Machine Learning Pipeline

- Data/Feature Preprocessing
- Model Selection
- Hyperparameter Tuning

So MANY choices
- Which feature transformation?
- Which model architecture?
- Which hyperparameters?
Machine Learning Pipeline

Data/Feature Preprocessing → Model Selection → Hyperparameter Tuning

AutoML
- Auto Feature Generation
- Neural Architecture Search
- Hyperparameters Optimization
- Meta Learning
Automated Machine Learning

AutoML: How to automate the process of applying machine learning components to various real-world tasks?
Automated Machine Learning

**Inductive bias (prior $\alpha$):** how we represent data, which kinds of models to be considered, how to tune hyper-parameter, how to transfer knowledge across tasks, etc...

Traditional ML:

$$\theta^* = \arg\max_{\theta} P(D|\theta) \cdot P(\theta|\alpha)$$

where $\alpha$ is given by experts

AutoML: bilevel optimization

$$\theta^* = \arg\max_{\theta} P(D|\theta) \cdot P(\theta|\alpha^*)$$

where $\alpha^* = \arg\max_{\alpha} f(D', \theta^*)$

$D' = D$ for HPO,

$D'$ is validation data for NAS, AutoFeature,

$D'$ is data of other tasks for Meta-learning.
Tutorial Schedule

Yaliang Li, Background and Overview of AutoML Hyperparameter Optimization

Zhen Wang, Neural Architecture Search Meta-Learning

Yuexiang Xie, Automatic Feature Generation

Ce Zhang, VolcanoML: End-to-End AutoML via Scalable Search Space Decomposition

Bolin Ding, Machine Learning Guided Database
Hyperparameter Optimization (HPO)
Hyperparameter Optimization

Model Training

Hyperparameter Optimization

Best Hyperparameter

Best Model
Hyperparameter Configuration v.s. Schedule

- Hyperparameter configuration search methods find a **fixed** hyperparameter setting to maximize the model performance.

- Hyperparameter schedule search methods seek a **dynamic** hyperparameter schedule in the model training process.
Hyperparameter Optimization

- Hyperparameter Configuration
  - Random search, Grid Search
  - Successive-halving, Hyperband
  - Bayesian optimization

- Hyperparameter Schedule
  - Population-based training
  - Hypergradient
Search Methods

(a) Grid search.

(b) Random search.

Successive-Halving

- Uniformly allocate *a budget* to a set of hyperparameter configurations
- Evaluate the performance of all configurations
- Throw out the worst half

Repeat until one configuration remains

Hyperband

• Successive-Halving needs to determine the number of configurations (i.e., $n$)
• Outer loop
  • Grid search for different $n$
• Inner loop
  • Successive-Halving for given $n$ configs
  • s.t. at least one config is trained for $R$

Bayesian Optimization

Given some tried \{\text{hyperparameter, performance}\} pairs, which hyperparameter should be the next one to try?
Given some tried \{hyperparameter, performance\} pairs, which hyperparameter should be the next one to try?

Independence assumption

Follow a certain distribution

Bayesian Optimization
Bayesian Optimization

Fit a probabilistic function $f(x)$ to model \(x=\text{hyperparameter}, \ f(x)=\text{performance}\)

- Function $f(x)$ isn’t required to be convex, differentiable
- Rich theoretical results: convergence, sync v.s. async, various model choices

- Exploration-exploitation trade-off
- Costly
Hyperparameter Optimization

- Hyperparameter Configuration
  - Random search, Grid Search
  - Successive-halving, Hyperband
  - Bayesian optimization

- Hyperparameter Schedule
  - Population-based training
  - Hypergradient
Hyperparameter Schedule


Practical Challenge (1)

Hyperparameter Optimization → Best Hyperparameter
Model Training → Best Model

Model Size
Data Size
ABC: Sampling

A New Method: $\varepsilon$GE

Each category of hyperparameter optimization methods has its advantages and disadvantages. Can we adaptively combine them and utilize their advantages for different tasks?

Existing methods

- Search-strategy based: Successive-halving, Hyperband, etc.
- Evolutionary algorithm: Population Based Training, etc.
- Bayesian optimization

A New Method: $\varepsilon$GE

- Random strategy: randomly choose a configuration with probability $\varepsilon$
- Greedy strategy: choose the best configuration
- Evolution strategy: choose the best configuration and perturb it with mutation and crossover

- Choose $C'$
- Increase $r'$
- Train and test model: $P'$
- Output $C$ with the best $P$
HPO: Sampling method-$\varepsilon$GE

The task-adaptively combination of different hyperparameter optimization methods leads to faster solutions!

- A soft version of Hyperband
- Evolutionary operation
- A simplified version of Bayesian optimization (i.e., local smoothness assumption)

Practical Challenge (2)

Mutation-driven **global** search
PBT, KDD2019

Hypergradient-guided **local** search
STN, ICLR2019
Hyperparameter Schedule

Trade-off between Evolutionary algorithm (PBT) and Hyper-gradient based method:
• Hyper-gradient based method performs better than PBT on the smooth optimization problems.
• Hyper-gradient based method performs worse than PBT on the cases of many local minima (non-smooth).

How to learn a good trade-off between the global search and local search?
HyperMutaion (HPM)

Randomly Initialization → Hyper Training → Hyper Training → Learnable Mutation → Hyper Training → Learnable Mutation → Hyper Training → ... 

\[ S_0^k (\theta_0^k, h_0^k), k \in \{1, 2, 3\} \]

\[ \nabla \theta_1 \leftarrow \nabla h_1 \]

\[ \nabla \theta_2 \leftarrow \nabla h_2 \]

\[ \nabla \theta_3 \leftarrow \nabla h_3 \]

\[ \theta_t, h_t \]

Hypertraining: a joint optimization over $\theta$ and $h$

Exploit by a truncation selection

Randomly Initialization → Hyper Training → Learnable Mutation → Hyper Training → Learnable Mutation → Hyper Training → ...

$S^k_0(\theta^k_0, h^k_0), k \in \{1,2,3\}$

Explore by the learnable mutation

Randomly Initialization → Hyper Training → Learnable Mutation → Hyper Training → Learnable Mutation → Hyper Training → ...

$S_0^k(\theta_0^k, h_0^k), k \in \{1, 2, 3\}$

$\nabla \theta_{t-1}^1 \vert_{h_{t-1}^1} \rightarrow S_t^1$

$\nabla \theta_{t-1}^2 \vert_{h_{t-1}^2} \rightarrow S_t^2$

$\nabla \theta_{t-1}^3 \vert_{h_{t-1}^3} \rightarrow S_t^3$

$h_t^k = \Phi(h_t^k, h_t^*) = \alpha \odot h_t^*$

Activate student  Top student  Middle student  Bottom student

Learning mutations with a teacher network

- Student-teaching schema

- Teacher model with attention networks

\[\alpha = 1 + \tanh(c), \quad c = W \text{softmax}(V^T h^k_t),\]
Continue hypertraining after exploit & explore

Randomly Initialization → Hyper Training →... → Learnable Mutation → Hyper Training → Learnable Mutation → Hyper Training →...

\[ S^k_0 (\theta^k_0, h^k_0), k \in \{1,2,3\} \]
Experiments on test functions

Figure: (a)-(b) The mean performance computed by different methods along with the standard deviation over 10 trials, in terms of different given budget of iterations. (c) The average mutation values learned by HPM over 10 trials. In each trial, HPM runs 30 iterations in total with a population size of 5, resulting in 6 training steps and 5 mutations.
Experiments on test functions

**Figure**: (a)-(b) The mean performance computed by different methods along with the standard deviation over 10 trials, in terms of different given budget of iterations. (c) The average mutation values learned by HPM over 10 trials. In each trial, HPM runs 30 iterations in total with a population size of 5, resulting in 6 training steps and 5 mutations.
# Experiments on benchmark datasets

Table 1: Performance comparison for the image classification task on the CIFAR10 dataset by validation/test loss and the language modeling task on the PTB corpus dataset by perplexity (PPL).

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR10</th>
<th></th>
<th>PTB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Val Loss</td>
<td>Test Loss</td>
<td>Val PPL</td>
<td>Test PPL</td>
</tr>
<tr>
<td>Fixed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grid Search</td>
<td>0.7940</td>
<td>0.8090</td>
<td>97.32</td>
<td>94.58</td>
</tr>
<tr>
<td>Random Search</td>
<td>0.9210</td>
<td>0.7520</td>
<td>84.81</td>
<td>81.86</td>
</tr>
<tr>
<td>Bayesian Optimization</td>
<td>0.6360</td>
<td>0.6510</td>
<td>72.13</td>
<td>69.29</td>
</tr>
<tr>
<td>Hyperband [20]</td>
<td>0.7156</td>
<td>0.7491</td>
<td>71.25</td>
<td>68.39</td>
</tr>
<tr>
<td>Schedule</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PBT [15]</td>
<td>0.6253</td>
<td>0.6437</td>
<td>72.07</td>
<td>69.33</td>
</tr>
<tr>
<td>STN [23]</td>
<td>0.5892</td>
<td>0.5878</td>
<td>71.49</td>
<td>68.29</td>
</tr>
<tr>
<td>HPM w/o T</td>
<td>0.5724</td>
<td>0.5802</td>
<td>73.18</td>
<td>70.48</td>
</tr>
<tr>
<td>HPM</td>
<td><strong>0.5636</strong></td>
<td><strong>0.5649</strong></td>
<td><strong>70.49</strong></td>
<td><strong>67.88</strong></td>
</tr>
</tbody>
</table>
Takeaways

- Hyperparameter Configuration
  - Random search, Grid Search
  - Successive-halving, Hyperband
  - Bayesian optimization

- Hyperparameter Schedule
  - Population-based training
  - Hypergradient
  - HyperMutation (HPM)

Diagram:
- Model Training
- Best Model
- Hyperparameter Optimization
- Best Hyperparameter
Future Directions

- Faster, Green
  - HPO via Meta-Learning

- HPO for a specific domain
  - a group of algorithm, e.g. Graph-related

- Interactive, Human-in-the-loop
Neural Architecture Search (NAS)
Neural Architecture Search

What is neural architecture search (NAS)?

