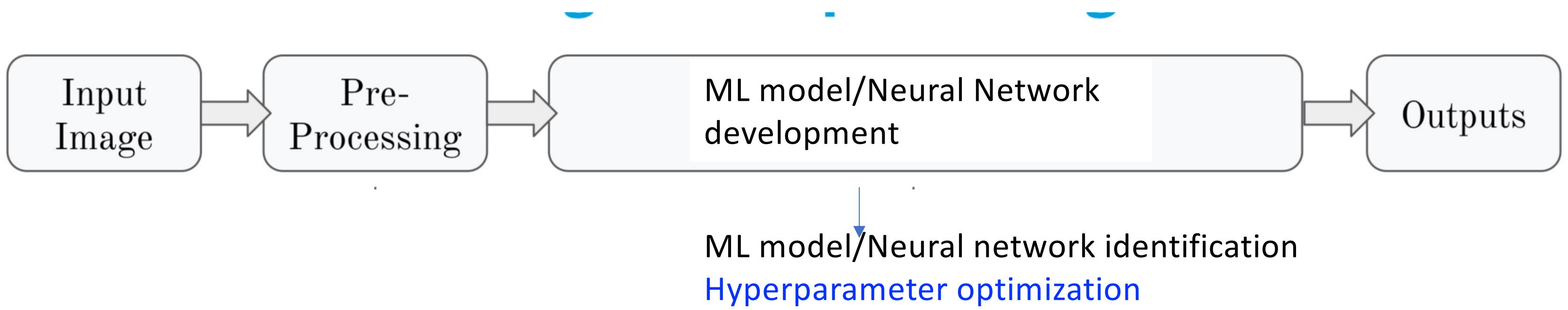
DS-UA 301
Advanced Topics in Data Science
Advanced Techniques in ML and Deep Learning

LECTURE 9

Parijat Dube

Automated Machine Learning



Input Image

Pre-Processing

ML model/Neural Network development

Outputs

ML model/Neural network identification

Hyperparameter optimization

- Data Preprocessing: normalization, data-augmentation
- Neural architecture search (NAS)
  - Standard, off the shelf
  - Synthesize a new
    - Types of layers (conv, maxpool), number of different layers, how to stack the layers
    - Convolution layer parameters (filter dim, stride dim)
- Hyperparameter optimization
  - Batch size, learning rate, momentum
- NAS and hyperparameter optimization can be done jointly or sequentially

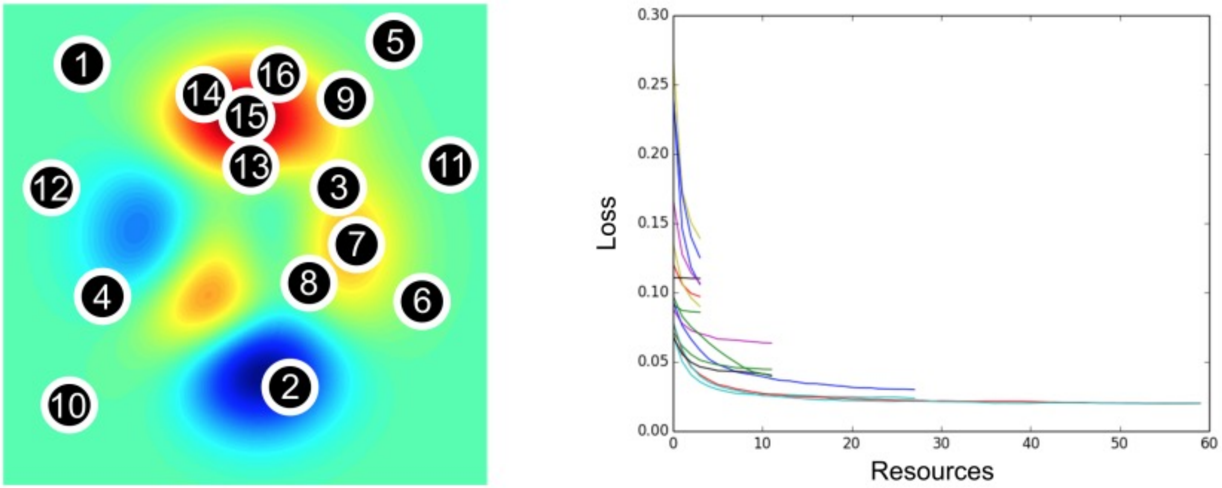
Automated Machine Learning

- IBM Auto AI
- Run a sample AutoAI experiment in IBM WML
- H20 AutoML

Hyperparameter Optimization

Hyperparameter optimization

- Problem of identifying good hyperparameter configuration(s) from the set of possible configurations
- Two sub-problems
  - Configuration selection
    - Efficient selection of good configuration
  - Configuration evaluation
    - Adaptive computation, allocating more resources to promising hyperparameter configurations while eliminating poor ones.



(a) Configuration Selection

(b) Configuration Evaluation

Figure 1: (a) The heatmap shows the validation error over a two-dimensional search space with red corresponding to areas with lower validation error. Configuration selection methods adaptively choose new configurations to train, proceeding in a sequential manner as indicated by the numbers. (b) The plot shows the validation error as a function of the resources allocated to each configuration (i.e. each line in the plot). Configuration evaluation methods allocate more resources to promising configurations.

