Taking the Human out of Learning Applications: A Survey on Automated Machine Learning

Quanming Yao, Mengshuo Wang,
Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, Yang Yu

Abstract—Machine learning techniques have deeply rooted in our everyday life. However, since it is knowledge- and labor-intensive to pursue good learning performance, humans are heavily involved in every aspect of machine learning. To make machine learning techniques easier to apply and reduce the demand for experienced human experts, automated machine learning (AutoML) has emerged as a hot topic with both industrial and academic interest. In this paper, we provide an up to date survey on AutoML. First, we introduce and define the AutoML problem, with inspiration from both realms of automation and machine learning. Then, we propose a general AutoML framework that not only covers most existing approaches to date, but also can guide the design for new methods. Subsequently, we categorize and review the existing works from two aspects, i.e., the problem setup and the employed techniques. The proposed framework and taxonomies provide a detailed analysis of AutoML approaches and explain the reasons underneath their successful applications. We hope this survey can serve as not only an insightful guideline for AutoML beginners but also an inspiration for future research.

Index Terms—automated machine learning, neural architecture search, hyper-parameter optimization

1 Introduction

Mitchell's famous machine learning textbook [1] begins with the statement: "Ever since computers were invented, we have wondered whether they might be made to learn. If we could understand how to program them to learn - to improve automatically with experience - the impact would be dramatic". This quest gave birth to a new research area, i.e., machine learning, for Computer Science decades ago. Till now, machine learning techniques have been deeply rooted in our every day's life, such as recommendation when we are reading news and handwriting recognition when we are using our cell-phones. Furthermore, machine learning has also gained significant achievements. For example, AlphaGO [2] defeated human champion in the game of GO, ResNet [3] surpassed human performance in image recognition, Microsoft's speech system [4] approximated human level in speech transcription.

However, these successful applications of machine learning are far from fully automated, i.e., "improving automatically with experience ". Since there are no algorithms that can achieve good performance on all possible learning problems with equal importance (according to No Free Lunch theorems [5] [6]), every aspect of machine learning applications, such as feature engineering, model selection, and algorithm selection (Figure 1), needs to be carefully configured. Human experts are hence heavily involved in machine learning applications. As these experts are rare, the success of machine learning comes at a great price.

- *Q. Yao, M. Wang, Y. Chen and W. Dai are with 4Paradigm Inc, Beijing, China; Y. Li and Y. Yu are with Nanjing uiversity, Jiangsu, China; and Q. Yang is with Hong Kong University of Science and Technology, Hong Kong, China.*
- *All authors are in alphabetical order of last name (except the first two). Correspondance to Q. Yao at yaoquanming@4paradigm.com*

Thus, automated machine learning (AutoML) does not just remain an academic dream as described in Michell's book, but also attracts more attention from practitioners. If we can take the human out of these machine learning applications, we can enable faster deployment of machine learning solutions across organizations, efficiently validate and benchmark the performance of deployed solutions, and make experts focus more on problems with more application and business values. These would make machine learning much more accessible for real-world usages, leading to new levels of competence and customization, of which the impact can be indeed dramatic.

Motivated by the above academic dream and practical needs, in recent years, AutoML has emerged as a new sub-area in machine learning. It has got more attention not only in machine learning but also in computer vision, data mining and natural language processing. Up to now, AutoML has already been successfully applied in many important problems (Table 1).

TABLE 1
Examples of AutoML approaches in industry and academic.

application	industry	academic
automated model selection	Auto-sklearn	[7], [8]
neural architecture search	Google's Cloud	[9], [10]
automated feature engineering	Feature Labs	[11], [12]

The first example is Auto-sklearn [7]. As different classifiers are applicable to different learning problems [1], [13], it is natural to try a collection of classifiers on a new problem, and then construct a final prediction from them. However, setting up classifiers and their hyper-parameters is a tedious task, which usually requires human involvement. Based on the popular scikit-learn machine learning library [14], Auto-sklearn can automatically find good models from some

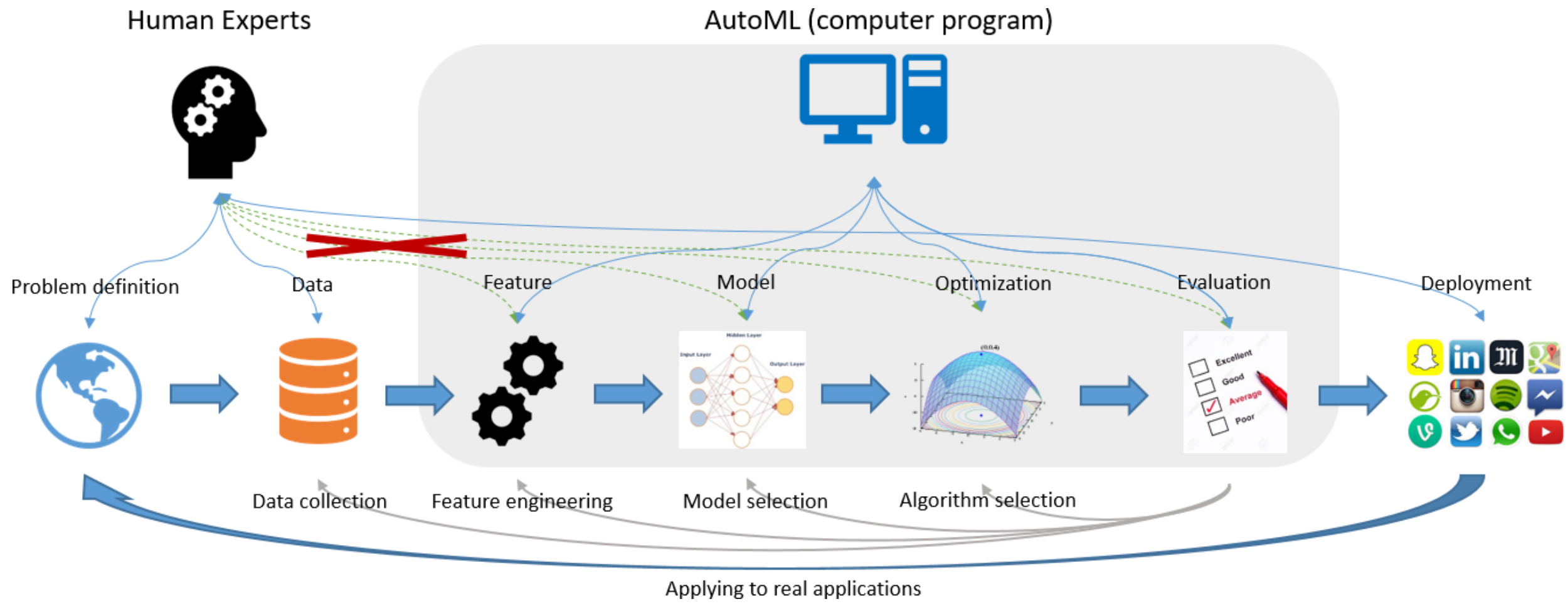

Fig. 1. To use machine learning techniques and obtain good performance, humans usually need to be involved in data collection, feature engineering, model and algorithm selection. This picture shows a typical pipeline of machine learning application, and how AutoML can get involved in the pipeline and minimize participation of humans.

out-of-the-box machine learning tools for classification by searching for proper models and optimizing their corresponding hyper-parameters.

The second example is the neural architecture search (NAS) [9], [15], [16]. Since the success of AlexNet [17] on image classification of ImageNet data set [18], architecture design has become the main source of performance improvement in the realm of deep learning. Examples are VGGNet [19], GoogleNet [20], ResNet [3] and DenseNet [21]. Hence, for the tasks in hand, automated design of neural architectures is of great importance to good learning performance. Many researchers have been working on NAS, e.g., [9], [10], [16], [22], [23], [24], [25], [26]. Besides, NAS has been used in Google's Cloud AutoML, which frees customers from the difficult and time-consuming architecture design process.

The last example is automated feature engineering. In traditional machine learning methods, the modeling performance depends greatly on the quality of features [1]. Hence, most machine learning applications take feature engineering as a vital preposition step, where useful features are generated or selected. Such operations, in the past, are usually carried out manually by human experts with in-depth domain knowledge in a trial-and-error manner. Automated feature engineering [11], [12] aims to construct a new features set, with which the performance of subsequent machine learning tools can be improved. By this means, intensive human knowledge and labor can be spared. Existing works on this topic include Data Science Machine (DSM) [12], ExploreKit [11] and FeatureHub [27]. Besides, we have also seen commercial products such as FeatureLabs [12].

With such a rapid development of AutoML in both research and industry, we feel it necessary to summarize existing works and conduct a survey on this topic at this moment. First, we discuss what the AutoML problem is. Then, we propose a general framework that summarizes how existing approaches work towards AutoML. Such a framework further motivates us to give taxonomies of existing works based on what (by problem setup) and how (by techniques) to automate. Specifically, problem setup helps us to clarify what learning process we want to use, while techniques give us the technical methods and details to address the AutoML problem under the corresponding setup. Based on these taxonomies, we further give a guidance on how AutoML approaches can be developed.

1.1 Contributions

Below, we summarize our contributions in this survey:

- We discuss the formal definition of AutoML. The definition is not only general enough to include all existing AutoML problems, but also specific enough to clarify what is the goal of AutoML. Such definition is helpful for setting future research target in the AutoML area.
- We propose a general framework for existing AutoML approaches. This framework is not only helpful for setting up taxonomies of existing works, but also gives insights of the problems existing approaches want to solve. Such framework can act as a guidance for developing new approaches.
- We systematically categorize existing AutoML works based on "what to automate" and "how to automate". Problem setups are from the "what" perspective, indicating which learning process we want to make automated. Techniques are from the "how" perspective, introducing the methods proposed to solve AutoML problems. For each category, we present detailed application scenarios for reference.
- Compared to existing AutoML related surveys[1], we provide a detailed analysis of existing techniques, which is based on the proposed framework. We not only investigate a more comprehensive set of existing works, but also present a summary of the insights behind each technique. This can serve as an good guideline not only for beginners' usage but also for future researches.

1. In this survey we focus on the usage of existing techniques in AutoML, for individual reviews on related topics please refer to [28], [29], [30] for meta-learning, [31] for transfer learning, [32] for hyper-parameter optimization and [33] for neural architecture search.

- We suggest four promising future research directions in the field of AutoML in terms of the problem setting, techniques, applications and theory. For each, we provide a thorough analysis of its disadvantages in the current work and propose future research directions.

1.2 Organization

The survey is organized as follows. The overview is in Section 2, which gives the definition of AutoML, the proposed framework of AutoML approaches, and taxonomies by problem setup and techniques of existing works. Section 3 describes the taxonomy by problem setup, and techniques are detailed in Section 4-5. Three application examples listed in Table 1 are detailed in Section 6. The survey is summarized in Section 7 with a brief history, the current status, and discussion on future works. Finally, we conclude the survey in Section 8.

1.3 Notation

In the rest of this survey, we denote a machine learning tool as $F(\mathbf{x}; \theta)$, where $\mathbf{x}$ is the model parameters learned by training and θ contains configurations of the learning tool. Besides, the most important concepts used in the survey are explained as follows.

- A **learning process** is a part or the whole of a machine learning pipeline. Examples of learning processes are feature engineering, model and/or algorithm selection, and neural architecture design.
- A **learning tool** is a method which can solve some problems appear in machine learning. For example, a support vector machine (SVM) model is a learning tool, which can solve specific classification problems; and sparse coding [34] is also a learning tool, which can address feature learning problem for certain types of data.
- We use the term **configuration** to denote all factors but the model parameters $\mathbf{x}$ (which are usually obtained from model training) that influence the performance of a learning tool. Examples of configurations are, the hypothesis class of a model, the features utilized by the model, hyper-parameters that control the training procedure, and the architecture of a neural network.

2 Overview

In Section 1, we have shown why we need to do AutoML. In this section, we first define what AutoML problem is in Section 2.1. Then, in Section 2.2 we propose a framework of how AutoML problems can be solved in general. Finally, taxonomies of existing works based on "what to automate" and "how to automate" are presented in Section 2.3.

2.1 Problem Definition

Inspired by automation and machine learning, here, we define what the AutoML problem is. Based on the definition, we also explain core goals of AutoML.

2.1.1 AutoML as an Interaction of Two Fields

From its name, we can see that AutoML is naturally the intersection of automation and machine learning. While automation has a long history, which can even date back to BC [35], machine learning was only invented decades ago [1]. The combination of these two areas has just become a hot research topic in recent years. The key ideas from these two fields and their impacts on AutoML are as follows, and they are the main motivations for our AutoML's definition in the sequel.

- Machine learning, as Definition 1, is a kind of computer program specified by E, T and P.

