Introduction to Hyper-Parameter Optimization

DMQA Open Seminar

2021.09.24

이민재

발표자 소개

❖ 이민재

- 고려대학교 산업경영공학과
- Data Mining & Quality Analytics Lab
- M.S. Student

❖ Research Interest

- Graph learning
- Efficient machine learning / AutoML

❖ Contact

- Email : raphaelnz@korea.ac.kr

Contents

Introduction

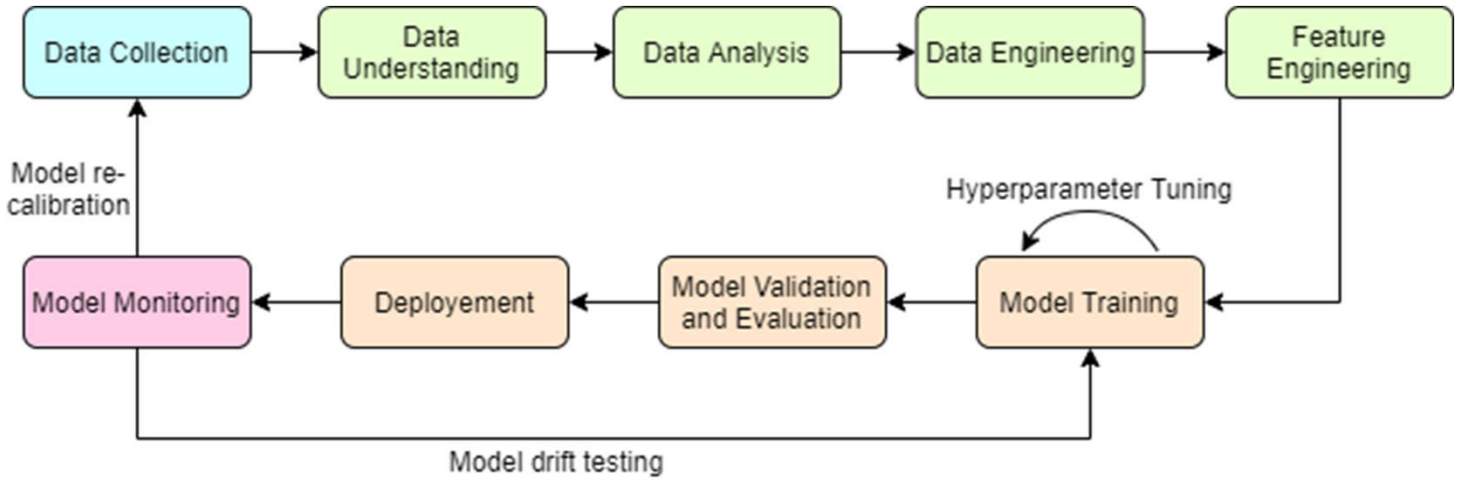
Introduction

Background

❖ 데이터 사이언스 실무는 Engineering과 뗄 수 없음

- 데이터 사이언스 라이프사이클 : [데이터 수집 → 데이터 정제 → 모델 학습/배포 → 모니터링] → […] → …

데이터 사이언스 파이프라인 (라이프사이클)



- https://miro.medium.com/max/705/1*WKJKqDs7GXJ-KAy6l2cUXg.png

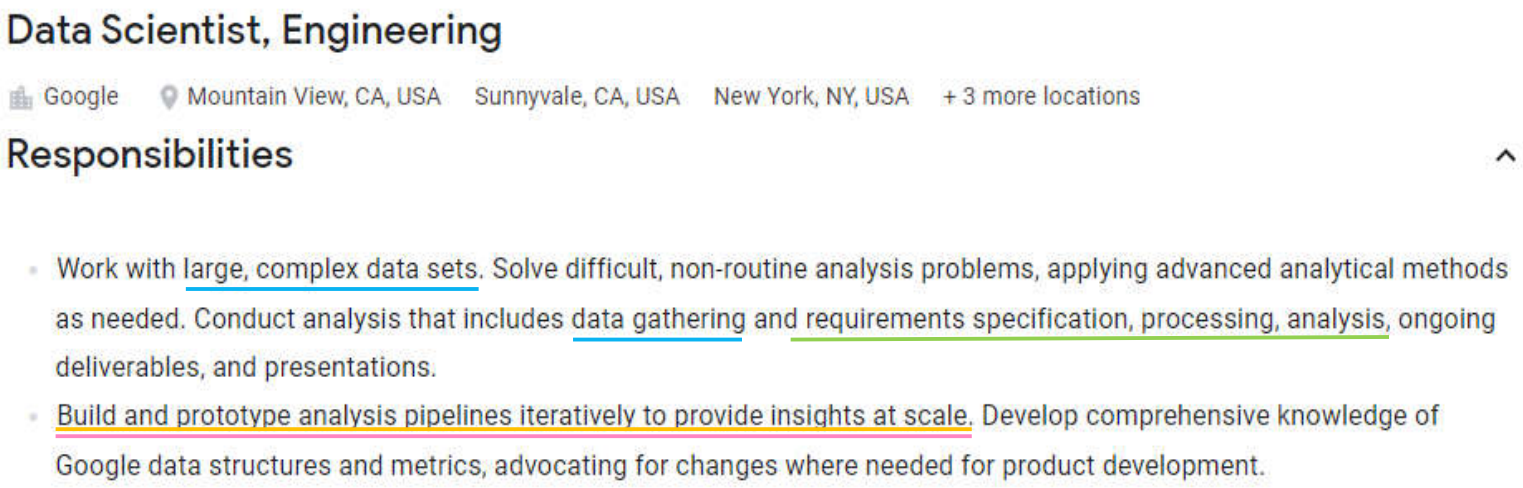
Introduction

Background

❖ 실.제.로 데이터 '사이언티스트' 업무에도 Engineering가 강조됨

- 데이터 사이언스 라이프사이클 : [데이터 수집 → 데이터 정제 → 모델 학습/배포 → 모니터링] → [⋯] → ⋯

구글 데이터 사이언티스트 모집 공고 (2021.09)



Data Scientist, Engineering

Google Mountain View, CA, USA Sunnyvale, CA, USA New York, NY, USA + 3 more locations

Responsibilities

- Work with large, complex data sets. Solve difficult, non-routine analysis problems, applying advanced analytical methods as needed. Conduct analysis that includes data gathering and requirements specification, processing, analysis, ongoing deliverables, and presentations.
- Build and prototype analysis pipelines iteratively to provide insights at scale. Develop comprehensive knowledge of Google data structures and metrics, advocating for changes where needed for product development.

- https://careers.google.com/jobs/results/103340634112172742-data-scientist-engineering/

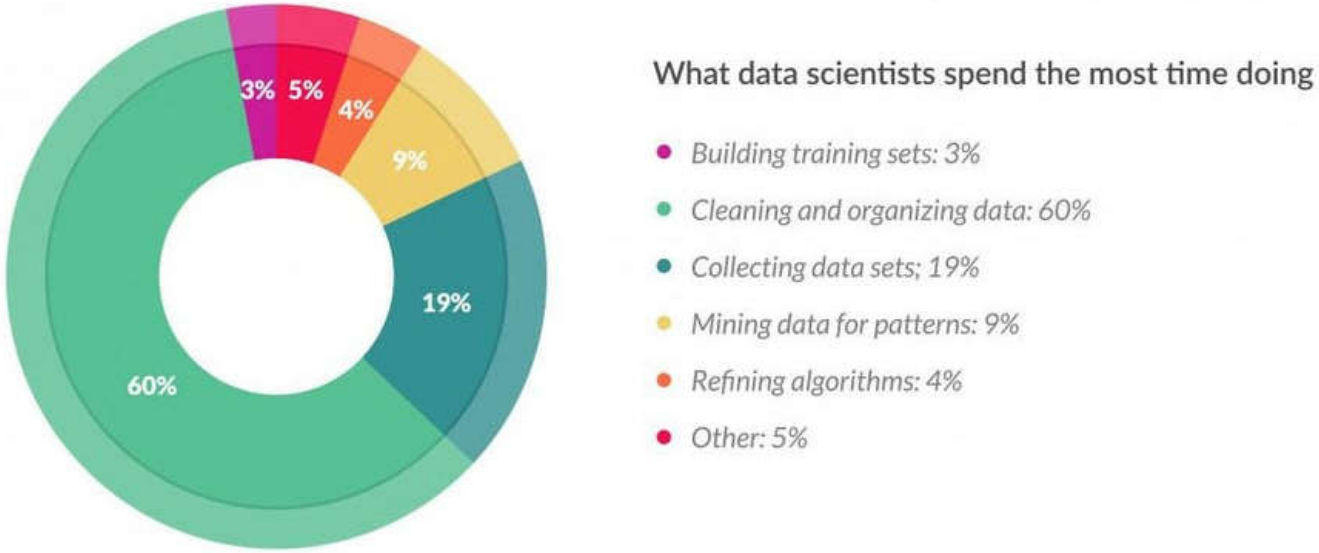
Introduction

Background

❖ 실.제.로 데이터 '사이언티스트' 업무에도 Engineering가 강조됨

- 데이터 사이언스 라이프사이클 : [데이터 수집 → 데이터 정제 → 모델 학습/배포 → 모니터링] → […] → …

데이터 사이언티스트의 하루 일과



- https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/?sh=5a6d0c866f63

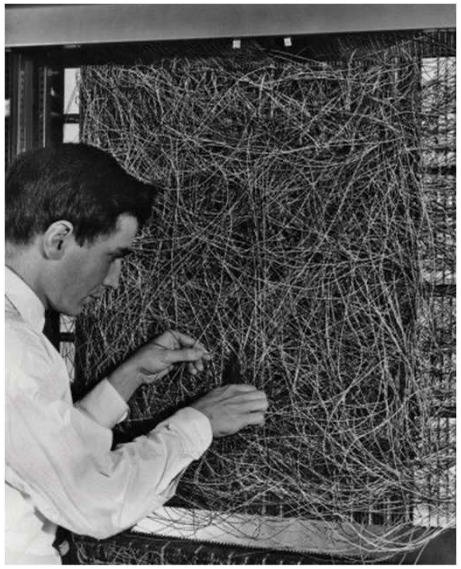
Introduction

Background

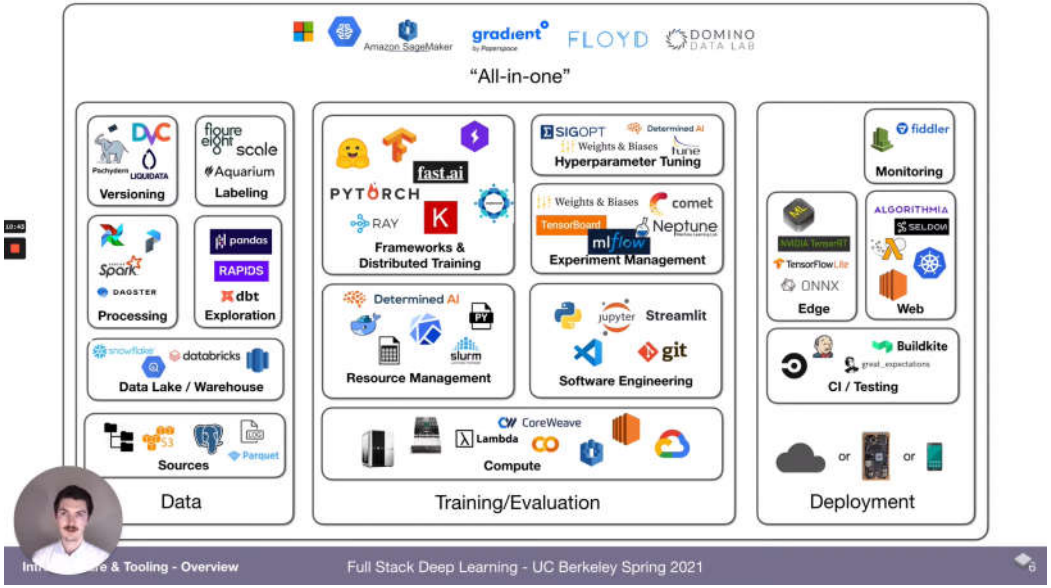
❖ 데이터 사이언스 분야에 Engineering 측면이 점차 더 강조되고 있음

- 1. [머신러닝 생태계의 성숙화] → 진입장벽이 과거보다 낮아진만큼 빠른 개발의 요구 증가 (개발주기↓)
- 2. [모델 복잡도와 데이터 증가] → 모델 학습은 어려워지는데 산업주기는 짧아지므로 더욱 빠른 개발 방법론 필요

1970년대 머신러닝 도구



2020년대 머신러닝 도구



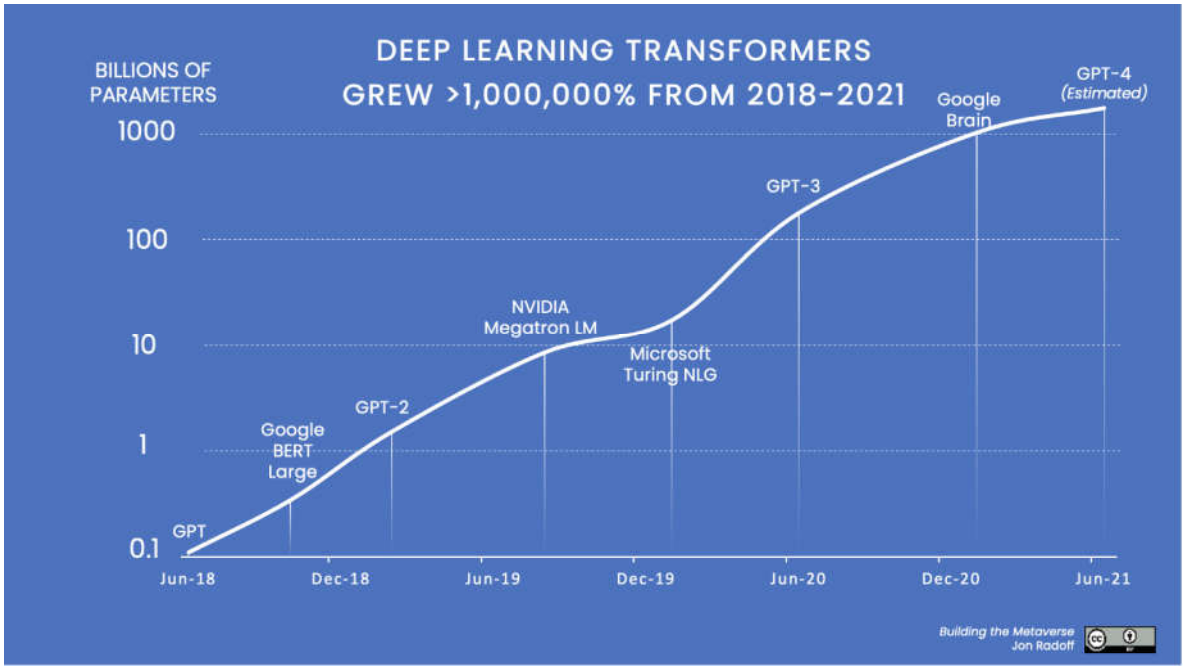
- https://en.wikipedia.org/wiki/Perceptron
- https://fullstackdeeplearning.com/spring2021/

Introduction

Background

❖ 데이터 사이언스 분야에 Engineering 측면이 점차 더 강조되고 있음

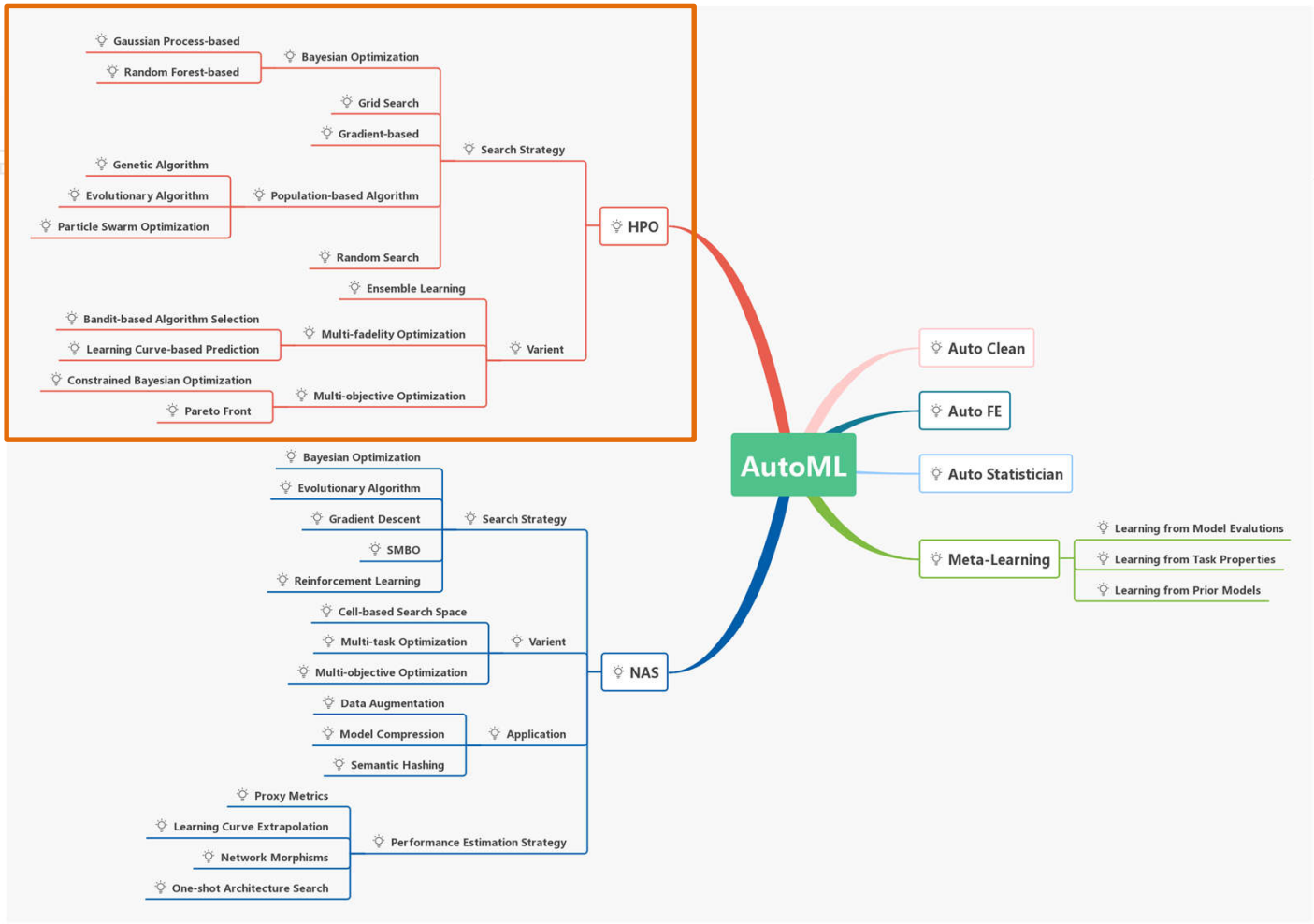
- 1. [머신러닝 생태계의 성숙화] → 진입장벽이 10년전보다 낮아진만큼 빠른 개발의 요구 증가 (개발주기 ↓)
- 2. [모델 복잡도와 데이터 증가] → 모델 학습은 어려워지는데 산업주기는 짧아지므로 더욱 빠른 개발 방법론 필요



- https://medium.com/building-the-metaverse/the-metaverse-and-artificial-intelligence-ai-577343895411
- https://github.com/hibayesian/awesome-automl-papers

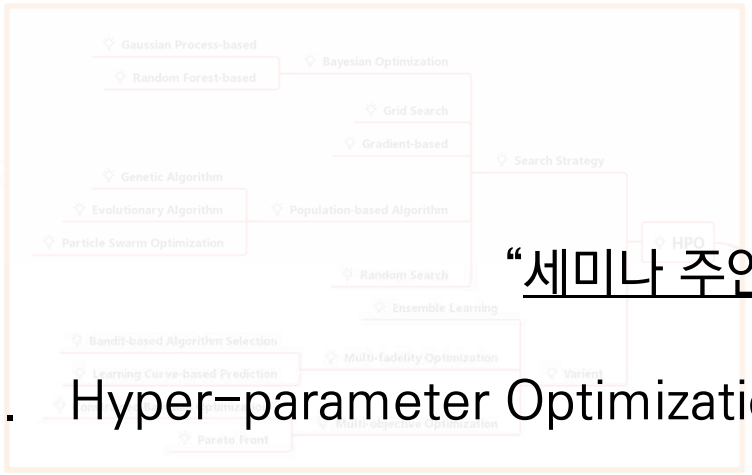
Introduction

Background



- https://medium.com/building-the-metaverse/the-metaverse-and-artificial-intelligence-ai-577343895411
- https://github.com/hibayesian/awesome-automl-papers

Introduction



"세미나 주안점"

1. Hyper-parameter Optimization(HPO)의 기본 방법론 소개
2. 현 시점(2021.09) 실무에서 쓰일 수 있는 SOTA 방법론 소개

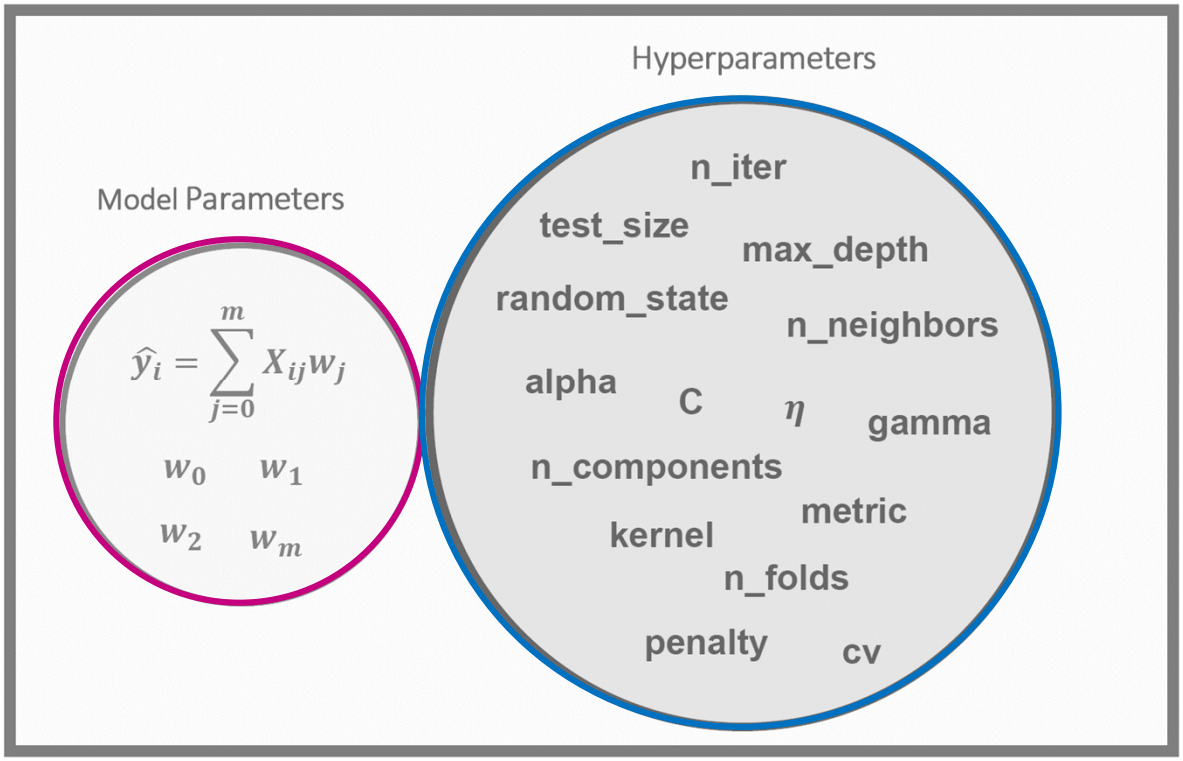
- https://medium.com/building-the-metaverse/the-metaverse-and-artificial-intelligence-ai-577343895411
- https://github.com/hibayesian/awesome-automl-papers

Introduction

Preliminaries

❖ Model-parameter *vs* Hyper-parameter

- 전자(●) : 학습 데이터에 적합되어야할 값 (e.g. 모델 가중치 $\boldsymbol{w}$)
- 후자(●) : 학습 데이터에 의해서 변경되지 않는 값 (e.g. 모델 깊이 d)



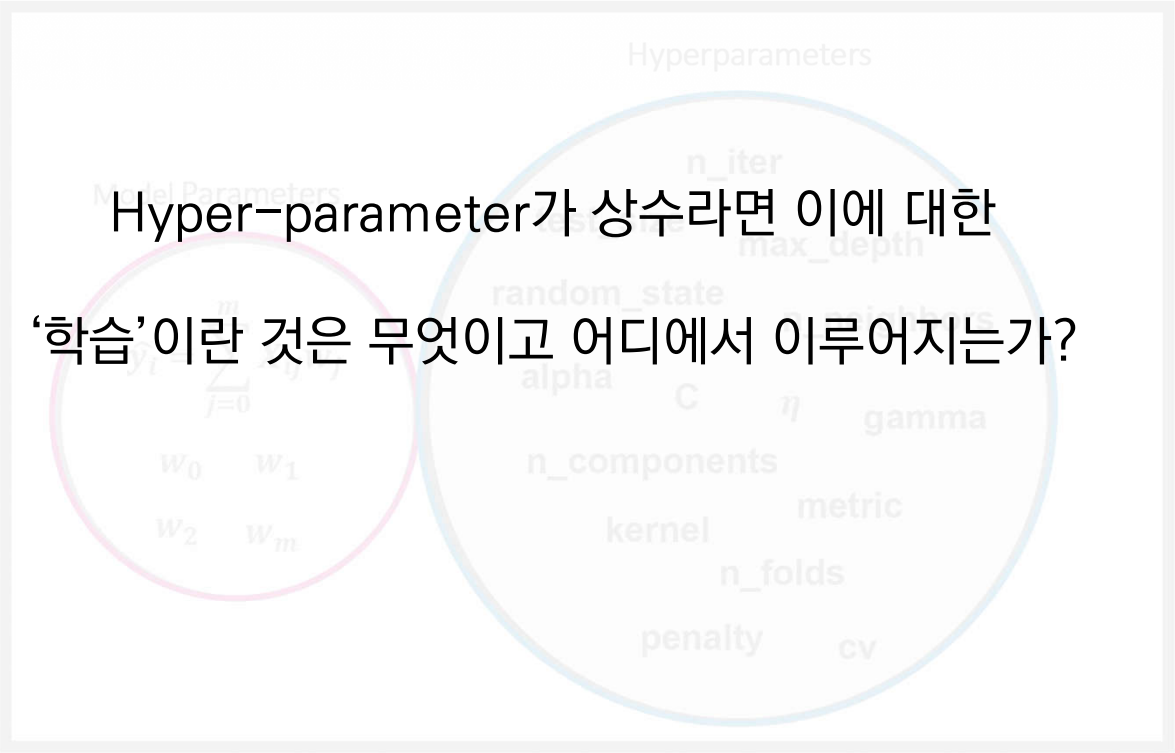
- https://towardsdatascience.com/model-parameters-and-hyperparameters-in-machine-learning-what-is-the-difference-702d30970f6

Introduction

Preliminaries

❖ Model-parameter *vs* Hyper-parameter

- 전자(●) : 학습 데이터에 적합되어야할 값 (e.g. 모델 가중치 w)
- 후자(●) : 학습 데이터에 의해서 변경되지 않는 값 (e.g. 모델 깊이 d)



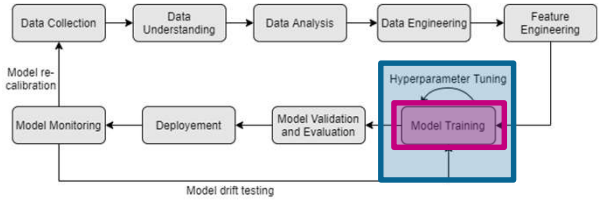
Hyper-parameter가 상수라면 이에 대한

'학습'이란 것은 무엇이고 어디에서 이루어지는가?

