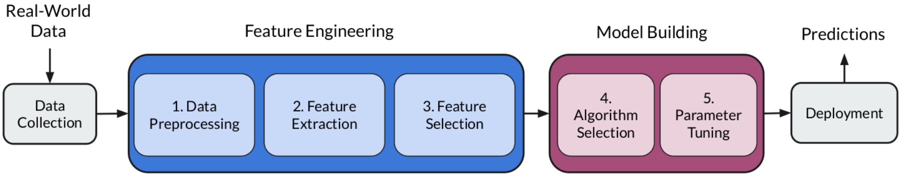
Automated Machine Learning (AutoML)

Slides by Amir H. Payberah
payberah@kth.se

The Machine Learning Process

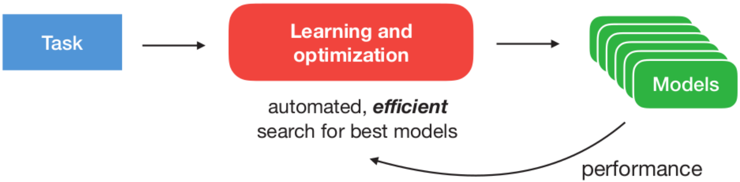
- Building an ML model is an iterative, complex, and time-consuming process.
- It can take a lot of trial and error.



[Elshawi et al., Automated Machine Learning: State-of-The-Art and Open Challenges, 2019]

Automated vs. Manual Machine Learning

- AutoML: build models in a data-driven, intelligent, and purposeful way.

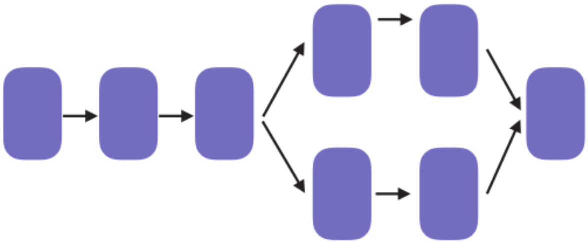


[Joaquin Vanschoren, Automatic Machine Learning - A Tutorial]


Search space design
Architecture search strategy
arch
score
Performance estimation strategy

AutoML Subproblems - Neural Architecture Search

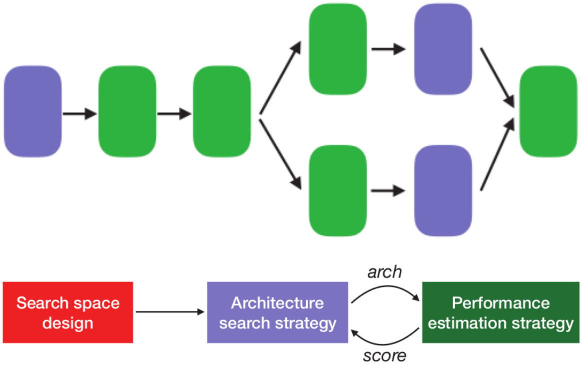
- Represent and search all pipelines or neural nets, e.g., neural layers, interconnections, etc.



[Joaquin Vanschoren, Automatic Machine Learning - A Tutorial]

AutoML Subproblems - Hyperparameter Optimization

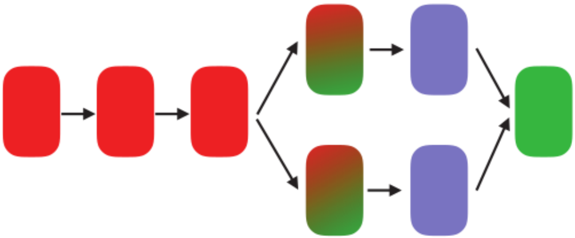
- Which hyperparameters are important? How to optimize them?



[Joaquin Vanschoren, Automatic Machine Learning - A Tutorial]

AutoML Subproblems - Meta-learning

- How can we transfer experience from previous tasks?
- Don't start from scratch (search space is too large).



