

CNN Hyperparameter Optimization

Methods

Akash, Hiroshi

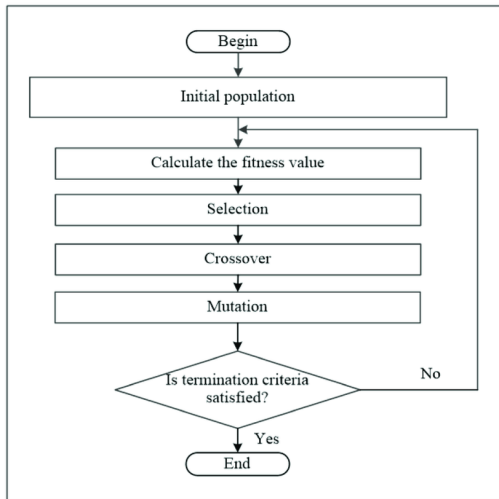
RWTH Aachen University

August 31, 2022

Overview

1. Genetic Algorithm
2. Particle Swarm Optimization
3. Bayesian Optimization
4. Simulated Annealing (SA)
5. Weighted Random Search
6. Hyperband Algorithm
7. BOHB Algorithm
8. DEHB Algorithm

Genetic Algorithm



Example: CNN hyperparameter optimization

- **Initial population:** CNNs with different configurations
- **Fitness value:** evaluation metrics (accuracy, loss, F1-score, etc.)
- **Selection:** select best N configs from current population
- **Crossover:** combine configs of 2 CNNs to get a new config
- **Mutation:** change config of a CNN to get a new config
- **Termination Criteria:** example accuracy $> 95\%$

Genetic / Evolutionary Algorithm

Related Works:

- **Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms** [AD19]
 - Accuracy on CIFAR-10: 80.62% (3Conv + 2FC + Dropout + stride 1 on all layers; 2 days)
- **Efficient Hyperparameter Optimization in Deep Learning using a variable length Genetic Algorithm** [XYB⁺20]
 - Consider variable length chromosomes → can react to architecture changes
 - Accuracy on CIFAR-10 after 30h: 88.92% (RS: 58.66%, Classical GA: 80.75%)
- **Optimization of Hyper-parameter for CNN Model using Genetic Algorithm** [YYK⁺19]
 - Accuracy on MNIST 99.6% after 30 generation

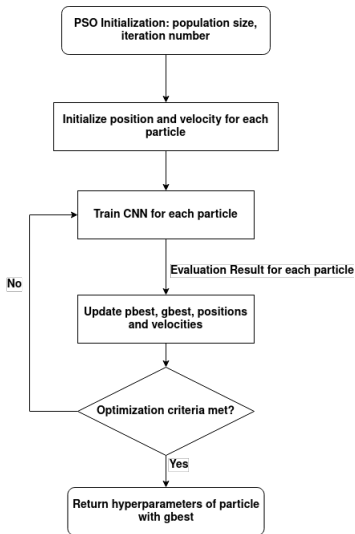
Requirements:

- Initialization of hyperparameters
- Access to evaluation metrics
- **Optional:** Parallel execution of training pipeline

Particle Swarm Optimization (PSO)

1. Initialize PSO: population size, iteration number
2. Initialization of particles and positions x and velocities v of each particle
 - First population is created randomly
 - Set positions and velocities according hyperparameters of CNN
3. Train CNN with the parameters of all particles
4. Measure cost (e.g. error) of all particles
 - **pbest** (local best): smallest cost for each particle
 - **gbest** (global best): smallest cost among all the pbest
5. Update position x and velocity v :
 - $v_{n+1} = v_n + c_1 r_1 (p_{\text{best}} - x_n) + c_2 r_2 (g_{\text{best}} - x_n)$
 - $x_{n+1} = x_n + v_{n+1}$
 - n : particle number; c_1, c_2 : learning factor (usually 2); v : particle velocity based on pbest and gbest; x : current particle; r_1, r_2 : random variable between (0,1)
6. Repeat 3-5: iteration number

Particle Swarm Optimization (PSO)



PSO: Comparison to GA

Advantages:

- Easy implementation
 - No setup for genetic operators
- Usually faster convergence
- Very few parameters → computational efficient
- Global search algorithm: Particles share the information of gbest
- Principally "2 populations": pbest and current positions
 - Greater diversity and exploration