- https://towardsdatascience.com/model-parameters-and-hyperparameters-in-machine-learning-what-is-the-difference-702d30970f6

Introduction

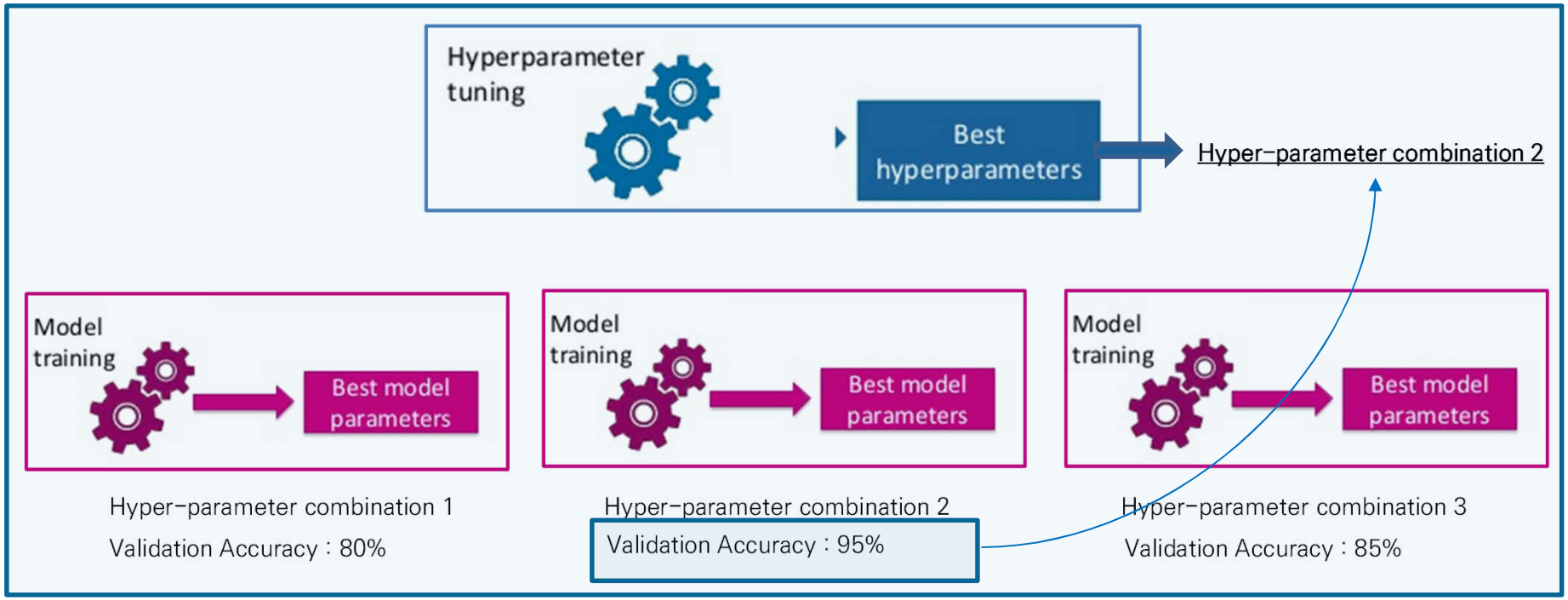
Preliminaries



❖ Model Training(●) *vs* Hyper-parameter Optimization(HPO)(●)

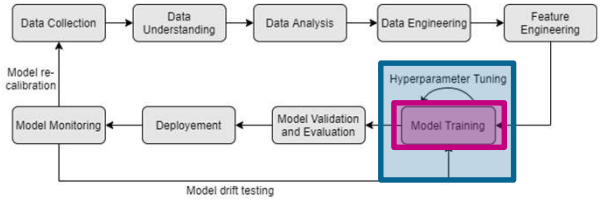
- 전자(●)는 Model parameter를 학습함 (고정된 hyperparameter 상에서) : Inner Loop
- 후자(●)는 Hyper-parameter를 학습함 (전자에 대한 Meta-optimization) : Outer Loop

Model Training vs Hyper-parameter Optimization(Tuning)



- https://learning.oreilly.com/library/view/evaluating-machine-learning/9781492048756/ch04.html#idp1753984

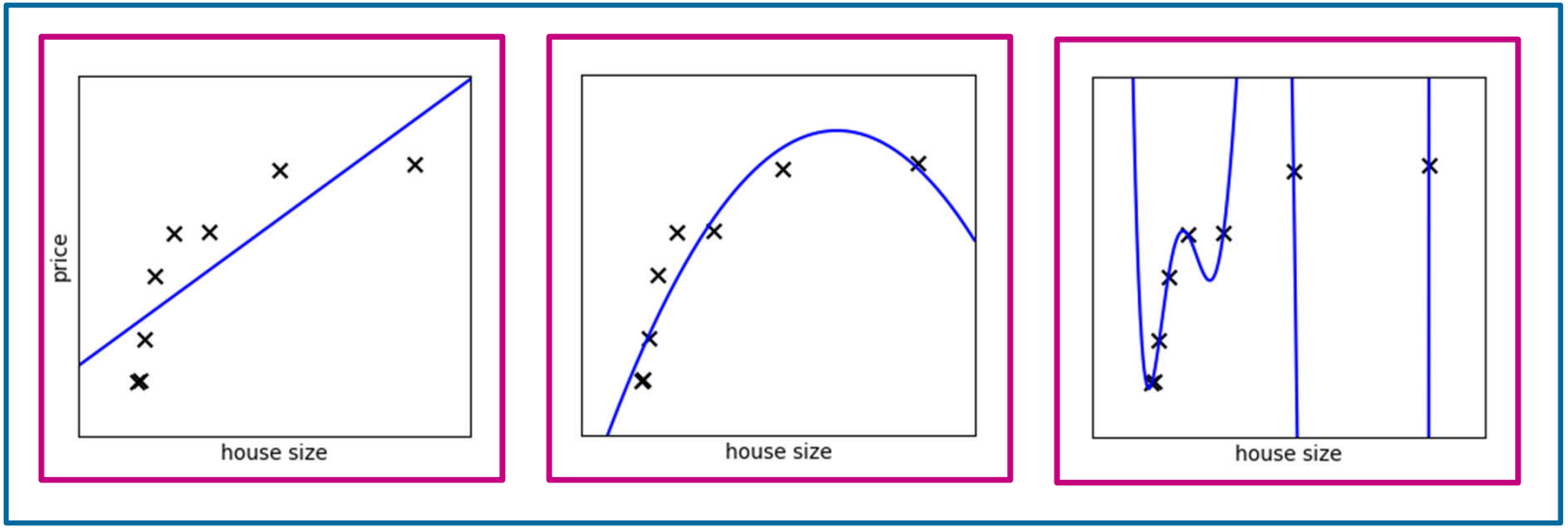
Introduction



Preliminaries

❖ Model Training(●) *vs* Hyper-parameter Optimization(HPO)(●)

- 전자(●): $\text{Cost} = \sum loss(y, \hat{y})$ 를 최소화하는 학습 파라미터 $\boldsymbol{w}$ 를 찾음
  - ➢ $d = 1 : (\hat{y} = f(\boldsymbol{w})_d = f(w_1, w_0)_1 = \ w_1 x + w_0)$
  - ➢ $d = 2 : (\hat{y} = f(\boldsymbol{w})_d = f(w_2, w_1, w_0)_2 = \ w_2 x^2 + w_1 x + w_0)$
  - ➢ $d = 6 : (\hat{y} = f(\boldsymbol{w})_d = f(w_6, w_5, \ldots, w_0)_6 = w_6 x^6 + w_5 x^5 + \cdots + w_0)$
- 후자(●): $f(\boldsymbol{w})_d$ 중, Validation set에 대한 성능(e.g. Valdiation Acc.)이 가장 좋게 만드는 Trial의 d 선택



$d = 1$ $d = 2$ $d = 6$

- https://scipy-lectures.org/packages/scikit-learn/auto_examples/plot_bias_variance.html

Introduction

Taxamony

❖ Classic taxamony of hyper-parameter optimization(HPO)

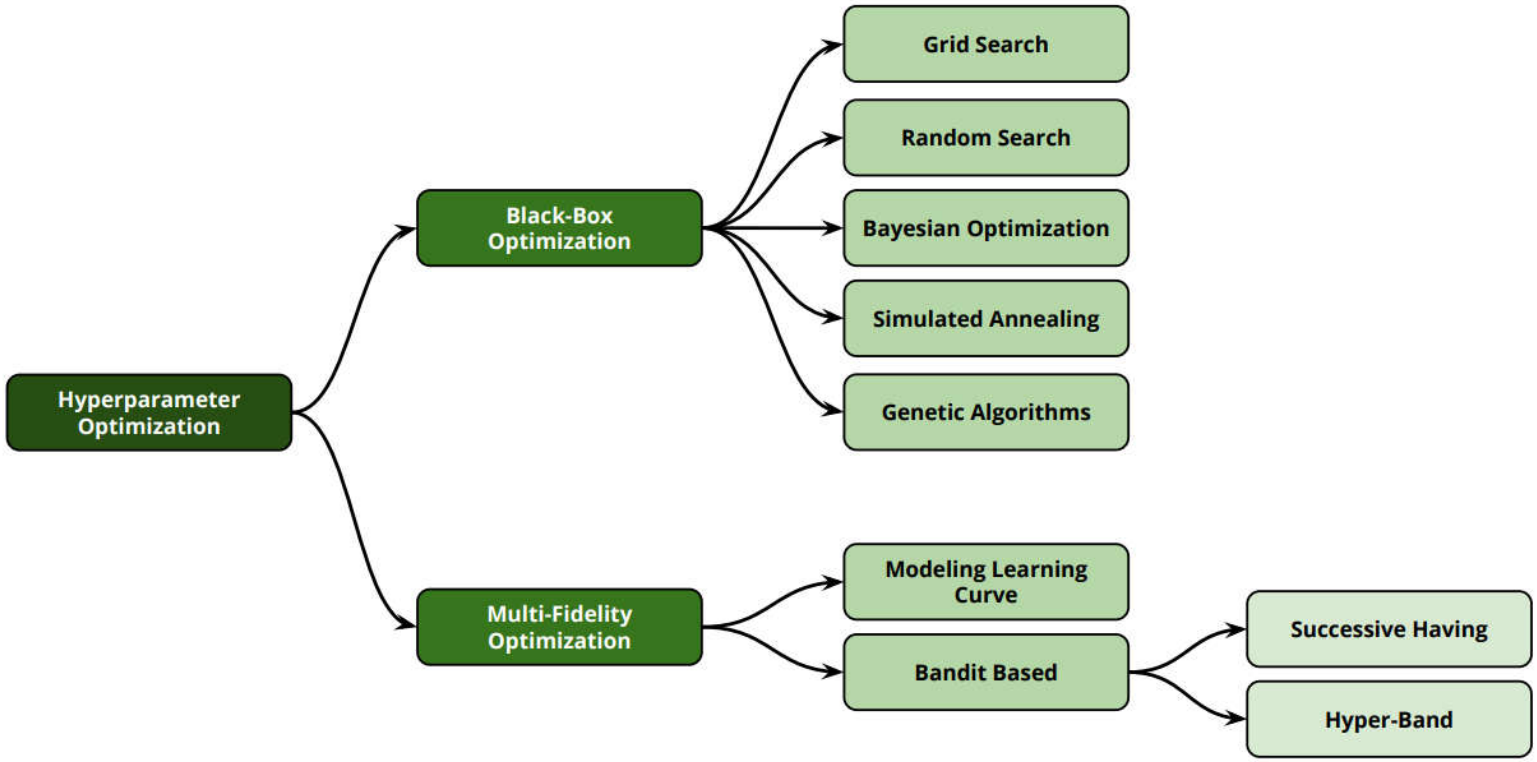
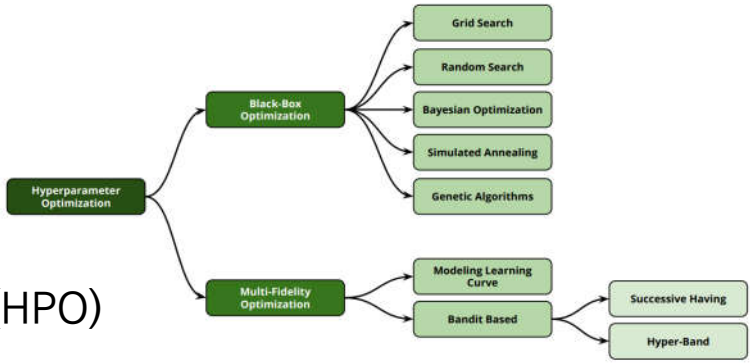


Figure 5: A Taxonomy for the Hyper-parameter Optimization Techniques.

- https://arxiv.org/pdf/1906.02287.pdf

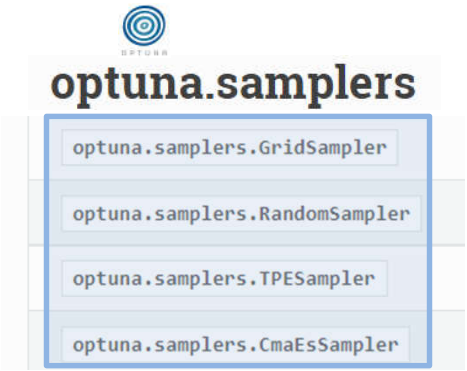
Introduction

Taxamony

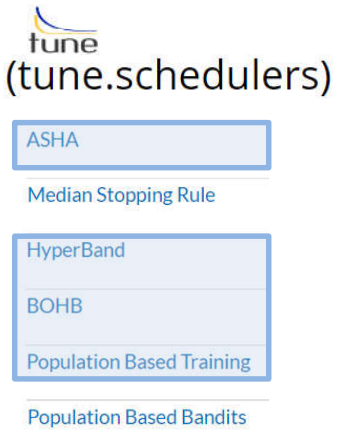


❖ Classic taxamony of hyper-parameter optimization(HPO)

1. Searching method(sampler)
   - *"조기종료(pruning) 없이 매 trial을 끝까지 탐색"*
   - Grid Search
   - Random Search
   - Bayesian Optimization
   - Evolutionary Search



2. Scheduling method
   - *"'Searching 도중에 성능이 안 좋은 Trial을 조기종료(prune)하여 전체 탐색 속도 ↑"*
   - Successive Halving
   - HyberBand / BOHB / ASHA
   - Population Based Training

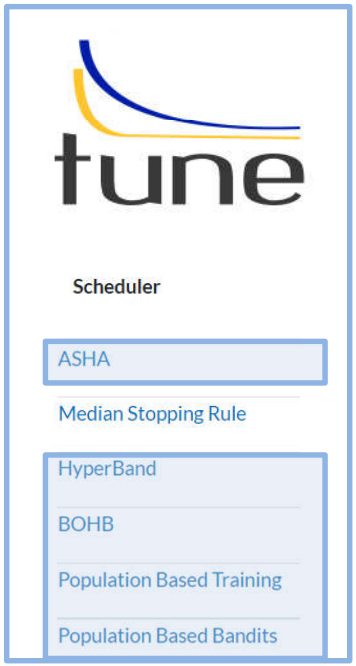


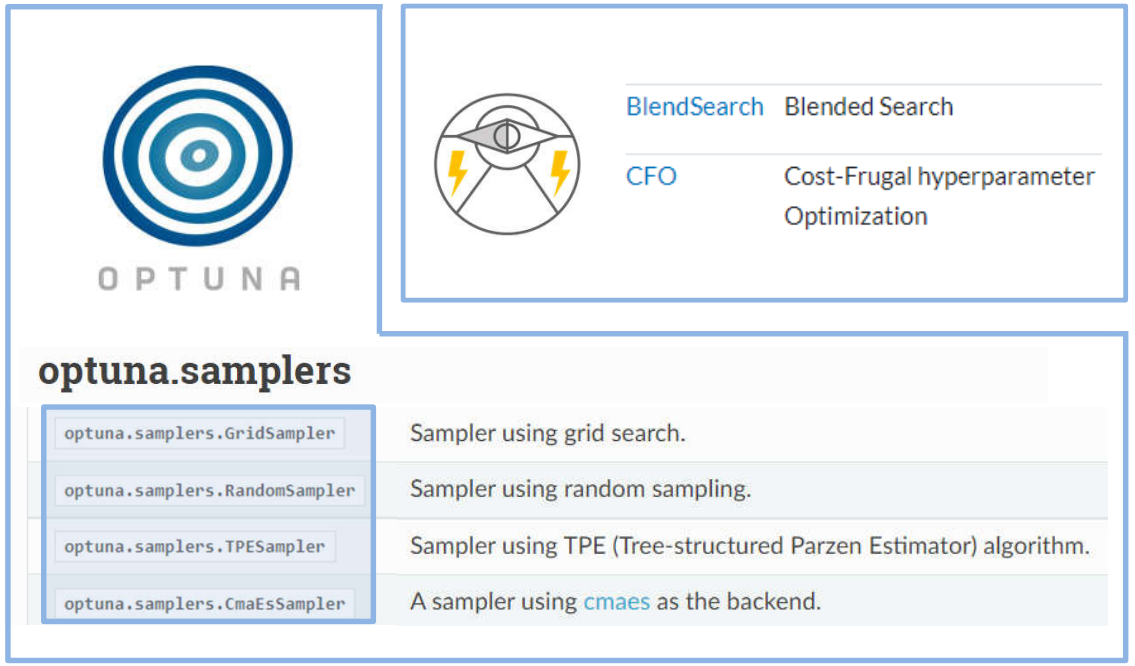
- https://arxiv.org/pdf/1906.02287.pdf

Introduction

❖ 이후 소개될 모든 방법론은 아래 HPO 라이브러리로부터 '손쉽게' 사용가능

- Ray Tune (☆:17.4k) (Made by UC Berkley, Widely used for distributed computing)
- Optuna (☆:5.2k) (CMA-ES Accepted @(AAAI2021), Compatibe with majority of the ML tech stacks (sklearn,PyTorch,TF,XGBoost,..)
- FLAML (☆:1.3k) (Made by Microsoft, accepted @(MLSys, AAAI2021, ICLR2021, ICML2021), Current(2021.09) SOTA @AutoML Benchmark)





- https://github.com/optuna/optuna
- https://github.com/ray-project/ray
- https://github.com/microsoft/FLAML

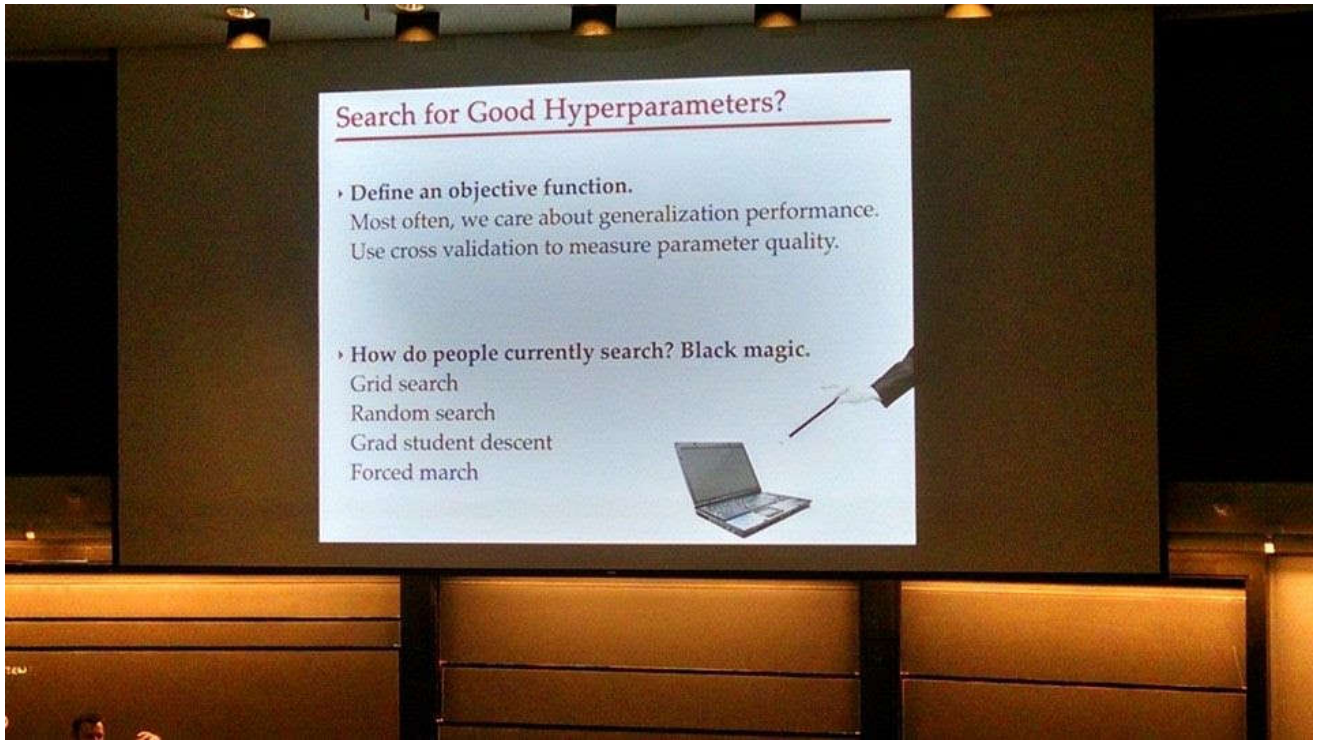
Searching Method

Methods

Searching method

❖ Baby Sitting

- 오직 실험자의 경험에 기반하여 모델이 잘 작동할 때까지 수동으로 하이퍼파라미터 조정
- Graduate Student Descent(GSD) 라고도 함