[Joaquin Vanschoren, Automatic Machine Learning - A Tutorial]

Meta-learning Neural Architecture Search Hyperparameter Optimization

Hyper-Parameter Optimization (HPO)

AutoML Definition

- $\mathtt{A}$ denotes a ML algorithm with $\mathtt{m}$ hyperparameters.
- $\{\mathtt{A}_1, \mathtt{A}_2, \cdots, \mathtt{A}_n\}$ is a set of ML algorithms.
- Λ_j is the domain of $\mathtt{j}$th hyperparameter.
- $\boldsymbol{\Lambda} = \Lambda_1 \times \Lambda_2 \times \cdots \times \Lambda_m$ is the overall hyperparameter configuration space.
- $\theta \in \boldsymbol{\Lambda}$ is a vector of hyperparameters.
- $\mathtt{J}(\theta, \mathbf{X}_{\mathtt{train}}, \mathbf{X}_{\mathtt{valid}})$ is the loss of the ML model created by θ, trained on $\mathbf{X}_{\mathtt{train}}$, and validated on $\mathbf{X}_{\mathtt{valid}}$.
- Find the configuration that minimizes the expected loss on a dataset $\mathbf{X}_{\mathtt{train}}$:
  $\theta^* = \arg\min_{\theta \in \boldsymbol{\Lambda}} \mathbb{E}_{(\mathbf{X}_{\mathtt{train}}, \mathbf{X}_{\mathtt{valid}}) \sim \mathbf{X}} \mathtt{J}(\theta, \mathbf{X}_{\mathtt{train}}, \mathbf{X}_{\mathtt{valid}})$

Types of Hyperparameters

- Continuous
  - E.g., learning rate
- Integer
  - E.g., number of hidden units
- Categorical
  - E.g., choice of operator (Convolution, MaxPooling, DropOut, etc.)
  - E.g., choice of activation function (ReLU, Leaky ReLU, tanh, etc.)
- Conditional
  - E.g., convolution kernel size, if convolution layer is selected

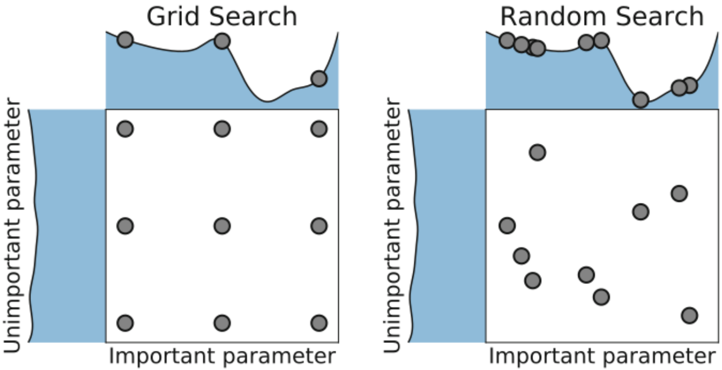
Hyper-Parameter Optimization

- Black-box optimization
  - Grid search
  - Random search
  - Population-based search
  - Bayesian optimization
- Multi-fidelity optimization
  - Modeling learning curve
  - Bandit based

Hyper-Parameter Optimization

- Black-box optimization
  - Grid search
  - Random search
  - Population-based search
  - Bayesian optimization
- Multi-fidelity optimization
  - Modeling learning curve
  - Bandit based

Black-box Optimization - Grid and Random Search



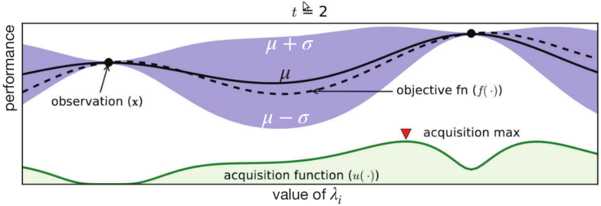
[Hutter et al., Automated Machine Learning, 2019]

Black-box Optimization - Population-based Search

- They maintain a population, i.e., a set of configurations.
- Improve this population to obtain a new generation of better configurations.
- Achieve this by applying:
  - Local perturbations (so-called mutations)
  - Combinations of different members (so-called crossover)
- E.g., genetic algorithms, evolutionary algorithms, particle swarm optimization

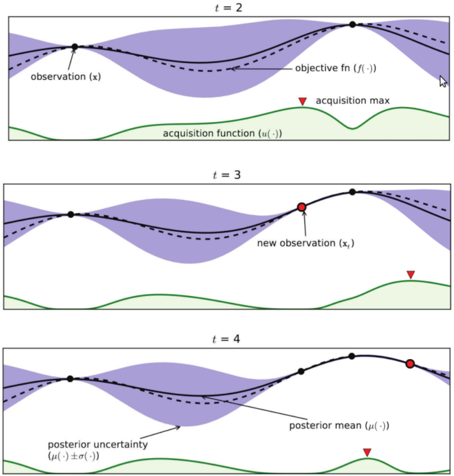
Black-box Optimization - Bayesian Optimization (1/3)

- Start with a few (random) hyperparameter configurations.
- Fit a surrogate model to predict other configurations.
- An acquisition function drives the proposition of new points to test, in an exploration and exploitation trade-off.
- Sample for the best configuration under that function.