Disadvantages:

- Continuous technique → poorly suited to combinatorial problems
- Poor local search: risk of trapping in local minima

Particle Swarm Optimization (PSO)

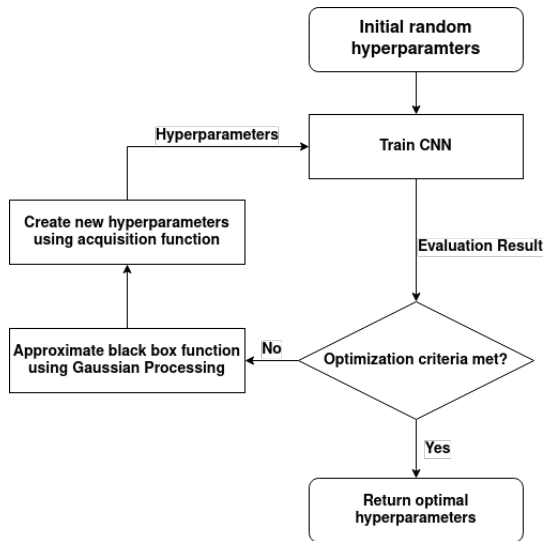
Related Works:

- **Hyper-parameter optimization of convolutional neural network based on particle swarm optimization algorithm [FARS21]**
 - PSO on convolution layer size and kernel size (swarm size: 15; iterations: 30)
 - Error on MNIST: 0.44% (LeNet-5: 0.95%)
- **Optimization of Convolutional Neural Network Using the Linearly Decreasing Weight Particle Swarm Optimization [SF22]**
 - Additional weight parameter during calculation of position and velocity
 - Weight decreases over time → LDWPSO (avoids trapping in local minima)
 - Accuracy on MNIST after 5 epochs: 98.95% (swarm size 10, iterations: 10); LeNet-5: 94.02%
 - Accuracy on CIFAR-10 after 10 epochs: 69.37%; LeNet-5: 28.07%

Requirements:

- Initialization of hyperparameters
- Access to evaluation metrics
- **Optional:** Parallel execution of training pipeline

Bayesian Optimization (BO)



Bayesian Optimization (BO)

- Commonly: BO with Gaussian Processing (SMAC: with Random Forest)
- **BO**: maximize blackbox function (e.g. accuracy) on hyperparameters
 - MLE for assumption of parameter distribution
- **Gaussian Processing**: approximation of blackbox function
- **Acquisition function**: decision where to sample next
 - Commonly: Expected Improvement function
- Search until stopping condition

Related Works:

- **Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization [WCZ⁺19]**
 - Optimization of learning rate of SGD and batch size
 - Accuracy on MNIST before tuning: 98.83%; after 50 iterations: 99.14%

Requirements:

- Initialization of hyperparameters
- Access to evaluation metrics

BO: Comparison to Evolutionary Algorithm (EA)

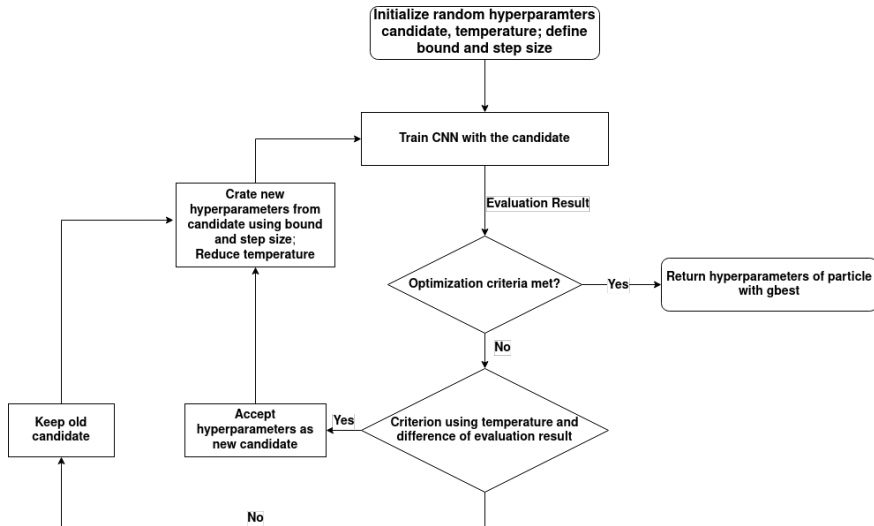
Advantages:

- Computational and memory efficient
- Takes less time on the evaluation
- Takes fewer experiments to reach better value

Disadvantages:

- Probabilistic model required: EA works directly with the objective function
- No parallelism: each experiment depends on the previous experiment
- More complex implementation
- Adapted to low dimensional problems

Simulated Annealing (SA)



Simulated Annealing (SA)

- Stochastic global search algorithm
- Extension of **Hill Climbing**
 - **Difference**: also accept probabilistically a worse point
- Use function whether to accept a new point
 - $\text{criterion} = \exp\left(\frac{f(x_{\text{new}}) - f(x_{\text{old}})}{\text{temperature}}\right)$
 - Temperature decreases over time \rightarrow less probability accepting worse solution
- **Effect on poor solution**: more chances to be accepted in early and less chances in later stages
 - **Beginning**: Search for area of global optima; **later**: search the optima in this area

SA: Comparison to Bayesian Optimization (BO)

Advantages:

- Less risk trapping in local minima
- No assumptions about the model
- Easier implementation

Disadvantages:

- Takes more steps to improve (jumps randomly around solution space)

Similarities:

- No parallelism: each experiment depends on the previous experiment
- Computational and memory efficient

Simulated Annealing (SA)

Related Works:

- **Simulated annealing algorithm for deep learning** [RFA15]
 - Accuracy on MNIST after 10 epochs: 97.71% (original CNN: 97.27%)
- **Multi-objective simulated annealing for hyper-parameter optimization in convolutional neural networks** [GK21]
 - MOSA: 2 objectives considered during optimization (accuracy and computational complexity)
 - Accuracy on CIFAR-10: 91.97% (iterations: 500); LeNet-5: 71.86%; VGGNet-16: 86.38%

Requirements:

- Initialization of hyperparameters
- Access to evaluation metrics

Weighted Random Search (WRS)

- **Intuition:** a value produced a good result is a good candidate for the next step
 - Test with new combinations of other hyperparameter values
- **First:** Execute common RS
 - Determine weight (importance) of each hyperparameter by computing the variation of the objective function
- **WRS:** greater probability of change to the variables with greater weight

Related Works:

- **Weighted Random Search for CNN Hyperparameter Optimization [AF20]**
 - 0.85 CNN accuracy obtained on CIFAR-10 after 300 iterations (RS: 0.81; PSO, BO: 0.83)

Requirements:

- Initialization of hyperparameters
- Access to evaluation metrics

Hyperband Algorithm I

- Hyperband extends the SuccessiveHalving algorithm proposed for hyperparameter optimization [JT15]
 - uniformly allocate a budget to a set of hyperparameter configurations, evaluate the performance of all configurations, throw out the worst half, and repeat until one configuration remains [LJD⁺17].
- There are two components to Hyperband; (1) the inner loop invokes SuccessiveHalving for fixed values of n and r and (2) the outer loop iterates over different values of n and r
 - “ n versus B/n ” where B = budget, considering several possible values of n for a fixed B .
 - with each value of n is a minimum resource r that is allocated to all configurations before some are discarded; a larger value of n corresponds to a smaller r and hence more aggressive early-stopping
 - Two inputs (1) R , the maximum amount of resource that can be allocated to a single configuration, and (2) η , an input that controls the proportion of configurations discarded in each round of SuccessiveHalving

Hyperband Algorithm II

- each run of SuccessiveHalving within Hyperband as a “bracket.”
- Hyperband begins with the most aggressive bracket $s = s_{\max}$, which sets n to maximize exploration, subject to the constraint that at least one configuration is allocated R resources
- Each subsequent bracket reduces n by a factor of approximately $\sqrt[s]{R}$ until the final bracket, $s = 0$, in which every configuration is allocated R resources (this bracket simply performs classical random search)
- at the end returns, Configuration with the smallest intermediate loss seen so far