  Definition 1 (Machine learning [1]). *A computer program is said to learn from experience E with respect to some classes of task T and performance measure P if its performance can improve with E on T measured by P.*

  From this definition, we can see that AutoML itself is also a computer program that has good generalization performance (i.e., P) on the input data (i.e., E) and given tasks (i.e., T). However, traditional machine learning researches focus more on inventing and analyzing learning tools, it does not care much about how easy can these tools be configured. One such example is exactly the recent trend from simple to deep models, which can offer much better performance but also much hard to be configured [36]. In the contrast, AutoML emphasizes on how easy learning tools can be used and controlled. This idea is illustrated in Figure 2.

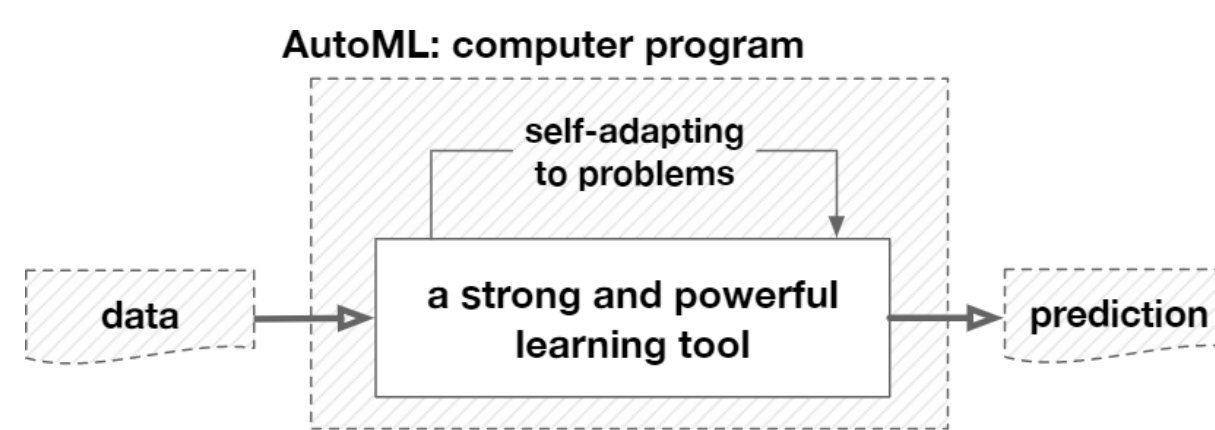

Fig. 2. AutoML from machine learning's perspectives.

- On the other hand, automation is to use various control methods operating underneath components [37]. In pursuit of better predicting performance, configurations of machine learning tools should be adapted to the task with input data, which is often carried out manually. As shown in Figure 3, the goal of AutoML from this perspective is to construct high-level controlling approaches over underneath learning tools so that proper configurations can be found by computer programs.

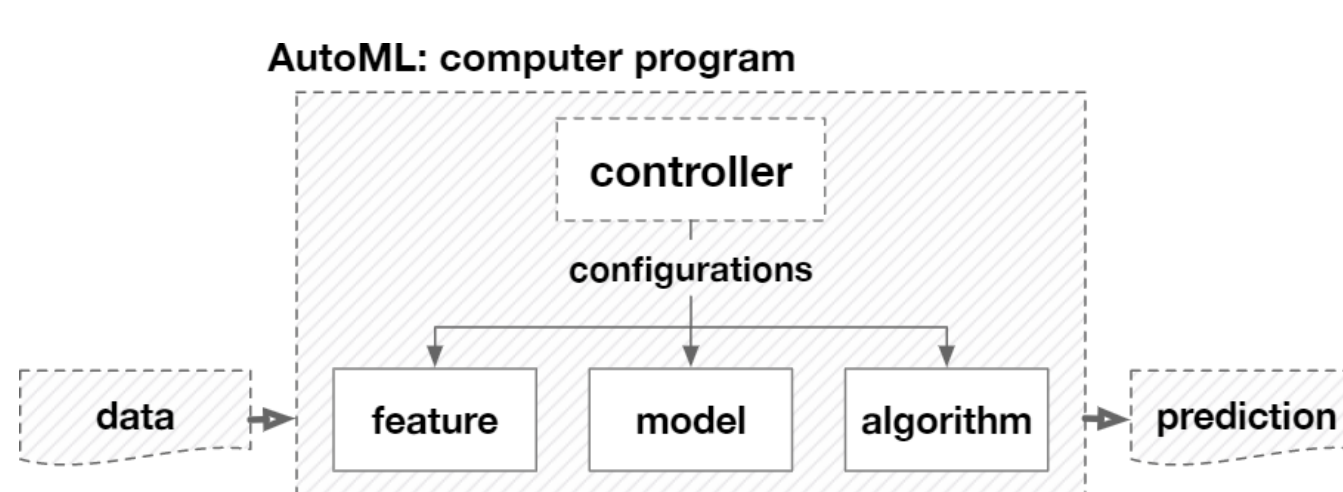

Fig. 3. AutoML from automation's perspectives.

TABLE 2
Why we need to have AutoML: an overview comparison of classical machine learning and AutoML.

		classical machine learning	AutoML
feature engineering		humans design and construct features from data	automated by the computer program
		humans process features making them more informative	
model selection		humans design or pick up some machine learning tools based on professional knowledge	
		humans adjust hyper-parameters of machine learning tools based on performance evaluation	
algorithm selection		humans pick up some optimization algorithms to find parameters	
summary		human are involved in every aspect of learning applications	the program can be directly reused on other learning problems

2.1.2 The Definition of AutoML

From Section 2.1.1, we can see that AutoML not only wants to have good learning performance (from machine learning's perspective) but also require such performance being achieved with less human assistance (from automation's perspective). Thus, an informal and intuitive description of AutoML can be expressed as

$$\max_{\text{configurations}} \text{ performance of learning tools,} \tag{1}$$

$$\text{s.t.} \begin{cases} \text{limited (or no) human assistance} \\ \text{limited computational budget} \end{cases}.$$

Put it more formally, we describe what AutoML is in Definition 2. Such definition is inspired by Definition 1 and the fact that AutoML itself can also be seen as another machine learning approach (Figure 2).

Definition 2 (AutoML)**.** *AutoML attempts to take the place of humans on identifying (all or a part of) configurations, which are proper to machine learning computer programs (specified by E, T and P in Definition 1), within limited computational budgets.*

A comparison of classical machine learning and AutoML is in Table 2. Basically, in classical machine learning, human are heavily involved in configuring learning tools by operating feature engineering, model selection and algorithm selection. As a result, human take the most labor and knowledge-intensive job in machine learning practices. However, in AutoML, all these can be done by computer programs. To understand Definition 2 better, let us look back at those three examples in Table 1:

- *Automated feature engineering*: When original features are not informative enough, we may want to construct more features to enhance the learning performance. In this case, E is the raw feature, T is construction of features, and P is the performance of models which are learned with the constructed features. DSM [12] and ExploreKit [11] remove human assistance by automatically construct new features based on interaction among input features.
- *Automated model selection*: Here, E denotes input training data, T is a classification task, and P is the performance on the given task. When features are given, Auto-sklearn can choose proper classifiers and find corresponding hyper-parameters without human assistance.
- *Neural architecture search (NAS)*: When we try to do some image classification problems with the help of NAS, E is the collection of images, T is the image classification problem, and P is the performance on testing images. NAS will automatically search for a neural architecture, i.e., a classifier based on neural networks, that has good performance on the given task.

From above discussion, we can see that while good learning performance is desired, AutoML requires such performance can be obtained in an automatic manner. These set up three main goals for AutoML (Remark 2.1).

Remark 2.1 (Core goals)**.** *The three goals of AutoML:*

(A). Good performance: good generalization performance across various input data and learning tasks can be achieved;

(B). Less assistance from humans: configurations can be automatically done for machine learning tools; and

(C). High computational efficiency: the program can return an reasonable output within a limited budget.

Since AutoML itself can be seen as a machine learning tool (Figure 2), here we remark that the goal (A) actually intends to escape the "curse" of the notorious No Free Lunch theorems stated in [5] and [6]. These theorems state that in a noise-free scenario of supervised learning, all learning algorithms have the same generalization performance (error rate) when averaged over *all* possible learning tasks. Although these theorems are mathematically proven, it is hard (and even impossible) to apply them to the reality and make empirical test. This is because that the average on the performance over all possible learning tasks (with equal weights) is very brutal. It is highly possible that learning tasks in reality take up only a very narrow spectrum in all theoretically possible tasks.

Once above three goals can be realized, we can fast deploy machine learning solutions across organizations, quickly validate and benchmark the performance of deployed solutions, and let human focus more on problems that really need humans' engagements, i.e., problem definition, data collection and deployment in Figure 1. All these make machine learning easier to apply and more accessible for everyone.

2.2 Basic Framework

In Section 2.1, we have defined the AutoML problem (Definition 2) and introduced its core goals (Remark 2.1). In this section, we propose a basic framework for AutoML approaches.

2.2.1 Human Tuning Process

However, before that, let us learn how configurations are tuned by human. Such process is shown in Figure 4. Once a

learning problem is defined, we need to find some learning tools to solve it. These tools, which are placed in the right part of Figure 4, can target at different parts of the pipeline, i.e., feature, model or optimization in Figure 1. To obtain a good learning performance, we will try to set a configuration using our personal experience or intuition about the underneath data and tools. Then, based on the feedback about how the learning tools perform, we will adjust the configuration wishing the performance can be improved. Such a trial-and-error process terminates once a desired performance is achieved or the computational budgets are run out.

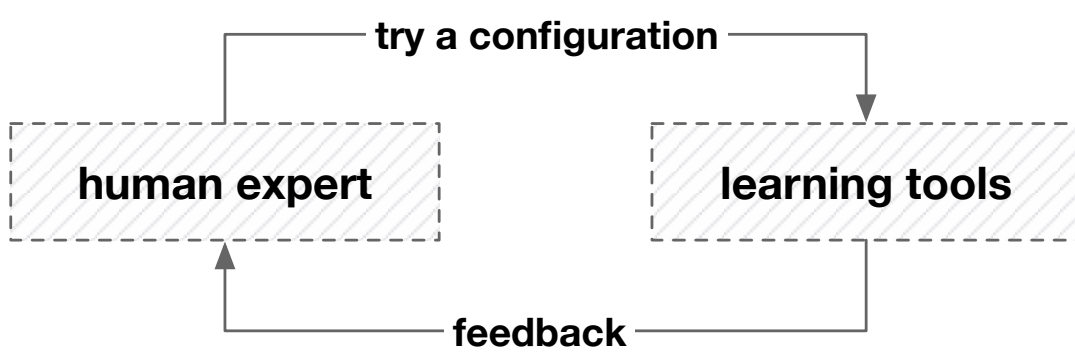

Fig. 4. The process of configurations tuned by humans.

2.2.2 Proposed AutoML Framework

Motivated by the human-involved process above and controlling with feedbacks in the automation [38], we summarize a framework for AutoML, as shown in Figure 6. Compared with Figure 4, in this figure, an AutoML controller takes the place of human to find proper configurations for the learning tools. Basically, we have two key ingredients inside the controller, i.e., the optimizer and the evaluator.

Their interactions with other components in Figure 6 are as follows:

- *Evaluator*: The duty of the evaluator is to *measure* the performance of the learning tools with configurations provided by the optimizer. After that, it generates *feedbacks* to the optimizer. Usually, to measure the performance of learning tools with given configuration, the evaluator needs to train a model based on the input data, which can be time-consuming. However, the evaluator can also directly estimate the performance based on external knowledge, which mimics humans' experience. Such estimation is very fast but may be inaccurate. Thus, for the evaluator, it needs to be efficient but also accurate in measuring the performance of configurations.
- *Optimizer*: Then, for the optimizer, its duty is to update or generate *configurations* for learning tools. The *search space* of the optimizer is determined by the targeted learning process, and new configurations are expected to have better performance than previous ones. However, feedbacks offered by the evaluator are not necessarily required or exploited by the optimizer. This depends on which type of the optimizer we are utilizing. Finally, the optimizer should be chosen based on the learning process and corresponding search space, as the latter determines the applicability of different optimization methods. We also wish the structure of the search space can be simple and compact so that more generic and efficient optimization methods can be employed.

As we will see, this framework is general enough to cover nearly all existing works (e.g. [7], [9], [11], [12], [22], [39], [40], [41], [42], [43], [44], [45], [46], just to name a few). In Section 6, we provide more detailed examples demonstrating how the it can cover existing works. Furthermore, this framework is also precise enough to help us setup taxonomies for AutoML approaches (Section 2.3), and it gives insight to the future direction of AutoML (Section 7.2).

2.3 Taxonomies of AutoML Approaches

In this section, we give taxonomies of existing AutoML approaches based on what and how to automate.

2.3.1 "What to automate": by problem setup

The choice of learning tools inspires the taxonomy based on problem setup in Figure 5(a), this defines "what" we want to make automated by AutoML. Basically, for general learning problems, we need to do feature engineering, model selection and optimization algorithm selection. These three parts together make up the full scope of general machine learning applications (Figure 1). We also list NAS there as a very important and special case. The reason is that NAS targets at deep models, where features, models and algorithms are configured simultaneously. The focus and challenges of AutoML problem under each setup are detailed in Section 3.