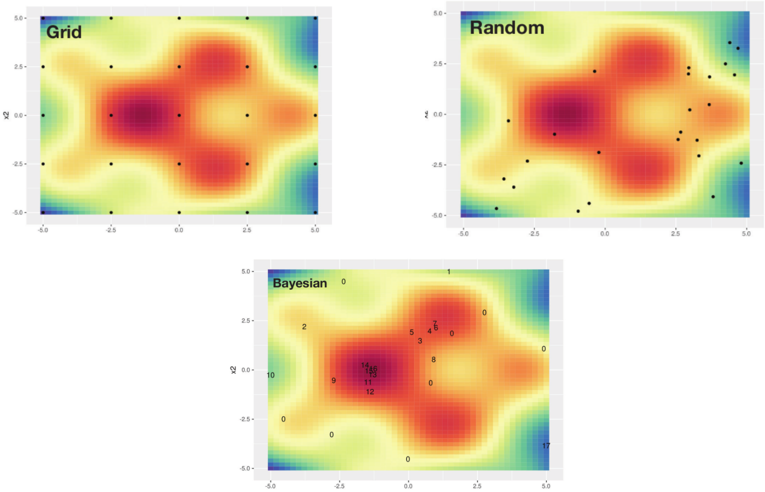
[Hutter et al., Automated Machine Learning, 2019]

Black-box Optimization - Bayesian Optimization (2/3)



[Hutter et al., Automated Machine Learning, 2019]

Black-box Optimization - Bayesian Optimization (3/3)



[Hutter et al., Automated Machine Learning, 2019]

Hyper-Parameter Optimization

- Black-box optimization
  - Grid search
  - Random search
  - Population-based search
  - Bayesian optimization
- Multi-fidelity optimization
  - Modeling learning curve
  - Bandit based

Multi-fidelity Optimization

- Massive dataset sizes and complex models make blackbox performance evaluation expensive.
- Probe a hyperparameter configuration on a small subset.
- Multi-fidelity methods use low fidelity approximations of the actual loss function to minimize.
- These approximations introduce a tradeoff between optimization performance and runtime.

Multi-fidelity Optimization - Modeling Learning Curves

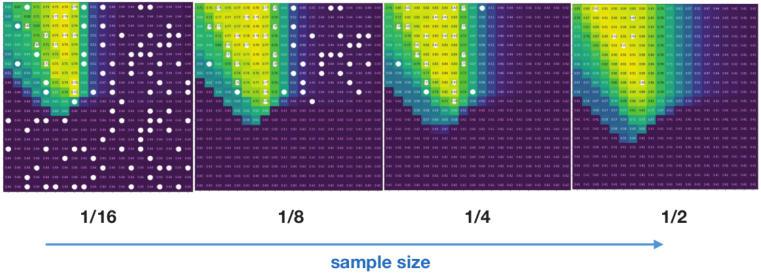
- Learning curve extrapolation is used in predicting early termination for a particular configuration.
- Models learning curves during hyper-parameter optimization.
- Decides whether to allocate more resources or to stop the training procedure for a particular configuration.
- The learning process is terminated if the performance of the predicted configuration is less than the performance of the best model trained so far in the optimization process.

Multi-fidelity Optimization - Bandit-Based

- Successive halving algorithm (SHA)
- HyperBand

Multi-fidelity Optimization - SHA (1/4)

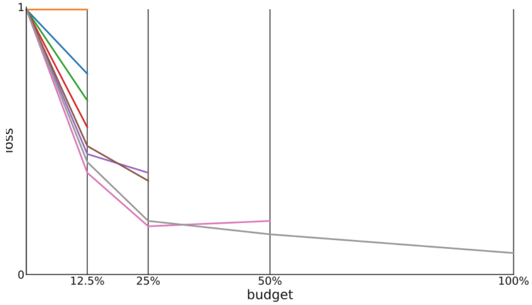
- Train on small subsets, infer which regions may be interesting to evaluate in more depth.
- Randomly sample candidates and evaluate on a small data sample.
- E.g., retrain the 50% best candidates on twice the data.



[Hutter et al., Automated Machine Learning, 2019]

Multi-fidelity Optimization - SHA (2/4)

- Successive halving for eight algorithms/configurations.
- After evaluating all algorithms on 1/8 of the total budget, half of them are dropped and the budget given to the remaining algorithms is doubled.