- To find the optimal topology and/or size configuration for the neural network.
  - E.g., select a filter from \(\{\text{CNN}_{3 \times 3}, \text{CNN}_{5 \times 5}, \text{DilatedCNN}_{5 \times 5}\}\).
  - E.g., determine the depth and width of a neural network.

Why NAS?

- Architecture matters a lot on the performance!
- The choices cannot be exhausted.
- Useful prior knowledge, e.g., the invariance possessed by the task, has been exploited.

Figure: Image classification on ImageNet (source: [https://paperswithcode.com/sota/image-classification-on-imagenet](https://paperswithcode.com/sota/image-classification-on-imagenet)).
Elements of NAS

- Search space
  - All the possible configurations.
  - E.g., filter size, activation functions, depth, etc.

- Search strategy
  - How to utilize experience?
  - How to propose new configuration to try?
  - E.g., RL, ES, and differentiable search.

- Performance estimation strategy
  - How to evaluate a configuration?
  - E.g., standard training and surrogate objective.
And the Theme of NAS

- **Search space**
  - All the possible configurations.
  - E.g., filter size, activation functions, depth, etc.

- **Search strategy**
  - How to utilize experience?
  - How to propose new configuration to try?
  - E.g., RL, ES, and differentiable search.

- **Performance estimation strategy**
  - How to evaluate a configuration?
  - E.g., standard training and surrogate objective.

**Exploitation v.s. Exploration**

Incorporating prior knowledge reduces search space but makes it constrained to some extent, e.g., Inception-v2/3 \(\rightarrow\) stacked cells [Zoph et al. 2018].

Instead of asymptotic regret, practitioners balance the exploitation and exploration to achieve best solution under a given finite horizon.

Standard training&validation is expensive but accurate.

The proposed surrogate objectives are efficient but less correlated.
Pioneer Works of NAS

- Search space
  - Consider both CNN and RNN cells.
  - The configuration of each layer can be determined respectively.

- Search strategy
  - RL with the policy parameterized by a RNN.

- Performance estimation strategy
  - Standard train&validation

Figure: An overview of the trial-and-error process of NAS.

Figure: How the controller (i.e., a RNN) samples a CNN with skip connection.

Pioneer Works of NAS

- Search space
  - Consider both CNN and RNN cells.
  - The configuration of each layer can be determined respectively.

- Search strategy
  - RL with the policy parameterized by a RNN.

- Performance estimation strategy
  - Standard train&validation

- Unfold the gain of NAS 😄 and also its pain ☹️
  - Searched CNN and RNN cells achieve competitive performances against manually designed architectures on CIFAR-10 and PTB respectively.
  - Searched architecture can be transferred to other tasks.
  - Trained 12,800 models in total on 800 GPUs.

Weight Sharing for One-shot NAS

- **Weight sharing**
  - Represent NAS’s search space using a single DAG.
  - An architecture can be realized by taking a subgraph.
  - E.g., deducing a RNN cell as follow:

- **One-shot NAS**
  - Each architecture (i.e., subgraph) is evaluated by inheriting the shared parameters.
  - Shared parameters are trained with sampled architecture.
  - Parameters and the controller are updated alternatively.

- **Advantage and concern**
  - ENAS [Pham and Guan et al., 2018] uses 10h of one GTX1080Ti, which is 1000x faster than [Zoph et al., 2017].
  - Does the performance of a stand-alone training correlate with that of one-shot NAS [Bender et al., 2018, Zhang et al., 2020]?
Differentiable NAS

- **Continuous relaxation**
  - Each edge denotes a mixture of ops in $O = \{\text{CNN}_{3\times3}, \text{DilatedCNN}_{3\times3}, \text{Zero}, \text{Identity}, \ldots\}$. 
  - For each edge $(i, j)$, they parameterize the weights of ops by architecture parameter $\alpha^{(i,j)}$. 
  - Suppose the tensor at node $i$ is $x$, then the tensor propagated to node $j$ will be:
    \[
    \tilde{o}^{(i,j)}(x) = \sum_{o \in O} \frac{\exp(\alpha^{(i,j)_o})}{\sum_{o' \in O} \exp(\alpha^{(i,j)_{o'}})} o(x)
    \]

- **Differentiable learning**
  - Formulated as a bilevel optimization problem:
    \[
    \min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha) \\
    \text{s.t. } w^*(\alpha) = \arg\min_{w} \mathcal{L}_{train}(w, \alpha)
    \]
  - Regarded as a Stackelberg game
    - Architecture parameters as leader
    - Model parameters as follower

Figure: An overview of DARTS. (a) Operations are initially unknown. (b) Continuous relaxation. (c) architecture parameters are optimized jointly. (d) Inducing the final architecture.

Differentiable NAS

Differentiable learning (contd’)

\[
\min_{\alpha} \quad L_{val}(w^*(\alpha), \alpha) \\
\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \quad L_{train}(w, \alpha)
\]

- No way to estimate the \( \nabla_{\alpha} L_{val}(w^*(\alpha), \alpha) \) exactly.
- DARTS approximates the gradient by looking ahead one-step for \( \omega \) like meta-learning.
- It is further simplified by treating the parameters equally [Li et al., 2021].

Deriving discrete architecture

- Retain the top-k strongest predecessors for each node \( j \) where strength of \((i, j)\) is defined as:
  \[
  \text{argmax}_{o \in O} \frac{\exp\{\alpha_{o}^{(i,j)}\}}{\sum_{o' \in O} \exp\{\alpha_{o'}^{(i,j)}\}}.
  \]
- Replace each edge by the most likely op:
  \[
  o^{(i,j)} = \text{argmax}_{o \in O} \alpha_{o}^{(i,j)}
  \]

Improve DARTS by annealing and pruning

\[
\Phi_o(\alpha^{(i,j)}; T) = \frac{\exp\left\{ \frac{\alpha_{o}^{(i,j)}}{T} \right\}}{\sum_{o' \in O} \exp\left\{ \frac{\alpha_{o'}^{(i,j)}}{T} \right\}}
\]

Dealing with Scalability Issue

- Horrible memory occupation of one-shot NAS
  - The supergraph cannot fit into GPU memory for large datasets.
  - Usually search architecture on CIFAR-10 and transfer to ImageNet.

- Binarized architecture
  - Transform real-valued path weights to binary gates.
  - Only one path is active in memory at runtime.

Figure: ProxylessNAS directly optimizes neural architecture on target task and hardware.

Figure: Note the straight-through estimator (STE) trick.

Rethinking the Search Space of NAS

- Explore less constrained search spaces [Xie et al. 19]
  - Consider stochastic network generator, e.g., ER, BA, and WS.
  - All yield >73% mean accuracy on ImageNet with a low variance!
  - Presented graph damage ablation.

- Design search space [Radosavovic et al. 20]
  - Evaluate a search space by its error distribution.
  - Input a search space and output a refined one.

Figure: randomly remove one node/edge.

Figure: two steps of refinement with the error distribution constantly improved.
Rethinking the Search Space of NAS

- From the view of graph structure [You et al. 20a]
  - From DAG to relational graph.
  - Sweet spots are consistent across different datasets and architectures.

Figure: proposed WS-flex provides a larger search space.
Rethinking the Search Space of NAS

- From the view of graph structure [You et al. 20a]
  - From DAG to relational graph.
  - Sweet spots are consistent across different datasets and architectures.

Figure: Key results.

Graph Structure of Neural Networks. ICML, 2020, [http://proceedings.mlr.press/v119/you20b/you20b.pdf](http://proceedings.mlr.press/v119/you20b/you20b.pdf)
Size Search Space

- **Model scaling**
  - Keep the architecture but adjust the size:
    - Depth $L$
    - Width $C$
    - And resolution $H, W$
  - Maximize the performance w.r.t. the size.

- **EfficientNet [Tan et al. 19]**

$$\max_{d,w,r} \text{Accuracy}(N(d, w, r))$$

$$s.t. \quad N(d, w, r) = \bigcap_{i=1 \ldots s} \mathcal{F}_{d_i} \cdot \hat{L}_i \left( X_{\{r, \hat{H}_i, r, \hat{W}_i, w, \hat{C}_i\}} \right)$$

- Memory($N$) $\leq$ target_memory
- FLOPS($N$) $\leq$ target_flops

- Compound scaling method: $d = \alpha^{\phi}$, $\omega = \beta^{\phi}$, $r = \gamma^{\phi}$ where $\alpha \times \beta^2 \times \gamma^2 \approx 2$, $\alpha \geq 1$, $\beta \geq 1$, $\gamma \geq 1$.
- Step1: Fix $\phi = 1$ and do a small grid search for $\alpha, \beta, \gamma$. Step2: Fix $\alpha, \beta, \gamma$ and scale up $\phi$. 

Figure: Model size v.s. ImageNet accuracy.
From CNN/RNN to GNN

- **Uniqueness in search space**
  - More dimensions of choices:
    - Micro: mainly aggregation and combine functions.
    - Macro: how node embeddings in each layer produce the final one.
  - Nodes are not independent, so how about in a node-wise manner?

- **Challenges of weight-sharing one-shot NAS**
  - Different options lead to quite different output statistics [Zhou et al. 19].