Mathematical formulation

$$\lambda^{(*)} = \operatorname*{argmin}_{\lambda \in \Lambda} \mathbb{E}_{x \sim \mathcal{G}_x} [\mathcal{L}\left(x; \mathcal{A}_\lambda(\mathcal{X}^{(\text{train})})\right)]$$

$$\begin{aligned} \lambda^{(*)} &\approx \operatorname*{argmin}_{\lambda \in \Lambda} \operatorname*{mean}_{x \in \mathcal{X}^{(\text{valid})}} \mathcal{L}\left(x; \mathcal{A}_\lambda(\mathcal{X}^{(\text{train})})\right) \\ &\equiv \operatorname*{argmin}_{\lambda \in \Lambda} \Psi(\lambda) \\ &\approx \operatorname*{argmin}_{\lambda \in \{\lambda^{(1)} \ldots \lambda^{(S)}\}} \Psi(\lambda) \equiv \hat{\lambda} \end{aligned}$$

critical step is to choose the set of trials $\{\lambda^{(1)} \ldots \lambda^{(S)}\}$

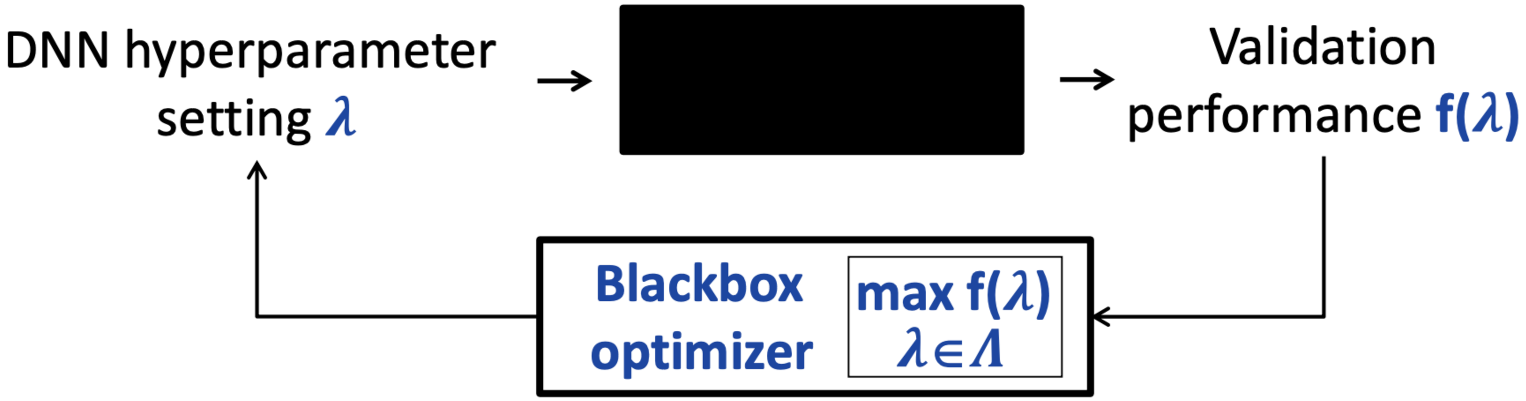
Types of Hyperparameters

- Continuous
  - Example: learning rate
- Integer
  - Example: #units
- Categorical
  - Finite domain, unordered
    - Example 1: algo ∈ {SVM, RF, NN}
    - Example 2: activation function ∈ {ReLU, Leaky ReLU, tanh}
    - Example 3: operator ∈ {conv3x3, separable conv3x3, max pool, …}
  - Special case: binary

Conditional Hyperparameters

- Conditional hyperparameters B are only active if other hyperparameters A are set a certain way
  - Example 1:
    - A = choice of optimizer (Adam or SGD)
    - B = Adam‘s second momentum hyperparameter (only active if A=Adam)
  - Example 2:
    - A = type of layer k (convolution, max pooling, fully connected, ...)
    - B = conv. kernel size of that layer (only active if A = convolution)
  - Example 3:
    - A = choice of classifier (RF or SVM)
    - B = SVM‘s kernel parameter (only active if A = SVM)

Blackbox Hyperparameter Optimization



DNN hyperparameter setting λ

Validation performance $\mathbf{f(\lambda)}$

Blackbox optimizer

$$\max_{\lambda \in \Lambda} f(\lambda)$$

- The blackbox function is expensive to evaluate → sample efficiency is important

