AutoML: Neural Architecture Search

Part 2: One-shot Neural Architecture Search

Bernd Bischl   Frank Hutter   Lars Kotthoff
Marius Lindauer   Joaquin Vanschoren

The One-Shot Model

One-shot models: convolutional neural fabrics [Saxena and Verbeek. 2017]

- A one-shot model is a big model that has all architectures in a search space as submodels
  - This allows weights sharing across architectures
  - One only needs to train the single one-shot model,
    and implicitly trains an exponential number of individual architectures

One-shot models: convolutional neural fabrics [Saxena and Verbeek. 2017]

- A one-shot model is a big model that has all architectures in a search space as submodels
  - This allows weights sharing across architectures
  - One only needs to train the single one-shot model,
    and implicitly trains an exponential number of individual architectures

- The first type of one-shot models: convolutional neural fabrics

One-shot models: convolutional neural fabrics [Saxena and Verbeek. 2017]

- A one-shot model is a big model that has all architectures in a search space as submodels
  - This allows weights sharing across architectures
  - One only needs to train the single one-shot model,
    and implicitly trains an exponential number of individual architectures
- The first type of one-shot models: convolutional neural fabrics

One-shot models: convolutional neural fabrics [Saxena and Verbeek. 2017]

- A one-shot model is a big model that has all architectures in a search space as submodels
  - This allows weights sharing across architectures
  - One only needs to train the single one-shot model,
    and implicitly trains an exponential number of individual architectures

- The first type of one-shot models: convolutional neural fabrics

One-shot models: convolutional neural fabrics [Saxena and Verbeek. 2017]

- A one-shot model is a big model that has all architectures in a search space as submodels
  - This allows weights sharing across architectures
  - One only needs to train the single one-shot model,
    and implicitly trains an exponential number of individual architectures

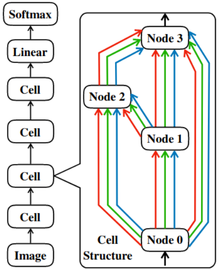
- The first type of one-shot models: convolutional neural fabrics

One-shot models: convolutional neural fabrics [Saxena and Verbeek. 2017]

- A one-shot model is a big model that has all architectures in a search space as submodels
  - This allows weights sharing across architectures
  - One only needs to train the single one-shot model, and implicitly trains an exponential number of individual architectures
- The first type of one-shot models: convolutional neural fabrics

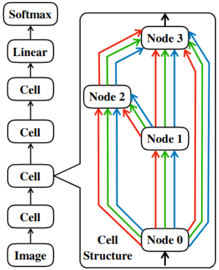
One-shot models for cell search spaces

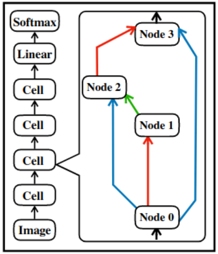
- Directed acyclic multigraph to capture all (exponentially many) cell architectures
  - The nodes represent tensors
  - The edges represent computations (e.g., 3x3 conv, 5x5 conv, max pool, ...)
  - The results of operations on multiple edges between two nodes are combined (addition/concatenation)

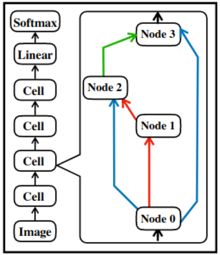


One-shot models for cell search spaces

- Directed acyclic multigraph to capture all (exponentially many) cell architectures
  - The nodes represent tensors
  - The edges represent computations (e.g., 3x3 conv, 5x5 conv, max pool, . . .)
  - The results of operations on multiple edges between two nodes are combined (addition/concatenation)
- Individual architectures are subgraphs of this multigraph
  - Weights for the operation on an edge are shared across all (exponentially many) architectures that have that edge

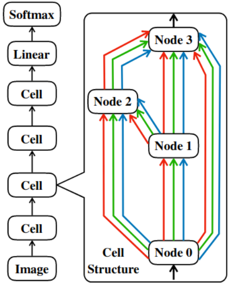






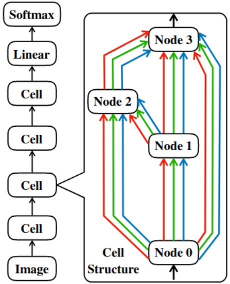
Training the one-shot model – standard SGD [Saxena and Verbeek. 2017]

- One-shot model is an acyclic graph; thus, backpropagation applies
  - Simplest method: standard training with SGD
  - This implicitly trains an exponential number of architectures



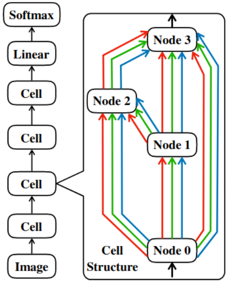
Training the one-shot model – standard SGD [Saxena and Verbeek. 2017]

- One-shot model is an acyclic graph; thus, backpropagation applies
  - Simplest method: standard training with SGD
  - This implicitly trains an exponential number of architectures
- Potential issue: co-adaptation of weights
  - Weights are implicitly optimized to work well on average across all architectures
  - They are not optimized specifically for the top-performing architecture



Training the one-shot model – DropPath [Bender et al. 2018]

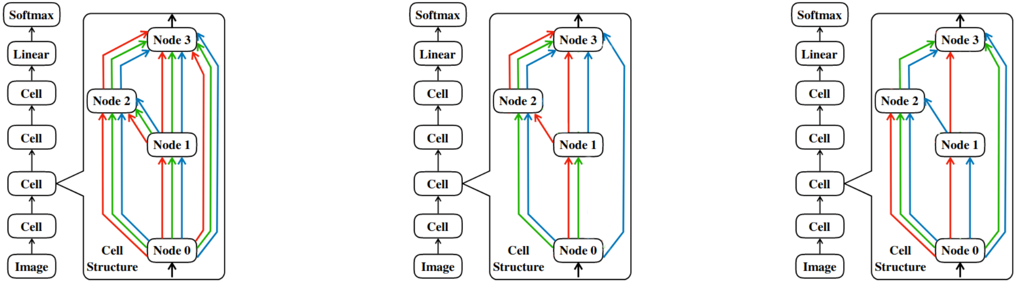
- To avoid coadaptation of weights, we can use DropPath, a technique analogous to Dropout [Srivastava et al., 2014]:



One-shot model

Training the one-shot model – DropPath [Bender et al. 2018]

- To avoid coadaptation of weights, we can use DropPath, a technique analogous to Dropout [Srivastava et al., 2014]:
  - At each mini-batch iteration:
    for each operation connecting 2 nodes, zero it out with probability p



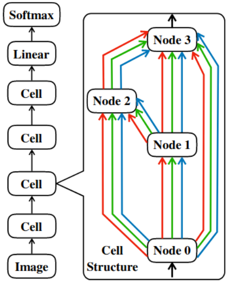
One-shot model

Architecture for batch 1

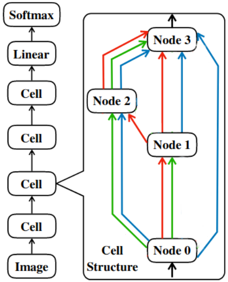
Architecture for batch 2

Training the one-shot model – DropPath [Bender et al. 2018]

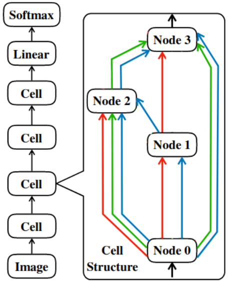
- To avoid coadaptation of weights, we can use DropPath, a technique analogous to Dropout [Srivastava et al., 2014]:
  - At each mini-batch iteration:
    for each operation connecting 2 nodes, zero it out with probability p
  - ScheduledDropPath: starts with $p = 0$ and increases p linearly to p_{max} at the end of training



One-shot model



Architecture for batch 1



Architecture for batch 2

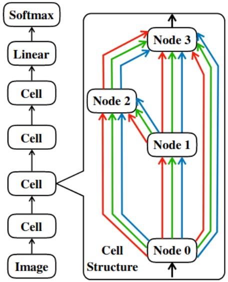
Training the one-shot model – Sampling

- At each mini-batch iteration during the training of the one-shot model sample a single architecture from the search space

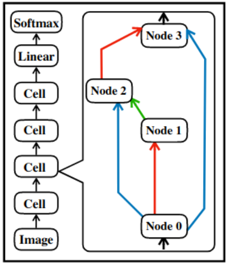
Training the one-shot model – Sampling

- At each mini-batch iteration during the training of the one-shot model sample a single architecture from the search space

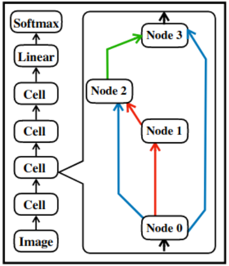
- Update the parameters of the one-shot model corresponding to only that architecture



One-shot model



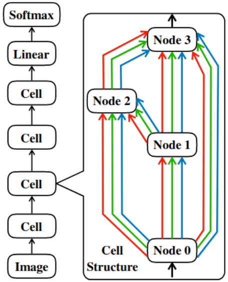
Architecture for batch 1



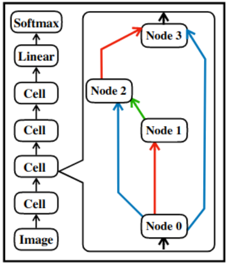
Architecture for batch 2

Training the one-shot model – Sampling

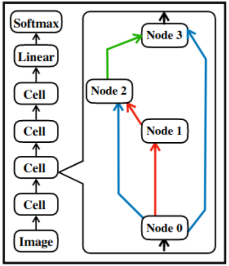
- At each mini-batch iteration during the training of the one-shot model sample a single architecture from the search space
  - **Random Search with Weight Sharing** [Li and Talwalkar. 2020] ⟶ sample from uniform distribution
  - **ENAS** [Pham et al. 2018] ⟶ sample from the learned policy of a RNN controller
- Update the parameters of the one-shot model corresponding to only that architecture



One-shot model



Architecture for batch 1



Architecture for batch 2

How to utilize the trained one-shot model?