- **Transfer across datasets and tasks [You et al. 20b]**
  - Collect 32 (diverse) tasks.
  - Use anchor models to calculate task similarities.

\[
\begin{align*}
\mathbf{m}_i^{(l)} &= \text{AGG}^{(l)} \left( \{a_{ij}^{(l)} \mathbf{W}^{(l)} \mathbf{h}_j^{(l)} \mid \forall j \in \mathcal{N}(i) \} \right) \\
\mathbf{h}_i^{(l+1)} &= \sigma \left( \text{COMBINE}^{(l)} \left[ \mathbf{m}_i^{(l)}, \mathbf{h}_i^{(l)} \right] \right),
\end{align*}
\]

Figure: General message passing.

Figure: Comparing the correlations.
Beyond Accuracy: Efficiency and Robustness

Making latency differentiable [Cai et al. 19]

\[ E[\text{Latency}] = \alpha \times F(\text{conv.3x3}) + \beta \times F(\text{conv.5x5}) + \gamma \times F(\text{identity}) + \cdots \]

\[ E[\text{latency}] = \sum_i E[\text{latency}_i] \]

\[ \text{Loss} = \text{Loss}_{CE} + \lambda_1 ||w||_2^2 + \lambda_2 E[\text{latency}] \]

Figure: Introducing latency regularization loss.

Searching robust architecture [Guo et al. 20]

Figure: Performance of 1k sampled architecture.

Figure: Analysis of top 300 robust v.s. non-robust architectures.
Beyond Accuracy: Compressed Model Search

- Pre-trained language model such as BERT achieves great performance on various tasks, but it is difficult to be deployed to real-time applications.
- Can we task-adaptively compresses original BERT for different tasks?

<table>
<thead>
<tr>
<th>Method</th>
<th>Averaged Performance</th>
<th>Inference Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>82.5</td>
<td>1x</td>
</tr>
<tr>
<td>BERT-PKD</td>
<td>80.6</td>
<td>1.9x</td>
</tr>
<tr>
<td>DistillBERT</td>
<td>76.8</td>
<td>3.0x</td>
</tr>
<tr>
<td>TinyBERT</td>
<td>80.6</td>
<td>9.4x</td>
</tr>
<tr>
<td>AdaBERT</td>
<td>80.1</td>
<td>12.7x ~ 29.3x</td>
</tr>
</tbody>
</table>

The proposed AdaBERT achieves significant speedup in inference time while maintaining comparable performance compared to uncompressed model.

Beyond Accuracy: Compressed Model Search

Figure: Searched structures of compressed models for different tasks

Table: Performance of searched structures across different tasks

<table>
<thead>
<tr>
<th>Models</th>
<th>Tasks</th>
<th>SST-2</th>
<th>MRPC</th>
<th>QQP</th>
<th>MNLI</th>
<th>QNLI</th>
<th>RTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SST-2</td>
<td>91.9</td>
<td>78.1</td>
<td>58.6</td>
<td></td>
<td>63.7</td>
<td>74.1</td>
<td>53.8</td>
</tr>
<tr>
<td>MRPC</td>
<td>81.5</td>
<td></td>
<td>84.7</td>
<td>68.9</td>
<td>75.7</td>
<td>82.2</td>
<td>60.3</td>
</tr>
<tr>
<td>QQP</td>
<td>81.9</td>
<td></td>
<td>84.1</td>
<td>70.5</td>
<td>76.2</td>
<td>82.5</td>
<td>60.5</td>
</tr>
<tr>
<td>MNLI</td>
<td>82.1</td>
<td>81.5</td>
<td>66.8</td>
<td></td>
<td>80.4</td>
<td>86.1</td>
<td>63.2</td>
</tr>
<tr>
<td>QNLI</td>
<td>81.6</td>
<td>82.3</td>
<td>67.7</td>
<td></td>
<td>79.1</td>
<td>87.2</td>
<td>62.9</td>
</tr>
<tr>
<td>RTE</td>
<td>82.9</td>
<td>81.1</td>
<td>66.5</td>
<td></td>
<td>79.6</td>
<td>86.0</td>
<td>64.1</td>
</tr>
</tbody>
</table>

These results demonstrate that the proposed AdaBERT compresses original BERT adaptively for different downstream tasks.

NAS Benchmarks

- **NAS-Bench-101** [Ying et al. 19]
  - Provides a lookup table for the 423k architectures.
  - Including their train/valid/test accuracies, number of parameters, and training time.

- **NATS-Bench** [Dong et al. 21]
  - Search space considers both size and topology factors.

Figure: The search space of NATS-Bench.
NAS Benchmarks

- **NAS-Bench-101** [Ying et al. 19]
  - Provides a lookup table for the 423k architectures.
  - Including their train/valid/test accuracies, number of parameters, and training time.

- **NATS-Bench** [Dong et al. 21]
  - Search space considers both size and topology factors.

<table>
<thead>
<tr>
<th></th>
<th>#Unique Architectures</th>
<th>#Datasets</th>
<th>Diagnostic Information</th>
<th>Search Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAS-Bench-101</td>
<td>423K</td>
<td>1</td>
<td>X</td>
<td>topology</td>
</tr>
<tr>
<td>St in NATS-Bench</td>
<td>6.5K</td>
<td>3</td>
<td>fine-grained</td>
<td>topology</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>information</td>
<td></td>
</tr>
<tr>
<td>Ss in NATS-Bench</td>
<td>32.8K</td>
<td>3</td>
<td>fine-grained</td>
<td>size</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>information</td>
<td></td>
</tr>
</tbody>
</table>

Figure: Comparison the benchmarks.
Takeaways

- Search space
  - Layer by layer
  - Pre-defined restricted design space
  - Pre-defined size
  - Repeated normal & reduction cell
  - Search for design space
  - Also search for optimal size

- Search strategy
  - Trial-and-error, e.g., RL and ES
  - One-shot NAS
  - Differentiable (+sampling ops)

- Performance estimation strategy
  - Stand training & validation
  - Single objective
  - With weight-sharing
  - Multiple objectives
Future Directions

- **Reduce the variance of one-shot NAS**
  - The interference between child models is a main factor [Zhang et al. 2020].
  - E.g., sharing unless some condition(s) are satisfied.

- **Select the truly useful architecture**
  - The magnitude of architecture parameters does not necessarily indicate how much the operation contributes to the supernet’s performance [Wang and Cheng et al. 2021].

Figure: Validation performance of each child model during the last 120 steps.
Beyond NAS: From static to dynamic neural architecture.

- Fine-grained tuning.
- Mainly focusing on CNNs and efficiency issue now.
References of NAS

- [You et al. 2020a] Graph Structure of Neural Networks. ICML. 2020a.
Meta-Learning
Meta-learning

What is meta-learning?

• Training on a meta-dataset consisting of many datasets, where each is a different task.
• Extract prior knowledge from it that accelerates the learning of new tasks.

Figure: Example of how meta-learning works (source: https://cs330.stanford.edu/slides/cs330_metalearning_bbox_2020.pdf).
When Meta-learning Meets AutoML

- AutoML as a service
  - What if users do not have a large dataset for training a deep model?
  - What if users want to quickly learn a new task?

Figure: The distribution of the scales of datasets. (source: https://cs330.stanford.edu/).
When Meta-learning Meets AutoML

- AutoML as a service
  - Assume different tasks share some common principles.
  - Can we exploit the cumulated experience?

Hyperparameter Optimization

NAS

AutoML

Automatic Feature Generation

Meta-Learning 😄
Meta-learning Basics

- Exploit the meta-dataset
  - Conventional ML: \( \arg \max_{\phi} \log p(\phi | D) \)
  - Meta-learning: \( \arg \max_{\phi} \log p(\phi | D, D_{\text{meta-train}}) \)

- Replace the meta-dataset by meta-parameters
  - Sufficient to represent the meta-dataset.

\[
\text{learn meta-parameters } \theta: p(\theta | D_{\text{meta-train}}) \\
\text{whatever we need to know about } D_{\text{meta-train}} \text{ to solve new tasks} \\
\log p(\phi | D, D_{\text{meta-train}}) = \log \int_{\Theta} p(\phi | D, \theta) p(\theta | D_{\text{meta-train}}) d\theta \\
\approx \log p(\phi | D, \theta^*) + \log p(\theta^* | D_{\text{meta-train}}) \\
\arg \max_{\phi} \log p(\phi | D, D_{\text{meta-train}}) \approx \arg \max_{\phi} \log p(\phi | D, \theta^*)
\]

\( \theta^* = \arg \max_{\theta} \log p(\theta | D_{\text{meta-train}}) \)

this is the meta-learning problem

this is the adaptation problem
Optimization-based Meta-learning

- **Adaptation problem**
  - Acquire $\phi_i$ via optimization $\phi_i = \arg\max_\phi \log p(D_i^{tr}|\phi) + \log p(\phi|\theta)$.
  - $\theta$ serves as a prior.

