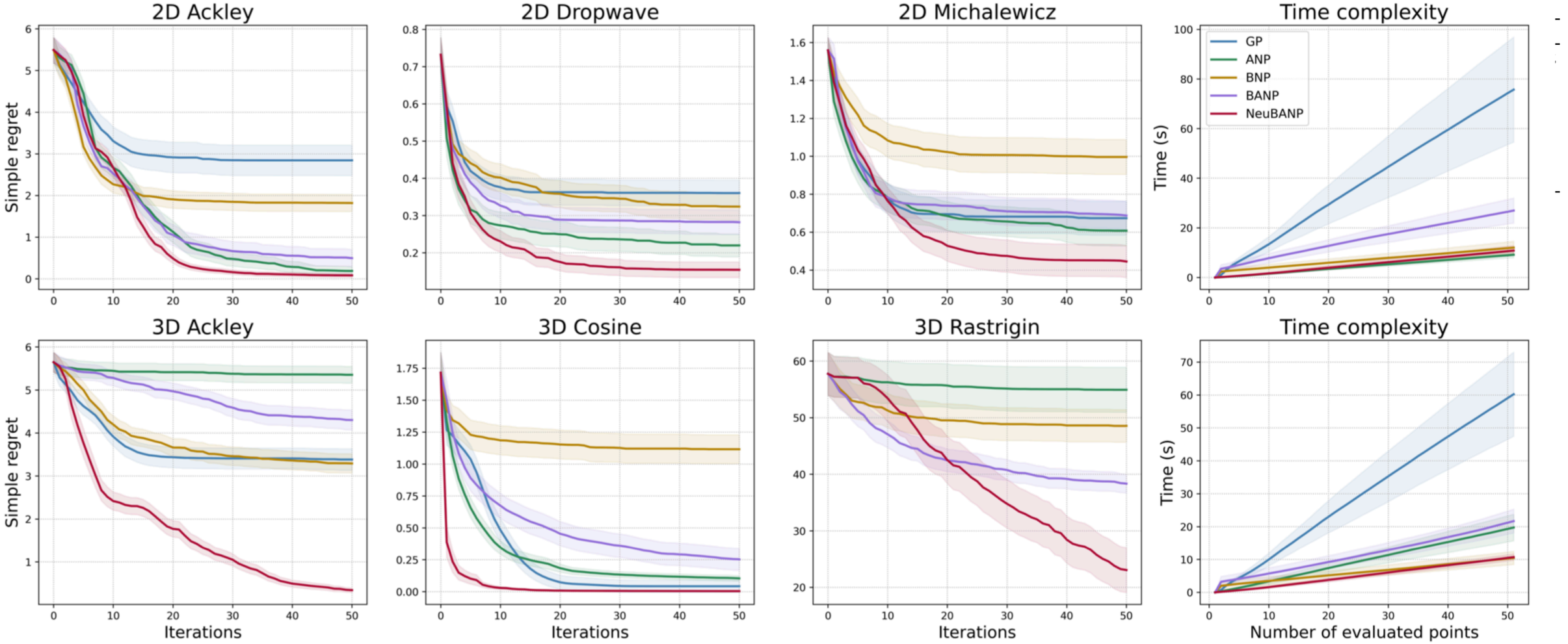
Min, Hyun-seok

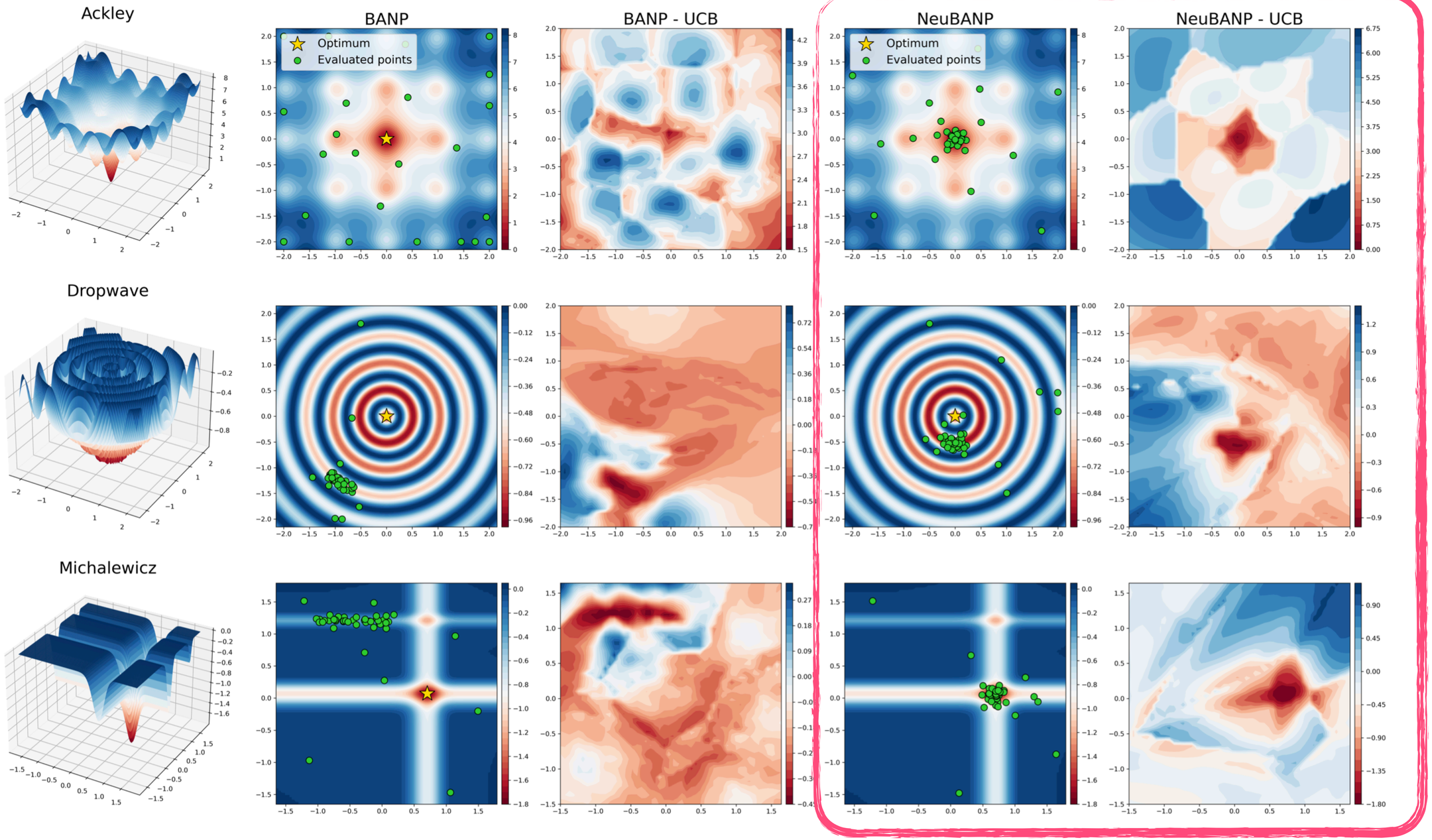
LIM

Neural Bootstrapper, ***NeurIPS*** (2021)

