* **A Comprehensive List of Hyperparameter Optimization & Tuning Solutions**
* auto\_ml — [Automated machine learning](https://github.com/ClimbsRocks/auto_ml)
* BTB — [Bayesian Tuning and Bandits](https://github.com/HDI-Project/BTB)
* Chocolate — [Decentralized Hyperparameter Optimization](https://github.com/AIworx-Labs/chocolate)
* Cornell-MOE — [parallel Bayesian optimization algorithms](https://github.com/wujian16/Cornell-MOE)
* deap — [Evolutionary Algorithm Optimization](https://github.com/DEAP/deap)
* devol — [Evolutionary Algorithm optimization](https://github.com/joeddav/devol)
* GPyOpt — [Gaussian Process Optimization](https://github.com/SheffieldML/GPyOpt)
* H20 — [Automatic Machine Learning](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/downloading.html)
* HORD — [Deterministic RBF Surrogates](https://github.com/ilija139/HORD)
* HPOlib — [Hyperparameter Optimizer Wrapper](https://github.com/automl/HPOlib)
* HpBandSter — [A distributed Hyperband implementation on Steroids](https://github.com/automl/HpBandSter)
* hypergrad — [Differentiation based optimization](https://github.com/HIPS/hypergrad)
* Hyperopt — [Distributed Async Optimization](https://github.com/hyperopt/hyperopt)
* mlrMBO — [Bayesian optimization for R](https://github.com/mlr-org/mlrMBO)
* pbt — [Population Based Training](https://github.com/MattKleinsmith/pbt)
* pycma — [CMA-ES optimization](https://github.com/CMA-ES/pycma)
* rbfopt — [Derivative-free optimization](https://github.com/coin-or/rbfopt)
* ROBO — [Bayesian Optimization Framework](https://github.com/automl/RoBO)
* SMAC3 — [Sequential Model-base Algorithm Configuration](https://github.com/automl/SMAC3)
* spearmint — [Bayesian-based Optimization](https://github.com/HIPS/Spearmint)
* TPOT — [Automated Machine Learning tool](https://github.com/EpistasisLab/tpot)
* test-tube — [Track and Test Machine Learning Codes](https://github.com/williamFalcon/test-tube)
* Tune — [Scalable Hyperparameter Search](https://ray.readthedocs.io/en/latest/tune.html)

**Bayesian optimization:** works by building a surrogate function (in the form of a probability model) of the objective function P (score | hyperparameters). The surrogate function is much cheaper to evaluate than the objective, so the algorithm chooses the next values to try in the objective based on maximizing a criterion on the surrogate (usually expected improvement),

The idea is that as the data accumulates, the surrogate function gets closer and closer to the objective function, and the hyperparameter values that are the best in the surrogate function will also do the best in the objective function.

Surrogate is The surrogate function, also called the response surface, is the probability representation of the objective function built using previous evaluations

Bayesian optimization methods differ in the algorithm used to build the surrogate function and choose the next hyperparameter values to try.

Some of the common choices are:

* Gaussian Process (implemented in Spearmint),
* Random Forest Regression (in SMAC),
* Tree Parzen Estimator (TPE) in Hyperopt

<https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f>

**non-parametric density estimators**

estimators have no fixed structure and depend upon all the data points to reach an estimate.

**Kernel Density Estimation**

technique to estimate the unknown probability distribution of a random variable based on a sample of points taken from that distribution. We are estimating the probability density function of the variable, and we use kernels to do this, hence the name.

<https://www.youtube.com/watch?v=x5zLaWT5KPs>

**Hyperband Optimization**

[**https://github.com/chocolocked/hyperband**](https://github.com/chocolocked/hyperband)

the BOHB algorithm consists of Bayesian Optimization (using Tree structure Parzen) + Hyperband.

I started with learning more about Hyperband algorithm and its implementation

Sources:

* <https://arxiv.org/pdf/1603.06560.pdf>
* <https://people.eecs.berkeley.edu/~kjamieson/hyperband.html>

Code Examples:

* <https://github.com/zygmuntz/hyperband>
* <https://github.com/chocolocked/hyperband>

my objective is to implement Hyperband Optimization on Spark (using Spark Distribution mechanism).