- **Which form of prior to take?**
  - Initialization and fine-tuning!

\[
\begin{align*}
\text{Fine-tuning} & : \quad \phi \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta, D_i^{tr}) \\
\text{Meta-learning} & : \quad \min_\theta \sum_{\text{task } i} \mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}(\theta, D_i^{tr}), D_i^{ts})
\end{align*}
\]

- $g_{\text{MAML}} = g_2 = \bar{g}_2 - \alpha \bar{H}_2 \bar{g}_1 + O(\alpha^2)$
- $g_{\text{FOMAML}} = g_2 = \bar{g}_2 - \alpha \bar{H}_2 \bar{g}_1 + O(\alpha^2)$
- $g_{\text{Reptile}} = g_1 + g_2 = \bar{g}_1 + \bar{g}_2 - \alpha \bar{H}_2 \bar{g}_1 + O(\alpha^2)$

Where $g_i = \frac{\partial}{\partial \theta_i} L(\theta_i, D^{tr})$, $\bar{g}_i = \frac{\partial}{\partial \theta_1} L(\theta_i, D^{tr})$, $\bar{H}_i = \frac{\partial}{\partial \theta_1} \bar{g}_i$ (Hessian w.r.t. $\theta_1$)

Figure: Illustrating the idea of optimization-based meta-learning (source: [https://arxiv.org/pdf/1703.03400.pdf](https://arxiv.org/pdf/1703.03400.pdf)).
Optimization-based Meta-learning

Probabilistic interpretation
- Maximize a posterior (MAP) with $\theta$ as the prior.

MAML [Finn et al. 17] approximates hierarchical Bayesian inference!
- Gradient descent with early stop = MAP inference under Gaussian prior with mean at initial parameters.
- Other forms, e.g.,

$$\phi \leftarrow \min_{\phi'} L(\phi', D^{tr}) + \frac{\lambda}{2} ||\theta - \phi'||^2$$

Figure: Probabilistic interpretation of optimization-based meta-learning (source: https://cs330.stanford.edu/).

😄 Model-agnostic
😄 Maximally expressive with sufficiently deep neural networks
☹ Typically requires second-order computation/memory intensive
Model-based Meta-learning

- Adaptation problem
  - From solving optimization problem to black-box adaptation $\phi_i = f_\theta(D_i^{tr}) = \arg\max_\phi \log p(\phi|D_i^{tr}, \theta)$
  - Train a neural networks to represent $p(\phi_i|D_i^{tr}, \theta)$
    - E.g., RNN, Neural Turing Machine, memory-augmented NN [Santoro et al. 16], etc.

Metric-based Meta-learning

- Use Non-parametric learner

![Diagram](https://cs330.stanford.edu/slides/cs330_nonparametric_2020.pdf)


- Entirely feedforward
- Easy to optimize
- Harder to generalize to varying k-ways (especially for very large k)
Metric-based Meta-learning

- Use Siamese neural networks

  - Meta-training: binary classification.
  - Meta-test: k-way classification.

Figure: Architecture of Siamese neural networks and its application to one-shot learning.

Metric-based Meta-learning

- Match the train&test phases by Matching networks
  - Fix the mismatch between meta-training and meta-test.
  - Map a (support) set $S = \{(x_i, y_i)\}$ to a classifier:
    $$ P(y|\hat{x}, S) = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i $$
    - The attention mechanism $a(\cdot, \cdot)$ fully specifies the classifier.

Figure: Architecture of Matching network.

GPT-3: meta-learning as pre-training

What’s the meta-dataset?
- Crawled text corpora.
- $D_i^{tr}$: sequence of characters, $D_i^{ts}$: the following sequence of characters.

What’s the meta-learning problem?
- Put different tasks all in the form of text.
- Thus trained on language generation tasks.

What’s the extracted prior knowledge?
- A “Transformer” model as the initialization.

Figure: The model is far from perfect (source: https://github.com/shreyashankar/gpt3-sandbox/blob/master/docs/priming.md).
Generalization v.s. Customization

Key assumption of meta-learning

- Meta-training and meta-test tasks are drawn i.i.d. from the same task distribution.
- E.g., Omniglot:
  - 1623 characters from 50 different alphabets.
  - 20 instances for each character.

Figure: Characters of different alphabets (source: https://omniglot.com/).
Generalization v.s. Customization

- **Key assumption of meta-learning**
  - Meta-training and meta-test tasks are drawn i.i.d. from the same task distribution.
  - E.g., Omniglot:
    - 1623 characters from 50 different alphabets.
    - 20 instances for each character.
  - Can NOT be strictly satisfied!

- **Experience cumulated on the cloud**
  - Different user experiments can be quite different.
  - Learning a global prior may be insufficient.
Relational Meta-Learning

The proposed relational meta-learning method can capture the relations among different tasks, which enhances the effectiveness of meta-learners.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Data: Bird</th>
<th>Data: Texture</th>
<th>Data: Aircraft</th>
<th>Data: Fungi</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAML</td>
<td>53.94 ± 1.45%</td>
<td>31.66 ± 1.31%</td>
<td>51.37 ± 1.38%</td>
<td>42.12 ± 1.36%</td>
</tr>
<tr>
<td>MetaSGD</td>
<td>55.58 ± 1.43%</td>
<td>32.38 ± 1.32%</td>
<td>52.99 ± 1.36%</td>
<td>41.74 ± 1.34%</td>
</tr>
<tr>
<td>MT-Net</td>
<td>58.72 ± 1.43%</td>
<td>32.80 ± 1.35%</td>
<td>47.72 ± 1.46%</td>
<td>43.11 ± 1.42%</td>
</tr>
<tr>
<td>MUMOMAML</td>
<td>56.82 ± 1.49%</td>
<td>33.81 ± 1.36%</td>
<td>53.14 ± 1.39%</td>
<td>42.22 ± 1.40%</td>
</tr>
<tr>
<td>HSML</td>
<td>60.98 ± 1.50%</td>
<td>35.01 ± 1.36%</td>
<td>57.38 ± 1.40%</td>
<td>44.02 ± 1.39%</td>
</tr>
<tr>
<td>ProtoNet</td>
<td>54.11 ± 1.38%</td>
<td>32.52 ± 1.28%</td>
<td>50.63 ± 1.35%</td>
<td>41.05 ± 1.37%</td>
</tr>
<tr>
<td>TADAM</td>
<td>56.58 ± 1.34%</td>
<td>33.34 ± 1.27%</td>
<td>53.24 ± 1.33%</td>
<td>43.06 ± 1.33%</td>
</tr>
<tr>
<td>ARML</td>
<td>\textbf{62.33 ± 1.47%}</td>
<td>\textbf{35.65 ± 1.40%}</td>
<td>\textbf{58.56 ± 1.41%}</td>
<td>\textbf{44.82 ± 1.38%}</td>
</tr>
</tbody>
</table>

Summary and Future Directions

- How to utilize existing experience---meta-learning
  - Learn a meta-parameter, so that we can quickly transfer to new task.
  - Optimization-based, model-based, metric-based

- What if tasks are heterogeneous?
  - Trade-off between generalization v.s. customization

- Use meta-learning for improving real-world services
  - AutoML as a service has cumulated a lot of experience.
  - Learning tasks on different domains and/or with different models share some intrinsic patterns of machine learning.
  - What kinds of features are transferable? How to represent a task, a model, and a objective?
References of Meta-learning

Auto Feature Generation
In practice, many data scientists search for useful interactive features in a trial-and-error manner, which has occupied a lot of their workloads. Therefore, automatic feature generation (AutoFeature), as one major topic of automated machine learning (AutoML), has received a lot of attention from both academia and industry.
Automatic Feature Generation

- Industries such as healthcare and finance need interpretability
- Can be applied to train lightweight models for real-time requirement
- The number of possible interactive features is too large to be traversed ($O(2^m)$ for $m$ original features)
The related works on automatic feature generation can be roughly divided into two categories:

- **DNN-based methods** design specific neural architectures to express the interactions among different features.
  - Implicit feature generation
  - One-shot training course
  - Lack of interpretable rules for feature interactions

- **Search-based methods** focus on designing different search strategies that prune as much of the candidates to be evaluated as possible, while aiming to keep the most useful interactive features.
  - Explicit feature generation
  - Trial-and-error training manner
  - Need lots of time and computing resource
AutoInt

- Map the original features into low-dimensional feature space and model the high-order feature interactions via self-attention.

1. **Input Layer:**
   Each feature field is represented as an one-hot vector (for categorical feature) or a scalar value (for numerical feature).

2. **Embedding Layer:**
   To transform the sparse and high-dimension features into a low-dimensional feature space via a learnable embedding matrix.

---

AutoInt

- Map the original features into low-dimensional feature space and model the high-order feature interactions via self-attention.

**3 Interacting Layer:**
The multi-head key-value attention mechanism is adopted to capture the interactions between different features.

\[
\alpha_{m,k}^{(h)} = \frac{\exp(\psi^{(h)}(e_m, e_k))}{\sum_{l=1}^{M} \exp(\psi^{(h)}(e_m, e_l))},
\]

\[
\psi^{(h)}(e_m, e_k) = \langle W^{(h)}_{\text{Query}} e_m, W^{(h)}_{\text{Key}} e_k \rangle,
\]

\[
\tilde{c}_m^{(h)} = \sum_{k=1}^{M} \alpha_{m,k}^{(h)} (W^{(h)}_{\text{Value}} e_k),
\]

The architecture of interacting layer.

AutoInt

- Experimental results on four real-world datasets show the advantages of AutoInt:
  - Performance comparison in offline AUC evaluation for click-through rate (CTR) prediction
  - Efficiency comparison
  - Explainable recommendations

Efficiency comparison on MovieLens-1M.

An instance of attention weights for feature interactions on MovieLens-1M.
Fi-GNN

- Fi-GNN proposes to represent multi-field features in a graph structure, and captures the feature interactions through node representation learning in the graph.
- Feature interaction via a graph view: nodes represent features and edges denote their interactions
- Model feature interactions via Graph Neural Networks (GNN)
- Attentional scoring for predictions

Overview of the proposed Fi-GNN.
Fi-GNN

- Feature interaction in Fi-GNN: The nodes interact with neighbors and update their states in a recurrent fashion.

**Feature Graph:**
The edge weights reflect the importance of interactions between the connected nodes (features), which are learned via an attention mechanism.