Techniques for Hyperparameter Optimization

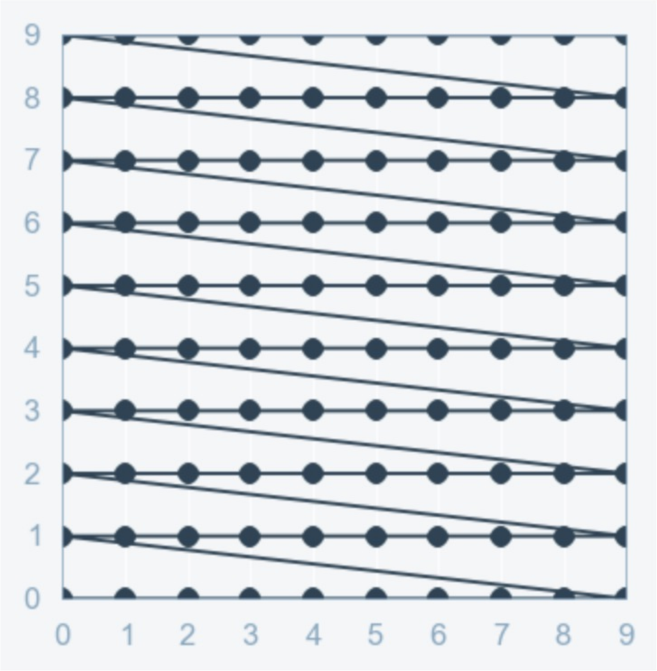
- Grid search
- Random search
- Hyperband : random configuration search with adaptive resource allocation
- Bayesian optimization methods
  - Focus on configuration selection
  - Identify good configurations more quickly than standard baselines like random search by selecting configurations in an adaptive manner
- Bayesian optimization with adaptive resource allocation

Grid Search

- Every combination of a preset list of values of the hyper-parameters and evaluate the model for each combination
- Let K be the number of hyperparameters
- grid search requires that we choose a set of values for each variable $(L^{(1)} \ldots L^{(K)})$
- Number of configurations to consider

$$S = \prod_{k=1}^{K} |L^{(k)}|$$

- Suffers from curse of dimensionality

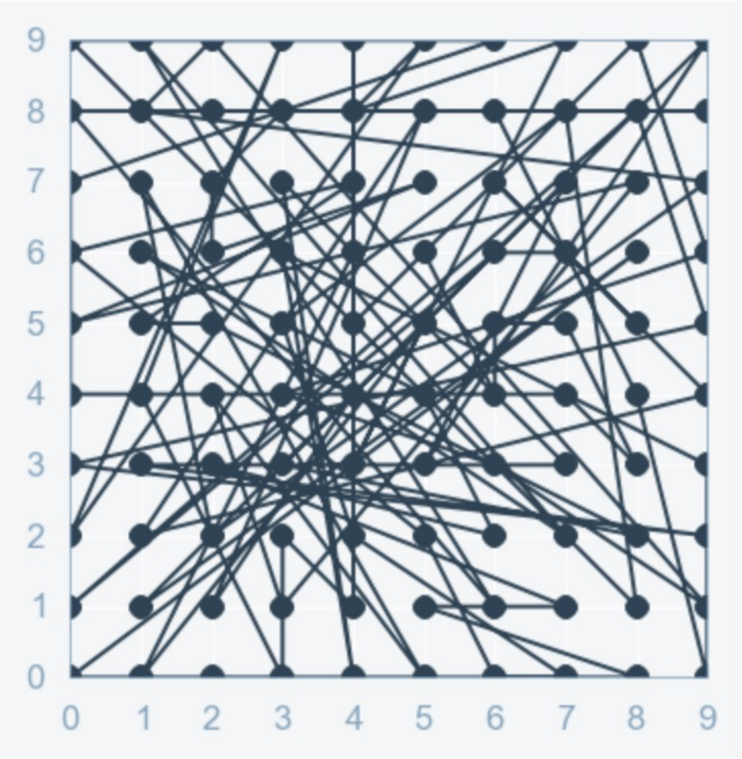


Visual Representation of grid search

Random Search

- Technique where random combinations of the hyperparameters are used to find the best solution for the built model

- Empirically and theoretically shown that random search is more efficient for parameter optimization than grid search.