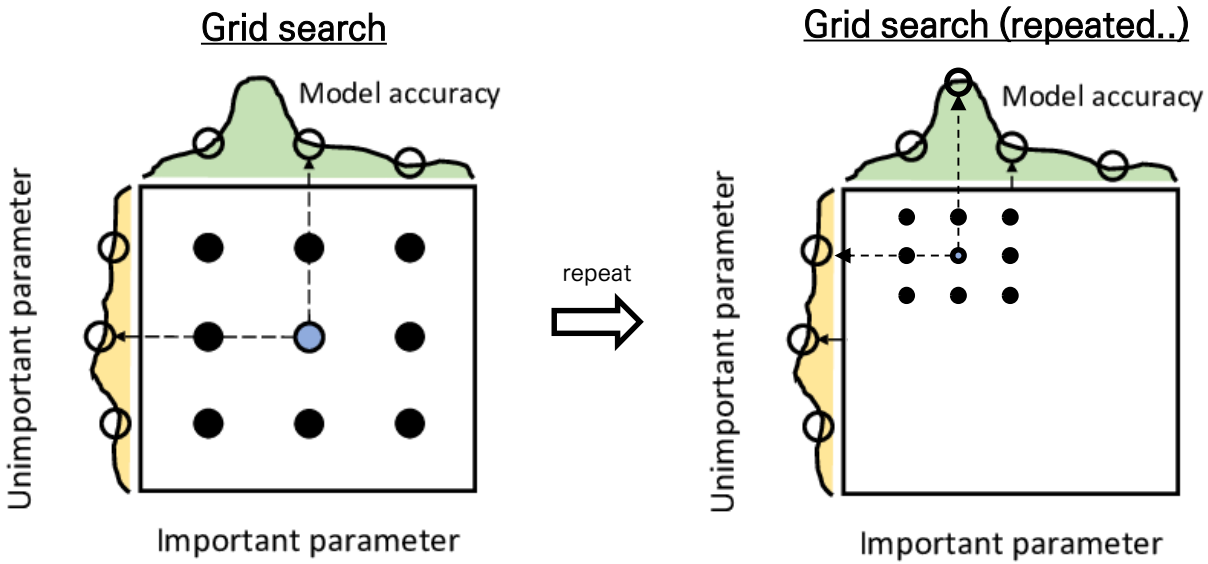
- https://twitter.com/GuyZys/status/592847074170896384/photo/1

Methods

Searching method

❖ Grid Search

- 사전 정의된 n^k 쌍의 Hyper-parameter를 탐색 (아래 예시: $Num\ of\ Trials = n^k = 3^2 = 9$)
- (+) 구현이 쉬우며, 각 Trial이 독립적이므로 병렬화 용이
- (-) Brute-force 탐색이며, Global optimum을 찾기 위해서는 탐색 범위를 수동으로 좁혀나가야함



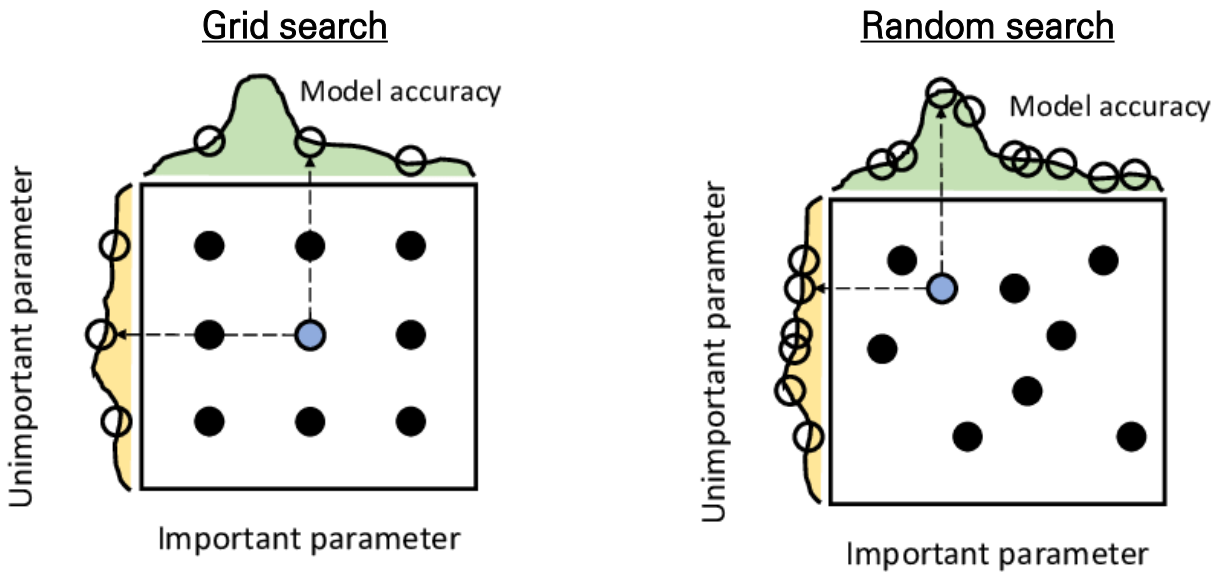
- https://www.researchgate.net/figure/Comparison-between-a-grid-search-and-b-random-search-for-hyper-parameter-tuning-The_fig2_341691661

Methods

Searching method

❖ Random Search

- 사전 정의된 Grid 상에서 무작위로 뽑힌 n쌍의 Hyper-parameter를 탐색 (아래 예시 : $Num\ of\ Trials = n = 9$)
- (+) 구현이 쉬우며, 각 Trial이 독립적이므로 병렬화 용이 + 시간복잡도는 $O(n)$ (cf: Grid Search $O(n^k)$)
- (+) "같은 자원이 주어질때 Grid search에 비해 global optimum을 찾을 확률이 더 높음" [Bengio et al.(2012)]
- (-) Grid Search에 비해 결과 해석이 어려움



- https://www.researchgate.net/figure/Comparison-between-a-grid-search-and-b-random-search-for-hyper-parameter-tuning-The_fig2_341691661

Methods

Searching method

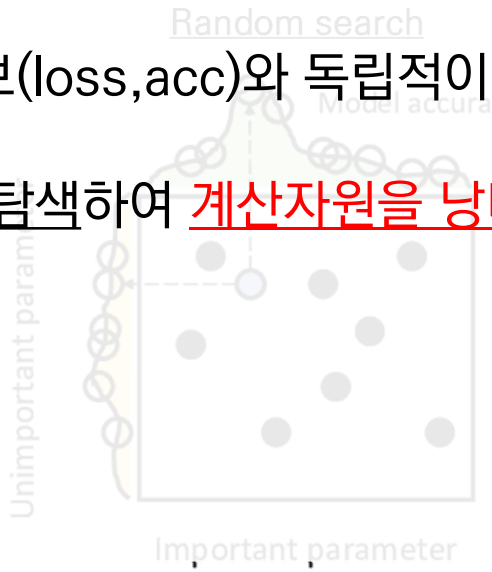
❖ Random Search

- 사전 정의된 Grid 상에서 무작위로 뽑힌 n쌍의 Hyper-parameter를 탐색 (아래 예시 : $Num\,of\,Trials = n = 9$)
- (+) 구현이 쉬우며, 각 Trial이 독립적이므로 병렬화 용이 + 시간복잡도는 $O(n)$ (cf: Grid Search $O(n^k)$)
- (+) "같은 자원이 주어질때 Grid search에 비해 global optimum을 찾을 확률이 더 높음" [Bergstra et al. (2012)]
- (-) Grid Search에 비해 결과 해석이 어려움

"Grid search / Random search의 단점"

후속 trial은 기존 Trial들의 평가 정보(loss,acc)와 독립적이기에,

성능이 좋지 않았던 근처도 모두 끝까지 탐색하여 계산자원을 낭비하게 됨



- https://www.researchgate.net/figure/Comparison-between-a-grid-search-and-b-random-search-for-hyper-parameter-tuning-The_fig2_341691661

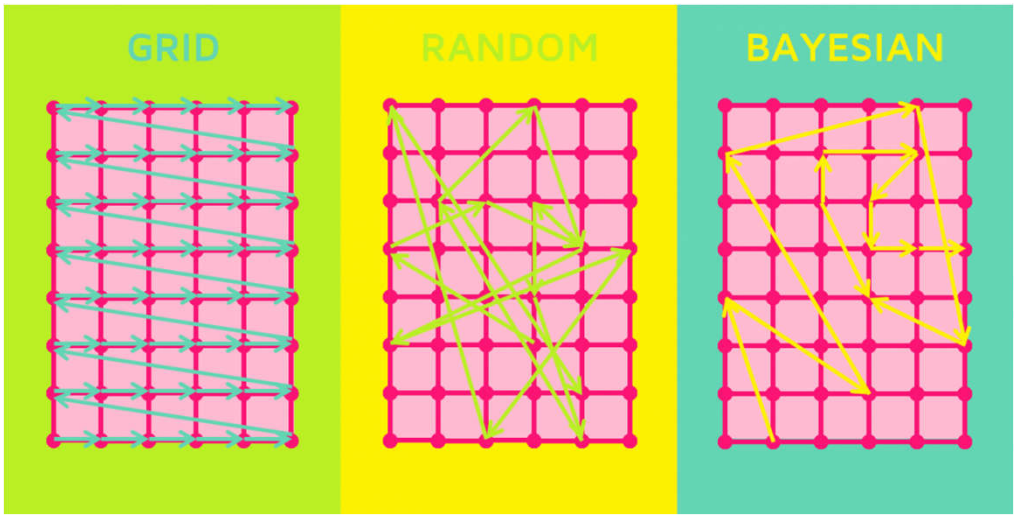
Methods

Searching method

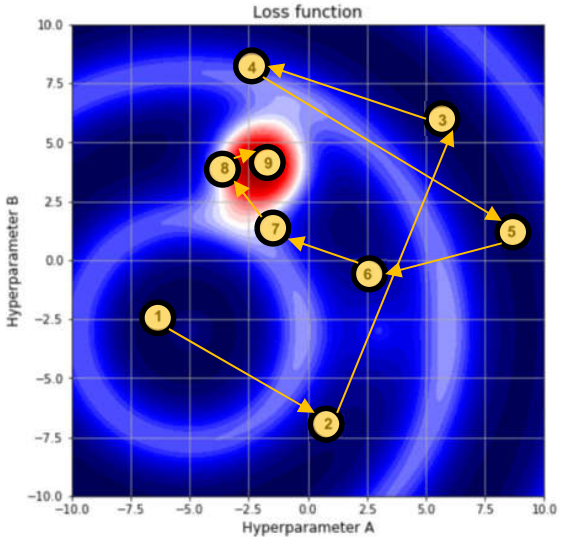
❖ Bayesian Optimization

- "기존 Trial들의 정보를 이용하여 후속 Trial의 Hyper-parameter를 선택 → 불필요한 자원소모 ↓"
- 앞선 정보를 활용하여 최적화한다는 점에서 Sequential model-based optimization 으로 분류

[HPO] Grid vs Random vs Bayesian



Bayesian search example



- https://quantdare.com/wp-content/uploads/2020/05/hyper2-1150x580.png
- https://miro.medium.com/max/770/1*BaYQTFh9R9T3sf2c7tUfTQ.png

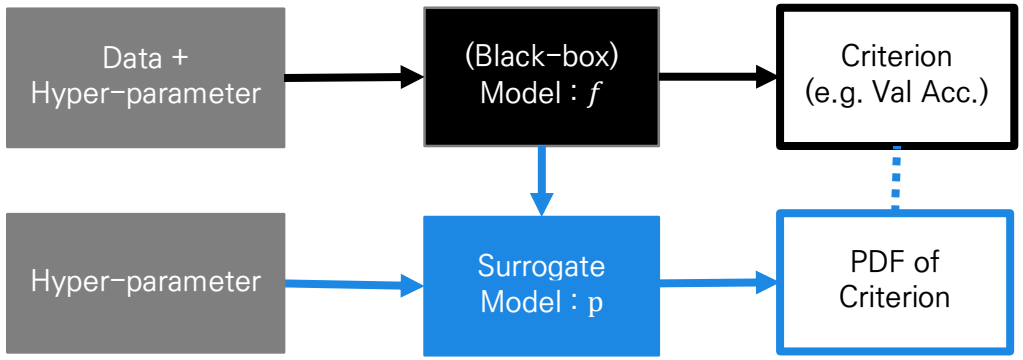
Methods

Searching method

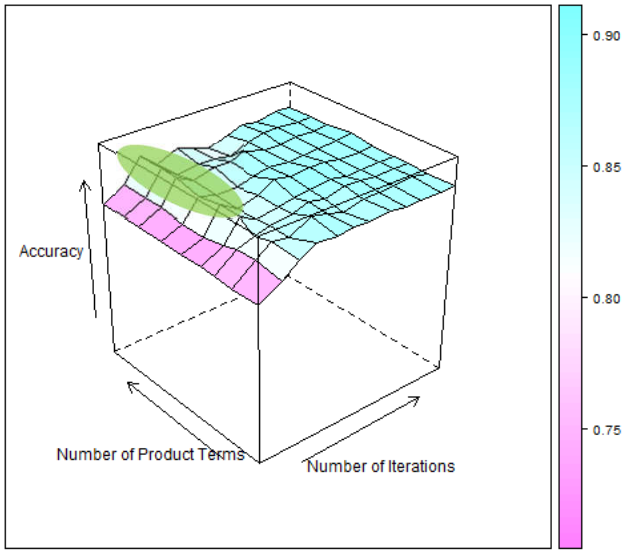
❖ Bayesian Optimization

- 아이디어 : "기존 Trial들의 정보를 이용하여 후속 Trial의 Hyper-parameter를 선택 → 불필요한 자원소모 ↓"
  - ✓ 원래 모델 f (e.g. Neural network)은 매 trial에 드는 비용이 커 직접 모두 반복하기 어려움 (기존 Search 단점)
  - ✓ 핵심은 계산비용이 상대적으로 저렴한 함수 p 로 아래 매핑을 근사하여,
    이를 통해 '좋은 Trial'로 보이는 Hyper-parameter 후보를 확률분포로부터 선정하여 불필요한 자원소모 ↓

$$Accuracy = f(Data, Hyperparameters)$$



$$p(Accuracy \mid Hyperparameters)$$



- http://www.hylap.org/meta_data/adaboost/

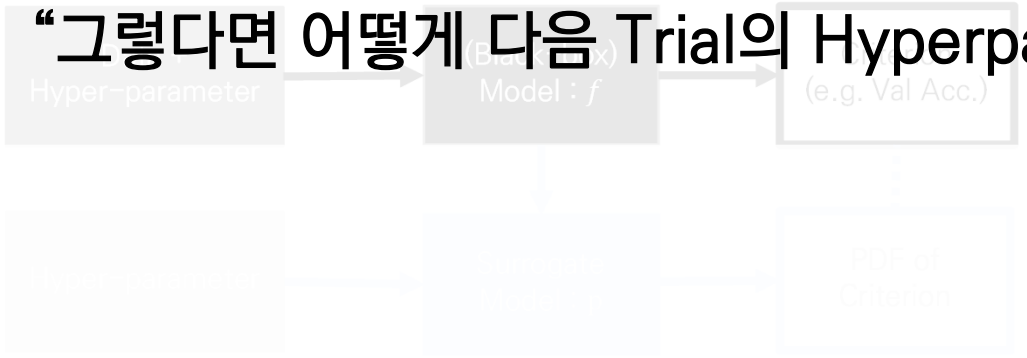
Methods

Searching method

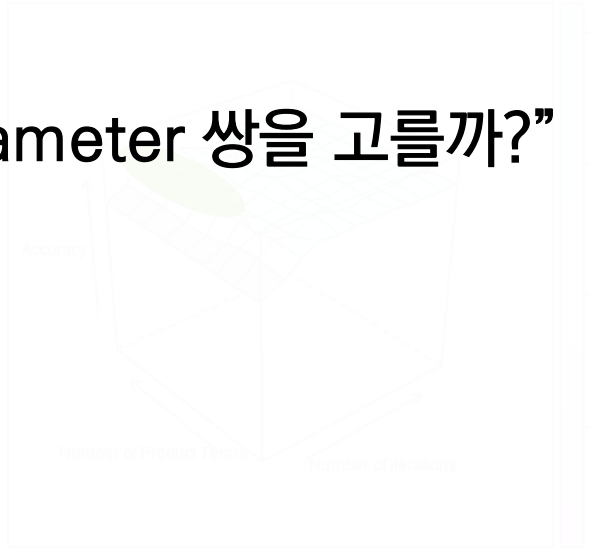
❖ Bayesian Optimization

- 아이디어 : "기존 Trial들의 정보를 이용하여 후속 Trial의 Hyper-parameter를 선택 → 불필요한 자원소모 ↓"
  - ✓ 원래 모델 f (e.g. Neural network)은 매 trial에 드는 비용이 커 직접 모두 반복하기 어려움 (기존 Search 단점)
  - ✓ 핵심은 계산비용이 상대적으로 저렴한 함수 p 로 아래 매핑을 근사하여,
    이를 통해 좋은 Trial로 보이는 Hyper-parameter 후보를 확률분포로부터 선정하여 불필요한 자원소모 ↓

$$Accuracy = f(Data, Hyperparameters)$$



"그렇다면 어떻게 다음 Trial의 Hyperparameter 쌍을 고를까?"



- http://www.hylap.org/meta_data/adaboost/

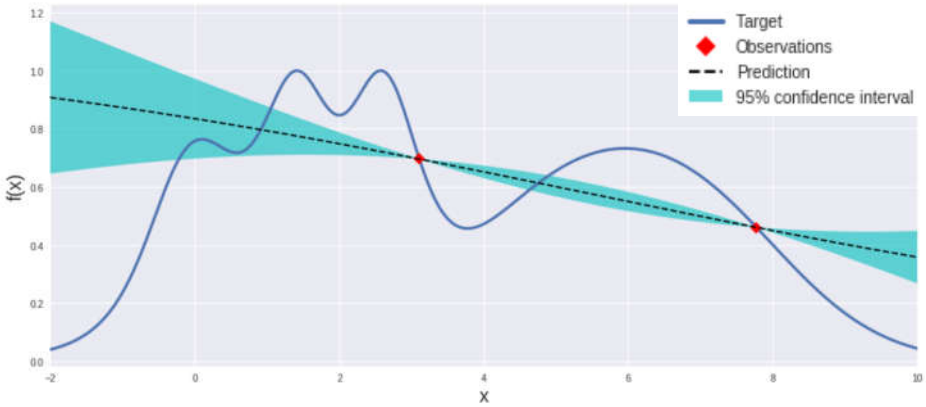
Methods

Preliminaries

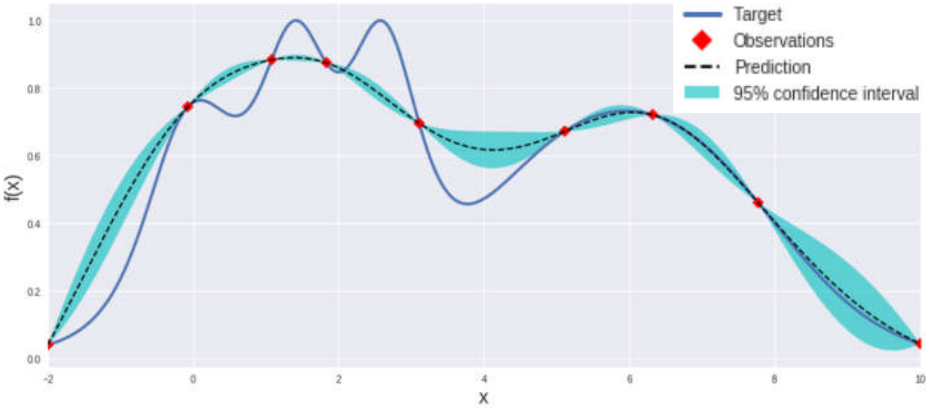
❖ Bayesian Reasoning

- "추가 정보로 기존 확률모델을 점점 업데이트시키면 더 정확한 근사모델을 만들 수 있지 않을까?"
  - 아래 예시 : Bayeisan reasoning 방법론 중 하나인 Gaussian process(---,■)
  - 관측치(◆)가 추가됨에 따라 실제 분포(–)에 Gaussian process 모델이 근접해지고 신뢰구간이 좁아짐을 확인

관측치가 2개 인 경우



관측치가 7개 인 경우



- https://nanonets.com/blog/hyperparameter-optimization/

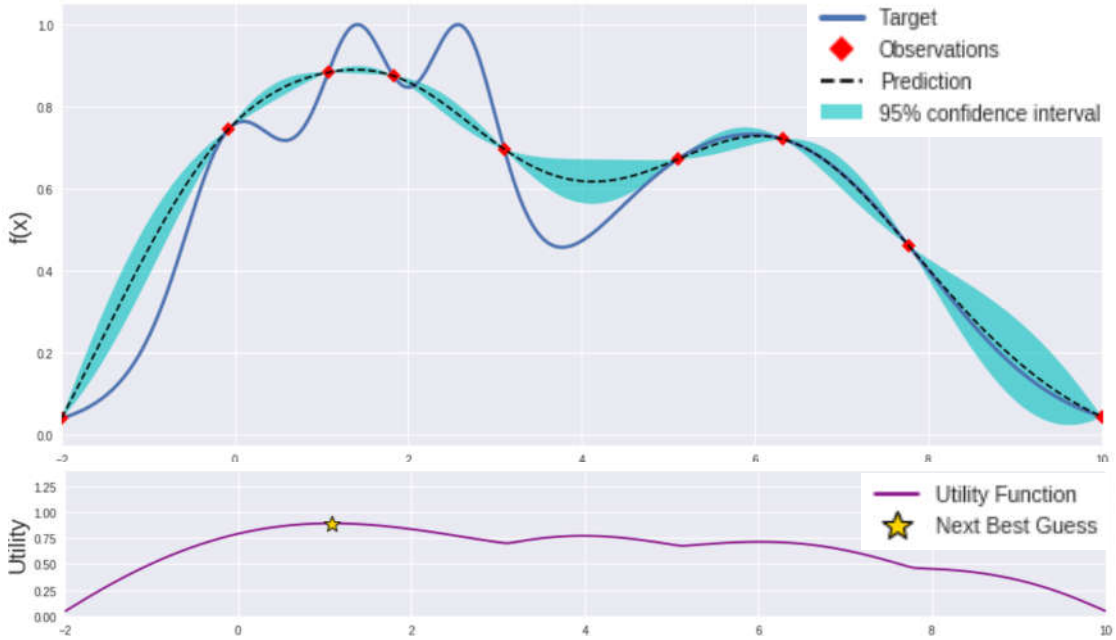
Methods

Searching method

❖ Selection Function

- 다음에 고를 ***hyperparameter*** 쌍 : $\boldsymbol{x_t} = \boldsymbol{argmax}(\boldsymbol{\alpha_t}(\boldsymbol{x}))$ $(y: current\ score, y^*: best\ score, \boldsymbol{x}: hyperparameters)$
- $\alpha_t(x) = ExpectedImprovement(x) = E[\max((y - y^*), 0)] = \int_{-\infty}^{y^*}(y^* - y)p(y|\boldsymbol{x})dy = \int_{-\infty}^{y^*}(y^* - y)N(y|\mu(\boldsymbol{x}), \sigma(\boldsymbol{x}))dy$

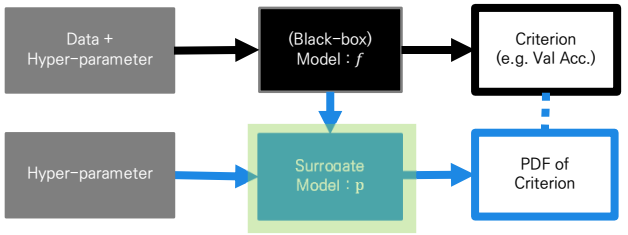
Gaussian Process and Selection function after 2→5→7 steps



- https://nanonets.com/blog/hyperparameter-optimization/
- https://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/slides/lec21.pdf
- https://medium.com/criteo-engineering/hyper-parameter-optimization-algorithms-2fe447525903
- https://towardsdatascience.com/bayesian-optimization-and-hyperparameter-tuning-6a22f14cb9fa

Methods

Searching method

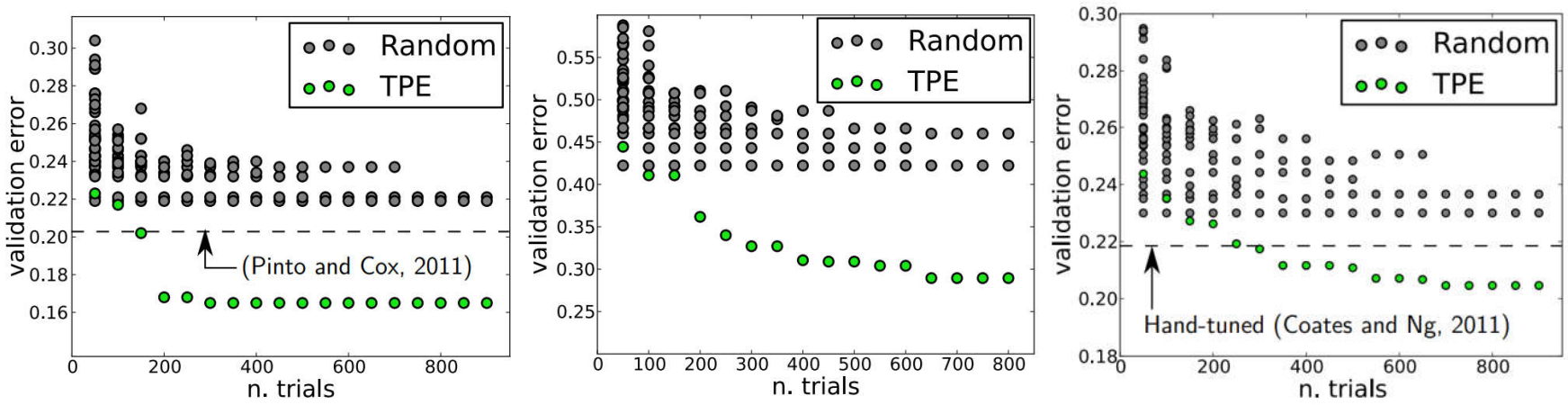


❖ Bayesian Optimization

- (+) "BO(-TPE)가 Random search보다 더 빠르게 global optimum에 도달" [Bergstra et al.(2013)]
- (-) Surrogate 함수가 Gaussian Process 경우 시간복잡도가 $O(n^3)$ + 연속변수 HPO 한해서만 적용가능
  - → Surrogate 함수가 Tree-Parzan Estimation 일 경우는 시간복잡도 $O(nlogn)$ / 모든 종류의 변수에 적용가능.