**Node Aggregation:**
The node aggregates the transformed information from neighbors and update its state according to the aggregated information and history via GRU and residual connection.

Fi-GNN: Modeling Feature Interactions via Graph Neural Networks for CTR Prediction. CIKM, 2019.
Fi-GNN

- Taking advantage of the strong representative power of graphs, Fi-GNN captures high-order feature interaction in an efficient way.

- Fi-GNN also provides good model explanations for CTR prediction.

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Model</th>
<th>AUC</th>
<th>RI-AUC</th>
<th>Logloss</th>
<th>RI-Logloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>First-order</td>
<td>LR</td>
<td>0.7820</td>
<td>3.00%</td>
<td>0.4695</td>
<td>5.43%</td>
</tr>
<tr>
<td>Second-order</td>
<td>FM [23]</td>
<td>0.7836</td>
<td>2.80%</td>
<td>0.4700</td>
<td>5.55%</td>
</tr>
<tr>
<td></td>
<td>AFM [34]</td>
<td>0.7938</td>
<td>1.54%</td>
<td>0.4584</td>
<td>2.94%</td>
</tr>
<tr>
<td>High-order</td>
<td>DeepCrossing [25]</td>
<td>0.8009</td>
<td>0.66%</td>
<td>0.4513</td>
<td>1.35%</td>
</tr>
<tr>
<td></td>
<td>NFM [8]</td>
<td>0.7957</td>
<td>1.57%</td>
<td>0.4562</td>
<td>2.45%</td>
</tr>
<tr>
<td></td>
<td>CrossNet [31]</td>
<td>0.7907</td>
<td>1.92%</td>
<td>0.4591</td>
<td>3.10%</td>
</tr>
<tr>
<td></td>
<td>CIN [15]</td>
<td>0.8009</td>
<td>0.63%</td>
<td>0.4517</td>
<td>1.44%</td>
</tr>
<tr>
<td></td>
<td>Fi-GNN (ours)</td>
<td><strong>0.8062</strong></td>
<td><strong>0.00%</strong></td>
<td><strong>0.4453</strong></td>
<td><strong>0.00%</strong></td>
</tr>
</tbody>
</table>

Performance comparison.

Heat map of attentional edge weights.
Automatic Feature Generation

- The related works on automatic feature generation can be roughly divided into two categories:
  - DNN-based methods
  - Search-based methods

DNN-based methods design specific neural architectures to express the interactions among different features.
  - Implicit feature generation
  - One-shot training course
  - Lack of interpretable rules for feature interactions

Search-based methods focus on designing different search strategies that prune as much of the candidates to be evaluated as possible, while aiming to keep the most useful interactive features.
  - Explicit feature generation
  - Trial-and-error training manner
  - Need lots of time and computing resource
AutoCross

- AutoCross searches useful feature interactions in the high-order interactive feature space by incrementally constructing local optimal feature set

- **Multi-granularity discretization**
  - Greedy & beam search
  - Field-wise logistic regression
  - Successive mini-batch gradient descent

Multi-granularity discretization:
- For automatic discretization, each numerical feature is discretized into several categorical features with different granularities.

An illustration of multi-granularity discretization.
AutoCross

AutoCross searches useful feature interactions in the high-order interactive feature space by incrementally constructing local optimal feature set

- Multi-granularity discretization
- **Greedy & beam search**
- Field-wise logistic regression
- Successive mini-batch gradient descent

**Greedy & beam search:**
- Tree-structured space with the original features as the root.
- The children are generated by added one pair-wise crossing to the parent.
- Only the most promising child will be expanded during the search

An illustration of the search space and beam search strategy.
AutoCross

- AutoCross searches useful feature interactions in the high-order interactive feature space by incrementally constructing local optimal feature set

  - Multi-granularity discretization
  - Greedy & beam search
  - Field-wise logistic regression
  - Successive mini-batch gradient descent

**Field-wise logistic regression:**
- For each node, the weights of the newly added interactive features are updated during training, while other weights are inherited from the parent and fixed.

**Successive mini-batch gradient descent:**
- The data are split into several blocks, and gradually added into the training process along with narrowing the candidate interactive features.

AutoCross

- The advantages of AutoCross:
  - Explicit high-order feature generation
  - Fast inference
  - Interpretability

The number of second/high-order interactive features.

Inference latency comparison.

<table>
<thead>
<tr>
<th>Method</th>
<th>Bank</th>
<th>Adult</th>
<th>Credit</th>
<th>Employee</th>
<th>Criteo</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC+LR</td>
<td>0.00048</td>
<td>0.00048</td>
<td>0.00062</td>
<td>0.00073</td>
<td>0.00156</td>
</tr>
<tr>
<td>AC+W&amp;D</td>
<td>0.01697</td>
<td>0.01493</td>
<td>0.00974</td>
<td>0.02807</td>
<td>0.02698</td>
</tr>
<tr>
<td>Deep</td>
<td>0.01413</td>
<td>0.01142</td>
<td>0.00726</td>
<td>0.02166</td>
<td>0.01941</td>
</tr>
<tr>
<td>xDeepFM</td>
<td>0.08828</td>
<td>0.05522</td>
<td>0.04466</td>
<td>0.06467</td>
<td>0.18985</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Data1</th>
<th>Data2</th>
<th>Data3</th>
<th>Data4</th>
<th>Data5</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC+LR</td>
<td>0.00367</td>
<td>0.00111</td>
<td>0.00185</td>
<td>0.00393</td>
<td>0.00279</td>
</tr>
<tr>
<td>AC+W&amp;D</td>
<td>0.03537</td>
<td>0.01706</td>
<td>0.04042</td>
<td>0.02434</td>
<td>0.02582</td>
</tr>
<tr>
<td>Deep</td>
<td>0.02616</td>
<td>0.01348</td>
<td>0.03150</td>
<td>0.01414</td>
<td>0.01406</td>
</tr>
<tr>
<td>xDeepFM</td>
<td>0.32435</td>
<td>0.11415</td>
<td>0.40746</td>
<td>0.12467</td>
<td>0.13235</td>
</tr>
</tbody>
</table>

AutoFIS

- AutoFIS automatically identifies important feature interactions for Factorization Models (FM).
  - **Search Stage**: Learn the relative importance of each feature interaction via architecture parameters within one full training process.
  - **Re-train Stage**: Remove the unimportance interactions and re-train the resulting neural networks.

Overview of AutoFIS.
AutoFIS

- Experiments on large-scale datasets demonstrate that AutoFIS can improve various FM based models in CTR prediction tasks.

<table>
<thead>
<tr>
<th>Model</th>
<th>AUC</th>
<th>log loss</th>
<th>top</th>
<th>Rel. Improv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM</td>
<td>0.8880</td>
<td>0.08881</td>
<td>100%</td>
<td>0</td>
</tr>
<tr>
<td>FwFM</td>
<td>0.8897</td>
<td>0.08826</td>
<td>100%</td>
<td>0.19%</td>
</tr>
<tr>
<td>AFM</td>
<td>0.8915</td>
<td>0.08772</td>
<td>100%</td>
<td>0.39%</td>
</tr>
<tr>
<td>FFM</td>
<td>0.8921</td>
<td>0.08816</td>
<td>100%</td>
<td>0.46%</td>
</tr>
<tr>
<td>DeepFM</td>
<td>0.8948</td>
<td>0.08735</td>
<td>100%</td>
<td>0.77%</td>
</tr>
<tr>
<td>AutoFM(2nd)</td>
<td>0.8944*</td>
<td>0.08665*</td>
<td>37%</td>
<td>0.72%</td>
</tr>
<tr>
<td>AutoDeepFM(2nd)</td>
<td>0.8979*</td>
<td>0.08560*</td>
<td>15%</td>
<td>1.11%</td>
</tr>
</tbody>
</table>

Performance comparison.

Correlations between the architecture parameters $\alpha$ and AUC.

FIVES

To possess both feature interpretability and search efficiency, the proposed method FIVES formulates the task of interactive feature generation as searching for edges on the defined feature graph.

1. Search Strategy

**Proposition 1.** Let $X_1, X_2$ and $Y$ be Bernoulli random variables with a joint conditional probability mass function, $p_{X_1, X_2 | y} := P(X_1 = x_1; X_2 = x_2 | Y = y)$ such that $x_1, x_2, y \in \{0, 1\}$. Suppose further that mutual information between $X_i$ and $Y$ satisfies $I(X_i; Y) < L$ where $i \in \{1, 2\}$ and $L$ is a non-negative constant. If $X_1$ and $X_2$ are weakly correlated given $y \in \{0, 1\}$, that is, $\left| \frac{\text{Cov}(X_1, X_2 | Y = y)}{\sigma_{X_1 | Y = y} \sigma_{X_2 | Y = y}} \right| \leq \rho$, we have

$$I(X_1, X_2; Y) < 2L + \log(2\rho^2 + 1). \tag{1}$$

Theoretical support for the search strategy.

- This proposition states that informative interactive features unlikely come from the uninformative lower-order ones.
- The theory motivates the bottom-up search strategy in FIVES: Searching for a group of informative $k$-order features from the interactions between original features and the group of $(k - 1)$-order features.
To possess both feature interpretability and search efficiency, the proposed method FIVES formulates the task of interactive feature generation as searching for edges on the defined feature graph.

2. Feature Graph

- To instantiate the proposed search strategy, the original features are conceptually regarded as a *feature graph* and their interactions are modeled by a designed GNN.
- Each node $n_i$ corresponds to a feature $f_i$. Each edge $e_{i,j}$ indicates an interaction between $n_i$ and $n_j$.