- After training the one-shot model we have to select the best individual architecture from it
- There are multiple ways one can approach this. Some of these are:

How to utilize the trained one-shot model?

- After training the one-shot model we have to select the best individual architecture from it
- There are multiple ways one can approach this. Some of these are:
  1. Sample uniformly at random M architectures and rank them based on their validation error using the one-shot model parameters

How to utilize the trained one-shot model?

- After training the one-shot model we have to select the best individual architecture from it
- There are multiple ways one can approach this. Some of these are:
  1. Sample uniformly at random M architectures and rank them based on their validation error using the one-shot model parameters

  $1b$. (Optional) Select top K $(K < M)$ and retrain them from scratch for a couple of epochs

How to utilize the trained one-shot model?

- After training the one-shot model we have to select the best individual architecture from it
- There are multiple ways one can approach this. Some of these are:
  1. Sample uniformly at random M architectures and rank them based on their validation error using the one-shot model parameters

  $1b$. (Optional) Select top K $(K < M)$ and retrain them from scratch for a couple of epochs

  2. Return the top performing architecture to retrain from scratch for longer

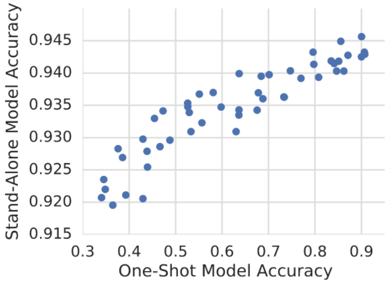
How to utilize the trained one-shot model?

- After training the one-shot model we have to select the best individual architecture from it
- There are multiple ways one can approach this. Some of these are:
  1. Sample uniformly at random M architectures and rank them based on their validation error using the one-shot model parameters

  $1b$. (Optional) Select top K ($K < M$) and retrain them from scratch for a couple of epochs

  2. Return the top performing architecture to retrain from scratch for longer

- **Pitfall:** the correlation between architectures evaluated with the one-shot weights and retrained from scratch (stand-alone models) should be high
- If not, **selecting the best architecture based on the one-shot weights** is sub-optimal.



From [Bender et al. 2018]

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  How are the weights shared in the one-shot model?
- Repetition:
  What is the difference between Random Search with Weight Sharing and ENAS?
- Discussion:
  What migth be some downsides of using the one-shot model for NAS?

AutoML: Neural Architecture Search (NAS)

Part 1: Search Spaces, Blackbox Methods, Speedup Techniques, and Best Practices

Bernd Bischl   Frank Hutter   Lars Kotthoff
Marius Lindauer   Joaquin Vanschoren

Outline

AutoML: Neural Architecture Search (NAS)

Overview

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

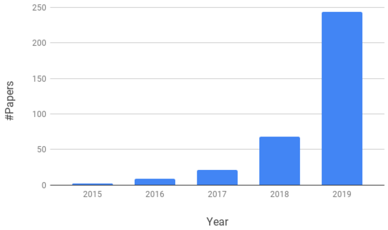
Neural Architecture Search (NAS)

- Goal: automatically find neural architectures with strong performance
  - Optionally, subject to a resource constraint

Neural Architecture Search (NAS)

- Goal: automatically find neural architectures with strong performance
  - Optionally, subject to a resource constraint

- A decade-old problem, but main stream since 2017 and now intensely researched
- One of the main problems AutoML is known for

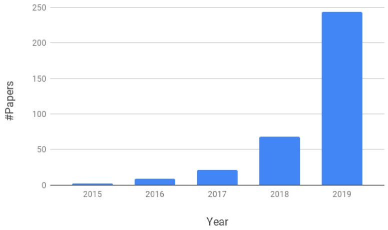


Neural Architecture Search (NAS)

- Goal: automatically find neural architectures with strong performance
  - Optionally, subject to a resource constraint

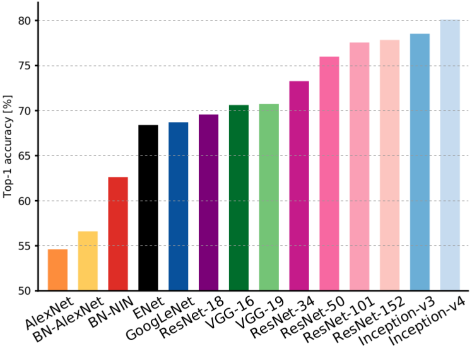
- A decade-old problem, but main stream since 2017 and now intensely researched
- One of the main problems AutoML is known for

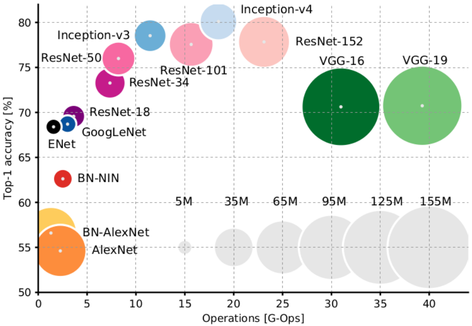
- Initially extremely expensive
- By now several methods promise low overhead over a single model training



Motivation for NAS

- Performance improvements on various tasks due to novel architectures
- Can we automate this design process, potentially discovering new components/topologies?

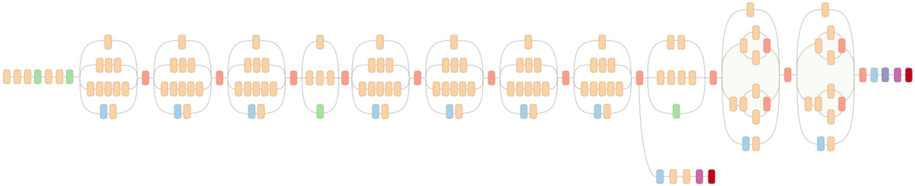




[Canziani et al. 2017]

Motivation for NAS

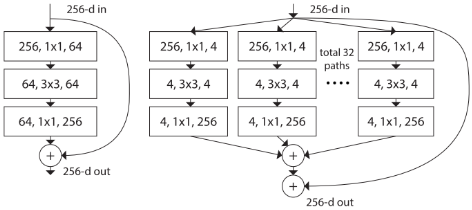
- Manual design of architectures is time consuming
- Complex state-of-the-art architectures are a result of years of trial and errors by experts



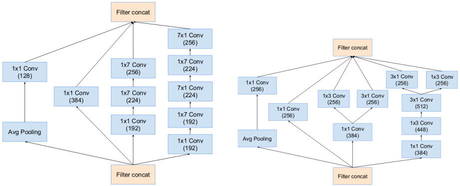
Inception-v3 [Szegedy et al. 2015]

Motivation for NAS

- Manual design of architectures is time consuming
- Complex state-of-the-art architectures are a result of years of trial and errors by experts
  - Main pattern: Repeated blocks with same structure (topology)

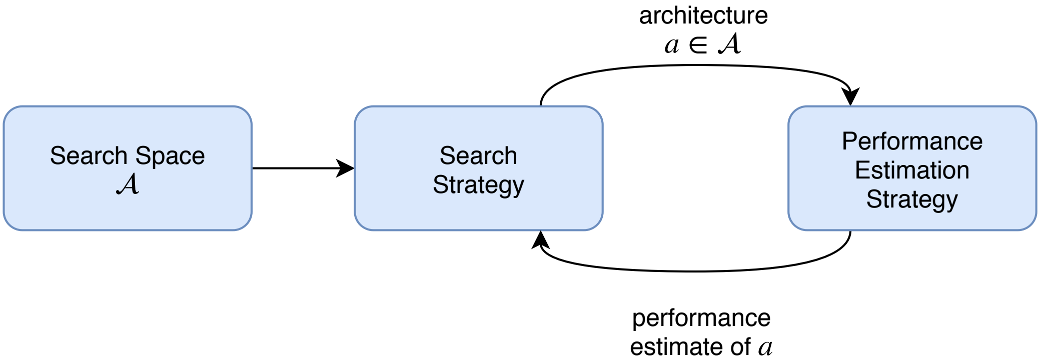


ResNet/ResNeXt blocks
[He et al. 2016; Xie et al. 2016]



Inception-v4 blocks [Szegedy et al. 2016]

NAS components [Elsken et al. 2019]



- Search Space: the types of architectures we consider; micro, macro, hierarchical, etc.
- Search Strategy: Reinforcement learning, evolutionary strategies, Bayesian optimization, gradient-based, etc.
- Performance Estimation Strategy: validation performance, lower fidelity estimates, one-shot model performance, etc.