2.3.2 "How to automate": by techniques

Figure 5(b) presents the taxonomy by AutoML techniques. These are the techniques used for the controller, and categorize "how" we solve an AutoML problem. In general, we divide existing techniques into basic and experienced ones:

- *Basic techniques*: As there are two ingredients, i.e., the optimizer and evaluator, in the controller, we categorize basic techniques based on which ingredient they operating on. The optimizer focus on the searching and optimizing configurations, and there are many methods can be used, from simple methods as grid search and random search [47] to more complex ones as reinforcement learning [9] and automatic differentiation [48]. However, for the evaluator, which mainly measures the performance of learning tools with current configurations by determine their parameters, there are not many methods can be taken as basic ones.
- *Experienced techniques*: Experienced techniques learn and accumulate knowledge from past searches or external data. They usually need to be combined with basic techniques and enhance the optimizer and/or evaluator in various manners. Generally, there are two main methods popularly used in AutoML, i.e., meta-learning [29], [49] and transfer learning [31]. Due to the page limit, we will discuss experienced techniques in the appendix.

Note that, as E, T and P are also involved in the AutoML's definition (Definition 2), taxonomies of machine learning, e.g., supervised learning, semi-supervised learning and unsupervised learning, can also be applied for AutoML. However, they do not necessarily connect with removing human assistance in finding configurations (Figure 4). Thus, taxonomies here are done based on the proposed framework in Figure 6 instead. Finally, we focus on supervised AutoML

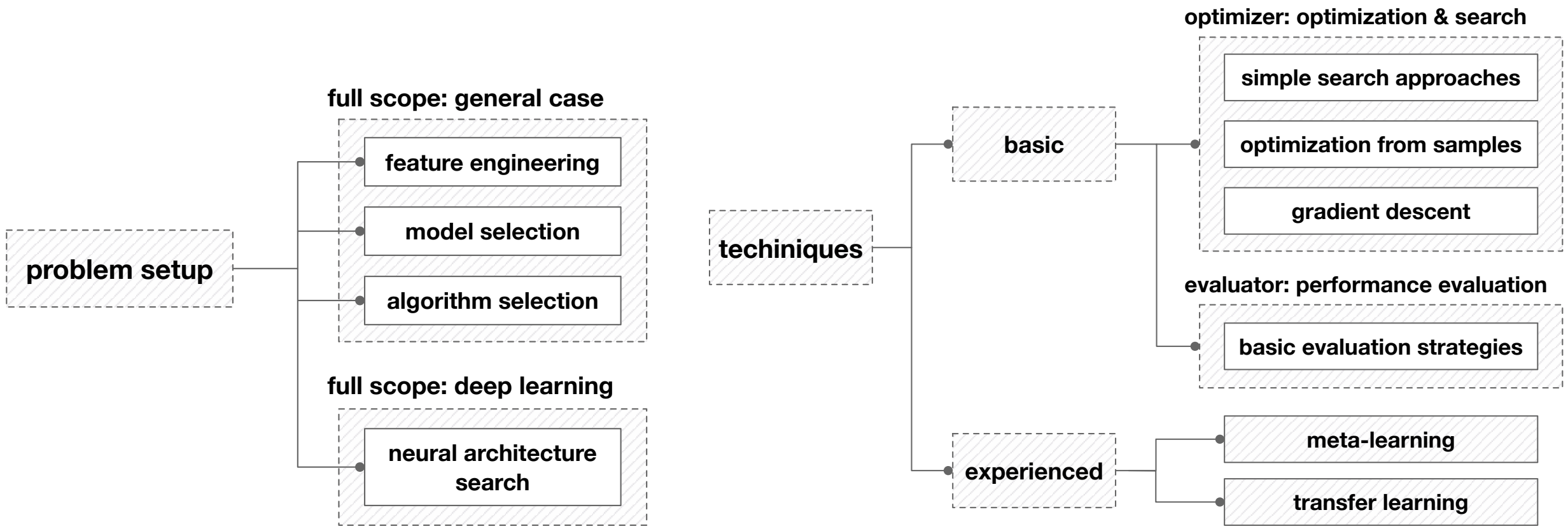

(a) "What to automate": by problem setup. (b) "How to automate": by techniques.

Fig. 5. AutoML approaches taxonomies by problem setup and techniques, which is inspired by the proposed framework in Figure 6. Taxonomy by problem setup depends on which learning tools we used, it clarifies "what" we want to make automated; taxonomy by techniques depends on the how we want to solve AutoML problems. Specifically, feature engineering, model selection and optimization algorithm selection together make up the full scope of general machine learning applications (Figure 1).

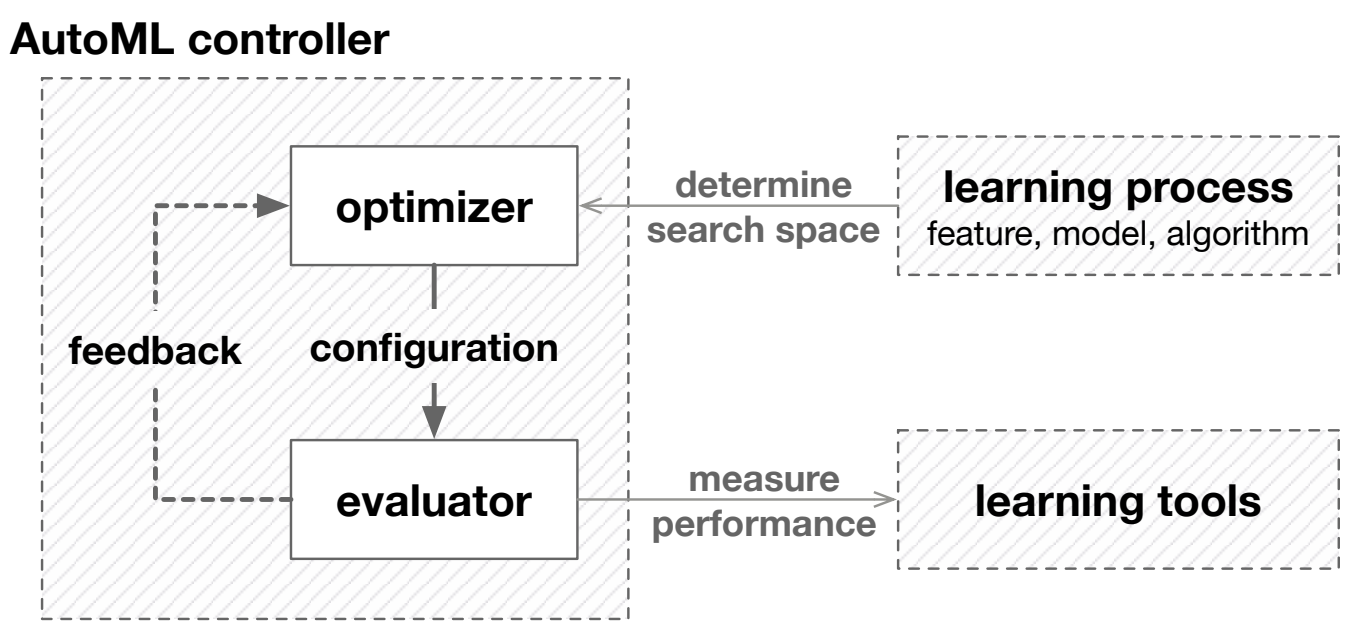

Fig. 6. Basic framework for how existing approaches solving AutoML problem. The dashed line (feedback) inside the controller, which depends on what techniques are used for the optimizer, is not a must.

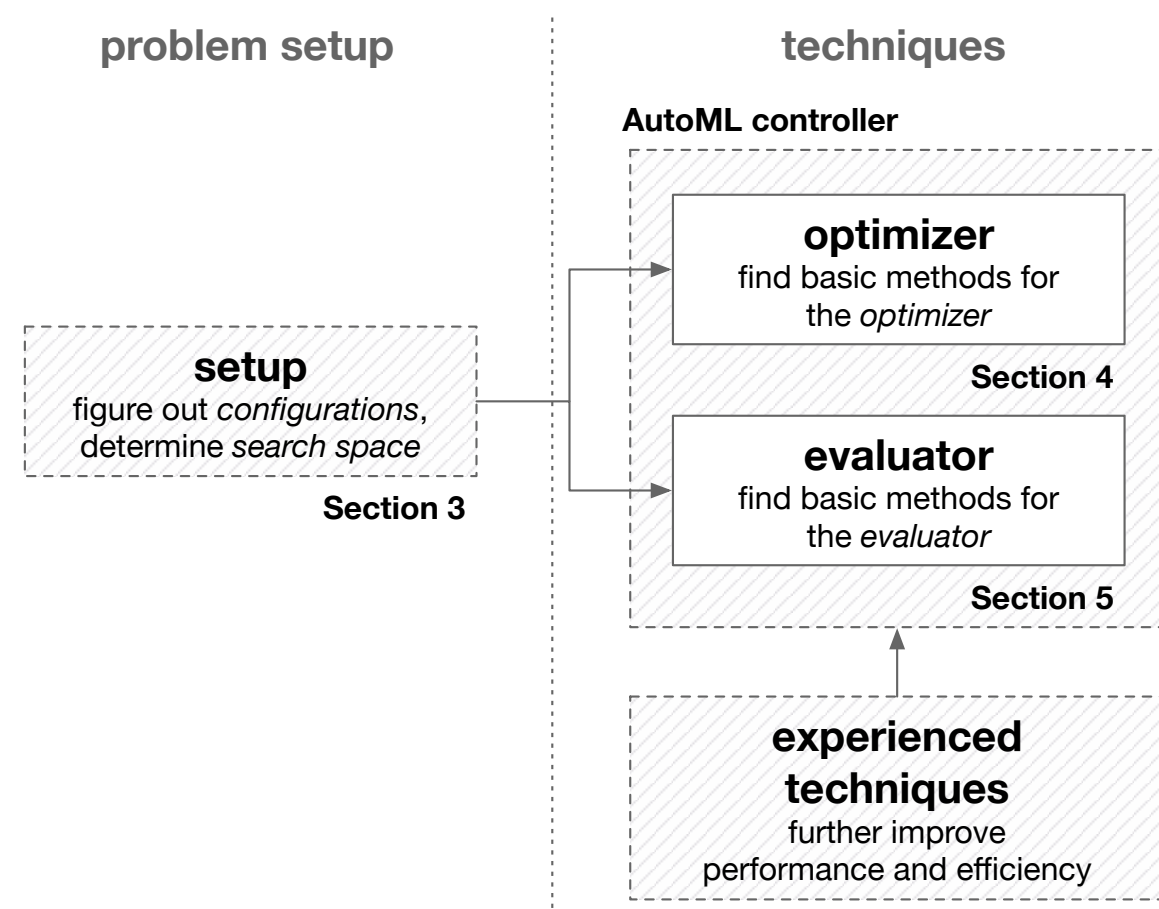

Fig. 7. Working flow of designing AutoML approaches based on the proposed framework (Figure 6) and taxonomies.

approaches in this survey as all existing works for AutoML are supervised ones.

2.3.3 Workflow based on Taxonomies

In the sequel, basic techniques and core issues they need to solve are introduced in Section 4 and 5 for the optimizer and evaluator respectively. The working flow of designing an AutoML approach is summarized in Figure 7, which also acts a a guidance through this survey.

3 Problem Settings

In this section, we give details on categorization based on problem setup (Figure 5(a)). Basically, it clarifies what to be automated. AutoML approaches do not necessarily cover the full machine learning pipeline in Figure 1, they can also focus on some parts of the learning process. In order to setup an AutoML problem, common questions that should be asked are:

Remark 3.1. *Three important questions to setup an AutoML problem are*

(A). What learning process we want to focus on?

(B). What learning tools can be designed and used?

(C). What are resultant corresponding configurations?

By answering these questions we can define the search space for an AutoML approach. Table 3 gives an overview on how the focused learning process changes the search space. In the sequel, we briefly summarize existing learning tools for each setup and what are the corresponding search space.

3.1 Feature Engineering

The quality of features, perhaps, is the most important perspective for the performance of subsequent learning models [1], [13]. Such importance is further verified by the success of deep learning models, which can directly learn a representation of features from the original data [67]. The problem of AutoML for feature engineering is to *automatically construct features from the data so that subsequent learning tools can have good performance*. The above goal can be further divided into two sub-problems, i.e., creating features from the data and enhance features' discriminative ability.

TABLE 3
The taxonomy of existing AutoML approaches by problem setup. For each setup, we need to select or design some learning tools, and then figure out the resulting configurations (see Remark 3.1).

learning process		learning tools	search space	examples
feature engineering		(subsequent) classifiers	feature sets	[11], [12], [50], [51], [52]
			feature enhancing methods and their hyper-parameters	[7], [8], [39]
model selection		classifiers	classifiers and their hyper-parameters	[7], [39], [42], [53], [54]
optimization algorithm selection		classifiers	algorithms and their hyper-parameters	[43], [55], [56], [57], [58]
full scope	general	classifiers	an union of search space in feature, model, and/or algorithm	[7], [8], [23], [39]
	neural architecture search (NAS)	neural networks	network structures	[9], [22], [25], [45], [59], [60], [61], [62], [63], [64], [65], [66]

However, the first problem heavily depends on application scenarios and humans' expertise, there are no common or principled methods to create features from data. AutoML only makes limited progress in this direction, we take it as one future direction and discuss it in Section 7.2.1. For now, we focus on feature enhancing methods.