[Hutter et al., Automated Machine Learning, 2019]

Multi-fidelity Optimization - SHA (3/4)

SUCCESSIVEHALVING (Finite horizon)

input: Budget B, and n arms where $\ell_{i,k}$ denotes the kth loss from the ith arm, maximum size R, $\eta \geq 2$ ($\eta = 3$ by default).

Initialize: $S_0 = [n]$, $s = \min\{t \in \mathbb{N} : nR(t+1)\eta^{-t} \leq B, t \leq \log_\eta(\min\{R, n\})\}$.

For $k = 0, 1, \ldots, s$

Set $n_k = \lfloor n\eta^{-k} \rfloor$, $r_k = \lfloor R\eta^{k-s} \rfloor$

Pull each arm in S_k for r_k times.

Keep the best $\lfloor n\eta^{-(k+1)} \rfloor$ arms in terms of the r_kth observed loss as S_{k+1}.

Output : $\hat{\imath}, \ell_{\hat{\imath},R}$ where $\hat{\imath} = \arg\min_{i \in S_{s+1}} \ell_{i,R}$

Multi-fidelity Optimization - SHA (4/4)

- Successive halving suffers from the budget-vs-number of configurations trade off.
- Given a total budget, the user has to decide beforehand whether:
  - to try many configurations and only assign a small budget to each, or
  - to try only a few and assign them a larger budget.
- Assigning too small a budget can result in prematurely terminating good configurations.
- Assigning too large a budget can result in running poor configurations too long and thereby wasting resources.

Multi-fidelity Optimization - HyperBand (1/2)

- HyperBand combats SHA problem when selecting from randomly sampled configurations.
- It divides the total budget into several combinations of number of configurations vs. budget for each.
- Then it calls SHA on each set of random configurations.

Multi-fidelity Optimization - HyperBand (2/2)

Algorithm 1: HYPERBAND algorithm for hyperparameter optimization.

input : R, η (default $\eta = 3$)
initialization : $s_{\max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\max} + 1)R$

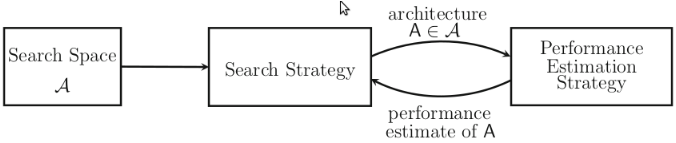
for $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ **do**
  $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil, \qquad r = R\eta^{-s}$
  `// begin SuccessiveHalving with` (n, r) `inner loop`
  $T =$ `get_hyperparameter_configuration`(n)
  for $i \in \{0, \ldots, s\}$ **do**
    $n_i = \lfloor n\eta^{-i}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

- The inner loop invokes SHA for fixed values of `n` and `r`.
- The outer loop iterates over different values of `n` and `r`.

Neural Architecture Search (NAS)

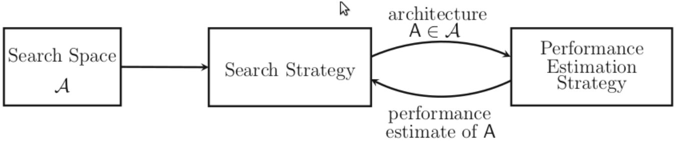
Neural Architecture Search

- The process of automating architecture engineering.
- Search space: which architectures can be represented in principle.
- Search strategy: how to explore the search space.
- Performance estimation: to perform a standard training and validation of the architecture on data.



[Hutter et al., Automated Machine Learning, 2019]

Search Space



Search Space

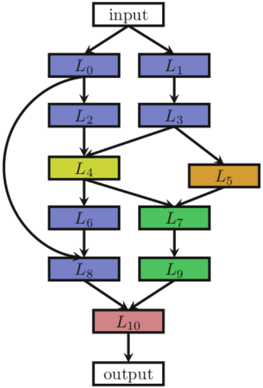
- Which neural architectures a NAS approach might discover.
- Chain-structured neural network
- Multi-branch networks
- Repeated motifs

Chain-Structured Neural Network

- A sequence of $\mathtt{n}$ layers.
- The $\mathtt{i}$'th layer $\mathtt{L_i}$ receives its input from layer $\mathtt{i}-1$ and its output serves as the input for layer $\mathtt{i}+1$.
- Parameters of the search space:
  - The (maximum) number of layers $\mathtt{n}$.
  - The type of operation every layer can execute, e.g., pooling, conv.
  - Hyperparameters associated with the operation, e.g., number of filters, kernel size and strides for a convolutional layer.