The constructed feature graph to represent high-order feature interactions.

FIVES

- To possess both feature interpretability and search efficiency, the proposed method FIVES formulates the task of interactive feature generation as searching for edges on the defined feature graph.

2. Feature Graph

- The feature graph consists of \( K \) subgraphs to represent high-order interactive feature. Each subgraph indicates a layer-wise interaction between features, represented by an adjacency matrix \( A^{(k)} \in \{0,1\}^{m \times m} \). The graph convolutional operator for aggregation are defined as:

\[
n_i^{(k)} = p_i^{(k)} \odot n_i^{(k-1)}, \quad \text{where} \quad p_i^{(k)} = \text{MEAN}_{j|A_{i,j}^{(k)}=1} \{W_j n_j^{(0)}\} \tag{1}
\]

- The node representation at \( k \)-th layer corresponds to the generated features:

\[
n_i^{(k)} = \text{MEAN}_{j|A_{i,j}^{(k)}=1} \{W_j n_j^{(0)}\} \odot n_i^{(k-1)} \approx \text{MEAN}_{(c_1,\ldots,c_k)|A_{i,c_j}^{(j)}=1,j=1,\ldots,k} \{f_{c_1} \otimes \ldots \otimes f_{c_k} \otimes f_i\} \tag{2}
\]
FIVES

- To possess both feature interpretability and search efficiency, the proposed method FIVES formulates the task of interactive feature generation as searching for edges on the defined feature graph.

3. Differentiable Edge Search

- The task of generating useful interactive features is equivalent to learning an optimal adjacency tensor $A$, so-called edge search.

$$
\min_A \mathcal{L}(\mathcal{D}_{val}|A, \Theta(A))
$$

$$
s. t. \quad \Theta(A) = \arg\min_{\Theta} \mathcal{L}(\mathcal{D}_{train}|A, \Theta)
$$

- To make the optimization more efficient, $A$ is regraded as Bernoulli random variables parameterized by $H \in [0,1]^{K \times m \times m}$, and a soft $A^{(k)}$ is allowed to be used for propagation at the $k$-th layer.
FIVES

To possess both feature interpretability and search efficiency, the proposed method FIVES formulates the task of interactive feature generation as searching for edges on the defined feature graph.

4. Interactive Feature Derivation

- The learned adjacency tensor can explicitly indicate which interactive features are useful.
- One can inductively derive useful high-order interactive features by specify layer-wise thresholds for binarizing the learned $\Lambda$.
- FIVES serves as a feature generator for lightweight models to meet the requirement of inference speed.

An example of interactive feature derivation.

FIVES: Feature Interaction Via Edge Search for Large-Scale Tabular Data. KDD, 2021.
FIVES

- Extensive experiments on five public datasets and two business datasets confirm that FIVES can generate useful interactive features.
  - FIVES as a predictive model for downstream tasks, such as CTR prediction
  - FIVES as the feature generator for lightweight models to meet the requirement of inference speed

Correlation between the entries of $A$ and the AUC of the corresponding indicated feature.

Efficiency comparisons.

Takeaways

**DNN-based methods**
- Implicit feature generation
- One-shot training course
- Lack of interpretable rules for feature interactions

**Search-based methods**
- Explicit feature generation
- Trial-and-error training manner
- Need lots of time and computing resource

✅ Feature Interpretability
✅ Search Efficiency
Future Directions

- How to introduce human experience as prior knowledge for AutoFeature?
- Causal features or spurious correlations?
- How to balance the trade-off between the usefulness of generated features and the completeness of them?
References of AutoFeature


VolcanoML: End-to-End AutoML via Scalable Search Space Decomposition
Two Complications of AutoML going E2E

\[ \alpha^* = \arg \max_{\alpha} f(D', \theta_{\alpha}^*) \]
\[ \text{s.t.}, \theta_{\alpha}^* = \arg \max_{\theta} P(D | \theta) P(\theta | \alpha) \]

**Two Complications**

1. \( \alpha \) is not a homogenous space, it is rather heterogenous

\[ \alpha \in \text{Feature} \times \text{HP} \times \text{Model} \]

2. From single-tenant to multi-tenant scenarios

**Personal perspectives, from our past experiences**

- **VolcanoML**: Speeding up End-to-End AutoML via Scalable Search Space Decomposition. *VLDB* 2021.

- **AutoML from Service Provider’s Perspective**: Multi-device, Multi-tenant Model Selection with GP-EI. *AISTATS* 2019.

Disclaimer

This segment of the tutorial is more opinioned and closer to our own experience than previous segments.

It is less about how much we know about these two problems, but more about discussing some observations and preliminary explorations to show you what we don’t know and a “cry for help”.
Heterogenous Search Space

- **Feature × HP × Model**
- **A strong baseline**: Treat the heterogenous space as a single joint space.
  - Model it with a single Bayesian optimization problem, a single genetic algorithm, or a single hyperband problem
  - Good? Very powerful approach, yet simple.
  - Could be improved?
    - “The curse of dimensionality”: often it is not easy to scale up when the dimensionality of the space is high.
    - Heterogeneity in algorithm: Different subspaces might benefit from different algorithms.

- **Can we do better?**
Heterogenous Search Space

• **Different ways to conduct search. Let’s take for example the space** \( \alpha \in X \times Y \)

• **Strategy 1. Joint**
  - Treating the space \( X \times Y \) as a single search space
  - (If you are doing BO) Create a surrogate model M to approximate \( f(\alpha) \)
  - Use M to select \( \bar{\alpha} \)
  - Evaluate \( f(\bar{\alpha}) \) and update the surrogate model M

• One can implement such a strategy using methods beyond BO.
Heterogenous Search Space

• **Different ways to conduct search. Let’s take for example the space** \( \alpha \in X \times Y \)

• **Strategy 2. Conditioning**
  • Idea: decompose \( X \times Y \) into multiple subspaces, e.g., one for each value of \( X \)
  • \( \min_{x,y} f(x, y) \Rightarrow \min_{x \in X} \min_y g_x(y) \)
  • Then treating each \( x \in X \) as a subproblem \( \min_y g_x(y) \)
  • Can be modeled as a Multi-armed bandit problem – each arm corresponds to a possible value of \( x \in X \), playing an arm means optimizing \( \min_y g_x(y) \) one step

• For example, think about \( X \) as Algorithm and \( Y \) as Feature – For each Algorithm, search for the best feature, and pick the best Algorithm
Heterogenous Search Space

• **Different ways to conduct search. Let’s take for example the space** \( \alpha \in X \times Y \)

• **Strategy 3. Alternating**
  - Idea: decompose \( X \times Y \) into two subspaces, \( X \) and \( Y \)
  - Solve two problems alternatively:
    - \( \min_x g_{\bar{y}}(x) \), where \( \bar{y} \) is the current best value for subspace \( Y \)
    - \( \min_y g_{\bar{x}}(y) \), where \( \bar{x} \) is the current best value for subspace \( X \)
  - Each subproblem can be solved either jointly or via some conditioning strategy
  - At each iteration, pick the subproblem with the largest expected improvement

• For example, think about \( X \) as Feature and \( Y \) as HP – Alternating the process of search for feature and search for HP
Heterogenous Search Space

• **Different ways to conduct search**

• **Strategy 1. Joint**
  • **Pros:** Simple, works well when dimensionality is low
  • **Cons:** Might suffer when the dimensionality is high

• **Strategy 2. Conditioning**
  • **Pros:** Effective when some dimension is categorical variable with small cardinality
  • **Cons:** Might not be applicable to other scenarios.

• **Strategy 3. Alternating**
  • **Pros:** Very effective in reducing dimensions
  • **Cons:** Assuming conditional independence of two subspaces
Heterogenous Search Space

- A single search space can be decomposed in different ways.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Plan 1</th>
<th>Plan 2</th>
<th>Plan 3</th>
<th>Plan 4</th>
<th>Plan 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>puma8NH</td>
<td>0.8275</td>
<td>0.8512</td>
<td>0.8271</td>
<td>0.8280</td>
<td>0.8303</td>
</tr>
<tr>
<td>kin8nm</td>
<td>0.8808</td>
<td>0.8886</td>
<td>0.8886</td>
<td>0.8654</td>
<td>0.8910</td>
</tr>
<tr>
<td>cpu_cmail</td>
<td>0.9122</td>
<td>0.9126</td>
<td>0.9126</td>
<td>0.9027</td>
<td>0.9127</td>
</tr>
<tr>
<td>puma32f1</td>
<td>0.8849</td>
<td>0.8864</td>
<td>0.8848</td>
<td>0.8835</td>
<td>0.8894</td>
</tr>
<tr>
<td>cpu_sct</td>
<td>0.9303</td>
<td>0.9315</td>
<td>0.9305</td>
<td>0.9302</td>
<td>0.9309</td>
</tr>
<tr>
<td>bank32nh</td>
<td>0.7896</td>
<td>0.7889</td>
<td>0.7838</td>
<td>0.7891</td>
<td>0.7957</td>
</tr>
<tr>
<td>mc1</td>
<td>0.8796</td>
<td>0.8904</td>
<td>0.8722</td>
<td>0.8721</td>
<td>0.8975</td>
</tr>
<tr>
<td>delta_elevators</td>
<td>0.8763</td>
<td>0.8760</td>
<td>0.8779</td>
<td>0.8766</td>
<td>0.8790</td>
</tr>
<tr>
<td>jm1</td>
<td>0.6718</td>
<td>0.6721</td>
<td>0.6581</td>
<td>0.6473</td>
<td>0.6692</td>
</tr>
<tr>
<td>pendigits</td>
<td>0.9932</td>
<td>0.9936</td>
<td>0.9929</td>
<td>0.9945</td>
<td>0.9937</td>
</tr>
<tr>
<td>delta_ailerons</td>
<td>0.9235</td>
<td>0.9240</td>
<td>0.9242</td>
<td>0.9225</td>
<td>0.9259</td>
</tr>
<tr>
<td>wind</td>
<td>0.8587</td>
<td>0.8589</td>
<td>0.8566</td>
<td>0.8583</td>
<td>0.8593</td>
</tr>
<tr>
<td>satimage</td>
<td>0.8961</td>
<td>0.8954</td>
<td>0.8955</td>
<td>0.8946</td>
<td>0.8981</td>
</tr>
<tr>
<td>optdigits</td>
<td>0.9889</td>
<td>0.9899</td>
<td>0.9883</td>
<td>0.9889</td>
<td>0.9889</td>
</tr>
<tr>
<td>phoneme</td>
<td>0.8799</td>
<td>0.8832</td>
<td>0.8808</td>
<td>0.8791</td>
<td>0.8866</td>
</tr>
<tr>
<td>spambase</td>
<td>0.9401</td>
<td>0.9406</td>
<td>0.9379</td>
<td>0.9387</td>
<td>0.9386</td>
</tr>
<tr>
<td>abalone</td>
<td>0.6688</td>
<td>0.6679</td>
<td>0.6618</td>
<td>0.6614</td>
<td>0.6680</td>
</tr>
<tr>
<td>mammography</td>
<td>0.8740</td>
<td>0.8783</td>
<td>0.8577</td>
<td>0.8755</td>
<td>0.8787</td>
</tr>
<tr>
<td>waveform</td>
<td>0.8948</td>
<td>0.8961</td>
<td>0.8900</td>
<td>0.8835</td>
<td>0.8952</td>
</tr>
<tr>
<td>pollen</td>
<td>0.4934</td>
<td>0.5013</td>
<td>0.5012</td>
<td>0.5013</td>
<td>0.5013</td>
</tr>
</tbody>
</table>