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  List three major components of NAS methods.
- Discussion:
  Is there a problem for which you would like to apply NAS yourself?

Outline

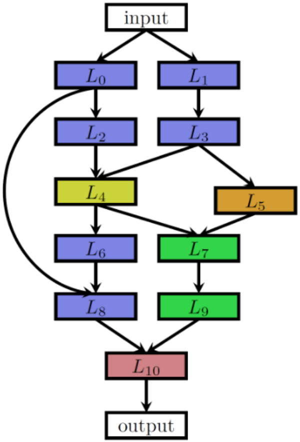
AutoML: Neural Architecture Search (NAS)

Search Spaces

Bernd Bischl   Frank Hutter   Lars Kotthoff
Marius Lindauer   Joaquin Vanschoren

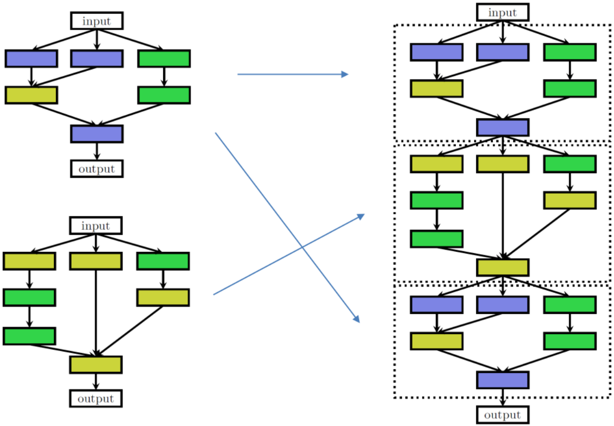
Basic Neural Architecture Search Spaces

Chain-structured space
(different colours:
different layer types)



More complex space
with multiple branches
and skip connections

Cell Search Spaces [Zoph et al. 2018]

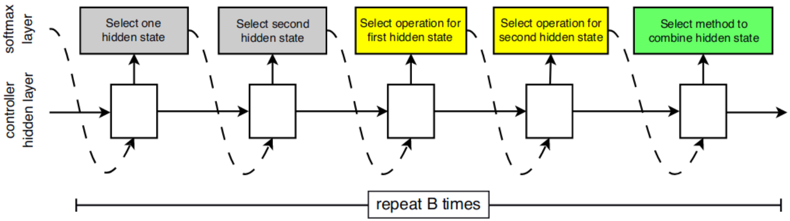


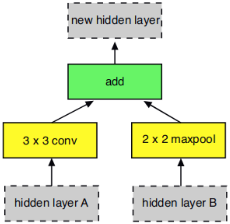
Two possible cells

Architecture composed of stacking together individual cells

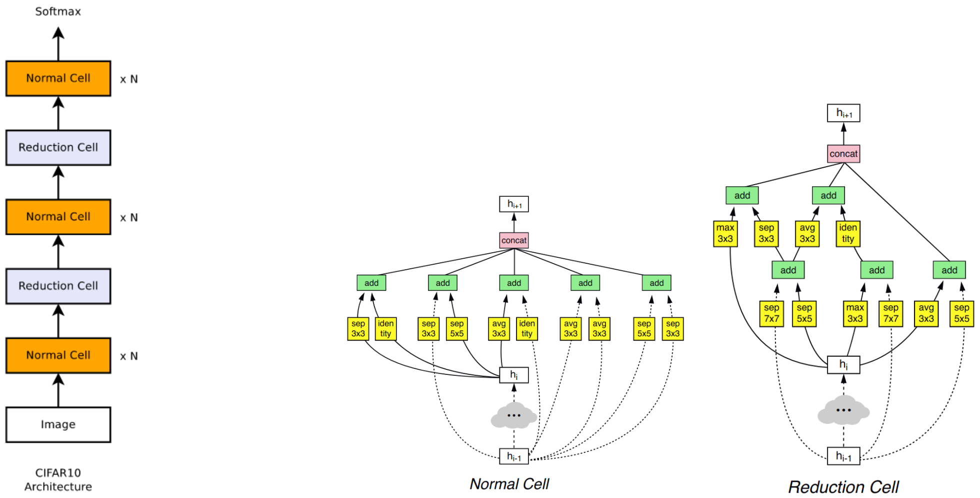
Details on Cell Search Spaces [Zoph et al. 2018]

- 2 types of cells: normal and reduction cells
- For each type of cell: B blocks, each with 5 choices
  - Choose two previous feature maps (from this cell)
  - For each of these, choose an operation (3×3 conv, max-pool, etc.)
  - Choose a merge operation to combine the two results (concat or add)





Example of an architecture sample with B=5



Source: [Zoph et al. 2018]

Pros and Cons of Cell Search Space

What are some pros and cons of the cell search space compared to the basic one?

Please think about this for a few minutes before continuing.

Pros and Cons of Cell Search Space

Pros:

- Reduced search space size; speed-ups in terms of search time.
- Transferability to other datasets (e.g., cells found on CIFAR-10 transfer to ImageNet)
- Stacking repeating patterns is proven to be a useful design principle (ResNet, Inception, etc.)

Pros and Cons of Cell Search Space

Pros:

- Reduced search space size; speed-ups in terms of search time.
- Transferability to other datasets (e.g., cells found on CIFAR-10 transfer to ImageNet)
- Stacking repeating patterns is proven to be a useful design principle (ResNet, Inception, etc.)

Cons:

- Still need to (manually) determine the *macro* architecture, i.e., the way in which cells are connected.
- Limiting if different cells work better in different parts of the network
  - E.g., different spatial resolutions may favour different convolutions

Hierarchical representation of search space [Liu et al. 2017]

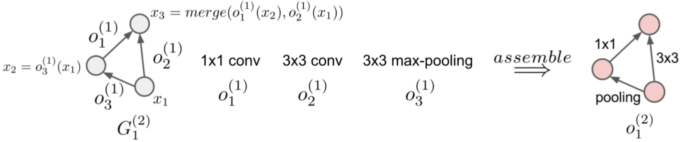
- Directed Acyclic Graph (DAG) representation of architectures
  - Each node is a latent representation; each edge is an operation/motif

Hierarchical representation of search space [Liu et al. 2017]

- Directed Acyclic Graph (DAG) representation of architectures
  - Each node is a latent representation; each edge is an operation/motif
- There are different levels of motivs
  - Level-1 primitives: standard operators; e.g., 3x3 conv, max pooling, . . .

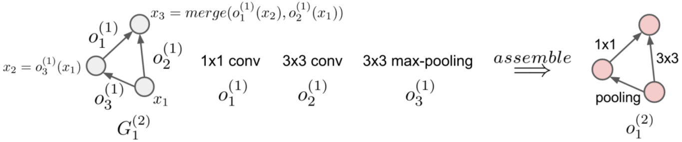
Hierarchical representation of search space [Liu et al. 2017]

- Directed Acyclic Graph (DAG) representation of architectures
  - Each node is a latent representation; each edge is an operation/motif
- There are different levels of motivs
  - Level-1 primitives: standard operators; e.g., 3x3 conv, max pooling, . . .
  - Level-2 motivs: combinations of level-1 primitives

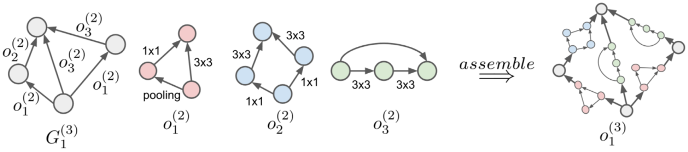


Hierarchical representation of search space [Liu et al. 2017]

- Directed Acyclic Graph (DAG) representation of architectures
  - Each node is a latent representation; each edge is an operation/motif
- There are different levels of motivs
  - Level-1 primitives: standard operators; e.g., 3x3 conv, max pooling, ...
  - Level-2 motivs: combinations of level-1 primitives



  - Level-3 motivs: combinations of level-2 motivs



Pros and Cons of Hierarchical Search Space

What are some pros and cons of a hierarchical search space compared to the cell search space?

Please think about this for a few minutes before continuing.

Pros and Cons of Hierarchical Search Space

Pros:

- Flexibility of constructing building blocks and reusing them many times
  - like a cell search space
- Flexibility of using different building blocks in different parts of the network
  - like a basic search space
- Ability to reuse building blocks at various levels of abstraction
  - again, this pattern has been used in manual design, e.g., in Inception nets

Pros and Cons of Hierarchical Search Space

Pros:

- Flexibility of constructing building blocks and reusing them many times
  - like a cell search space
- Flexibility of using different building blocks in different parts of the network
  - like a basic search space
- Ability to reuse building blocks at various levels of abstraction
  - again, this pattern has been used in manual design, e.g., in Inception nets

Cons:

- Larger than cell search space
- Vastly more expressive than cell search space $\rightarrow$ potentially much harder to search

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  What are some pros and cons of the cell search space compared to the basic one?
- Repetition:
  Explain the way in which level-3 motivs in the hierarchical search space use level-2 motivs.
- Repetition:
  What are some pros and cons of the hierarchical search space compared to the other ones?