Visual Representation of Random search

Grid Search Vs Random Search

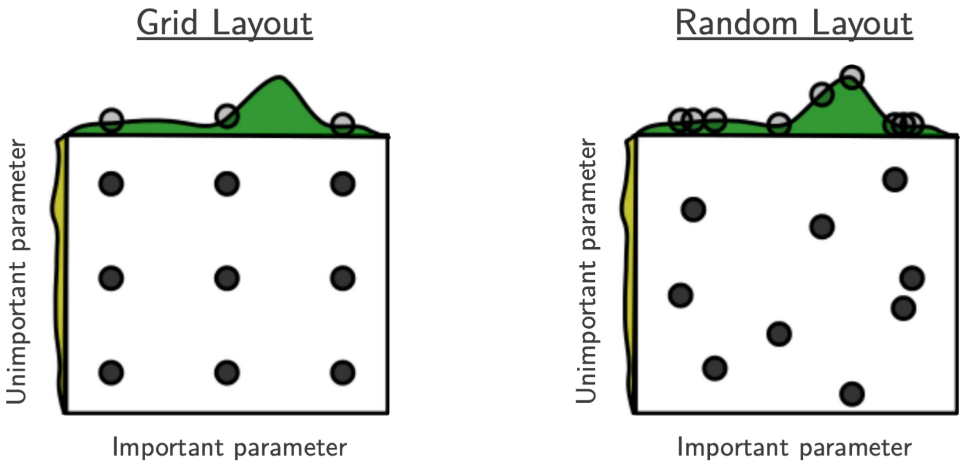


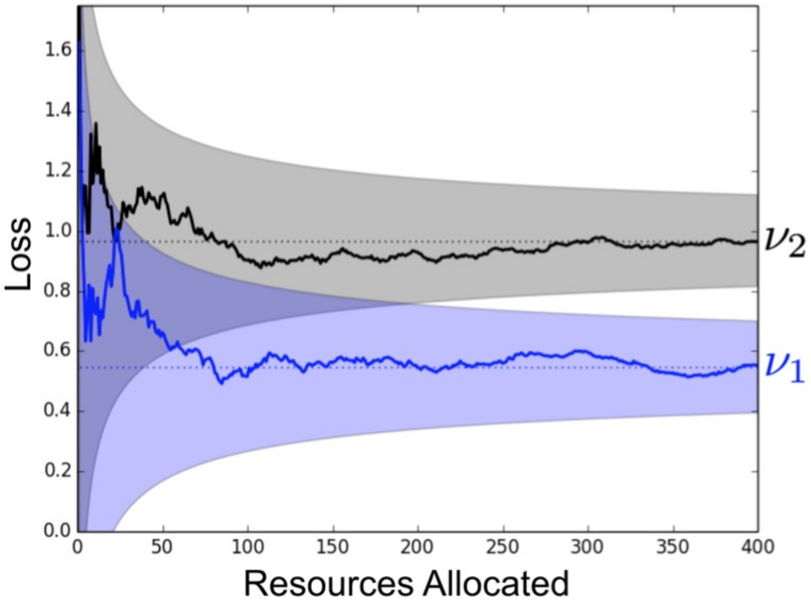
Figure 1: Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

Training resources

- Size of training set
- Number of features
- Number of iterations for iterative algorithms
- Hours of training time

Validation loss vs Resource allocated

Validation loss as a function of total resources allocated for two configurations with terminal validation losses ν_1 and ν_2



The shaded areas bound the maximum distance of the intermediate losses from the terminal validation loss and monotonically decrease with the resource.

Possible to distinguish between the two configurations when the envelopes no longer overlap

More resources are needed to differentiate between the two configurations when either

(1) the envelope functions are wider or

(2) the terminal losses are closer together.

Successive Halving

- Underlying principle: Even if performance after a small number of iterations is very unrepresentative of the *absolute* performance of any configuration, its *relative* performance compared with many alternatives trained with the same number of iterations is roughly maintained.
  - Relative ordering among configurations converges much faster than their true values
- 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
- Allocates exponentially more resources to more promising configurations
- Given some finite budget B and n configurations, it is not clear a priori whether we should
  - Consider many configurations (large n) with a small average training resources; or
  - Consider a small number of configurations (small n) with longer average training resources.
- Successive Halving suffers from the "*n vs B/n*" trade-off