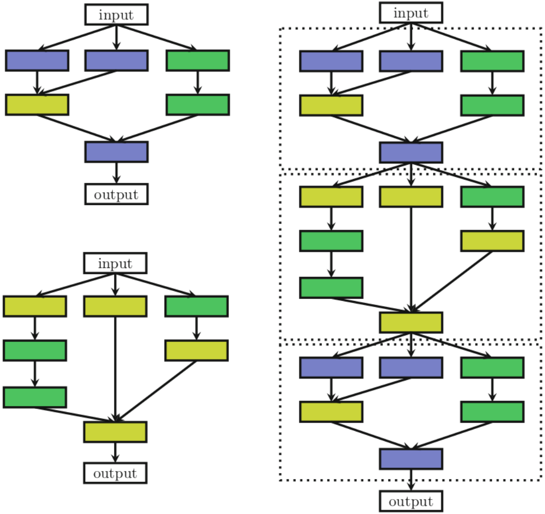
Multi-Branch Networks

- The input of layer $\mathtt{i}$: a function $\mathtt{g_i}(\mathtt{L^{out}_{i-1}}, \cdots, \mathtt{L^{out}_0})$ of previous layer outputs.
- Special cases:
  - The chain-structured networks: $\mathtt{g_i}(\mathtt{L^{out}_{i-1}}, \cdots, \mathtt{L^{out}_0}) = \mathtt{L^{out}_{i-1}}$
  - Residual networks, where previous layer outputs are summed: $\mathtt{g_i}(\mathtt{L^{out}_{i-1}}, \cdots, \mathtt{L^{out}_0}) = \mathtt{L^{out}_{i-1}} + \mathtt{L^{out}_j}, \mathtt{j} < \mathtt{i}$
  - DenseNets, where previous layer outputs are out concatenated: $\mathtt{g_i}(\mathtt{L^{out}_{i-1}}, \cdots, \mathtt{L^{out}_0}) = \mathtt{concat}(\mathtt{L^{out}_{i-1}}, \cdots, \mathtt{L^{out}_0})$

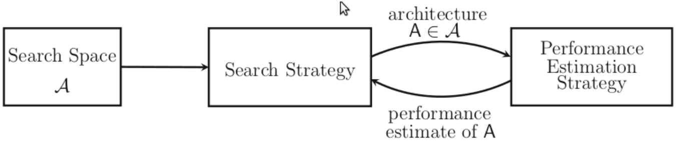


Repeated Motifs

- Normal cell: preservers the dimensionality of the input.
- Reduction cell: reduces the spatial dimension.



Search Strategy



Search Strategy

- Random search
- Reinforcement learning
- Gradient-based optimization
- Bayesian optimization
- Evolutionary methods

Random Search

- For each node in the DAG, determine what decisions must be made.
  - Choose a node as input and a corresponding operation to apply to generate the output of the node.
  - E.g., node `i` can take the outputs of nodes `0` to node `i` $-$ `1` as input.
  - E.g., choose an operation, e.g., tanh, relu, sigmoid to apply to the output of node `i`.
- Sample uniformly from the set of possible choices for each decision that needs to be made.
- Moving from node to node.

[Li et al., Random Search and Reproducibility for Neural Architecture Search, 2020]

Evolutionary Methods

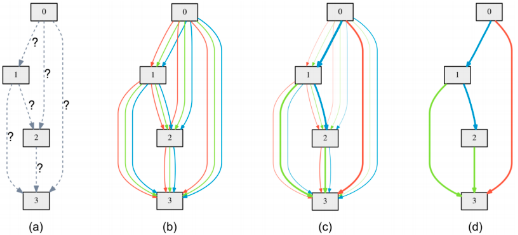
- Evolves a population of models, i.e., a set of (possibly trained) networks.
- In every evolution step, at least one model from the population is sampled and serves as a parent to generate offsprings by applying mutations to it.
  - E.g., adding or removing a layer, altering the hyperparameters of a layer, adding skip connections, etc.
- After training the offsprings, their fitness (e.g., performance on a validation set) is evaluated and they are added to the population.
- Evolutionary methods differ in how they sample parents, update populations, and generate offsprings.

Reinforcement Learning

- Action: the generation of a neural architecture.
- Action space: the search space.
- Reward: based on an estimate of the performance of the trained architecture on unseen data.
- Policy: different approaches.

Gradient-based Optimization

- The previous methods search over a discrete set of candidate architectures.
- Here, it relaxes the search space to be continuous, so that the architecture can be optimized with respect to its validation set performance by gradient descent.
- We relax the categorical choice of a particular operation to a softmax over all possible operations.