Joint work with M. Shin (Univ. of South Carolina), H. Cho (SNU), H. Min (Tomocube), **LIM**

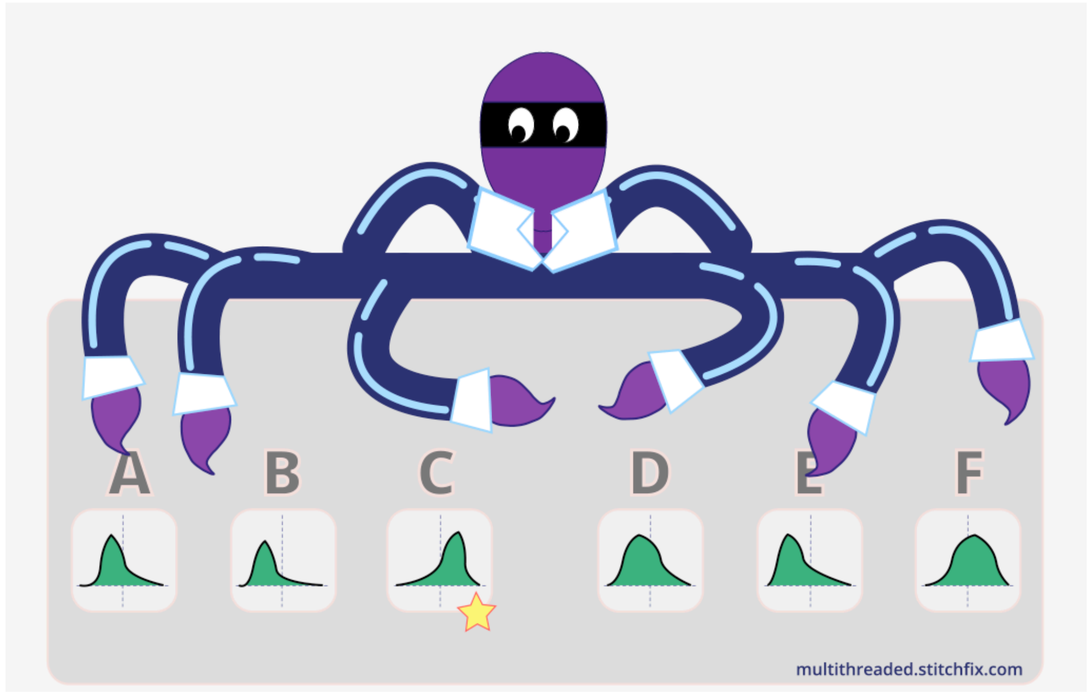
NeuBoots for Bayesian Optimization

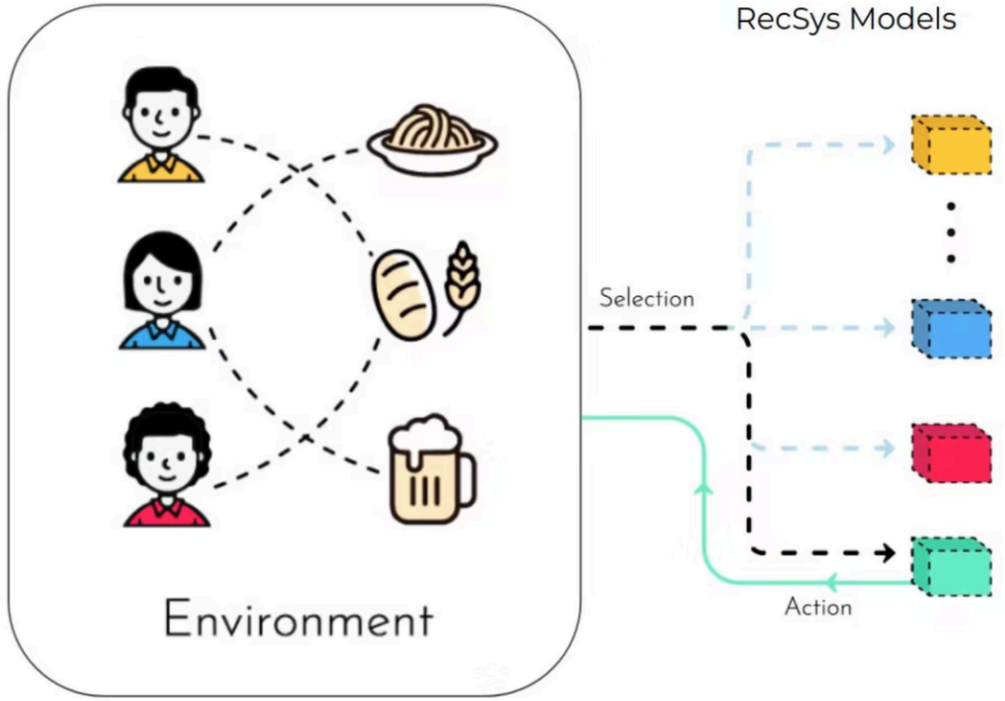



Ackley
Dropwave
Michalewicz
BANP
BANP - UCB
NeuBANP
NeuBANP - UCB
Optimum
Evaluated points

Contextual Multi-Armed Bandits

- How to find the best choice in a data-efficient way?





Neural Bootstrapping Attention Neural Processes, ***submitted*** (2021)

Joint work with M. Lee (UNIST), J. Park (UNIST), S. Jang (UNIST), C. Lee (UNIST), H. Cho (SNU), M. Shin (Univ. of South Carolina)

Contextual Multi-Armed Bandits

Difficult Problem

Regret	Method	$\delta = 0.5$	$\delta = 0.7$	$\delta = 0.9$	$\delta = 0.95$	$\delta = 0.99$
Cumulative	Uniform	100.00 ± 0.08	100.00 ± 0.09	100.00 ± 0.25	100.00 ± 0.37	100.00 ± 0.78
	Neural Linear	0.95 ± 0.02	1.60 ± 0.03	4.65 ± 0.18	9.56 ± 0.36	49.63 ± 2.41
	MAML	2.95 ± 0.12	3.11 ± 0.16	4.84 ± 0.22	7.01 ± 0.33	22.93 ± 1.57
	NP	1.60 ± 0.06	1.75 ± 0.05	3.31 ± 0.10	5.71 ± 0.24	22.13 ± 1.23
	NeuBANP	**0.85** ± 0.22	**1.02** ± 0.27	**1.85** ± 0.56	**3.04** ± 0.88	**9.76** ± 1.93
Simple	Uniform	100.00 ± 0.45	100.00 ± 0.78	100.00 ± 1.18	100.00 ± 2.21	100.00 ± 4.21
	Neural Linear	**0.33** ± 0.04	**0.79** ± 0.07	2.17 ± 0.14	4.08 ± 0.20	35.89 ± 2.98
	MAML	2.49 ± 0.12	3.00 ± 0.35	4.75 ± 0.48	7.10 ± 0.77	22.89 ± 1.41
	NP	1.04 ± 0.06	1.26 ± 0.21	2.90 ± 0.35	5.45 ± 0.47	21.45 ± 1.3
	NeuBANP	0.86 ± 0.06	1.04 ± 0.08	**1.88** ± 0.14	**3.09** ± 0.23	**9.96** ± 0.70