Li et al. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. 2018

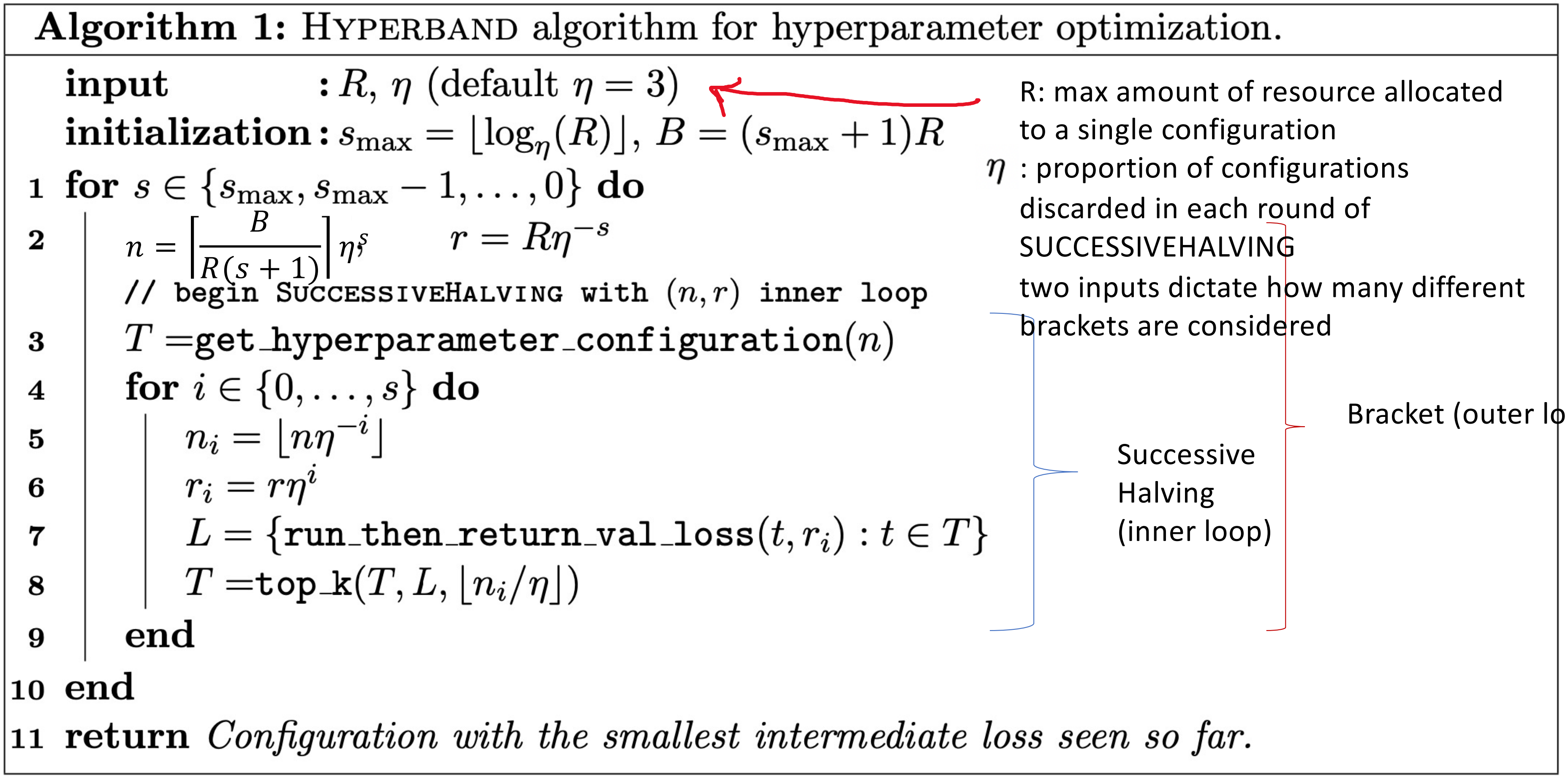
n Vs. B/n

- Consider a simple strategy
  - If hyper-parameter configurations can be discriminated quickly
    - *n* should be chosen large
  - If hyper-parameter configurations are slow to differentiate
    - *B/n* should be large
- Drawbacks of the simple strategy
  - If *n* is large, then some good configurations which can be slow to converge at the beginning will be killed off early.
  - If *B/n* is large, then bad configurations will be given a lot of resources, even though they could have been stopped before.

Hyperband

- Formulating hyperparameter optimization as a pure-exploration adaptive resource allocation problem addressing *how to allocate resources among randomly sampled hyperparameter configurations*.
- *Considers several possible values of n for a fixed B, in essence performing a grid search over feasible value of n*
- Hedges and loops over varying degrees of the aggressiveness balancing breadth versus depth based search.
- Resource constrained hyperparameter optimization

Li et al. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. 2018



Algorithm 1: Hyperband algorithm for hyperparameter optimization.

```
input          : $R$, $\eta$ (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
    $n = \left\lceil \frac{B}{R(s+1)} \right\rceil \eta^s$        $r = R\eta^{-s}$
    // begin SuccessiveHalving with $(n, r)$ inner loop
    $T$ =get_hyperparameter_configuration$(n)$
    for $i \in \{0, \ldots, s\}$ do
        $n_i = \lfloor n\eta^{-i} \rfloor$
        $r_i = r\eta^i$
        $L = \{$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.
```

R: max amount of resource allocated to a single configuration

η : proportion of configurations discarded in each round of SUCCESSIVEHALVING

two inputs dictate how many different brackets are considered

Bracket (outer loop)

Successive Halving (inner loop)

Hyperband

- **`get_hyperparameter_configuration(n):`** returns a set of *n* i.i.d samples from some distribution defined over the hyperparameter configuration space. Uniformly sample the hyperparameters from a predefined space (hypercube with min and max bounds for each hyperparameter).

- **`run_then_return_val_loss(t, r):`** a function that takes a hyperparameter configuration t and resource allocation *r* as input and returns the validation loss after training the configuration for the allocated resources.

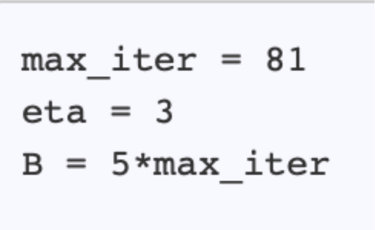
- **`top_k(configs, losses, k):`** a function that takes a set of configurations as well as their associated losses and returns the top *k* performing configurations.