  → 나아가 TPE는 조건부관계에 있는 hyper-parameter 쌍(e.g. SVM kernel 종류에 따라 해당 kernel만이 갖는 hyperparameter)을 모델 자체적으로 표현가능하여 BO-GP보다 SVM 같은 모델에서의 HPO 성능이 좋음

Bayesian Optimization(TPE) result on three image datasets



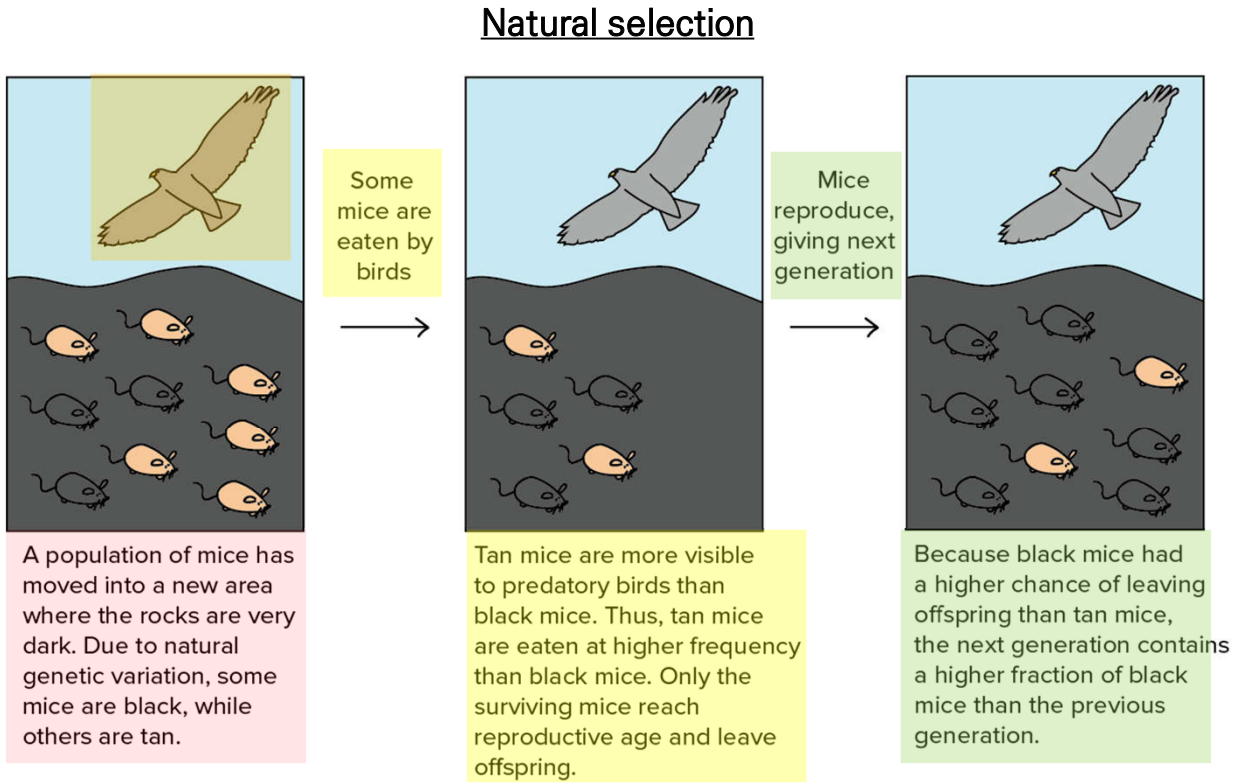
- http://proceedings.mlr.press/v28/bergstra13.pdf

Methods

Searching method

❖ Evolutionary Search

- "생존에 유리한 특성을 갖는 개체들의 특징이 다음 세대에게 넘어간다"
- $(Initialize \rightarrow Fitness\ function \rightarrow selection \rightarrow mutation \rightarrow crossover) \rightarrow (\dots) \rightarrow \cdots$



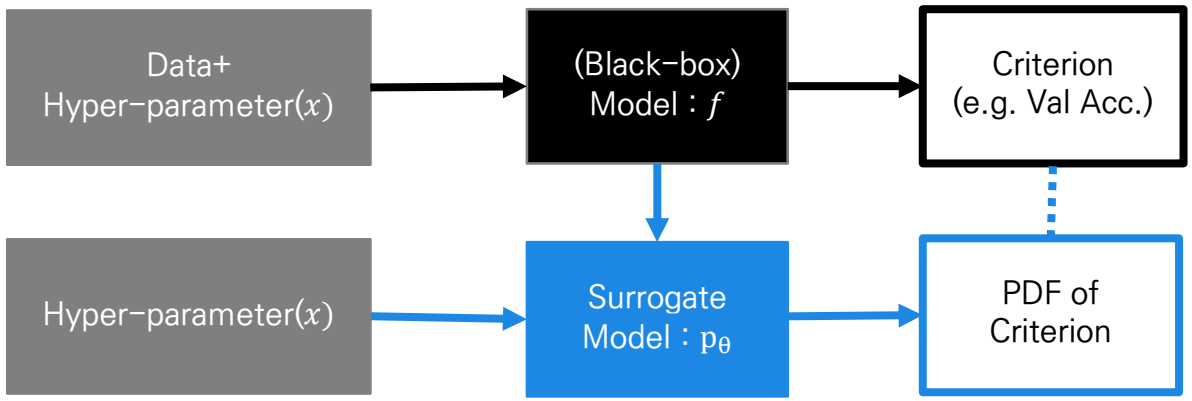
- https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.html

Methods

Searching method

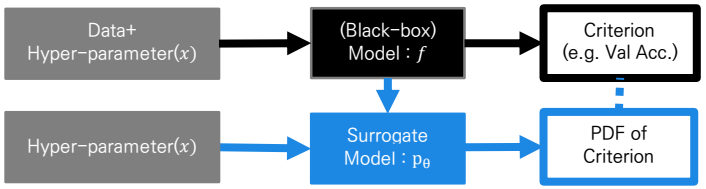
❖ Evolutionary Search

- 원래 함수의 objective function($f(x)$)의 gradient를 직접 구하지 못하므로, 대신 확률분포함수 $p_\theta(x)$로 기술
  1. ( ) $x_i \sim p_\theta(x)$ 로부터 $D = \{(x_i, f(x_i))\}$ 를 샘플링 ($Initialize \rightarrow Fitness\ function \rightarrow selection \rightarrow mutation \rightarrow crossover$)
  2. ( ) D 안의 sample의 적합도(e.g. 정확도)를 평가
  3. ( ) '2'에서 적합도가 가장 좋았던 상위 개체(e.g. 성능이 좋았던 x 집합)들을 이용하여 θ 업데이트
- Bayesian optimization와 비교
  - ✓ (공통점) 둘 다 원래 objective function f를 근사하는 함수p로 하이퍼파라미터 최적화 수행
  - ✓ (차이점) Bayesian optimization은 mutation 개념이 없음 ($Initialize \rightarrow Fitness\ function \rightarrow selection \rightarrow mutation \rightarrow crossover$)



Methods

Searching method



❖ Evolutionary Search (Simple Gaussian Evolution Strategies)

- $\theta = (\boldsymbol{\mu}, \sigma),\ p_\theta(x) \sim N(\boldsymbol{\mu}, \sigma^2 I) = \boldsymbol{\mu} + \sigma N(0, I) \in R^{n \times n},\ x \in R^n\ where\ n = types\ of\ hyperparameters$
- p_θ 는 가우스분포의 중심 $\boldsymbol{\mu}$과 Isotropic한 표준편차행렬 $\sigma N(0, I)$ 에 의해서 단순하게 기술됨

The process of Simple-Gaussian-ES, given $x \in \mathcal{R}^n$: (*Initialize* → *Fitness function* → *selection* → *mutation* → *crossover*)

Initialize

1. Initialize $\theta = \theta^{(0)}$ and the generation counter $t = 0$
2. Generate the offspring population of size Λ by sampling from the Gaussian distribution:

$$D^{(t+1)} = \{x_i^{(t+1)} \mid x_i^{(t+1)} = \mu^{(t)} + \sigma^{(t)} y_i^{(t+1)} \text{ where } y_i^{(t+1)} \sim \mathcal{N}(x|0, \mathbf{I}),\ i = 1, \dots, \Lambda\}$$

Fitness function & selection

3. Select a top subset of λ samples with optimal $f(x_i)$ and this subset is called **elite** set. Without loss of generality, we may consider the first k samples in $D^{(t+1)}$ to belong to the elite group — Let's label them as

$$D_{\text{elite}}^{(t+1)} = \{x_i^{(t+1)} \mid x_i^{(t+1)} \in D^{(t+1)}, i = 1, \dots, \lambda, \lambda \leq \Lambda\}$$

mutation & crossover

4. Then we estimate the new mean and std for the next generation using the elite set:

$$\mu^{(t+1)} = \text{avg}(D_{\text{elite}}^{(t+1)}) = \frac{1}{\lambda} \sum_{i=1}^{\lambda} x_i^{(t+1)}$$

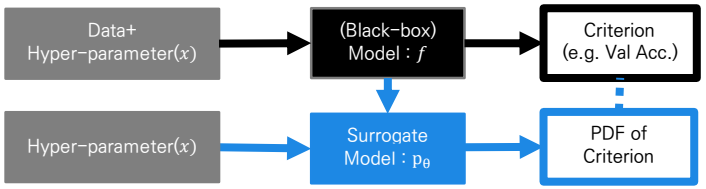
$$\sigma^{(t+1)^2} = \text{var}(D_{\text{elite}}^{(t+1)}) = \frac{1}{\lambda} \sum_{i=1}^{\lambda} (x_i^{(t+1)} - \mu^{(t)})^2$$

5. Repeat steps (2)-(4) until the result is good enough 👌

- https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.html

Methods

Searching method



❖ Evolutionary Search (Simple Gaussian Evolution Strategies)

- $\theta = (\boldsymbol{\mu}, \sigma),\ p_\theta(x) \sim N(\boldsymbol{\mu}, \sigma^2 I) = \boldsymbol{\mu} + \sigma N(0, I) \in R^{n \times n},\ x \in R^n\ where\ n = types\ of\ hyperparameters$
- p_θ 는 가우스분포의 중심 $\boldsymbol{\mu}$과 Isotropic한 표준편차행렬 $\sigma N(0, I)$ 에 의해서 단순하게 기술됨

"p_θ 분산이 isotropic($\sigma N(0, I)$)함으로써 생기는 문제점?"

→ 모든 방향에 대해 분포(σ)가 같으므로 다음 세대에서 탐색(sample)될 하이퍼파라미터 공간에 더 선호되는 방향이 표현되지 못함

→ " 더 찾아야할 곳과 적게 찾아야할 곳의 sampling 정도가 *같게 됨*"

- https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.html

Methods

Searching method

❖ Evolutionary Search (Covariance Matrix Adaptation Evolution Stretegy(CMA-ES))

"그렇다면 특정방향(e.g learning rate)에 대한

hyper-parameter 분포 크기를 각기 다르게 업데이트해보자!

→ Covariance Matrix Adaptation Evolution Stretegy(**CMA-ES**))

" 무엇(1)을 기준으로 무엇(2)을 **A**daptive하게 업데이트?"

→ (1) 과거 trial들의 경로(history) : Evolution path($\boldsymbol{path_\sigma}, \boldsymbol{path_C}$)

→ (2) p_θ분포의 전체적인 크기(step size)인 $\boldsymbol{\sigma}$와

개별 하이퍼파라미터 방향성과 상관관계를 기술하는 C 행렬

- https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.html

Methods

Searching method

❖ Evolutionary Search (Covariance Matrix Adaptation Evolution Stretegy(CMA-ES))

- $\theta = (\boldsymbol{\mu}, \sigma, C),\ p_\theta(x) \sim N(\boldsymbol{\mu}, \sigma^2 C) = \boldsymbol{\mu} + \sigma N(0, C)$ (cf:Vanilla-ES, $\theta = (\boldsymbol{\mu}, \sigma),\ p_\theta(x) \sim N(\boldsymbol{\mu}, \sigma^2 I) = \boldsymbol{\mu} + \sigma N(0, I)$)
- 1. 공분산 행렬(C)를 이용하여 를 이용하여 분포에 있는 표본 간의 pairwise dependency를 표현하며(cf:Vanilla-ES는 단순 isotropic)
- 2. 이때, 과거 Trial의 정보(evolution path (p_σ, p_C))를 기준으로 θ 업데이트 (cf:Vanilla-ES는 바로 전 step만을 고려하는 반면 CMA-ES는 전체 경로 고려)

$(Initialize \rightarrow Fitness\ function \rightarrow selection \rightarrow mutation \rightarrow crossover)$

Algorithm 1 CMA-ES: Covariance Matrix Adaptation Evolution Strategies

Require: $\alpha_\mu,\ \alpha_\sigma,\ \alpha_{cp},\ \alpha_{c1},\ \alpha_{c\lambda}$ ▷ Learning rates
Require: d_σ ▷ Damping factor
Require: $t = 0$ ▷ Generation counter
Require: $\mu^{(0)} \in \mathbb{R}^n, \sigma \in \mathbb{E}_+$ ▷ Inputs: initial mean vectors and step size
Require: $C^{(0)} = I,\ p_\sigma^{(0)} = 0,\ p_c^{(0)} = 0$ ▷ Initialize covariance matrix and evolution paths.
repeat
Sample $x_i^{(t+1)} = \mu^{(t)} +^{(t)} y_i$ where $y_i \sim \mathcal{N}(0, C^{(t)}),\ i = 1, \ldots, \Lambda$
Select top λ samples with the best performance $x_i^{(t+1)}, i = 1, \ldots, \lambda$
$\mu^{(t+1)} \leftarrow \mu^{(t)} + \alpha_\mu \frac{1}{\lambda} \sum_{i=1}^{\lambda} (x_i^{(t+1)} - \mu^{(t)})$
$p_\sigma^{(t+1)} \leftarrow (1 - \alpha_\sigma) p_\sigma^{(t)} + \sqrt{\alpha_\sigma (2 - \alpha_\sigma) \lambda}\ C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$
$\sigma^{(t+1)} \leftarrow \sigma^{(t)} \exp\left(\frac{\alpha_\sigma}{d_\sigma}\left(\frac{\|p_\sigma^{(t+1)}\|}{\mathbb{E}\|\mathcal{N}(0,I)\|} - 1\right)\right)$
$p_c^{(t+1)} \leftarrow (1 - \alpha_{cp}) p_c^{(t)} + \sqrt{\alpha_{cp} (2 - \alpha_{cp}) \lambda}\ \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$
$C^{(t+1)} \leftarrow (1 - \alpha_{c1}) C^{(t)} + \alpha_{c1}\ p_c^{(t+1)} p_c^{(t+1)^\top}$
$C^{(t+1)} \leftarrow (1 - \alpha_{c\lambda} - \alpha_{c1}) C^{(t)} + \alpha_{c1}\ p_c^{(t+1)} p_c^{(t+1)^\top} + \alpha_{c\lambda}\ \frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} y_i^{(t+1)^\top}$
$t \leftarrow t + 1$
until hit stopping criteria
return $\mu^{(t)},\ \sigma^{(t)},\ C^{(t)}$

Symbol	Meaning
$x_i^{(t)} \in \mathbb{R}^n$	the i-th samples at the generation (t)
$y_i^{(t)} \in \mathbb{R}^n$	$x_i^{(t)} = \mu^{(t-1)} + \sigma^{(t-1)} y_i^{(t)}$
$\mu^{(t)}$	mean of the generation (t)
$\sigma^{(t)}$	step size
$C^{(t)}$	covariance matrix
$B^{(t)}$	a matrix of C's eigenvectors as row vectors
$D^{(t)}$	a diagonal matrix with C's eigenvalues on the diagnose.
$p_\sigma^{(t)}$	evaluation path for σ at the generation (t)
$p_c^{(t)}$	evaluation path for C at the generation (t)
α_μ	learning rate for μ's update
α_σ	learning rate for p_σ
d_σ	damping factor for σ's update
α_{cp}	learning rate for p_c
$\alpha_{c\lambda}$	learning rate for C's rank-min(λ, n) update
α_{c1}	learning rate for C's rank-1 update

- https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.htm

Methods

Searching method

❖ Evolutionary Search (Covariance Matrix Adaptation Evolution Stretegy(CMA-ES))

- $\theta = (\boldsymbol{\mu}, \sigma, C)$, $p_\theta(x) \sim N(\boldsymbol{\mu}, \sigma^2 C) = \boldsymbol{\mu} + \sigma N(0, C)$ (cf:Vanilla-ES, $\theta = (\boldsymbol{\mu}, \sigma)$, $p_\theta(x) \sim N(\boldsymbol{\mu}, \sigma^2 I) = \boldsymbol{\mu} + \sigma N(0, I)$)
- 1. 공분산 행렬(C)를 이용하여 를 이용하여 분포에 있는 표본 간의 pairwise dependency를 표현하며 (cf:Vanilla-ES는 단순하게 isotropic)
- 2. 이때, 과거 Trial의 정보(evolution path (p_σ, p_C))를 기준으로 θ 업데이트 (cf:Vanilla-ES는 바로 전 step만을 고려하는 반면 CMA-ES는 전체 경로 고려)

Algorithm 1 CMA-ES: Covariance Matrix Adaptation Evolution Strategies

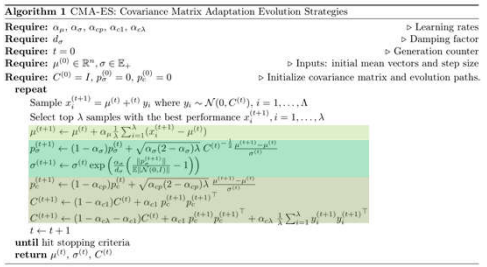
Require: α_μ, α_σ, α_{cp}, α_{c1}, $\alpha_{c\lambda}$ ▷ Learning rates
Require: d_σ ▷ Damping factor
Require: $t = 0$ ▷ Generation counter
Require: $\mu^{(0)} \in \mathbb{R}^n, \sigma \in \mathbb{E}_+$ ▷ Inputs: initial mean vectors and step size
Require: $C^{(0)} = I$, $p_\sigma^{(0)} = 0$, $p_c^{(0)} = 0$ ▷ Initialize covariance matrix and evolution paths.
repeat
Sample $x_i^{(t+1)} = \mu^{(t)} +^{(t)} y_i$ where $y_i \sim \mathcal{N}(0, C^{(t)})$, $i = 1, \ldots, \Lambda$
Select top λ samples with the best performance $x_i^{(t+1)}, i = 1, \ldots, \lambda$
$\mu^{(t+1)} \leftarrow \mu^{(t)} + \alpha_\mu \frac{1}{\lambda} \sum_{i=1}^{\lambda} (x_i^{(t+1)} - \mu^{(t)})$
$p_\sigma^{(t+1)} \leftarrow (1 - \alpha_\sigma) p_\sigma^{(t)} + \sqrt{\alpha_\sigma (2 - \alpha_\sigma) \lambda}\, C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$
$\sigma^{(t+1)} \leftarrow \sigma^{(t)} \exp\left(\frac{\alpha_\sigma}{d_\sigma}\left(\frac{\|p_\sigma^{(t+1)}\|}{\mathbb{E}\|\mathcal{N}(0,I)\|} - 1\right)\right)$
$p_c^{(t+1)} \leftarrow (1 - \alpha_{cp}) p_c^{(t)} + \sqrt{\alpha_{cp} (2 - \alpha_{cp}) \lambda}\, \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$
$C^{(t+1)} \leftarrow (1 - \alpha_{c1}) C^{(t)} + \alpha_{c1}\, p_c^{(t+1)} p_c^{(t+1)^\top}$
$C^{(t+1)} \leftarrow (1 - \alpha_{c\lambda} - \alpha_{c1}) C^{(t)} + \alpha_{c1}\, p_c^{(t+1)} p_c^{(t+1)^\top} + \alpha_{c\lambda} \frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} y_i^{(t+1)^\top}$
$t \leftarrow t + 1$
until hit stopping criteria
return $\mu^{(t)}$, $\sigma^{(t)}$, $C^{(t)}$

Symbol	Meaning
$x_i^{(t)} \in \mathbb{R}^n$	the i-th samples at the generation (t)
$y_i^{(t)} \in \mathbb{R}^n$	$x_i^{(t)} = \mu^{(t-1)} + \sigma^{(t-1)} y_i^{(t)}$
$\mu^{(t)}$	mean of the generation (t)
$\sigma^{(t)}$	step size
$C^{(t)}$	covariance matrix
$B^{(t)}$	a matrix of C's eigenvectors as row vectors
$D^{(t)}$	a diagonal matrix with C's eigenvalues on the diagnose.
$p_\sigma^{(t)}$	evaluation path for σ at the generation (t)
$p_c^{(t)}$	evaluation path for C at the generation (t)
α_μ	learning rate for μ's update
α_σ	learning rate for p_σ
d_σ	damping factor for σ's update
α_{cp}	learning rate for p_c
$\alpha_{c\lambda}$	learning rate for C's rank-min(λ, n) update
α_{c1}	learning rate for C's rank-1 update