Outline

AutoML: Neural Architecture Search (NAS)

Blackbox Optimization Methods

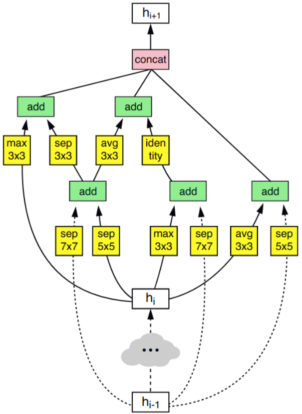
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

NAS as Hyperparameter Optimization

- NAS can be formulated as a HPO problem

NAS as Hyperparameter Optimization

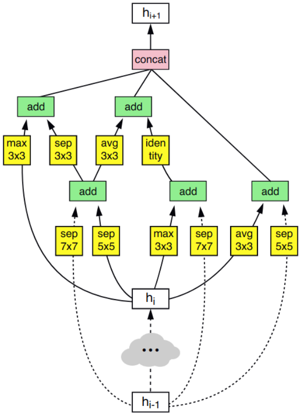
- NAS can be formulated as a HPO problem
- E.g., cell search space by [Zoph et al. 2018] has 5 categorical choices per block
  - 2 categorical choices of hidden states
    - ★ For block N, the domain of these categorical variables is $\{h_i, h_{i-1}, \text{output of block } 1, ..., \text{output of block } N-1\}$
  - 2 categorical variables choosing between operations
  - 1 categorical variable choosing the combination method
  - Total number of hyperparameters for the cell: 5B (with B=5 by default)



Reduction Cell

NAS as Hyperparameter Optimization

- NAS can be formulated as a HPO problem
- E.g., cell search space by [Zoph et al. 2018] has 5 categorical choices per block
  - 2 categorical choices of hidden states
    - ★ For block N, the domain of these categorical variables is $\{h_i, h_{i-1}, \text{output of block } 1, ..., \text{output of block } N-1\}$
  - 2 categorical variables choosing between operations
  - 1 categorical variable choosing the combination method
  - Total number of hyperparameters for the cell: 5B (with B=5 by default)
- In general: one may require conditional hyperparameters
  - E.g., chain-structured search space
    - ★ Top-level hyperparameter: number of layers L
    - ★ Hyperparameters of layer k conditional on L $\geq$ k



Reduction Cell

Early Work on Neuroevolution (already since the 1990s)

[Kitano. 1990; Angeline et al. 1994; Stanley and Miikkulainen. 2002; Bayer et al. 2009; Floreano et al. 2008]

- Evolves architectures & often also their weights

Early Work on Neuroevolution (already since the 1990s)

[Kitano. 1990; Angeline et al. 1994; Stanley and Miikkulainen. 2002; Bayer et al. 2009; Floreano et al. 2008]

- Evolves architectures & often also their weights
- Typical approach:
  - Initialize a population of N random architectures
  - Sample N individuals from that population (with replacement) according to their fitness
  - Apply mutations to those N individuals to produce the next generation's population
  - Optionally: elitism to keep best individuals in the population

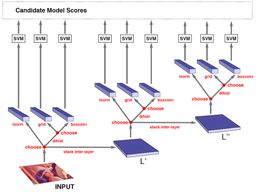
Early Work on Neuroevolution (already since the 1990s)

[Kitano. 1990; Angeline et al. 1994; Stanley and Miikkulainen. 2002; Bayer et al. 2009; Floreano et al. 2008]

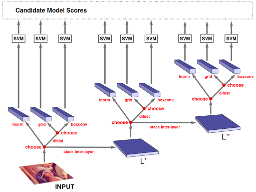
- Evolves architectures & often also their weights
- Typical approach:
  - Initialize a population of N random architectures
  - Sample N individuals from that population (with replacement) according to their fitness
  - Apply mutations to those N individuals to produce the next generation's population
  - Optionally: elitism to keep best individuals in the population
- Mutations include adding, changing or removing a layer

Early Work on Bayesian Optimization (since 2013)

- With TPE [Bergstra et al. 2011]:
  - Joint optimization of a vision architecture with 238 hyperparameters [Bergstra et al. 2013]
  - State-of-the-art performance on 3 disparate problems:
    - Face matching, face identification, and object recognition



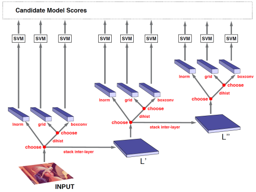
Early Work on Bayesian Optimization (since 2013)



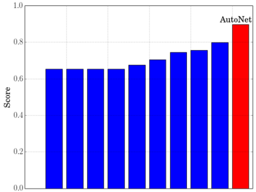
- With TPE [Bergstra et al. 2011]:
  - Joint optimization of a vision architecture with 238 hyperparameters [Bergstra et al. 2013]
  - State-of-the-art performance on 3 disparate problems:
    - ★ Face matching, face identification, and object recognition
- With SMAC [Hutter et al. 2011]:
  - New state-of-the-art performance on CIFAR-10 w/o data augmentation [Domhan et al. 2015]
  - Joint architecture and hyperparameter search, yielding Auto-Net [Mendoza et al. 2016]

Early Work on Bayesian Optimization (since 2013)

- With TPE [Bergstra et al. 2011]:
  - Joint optimization of a vision architecture with 238 hyperparameters [Bergstra et al. 2013]
  - State-of-the-art performance on 3 disparate problems:
    - Face matching, face identification, and object recognition

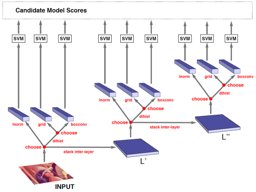


- With SMAC [Hutter et al. 2011]:
  - New state-of-the-art performance on CIFAR-10 w/o data augmentation [Domhan et al. 2015]
  - Joint architecture and hyperparameter search, yielding Auto-Net [Mendoza et al. 2016]
  - In 2015, Auto-Net already had several successes in ML competitions
    - E.g., human action recognition:
      54491 data points, 5000 features, 18 classes
    - First automated deep learning (Auto-DL) method to win a machine learning competition dataset against human experts

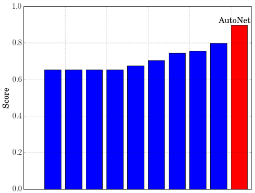


Early Work on Bayesian Optimization (since 2013)

- With TPE [Bergstra et al. 2011]:
  - Joint optimization of a vision architecture with 238 hyperparameters [Bergstra et al. 2013]
  - State-of-the-art performance on 3 disparate problems:
    - Face matching, face identification, and object recognition

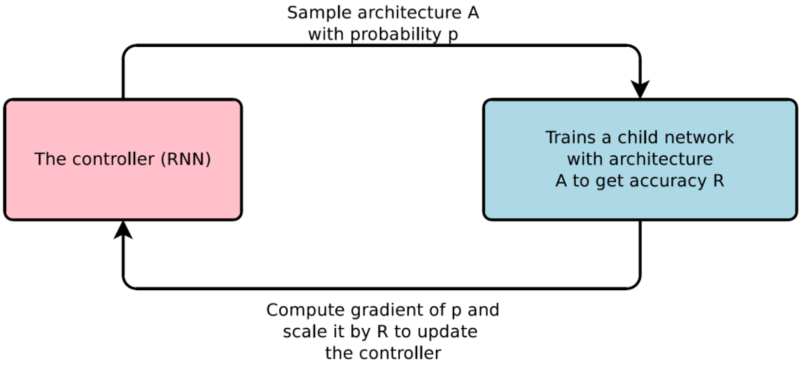


- With SMAC [Hutter et al. 2011]:
  - New state-of-the-art performance on CIFAR-10 w/o data augmentation [Domhan et al. 2015]
  - Joint architecture and hyperparameter search, yielding Auto-Net [Mendoza et al. 2016]
  - In 2015, Auto-Net already had several successes in ML competitions
    - E.g., human action recognition:
      54491 data points, 5000 features, 18 classes
    - First automated deep learning (Auto-DL) method to win a machine learning competition dataset against human experts



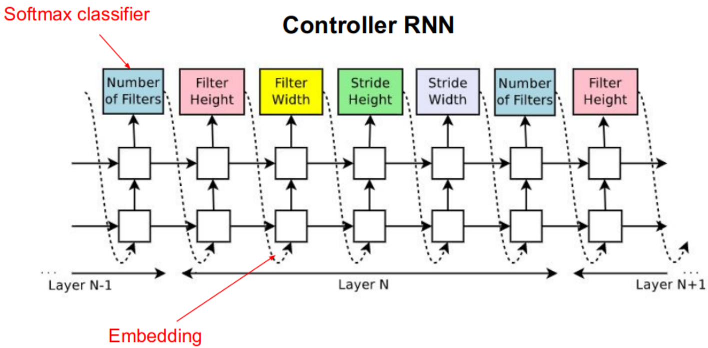
- With Gaussian processes:
  - Arc kernel [Swersky et al. 2013]

Reinforcement Learning [Zoph and Le. 2016]