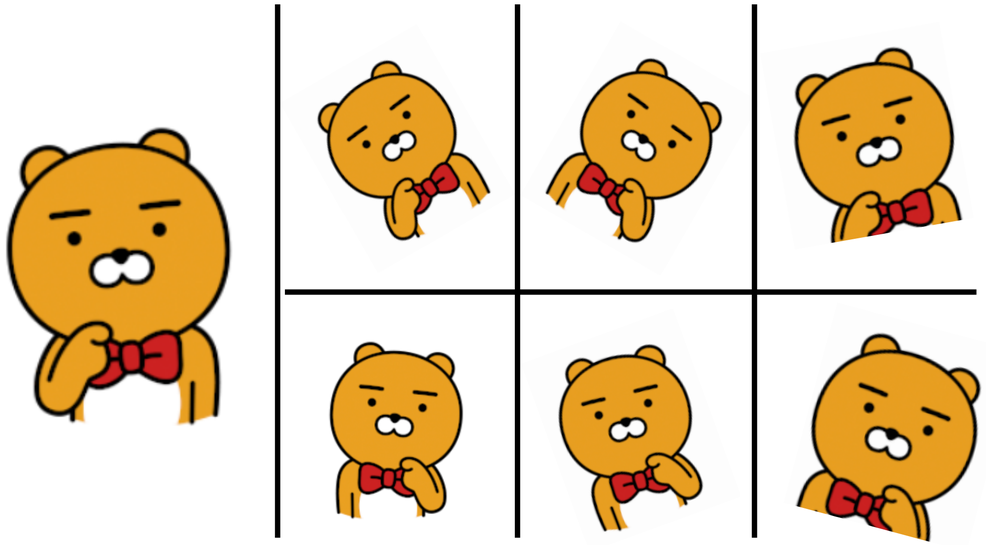
Neural Bootstrapping Attention Neural Processes, ***submitted*** (2021)

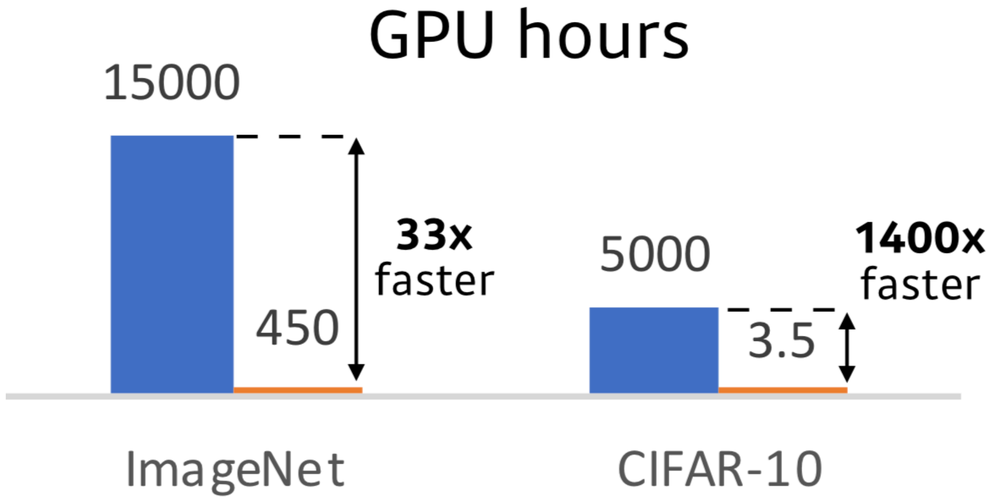
Joint work with M. Lee (UNIST), J. Park (UNIST), S. Jang (UNIST), C. Lee (UNIST), H. Cho (SNU), M. Shin (Univ. of South Carolina)

Applications
Hyperparameter Optimization

Automated Augmentation

- Fast AutoAugment
  - find augmentation policies without human intervention



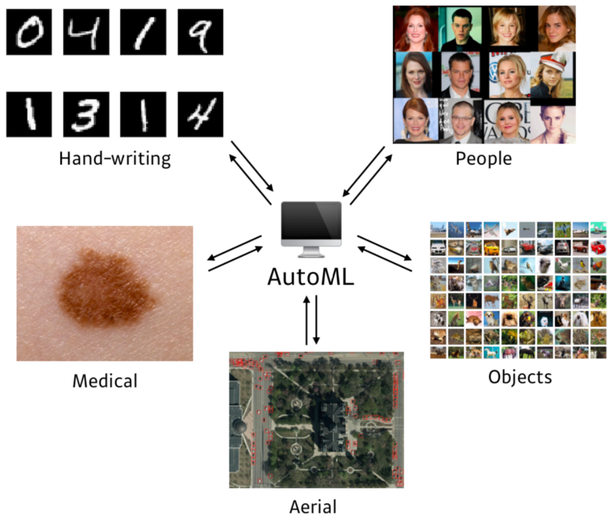


Fast AutoAugment, ***NeurIPS*** (2019)

Joint work with I. Kim (Kakao Brain), T. Kim (MILA), C. Kim (Kakao Brain), S. Kim (Kakao Brain)

Fast Training with Time Constraint

- Computationally light training with AutoAugment
  - 20 mins with single GPU



Username	Rank	Major DL Framework	Strategy	Prize
kakaobrain	1st	PyTorch	Fast AutoAugment	$2000
tanglang	2nd	PyTorch	Inspired by Fast AutoAugment	$1500
kvr	3rd	PyTorch	Inspired by Fast AutoAugment	$500

NeurIPS AutoDL challenges

Enter AutoSpeech (deadline Oct 15)

Enter AutoWeakly (deadline October 29)

Congratulations to the **ECML PKDD conf**. AutoCV2 winners:

- **First place:** kakaobrain [**GitHub repo**]
- **Second place:** tanglang [**GitHub repo**]
- **Third place:** kvr [**GitHub repo**]

They will be invited to present at the **discovery challenge workshops of ECML PKDD**.