- https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.htm

Methods

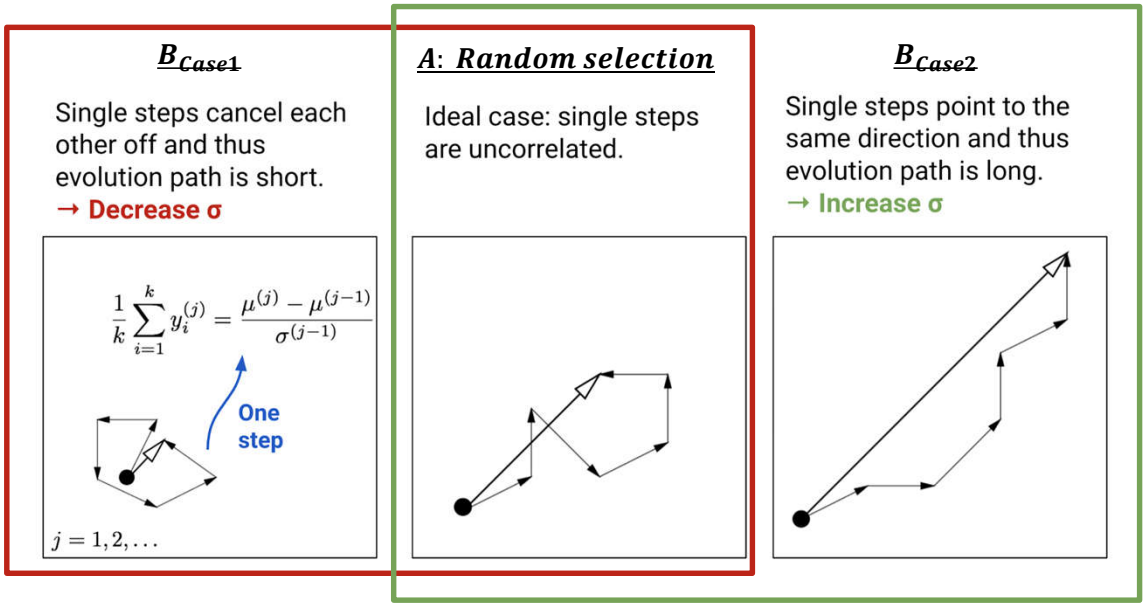
Searching method



❖ Evolutionary Search (Covariance Matrix Adaptation Evolution Stretegy(CMA-ES))

- $\theta = (\boldsymbol{\mu}, \sigma, C),\ p_\theta(x) \sim N(\boldsymbol{\mu}, \sigma^2 C) = \boldsymbol{\mu} + \sigma N(0, C)$
- $p_\sigma^{(t)} = \frac{1}{\lambda}\sum_i^\lambda y_i^{(j)} = \frac{\mu^j - \mu^{(j-1)}}{\sigma^{j-1}}, j = 1, \ldots, t$: 과거 step 벡터들의 총 합
- Key Idea
  - A: 랜덤으로 움직인 이상적인 경로($\sim E(N(0,I))$) *vs* B:과거 실제로 움직였던 step 벡터들의 합(p_σ)
  - Case 1. $|A| > |B|$: "뭔가 중요한걸 찾았다!(=맴돈다) → 많이 움직이지 말고 그 근방에 집중하자 → **Decrease σ**
  - Case 2. $|B| < |A|$: 지금까지 본 결과 이 근방에는 아직 보물(global optimum이 없다!(=멀리 움직임) → 더 움직이자 → **Increase σ**

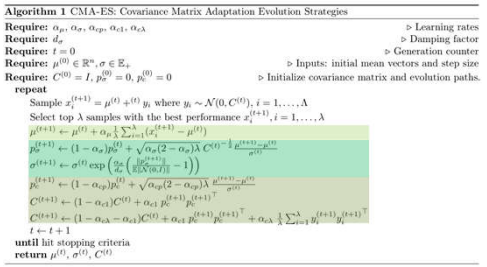
$$p_\sigma^{(t+1)} \leftarrow (1-\alpha_\sigma) p_\sigma^{(t)} + \sqrt{\alpha_\sigma(2-\alpha_\sigma)\lambda}\, C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$$
$$\sigma^{(t+1)} \leftarrow \sigma^{(t)} \exp\left(\frac{\alpha_\sigma}{d_\sigma}\left(\frac{\|p_\sigma^{(t+1)}\|}{\mathbb{E}\|\mathcal{N}(0,I)\|} - 1\right)\right)$$



- https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.htm

Methods

Searching method



❖ Evolutionary Search (Covariance Matrix Adaptation Evolution Stretegy(CMA-ES))

- $\theta = (\boldsymbol{\mu}, \sigma, C),\ p_\theta(x) \sim N(\boldsymbol{\mu}, \sigma^2 C) = \boldsymbol{\mu} + \sigma N(0, C)$
- $p_\sigma^{(t)} = \frac{1}{\lambda}\sum_i^\lambda y_i^{(j)} = \frac{\mu^j - \mu^{(j-1)}}{\sigma^{j-1}}, j = 1, \ldots, t$: 과거 step 벡터들의 총 합
- Techniques
  - 1. p_σ update시 계산의 단순화를 위해 켤레사전분포($\sim N(0, I)$)가 되도록 꼴 유도
  - 2. 최근 생성된 세대 path에 더 큰 가중치를 주자 : $learning\ rate:\ \alpha_\sigma$

$$p_\sigma^{(t+1)} \leftarrow (1 - \alpha_\sigma) p_\sigma^{(t)} + \sqrt{\alpha_\sigma (2 - \alpha_\sigma) \lambda}\, C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$$
$$\sigma^{(t+1)} \leftarrow \sigma^{(t)} \exp\left(\frac{\alpha_\sigma}{d_\sigma}\left(\frac{\|p_\sigma^{(t+1)}\|}{\mathbb{E}\|\mathcal{N}(0,I)\|} - 1\right)\right)$$

Technique 1. Conjugate Prior Form

$$\frac{1}{\lambda}\sum_{i=1}^{\lambda} y_i^{(t+1)} = \frac{1}{\lambda}\frac{\sum_{i=1}^{\lambda} x_i^{(t+1)} - \lambda\mu^{(t)}}{\sigma^{(t)}} = \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$$
$$\frac{1}{\lambda}\sum_{i=1}^{\lambda} y_i^{(t+1)} \sim \frac{1}{\lambda}\mathcal{N}(0, \lambda C^{(t)}) \sim \frac{1}{\sqrt{\lambda}} C^{(t)^{\frac{1}{2}}} \mathcal{N}(0, I)$$
$$\text{Thus } \sqrt{\lambda}\, C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}} \sim \mathcal{N}(0, I)$$

Technique 2. Learning rate+ polyak averaing

$$p_\sigma^{(t+1)} = (1 - \alpha_\sigma) p_\sigma^{(t)} + \sqrt{1 - (1 - \alpha_\sigma)^2}\, \sqrt{\lambda}\, C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$$
$$= (1 - \alpha_\sigma) p_\sigma^{(t)} + \sqrt{c_\sigma (2 - \alpha_\sigma) \lambda}\, C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$$

- https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.htm

Methods

Searching method

❖ Evolutionary Search (Covariance Matrix Adaptation Evolution Stretegy(CMA-ES))

- $\theta = (\boldsymbol{\mu}, \sigma, C)$, $p_\theta(x) \sim N(\boldsymbol{\mu}, \sigma^2 C) = \boldsymbol{\mu} + \sigma N(0, C)$
- 1. 공분산 행렬(C)를 이용하여 를 이용하여 분포에 있는 표본 간의 pairwise dependency를 표현하며 (cf:Vanilla-ES는 단순하게 isotropic)
- 2. 이때, 과거 Trial의 정보(evolution path (p_σ, p_C))를 기준으로 θ 업데이트 (cf:Vanilla-ES는 바로 전 step만을 고려하는 반면 CMA-ES는 전체 경로 고려)

Algorithm 1 CMA-ES: Covariance Matrix Adaptation Evolution Strategies

Require: α_μ, α_σ, α_{cp}, α_{c1}, $\alpha_{c\lambda}$ ▷ Learning rates
Require: d_σ ▷ Damping factor
Require: $t = 0$ ▷ Generation counter
Require: $\mu^{(0)} \in \mathbb{R}^n, \sigma \in \mathbb{E}_+$ ▷ Inputs: initial mean vectors and step size
Require: $C^{(0)} = I$, $p_\sigma^{(0)} = 0$, $p_c^{(0)} = 0$ ▷ Initialize covariance matrix and evolution paths.
repeat
Sample $x_i^{(t+1)} = \mu^{(t)} +^{(t)} y_i$ where $y_i \sim \mathcal{N}(0, C^{(t)})$, $i = 1, \ldots, \Lambda$
Select top λ samples with the best performance $x_i^{(t+1)}, i = 1, \ldots, \lambda$
$\mu^{(t+1)} \leftarrow \mu^{(t)} + \alpha_\mu \frac{1}{\lambda} \sum_{i=1}^{\lambda} (x_i^{(t+1)} - \mu^{(t)})$
$p_\sigma^{(t+1)} \leftarrow (1 - \alpha_\sigma) p_\sigma^{(t)} + \sqrt{\alpha_\sigma (2 - \alpha_\sigma) \lambda}\, C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$
$\sigma^{(t+1)} \leftarrow \sigma^{(t)} \exp\left(\frac{\alpha_\sigma}{d_\sigma}\left(\frac{\|p_\sigma^{(t+1)}\|}{\mathbb{E}\|\mathcal{N}(0,I)\|} - 1\right)\right)$
$p_c^{(t+1)} \leftarrow (1 - \alpha_{cp}) p_c^{(t)} + \sqrt{\alpha_{cp} (2 - \alpha_{cp}) \lambda}\, \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$
$C^{(t+1)} \leftarrow (1 - \alpha_{c1}) C^{(t)} + \alpha_{c1}\, p_c^{(t+1)} p_c^{(t+1)^\top}$
$C^{(t+1)} \leftarrow (1 - \alpha_{c\lambda} - \alpha_{c1}) C^{(t)} + \alpha_{c1}\, p_c^{(t+1)} p_c^{(t+1)^\top} + \alpha_{c\lambda} \frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} y_i^{(t+1)^\top}$
$t \leftarrow t + 1$
until hit stopping criteria
return $\mu^{(t)}$, $\sigma^{(t)}$, $C^{(t)}$

$$\frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} = \frac{1}{\lambda} \frac{\sum_{i=1}^{\lambda} x_i^{(t+1)} - \lambda \mu^{(t)}}{\sigma^{(t)}} = \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$$

$$\frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} \sim \frac{1}{\lambda} \mathcal{N}(0, \lambda C^{(t)}) \sim \frac{1}{\sqrt{\lambda}} C^{(t)^{\frac{1}{2}}} \mathcal{N}(0, I)$$

Thus $\sqrt{\lambda}\, C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}} \sim \mathcal{N}(0, I)$

$$\sqrt{k} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}} \sim \mathcal{N}(0, C)$$

- https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.htm

Methods

Searching method

❖ Evolutionary Search (Covariance Matrix Adaptation Evolution Stretegy(CMA-ES))

- $\theta = (\boldsymbol{\mu}, \sigma, C),\ p_\theta(x) \sim N(\boldsymbol{\mu}, \sigma^2 C) = \boldsymbol{\mu} + \sigma N(0, C)$
- 1. 공분산 행렬(C)를 이용하여 를 이용하여 분포에 있는 표본 간의 pairwise dependency를 표현하며 (cf:Vanilla-ES는 단순하게 isotropic)
- 2. 이때, 과거 Trial의 정보(evolution path (p_σ, p_C))를 기준으로 θ 업데이트 (cf:Vanilla-ES는 바로 전 step만을 고려하는 반면 CMA-ES는 전체 경로 고려)

Algorithm 1 CMA-ES: Covariance Matrix Adaptation Evolution Strategies

Require: $\alpha_\mu,\ \alpha_\sigma,\ \alpha_{cp},\ \alpha_{c1},\ \alpha_{c\lambda}$ ▷ Learning rates

Require: d_σ ▷ Damping factor

Require: $t = 0$ ▷ Generation counter

Require: $\mu^{(0)} \in \mathbb{R}^n, \sigma \in \mathbb{E}_+$ ▷ Inputs: initial mean vectors and step size

Require: $C^{(0)} = I,\ p_\sigma^{(0)} = 0,\ p_c^{(0)} = 0$ ▷ Initialize covariance matrix and evolution paths.

repeat

Sample $x_i^{(t+1)} = \mu^{(t)} +^{(t)} y_i$ where $y_i \sim \mathcal{N}(0, C^{(t)}),\ i = 1, \ldots, \Lambda$

Select top λ samples with the best performance $x_i^{(t+1)}, i = 1, \ldots, \lambda$

$\mu^{(t+1)} \leftarrow \mu^{(t)} + \alpha_\mu \frac{1}{\lambda} \sum_{i=1}^{\lambda} (x_i^{(t+1)} - \mu^{(t)})$

$p_\sigma^{(t+1)} \leftarrow (1 - \alpha_\sigma) p_\sigma^{(t)} + \sqrt{\alpha_\sigma (2 - \alpha_\sigma) \lambda}\ C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$

$\sigma^{(t+1)} \leftarrow \sigma^{(t)} \exp\left(\frac{\alpha_\sigma}{d_\sigma}\left(\frac{\|p_\sigma^{(t+1)}\|}{\mathbb{E}\|\mathcal{N}(0,I)\|} - 1\right)\right)$

$p_c^{(t+1)} \leftarrow (1 - \alpha_{cp}) p_c^{(t)} + \sqrt{\alpha_{cp} (2 - \alpha_{cp}) \lambda}\ \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$

$C^{(t+1)} \leftarrow (1 - \alpha_{c1}) C^{(t)} + \alpha_{c1}\ p_c^{(t+1)} p_c^{(t+1)^\top}$

$C^{(t+1)} \leftarrow (1 - \alpha_{c\lambda} - \alpha_{c1}) C^{(t)} + \alpha_{c1}\ p_c^{(t+1)} p_c^{(t+1)^\top} + \alpha_{c\lambda}\ \frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} y_i^{(t+1)^\top}$

$t \leftarrow t + 1$

until hit stopping criteria

return $\mu^{(t)},\ \sigma^{(t)},\ C^{(t)}$

Rank-one update (전체 경향:'sign info')

Rank-min update (지엽적 정보)

- https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.htm

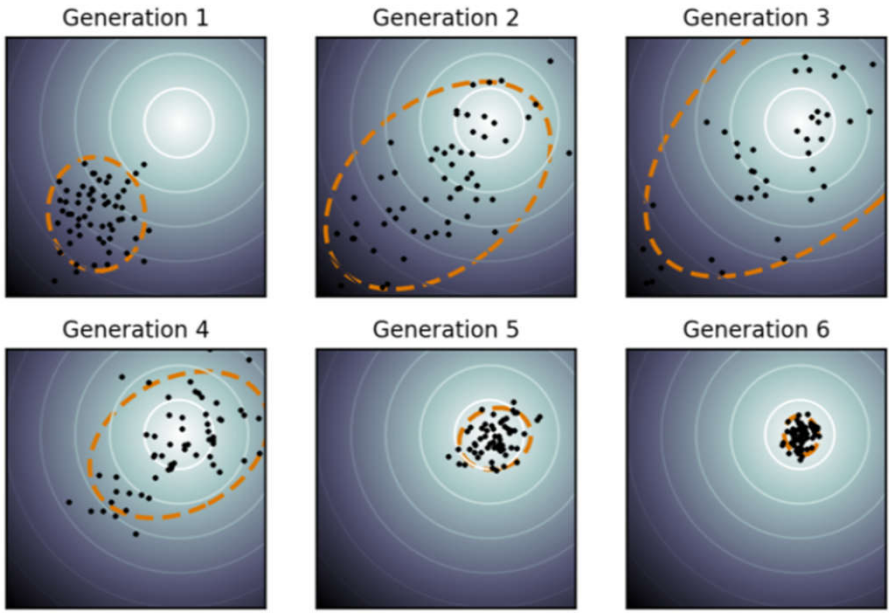
Methods

Searching method

❖ Evolutionary Search (Covariance Matrix Adaptation Evolution Stretegy(CMA-ES))

- 1. Covariance matrix를 사용하므로 타원형의 population 분포 생성 가능 (특정 방향으로 더 탐색 가능)
- 2. Vanilla-ES보다 σ^t(sampling 분포 크기)가 훨씬 능동적으로 업데이트 (exploration/exploitation 능력 ↑)

2D 최적화 문제에서 CMA-ES step별 변화 모습



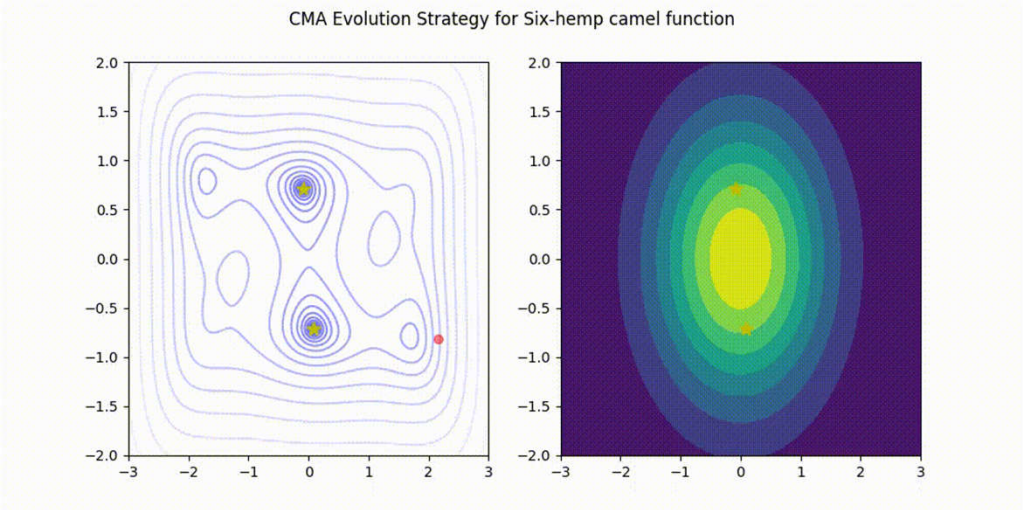
- https://en.wikipedia.org/wiki/CMA-ES

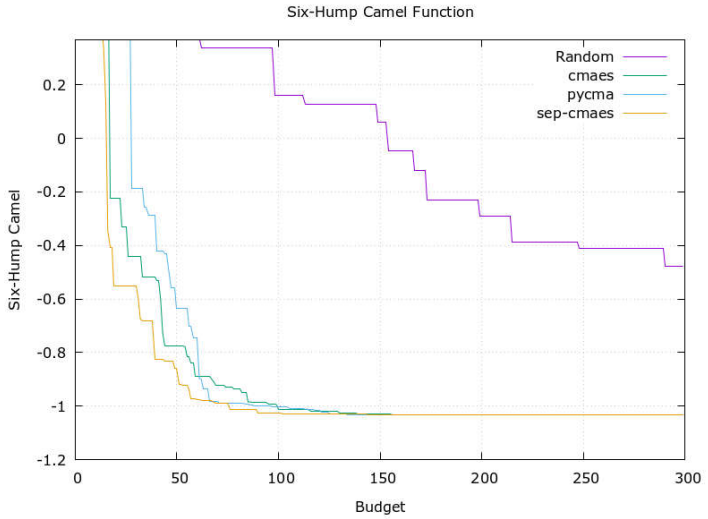
Methods

Searching method

❖ Evolutionary Search (Covariance Matrix Adaptation Evolution Stretegy(CMA-ES))

- (+) Non-convex한 search-space에서 타 방법론들보다 우수한 성능 (e.g Six-hump Camel Function)
- (+) SOTA(Best performance out of over 100 black-box optimization methods for various benchmark problems)[1]
- (-) 시간복잡도 : O(n^2) / CMA-ES의 경우 범주형 변수 HPO 미지원(하지만 라이브러리 내에서 해당 변수에 대해서는 자동으로 TPE-Sampler 사용)





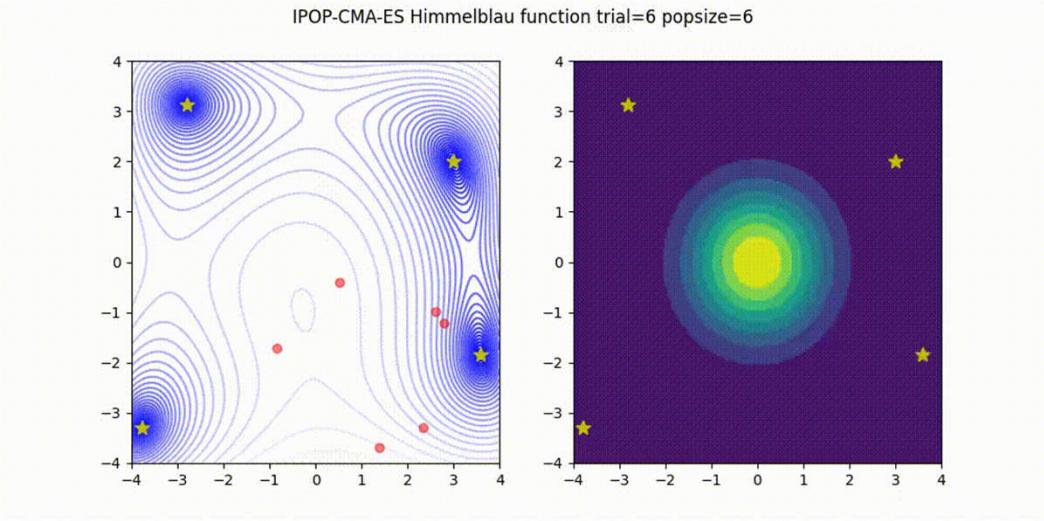
- [1] I. Loshchilov, M. Schoenauer, and M. Sebag. Bi-population CMA-ES Algorithms with Surrogate Models and Line Searches, GECCO Workshop, 2013.
- https://medium.com/optuna/introduction-to-cma-es-sampler-ee68194c8f88

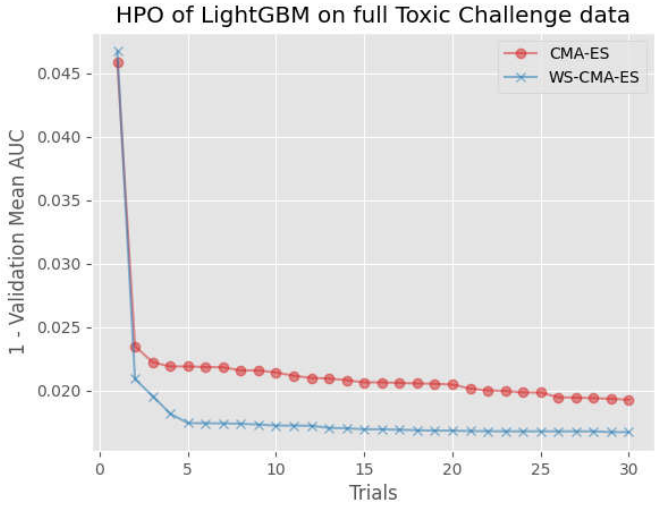
Methods

Searching method

❖ Evolutionary Search (Covariance Matrix Adaptation Evolution Stretegy(CMA-ES))

- IPOP-CMA-ES(Increasing POPulation size) : Local minima에 빠졌을 때 population size ↑
- WS-CMA-ES(Warm Starting) : 비슷한 HPO task의 최적화 결과를 사용하여, HPO 성능 ↑ …[1]
  - 특히, data distribution shift로 인해 HPO를 주기적으로 해줘야 하는 실제 업무현장에서 기존 HPO 결과를 다가오는 HPO의 prior knowledge로 사용가능



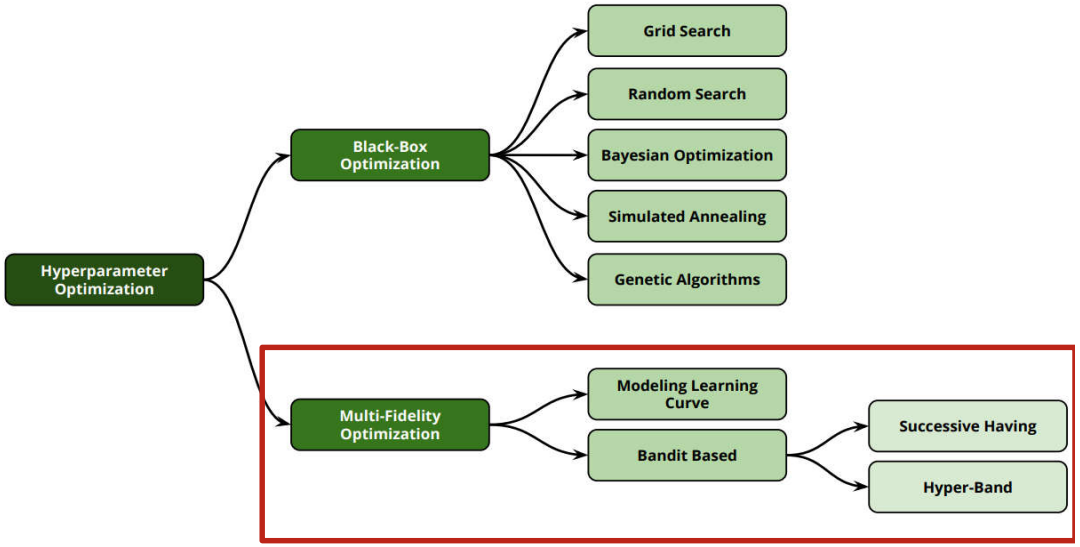


- [1] M. Nomura, S. Watanabe, Y. Akimoto, Y. Ozaki, M. Onishi. Warm Starting CMA-ES for Hyperparameter Optimization, AAAI, 2021.
- https://medium.com/optuna/introduction-to-cma-es-sampler-ee68194c8f88

Scheduling Method

Methods

Scheduling method



[(Sequential) Searching method 단독 사용의 한계]

모든 Trial은 순차적으로 끝까지 진행되므로 속도 상한이 명백

→ 더 빠르게 하기 위해선 시간/계산 자원(budget)을 고려해야함!