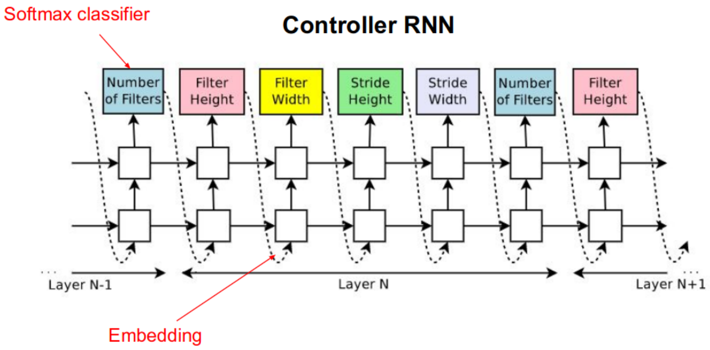
- Use RNN ("Controller") to generate a NN architecture piece-by-piece
- Train this NN ("Child Network") and evaluate it on a validation set
- Use Reinforcement Learning (RL) to update the parameters of the Controller RNN to optimize the performance of the child models

Learning CNNs with RL [Zoph and Le. 2016]



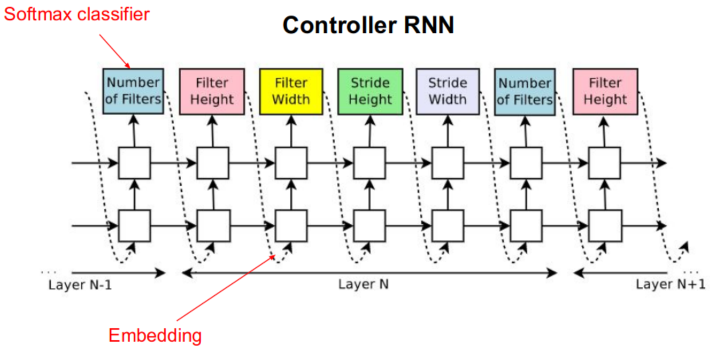
- For a fixed number of layers, select:
  - Filter width/height, stride width/height, number of filters

Learning CNNs with RL [Zoph and Le. 2016]



- For a fixed number of layers, select:
  - Filter width/height, stride width/height, number of filters
- Large computational demands (800 GPUs for 2 weeks, 12.800 architectures evaluated)

Learning CNNs with RL [Zoph and Le. 2016]

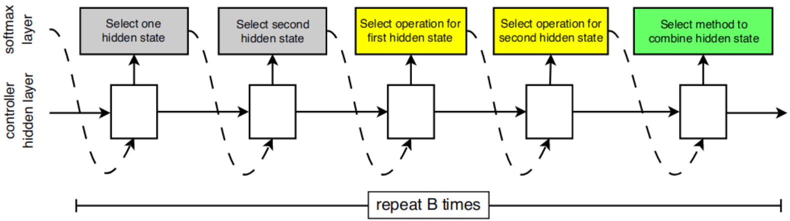


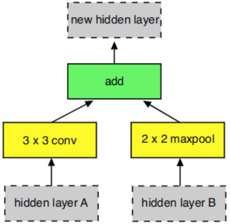
- For a fixed number of layers, select:
  - Filter width/height, stride width/height, number of filters
- Large computational demands (800 GPUs for 2 weeks, 12.800 architectures evaluated)
- State-of-the-art results for CIFAR-10 & Penn Treebank architecture

  → Brought NAS into the limelight

Learning CNN cells with RL [Zoph et al. 2018]

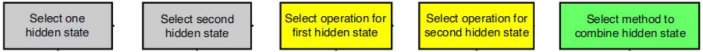
- 2 types of cells: normal and reduction cells
- For each type of cell: B blocks, each with 5 choices
  - Choose two previous feature maps (from this cell)
  - For each of these, choose an operation (3×3 conv, max-pool, etc.)
  - Choose a merge operation to combine the two results (concat or add)





Learning CNN cells with evolution [Real et al. 2018]

- 2 types of cells: normal and reduction cells
- For each type of cell: B blocks, each with 5 choices
  - Choose two previous feature maps (from this cell)
  - For each of these, choose an operation (3$\times$3 conv, max-pool, etc.)
  - Choose a merge operation to combine the two results (concat or add)
- Evolution simply tackles this as a HPO problem with 2$\times$5$\times$B variables:

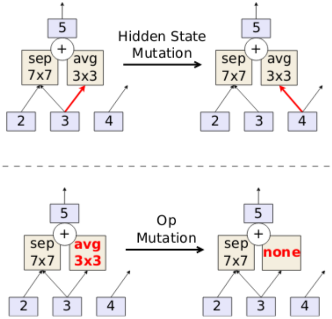


Regularized/Aging Evolution [Real et al. 2018]

- Quite standard evolutionary algorithm
  - But oldest solutions are dropped from population, instead of the worst

Regularized/Aging Evolution [Real et al. 2018]

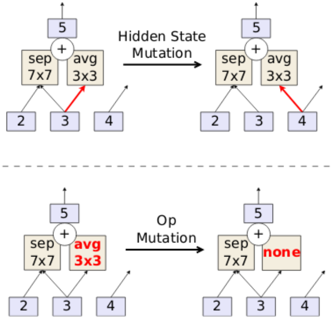
- Quite standard evolutionary algorithm
  - But oldest solutions are dropped from population, instead of the worst
- Standard SGD for training weights (optimizing the same blackbox as RL)
- Same fixed-length (HPO) search space as used for RL [Zoph et al. 2018]



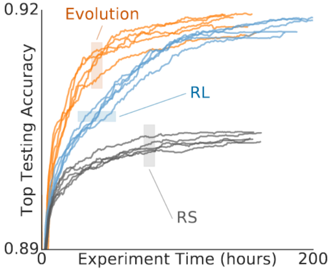
Different types of mutations in cell search space

Regularized/Aging Evolution [Real et al. 2018]

- Quite standard evolutionary algorithm
  - But oldest solutions are dropped from population, instead of the worst
- Standard SGD for training weights (optimizing the same blackbox as RL)
- Same fixed-length (HPO) search space as used for RL [Zoph et al. 2018]



Different types of mutations in cell search space



State-of-the-art performance in apples-to-apples comparison $\rightarrow$ AmoebaNet

Bayesian Optimization (BO)

- Encode the architecture space by categorical hyperparameters (like regularized evolution)
- Strong performance with tree-based models
  - TPE [Bergstra et al. 2013]
  - SMAC [Domhan et al. 2015; Mendoza et al. 2016; Zela et al. 2018]

Bayesian Optimization (BO)

- Encode the architecture space by categorical hyperparameters (like regularized evolution)
- Strong performance with tree-based models
  - TPE [Bergstra et al. 2013]
  - SMAC [Domhan et al. 2015; Mendoza et al. 2016; Zela et al. 2018]
- Kernels for GP-based NAS
  - Arc kernel [Swersky et al. 2013]
  - NASBOT [Kandasamy et al. 2018]

Bayesian Optimization (BO)

- Encode the architecture space by categorical hyperparameters (like regularized evolution)
- Strong performance with tree-based models
  - TPE [Bergstra et al. 2013]
  - SMAC [Domhan et al. 2015; Mendoza et al. 2016; Zela et al. 2018]
- Kernels for GP-based NAS
  - Arc kernel [Swersky et al. 2013]
  - NASBOT [Kandasamy et al. 2018]
- There are also several recent promising BO approaches based on neural networks
  - BANANAS [White et al. 2019]

Bayesian Optimization (BO)

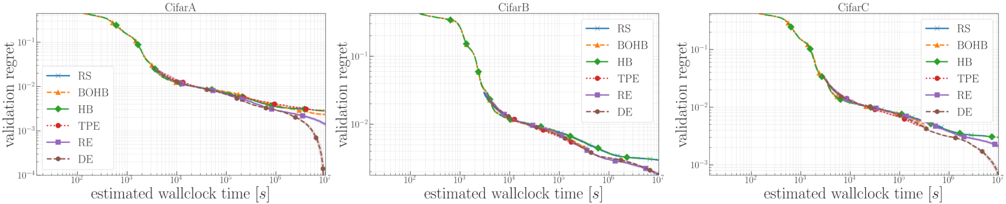
- Encode the architecture space by categorical hyperparameters (like regularized evolution)
- Strong performance with tree-based models
  - TPE [Bergstra et al. 2013]
  - SMAC [Domhan et al. 2015; Mendoza et al. 2016; Zela et al. 2018]
- Kernels for GP-based NAS
  - Arc kernel [Swersky et al. 2013]
  - NASBOT [Kandasamy et al. 2018]
- There are also several recent promising BO approaches based on neural networks
  - BANANAS [White et al. 2019]
- BO is very competitive, has been shown to outperform RL [Ying et al. 2019]

Current State of the Art: Differential Evolution

- Comprehensive experiments on a wide range of 12 different NAS benchmarks
  [Awad et al. 2020]

Current State of the Art: Differential Evolution

- Comprehensive experiments on a wide range of 12 different NAS benchmarks [Awad et al. 2020]
- Results:
  - Regularized evolution is very robust, typically amongst best of the methods discussed so far
  - Evolution variant of differential evolution is yet better; most efficient and robust method



Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  What are some pros and cons of using black-box optimizers for NAS?
- Repetition:
  How can NAS be modelled as a HPO problem?
- Discussion:
  Given enough resources, will blackbox NAS approaches always improve performance?
- Discussion:
  Why does discarding the oldest individual (rather than the worst) help in regularized/aging evolution?
- Transfer:
  How would you write NAS with the hierarchical search space as a HPO problem?