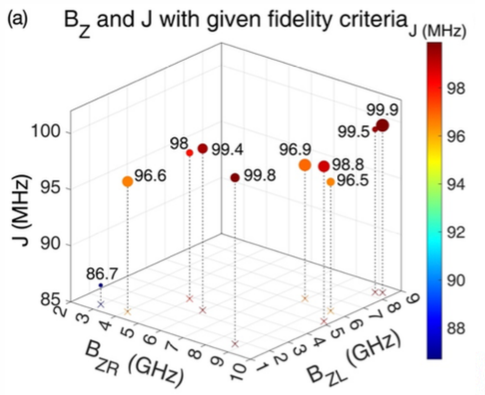
The **IJCNN conf.** AutoCV winners [**slides**] were:

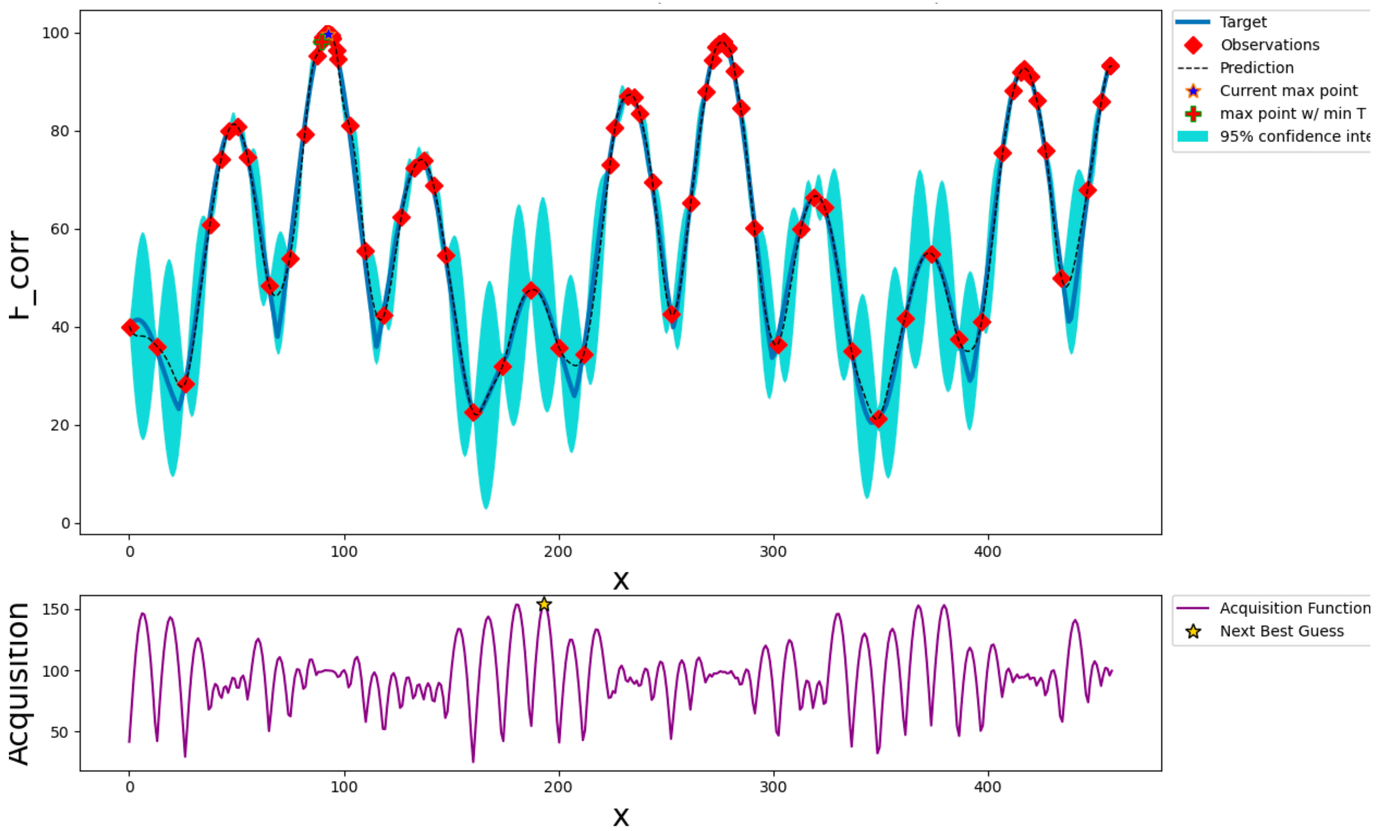
- **First place:** KakaoBrain [**GitHub repo**]
- **Second place:** DKKimHCLee [**GitHub repo**][**slides**]
- **Third place:** base_1 [**GitHub repo**][**slides**]

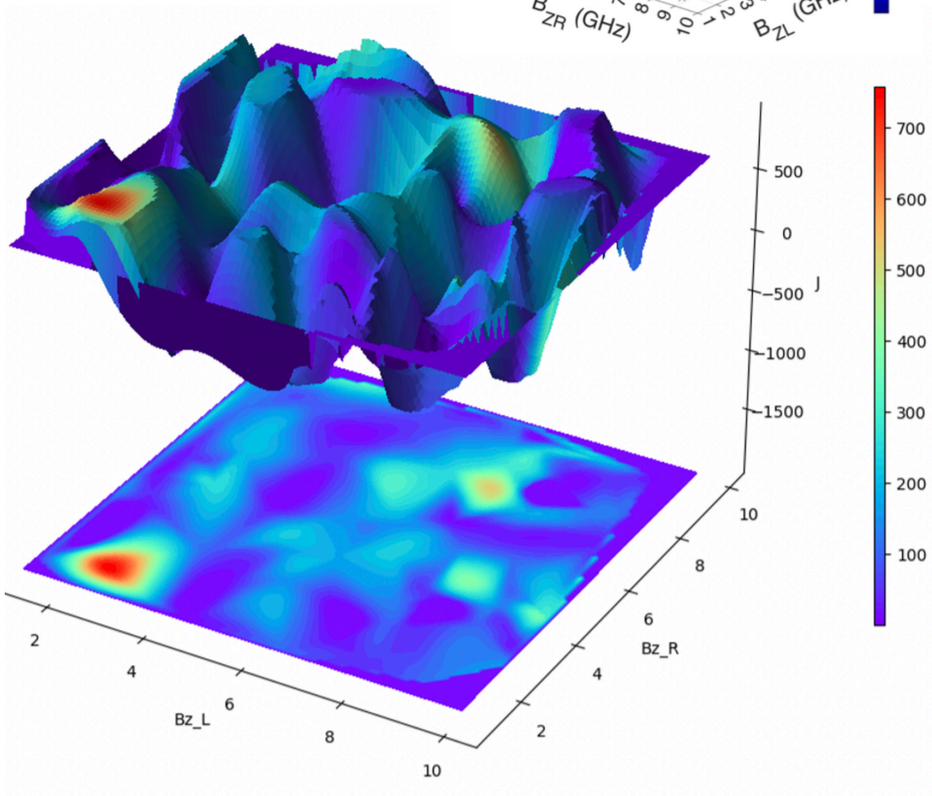
AutoCLINT: Computationally Light Network Training, ***NeurIPS AutoDL Challenges Winner (AutoCV, AutoCV2)*** (2019)

Joint work with W. Baek (Kakao Brain), I. Kim (Kakao Brain), S. Kim (Kakao Brain)