[Liu et al., DARTS: Differentiable Architecture Search, 2019]

Bayesian Optimization (1/3)

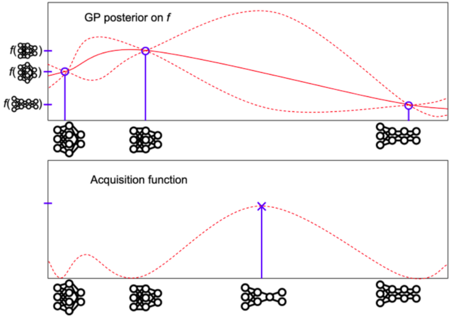
- Find the architecture $\mathtt{a} \in \mathtt{A}$ that maximizes $\mathtt{f}(\mathtt{a})$.
- Choose several architectures from $\mathtt{A}$ at random and evaluating $\mathtt{f}(\mathtt{a})$ for each of them.
- Based on these results, iteratively choose new architectures to evaluate.
- The full algorithm: $\mathtt{T}$ rounds of choosing an architecture $\mathtt{a}_\mathtt{i}$ and computing $\mathtt{f}(\mathtt{a}_\mathtt{i})$.
- The output is the architecture $\mathtt{a}^*$ with the largest value of $\mathtt{f}(\mathtt{a}^*)$ among all those that were tried in the previous rounds.

Bayesian Optimization (2/3)

- Choose the next architecture in round $i+1$, given $f(a_1), \cdots, f(a_i)$.
- Assume $f : A \to [0, 1]$ follows a Gaussian Process (GP).
- Makes an assumption about the distribution $f(A)$.
- The assumptions about the mean and variance of $f(A)$ are constantly being updated as the algorithm gathers more data in the form of $f(a_1), \cdots, f(a_i)$.
- Chooses the architecture with the greatest chance of giving a large improvement.
- The algorithm chooses $a_{i+1} = \arg\max_{a \in A} \max(0, E[f(a) - f^*]) = \arg\max_{a \in A} E[f(a)]$.
- f^* is the best accuracy observed so far.

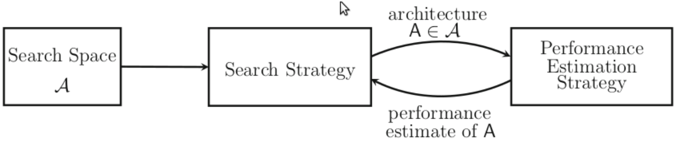
Bayesian Optimization (3/3)

- The top graph: three evaluations of `f` (blue circles), an estimate of `f` (solid red line), and confidence intervals (dotted red lines).
- The bottom graph: the expected improvement value for each architecture. The architecture with the largest expected improvement is chosen (blue `x`).



[https://medium.com/abacus-ai/an-introduction-to-bayesian-optimization-for-neural-architecture-search-d324830ec781]

Performance Estimation



Performance Estimation

- The search strategies need to estimate the performance of a given architecture $\mathtt{A}$ they consider.
- The simplest way of doing this is to train $\mathtt{A}$ on training data and evaluate its performance on validation data.
- However, training each architecture to be evaluated from scratch frequently yields computational demands in the order of thousands of GPU days for NAS.

Reduce the Computational Burden

- Low-fidelity approximation
- Learning curve extrapolation
- One-shot architecture

Meta-Learning

Meta-Learning

- Meta-learning or learning to learn
- Systematically observe how different ML approaches perform on a wide range of learning tasks.
- Then, learning from this experience (meta-data), to learn new tasks much faster than otherwise possible.

Meta-Learning

- Learning from task properties
  - Using meta-features
  - Building meta-models
- Learning from model evaluation
  - Relative landmarks
  - Surrogate model
  - Warm-started multi-task learning
- Learning from prior models
  - Transfer learning
  - Few-shot learning

Learning from Task Properties

Learning from Task Properties (1/3)

- Uses meta-features:
  - Number of instances
  - Number of features
  - Statistical features (e.g., skewness, correlation, average, etc.)
  - Information theoretic features (e.g., the entropy of class labels)

- The selection of meta-features is highly dependent on the application.

Learning from Task Properties (2/3)

- Each prior task $\mathtt{t_j}$ is characterized by a meta-feature vector $\mathtt{m(t_j)}$.
- Information from a prior task $\mathtt{t_j}$ can be transferred to a new task $\mathtt{t_{new}}$ based on their similarity.
- The similarity between two tasks is the distance between the feature vectors.

Learning from Task Properties (3/3)

- Building meta-model.
- Building a meta-model $\mathtt{L}$ to learn the relationships between meta-features of prior tasks $\mathtt{t_j}$.
- For a new task $\mathtt{t_{new}}$, the meta-model $\mathtt{L}$ recommends the best configurations.