3.1.1 Feature Enhancing Methods

In many cases, the original features from the data may not be good enough, e.g., their dimensionality may be too high or samples may not be discriminable in the feature space [67]. Consequently, we may want to perform some post-processing on these features to improve the learning performance. Fortunately, while human assistance is still required, there are common methods and principled ways to enhance features. They are listed as follows:

- *Dimension reduction*: It is the process of reducing the number of random variables under consideration by obtaining a set of principal variables. Dimension reduction is useful when the features have great redundancy or the feature dimensionality is too high. Techniques of this kind can be divided into feature selection and feature projection. Feature selection tries to select a subset of features from the original ones, where popular methods are greed search and lasso. Feature projection transforms original features to a new low-dimensional space, e.g., PCA [68], LDA [69], and recently developed auto-encoders [70].
- *Feature generation*: Unexplored interactions among original features, once discovered, may significantly improve the learning performance. Feature generation is to construct new features from the original ones based on some pre-defined operations [11], [12], [50], [51], [52], [71], [72], e.g., multiplication of two features, and standard normalization.
- *Feature encoding*: The last category is feature encoding, which re-interprets original features based on some dictionaries learned from the data. Since the dictionary can capture the collaborative representation in the training data, training samples that are not discriminable in the original space become separable in the new space. Popular examples of this kind are sparse coding [34] (and its convolutional variants [73]) and local-linear coding [74]. Besides, kernel methods can also be considered as feature coding, where basis functions act as the dictionary. However, kernel methods have to be used with SVM, and basis functions are designed by hand and is not driven by data.

While there are practical suggestions for using above feature enhancing tools, when facing with a new task, we still need to try and test.

3.1.2 Search Space

There are two types of search space for above feature enhancing tools. The first one is made up by hyper-parameters of these tools, and configuration exactly refers to these hyper-parameters [7], [8], [39]. It covers dimension reduction and feature encoding methods. For example, we need to determine the dimension of features when employing PCA, and the level of sparsity if sparse coding is used.

The second type of search space contains feature to be generated and selected. It commonly considered in feature generation, e.g., [11], [12], [50], [51], [52], [71], [72]. Basically, the search space is spanned by operations on original features. One example of new feature generated from plus, minus and times operations is shown in Figure 8. For these methods, a configuration is a newly generated feature in the search space.

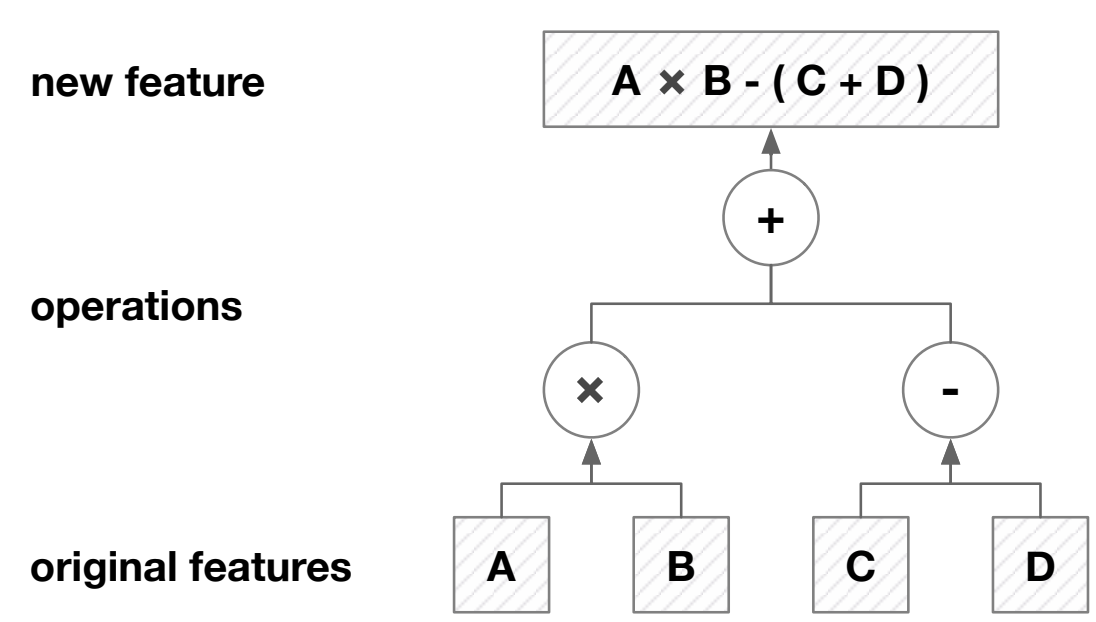

Fig. 8. An example of a newly generated feature ($A \times B - (C + D)$), which is based on plus ($+$), minus ($-$) and times ($\times$) operations.

3.2 Model Selection

Once features have been obtained, we need to find a model to predict the labels. Models selection contains two components, i.e., picking up some classifiers and setting their corresponding hyper-parameters. In this AutoML setup , the task is to *automatically select classifiers and set their hyper-parameters so that good learning performance can be obtained*.

3.2.1 Classification Tools

Many classification tools have been proposed in the literature, e.g., tree classifiers, linear classifiers, kernel machines and, more recently, deep networks. Each classifier has its own strength and weakness in modeling underneath data [1], [13]. Some out-of-the-box classifiers implemented in scikit-learn are listed in Table 4. As can be seen, different hyper-parameters are associated with each classifier. Traditionally, the choice among different classifiers and their hyper-parameters are usually determined by human with his/her experience in a trial-and-error manner.

TABLE 4
Example classifiers in Scikit-Learn and their hyper-parameters. Generally, hyper-parameters can be (a) discrete, e.g., number of neighbors in kNN, or (b) continuous. e.g., the value of penalty in logistic regression.

	number of hyper-parameters		
	total	discrete	continuous
AdaBoost	4	1	3
Bernoulli naive Bayes	2	1	1
decision tree	4	1	3
gradient boosting	6	0	6
kNN	3	2	1
linear SVM	4	2	2
kernel SVM	7	2	5
random forest	5	2	3
logistic regression	10	4	6

3.2.2 Search Space

In the context of model selection, the candidate classifiers and their corresponding hyper-parameters make up the search space. Figure 9 shows a hierarchical structure that is commonly used to represent the search space [7], [8], [39], [42], [53], [54]. The rationale behind this structure is that we need to determine the hyper-parameters only if the corresponding classifier is considered.

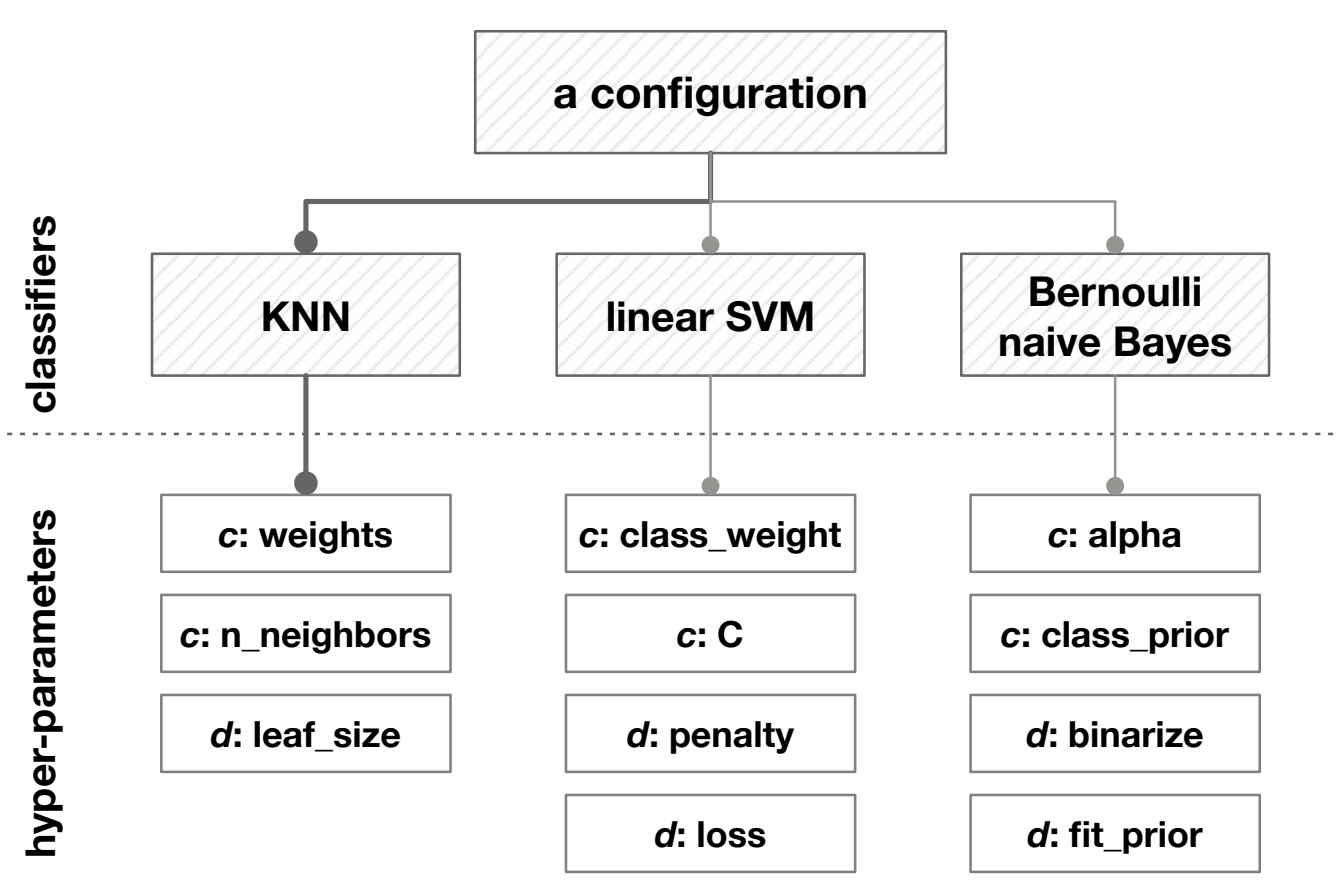

Fig. 9. An illustration of the search space for model selection, where KNN, linear SVM and Bernoulli naive Bayes classifiers are considered. Hyper-parameters are derived based on Scikit-Learn, "*c*:" indicates that the hyper-parameter is continuous while "*d*:" means that it is discrete. In this figure, a configuration is made up by the selection of KNN classifier and its values in corresponding hyper-parameters.

3.3 Optimization Algorithm Selection

The last and the most time consuming step of machine learning is the model training, where optimization is usually involved. For classical learning models, optimization is not a concern, since they usually employs convex loss functions and their performance obtained from various optimization algorithms are nearly the same [75]. Hence, efficiency is the main focus on the choice of optimization algorithm.

However, as the learning tools get increasingly more complex, e.g. from SVM to deep networks, optimization is not only the main consumer of computational budgets but also has a great impact on the learning performance [22], [76]. Consequently, the goal of algorithm selection is to *automatically find an optimization algorithm so that efficiency and performance can be balanced*.

3.3.1 Optimization Algorithms

For each learning tool, many algorithms can be used. Some popularly approaches to minimize smooth objective functions, like logistic regression, are summarized in Table 5. While gradient descent (GD) does not involve extra parameters, it suffers from slow convergence and expensive per-iteration complexity. Two popular variants of GD are limited memory-BFGS (L-BFGS) and stochastic gradient descent (SGD). The former is more expensive but converges faster [77], while in the latter each iteration is very cheap but many iterations are need before convergence [75].

TABLE 5
Some popular optimization algorithms for minimizing smooth objectives. L-BFGS needs to select the length of stored gradient (discrete); SGD needs to determine mini-batch size (discrete) and step-size (e.g. $\eta_0/(1+\lambda\eta_0 t)^c$ where t is the number of iterations, η_0, λ and c are continuous hyper-parameters [75]).

	number of hyper-parameters		
	total	discrete	continuous
GD	0	0	0
L-BFGS	1	1	0
SGD	4	1	3

3.3.2 Search Space

Traditionally, both the choices of optimization algorithms and their hyper-parameters are made by humans based on their understanding of the learning tools and observations of the training data. To automate algorithm selection, the search space is determined by configurations of optimization algorithms, which contains the choice of optimization algorithms and the values of their hyper-parameters, e.g, [55], [56], [57], [58]. There is also naturally a hierarchy in such search space, which is similar to that shown in Figure 9, as hyper-parameters of an algorithm will be considered only when the corresponding algorithm is selected.