(이때 여러 configuration을 한번에 여러 개 검토하기 때문에 Multi-Fidelity Optimization 이라고도 부름

- https://arxiv.org/pdf/1906.02287.pdf

Methods

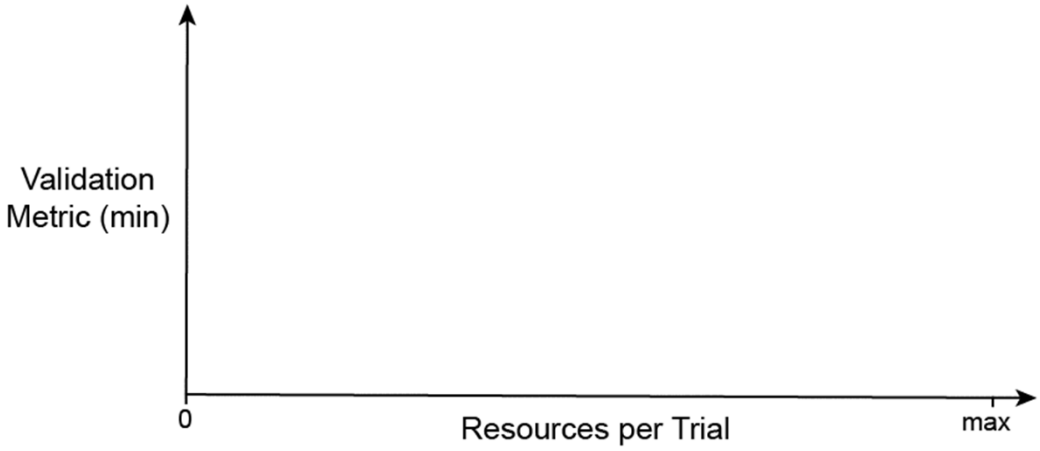
Scheduling method

❖ Successive Halving Algorithm (SHA)

- 가용 가능한 자원($budget = B$)일때, 초기에는 모든 hyper-parameter 쌍(n개)에 대해 $b = B/n$ 만큼 자원 배분
- $\min\_resource * \eta^{i-1}$ 마다 전체 모델 중 저조한 $1/\eta$ 개를 중단하고, 해당 자원을 나머지 모델에게 배분하여,
  나머지 모델은 $b \leftarrow \eta * b$ 의 자원으로 배분됨. (아래 예시: $n = 27, \eta = 3, \min\_resource = 1, i = 1,2,3,4$)
- (+) Random search보다 효율적 (모든 Trial을 끝까지 안 돌려도 되므로!)
- (-) (선택되는 configuration 수) *vs* (배분되는 budget) 중 어느곳에 더 비중을 둬야하는가? → Sol : HyperBand / ASHA
  - [많고 얇게 vs 적고 깊게] = [Configuration 종류 多, budget少] VS [Configuration 종류少, budget多]
- (-) 매 Rung 생성시, 즉 Top $1/\eta$ 을 고를때 항상 기다려야함! → Bottleneck → Sol: ASHA

$lr \in (0.1, 0.01, 0.001)$
$momentum \in (0.85, 0.9, 0.95)$
$\text{WeightDecay} \in (0.01, 0.001, 0.0001)$

Rung i	Configurations Remaining	Epochs per Configuration
Rung 1	27	1
Rung 2	9	3
Rung 3	3	9
Rung 4	1	27



- https://blog.ml.cmu.edu/2018/12/12/massively-parallel-hyperparameter-optimization/
- https://wood-b.github.io/post/a-novices-guide-to-hyperparameter-optimization-at-scale/sha.gif

Methods

Scheduling method

❖ HyperBand

- SHA 문제: [(a)많고 얇게 vs (b)적고 깊게] → "초기 자원 배분(config)에 따라서 최종 Best configuration이 달라지는 문제"

tions. However, for a fixed B, it is not clear a priori whether we should (a) consider many configurations (large n) with a small average training time; or (b) consider a small number of configurations (small n) with longer average training times.

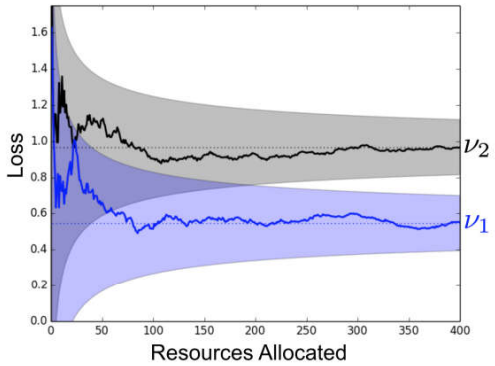


Figure 2: The validation loss as a function of total resources allocated for two configurations is shown. ν_1 and ν_2 represent the terminal validation losses at convergence. The shaded areas bound the maximum distance of the intermediate losses from the terminal validation loss and monotonically decrease with the resource.

- https://arxiv.org/abs/1603.06560

Methods

Scheduling method

❖ HyperBand

- SHA 문제: [(a)많고 얇게 vs (b)적고 깊게] → "초기 자원 배분에 따라서 최종 Best configuration이 달라지는 문제"

  tions. However, for a fixed B, it is not clear a priori whether we should (a) consider many configurations (large n) with a small average training time; or (b) consider a small number of configurations (small n) with longer average training times.

- Solution : 여러 초기 조합(n_i, r_i) 에 대해 다 돌려보자! (아래 예시 : brackets $s = 4,3,\ldots,1,0$)

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

The values of n_i and r_i for the brackets of HYPERBAND corresponding to various values of s, when $R = 81$ and $\eta = 3$.

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.*

- https://arxiv.org/abs/1603.06560
- https://arxiv.org/pdf/1810.05934.pdf

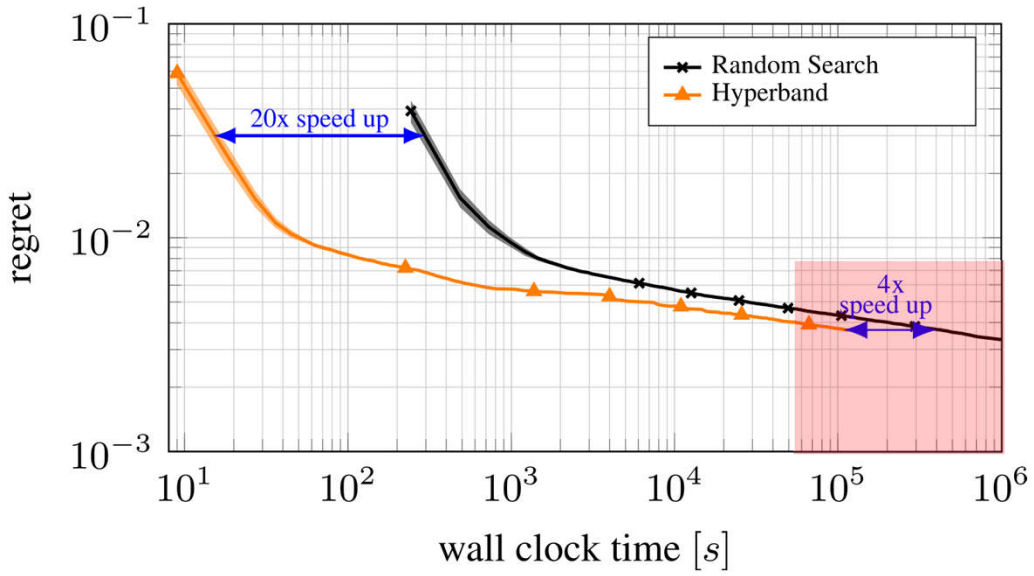
Methods

Scheduling method

❖ HyperBand

- Budget이 작을 때는 큰 폭으로 RS를 앞서지만, budget이 커짐에 따라 그 폭이 줄어든다
- 의문 : “큰 Budget에서도 좋은 효율성을 보일 수는 없을까?”

HyperBand vs Random Search



- http://proceedings.mlr.press/v80/falkner18a/falkner18a.pdf

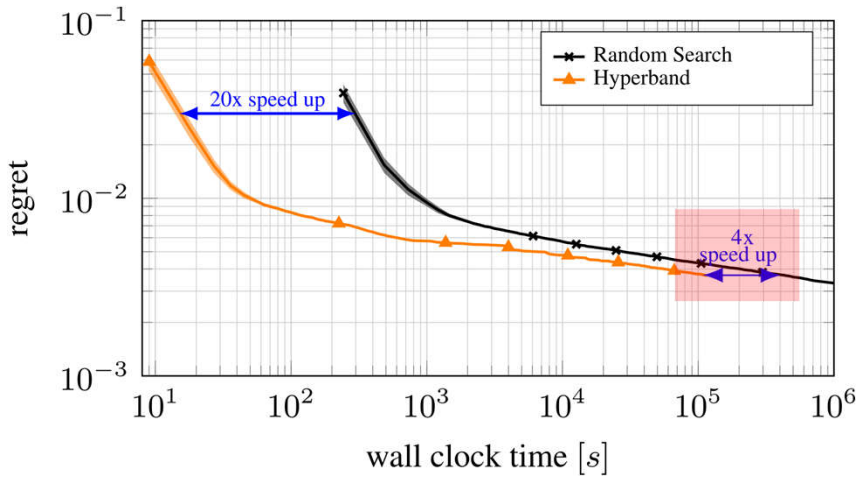
Methods

Scheduling method

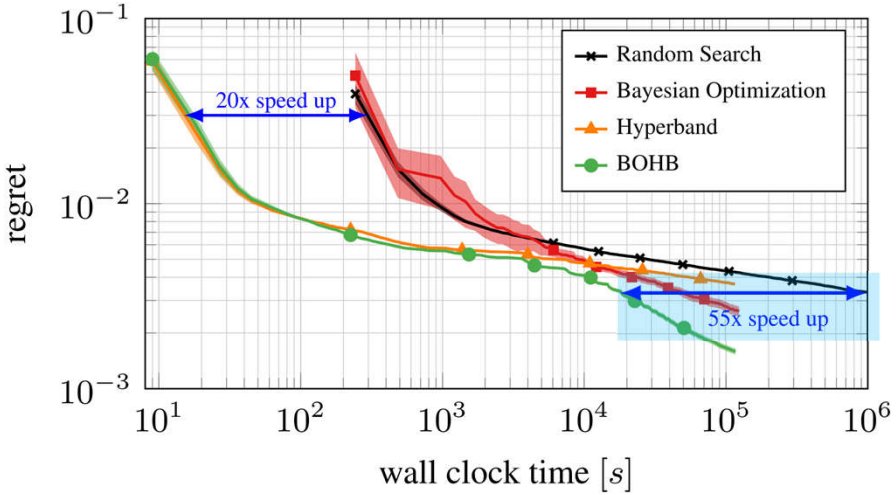
❖ Bayesian Optimization with HyperBand (BOHB)

- Vanilla HyperBand(RandomSearch+HyperBand)의 성능한계를 Bayesian Optimization으로 극복
- Budget이 클 때 Random Search와의 격차 : 4x → 55x

HyperBand vs Random Search



BOHB vs HyperBand vs BO vs RS



- http://proceedings.mlr.press/v80/falkner18a/falkner18a.pdf

Methods

Scheduling method

❖ Asynchronous Successive Halving Alogrithm (ASHA)

- SHA 문제 : 매 Rung 생성시, 즉 Top $1/\eta$ 을 고를(promote) 때 worker 여러 개를 사용해도 항상 다 끝날때까지 기다려야함!
- Sol. 여러 worker를 사용한다고 할때…
  1. 임의의 worker의 작업이 끝나면 이 free worker는 <u>기다리지 않고</u> 아래와 같이 작업에 다시 들어감
  2. 먼저 promotable config가 있는지 확인 (Top $1/\eta$ 이 아직 안 골라진 rung에 달라붙어서 일함)
  3. 없다면 bottom rung을 생성 (새로운 configuration set에 대해 exploration 수행)
  4. '1-3'을 desired condition 만족할때까지 계속 일함 (e.g. 총 Trial 횟수)

Algorithm 2 Asynchronous Successive Halving (ASHA)

input minimum resource r, maximum resource R, reduction factor η, minimum early-stopping rate s

function ASHA()
  repeat
    for for each free worker **do**
      $(\theta, k) =$ get_job()
      run_then_return_val_loss$(\theta, r\eta^{s+k})$
    end for
    for completed job (θ, k) with loss l **do**
      Update configuration θ in rung k with loss l.
    end for
  until desired
end function

function get_job()
  // Check if there is a promotable config.
  for $k = \lfloor \log_\eta(R/r) \rfloor - s - 1, \ldots, 1, 0$ **do**
    candidates $=$ top_k(rung k, |rung k|$/\eta$)
    promotable $= \{t \in$ candidates : t not promoted$\}$
    if |promotable| > 0 **then**
      return promotable[0], $k+1$
    end if
  // If not, grow bottom rung.
  Draw random configuration θ.
  return $\theta, 0$
  end for
end function

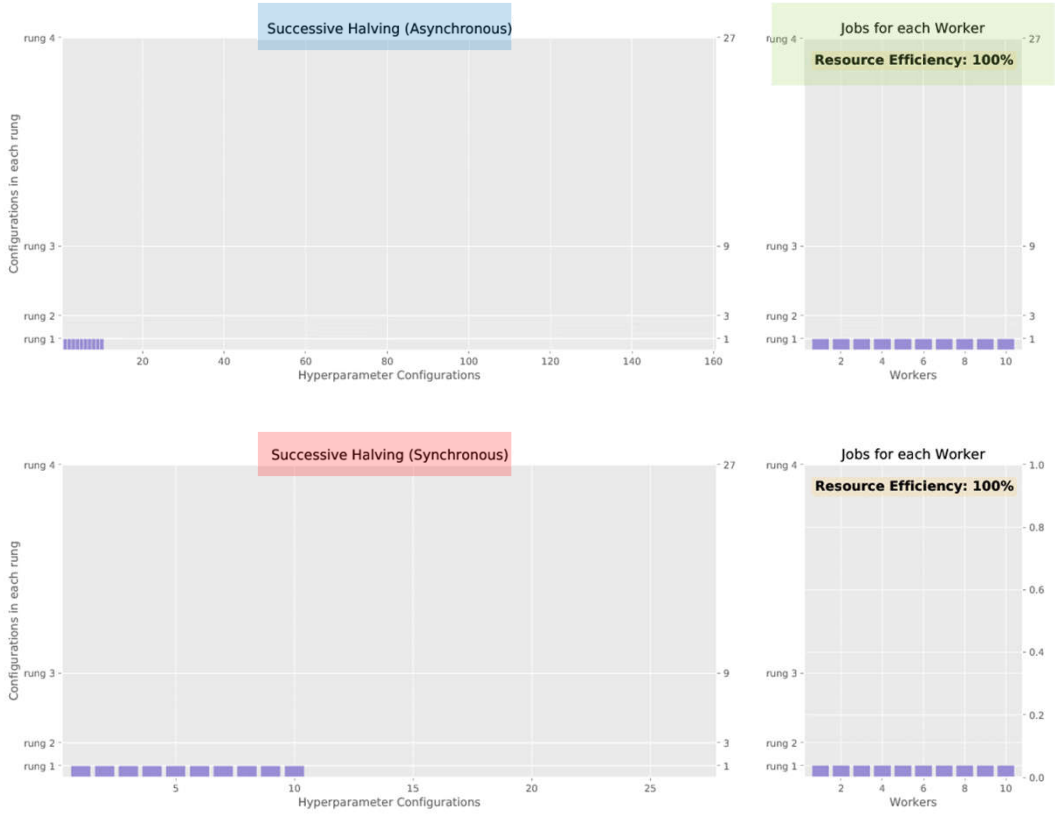
- https://blog.ml.cmu.edu/2018/12/12/massively-parallel-hyperparameter-optimization/

Methods

Scheduling method

❖ Asynchronous Successive Halving Alogrithm (ASHA)

- ASHA(上)의 경우 현재 맡고 있던 작업이 끝나면 기다리지 않고, 다른 일(1.promote config / 2. grow bottom rung)을 수행하므로써, Synchronous한 병렬화(下)보다 같은 시간 안에 더 높은 효율성(Resource Efficiency)을 보임



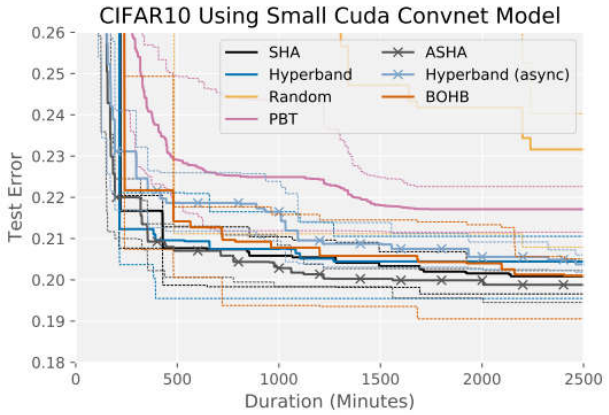
- https://blog.ml.cmu.edu/2018/12/12/massively-parallel-hyperparameter-optimization/

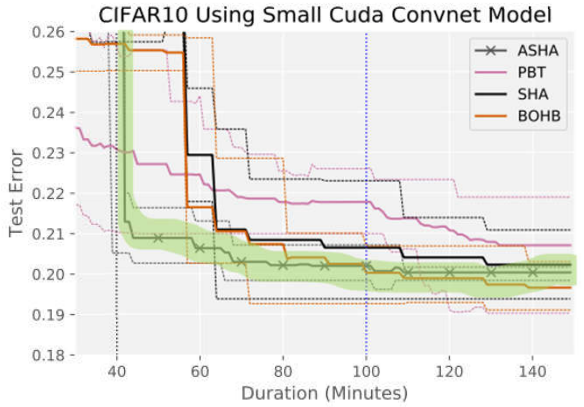
Methods

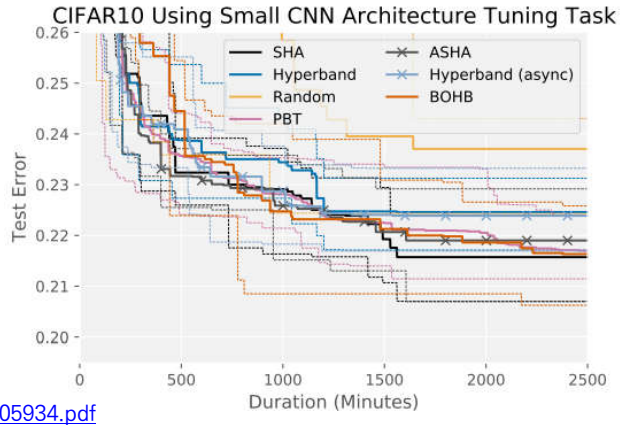
Scheduling method

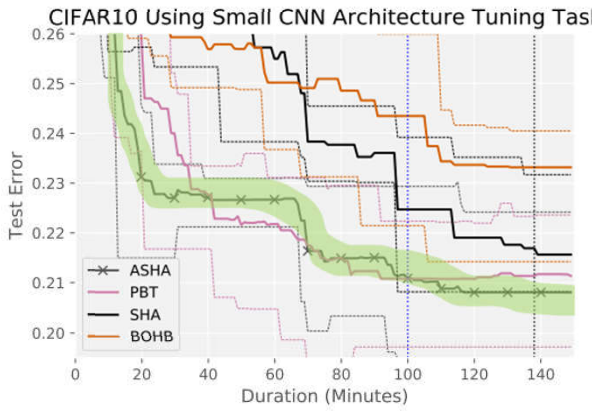
❖ Asynchronous Successive Halving Alogrithm (ASHA)

- (left) Single Worker : SHA 계열 알고리즘 사이에 큰 차이 없음
- (right) Multi-Worker : ASHA ≥ PBT ≥ BOHB ≥ SHA









- https://arxiv.org/pdf/1810.05934.pdf

Methods

Scheduling method

❖ Population-Based Training (PBT)

[Preliminaries]

1. Model hyper-parameters

→ 모델 자체의 **정적인 형태**를 결정하는 하이퍼파라미터

→ E.g.) 은닉층의 개수, 은닉층의 차원,…

2. Algorithm hyper-parameters(optimizer parameters)

→ 모델의 **행동 방식**을 결정하는 하이퍼파라미터

→ E.g.) 학습률, 드롭아웃률,…

- https://deepmind.com/blog/article/population-based-training-neural-networks

Methods

Scheduling method

❖ Population-Based Training (PBT)

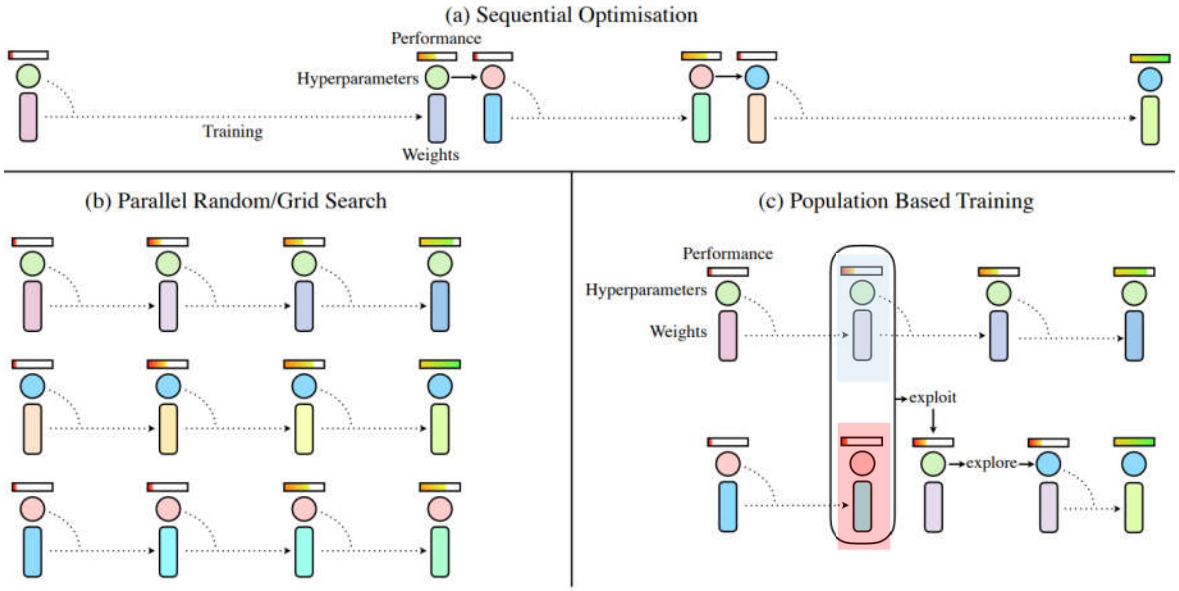
- PBT = "Evolutionary Strategy + Parallel Search" (단, algorithm hyper-parameter만 HPO 가능)
- (b)와 동일 방식으로 초기화하지만, 주기적으로 exploit/explore 시행
  - ➢ If (내 population 성능 < 다른 population 성능):

    다른 population의 hyper-parameter로 update(=exploit);

    학습 재개 전 해당 hyper-parameter를 mutate(=explore);

    학습 재개;

    Else : pass



- https://arxiv.org/pdf/1711.09846.pdf

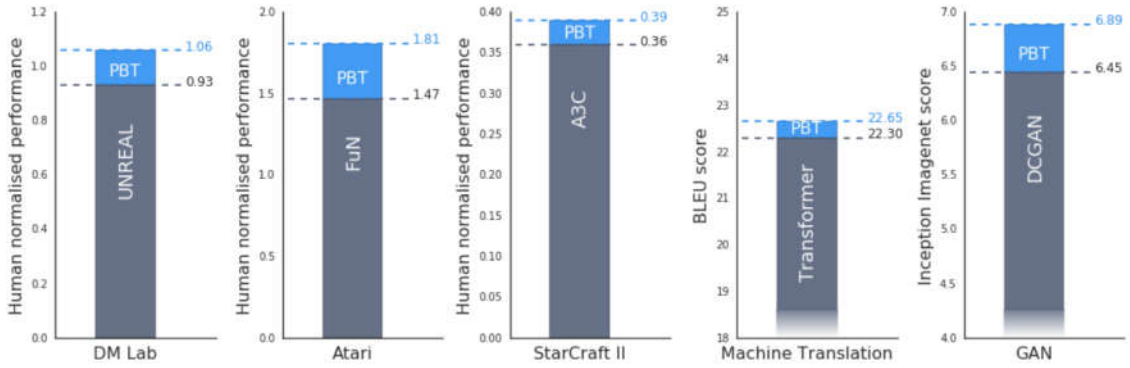
Methods

Scheduling method

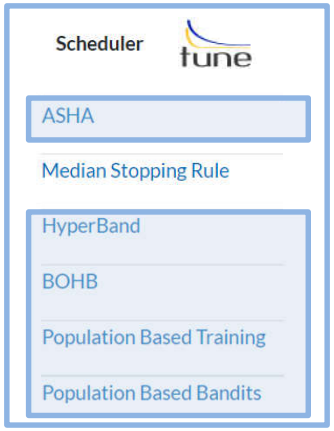
❖ Population-Based Training (PBT)

- (+) Algorithm hyper-parameter HPO에 관해서는 SOTA 성능 기대가능
- (+) 특히 Hyperparameter가 많은 대형 Transformer 모델 HPO에 큰 도움이 됨
- (-) Algorithm hyper-parameter HPO만 가능
- (-) PBT는 population 상태를 저장해야 되기 때문에 원래 코드에서 수정해야할 부분이 상대적으로 많음
- Cf) PBT는 현재 Ray Tune과 NNI 라이브러리에서만 지원. (후속 알고리즘인 PB2은 Ray Tune 에만)

PBT performance against baseline (best hand-tuned architecture)



PBT&PB2 @RayTune



- https://arxiv.org/pdf/1711.09846.pdf
- https://arxiv.org/pdf/2002.02518.pdf

Experiments

Experiments

Classic methods

❖ HPO methods to the (RF/SVM/KNN) classifier on the MNIST dataset

- GS/RS : 시간 효율이 최악
- BO-GP : 정확도는 좋으나 시간효율이 좋지 않음
- BO-TPE : 조건부 관계에 갖는 HPO에 효과적(SVM)
- Hyperband : GS/RS보다 조금 나은 정도
- BOHB : 정확도와 효율성 모두 좋음

Table 4
Performance evaluation of applying HPO methods to the RF classifier on the MNIST dataset.