Hyperband: Sweeping over different s

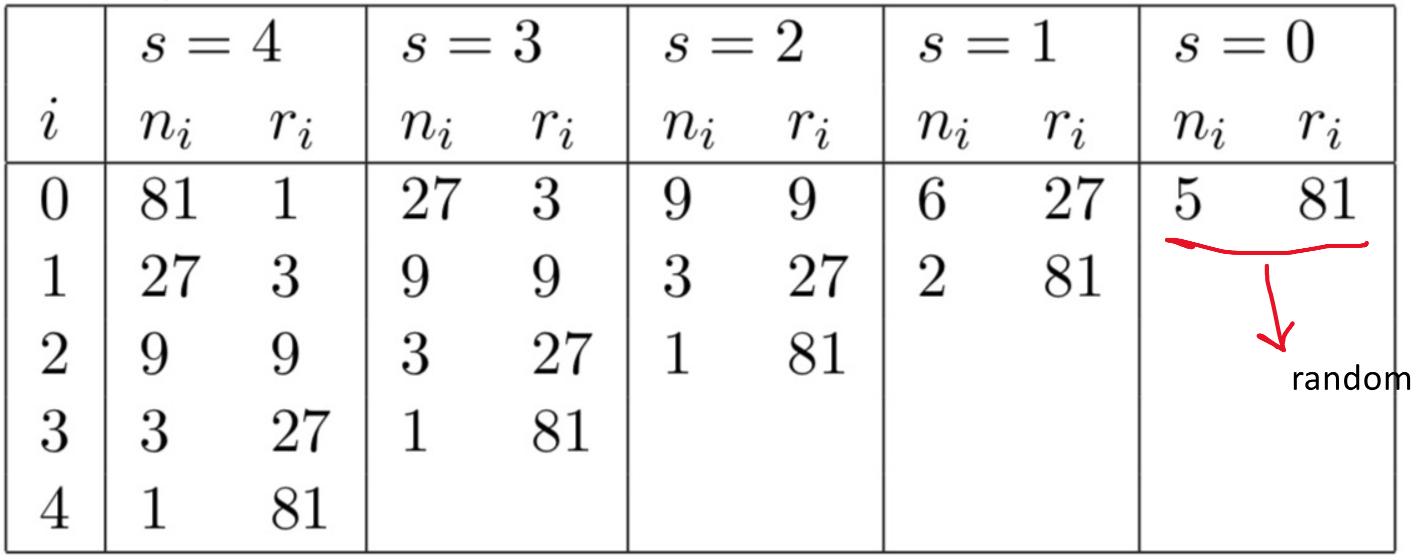
- Each inner loop indexed by s is designed to take B total iterations
- Each value of s takes about the same amount of time on average.
- For large values of s the algorithm considers many configurations (max_iter at the most) but discards hyperparameters based on just a very small number of iterations which may be undesirable for hyperparameters like the learning_rate.
- For small values of s the algorithm will not throw out hyperparameters until after many iterations have been performed but fewer configurations are considered (logeta(max_iter)+1 at the least). The outerloop hedges over all possibilities.

Hyperband in action

Each bracket uses B total resources and corresponds to a different tradeoff of n and B/n



```
max_iter = 81
eta = 3
B = 5*max_iter
```



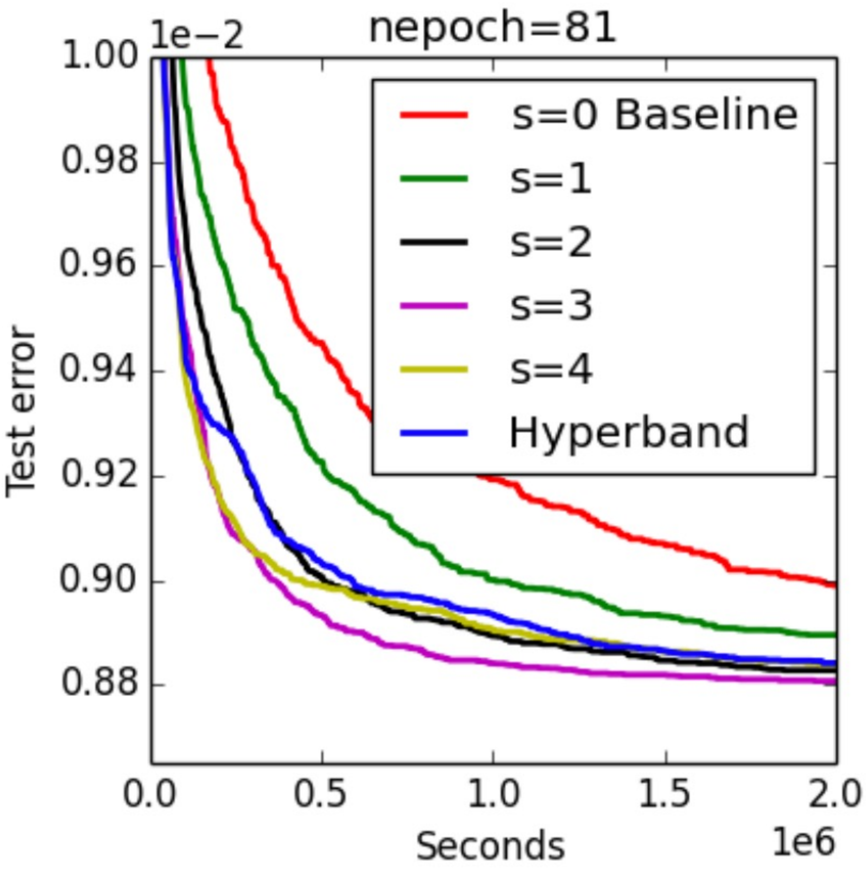
	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

random search

A larger value of n corresponds to a smaller r and hence more aggressive early-stopping
A single execution of Hyperband takes a finite budget of $(s_{max} + 1)B$

Hyperband Performance

- Resource allocated to each configuration to be number of iterations of SGD, with one unit of resource corresponding to one epoch, i.e., a full pass over the data set



https://homes.cs.washington.edu/~jamieson/hyperband.html

AutoML as Hyperparameter Optimization

Definition: Combined Algorithm Selection and Hyperparameter Optimization (CASH)