3.4 Full Scope

In the this section, we discuss the full pipeline in Figure 1. There are generally two classes of full-scope AutoML approaches.

- The first one is *general case*. The learning process considered in this case is a combination of feature engineering, model selection and algorithm selection. The resulting search space is also an union of previous ones discussed in Section 3.1-3.3, as has been considered in [7], [8], [39], [42], [78] already.

- The second one is *NAS*, which targets at searching good deep network architectures that suit the learning problem. There are three main reasons why we discuss it in parallel with the full scope. First, NAS itself is currently an extremely hot research topic under which many papers have been published, i.e., [9], [22], [25], [26], [45], [59], [60], [61], [62], [63], [64], [65], [66], [79] and etc. The second reason is that the application domain for deep networks is relative clear, i.e., the domain of learning from low-semantic-level data such as image pixels. Finally, since the application domain is clear, domain-specific network architectures can fulfill the learning purpose, where feature engineering and model selection are both done by NAS.

3.4.1 Network Architecture Search (NAS)

Before describing the search space of NAS, let us look at what is a typical architecture of a convolutional neural network (CNN). As shown in Figure 10, basically, CNN is mainly made up by two parts, i.e., a series of convolutional layers and a fully connected layer in the last.

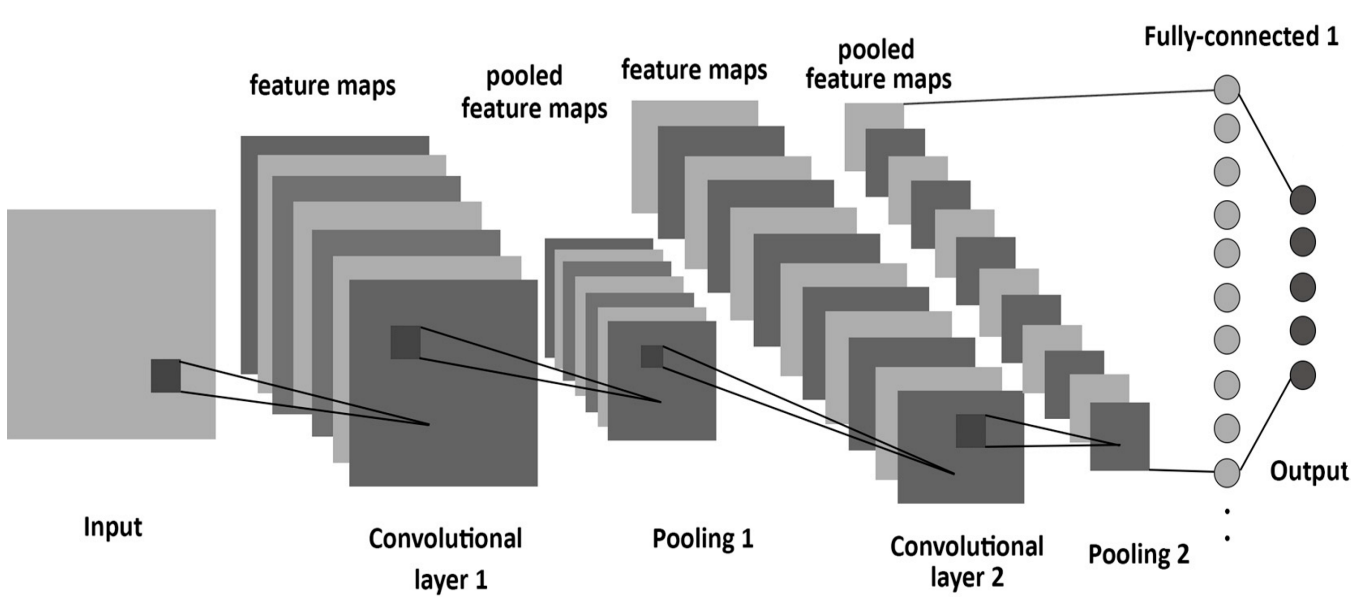

Fig. 10. A very typical CNN architecture, which contains filters, pooling and jump connections (the image is from [80]).

The performance of a CNN is mostly influenced by the design of convolutional layers [81], of which some common design choices are listed in Figure 11. The search space is made up by above design choices among all convolutional layers, and one configuration for NAS is a point in such a search space.

- number of filters
- filter height
- filter width
- stride height
- stride width
- skip connections

Fig. 11. Some design choices for one convolutional layer in a CNN.

Among various DNN architectures, we focus on CNN in this survey, but the presented idea can be similarly applied for other architectures, such as long-short-term-memory [82] and deep sparse networks [83].

4 Techniques for Optimizer

Once the search space is defined, as in the proposed framework (Figure 6), we need to find an optimizer to guide the search in the space. In this section, we discuss the basic techniques for the optimizer.

Remark 4.1. *Three important questions here are*

(A). what kind of search space can the optimizer operate on?

(B). what kind of feedbacks it needs?

(C). how many configurations it needs to generate/update before a good one can be found?

The first two questions determine which type of techniques can be used for the optimizer, and the last one clarifies the efficiency of techniques. While efficiency is a major concern in AutoML (see Remark 2.1), in this section, we do not categorize existing techniques based on it. This is because the search space is so complex where convergence rates for each technique are hard to analyze. We take it as one future direction in Section 7.2.4.

In the sequel, we divide the techniques for optimizer into four categories, i.e., simple search approaches, optimization from samples, gradient descent, and greedy search, based on the first two questions. An overview of the comparison among these techniques are in Table 6.

4.1 Simple Search Approaches

Simple search is a naive search approach, they make no assumptions about the search space. Each configuration in the search space can be evaluated independently. Grid search and random search are two common approaches.

- *Grid search* (brute-force): it is the most traditional way of hyper-parameters tuning. To get the optimal hyper-parameter setting, grid search have to enumerate every possible configurations in the search space. Discretization is necessary when the search space is continuous.
- *Random search*: it randomly samples configurations in the search space. Random search empirically performs better than brute-force grid search [47]. As shown in Figure 12, random search can explore more on important dimensions than grid search.

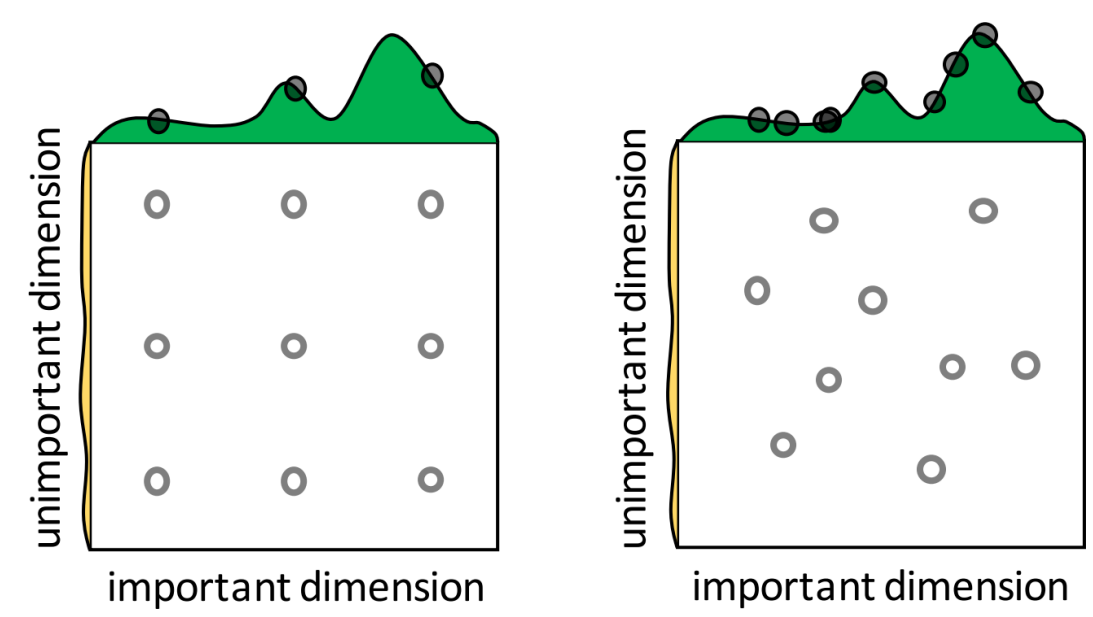

(a) Grid search. (b) Random search.

Fig. 12. Illustration of grid and random search with 9 trials in 2-D search problem. This figure also illustrates that random search does more exploration than grid search when the number of trials is same (the image is from [47]).

Simple search approaches gather the feedbacks from the evaluator merely to keep track of the good configurations. Because simple search does not exploit the knowledge gained from the past evaluations, it is usually inefficient. However, due to its simplicity, it is still popularly used in AutoML.

4.2 Derivative-Free Optimization

Optimization from samples [84] is a kind of smarter search approach compared with simple ones in Section 4.1. It it-

TABLE 6
Comparison of various techniques for the optimizer based on Remark 4.1. Multi-step means the configuration can be made up by a several decision steps.

type	method	conti-nuous	discrete	multi-step	example	feedback in example
simple search		√	√	×	[47]	none
optimization from samples	evolutionary algorithm	√	√	×	[15]	performance on validation set
	Bayesian optimization	√	√	×	[39]	performance on validation set
	reinforcement learning	√	√	√	[9]	performance on validation set (reward) and a sequence of configurations (state)
gradient descent		√	×	×	[41]	performance on validation set and gradients w.r.t hyper-parameters
greedy search		×	√	√	[10]	performance on validation set

eratively generates new configurations based on previously evaluated samples. Thus, it is also generally more efficient than simple search methods. Besides, it does not make specific assumptions about the objective.

In the sequel, according to different optimization strategies, we divide existing approaches into three categories, i.e., heuristic search, model-based derivative-free optimization, and reinforcement learning.

4.2.1 Heuristic Search

Heuristic search methods are often inspired by biologic behaviors and phenomenons. They are widely used to solve optimization problems that are non-convex, non-smooth, or even non-continuous. The majority of them are population-based optimization methods, and differences among them are how to generate and select populations. The framework of heuristic search is shown in Figure 13. The initialization step generates the first population (a bunch of configurations in AutoML). At each iteration, a new population is generated based on the last one, and the fitness (performances) of the individuals are evaluated. The core idea of heuristic search is how to update the population.

Some popular heuristic search methods are listed as follow:

- *Particle swarm optimization* (PSO) [53]: PSO is inspired by the behavior of biological communities that exhibit both individual and social behavior; examples of these communities are flocks of birds, schools of fishes and swarms of bees. Members of such societies share common goals (e.g., finding food) that are realized by exploring its environment while interacting among them. At each iteration, the population is updated by moving towards the best individuals. PSO optimizes by searching the neighborhoods of the best samples. It has a few hyper-parameters itself and can be easily parallelized. In such way, PSO hopes to find the best position in search space.
- *Evolutionary algorithms* [85]: Evolutionary algorithms are inspired by biological evolution. The generation step of evolutionary algorithms contains crossover and mutation. Crossover involves two different individuals (ancestors) from the last generation. It combines them in some way to generate an new individual. In principal, the more promising an individual is, the more likely is it to be chosen as an ancestor. Mutation, on the other hand, slightly changes an individual to generate a new one. With crossover mainly to exploit and mutation mainly to explore, the population is expected to evolve towards better performance.

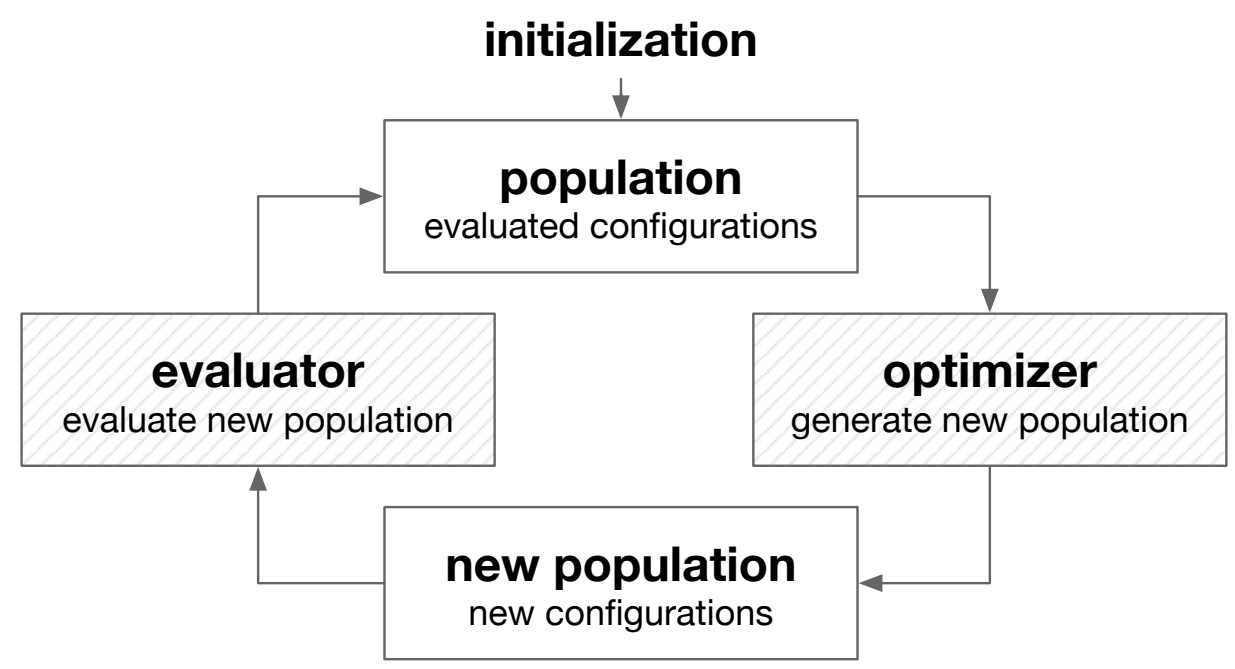

Fig. 13. Work flow of heuristic search. It is the population-based search approach, and starts with a initialization process.