Outline

AutoML: Neural Architecture Search (NAS)

Speedup Techniques

Bernd Bischl   Frank Hutter   Lars Kotthoff
Marius Lindauer   Joaquin Vanschoren

Overview of NAS Speedup Methods

- Multi-fidelity optimization
- Learning curve prediction
- Meta-learning across datasets
- Network morphisms & weight inheritance
- Weight sharing & the one-shot model

NAS Speedup Technique 1: Multi-fidelity optimization

- Analogous to multi-fidelity optimization in HPO
  - Many evaluations for cheaper fidelities (less epochs, smaller datasets, down-sampled images, shallower networks, etc)
  - Fewer evaluations necessary for more expensive fidelities

NAS Speedup Technique 1: Multi-fidelity optimization

- Analogous to multi-fidelity optimization in HPO
  - Many evaluations for cheaper fidelities (less epochs, smaller datasets, down-sampled images, shallower networks, etc)
  - Fewer evaluations necessary for more expensive fidelities
- Compatible with any blackbox optimization method
  - Using random search: ASHA [Li and Talwalkar. 2019]
  - Using Bayesian optimization: BOHB [Zela et al. 2018]
  - Using differential evolution: DEHB [Awad et al. under review]
  - Using regularized evolution: progressive dynamic hurdles [So et al. 2019]

NAS Speedup Technique 1: Multi-fidelity optimization

- Analogous to multi-fidelity optimization in HPO
  - Many evaluations for cheaper fidelities (less epochs, smaller datasets, down-sampled images, shallower networks, etc)
  - Fewer evaluations necessary for more expensive fidelities
- Compatible with any blackbox optimization method
  - Using random search: ASHA [Li and Talwalkar. 2019]
  - Using Bayesian optimization: BOHB [Zela et al. 2018]
  - Using differential evolution: DEHB [Awad et al. under review]
  - Using regularized evolution: progressive dynamic hurdles [So et al. 2019]
- Often used for joint optimization of architecture & hyperparameters
  - Auto-Pytorch [Mendoza et al. 2019; Zimmer et al. 2020]
  - "Auto-RL" [Runge et al. 2019]

NAS Speedup Technique 2: Learning Curve Prediction

- Analogous to learning curve prediction in HPO
  - Observe initial learning curve and predict performance at the end
  - Can use features of the architecture as input (just like hyperparameters as inputs)

NAS Speedup Technique 2: Learning Curve Prediction

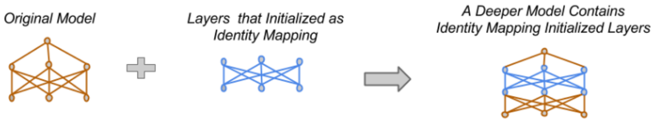
- Analogous to learning curve prediction in HPO
  - Observe initial learning curve and predict performance at the end
  - Can use features of the architecture as input (just like hyperparameters as inputs)
- Often used for joint optimization of architecture & hyperparameters
- Compatible with any blackbox optimization method
  - Using random search and Bayesian optimization: [Domhan et al. 2015]
  - Using reinforcement learning: [Baker et al. 2018]

NAS Speedup Technique 3: Meta-Learning

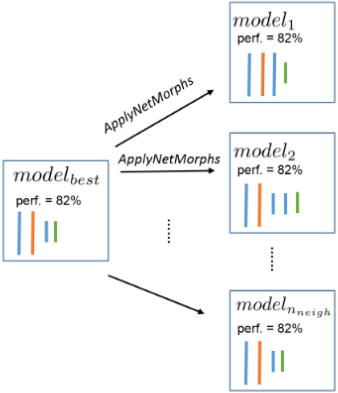
- Lots of work on meta-learning for HPO
- Only little work on meta-learning for NAS
  - Find a set of good architectures to initialize BOHB in Auto-Pytorch [Zimmer et al. 2020]
  - Learn RL agent's policy network on previous datasets [Wong et al. 2018]
  - Learn neural architecture that can be quickly adapted [Lian et al. 2019; Elsken et al. 2019]

NAS Speedup Technique 4: Network Morphisms

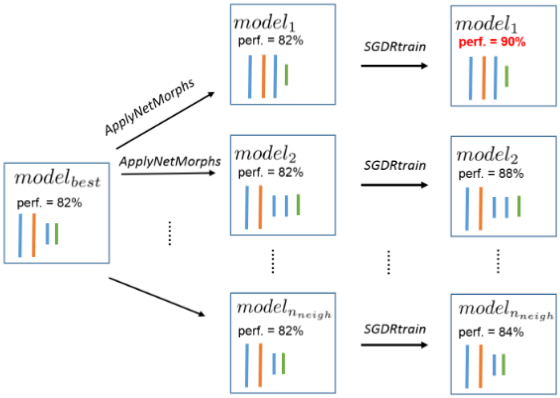
- Network Morphisms [Chen et al. 2016; Wei et al. 2016; Cai et al. 2017]
  - Change the network structure, but not the modelled function
  - I.e., for every input the network yields the same output as before applying the network morphisms operations
  - Examples: “Net2DeeperNet”, “Net2WiderNet”, etc.



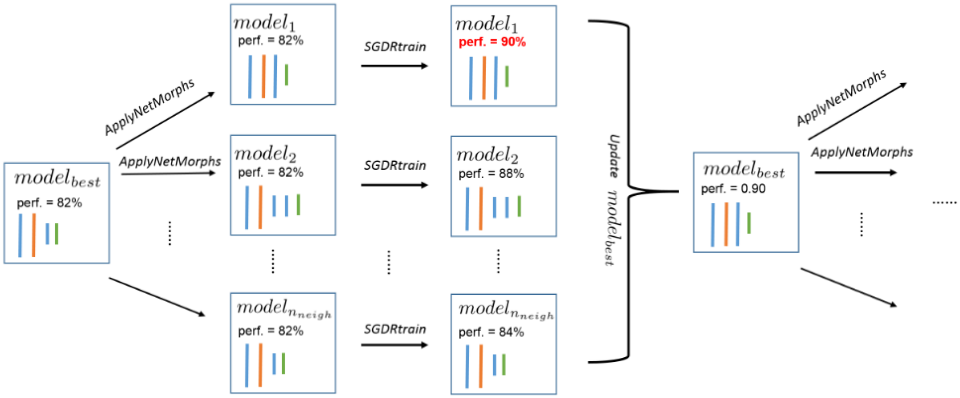
Network Morphisms Allow Efficient Moves in Architecture Space



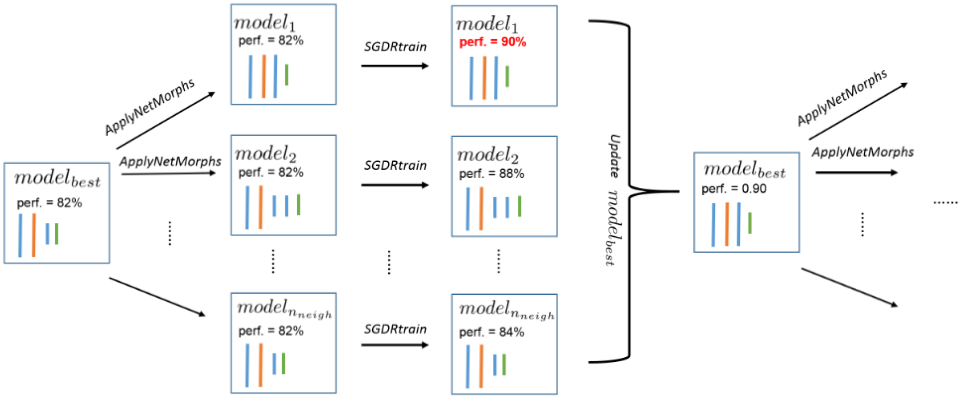
Network Morphisms Allow Efficient Moves in Architecture Space



Network Morphisms Allow Efficient Moves in Architecture Space



Network Morphisms Allow Efficient Moves in Architecture Space

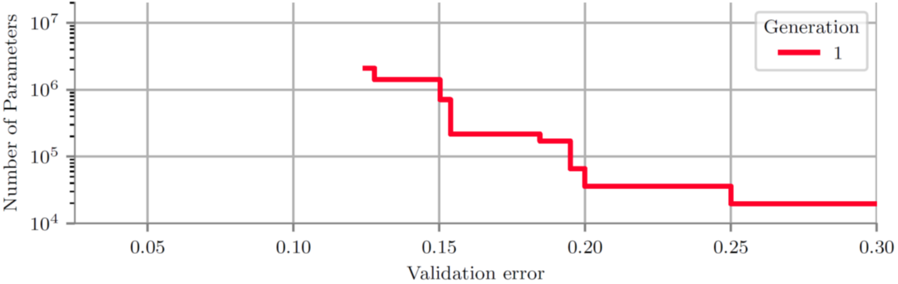


Weight inheritance avoids expensive retraining from scratch

[Real et al. 2017; Cai et al. 2018; Elsken et al. 2017; Cortes et al. 2017; Cai et al. 2018; Elsken et al. 2019]