Learning from Prior Model Evaluation

Learning from Prior Model Evaluation (1/3)

- $\mathtt{t_j} \in \mathtt{T}$: $\mathtt{t_j}$ is a ML task and $\mathtt{T}$ is the set of all prior ML tasks.
- Θ: the configuration space (hyper-parameter setting, pipeline components, etc.).
- $\mathbf{P}$: the set of all prior evaluations $\mathtt{P_{i,j}}$ of configuration $\theta_{\mathtt{i}}$ on a prior task $\mathtt{t_j}$.
- Learn a meta-learner $\mathtt{L}$ that is trained on meta-data $\mathbf{P} \cup \mathbf{P}_{\mathtt{new}}$ to predict recommended configuration $\Theta_{\mathtt{new}}$ for a new task $\mathtt{t_{new}}$.
- Three different ways:
  1. Relative landmarks
  2. Surrogate models
  3. Warm-started multitask learning

Learning from Prior Model Evaluation (2/3)

- Relative landmarks measure the performance difference between two model configurations on the same task
- Two tasks $\mathtt{t_{new}}$ and $\mathtt{t_j}$ are considered similar, if their relative landmarks performance of the considered configurations are also similar.
- Once similar tasks have been identified, a meta-learner can be trained on the evaluations $\mathtt{P_{i,j}}$ and $\mathtt{P_{i,new}}$ to recommend new configurations for task $\mathtt{t_{new}}$.

Learning from Prior Model Evaluation (3/3)

- Surrogate models get trained on all prior evaluations **P** of all prior tasks $\mathtt{t_j}$.
- For a particular task $\mathtt{t_j}$, if the surrogate model can predict accurate configuration for a new task $\mathtt{t_{new}}$, then tasks $\mathtt{t_{new}}$ and $\mathtt{t_j}$ are considered similar.

Learning from Prior Models

Learning from Prior Models

- Using transfer learning that utilizes pretrained models on prior tasks $\mathtt{t_j}$ to be adapted on a new task $\mathtt{t_{new}}$, where tasks $\mathtt{t_j}$ and $\mathtt{t_{new}}$ are similar.
- E.g., NN architecture and parameters are trained on prior task $\mathtt{t_j}$ that can be used as an initialization for model adaptation on a new task $\mathtt{t_{new}}$.
- Then, the new model can be fine-tuned.
- Transfer learning usually works well when the new task to be learned is similar to the prior tasks.

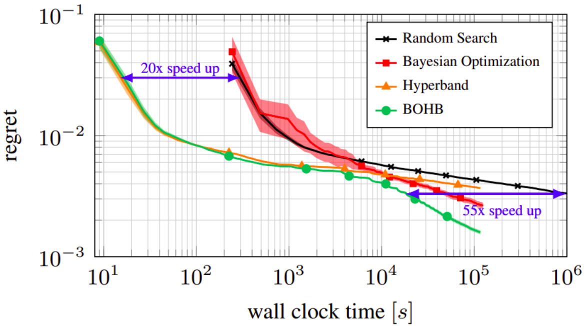
BOHB: Robust and Efficient Hyperparameter Optimization at Scale

BOHB: Bayesian Optimization and Hyperband

- Bayesian optimization (BO): for choosing the configuration to evaluate
- Hyperband (HB): for deciding how to allocate budgets