The above methods have been widely applied in AutoML. For example, evolutionary algorithms has been applied in feature selection and generation [50], [86], [87], [88], [89], and model selection [90]. PSO has been used for model selection [53], [91], feature selection for support vector machine (SVM) [91], and hyper-parameter tuning for deep networks [92]. While evolutionary algorithms have already been used in NAS one decade ago [93], [94], [95], it is only recently that better performance than human designed architecture are achieved [15], [96], [97], [98]. In these works, network structures are encoded with binary strings, on which the evolutionary operations are performed.

4.2.2 Model-Based Derivative-Free Optimization

The general framework of model-based derivative-free optimization is showed in Figure 14. It is different from the heuristic search in sense that model-based optimization builds a model based on visited samples. Full utilization of feedbacks from the evaluator helps it generate more promising new samples. The popular methods of this kind are Bayesian optimization, classification-based optimization and optimistic optimization:

- *Bayesian optimization* [99], [100], [101]: Bayesian optimization builds a probabilistic model, e.g., Gaussian process [102], [103], tree-based model [104], [105], or deep network [106], that maps the configurations to their performance with uncertainty. Then, it defines an acquisition function based on the probabilistic model, e.g., expected improvement, upper confidence bounds, to balance exploration and exploitation during search. At each

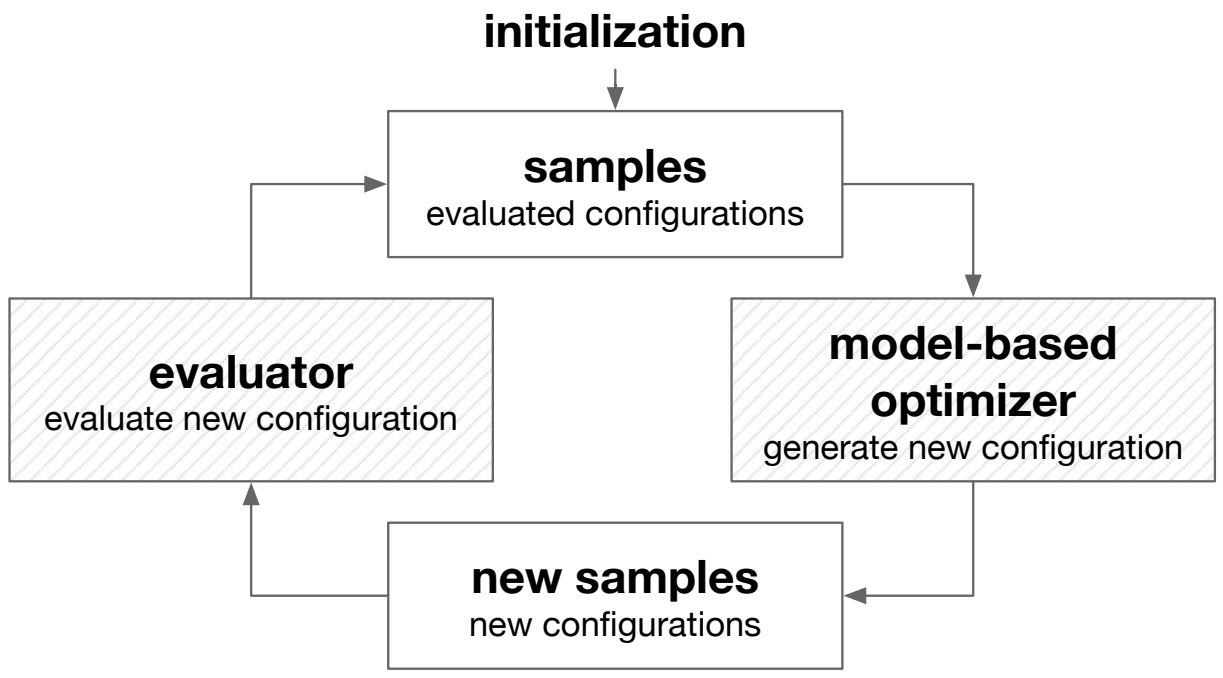

Fig. 14. Work flow of model-based derivative-free optimization. It is different from the heuristic search. The most important component of model-based optimization is the model built on previous samples.

iteration, a new sample is generated by optimizing the acquisition function, and used to update the probabilistic model once it is evaluated.

- *Classification-based optimization* (CBO) [44], [108], [109]: Based on previous samples, classification-based optimization learns a classifier that divides the search space into positive and negative areas. Then, new samples are randomly generated in the positive area where it is more likely to get better configurations. The learned classifiers can be very simple, which produce decision boundary in parallel with coordinates of the search space. Thus, classification-based optimization is usually very efficient.
- *Simultaneous Optimistic optimization* (SOO) [110], [111]: SOO is a branch-and-bound optimization algorithm. A tree structure is built on the search space where each leaf node bounds a sub area. SOO deeply explores the search space by expanding leaf nodes according to some strategies. Trough the tree model, SOO can balance exploration and exploitation to find the global optimum when the objective function is local-Lipschitz continuous [111]. But it also suffers from the curse of dimensionality because the tree grows extremely complicated when dimensionality the search space is high.

Due to its long history and sound theoretical justification, Bayesian optimization is perhaps the most popularly used method in this category. Early attempts include [99], [102], [112], [113], which have shown a promising performance of Bayesian optimization for hyper-parameter tuning. Later on, it has been applied in sklearn [7], [114] and Weka [8], [39] for automatic configuration of out-of-box classifiers. More recently, CBO has been developed as a better method than Bayesian optimization for hyper-parameter tuning [44] and policy search [108].

4.2.3 Reinforcement Learning

Reinforcement learning (RL) [115] is a very general and strong optimization framework, which can solve problems with delayed feedbacks. Figure 15 illustrates its general framework when used in AutoML. Basically, the policy in RL acts as the optimizer, and its actual performance in the environment is measured by the evaluator. However, unlike previous methods, the feedbacks (i.e., reward and state) do not need to be immediately returned once an action is taken. They can be returned after performing a sequence of actions.

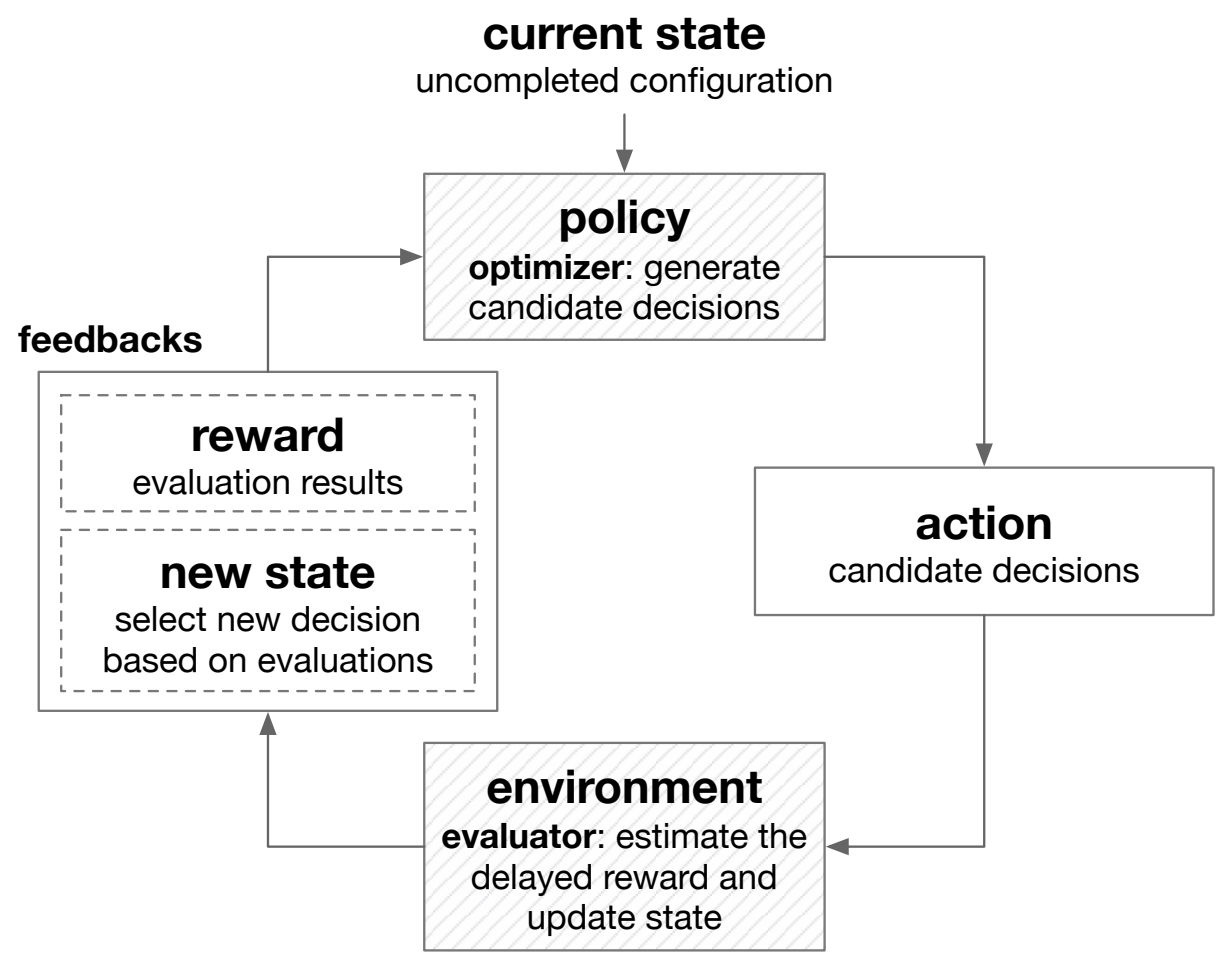

Fig. 15. Workflow of reinforcement learning. It is different from heuristic search and model-based derivative-free optimization as feedbacks need not be immediately returned for receiving a configuration.

Resulting from the above-mentioned unique property, RL is recently used in NAS [9], [16]. The reason is that CNN can be built layer-by-layer, and the design of one layer can be seen as one action given by the optimizer. However, the performance of an architecture can only be evaluated after its whole structure is composed, which implies a delayed reward. Thus, the iterative architecture generation naturally follows the property of RL (see details in Section 6.2). However, due to the delayed feedbacks, AutoML with reinforcement learning is highly source-consuming, and more efficient methods needs to be explored. Some current endeavors addressing this problems are learning transferable architectures from smaller data sets [23], and cutting the search space by sharing parameter [116], [117].

Besides, a special case of RL, i.e., bandit-based approach, where rewards are returned for each action without a delay, is introduced to AutoML for hyper-parameter optimization [118], [119]. Finally, RL has also been used for optimization algorithms search [22], automated feature selection [120], and training data selection in active learning [121].

4.3 Gradient descent

Optimization problems of AutoML is very complex, and the objective is usually not differentiable or even not continuous. Thus, gradient descent is not as popular as methods in Section 4.2. However, focusing on some differentiable loss function [122], e.g., squared loss and logistic loss, continuous hyper-parameters can be optimized by gradient descent. Compared with above methods, gradients offer the most accurate information where better configurations locates.

Unlike traditional optimization problems whose gradients can be explicitly derived from the objective, in AutoML problems, the gradients need to be numerically computed. Usually, this can be done with finite differentiation methods [122] but at high costs. For some traditional machine learning methods, e.g., logistic regression and SVM, the approximate gradient is proposed to search continuous hyper-parameters [123]. The computation of exact gradients relies on the convergence of model training. Through inexact gradient, hyper-parameters can be updated before the model

training converges, which makes gradient descent method more efficient.

Another way to compute gradients is through reversible learning (also named as automatic differentiation) [48]. It computes gradients with chain-rule, which is also used in the back-propagation process of network training. It has been applied in deep learning hyper-parameter search [41], [124].