Average Rank: 3.28, 2.33, 3.80, 3.98, 1.63

Different plans have different performance

Potentially, can learn to decompose given a target workload
Heterogenous Search Space

• **Moving Forward**
  • Build up a suite of different building blocks – what is the unified framework to talk about different search algorithms?
  • How to automatically construct search space decomposition?
  • How to automatically conduct building block selection? AutoML for AutoML?
AutoML: From Single-tenant to Multi-tenant

<table>
<thead>
<tr>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M7</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>0.9</td>
<td>0.2</td>
<td>0.2</td>
<td>0.6</td>
<td>0.5</td>
<td>0.6</td>
</tr>
<tr>
<td>D2</td>
<td>0.6</td>
<td>0.7</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.1</td>
</tr>
<tr>
<td>D3</td>
<td>0.9</td>
<td>0.2</td>
<td>0.2</td>
<td>0.6</td>
<td>0.5</td>
<td>0.6</td>
</tr>
<tr>
<td>D4</td>
<td>0.9</td>
<td>0.2</td>
<td>0.2</td>
<td>0.6</td>
<td>0.5</td>
<td>0.6</td>
</tr>
<tr>
<td>D5</td>
<td>0.6</td>
<td>0.7</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.1</td>
</tr>
<tr>
<td>D6</td>
<td>?</td>
<td>?</td>
<td>0.5</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Existing Models

Existing Datasets

New Dataset

Pool of Resources

Single-tenant Scenario: One target dataset

What if multiple users running their own AutoML workload over a shared infrastructure?

Interesting problem especially when AutoML as a service becomes more and more popular.
AutoML: From Single-tenant to Multi-tenant

<table>
<thead>
<tr>
<th>Existing Datasets</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M7</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>0.9</td>
<td>0.2</td>
<td>0.2</td>
<td>0.6</td>
<td>0.5</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>D2</td>
<td>0.6</td>
<td>0.7</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.1</td>
<td>0.7</td>
</tr>
<tr>
<td>D3</td>
<td>0.9</td>
<td>0.2</td>
<td>0.2</td>
<td>0.6</td>
<td>0.5</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>D4</td>
<td>0.9</td>
<td>0.2</td>
<td>0.2</td>
<td>0.6</td>
<td>0.5</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>D5</td>
<td>0.6</td>
<td>0.7</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.1</td>
<td>0.7</td>
</tr>
<tr>
<td>D6</td>
<td>?</td>
<td>?</td>
<td>0.5</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

How to balance resource allocations to different users?
AutoML: From Single-tenant to Multi-tenant

- **Regret: A Single User’s Unhappiness**

<table>
<thead>
<tr>
<th>Decisions</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0.5</td>
</tr>
<tr>
<td>M2</td>
<td>0.7</td>
</tr>
<tr>
<td>M3</td>
<td>0.76</td>
</tr>
<tr>
<td>M4</td>
<td>0.79</td>
</tr>
<tr>
<td>M5</td>
<td>0.85</td>
</tr>
<tr>
<td>M6</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Regret after $T$ trials: $R_T$

(Regret: We could have serve the user a better model if we magically knows the best model to try)
AutoML: From Single-tenant to Multi-tenant

Which user should we serve next?
AutoML: From Single-tenant to Multi-tenant

User 1: [0.99] [0.99]
User 2: [0.10] [0.35]

Extreme Case: User 1 is not worth serving any more

How about more general case?
AutoML: From Single-tenant to Multi-tenant

01 Each user runs their own GP-EI model selection

02 Serve the user with highest expected improvement.

Informal Theorem. If the performance of all models is a linear combination of a finite, shared set of hidden Gaussian variables, the global regret converges to 0 with rate $O(1 / \text{runtime})$. 
## AutoML: From Single-tenant to Multi-tenant

### Machine Learning Models

<table>
<thead>
<tr>
<th>Datasets (Users)</th>
<th>Existing Datasets</th>
<th>New Datasets</th>
<th>Computation Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>0.9 0.2 0.2 0.6 0.5 0.6 0.2</td>
<td>? ? ? ?</td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>0.6 0.7 0.2 0.4 0.6 0.1 0.7</td>
<td>? ? ? ?</td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td>0.9 0.2 0.2 0.6 0.5 0.6 0.2</td>
<td>? 0.2 ?</td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td>0.9 0.2 0.2 0.6 0.5 0.6 0.2</td>
<td>? ? ? ?</td>
<td></td>
</tr>
<tr>
<td>D5</td>
<td>0.6 0.7 0.2 0.4 0.6 0.1 0.7</td>
<td>? ? ? ?</td>
<td></td>
</tr>
</tbody>
</table>

### Existing Models

- M1
- M2
- M3
- M4
- M5
- M6
- M7
- M8
- ...
- Mk

### New Models

Each user runs their own GP-UCB algorithm

Serve the user with a factor that is very similar to expected improvement (directly comparing each user’s UCB does not work, for obvious reason)

In this case, the total regret is bounded (up to some constant) by

$$
\sqrt{n^{3/2}} \beta^* T \sum_{i=1}^{n} \log(|T(i)|) \leq n^{3/2} \sqrt{\beta^* T \log \left( \frac{T}{n} \right)}.
$$

the regret for RR, see (1)
AutoML: From Single-tenant to Multi-tenant

Modeling error dominates
AutoML: From Single-tenant to Multi-tenant

<table>
<thead>
<tr>
<th>Machine Learning Models</th>
<th>New Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>M1</td>
</tr>
<tr>
<td>0.9</td>
<td>0.2</td>
</tr>
<tr>
<td>D2</td>
<td>M1</td>
</tr>
<tr>
<td>0.6</td>
<td>0.7</td>
</tr>
<tr>
<td>D3</td>
<td>M1</td>
</tr>
<tr>
<td>0.9</td>
<td>0.2</td>
</tr>
<tr>
<td>D4</td>
<td>M1</td>
</tr>
<tr>
<td>0.9</td>
<td>0.2</td>
</tr>
<tr>
<td>D5</td>
<td>M1</td>
</tr>
<tr>
<td>0.6</td>
<td>0.7</td>
</tr>
<tr>
<td>D6</td>
<td>M1</td>
</tr>
<tr>
<td>...</td>
<td>M1</td>
</tr>
<tr>
<td>D_n</td>
<td>M1</td>
</tr>
</tbody>
</table>

Need some special care on the diversity: don’t put all GPUs on a single user.

**Theorem.** Near linear speed up with respect to the number of devices when # devices << # users.
AutoML: From Single-tenant to Multi-tenant

• **Moving Forward**
  • In my opinion, it is exciting future direction to try to understand resource allocation and scheduling for AutoML workloads
  • What’s the unified way to talk about and think about different AutoML workloads, e.g., those we have been talking about over the last two hours
  • Fairness? Efficiency? How should we aggregate unhappiness from multiple users?
Two Complications of AutoML going E2E

\[ \alpha^* = \arg \max_{\alpha} f(D', \theta_{\alpha}^*) \]
\[ \text{s.t, } \theta_{\alpha}^* = \arg \max_{\theta} P(D | \theta) P(\theta | \alpha) \]

Two Complications

1. \( \alpha \) is not a homogenous space, it is rather heterogeneous

   \[ \alpha \in \text{Feature} \times \text{HP} \times \text{Model} \]

   (auto-sklearn)

A lot of challenges and exciting opportunities when bring AutoML to end-to-end production scenario!

2. From single-tenant to multi-tenant scenarios
AutoML: A Small Personal Remark

**ML today is now a Data Problem**

- For many tasks, given the raw features from Kaggle, most AutoML platforms rank in the bottom 50%.
- It is the data that we need to improve, and knowledge that we need to integrate, to build better ML applications.
- To improve data, we need to first understand them.