Let

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

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

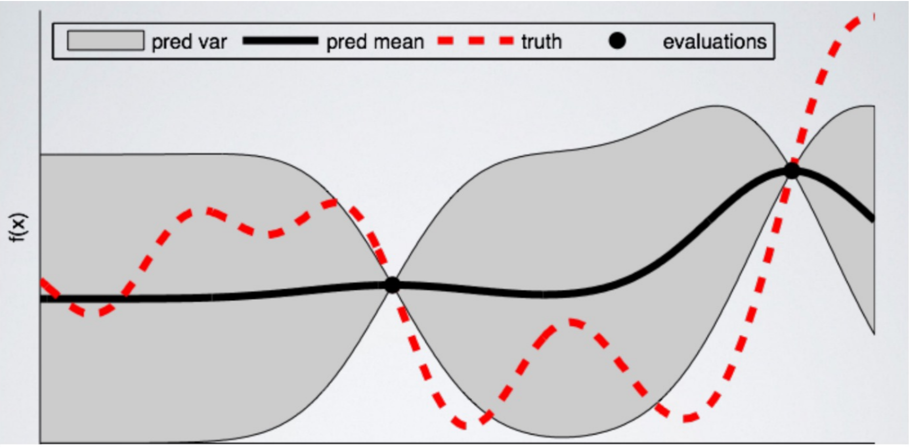
$$A^*_{\boldsymbol{\lambda}^*} \in \operatorname*{arg\,min}_{A^{(i)} \in \mathcal{A}, \boldsymbol{\lambda} \in \boldsymbol{\Lambda}^{(i)}} \mathcal{L}(A^{(i)}_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$$

Bayesian Hyperparameter Optimization

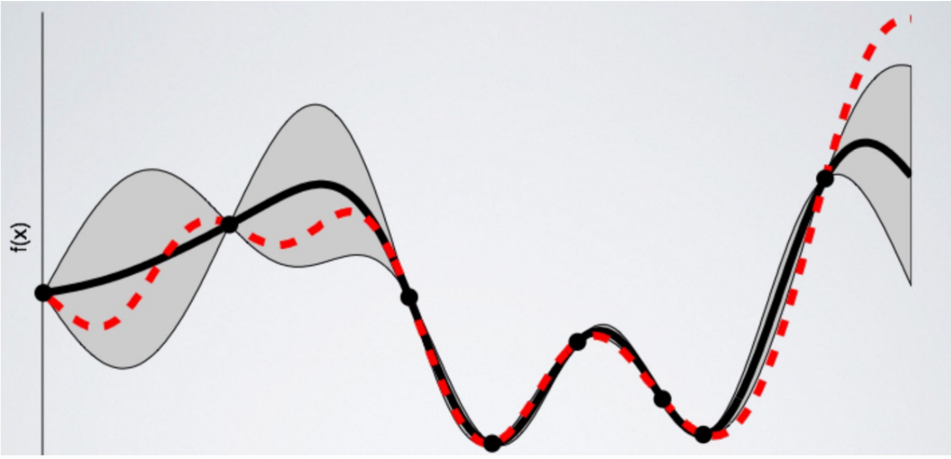
- Idea: Build a probability model of the objective function and use it to select the most promising hyperparameters to evaluate in the true objective function.
- Approach
  1. Fit a proabilistic model to the function evaluations $\langle\lambda, f\ (\lambda)\ \rangle$
  2. Use that model to trade off exploration vs. exploitation
- Surrogate probability model for the objective function: p(score|hyperparameters)
- Steps:
  1. Build a surrogate probability model of the objective function
  2. Find the hyperparameters that perform best on the surrogate
  3. Apply these hyperparameters to the true objective function
  4. Update the surrogate model incorporating the new results
  5. Repeat steps 2–4 until max iterations or time is reached
- The surrogate probability model after each evaluation of the objective function

Surrogate model performance

After 2 evaluations



After 8 evaluations

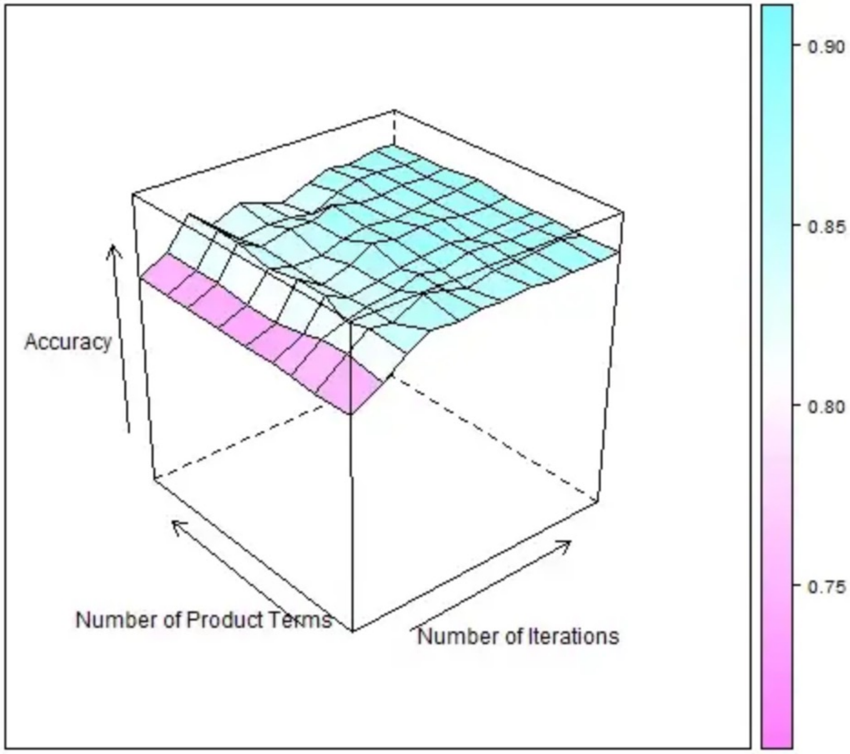


Sequential Model-Based Optimization (SMBO)