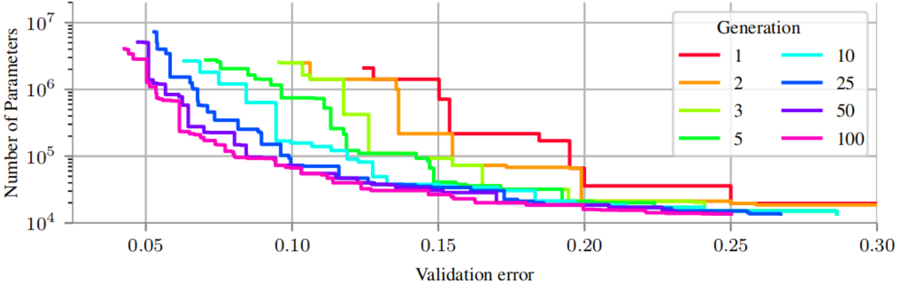
Network Morphisms for Multi-objective NAS [Elsken et al. 2019]

- To trade off error vs. resource consumption (e.g, #parameters):
  - Maintain a Pareto front of the two objectives
  - Evolve a population of Pareto-optimal architectures over time



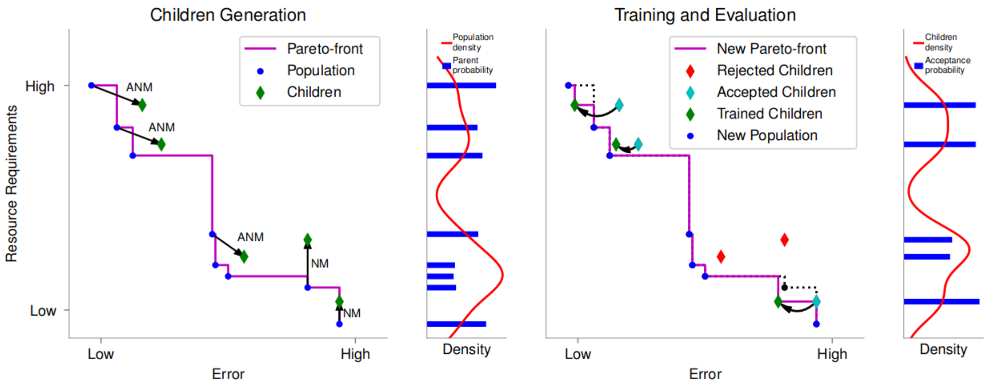
Network Morphisms for Multi-objective NAS [Elsken et al. 2019]

- To trade off error vs. resource consumption (e.g, #parameters):
  - Maintain a Pareto front of the two objectives
  - Evolve a population of Pareto-optimal architectures over time



Network Morphisms for Multi-objective NAS [Elsken et al. 2019]

- LEMONADE: Lamarckian Evolution for Multi-Objective Neural Architecture Design
- Weight inheritance through approximate morphisms (ANMs)
  - Dropping layers, dropping units within a layer, etc (function not preserved perfectly)



NAS Speedup Technique 5: Weight Sharing and One-shot Models

[Pham et al. 2018; Bender et al. 2018]

- All possible architectures are subgraphs of a large supergraph: the one-shot model

NAS Speedup Technique 5: Weight Sharing and One-shot Models

[Pham et al. 2018; Bender et al. 2018]

- All possible architectures are subgraphs of a large supergraph: the one-shot model
- Weights are shared between different architectures with common edges in the supergraph

NAS Speedup Technique 5: Weight Sharing and One-shot Models

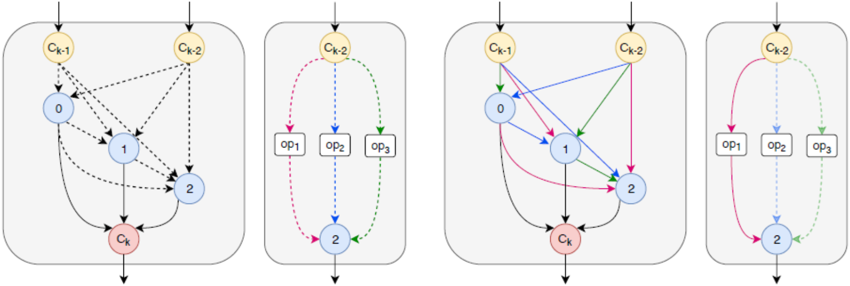
[Pham et al. 2018; Bender et al. 2018]

- All possible architectures are subgraphs of a large supergraph: the one-shot model
- Weights are shared between different architectures with common edges in the supergraph
- Search costs are reduced drastically since one only has to train a single model (the one-shot model).

NAS Speedup Technique 5: Weight Sharing and One-shot Models

[Pham et al. 2018; Bender et al. 2018]

- The one-shot model can be seen as a directed acyclic multigraph
  - ⇒ Nodes - latent representations.
  - ⇒ Edges (dashed) - operations.



(a) One-shot search **(b)** Final evaluation

- Architecture optimization problem: Find optimal path from the input to the output

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  List five methods to speed up NAS over blackbox approaches
- Repetition:
  Which speedup techniques directly carry over from HPO to NAS?
- Discussion:
  Why do network morphisms and the one-shot model only apply to NAS, and not to HPO?

Outline

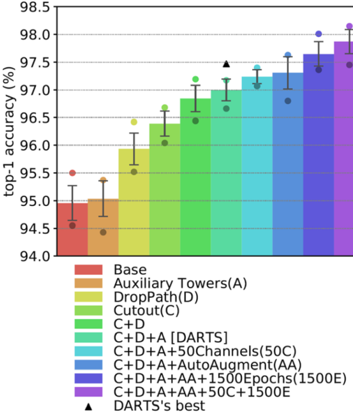
AutoML: Neural Architecture Search (NAS)

Issues and Best Practices in NAS Research

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Issues in NAS Research & Evaluations

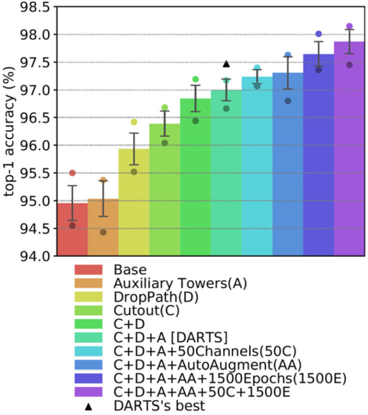
- Most NAS methods are extremely difficult to reproduce and compare [Li and Talwalkar. 2019]
- For benchmarks used in almost all NAS papers:
  - Training pipeline matters much more than neural architecture



[Yang et al. 2020]

Issues in NAS Research & Evaluations

- Most NAS methods are extremely difficult to reproduce and compare [Li and Talwalkar. 2019]
- For benchmarks used in almost all NAS papers:
  - Training pipeline matters much more than neural architecture
- The final benchmark results reported in different papers are typically incomparable
  - Different training code (often unavailable)
  - Different search spaces
  - Different evaluation schemes

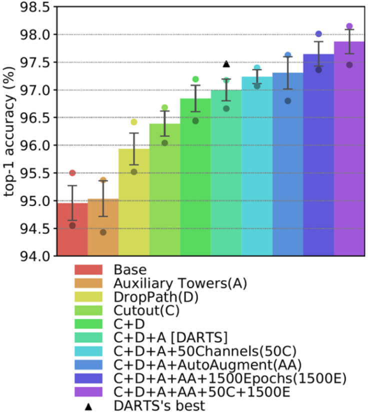


[Yang et al. 2020]

Issues in NAS Research & Evaluations

- Most NAS methods are extremely difficult to reproduce and compare [Li and Talwalkar. 2019]
- For benchmarks used in almost all NAS papers:
  - Training pipeline matters much more than neural architecture
- The final benchmark results reported in different papers are typically incomparable
  - Different training code (often unavailable)
  - Different search spaces
  - Different evaluation schemes

→ We emphasize concepts, not published performance numbers



[Yang et al. 2020]

Building a Scientific Community for NAS

- Benchmarks
  - NAS-Bench-101 [Ying et al. 2019]
  - NAS-Bench-201 [Dong and Yang. 2020]
  - NAS-Bench-1Shot1 [Zela et al. 2020]

Building a Scientific Community for NAS

- Benchmarks
  - NAS-Bench-101 [Ying et al. 2019]
  - NAS-Bench-201 [Dong and Yang. 2020]
  - NAS-Bench-1Shot1 [Zela et al. 2020]
- Best Practice Checklist for Scientific Research in NAS
  [Lindauer and Hutter. 2020]

Building a Scientific Community for NAS

- Benchmarks
  - NAS-Bench-101 [Ying et al. 2019]
  - NAS-Bench-201 [Dong and Yang. 2020]
  - NAS-Bench-1Shot1 [Zela et al. 2020]
- Best Practice Checklist for Scientific Research in NAS [Lindauer and Hutter. 2020]
- Unifying open-source implementation of modern NAS algorithms [Zela et al. 2020]
  - Finally enables empirical comparisons without confounding factors

Building a Scientific Community for NAS

- Benchmarks
  - NAS-Bench-101 [Ying et al. 2019]
  - NAS-Bench-201 [Dong and Yang. 2020]
  - NAS-Bench-1Shot1 [Zela et al. 2020]
- Best Practice Checklist for Scientific Research in NAS [Lindauer and Hutter. 2020]
- Unifying open-source implementation of modern NAS algorithms [Zela et al. 2020]
  - Finally enables empirical comparisons without confounding factors
- First NAS workshop at ICLR 2020

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for releasing code
  - Code for the training pipeline used to evaluate the final architectures
  - Hyperparameters used for the final evaluation pipeline, as well as random seeds
  - Code for the search space

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for releasing code
  - Code for the training pipeline used to evaluate the final architectures
  - Hyperparameters used for the final evaluation pipeline, as well as random seeds
  - Code for the search space
  - Code for your NAS method
  - Hyperparameters for your NAS method, as well as random seeds

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for releasing code
  - Code for the training pipeline used to evaluate the final architectures
  - Hyperparameters used for the final evaluation pipeline, as well as random seeds
  - Code for the search space
  - Code for your NAS method
  - Hyperparameters for your NAS method, as well as random seeds
- Note that the easiest way to satisfy the first three is to use existing NAS benchmarks

Definition: NAS Benchmark [Lindauer and Hutter. 2020]

A NAS benchmark consists of a dataset (with a predifiend training-test split), a search space, and available runnable code with pre-defined hyperparameters for training the architectures.