HPO for Device Optimization



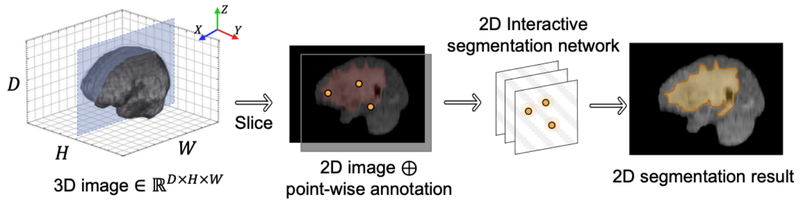




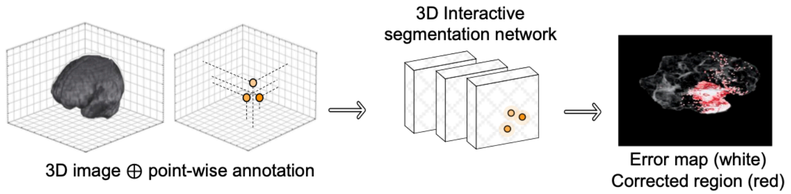
Procuring Design Factors of Efficient Si-base Quantum Logic Operations with Bayesian Optimization, ***submitted*** (2021)

Joint work with J. Kang (KISTI), T. Yoon (UNIST), C. Lee (UNIST), H. Ryu (KISTI)

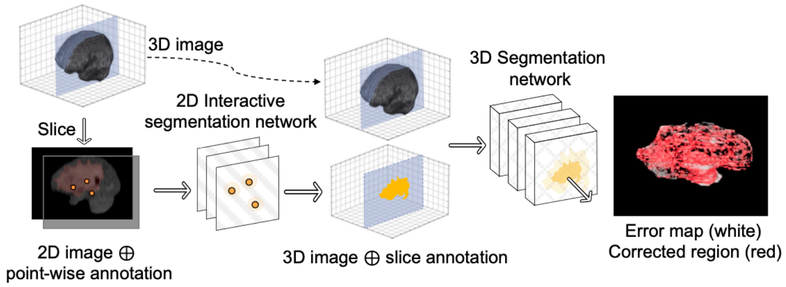
Human-in-the-loop ML



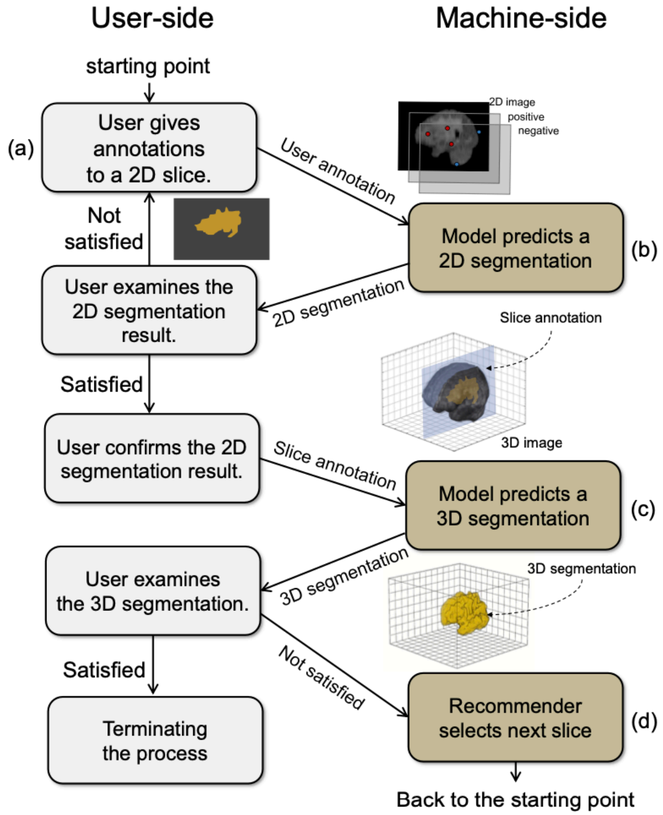
(a) Propagating point-wise annotation to a 2D image



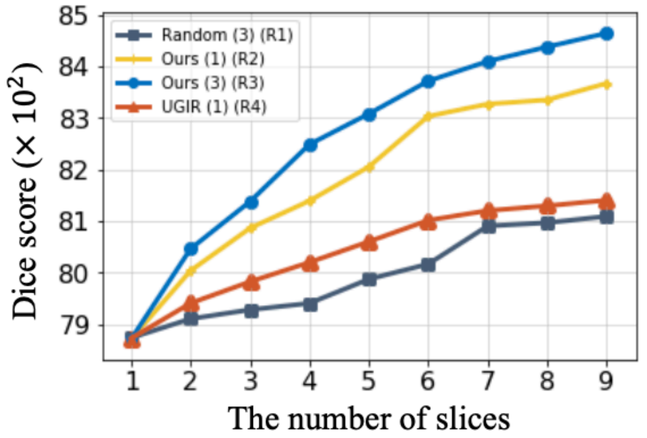
(b) Directly propagating point-wise annotation to a 3D image



(c) Gradually propagating point-wise annotation to a 3D image



MSD	Method	points 1	5	10	15	20	25
Heart	DOS [25]	87.95	87.96	87.96	87.95	87.88	87.86
	f-BRS [18]	87.95	83.0	86.92	83.92	83.90	84.12
	FCA [8]	82.77	82.53	82.66	82.63	82.55	82.52
	Ours-A		86.87	88.61	89.65	90.20	90.67
	Ours-B		**90.21**	**91.25**	**91.45**	**91.81**	**91.96**
Liver	DOS [25]	90.35	90.35	90.34	90.34	90.33	90.31
	f-BRS [18]	90.34	86.93	86.85	86.14	88.07	88.73
	FCA [8]	92.62	92.76	92.81	92.81	92.91	92.81
	Ours-A		92.17	92.99	93.55	93.64	93.69
	Ours-B		**95.37**	**95.25**	**95.22**	**95.22**	**95.27**
Pancreas	DOS [25]	74.48	75.27	75.06	74.77	74.62	74.51
	f-BRS [18]	73.87	70.53	74.60	75.57	77.41	77.29
	FCA [8]	64.26	67.09	67.08	67.11	67.18	67.21
	Ours-A		76.58	77.73	78.97	79.89	80.40
	Ours-B		**78.71**	**80.46**	**81.38**	**82.49**	**83.09**
Lung Tumor	DOS [25]	53.98	65.21	72.82	76.25	74.59	73.84
	f-BRS [18]	53.98	60.14	72.89	**76.94**	**78.62**	73.62
	FCA [8]	46.80	66.50	67.32	67.79	67.56	67.42
	Ours-A		67.22	72.14	72.76	74.21	75.16
	Ours-B		**72.27**	**74.83**	75.06	76.07	**76.68**
Brain Tumor	DOS [25]	88.54	89.06	89.30	89.44	89.57	89.7
	f-BRS [18]	88.54	87.42	88.61	88.65	88.98	89.16
	FCA [8]	88.60	**89.59**	89.76	89.84	89.92	89.94
	Ours-A		89.12	**89.56**	**90.00**	**90.25**	**90.08**
	Ours-B		88.68	89.61	89.96	89.97	89.72