*Moving from a Model-driven development to a Data-driven development.*

MLBench

VLDB (2018)

ML-Guided Database
Where DB Meets ML

• Human involved in research/engineering/analyzing/administrating:
  • Building and maintaining indexes
  • Query optimization
  • Physical design tuning
  • Optimizing view materialization

• Learning to automatically designing/optimizing/tuning?
Where DB Meets ML: Learning to Index

- Human involved in research/engineering/analyzing/administrating:
  - Building and maintaining indexes
    - Query optimization
    - Physical design tuning
    - Optimizing view materialization

- Learning to automatically designing/optimizing/tuning?
B-Tree Index from Learning Perspective

Input: Key
Output: Position
B-Tree Index: position = B-tree(Key)

Input: Key
Output: Position
Learned Index: position = function(Key)

[Image source] Kraska et al., The case for learned index structures. SIGMOD, 2018
Why Learning Index from Data?

• Consider this (ideal) case: build an index to store and query over a table of $n$ rows with continuous integer keys, i.e., Keys = [11, 12, 13, 14, 15, ...] and Pos = [0, 1, 2, 3, 4, ...]
  • B-Tree: seeking Pos in time $O(\log n)$
  • a learned function $\text{Pos} = M(\text{Key}) = \text{Key} + \text{offset}$ : $O(1)$

• Main motivation: the hidden yet useful distribution information about the data to be indexed has not been fully explored and utilized in the classic index techniques
  • learned index: an automatic way to explore and utilize such information
Recursive-Model Index (RMI)

[Image source] Kraska et al., The case for learned index structures. SIGMOD, 2018
FITing-Tree

Error-Bounded Linear Segment: Given threshold $\text{error}$, a segment from $(x_1, y_1)$ to $(x_3, y_3)$ is not valid if $(x_2, y_2)$ is further than $\text{error}$ from the interpolated line.

ShrinkingCone (building a segment): Point 1 is the origin of the cone. Point 2 is then added, resulting in the dashed cone. Point 3 is added next, yielding in the dotted cone. Point 4 is outside the dotted cone and therefore starts a new segment.

[Image source] Galakatos et al., FITing-Tree: A Data-aware Index Structure. SIGMOD, 2019
RMI v.s. FITing-Tree

Root Model
\[ y = ax + b \]

Sub-model Organization

Sub-models

B-Tree

RMI

FITing-Tree
More Learned Index Methods

• PGM [1] improves FITing-Tree by finding the optimal number of learned segments given an error bound.

• ALEX [2] proposes an adaptive RMI with workload-specific optimization, achieving high performance on dynamic workloads.


• Multi-dimensional indexes: NEIST [4], Flood [5], Tsunami [6] and LISA [7].
More Learned Index Methods


Questions about Learned Indexes

How to systematically analyze and design machine learning based indexing methods?

More scalable index learning methods?

Which class of models suffice?
Task Definition

• Given a database $\mathcal{D}$ with $n$ records (rows), let’s assume that a range index structure will be built on a specific column $x$. For each record $i \in [n]$, the value of this column, $x_i$, is adopted as the key, and $y_i$ is the position where the record is stored.

• We want to learn a mechanism $M$ with the key $x$ as input and outputs a predicated position $\hat{y} \leftarrow M(x)$ for accessing data.
Learning Index: A Machine Learning Task

$L(M)$ measures the cost of calculating $\hat{y} \leftarrow M(x)$

$L(D|M) = \mathbb{E}_{(x,y) \in D} L(y, \hat{y})$

(a) B-Tree Index

Key

BTree

pos

pos - 0

pos + pagezise

(b) Learned Index

Key

Model (e.g., NN)

pos

pos - min_err

pos + max_err

Learning Index: A Machine Learning Task

\[ M^* = \arg \min_{M \in \mathcal{M}} \text{MDL}(M, \mathcal{D}) \]

\[ = \arg \min_{M \in \mathcal{M}} (L(M) + \alpha L(\mathcal{D}|M)) \]

\[ = \arg \min_{M \in \mathcal{M}} (L(M) + \alpha \mathbb{E}_{\{x,y\} \in \mathcal{D}} L(y, \hat{y})) \]

- Regularization
- Training loss
- Trade-off

Objective function
Benefits of Learned Index

- Smaller Size
- Faster Index Seek
- Better Handling Index Update
  - Generalization ability of machine learning
  - Incremental learning
- Question Mark
  - Is model training/inference scalable enough?

How to systematically analyze and design machine learning based indexing methods?
More scalable index learning methods?
Which class of models suffice?
Learned Index with Sampling

- How large the sample needs to be?
  - $n$ is the data size
  - $M^*$ is fully optimized

**Theorem 1.** Consider the optimization problem:

$$M^* = \arg\min_{M \in M} \text{MDL}(M, \mathcal{D}) = \arg\min_{M \in M} (L(M) + \alpha L(\mathcal{D}|M)) .$$

We can solve it on a random sample $\mathcal{D}_s$ with size $s = O(\alpha^2 \log^2 n)$ as

$$\hat{M}^* = \arg\min_{M \in M} \text{MDL}(M, \mathcal{D}_s)$$

s.t., $\text{MDL}(\hat{M}^*, \mathcal{D}) \leq \text{MDL}(M^*, \mathcal{D}) + O(1)$ with high probability.

Fig: Illustration of sampling

Learned Index with Sampling

- Up to 78x building speedup
- Non-degraded performance in terms of query time and prediction error)

Fig: Illustration of sampling

Is Linear Model Sufficient?

- Linearization of a learned model

\[ \bar{y} = M(x) \]
Is Linear Model Sufficient?

• Linearization of a learned model

\[
\bar{y} = M(x) + \sum (x_l, y_l), (x_r, y_r), ...
\]
Is Linear Model Sufficient?

- Linearization of a learned model

\[
\hat{y} = M_L(x)
\]

connecting \((x_l, \bar{y}_l = M(x_l))\) to \((x_r, \bar{y}_r = M(x_r))\)

\[
\bar{y} = M(x) + \ldots, (x_l, y_l), (x_r, y_r), \ldots
\]
Is Linear Model Sufficient?

• Linearization of a learned model

A learned model

\[ \tilde{y} = M(x) \]

Landmark points

... \((x_l, y_l), (x_r, y_r), \ldots\)

Linearized model

connecting \((x_l, \tilde{y}_l = M(x_l))\) to \((x_r, \tilde{y}_r = M(x_r))\)

\[ \hat{y} = M_L(x) \]

**Theorem 2.** Suppose \(\forall x, |\tilde{y} - y| \leq \varepsilon\), after linearization, we have \(\forall x, |\hat{y} - y| \leq 3\varepsilon + 2(y_r - y_l)\).
Sampling-Restriction-Linearization

Learned Index on Sampled Data → Restriction → Linearization

Sampled data points as landmark points: ..., \((x_l, y_l), (x_r, y_r), ...

Open Questions

• How to handle extremely outlier keys?

• How to maintain index on updating data? [2]

• How to handle multi-dim data? [5, 6, 7]

• How to build it into real DB systems?
  • without too much modification to the current system
AutoML Tools
Availability

*AdaBERT: Task-Adaptive BERT Compression with D-NAS, IJCAI 2020*

*FIVES: Feature Interaction Via Edge Search for Large-Scale Tabular Data, KDD 2021.*

*Automated Relational Meta-learning, ICLR 2020.*

---

Availability

Hyperparameter Optimization

Publicly available at Alibaba Platform of A.I., AutoML product

Feature Generation

AutoML

Compressed Model Search

Publicly available at Alibaba Platform of A.I., EasyTransfer product

Meta-Learning
# A Summary of AutoML Tools

<table>
<thead>
<tr>
<th>Name</th>
<th>Authors</th>
<th>Functionalities</th>
<th>Algorithms</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Tune Models (ATM)</td>
<td>MIT</td>
<td>AutoFeature, Model Selection, HPO</td>
<td>BO and Bandit</td>
<td>Python</td>
</tr>
<tr>
<td>AutoKeras</td>
<td>Texas A&amp;M University</td>
<td>NAS</td>
<td>BO</td>
<td>Python</td>
</tr>
<tr>
<td>NNI</td>
<td>Microsoft</td>
<td>AutoFeature, HPO, NAS, Model Selection</td>
<td>Comprehensive</td>
<td>Python</td>
</tr>
<tr>
<td>emukit</td>
<td>Amazon</td>
<td>HPO</td>
<td>Meta-surrogate model</td>
<td>Python</td>
</tr>
<tr>
<td>Ray Tune</td>
<td>Berkeley</td>
<td>HPO</td>
<td>Comprehensive</td>
<td>Python</td>
</tr>
<tr>
<td>TPOT</td>
<td>University of Pennsylvania</td>
<td>AutoFeature, Model Selection, HPO</td>
<td>Genetic programming</td>
<td>Python</td>
</tr>
</tbody>
</table>

More AutoML packages include AutoFolio, Auto-sklearn, Auto-PyTorch, Auto-WEKA, etc.
Tutorial Schedule

Yaliang Li, Background and Overview of AutoML Hyperparameter Optimization

Zhen Wang, Neural Architecture Search Meta-Learning

Yuexiang Xie, Automatic Feature Generation

Ce Zhang, VolcanoML: End-to-End AutoML via Scalable Search Space Decomposition

Bolin Ding, Machine Learning Guided Database
Thank you!

Yaliang Li, Zhen Wang, Yuexiang Xie, Bolin Ding, and Ce Zhang

Email: yaliang.li@alibaba-inc.com
Please feel free to contact us if you have any questions, or you are interested in full-time or research intern positions.