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for comparing NAS methods
  - For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for comparing NAS methods
  - For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
  - Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for comparing NAS methods
  - For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
  - Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
  - Did you run ablation studies?

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for comparing NAS methods
  - For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
  - Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
  - Did you run ablation studies?
  - Did you use the same evaluation protocol for the methods being compared?

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for comparing NAS methods
  - For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
  - Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
  - Did you run ablation studies?
  - Did you use the same evaluation protocol for the methods being compared?
  - Did you compare performance over time?

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for comparing NAS methods
  - For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
  - Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
  - Did you run ablation studies?
  - Did you use the same evaluation protocol for the methods being compared?
  - Did you compare performance over time?
  - Did you compare to random search?

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for comparing NAS methods
  - For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
  - Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
  - Did you run ablation studies?
  - Did you use the same evaluation protocol for the methods being compared?
  - Did you compare performance over time?
  - Did you compare to random search?
  - Did you perform multiple runs of your experiments and report seeds?

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for comparing NAS methods
  - For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
  - Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
  - Did you run ablation studies?
  - Did you use the same evaluation protocol for the methods being compared?
  - Did you compare performance over time?
  - Did you compare to random search?
  - Did you perform multiple runs of your experiments and report seeds?
  - Did you use tabular or surrogate benchmarks for in-depth evaluations?

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for reporting important details
  - Did you report how you tuned hyperparameters, and what time and resources this required?

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for reporting important details
  - Did you report how you tuned hyperparameters, and what time and resources this required?
  - Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)?

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for reporting important details
  - Did you report how you tuned hyperparameters, and what time and resources this required?
  - Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)?
  - Did you report all the details of your experimental setup?

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for reporting important details
  - Did you report how you tuned hyperparameters, and what time and resources this required?
  - Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)?
  - Did you report all the details of your experimental setup?

- It might not always be possible to satisfy all these best practices, but being aware of them is the first step . . .

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for reporting important details
  - Did you report how you tuned hyperparameters, and what time and resources this required?
  - Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)?
  - Did you report all the details of your experimental setup?

- It might not always be possible to satisfy all these best practices, but being aware of them is the first step . . .

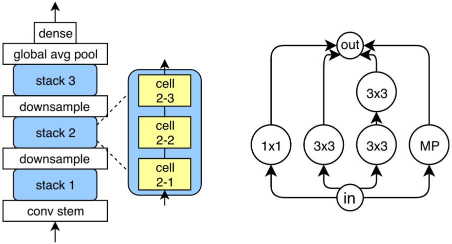
- We believe the community would benefit a lot from:
  - Clean NAS benchmarks for new applications
    - Including all details for the application. No need to also develop a new method.
  - Open-source library of NAS methods to compare methods without confounding factors
    - First version already developed: NASlib [Zela et al, under review]

NAS-Bench-101: The First NAS Benchmark [Ying et al. 2019]

- Dataset: CIFAR-10, with the standard training/test split
- Runnable open-source code provided in Tensorflow
- Cell-structured search space consisting of all directed acyclic graphs (DAGs) on V nodes, where each possible node has L operation choices.

NAS-Bench-101: The First NAS Benchmark [Ying et al. 2019]

- Dataset: CIFAR-10, with the standard training/test split
- Runnable open-source code provided in Tensorflow
- Cell-structured search space consisting of all directed acyclic graphs (DAGs) on V nodes, where each possible node has L operation choices.
- To limit the number of architectures, NAS-Bench-101 has the following constraints:
  - $L = 3$ operators:
    - 3×3 convolution - 1×1 convolution - 3×3 max-pooling
  - $V \leq 7$ nodes
  - A maximum of 9 edges

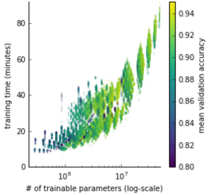


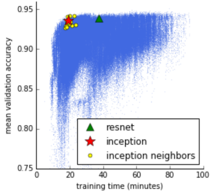
NAS-Bench-101: The First Tabular NAS Benchmark [Ying et al. 2019]

- Tabular benchmark: we exhaustively trained and evaluated all possible models on CIFAR-10 to create a tabular (look-up table) benchmark
- Based on this table, anyone can now run NAS experiments in seconds without a GPU.

NAS-Bench-101: The First Tabular NAS Benchmark [Ying et al. 2019]

- Tabular benchmark: we exhaustively trained and evaluated all possible models on CIFAR-10 to create a tabular (look-up table) benchmark
- Based on this table, anyone can now run NAS experiments in seconds without a GPU.
- Around 423k unique cells
  - 4 epoch budgets: 4, 12, 36, 108
  - 3 repeats
  - around 5M trained and evaluated models
  - 120 TPU years of computation
  - the best architecture mean test accuracy: 94.32%

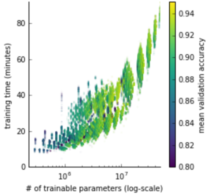




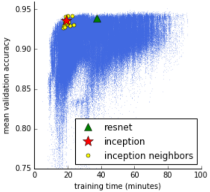
NAS-Bench-101: The First Tabular NAS Benchmark [Ying et al. 2019]

- Tabular benchmark: we exhaustively trained and evaluated all possible models on CIFAR-10 to create a tabular (look-up table) benchmark
- Based on this table, anyone can now run NAS experiments in seconds without a GPU.

- Around 423k unique cells
  - 4 epoch budgets: 4, 12, 36, 108
  - 3 repeats
  - around 5M trained and evaluated models
  - 120 TPU years of computation
  - the best architecture mean test accuracy: 94.32%

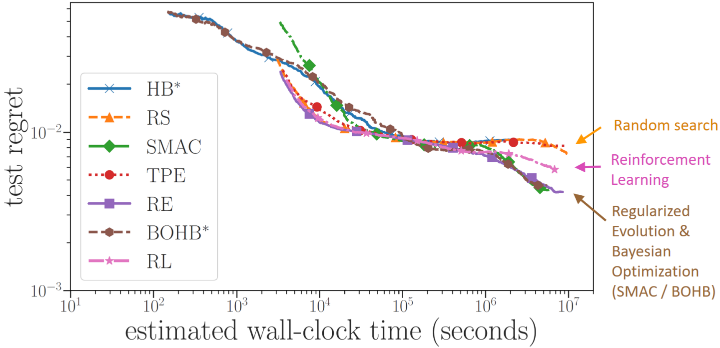


- Given an architecture encoding A, budget E_{stop} and trial number, one can query from NAS-Bench-101 the following quantities:
  - training/validation/test accuracy
  - training time in seconds
  - number of trainable model parameters



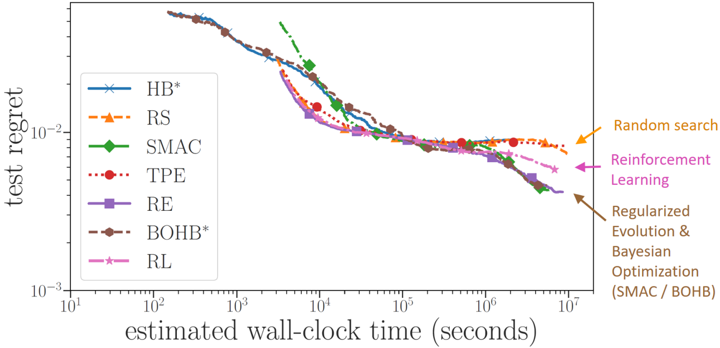
Evaluation of Blackbox NAS Methods on NAS-Bench-101 [Ying et al. 2019]

- RL outperforms random search
- BO and regularized evolution perform best, better than RL



Evaluation of Blackbox NAS Methods on NAS-Bench-101 [Ying et al. 2019]

- RL outperforms random search
- BO and regularized evolution perform best, better than RL



- Note that the BO method SMAC [Hutter et al. 2011] predated RL for NAS [Zoph and Le. 2017] by 6 years
  - Only now, benchmarks like NAS-Bench-101 allow for efficient comparisons

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  For the most common NAS search space, how important is the NAS component compared to the importance of the training pipeline used?
- Repetition:
  Why do we need proper benchmarking of NAS algorithms?
- Repetition:
  What does a NAS benchmark consist of?
- Repetition:
  List all best practices for NAS you remember.

Further Reading

Survey on NAS: [Elsken et al. 2019]