4.4 Greedy search

Greedy search is a natural strategy to solve multi-step decision-making problem. It follows a heuristic that makes locally optimal decision at each step with the intent of finding a global optimum. For example, in travel salesman problem, greedy search selects to visit the nearest city at each step of the journey. Greedy search cannot find the global optimum, but it can usually find a local optimum which approximates the global optimum in a reasonable time cost. Besides, such good empirical performance is also theoretically justified in many applications, e.g., feature selection [125] and submodular optimization [126].

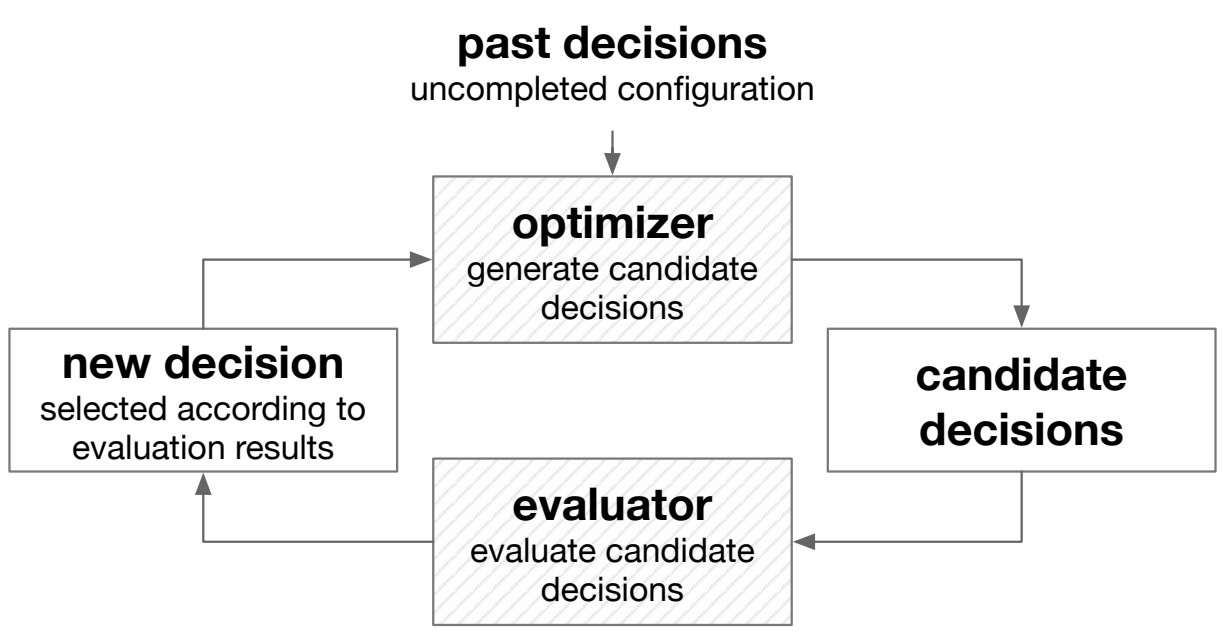

Fig. 16. Workflow of greedy search. It targets at multi-step decision problems, where a local optimal decision is made at each step.

Multi-step decision-making problems are also commonly encountered in AutoML. For example, in NAS problem, the architecture for each layer needs to be decided, and greedy search is applied in [24] for multi-attribute learning problems; greedy search is also employed in [10], [117] to search block structures within a cell, which is later used to construct a full CNN. Besides, in feature generation problems where the search space can be prohibitively large, greedy search is recently considered in [11], [12] to generate more discriminative features with original ones.

4.5 Other techniques

Finally, there are some techniques that do not fall into above categories. They are usually developed case-by-case. Currently, a popular one is to change the landscape of the search space so that more powerful optimization techniques can be used. For example, in NAS, as the configuration space is discrete soft-max is used in [46] to change the search space to a continuous one, which enables the usage of gradient descent instead of RL; an encoder-decoder framework is used in [127], which also maps discrete configurations into a continuous search space.

5 Techniques for Evaluator

In Section 4, we discussed how to choose a proper basic technique for the optimizer. In this section, we will visit techniques for another component, i.e., the evaluator in Figure 6. Once a candidate configuration is generated, the evaluator needs to measure its performance. This process is usually very time-consuming as it involves model training for most of the times.

Remark 5.1. *Three important questions to determine the basic technique for the evaluator are:*

(A). Can the technique provide fast evaluation?
(B). Can the technique provide accurate evaluation?
(C). What feedbacks need to be provided by the evaluator?

As illustrated in Figure 17, there is usually a trade-off between the focus of questions (A) and (B) as faster evaluation usually leads to degraded result, i.e., with lower accuracy but larger variance. The last question in Remark 5.1 is a design choice, it also depends on choices of the optimizer. For example, as shown in Table 6, while Bayesian optimization only requires the performance, gradient descent methods in addition need gradient information.

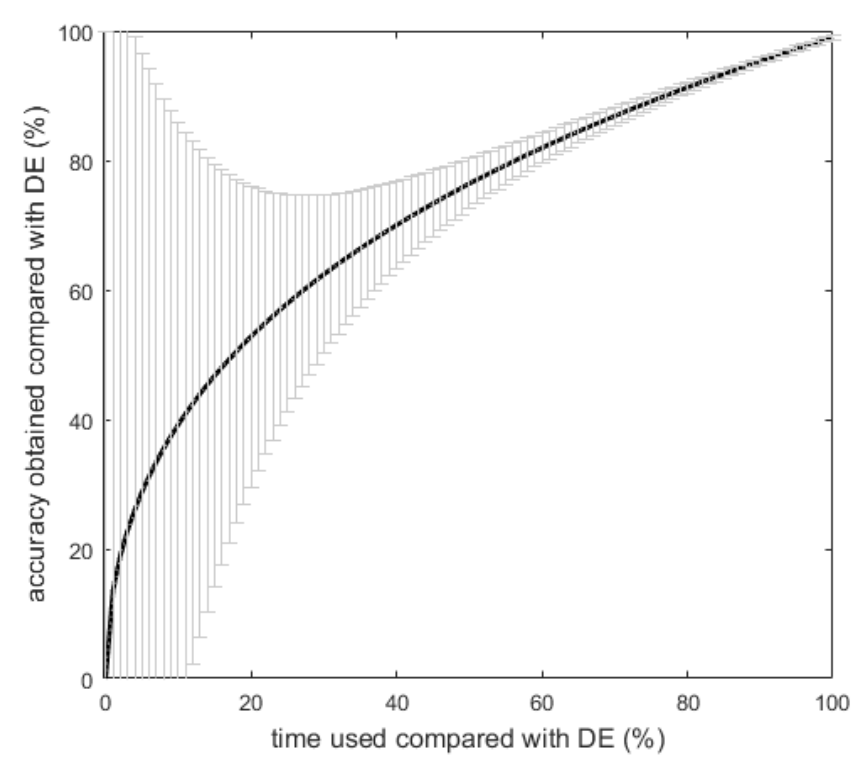

Fig. 17. The trade off between evaluation's accuracy and time, where DE denotes direct evaluation (see Section 5.1) and both time and accuracy are measured relatively to that of DE. The gray lines indicate variance in accuracy obtained.

5.1 Techniques

Unlike the optimizer, the evaluator seldom cares about the search space of configurations. Since the majority of optimizers generate candidates that can be directly applied on learning tools, the most simple and straightforward way to evaluate them is to learn the model parameters and estimate the performance:

- *Direct evaluation*: This is the simplest method, where the model parameters are learned on the training set, and the performance is measured on the validation set afterwards. Direct evaluation is often accurate but expensive.

In AutoML problems, usually, many candidate configurations will be generated and evaluated. The direct evaluation approach, though very accurate, is usually prohibitively expensive to be invoked repeatedly. Consequently, some

other methods have been proposed for acceleration by trading evaluation accuracy for efficiency:

- *Sub-sampling*: As the training time depends heavily on the amount of training data, an intuitive method to accelerate evaluation is to train parameters with a subset of the training data. This can be done by either using a subset of samples, a subset of features or multi-fidelity evaluations [128]. In general, the less training data is used, the faster and more noisy will be the evaluation.
- *Early stop*: In classical machine learning, early stop is a popular method to prevent over-fitting. However, in the context of AutoML, it is usually used to cut down the training time for unpromising configurations. Such configurations can usually be easily identified at the early stage of model training, with their performance monitored on the validation set [65], [105], [129], [130]. If a poor early-stage performance is observed, the evaluator can terminate the training and report a low performance to indicate that the candidate is unpromising. Early stop cuts down the total running time of AutoML, but also introduces noise and bias to the estimation as some configurations with bad early-stage performance may eventually turn out to be good after sufficient training.
- *Parameter reusing*: Another technique is to use parameters, of the trained models in previous evaluations, to warm-start the model training for the current evaluation. Intuitively, parameters learned with similar configurations can be close with each other. For a candidate that is close to previously evaluated configurations, parameters of the latter can be a good start point for training and may lead to faster convergence and better performance. In such cases, parameter reusing can be very helpful [131]. However, as different start points may lead convergence to different local optima, it sometimes brings bias in the evaluation [132].
- *Surrogate evaluator*: For configurations that can be readily quantized, one straightforward method to cut down the evaluation cost is to build a model that predicts the performance of given configurations, with experience of past evaluations [10], [43], [59], [130], [133], [134]. These models, serving as surrogate evaluators, spare the computationally expensive model training, and significantly accelerate AutoML. Surrogate evaluators can predict not only the performance of learning tools, but also the training time and model parameters. However, their application scope is limited to hyper-parameter optimization since other kinds of configurations are often hard to quantize, which hinders surrogate model training. Finally, it should be noted that, while surrogate models are also used in sampled-based optimization techniques (Section 4.2.2), they do not act as surrogate evaluators, but are used to generate potentially promising configurations.

Direct evaluation, due to its simplicity and reliability, is perhaps the most commonly used basic evaluator technique in AutoML. Sub-sampling, early stop, parameter reusing, and surrogate evaluator, enhance *Direct evaluation* in various directions, and they can be combined for faster and more accurate evaluation. However, the effectiveness and efficiency of these techniques depend on the AutoML problem and data, and it is hard to quantitatively analyze their improvement over *Direct evaluation*.

While the basic techniques for evaluators are seemingly much fewer than those for optimizers, it does not make the evaluator less important. In fact, the accuracy and efficiency of the evaluator have a great impact on both the quality of the final result and the runtime of the AutoML program.

6 Representative Examples

In this section, we revisit three examples mentioned in Section 1, i.e., Auto-sklearn [7], NASNet [9] and ExploreKit [11]. We will show in detail how these methods follow the basic framework of AutoML proposed in Section 2.2.2.

6.1 Model Selection using Auto-sklearn

As each learning problem has its own preference over learning tools [1], suitable machine learning models should be chosen specifically for each learning problem. Automated model selection is a very typical application of AutoML (Section 3.2). In the context of supervised classification, the purpose is to automatically select best classifiers and setup their hyper-parameters properly.

An representative example of automated model selection approach is Auto-sklearn [7], which is built on Scikit-Learn [14] package. Some widely used classifiers and their hyper-parameters are listed in Table 4. In Auto-sklearn, model selection is formulated as a CASH problem (Example 1), which aims to minimize the validation loss with respect to the model as well as its hyper-parameters and parameters.

Example 1 (CASH Problem [7], [39]). *Let $\mathcal{F} = \{F_1, \cdots, F_R\}$ be a set of learning models, and each model has hyper-parameter θ_j with domain Λ_j, $\mathcal{D}_{train} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)\}$ be a training set which is split into K cross-validation folds $\{\mathcal{D}^1_{train}, \cdots, \mathcal{D}^K_{train}\}$ and $\{\mathcal{D}^1_{valid}, \cdots, \mathcal{D}^K_{valid}\}$ with $\mathcal{D}^i_{train} \cup \mathcal{D}^i_{valid} = \mathcal{D}_{train}$ for $i = 1, \ldots, K$. Then, the Combined Algorithm Selection and Hyper-parameter (CASH) optimization problem is defined as*

$$F^{*}, \theta^{*} = \underset{\substack{\theta\in\Lambda_j \\ F_j\in\mathcal{F}}}{\operatorname{argmin}} \frac{1}{K} \sum_{i=1}^{K} \min_{\mathbf{w}} \mathcal{L}\left(F_j(\mathbf{w};\theta), \mathcal{D}^i_{train}, \mathcal{D}^i_{valid}\right), \quad (2)$$

where $\mathcal{L}\left(F_j(\mathbf{w}_j;\theta_j), \mathcal{D}^i_{train}, \mathcal{D}^i_{valid}\right)$ denotes the loss that F_j achieves on $\mathcal{D}^i_{valid}$ with parameter $\mathbf{w}_j$, hyper-parameter θ_j and training data $\mathcal{D}^i_{train}$.

As discussed in Section 3.2.2, in the model selection problem, a candidate configuration comprises a classifier in Scikit-Learn and its hyper-parameters, and the search space is spanned by configurations of this kind. However, (2) is very hard to optimize. First, since there is no explicit expression for the objective, we do not know its properties, e.g., the degree of smoothness, which can be very helpful for traditional optimization problems. In addition, the decision variables, i.e., θ and F_j, may not even be continuous, e.g., θ for kNN includes the number of nearest neighbors that is a discrete variable. Finally, in order to estimate the validation loss, we need to train the model F_j and update its parameter $\mathbf{w}_j$, which is usually very expensive.