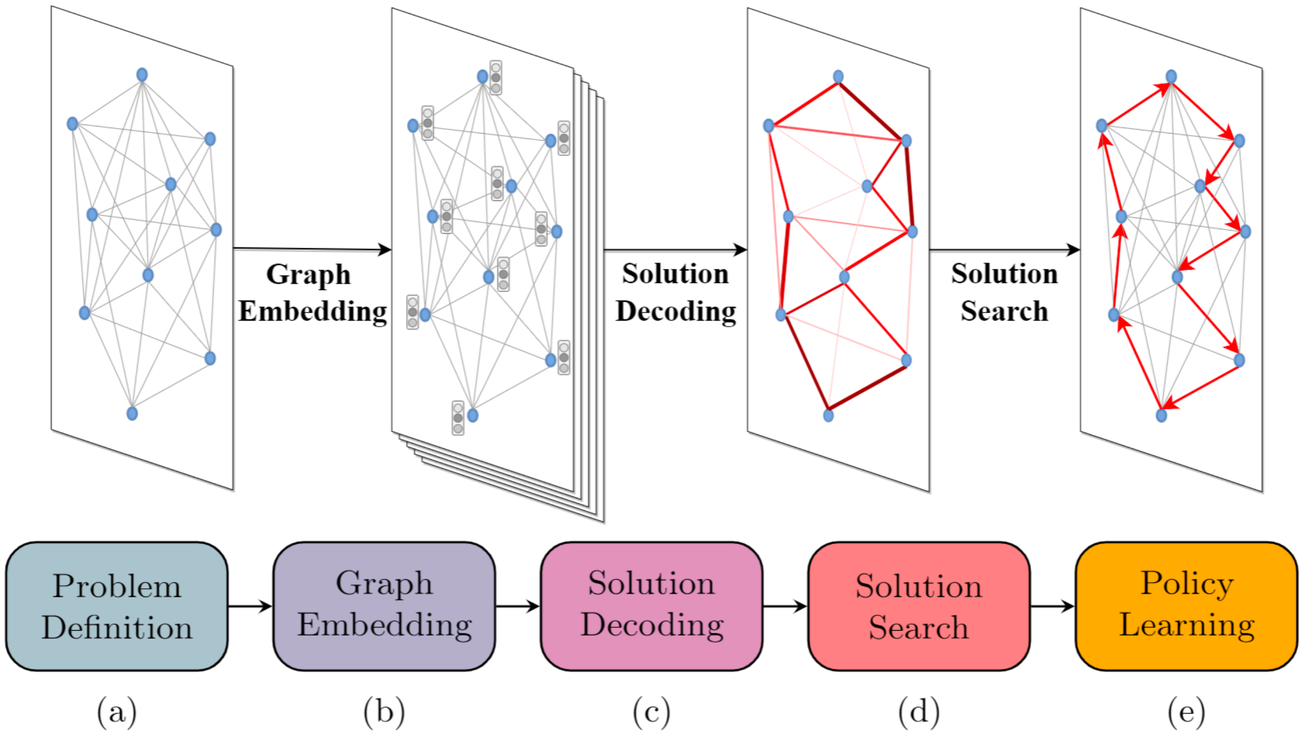


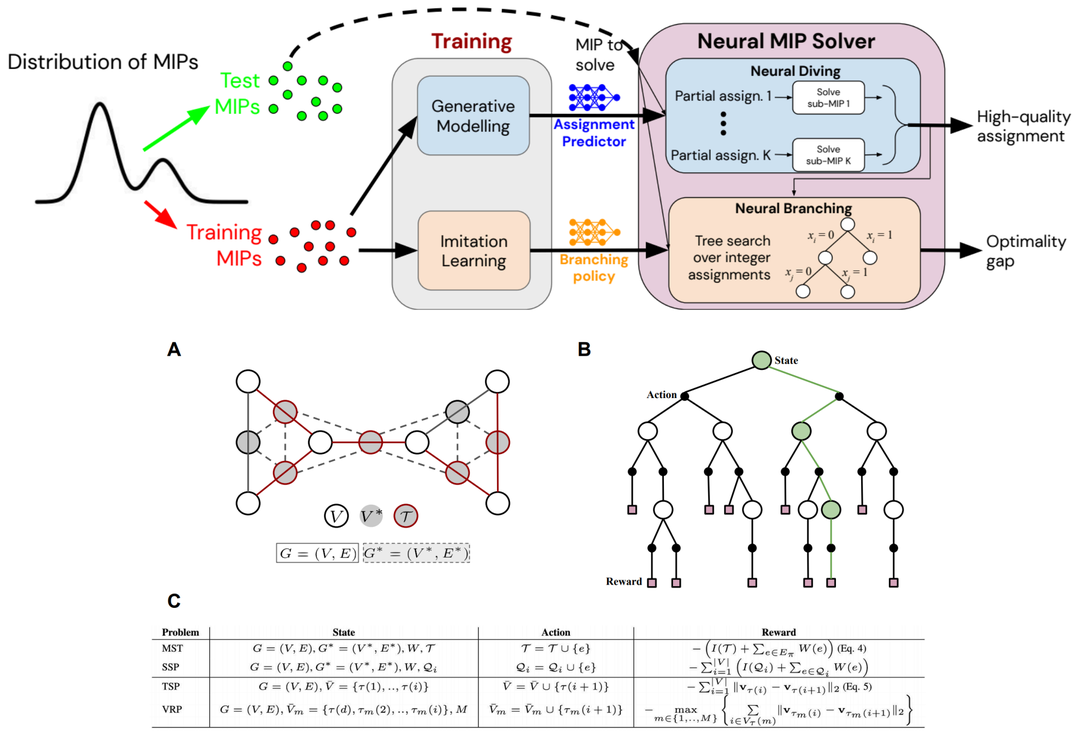
Slice and Conquer: Two-stage Interactive 3D Image Segmentation, ***submitted*** (2021)

Joint work with G. Sim (KAIST), J. Choi (Tomocube), H. Lim (KAIST), H. Min (Tomocube), J. Choo (KAIST)