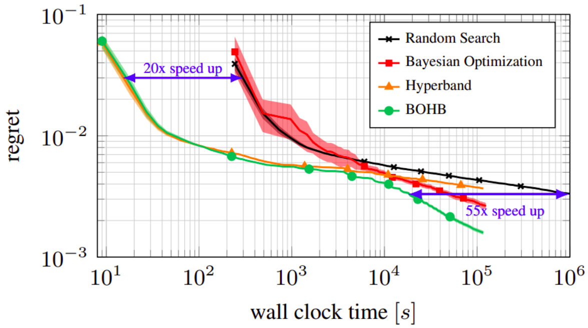
Bayesian Optimization vs. Random Search

- BO advantage: much improved final performance



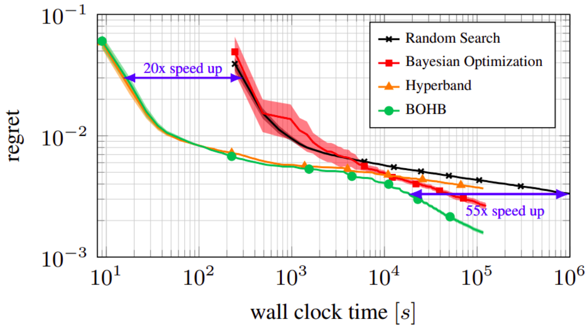
Hyperband vs. Random Search

- HB advantage: much improved anytime performance



Combining Bayesian Optimization and Hyperband

- Best of both worlds: strong anytime and final performance



HBOB Algorithm

- Relies on HB to determine how many configurations to evaluate with which budget.
- Replaces the random selection of configurations at the beginning of each HB iteration by a BO model-based search.
- Once the desired number of configurations for the iteration is reached, the SHA procedure is carried out using these configurations.

A System for Massively Parallel Hyperparameter Tuning

SHA

- SHA allocates a small budget to each configuration, evaluate all configurations and keep the top $\frac{1}{\rho}$.
- It then increases the budget per configuration by a factor of ρ.
- Repeats until the maximum per-configuration budget of $\mathtt{R}$ is reached.
- SHA requires the number of configurations, a min and max resource, a reduction factor, and a minimum early-stopping rate.

Asynchronous SHA (ASHA)

- ASHA is a technique to parallelize SHA, leveraging asynchrony to mitigate stragglers and maximize parallelism.
- ASHA promotes configurations to the next rung whenever possible, instead of waiting for a rung to complete before proceeding to the next rung.
- If no promotions are possible, ASHA simply adds a configuration to the base rung, so that more configurations can be promoted to the upper rungs.
- Given its asynchronous nature it does not require the user to pre-specify the number of configurations to evaluate, but it otherwise requires the same inputs as SHA.

DARTS: Differentiable Architecture Search

Differentiable ARchiTecture Search (DARTS)

- Instead of searching over a discrete set of candidate architectures, we relax the search space to be continuous.
- The architecture can be optimized with respect to its validation set performance by gradient descent.

Search Space

- It searches for a computation cell as the building block of the final architecture.
- A cell is a DAG consisting of an ordered sequence of N nodes.
- Each node $x^{(i)}$ is a latent representation (e.g. a feature map in CNNs).
- Each directed edge (i, j) is associated with some operation $o^{(i,j)}$ that transforms $x^{(i)}$.
- Each intermediate node is computed based on all of its predecessors: $x^{(j)} = \sum_{i<j} o^{(i,j)}(x^i)$

Continuous Relaxation and Optimization

- Let $\mathcal{O}$ be a set of candidate operations, where each operation represents some function $\mathtt{o}$ to be applied to $\mathtt{x}^{(\mathtt{i})}$.
- To make the search space continuous, it relaxes the categorical choice of a particular operation to a softmax over all possible operations:
$\overline{\mathtt{o}}^{(\mathtt{i},\mathtt{j})}(\mathtt{x}) = \sum_{\mathtt{o}\in\mathcal{O}} \frac{\mathtt{exp}(\alpha_{\mathtt{o}}^{(\mathtt{i},\mathtt{j})})}{\sum_{\mathtt{o}'\in\mathcal{O}} \mathtt{exp}(\alpha_{\mathtt{o}'}^{(\mathtt{i},\mathtt{j})})} \mathtt{o}(\mathtt{x})$
- The operation mixing weights for a pair of nodes $(\mathtt{i},\mathtt{j})$ are parameterized by a vector $\alpha^{(\mathtt{i},\mathtt{j})}$ of dimension $|\mathcal{O}|$.
- At the end of search, a discrete architecture can be obtained by replacing each mixed operation $\overline{\mathtt{o}}^{(\mathtt{i},\mathtt{j})}$ with the most likely operation, i.e., $\mathtt{o}^{(\mathtt{i},\mathtt{j})} = \arg\max_{\mathtt{o}\in\mathcal{O}} \alpha_{\mathtt{o}}^{(\mathtt{i},\mathtt{j})}$.

Summary

Summary

- Hyperparameter optimization
  - Black-box optimization
  - Multi-fidelity optimization
- Nural architecture search
  - Search space
  - Search strategy
  - Performance estimation
- Meta-learning
  - Learning from task properties
  - Learning from prior model evaluation
  - Learning from prior models

Reference

- Elshawi et al., Automated Machine Learning: State-of-The-Art and Open Challenges, 2019
- Falkner et al., BOHB: Robust and Efficient Hyperparameter Optimization at Scale, 2018
- Li et al., A System for Massively Parallel Hyperparameter Tuning, 2020
- Liu et al., DARTS: Differentable Architecture Search, 2019

Questions?