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

Hyperband Experiments I

- 3 data sets: CIFAR-10, MRBI, SVHN
- Hyperband with three different resource types: iterations, data set subsamples, and feature samples
- compared with Bayesian optimization algorithms—SMAC, TPE, and Spearmint—using their default settings
- Iterations(Early-Stopping Iterative Algorithms for Deep Learning)
 - For CIFAR-10, Hyperband is over an order of-magnitude faster than its competitors
 - For MRBI hyperband is over an order of-magnitude faster than its competitors
 - For SVHN, while Hyperband finds a good configuration faster, Bayesian optimization methods are competitive and SMAC (early) outperforms Hyperband.
- Data Set Subsampling
 - The results on all 117 data sets, show that Hyperband outperforms random search in test error rank despite performing worse in validation error rank

Hyperband Experiments II

- Bayesian methods outperform Hyperband and random search in test error performance but also exhibit signs of overfitting to the validation set, as they outperform Hyperband by a larger margin on the validation error rank
- However, for the subset of 21 data sets, shows that Hyperband outperforms all other searchers on test error rank, including random $2\times$ by a very small margin.
- for smaller data sets, the startup overhead was high relative to total training time, while for larger data sets, only a handful of configurations could be trained within the hour window.
- Feature Subsampling to Speed Up Approximate Kernel Classification
 - Hyperband is around $6\times$ faster than Bayesian methods and random search

BOHB (Robust and Efficient Hyperparameter Optimization at Scale)

- Combines Bayesian optimization (BO) and Hyperband (HB).
- BOHB relies on HB to determine how many configurations to evaluate with which budget [FKH18].
- Replaces the random selection of configurations at the beginning of each HB iteration by a model-based search(BO).
- applies HB once configurations are selected.

BOHB Experiment I

- Counting ones problem, 16 dimensional space with 8 categorical and 8 continuous hyperparameters.
 - BOHB worked as well as HB in the beginning and then quickly started to perform better
 - Random search worked very poorly on this benchmark and was quickly dominated by the model-based methods SMAC and TPE.
 - Even though HB was faster in the beginning, SMAC and TPE clearly outperformed it after having obtained a sufficiently informative model.
 - for 64 dimensions, TPE and SMAC started to perform better than BOHB since the noise grows and evaluating configurations on a smaller budget does not help to build better models for the full budget.
- Comprehensive Experiments on Surrogate Benchmarks
 - SUPPORT VECTOR MACHINE ON MNIST
 - BOHB achieved similar performance as Fabolas and worked slightly better than HB
 - FEED-FORWARD NEURAL NETWORKS ON OPENML DATASETS
 - BOHB achieved the same final performance as HB 100 times faster

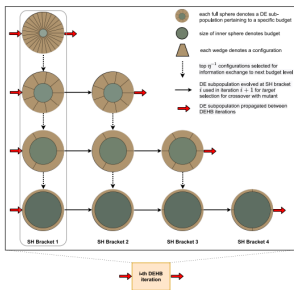
BOHB Experiment II

- Bayesian Neural Networks
 - THB initially performed better than TPE, but TPE caught up given enough time
 - BOHB converged faster than both HB and TPE and even found a better configuration than the baselines on the Boston housing dataset
- Reinforcement Learning
 - HB and BOHB worked equally well in the beginning, but BOHB converged to better configurations in the end
- Convolutional Neural Networks on CIFAR-10
 - strong indication of the practical usefulness of BOHB for resource constrained optimization.

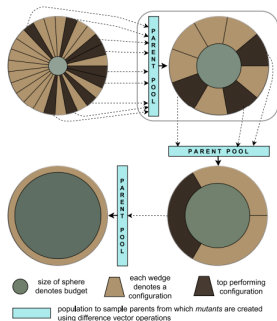
DEHB (Evolutionary Hyperband for Scalable, Robust and Efficient Hyperparameter Optimization)

- While BOHB is among the best general-purpose HPO methods, it still has problems with optimizing discrete dimensions and does not scale as well to high dimensions.
- Idea is to improve BOHB
- DEHB is based on a combination of the evolutionary optimization method of differential evolution and Hyperband [AMH21].
- DEHB fulfills all the desiderata of a good HPO optimizer, and in particular achieves more robust strong final performance than BOHB, especially for high-dimensional and discrete-valued problems
- A key design principle of DEHB is to share information across the runs it executes at various budgets

DEHB (Evolutionary Hyperband for Scalable, Robust and Efficient Hyperparameter Optimization)



(a) Internals of a DEHB iteration showing information flow across fidelities (top-down), and how each subpopulation is updated in each DEHB iteration (left-right).