Optimization Algorithm	Accuracy (%)	CT (s)
Default HPs	90.65	0.09
GS	93.32	48.62
RS	93.38	16.73
BO-GP	93.38	20.60
BO-TPE	93.88	12.58
Hyperband	93.38	8.89
BOHB	93.38	9.45

Table 5
Performance evaluation of applying HPO methods to the SVM classifier on the MNIST dataset.

Optimization Algorithm	Accuracy (%)	CT (s)
Default HPs	97.05	0.29
GS	97.44	32.90
RS	97.35	12.48
BO-GP	97.50	17.56
BO-TPE	97.44	3.02
Hyperband	97.44	11.37
BOHB	97.44	8.18

Table 6
Performance evaluation of applying HPO methods to the KNN classifier on the MNIST dataset.

Optimization Algorithm	Accuracy (%)	CT (s)
Default HPs	96.27	0.24
GS	96.22	7.86
RS	96.33	6.44
BO-GP	96.83	1.12
BO-TPE	96.83	2.33
Hyperband	96.22	4.54
BOHB	97.44	3.84

- https://www.sciencedirect.com/science/article/pii/S0925231220311693

Experiments

Classic methods

❖ HPO methods to the (RF/SVM/KNN) classifier on the Boston-housing dataset

- GS/RS : 시간 효율이 최악
- BO-GP : 정확도는 좋으나 시간효율이 좋지 않음
- BO-TPE : 조건부 관계에 갖는 HPO에 효과적(SVM)
- Hyperband : GS/RS보다 조금 나은 정도
- BOHB : 정확도와 효율성 모두 좋음

Table 7
Performance evaluation of applying HPO methods to the RF regressor on the Boston-housing dataset.

Optimization Algorithm	MSE	CT (s)
Default HPs	31.26	0.08
GS	29.02	4.64
RS	27.92	3.42
BO-GP	26.79	17.94
BO-TPE	25.42	1.53
Hyperband	26.14	2.56
BOHB	25.56	1.88

Table 9
Performance evaluation of applying HPO methods to the KNN regressor on the Boston-housing dataset.

Optimization Algorithm	MSE	CT (s)
Default HPs	81.48	0.004
GS	81.53	0.12
RS	80.77	0.11
BO-GP	80.77	0.49
BO-TPE	80.83	0.08
Hyperband	80.87	0.10
BOHB	80.77	0.09

Table 8
Performance evaluation of applying HPO methods to the SVM regressor on the Boston-housing dataset.

Optimization Algorithm	MSE	CT (s)
Default HPs	77.43	0.02
GS	67.07	1.33
RS	61.40	0.48
BO-GP	61.27	5.87
BO-TPE	59.40	0.33
Hyperband	73.44	0.32
BOHB	59.67	0.31

- https://www.sciencedirect.com/science/article/pii/S0925231220311693

Experiments

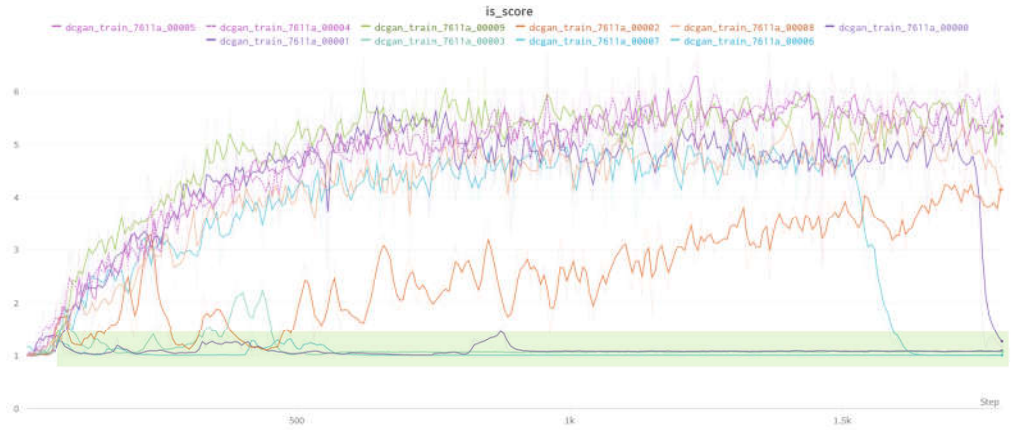
SOTA methods (with ASHA, PBT)

```
config = {
    "netG_lr": lambda: np.random.uniform(1e-2, 1e-5),
    "netD_lr": lambda: np.random.uniform(1e-2, 1e-5),
    "beta1": [0.3,0.5,0.8]
```

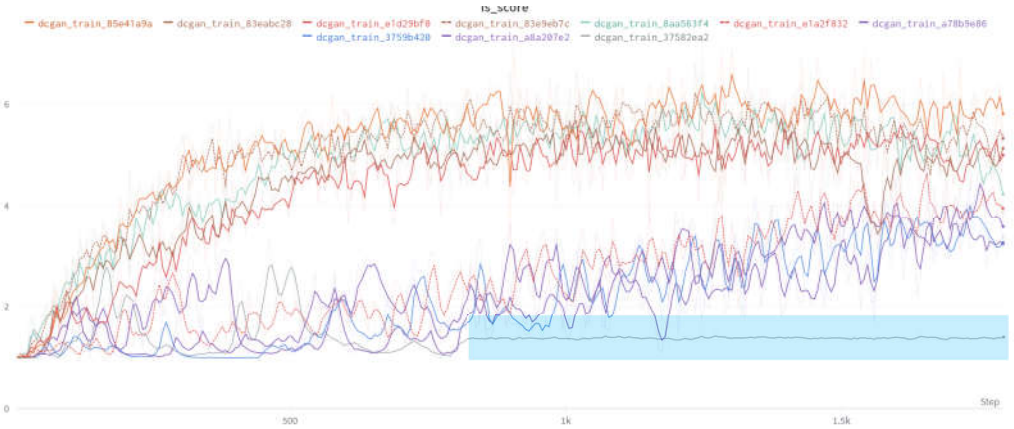
❖ DCGAN on MNIST (10 samples)

- RS의 경우 세 Trial에서 flat한 inception score를 보임 (반면, BO는 한번으로 줄음)

Random Search



Bayesian Optimization



- https://wandb.ai/wandb/DistHyperOpt/reports/Modern-Scalable-Hyperparameter-Tuning-Methods--VmlldzoyMTQxODM

Experiments

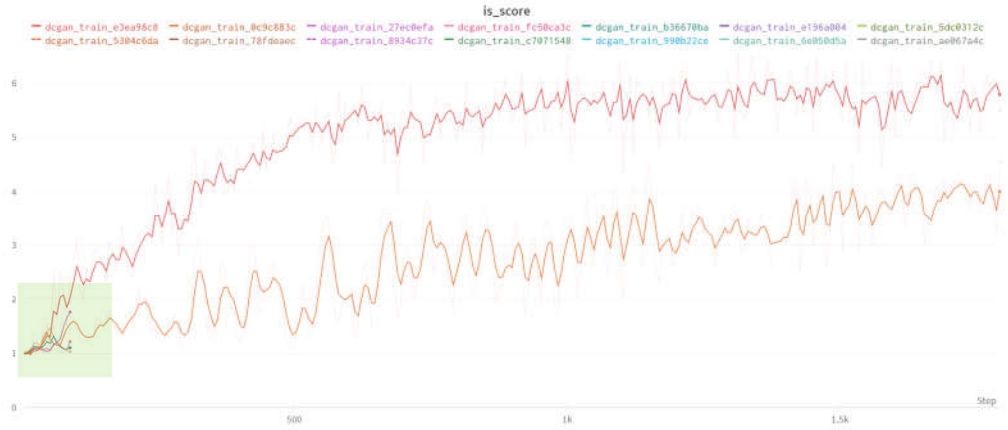
SOTA methods (with ASHA, PBT)

```
config = {
    "netG_lr": lambda: np.random.uniform(1e-2, 1e-5),
    "netD_lr": lambda: np.random.uniform(1e-2, 1e-5),
    "beta1": [0.3,0.5,0.8]
```

❖ DCGAN on MNIST (10 samples)

- ASHA는 초기에 가망 없는 Trial을 빠르게 솎아내었음 / 한편 PBT는 exploit/explore으로 거의 Trial의 최고성능으로 수렴

ASHA



PBT

- https://wandb.ai/wandb/DistHyperOpt/reports/Modern-Scalable-Hyperparameter-Tuning-Methods--VmlldzoyMTQxODM

Experiments

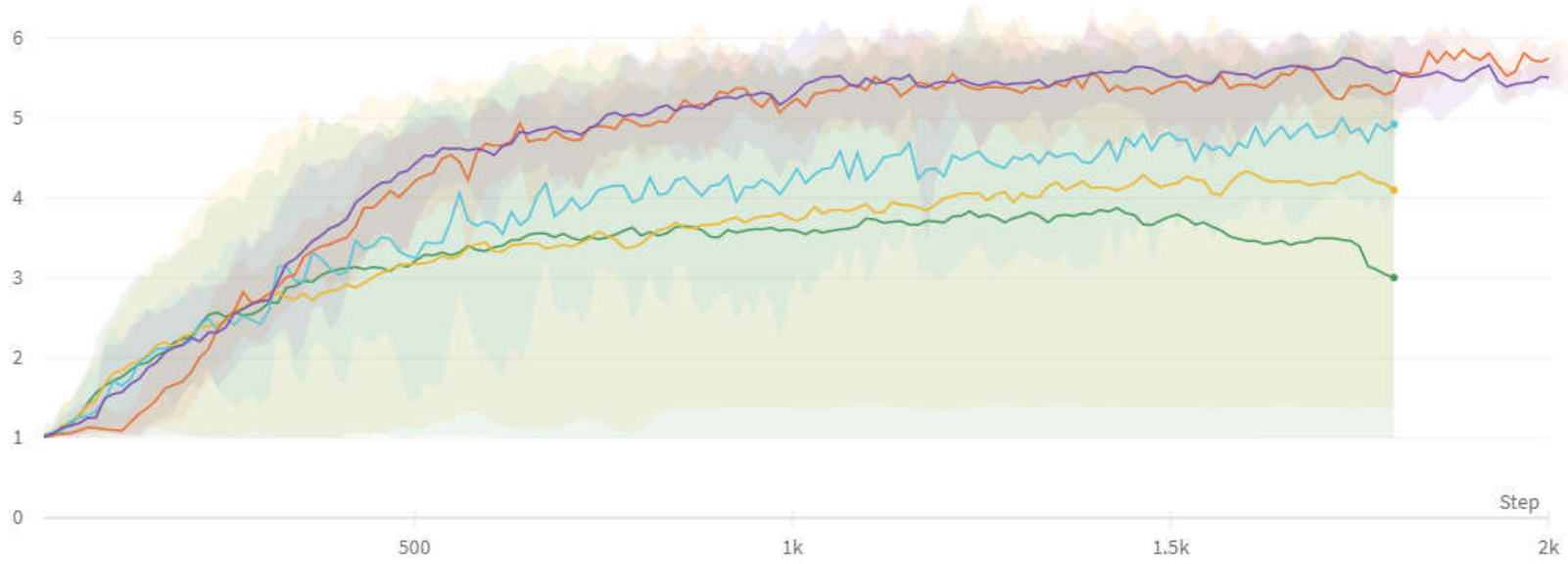
SOTA methods – 1 (ASHA, PBT)

```
config = {
    "netG_lr": lambda: np.random.uniform(1e-2, 1e-5),
    "netD_lr": lambda: np.random.uniform(1e-2, 1e-5),
    "beta1": [0.3,0.5,0.8]
```

❖ DCGAN on MNIST

- PBT(samples=5)≒PBT(samples=10)≥ ASHA(samples=20) ≥ BO(samples=10) ≥ RS(samples=10)

각 HPO 방법론들의 Inception Score 비교



- https://wandb.ai/wandb/DistHyperOpt/reports/Modern-Scalable-Hyperparameter-Tuning-Methods--VmlldzoyMTQxODM
- https://arxiv.org/pdf/1810.05934

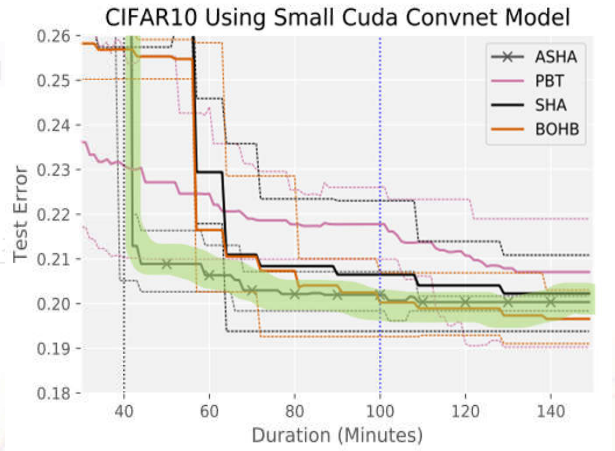
Experiments

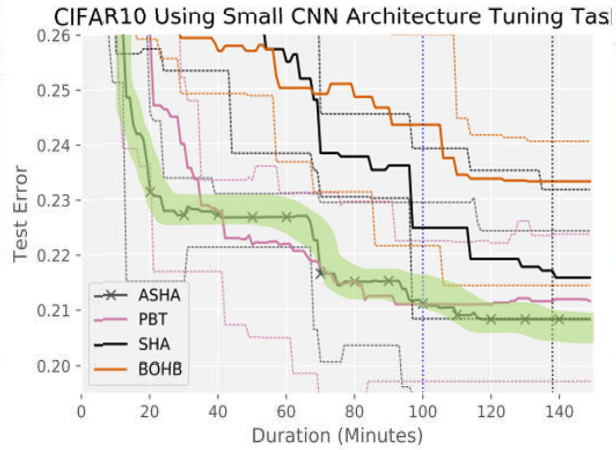
SOTA methods – 1 (ASHA, PBT)

```
config = {
    "netG_lr": lambda: np.random.uniform(1e-2, 1e-5),
    "netD_lr": lambda: np.random.uniform(1e-2, 1e-5),
    "beta1": [0.3,0.5,0.8]
```

❖ DCGAN on MNIST

- PBT(samples=5)≒PBT(samples=1 ≥ RS(samples=10)



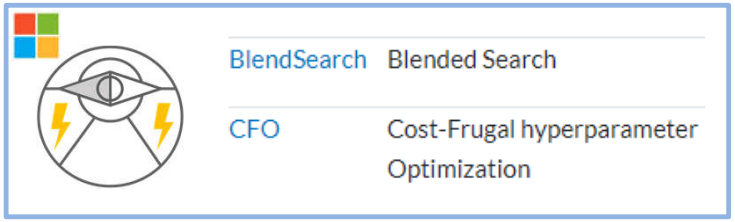


- https://wandb.ai/wandb/DistHyperOpt/reports/Modern-Scalable-Hyperparameter-Tuning-Methods--VmlldzoyMTQxODM
- https://arxiv.org/pdf/1810.05934

Experiments (Additional)

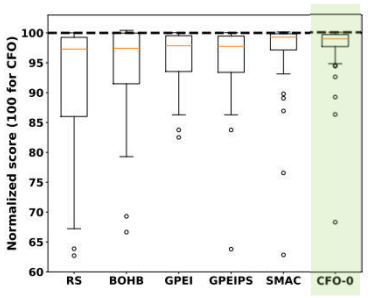
Experiments

SOTA methods – 2 (FLAML)

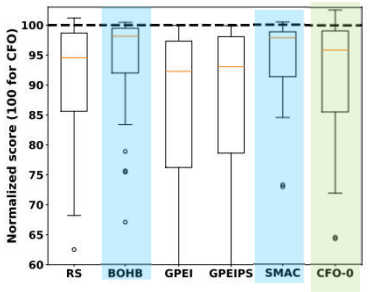


❖ FLAML

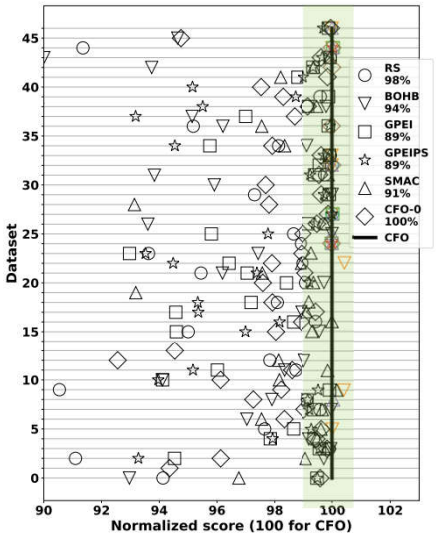
- 2개의 SOTA Searcher 알고리즘 : Cost-Frugal Optimization(CFO) & BlendSearch
- (Made by Microsoft, accepted @(MLSys, AAAI2021, ICLR2021, ICML2021), Current(2021.09) SOTA @AutoML Benchmark)
- Optuna의 CMA-ES와 더불어 현재 가장 손쉽게 접근가능한 SOTA Searcher



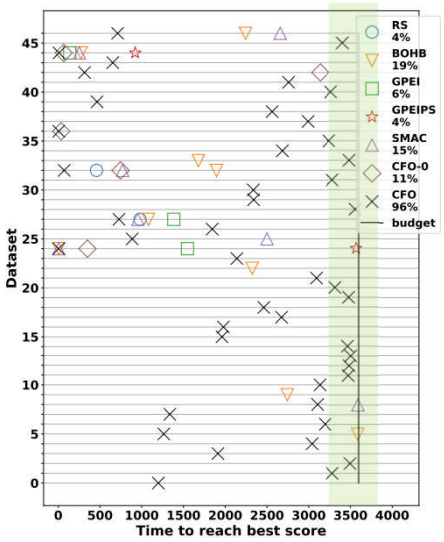
(a) Scores in tuning XGBoost



(b) Scores in tuning DNN



(c) Scores in tuning XGBoost, the higher the better. The legends display the fraction of datasets on which the scores are over 90 for each method



(d) Time used in reaching best score for XGBoost. The legends display the fraction of datasets on which the best score can be reached within 1h

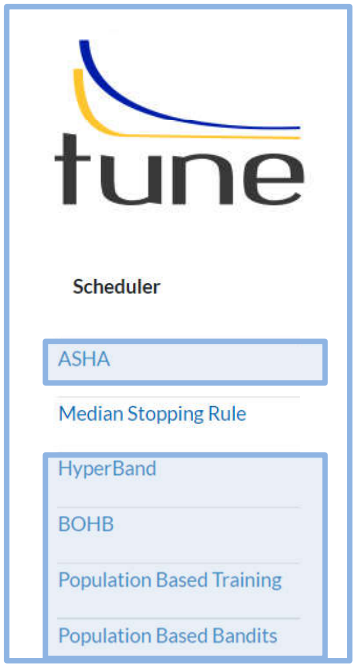
- https://www.anyscale.com/blog/fast-automl-with-flaml-ray-tune
- https://arxiv.org/pdf/2005.01571.pdf
- https://openreview.net/pdf?id=VbLH04pRA3

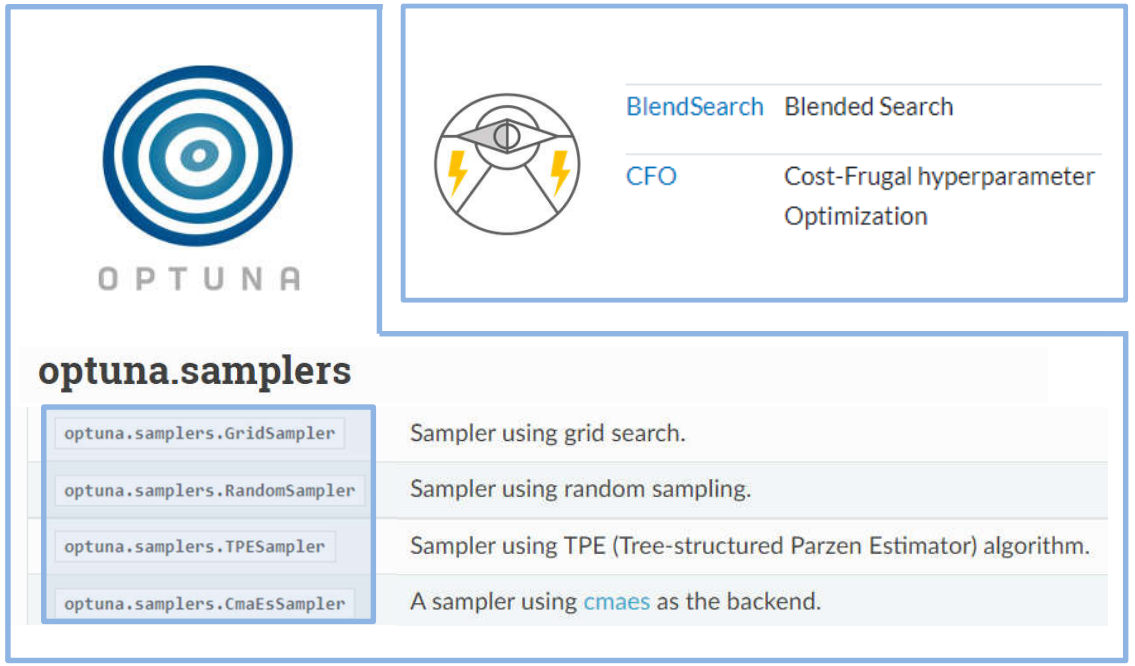
Frameworks

Frameworks

❖ 지금껏 소개된 모든 방법론은 아래 HPO 라이브러리로부터 '손쉽게' 사용가능

- Ray Tune (☆:17.4k) (Made by UC Berkley, Widely used for distributed computing)
- Optuna (☆:5.2k) (CMA-ES Accepted @(AAAI2021), Compatibe with majority of the ML tech stacks (sklearn,PyTorch,TF,XGBoost,..)
- FLAML (☆:1.3k) (Made by Microsoft, accepted @(MLSys, AAAI2021, ICLR2021, ICML2021), Current(2021.09) SOTA @AutoML Benchmark)