Hyperband:

* allocating more resources to promising hyperparameter conﬁgurations while quickly eliminating poor ones
* Resources can take various forms, including size of training set, number of features, or number of iterations for iterative algorithms
* It addressing how to allocate resources among randomly sampled hyperparameter conﬁgurations
* Hyperband extends the SuccessiveHalving algorithm
* The idea behind the original SuccessiveHalving algorithm follows directly from its name: 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, The algorithm allocates exponentially more resources to more promising conﬁgurations.
* if more resources are required before conﬁgurations can diﬀerentiate themselves in terms of quality then it would be reasonable to work with a small number of conﬁgurations. In contrast, if the quality of a conﬁguration is typically revealed after a small number of resources then n is the bottleneck and we should choose n to be large.
* Consider the number of conﬁgurations = n and budget B, B/n resources are allocated on average across the conﬁgurations
* Hyperband, addresses this “n versus B/n” problem by considering several possible values of n for a ﬁxed B, in essence performing a grid search over feasible value of n.
* Associated 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
* There are two components to Hyperband
  + (1) the inner loop invokes SuccessiveHalving for ﬁxed values of n and r
  + the outer loop iterates over diﬀerent values of n and r
* Hence, a single execution of Hyperband takes a ﬁnite budget of (smax + 1)B;
* B, is Fixed Budget
* R, is the minimum resource that is allocated to all conﬁgurations before some are discarded
* R, the maximum amount of resource that can be allocated to a single conﬁguration
* η, an input that controls the proportion of conﬁgurations discarded in each round of SuccessiveHalving
* (Smax + 1) = [ logη (R) ], how many diﬀerent brackets are considered

Hyperband begins with the most aggressive bracket s = smax, which sets n to maximize exploration

Each subsequent bracket reduces n by a factor of approximately η until the ﬁnal bracket, s = 0, in which every conﬁguration is allocated R resources

removes the need to select n for a ﬁxed budget at the cost of approximately smax + 1 times more work than running SuccessiveHalving for a single value of n.

Run concurrent Jobs in Spark

parallel model evaluation in ML tuning

<https://github.com/apache/spark/commit/16c4c03c71394ab30c8edaf4418973e1a2c5ebfe>

<https://spark.apache.org/docs/2.2.0/ml-tuning.html>

Spark Random Search

<https://towardsdatascience.com/hyperparameters-part-ii-random-search-on-spark-77667e68b606>

hyperband

<https://arxiv.org/pdf/1603.06560.pdf>

<https://people.eecs.berkeley.edu/~kjamieson/hyperband.html>

<https://github.com/benoitdescamps/Hyperparameters-tuning>

<https://github.com/zygmuntz/hyperband>

<https://github.com/chocolocked/hyperband>

BOHB

<https://automl.github.io/HpBandSter/build/html/quickstart.html>

<https://www.automl.org/blog_bohb/>

<https://github.com/automl/HpBandSter>

since the BOHB algorithm consists of ( Bayesian Optimization + Hyperband),

Dataset

* Covertype Data Set
  + URL: <https://archive.ics.uci.edu/ml/datasets/Covertype>
  + Instances: 581012
  + Size: 11 MB
* Census-Income (KDD)
  + URL: <https://archive.ics.uci.edu/ml/datasets/Census-Income+%28KDD%29>
  + Instances: 299285
  + Size: 9.3 MB
* KDD Cup 1999
  + URL: <https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data>
  + Instances: 4000000
  + Size: 18 MB
* Statlog (Shuttle)
  + URL: <https://archive.ics.uci.edu/ml/datasets/Statlog+%28Shuttle%29>
  + Instance: 58000
  + Size: 300K
* Poker Hand
  + URL: <https://archive.ics.uci.edu/ml/datasets/Poker+Hand>
  + Instances: 1025010
  + Size: 23 MB
* URL Reputation
  + URL: <https://archive.ics.uci.edu/ml/datasets/URL+Reputation>
  + Instances: 2396130
  + Size: 234 MB
* Record Linkage Comparison Patterns
  + URL: <https://archive.ics.uci.edu/ml/datasets/Record+Linkage+Comparison+Patterns>
  + Instances: 5749132
  + Size: 54 MB