Neural Combinatorial Optimization

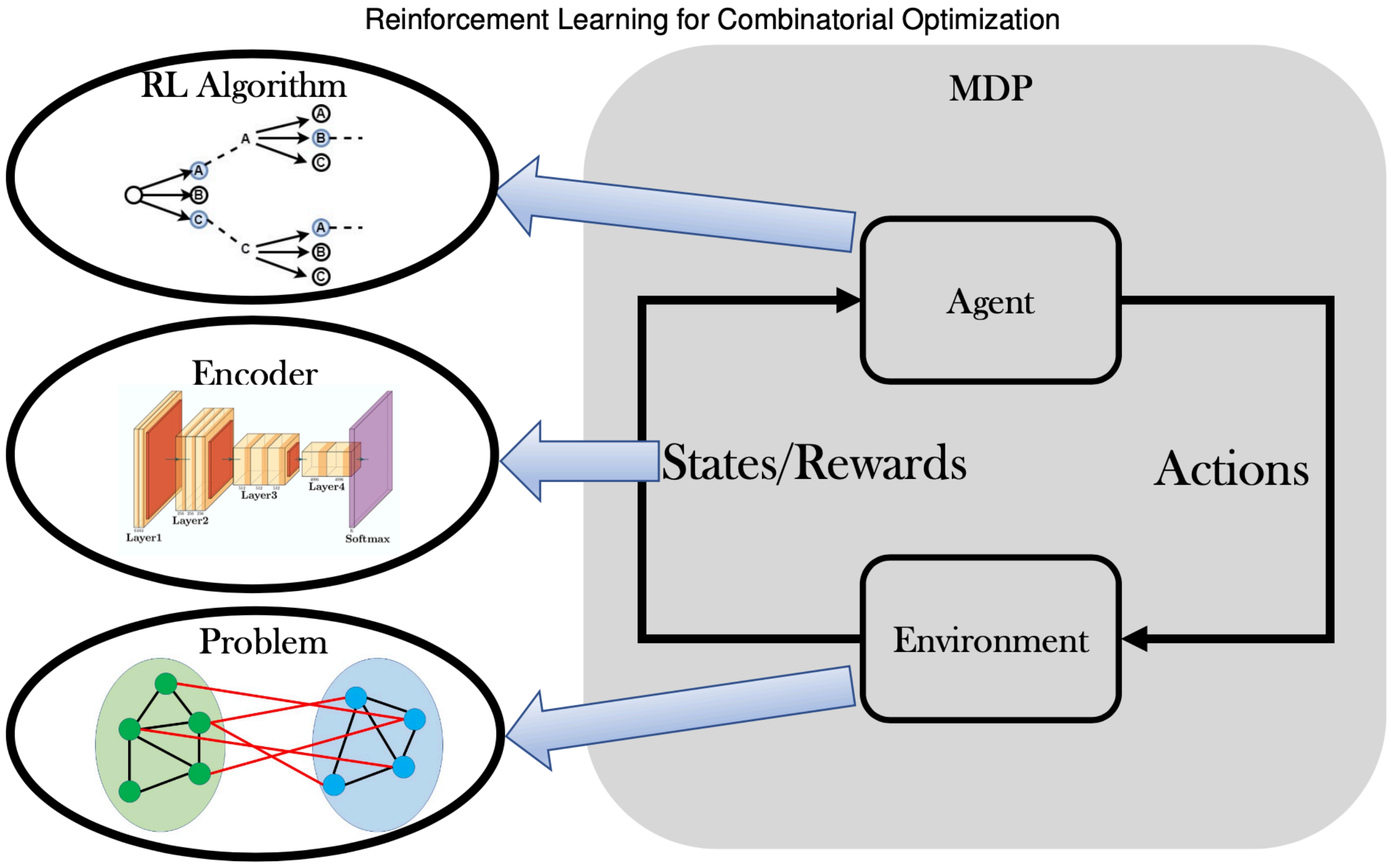
- Deep Learning can solve NP-hard combinatorial optimization

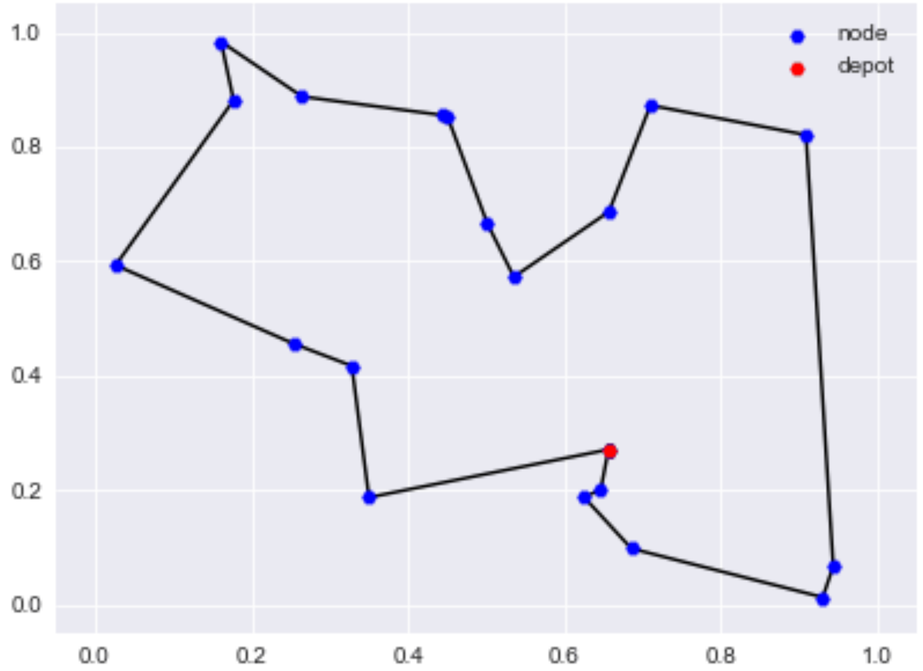




Problem	State	Action	Reward
MST	$G=(V,E), G^*=(V^*,E^*), W, \mathcal{T}$	$\mathcal{T}=\mathcal{T}\cup\{e\}$	$-\left(I(\mathcal{T})+\sum_{e\in E_\pi} W(e)\right)$ (Eq. 4)
SSP	$G=(V,E), G^*=(V^*,E^*), W, \mathcal{Q}_i$	$\mathcal{Q}_i=\mathcal{Q}_i\cup\{e\}$	$-\sum_{i=1}^{\lvert V\rvert}\left(I(\mathcal{Q}_i)+\sum_{e\in\mathcal{Q}_i} W(e)\right)$
TSP	$G=(V,E), \bar{V}=\{\tau(1),..,\tau(i)\}$	$\bar{V}=\bar{V}\cup\{\tau(i+1)\}$	$-\sum_{i=1}^{\lvert V\rvert}\lVert \mathbf{v}_{\tau(i)}-\mathbf{v}_{\tau(i+1)}\rVert_2$ (Eq. 5)
VRP	$G=(V,E), \bar{V}_m=\{\tau(d),\tau_m(2),..,\tau_m(i)\}, M$	$\bar{V}_m=\bar{V}_m\cup\{\tau_m(i+1)\}$	$-\max_{m\in\{1,..,M\}}\left\{\sum_{i\in\bar{V}_\tau(m)}\lVert \mathbf{v}_{\tau_m(i)}-\mathbf{v}_{\tau_m(i+1)}\rVert_2\right\}$