(b) Modified SH routine under DEHB

DEHB Experiments I

- BOHB VS DEHB, random search as a base bench mark.
- Artificial Toy Function: Stochastic Counting Ones
 - BOHB requires many more samples to switch to model-based search which slows its convergence in comparison to the lower dimensional cases.
 - DEHB's convergence rate is almost agnostic to the increase in dimensionality
- Surrogates for Feedforward Neural Networks
 - DEHB and BOHB having similar anytime performance and DEHB achieving the best final score
 - 1000x speedup over random search
- Bayesian Neural Networks
 - BOHB and DEHB perform similarly
 - Both are about 2x faster than RS
- Reinforcement Learning
 - BOHB and DEHB are able to improve continuously, showing similar performance

DEHB Experiments II

- speeding up over random search by roughly 2x
- NAS Benchmarks
 - BOHB and DEHB are able to improve continuously, showing similar performance
 - speeding up over random search by roughly 2x

DEHB compared with others

- Based on the mean validation regret, all algorithms can be ranked for each benchmark

	RS	HB	BOHB	TPE	SMAC	RE	DE	DEHB
<i>Avg. rank</i>	7.46	6.54	4.42	4.35	4.73	3.16	2.96	2.39

- (a) Mean ranks based on final mean validation regret for all algorithms tested for all benchmarks.



Nurshazlyn Mohd Aszemi and PDD Dominic.

Hyperparameter optimization in convolutional neural network using genetic algorithms.

International Journal of Advanced Computer Science and Applications, 10(6), 2019.



Razvan Andonie and Adrian-Catalin Florea.

Weighted random search for cnn hyperparameter optimization.

arXiv preprint arXiv:2003.13300, 2020.



Noor H. Awad, Neeratyoy Mallik, and Frank Hutter.

DEHB: evolutionary hyperband for scalable, robust and efficient hyperparameter optimization.


CoRR, abs/2105.09821, 2021.





Zainab Fouad, Marco Alfonse, Mohamed Roushdy, and Abdel-Badeeh M Salem.

Hyper-parameter optimization of convolutional neural network based on particle swarm optimization algorithm.

Bulletin of Electrical Engineering and Informatics, 10(6):3377–3384, 2021.

 Stefan Falkner, Aaron Klein, and Frank Hutter.
BOHB: robust and efficient hyperparameter optimization at scale.
CoRR, abs/1807.01774, 2018.

 Ayla Gülcü and Zeki Kuş.
Multi-objective simulated annealing for hyper-parameter optimization in convolutional neural networks.
PeerJ Computer Science, 7:e338, 2021.

 Kevin Jamieson and Ameet Talwalkar.
Non-stochastic best arm identification and hyperparameter optimization, 2015.

 Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar.
Hyperband: A novel bandit-based approach to hyperparameter optimization.
The Journal of Machine Learning Research, 18(1):6765–6816, 2017.

 LM Rasdi Rere, Mohamad Ivan Fanany, and Aniati Murni Arymurthy.
Simulated annealing algorithm for deep learning.

Procedia Computer Science, 72:137–144, 2015.



Tatsuki Serizawa and Hamido Fujita.

Optimization of convolutional neural network using the linearly decreasing weight particle swarm optimization.

In 36 (2022), pages 2S4IS2b03–2S4IS2b03. , 2022.



Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng.

Hyperparameter optimization for machine learning models based on bayesian optimizationb.

Journal of Electronic Science and Technology, 17(1):26–40, 2019.



Xueli Xiao, Ming Yan, Sunitha Basodi, Chunyan Ji, and Yi Pan.

Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm.

arXiv preprint arXiv:2006.12703, 2020.



Ji-Hyun Yoo, Hyun-il Yoon, Hyeong-Gyun Kim, Hee-Seung Yoon, and Seung-Soo Han.

Optimization of hyper-parameter for cnn model using genetic algorithm.

In 2019 1st International Conference on Electrical, Control and Instrumentation Engineering (ICECIE), pages 1–6, 2019.

The End