- https://github.com/optuna/optuna
- https://github.com/ray-project/ray
- https://github.com/microsoft/FLAML

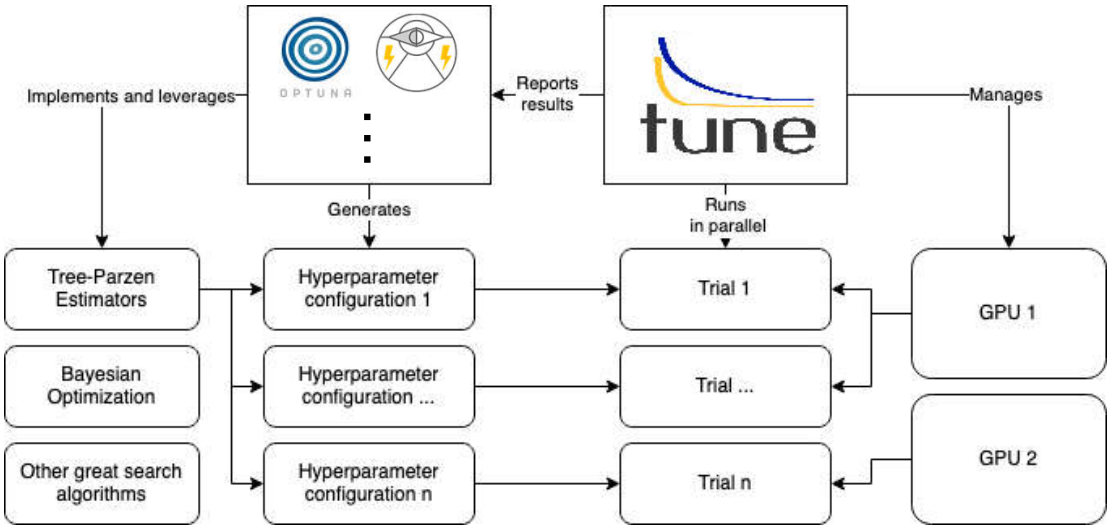
Frameworks

Ray Tune (☆:17.4k)

❖ Ray Tune은 분산처리와 GPU가속을 지원하는 Wrapper 라이브러리

- Searching method는 wrapping된 native 라이브러리(OptunaSearch/BlendSearch/…) 中 선택
- Scheduling method는 Ray Tune에서 native로 지원되며, 필요시 선택해서 사용하면 됨
- Ray Tune 사용시 : (+) 분산처리,GPU가속,Scheduler 사용가능 / (-) 몇몇 native 기능은 사용불가능

SearchAlgorithm	Summary
Random search/grid search	Random search/grid search
AxSearch	Bayesian/Bandit Optimization
BlendSearch	Blended Search
CFO	Cost-Frugal hyperparameter Optimization
DragonflySearch	Scalable Bayesian Optimization
SkoptSearch	Bayesian Optimization
HyperOptSearch	Tree-Parzen Estimators
BayesOptSearch	Bayesian Optimization
TuneBOHB	Bayesian Opt/HyperBand
NevergradSearch	Gradient-free Optimization
OptunaSearch	Optuna search algorithms
ZOOptSearch	Zeroth-order Optimization
SigOptSearch	Closed source
HEBOSearch	Heteroscedastic Evolutionary Ba Optimization



Scheduler
ASHA
Median Stopping Rule
HyperBand
BOHB
Population Based Training
Population Based Bandits

- https://medium.com/optuna/scaling-up-optuna-with-ray-tune-88f6ca87b8c7
- https://docs.ray.io/en/latest/tune/api_docs/suggestion.html
- https://docs.ray.io/en/latest/tune/api_docs/schedulers.html

Frameworks

Ray Tune (☆:17.4k)

❖ Ray Tune은 분산처리와 GPU가속을 지원하는 Wrapper 라이브러리

- Searching method는 wrapping된 native 라이브러리(OptunaSearch/BlendSearch/…) 中 선택
- Scheduling method는 Ray Tune에서 native로 지원되며, 필요시 선택해서 사용하면 됨
- tune.run()에 들어갈 함수는 자체적으로 {Data, model, train, evaluation}을 할 수 있게 사전 구성
- tune.report()에는 최적화하고자 하는 objective function의 criterion(e.g. mean acc.)을 report
- tune.run() 인자 중 config에 탐색할 hyper-parameter 범위를 정의한다
- tune.run() 이외 인자는 HPO 설정값(search_alg, scheduler, metric, mode, num_samples, …)을 정의

```python
def train_mnist(config):
    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda" if use_cuda else "cpu")
    train_loader, test_loader = get_data_loaders()
    model = ConvNet().to(device)

    optimizer = optim.SGD(
        model.parameters(), lr=config["lr"], momentum=config["momentum"])

    for i in range(20):
        train(model, optimizer, train_loader, device)
        acc = test(model, test_loader, device)
        tune.report(mean_accuracy=acc)
```

```python
analysis = tune.run(
    train_mnist,
    config={
        "lr": tune.loguniform(1e-4, 1e-2),
        "momentum": tune.uniform(0.1, 0.9),
    },
    metric="mean_accuracy",
    mode="max",
    search_alg=OptunaSearch(),
    num_samples=10)
print(f"Best config: {analysis.best_config}")
```

- https://medium.com/optuna/scaling-up-optuna-with-ray-tune-88f6ca87b8c7
- https://docs.ray.io/en/latest/tune/api_docs/suggestion.html
- https://docs.ray.io/en/latest/tune/api_docs/schedulers.html

Frameworks

Optuna (☆:5.2k)

❖ "타 ML 라이브러리들에 대한 호환성과 사용자 편의성이 좋은 HPO 라이브러리"

- PyTorch Chainer TensorFlow Keras MXNet Scikit-Learn dmlc XGBoost LightGBM
- 다양한 Searcher(Sampler)와 Scheduler(Pruner)가 Native로 동시에 지원됨(다만, GPU 가속은 안됨)
- 자체적으로 HPO 결과에 대한 다양한 시각화 함수 제공

Optuna Searcher (Sampler)

`optuna.samplers.BaseSampler`	Base class for samplers.
`optuna.samplers.GridSampler`	Sampler using grid search.
`optuna.samplers.RandomSampler`	Sampler using random sampling.
`optuna.samplers.TPESampler`	Sampler using TPE (Tree-structured Parzen Estimator) algorithm.
`optuna.samplers.CmaEsSampler`	A sampler using cmaes as the backend.
`optuna.samplers.PartialFixedSampler`	Sampler with partially fixed parameters.
`optuna.samplers.NSGAIISampler`	Multi-objective sampler using the NSGA-II algorithm.
`optuna.samplers.MOTPESampler`	Multi-objective sampler using the MOTPE algorithm.
`optuna.samplers.IntersectionSearchSpace`	A class to calculate the intersection search space of a `BaseStudy`.
`optuna.samplers.intersection_search_space`	Return the intersection search space of the `BaseStudy`.

Optuna Scheduler (Pruner)

`optuna.pruners.BasePruner`	Base class for pruners.
`optuna.pruners.MedianPruner`	Pruner using the median stopping rule.
`optuna.pruners.NopPruner`	Pruner which never prunes trials.
`optuna.pruners.PatientPruner`	Pruner which wraps another pruner with tolerance.
`optuna.pruners.PercentilePruner`	Pruner to keep the specified percentile of the trials.
`optuna.pruners.SuccessiveHalvingPruner`	Pruner using Asynchronous Successive Halving Algorithm.
`optuna.pruners.HyperbandPruner`	Pruner using Hyperband.
`optuna.pruners.ThresholdPruner`	Pruner to detect outlying metrics of the trials.

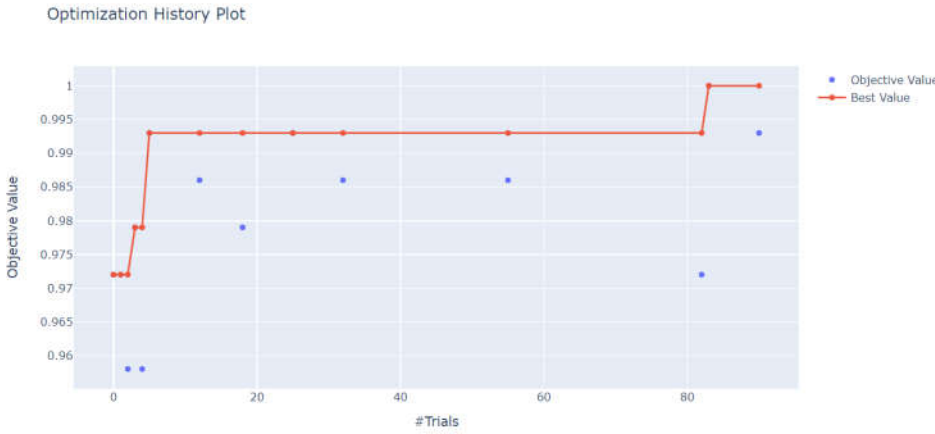
- https://github.com/optuna/optuna

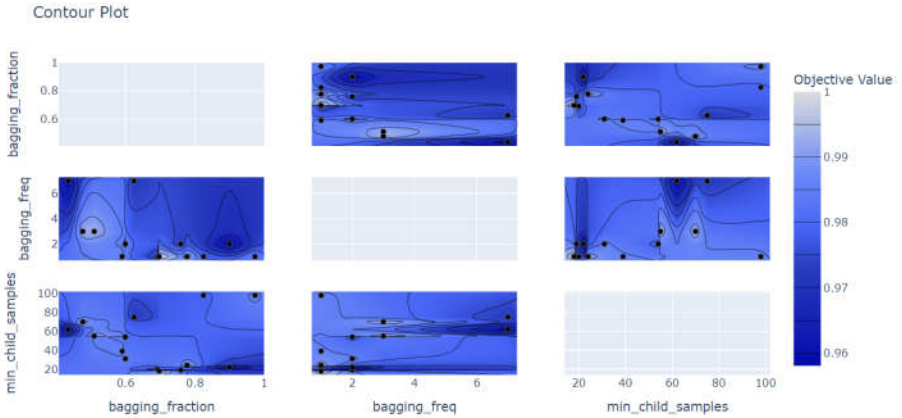
Frameworks

Optuna (☆:5.2k)

❖ "타 ML 라이브러리들에 대한 호환성과 사용자 편의성이 좋은 HPO 라이브러리"

- PyTorch Chainer TensorFlow Keras MXNet Scikit-Learn dmlc XGBoost LightGBM
- 다양한 Searcher(Sampler)와 Scheduler(Pruner)가 Native로 동시에 지원됨(다만, GPU 가속은 안됨)
- 자체적으로 HPO 결과에 대한 다양한 interactive 시각화 툴 제공 (아래는 TensorBoard가 지원하지 않는 류의 plot)
  - ✓ history plot, contour plot, hyperparameter importance



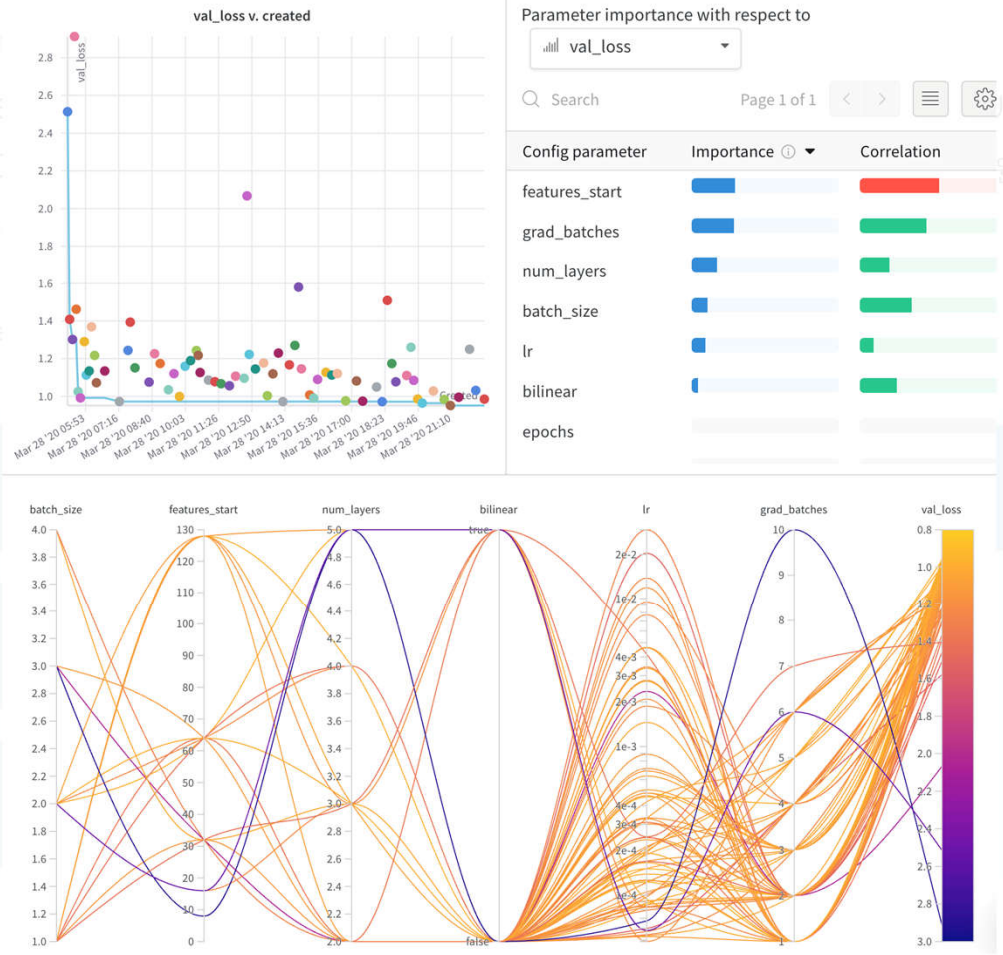


- https://github.com/optuna/optuna
- https://optuna.readthedocs.io/en/stable/tutorial/10_key_features/005_visualization.html
- https://wandb.ai/site

Frameworks

Optuna (☆:5.2k)

❖ Wandb log results



- https://github.com/optuna/optuna
- https://optuna.readthedocs.io/en/stable/tutorial/10_key_features/005_visualization.html
- https://wandb.ai/site

Frameworks

Optuna (☆:5.2k)

❖ “타 ML 라이브러리들에 대한 호환성과 사용자 편의성이 좋은 HPO 라이브러리”

- PyTorch Chainer TensorFlow Keras MXNet Scikit-Learn XGBoost LightGBM
- 사용 방법

1. Wrap model training with an `objective` function and return accuracy
2. Suggest hyperparameters using a `trial` object
3. Create a `study` object and execute the optimization

- 실제 코드 예시)

PyTorch

```python
import torch

import optuna

# 1. Define an objective function to be maximized.
def objective(trial):

    # 2. Suggest values of the hyperparameters using a trial object.
    n_layers = trial.suggest_int('n_layers', 1, 3)
    layers = []

    in_features = 28 * 28
    for i in range(n_layers):
        out_features = trial.suggest_int('n_units_l{}'.format(i), 4, 128)
        layers.append(torch.nn.Linear(in_features, out_features))
        layers.append(torch.nn.ReLU())
        in_features = out_features
    layers.append(torch.nn.Linear(in_features, 10))
    layers.append(torch.nn.LogSoftmax(dim=1))
    model = torch.nn.Sequential(*layers).to(torch.device('cpu'))
    ...
    return accuracy

# 3. Create a study object and optimize the objective function.
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

XGBoost

```python
import xgboost as xgb

import optuna

# 1. Define an objective function to be maximized.
def objective(trial):
    ...

    # 2. Suggest values of the hyperparameters using a trial object.
    param = {
        'silent': 1,
        'objective': 'binary:logistic',
        'booster': trial.suggest_categorical('booster', ['gbtree', 'gblinear', 'dart']),
        'lambda': trial.suggest_loguniform('lambda', 1e-8, 1.0),
        'alpha': trial.suggest_loguniform('alpha', 1e-8, 1.0)
    }

    bst = xgb.train(param, dtrain)
    ...
    return accuracy

# 3. Create a study object and optimize the objective function.
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

- https://github.com/optuna/optuna

Frameworks

FLAML (☆:1.3k)

❖ “빠르고 가벼운 SOTA AutoML 라이브러리”

- 기능 1. Data와 task만 정의해주면 HPO까지 모두 마치는 AutoML
  - 사용가능모델 : [XGBoost, LightGBM, RandomForest, AnyCustomLearner]
  - 가능한 task : [회귀, 분류, 시계열예측,…]

• A basic classification example.

```
from flaml import AutoML
from sklearn.datasets import load_iris
# Initialize an AutoML instance
automl = AutoML()
# Specify automl goal and constraint
automl_settings = {
    "time_budget": 10,  # in seconds
    "metric": 'accuracy',
    "task": 'classification',
    "log_file_name": "test/iris.log",
}
X_train, y_train = load_iris(return_X_y=True)
# Train with labeled input data
automl.fit(X_train=X_train, y_train=y_train,
           **automl_settings)
# Predict
print(automl.predict_proba(X_train))
# Export the best model
print(automl.model)
```

```
from flaml import AutoML
automl = AutoML()
automl.fit(X_train=X_train, y_train=y_train, time_budget=60, estimator_list=['lgbm'])

''' retrieve best model and best configuration found'''
print('Best ML model:', automl.model)
print('Best hyperparameter config:', automl.best_config)
```

- 기능 2. HPO만 수행 (Ray Tune 코드 스타일로)

```
from flaml import tune
tune.run(train_with_config, config={…}, low_cost_partial_config={…}, time_budget_s=3600)
```

- https://github.com/microsoft/FLAML
- https://www.anyscale.com/blog/fast-automl-with-flaml-ray-tune

Conclusion

Conclusion

HPO 방법론들을 쓰기 앞서서…

❖ 모델이 잘 동작하는지부터 먼저 확인!

- 가장 단순한 모델/데이터로 시작
- Single batch에 먼저 overfitting이 가능한지 여부부터 파악 (Andrej Karpathy)

❖ 본격적인 HPO 들어가기 전에 하이퍼파라미터에 대한 대충의 감을 먼저 갖자 (어떤게 민감한건지)

- E.g.) 일반적으로 Learning rate의 민감도는 큼

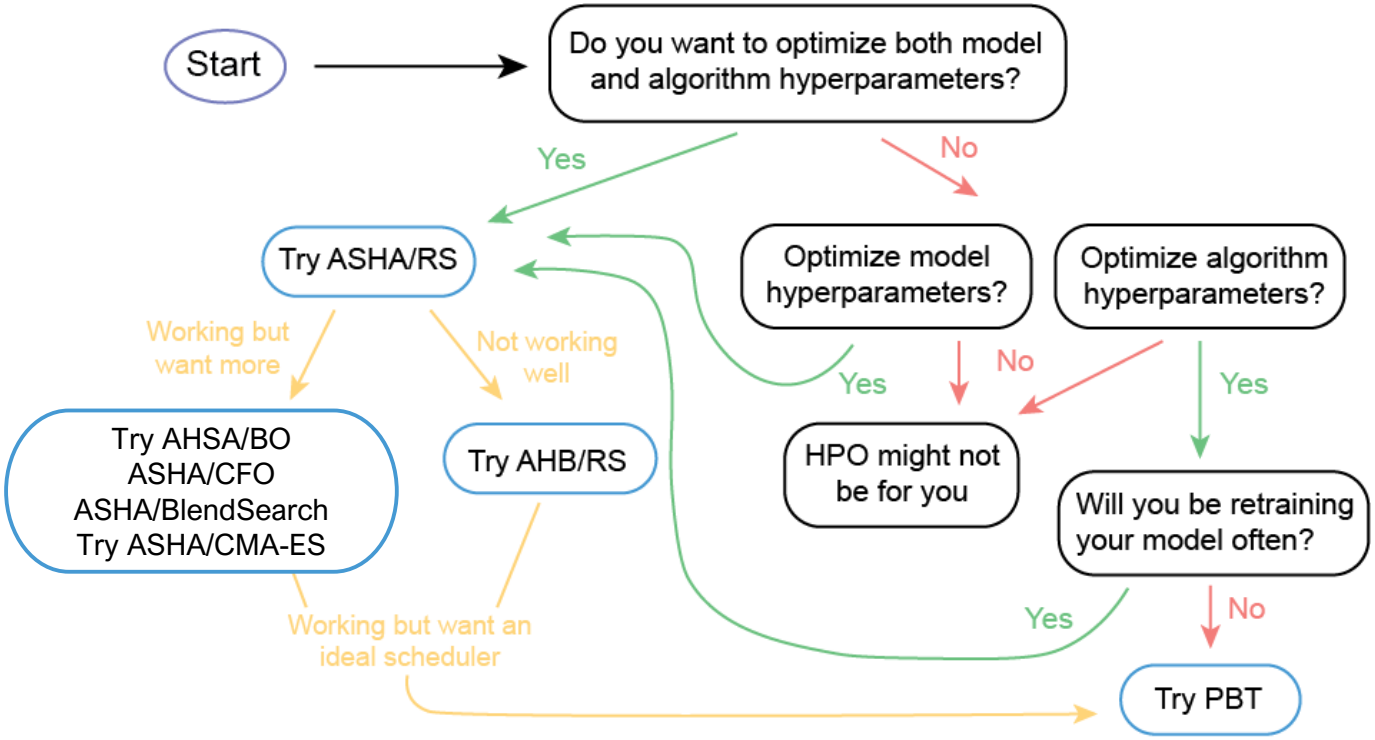
❖ 과대적합하고 있는지, 과소적합하고 있는지 잘 파악하자

- 과소적합 시, 모델 크기(복잡도)를 증가
- 과대적합 시, 데이터를 추가하거나 정규화 시도

Conclusion

앞선 항목들을 확인했다면…

❖ 일반적인 케이스의 경우 아래 순서도 참고



❖ 현 시점 HPO 라이브러리 중 RayTune과 Optuna(+FLAML)가 사용하기 용이하며 알고리즘 구현 多

- HPO Framekwork 비교 : https://neptune.ai/blog/best-tools-for-model-tuning-and-hyperparameter-optimization

- https://wood-b.github.io/post/a-novices-guide-to-hyperparameter-optimization-at-scale/#schedulers-vs-search-algorithms

"Weaker models with well–tuned hyperparameters can outperform stronger, fancier models" [Melis et al. 2018]

Thank you

On the State of the Art of Evaluation in Neural Language Models (Melis et al. 2018)

References

[Papers]

1. Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing, 415, 295–316.
2. Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., & Talwalkar, A. (2018). A system for massively parallel hyperparameter tuning. arXiv preprint arXiv:1810.05934.
3. Loshchilov, I., & Hutter, F. (2016). CMA-ES for hyperparameter optimization of deep neural networks. arXiv preprint arXiv:1604.07269.
4. Wang, C., Wu, Q., Huang, S., & Saied, A. (2020, September). Economic hyperparameter optimization with blended search strategy. In International Conference on Learning Representations.
5. Wu, Q., Wang, C., & Huang, S. (2021, May). Frugal optimization for cost-related hyperparameters. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 35, No. 12, pp. 10347-10354).

[Websites]

1. https://github.com/hibayesian/awesome-automl-papers
2. https://github.com/optuna/optuna
3. https://github.com/ray-project/ray
4. https://github.com/microsoft/FLAML
5. https://wandb.ai/wandb/DistHyperOpt/reports/Modern-Scalable-Hyperparameter-Tuning-Methods--VmlldzoyMTQxODM
6. https://wood-b.github.io/post/a-novices-guide-to-hyperparameter-optimization-at-scale/#schedulers-vs-search-algorithms
7. https://blog.ml.cmu.edu/2018/12/12/massively-parallel-hyperparameter-optimization/
8. https://medium.com/optuna/introduction-to-cma-es-sampler-ee68194c8f88
9. https://www.anyscale.com/blog/fast-automl-with-flaml-ray-tune
10. https://neptune.ai/blog/best-tools-for-model-tuning-and-hyperparameter-optimization

[Videos]

1. https://www.youtube.com/watch?v=6wRBGNLgQFU
2. https://www.youtube.com/watch?v=2QX6jjMt1Eg
3. https://www.youtube.com/watch?v=IqQT8se9ofQ