Neural Combinatorial Optimization





Neural Combinatorial Optimization

Primal task

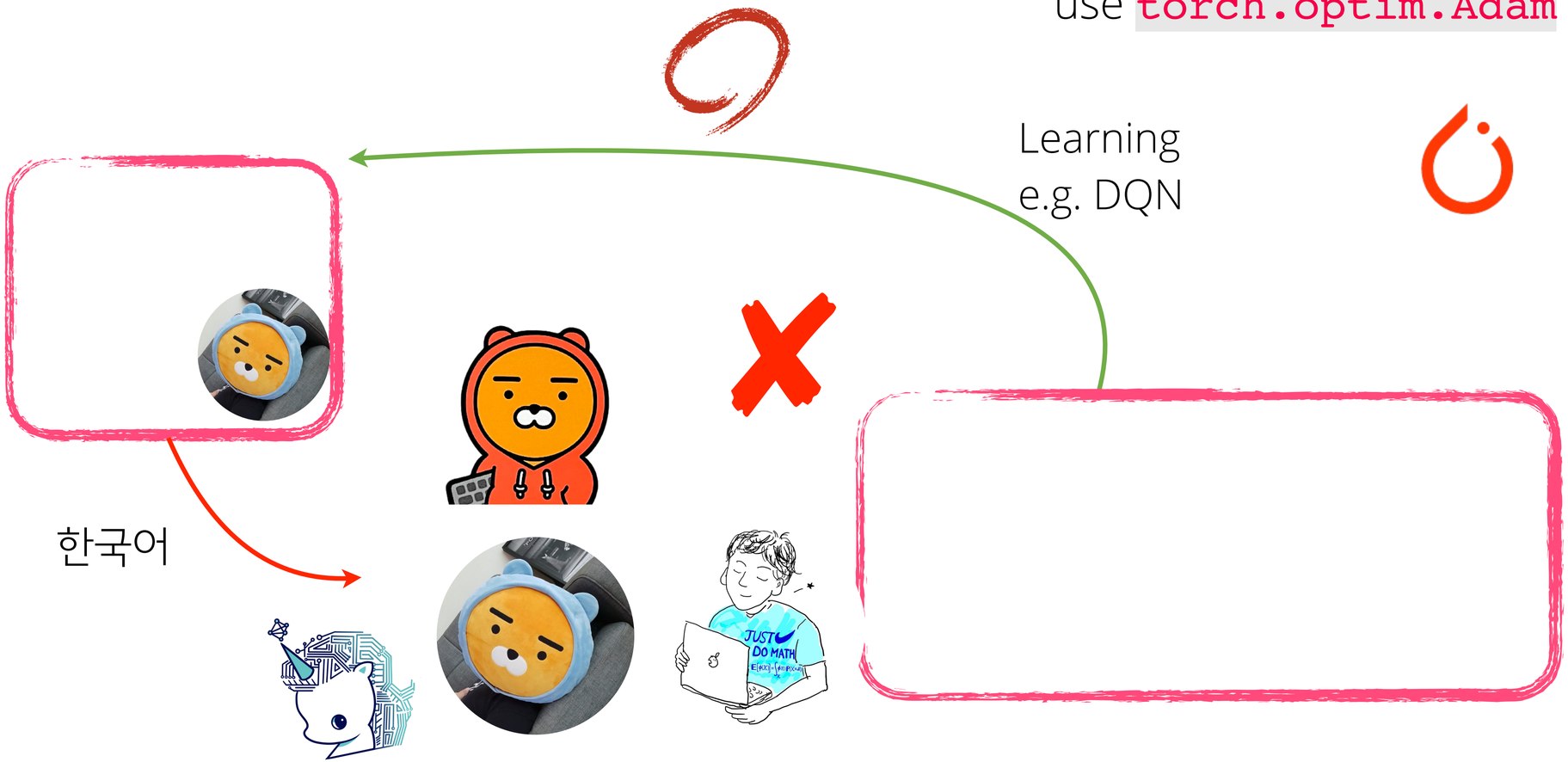
rank	team	item_placement		load_balancing			anonymous	
		primal integral	(cum. reward)	rank	primal integral	(cum. reward)	rank	primal integral	(cum. re
1	UNIST-LIM-Lab	3711.29	(-6797.28)	1	5000.58	(-218743.08)	1	390828725.99	(-413150
2	ethz-scuba	3999.04	(-7085.03)	2	8865.19	(-222607.69)	2	nan	
3	EasyML	181351.93	(-184437.92)	3	54345.18	(-268087.68)	3	552089962.36	(-574411
4	nf-lzg	181352.07	(-184438.06)	4	54346.18	(-268088.68)	4	552061475.56	(-574382

Q & A

Layer Selection ⇝ MDP

use `torch.optim.Adam`

Learning
e.g. DQN



QNAS: Designing Neural Network Architectures using RL, Baker et al., ***ICLR*** 2017

Variance modeling via MDN

- Mixture of Gaussian can approximate any given density function (if K is very large)

$$p(\mathbf{y}|\mathbf{x}) = \sum_{k=1}^{K} \pi_k(\mathbf{x}) \mathcal{N}(\mathbf{y}; \mu_k(\mathbf{x}), \Sigma_k(\mathbf{x}))$$

- Variance decomposition

$$\begin{aligned}\mathbb{V}(\mathbf{y}|\mathbf{x}) &= \mathbb{V}_k(\mathbb{E}[\mathbf{y}|\mathbf{x}, k]) + \mathbb{E}_k(\mathbb{V}(\mathbf{y}|\mathbf{x}, k)) \\ &= \mathbb{V}_k(\mu_k(\mathbf{x})) + \mathbb{E}_k(\Sigma_k(\mathbf{x})))\end{aligned}$$

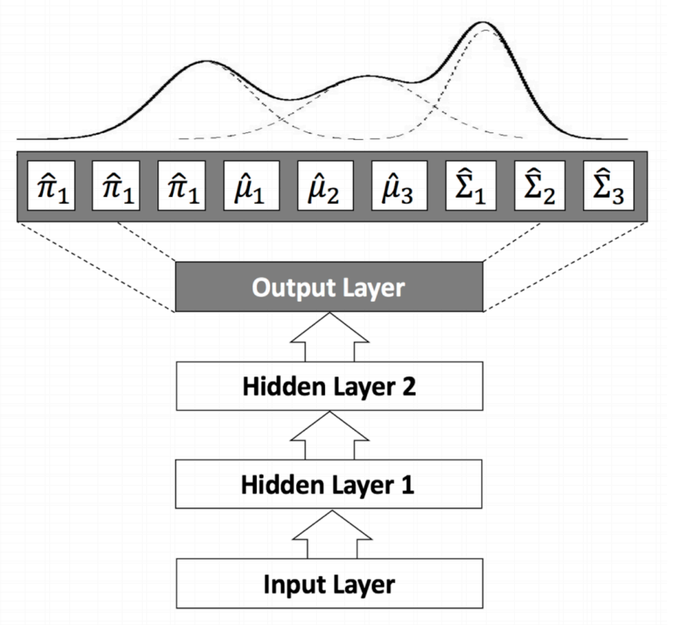


Fig. 1: A mixture density network ($K = 3$) with two hidden layers where the output of the network is decomposed into $\hat{\pi}$, $\hat{\mu}$, and $\hat{\Sigma}$.

Uncertainty-Aware Learning from Demonstration using MDN with Sampling-Free Variance Modeling, ***ICRA*** (2018)

Joint work with S. Choi (SNU), K. Lee (SNU), S. Oh (SNU)