Katib: A Distributed General AutoML Platform on Kubernetes

Jinan Zhou, Andrey Velichkevich, Kirill Prosvirov, and Anubhav Garg, Cisco Systems; Yuji Oshima, NTT Software Innovation Center; Debo Dutta, Cisco Systems

https://www.usenix.org/conference/opml19/presentation/zhou

This paper is included in the Proceedings of the 2019 USENIX Conference on Operational Machine Learning (OpML ’19).

May 20, 2019 • Santa Clara, CA, USA

ISBN 978-1-939133-00-7

Open access to the Proceedings of the 2019 USENIX Conference on Operational Machine Learning is sponsored by USENIX.
Katib: A Distributed General AutoML Platform on Kubernetes

Jinan Zhou Andrey Velichkevich Kirill Prosivirov Anubhav Garg
Cisco Systems, USA
{jinazhou, avelichk, kprosvir, anubhgar}@cisco.com

Yuji Oshima
NTT Software Innovation Center, Japan
yuuji.ooshima.fn@hco.ntt.co.jp

Debo Dutta
Cisco Systems, USA
dedutta@cisco.com

Abstract

Automatic Machine Learning (AutoML) is a powerful mechanism to design and tune models. We present Katib, a scalable Kubernetes-native general AutoML platform that can support a range of AutoML algorithms including both hyper-parameter tuning and neural architecture search. The system is divided into separate components, encapsulated as micro-services. Each micro-service operates within a Kubernetes pod and communicates with others via well-defined APIs, thus allowing flexible management and scalable deployment at a minimal cost. Together with a powerful user interface, Katib provides a universal platform for researchers as well as enterprises to try, compare and deploy their AutoML algorithms, on any Kubernetes platform.

1 Introduction

Automatic Machine Learning (AutoML) determines the optimal hyper-parameters or the neural network structure for a specific task. Thus it enables less technical users, and can discover state-of-art models that are almost as good as handcrafted ones ([21], [14], [16], [4], [10]). However, we have a long way before AutoML becomes mainstream. The first is the diversity of AutoML algorithms. Algorithms for hyper-parameter tuning are generally different from those for neural architecture search (NAS). Even within NAS, different algorithms follow separate structural mechanisms. This diversity makes it difficult to reuse infrastructure and code, thus increasing the cost of deploying AutoML widely. The second problem is the prohibitive computational cost. The algorithm may search for either the optimal network or the optimal hyper-parameters or the neural network structure for a specific task. Thus it enables less technical users, and can discover state-of-art models that are almost as good as handcrafted ones ([21], [14], [16], [4], [10]). However, we have a long way before AutoML becomes mainstream. The first is the diversity of AutoML algorithms. Algorithms for hyper-parameter tuning are generally different from those for neural architecture search (NAS). Even within NAS, different algorithms follow separate structural mechanisms. This diversity makes it difficult to reuse infrastructure and code, thus increasing the cost of deploying AutoML widely. The second problem is the prohibitive computational cost. The algorithm may search for either the optimal network or the optimal hyper-parameters or the neural network structure for a specific task. Thus it enables less technical users, and can discover state-of-art models that are almost as good as handcrafted ones ([21], [14], [16], [4], [10]). However, we have a long way before AutoML becomes mainstream. The first is the diversity of AutoML algorithms. Algorithms for hyper-parameter tuning are generally different from those for neural architecture search (NAS). Even within NAS, different algorithms follow separate structural mechanisms. This diversity makes it difficult to reuse infrastructure and code, thus increasing the cost of deploying AutoML widely. The second problem is the prohibitive computational cost. The algorithm may search for either the optimal network or the optimal hyper-parameters or the neural network structure for a specific task. Thus it enables less technical users, and can discover state-of-art models that are almost as good as handcrafted ones ([21], [14], [16], [4], [10]). However, we have a long way before AutoML becomes mainstream. The first is the diversity of AutoML algorithms. Algorithms for hyper-parameter tuning are generally different from those for neural architecture search (NAS). Even within NAS, different algorithms follow separate structural mechanisms. This diversity makes it difficult to reuse infrastructure and code, thus increasing the cost of deploying AutoML widely. The second problem is the prohibitive computational cost. The algorithm may search for either the optimal network or the optimal hyper-parameters or the neural network structure for a specific task. Thus it enables less technical users, and can discover state-of-art models that are almost as good as handcrafted ones ([21], [14], [16], [4], [10]). However, we have a long way before AutoML becomes mainstream. The first is the diversity of AutoML algorithms. Algorithms for hyper-parameter tuning are generally different from those for neural architecture search (NAS). Even within NAS, different algorithms follow separate structural mechanisms. This diversity makes it difficult to reuse infrastructure and code, thus increasing the cost of deploying AutoML widely. The second problem is the prohibitive computational cost. The algorithm may search for either the optimal network or the optimal hyper-parameters or the neural network structure for a specific task. Thus it enables less technical users, and can discover state-of-art models that are almost as good as handcrafted ones ([21], [14], [16], [4], [10]). However, we have a long way before AutoML becomes mainstream. The first is the diversity of AutoML algorithms. Algorithms for hyper-parameter tuning are generally different from those for neural architecture search (NAS). Even within NAS, different algorithms follow separate structural mechanisms. This diversity makes it difficult to reuse infrastructure and code, thus increasing the cost of deploying AutoML wide...
by duplicating the topology of $G'$. In terms of evolving strategy, some NAS algorithms adopt a generation approach while others use modification. With the modification approach, the algorithm will propose a new neural architecture in each iteration. With the modification approach, however, the algorithm will modify the current architecture by adding or deleting some parts of it instead of creating a brand-new candidate. Based on this categorization, the latest NAS algorithms can be summarized as follows:

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Search for Network</th>
<th>Search for Cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evolve by</td>
<td>[23], [16], [20], [11], [1], [2]</td>
<td>[23], [16], [24], [22], [13]</td>
</tr>
<tr>
<td>Modification</td>
<td>[9], [4], [18], [7], [10], [15], [6], [5]</td>
<td>[17], [21], [14], [10], [15], [6], [5]</td>
</tr>
</tbody>
</table>

Table 1: Summary of neural architecture search algorithms

Diverse as they are, those algorithms can be integrated into one system. Compared with hyperparameter tuning, NAS only needs one extra ModelManager service to store, construct and manipulate models. In each iteration, Suggestion provides the topology of the next candidate or the modification decisions of the previous architecture to ModelManager, which constructs the model and sends it to Trial. Then the model is evaluated and the performance metrics are fed back to Suggestion, starting a new iteration.

All these workflows can be summarized by Figure 1:

Figure 1: Summary of AutoML workflows

3 Katib System Design

We combine the requirements of these AutoML workflows and the ideas from Google’s black-box optimization tool Vizier [8], with the design shown in Figure 2. The user starts from defining an AutoML task with Katib’s interface, the details of which can be found at http://bit.ly/2E5B9pV. A controller examines the task definition and spawns the necessary services. The data communication between different containers is managed by Vizier Core. The searching procedure follows exactly the workflow defined in Section 2.

Consider EnvelopeNet [10] as an example of non-standard NAS algorithm. In EnvelopeNet, the neural networks are updated by pruning and expanding EnvelopeCells, which are convolution blocks connected in parallel. And these modification decisions are based on feature statistics instead of validation accuracy. The Suggestion and training containers will be pre-built so the user only needs to specify the structure of EnvelopeCells and other necessary parameters in StudyJob yaml file. In each iteration, Vizier Core

Manager first requests one or more modification decisions from Suggestion and sends them to ModelManager. Then ModelManager calculates the current architectures, compiles the models into runnable objects, and sends them to Trials, which will carry out a truncated training process. Once finished, a MetricsCollector is spawned to parse feature statistics from the training logs. Finally, this information is fed back to Suggestion via the Vizier Core and a new iteration starts. During the process, all the model topologies and metrics are stored in a database and presented to the user. Katib is scalable. The controller can spawn multiple parallel Trials in each iteration to accelerate the search. These service components can be shared globally among all the users.

The initial version provides hyper-parameter tuning with Bayesian optimization [19], Hyperband [12], grid search and neural architecture search with reinforcement learning ([23]). The user can also deploy customized tasks by creating her own algorithm for the Suggestion and the training container for each Trial. We will add more algorithms such as EnvelopeNet [10] and integrate the support for advanced acceleration techniques such as parameter sharing [16].

4 Conclusions

This paper presents Katib, a distributed general AutoML system based on Kubernetes. The key idea is to abstract AutoML algorithms into functionally isolated components and containerize each component as a micro-service. With this extendable and scalable design, Katib can be a powerful tool for both advancing machine learning research and delivering turnkey AI solutions for enterprise users.

Acknowledgement

The authors would like to thank Cisco Systems for their support, especially Amit Saha for help with the paper.

References