- Running trials one after another, each time trying better hyperparameters by applying Bayesian reasoning and updating a probability model (surrogate)
- Main components
  1. A domain of hyperparameters over which to search
  2. An objective function which takes in hyperparameters and outputs a score that we want to minimize (or maximize)
  3. The surrogate model of the objective function
  4. A criteria, called a selection function, for evaluating which hyperparameters to choose next from the surrogate model
  5. A history consisting of (score, hyperparameter) pairs used by the algorithm to update the surrogate model

Surrogate model

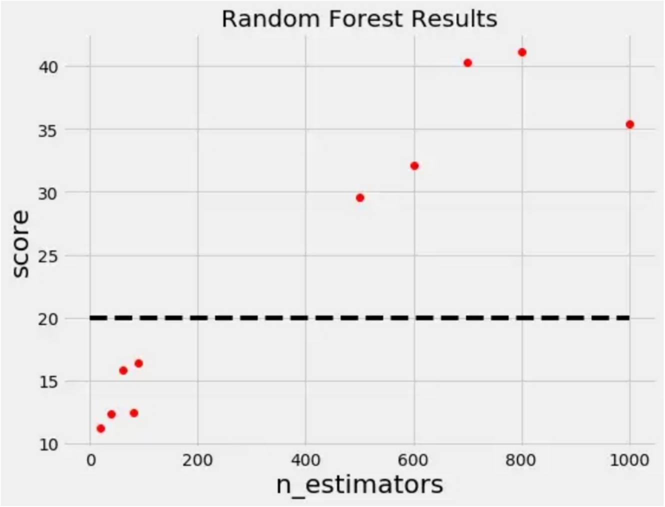
- Gaussian Processes
- Random Forest Regressions
- Tree Parzen Estimators (TPE)

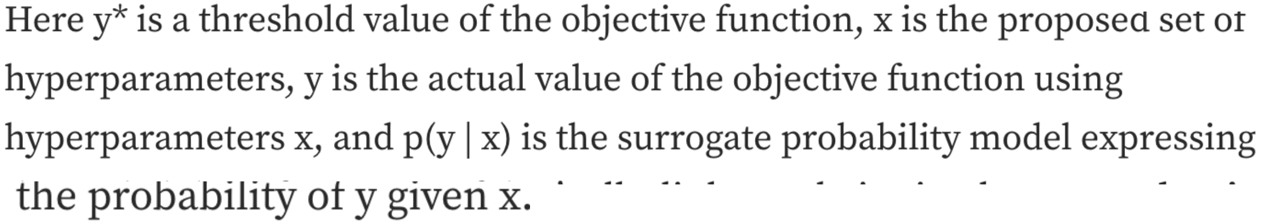


Selection Function

- Expected Improvement

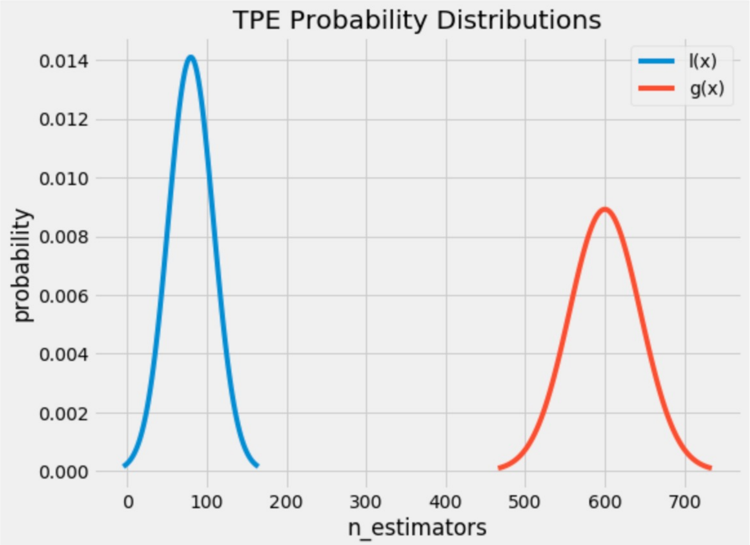
$$\mathrm{EI}_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y) p(y|x) dy$$




Here y* is a threshold value of the objective function, x is the proposed set of hyperparameters, y is the actual value of the objective function using hyperparameters x, and p(y | x) is the surrogate probability model expressing the probability of y given x.

Tree structured Parzen estimator

$$p(y|x) = \frac{p(x\,|y) * p(y)}{p(x)}$$



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

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