TABLE 7
Illustration of how examples Section 6 fall into the proposed framework in Figure 6. The "naive" means there is no special design in the evaluator, the evaluation is directly done by optimizing parameters of learning tools on the training data.

example	controller		learning tools
	optimizer	evaluator	
Auto-sklearn [7]	SMAC [105] algorithm (warm-start by meta-learning)	direct evaluation (train model parameter with optimization algorithms)	out-of-box classifiers
NASNet [9]	recurrent neural networks (trained with REINFORCE algorithm [135])	direct evaluation (train child network with stochastic gradient descent)	convolutional neural networks
ExploreKit [11]	greedy search algorithm	classifiers trained with meta-features	subsequent learning models

In [39], sequential model-based algorithm configuration (SMAC) [105], a tree-based Bayesian optimization method, was used as the optimizer to solve (2). Then, for the evaluator, the basic method, i.e., direct evaluation was used. Besides, meta-learning was employed as the experienced technique to get better initialization. Finally, rather than discarding models searched in the configuration space, Auto-sklearn stores them and to use a post-processing method to construct an ensemble of them. This automatic ensemble construction makes the final result more robust against overfitting.

Table 3 in Appendix A (from Table 3 in [7]) shows the performance of Auto-sklearn. As we can see, Auto-sklearn consistently finds the best configuration in its search space, which demonstrates the success of AutoML in model selection.

6.2 Reinforcement Learning for NAS (NASNet)

As mentioned in Section 1, since the success of AlexNet on image classification of ImageNet data set [17], the design of new neural architectures has become the main means to get better predicting performance in the deep learning domain. This raises the research interests in automatic searching network architectures for given tasks [9], [15], [16].

Taking CNN as an example, the design choices for each convolution layer are listed in Figure 11. A configuration (architecture design) contains designs of all convolution layers in a CNN, which leads to a very large search space. The common approach is to build the architecture layer-by-layer. As shown in Figure 18 one configuration in the NAS problem is a sequence of design decisions. However, in a CNN architecture, the effect of lower layer design depends on those of higher ones [81], [131]. This makes the final performance of the whole architecture a delayed reward. Motived by such facts, RL is employed in NAS to search for a optimal sequence of design decisions [16], [64]. Besides, the direct evaluation is used in these works as the evaluator.

As RL is slow to converge, to make the search faster, transfer learning, which is firstly used to cut the search space into several blocks, was developed in [23], [64]; then, a parameter sharing scheming is proposed in [116] to further narrow down the search space; and greedy search has also been considered as a replacement for RL [10], [117].

Table 1 in Appendix A presents a comparison between human-designed CNN and those searched by NAS. As can be seen, through AutoML, CNN with less depth and fewer parameters with comparable performance to that of state-of-art CNN designed by humans.

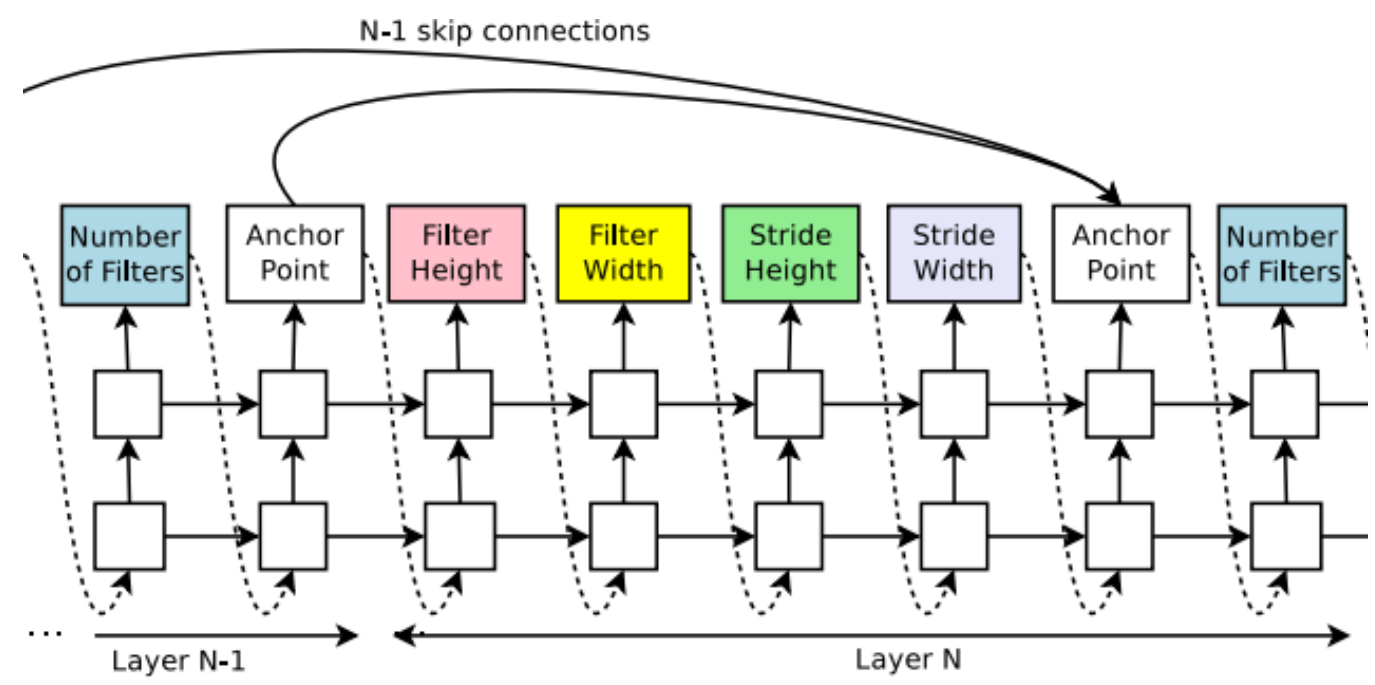

Fig. 18. An illustration of the multi-step decision process of generating a configuration for one convolutional layer using recurrent neural network (RNN) (the image is from [22]), where anchor point is used to deal with skip connections.

6.3 Feature Construction using ExploreKit

One of the most representative works in automatic feature construction is ExploreKit [11]. It aims to generate new features to improve the performance of the learning tools. In this setting, a candidate configuration is a set of generated features. Figure 19 shows the system architecture of ExploreKit. Its main body of the is an iterative process, where each iteration comprises three steps, candidate feature generation, candidate feature ranking, candidate feature evaluation and selection. It is by instinct a greedy search strategy since the search space is extremely large. Additionally, meta-learning techniques are employed in the ranking step to fast estimate the usefulness of candidate features and accelerate the subsequent evaluation step.

The optimizer of ExploreKit employs a greed rule-based strategy to explore the search space. At the candidate feature generation step, new features are constructed by applying operators on features that are already selected. Employed operators include: 1) unary ones (e.g., discretization, normalization), 2) binary ones (e.g., $+$, $-$, $\times$, $\div$), and 3) higher-order ones (e.g., GroupByThenMax, GroupByThenAvg). A predetermined enumerating procedure is invoked to apply these operators on all selected features that are applicable to generate candidates. In order to limit the size of candidate feature set, generated features will not be reused to further generate new candidates.

Since ExploreKit generates candidates exhaustively, evaluating all these features may be computational intractable. To address this issue, ahead of the evaluation and selection step, ExploreKit uses meta-learning to roughly rank all candidate features. At this step, a ranking classifier, trained with historical knowledge on feature engineering, is

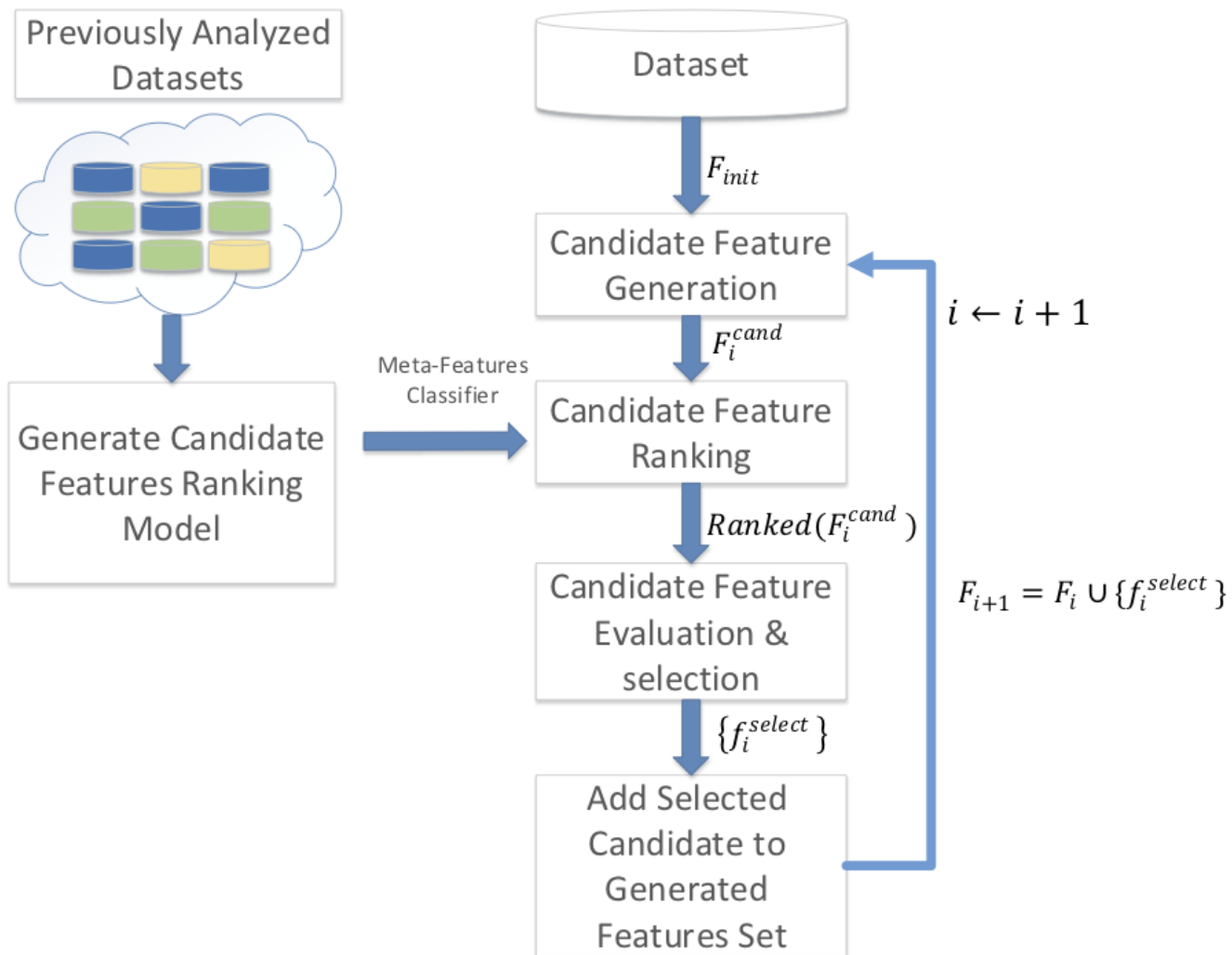

Fig. 19. The system architecture of ExploreKit (the image is from [11]).

used to fast identify promising candidates. In the evaluation step that follows, features predicted more useful will be considered first.

Finally, ExploreKit conducts more expensive and accurate evaluations on the candidate features. Error reduction on the validation set is used as the metric for feature importance. Candidate features are evaluated successively according to their ranking, and selected if their usefulness surpass a threshold. This procedure terminates if enough improvement is achieved. The selected features will be used to generate new candidates in the following iterations.

Table 2 in Appendix A presents the improvement ExploreKit achieves on ten representative data sets. The performance of the generated features depends on the data set and the base classifier, and some encouraging results have been observed.

7 Summary

In this section, we briefly summarize the current status of AutoML in the academy and industry (Section 7.1), and discuss its future works (Section 7.2).

7.1 Current Status

Nowadays, in academy, AutoML is a very complex problem and also an extremely active research area, and many new opportunities and problems in AutoML are not yet visited. Lots of papers focusing on AutoML appear on various conferences and journals, such as ICML, NIPS, KDD, AAAI, IJCAI and JMLR (see the reference list) and some of their open-source projects are extremely popular on GitHub (Table 9); many workshops are organized, such as AutoML workshop at ICML from 2014 to 2018 [136], [137], [138], [139], [140]; some competitions such as, AutoML Challenge at PAKDD [141] and NIPS [142], are hold as well.

In industry, many products of AutoML are also available. Some examples are listed in Table 8. All these companies try to develop an end-to-end AutoML pipeline (Figure 1), but with different focuses. For example, Feature Labs targets at feature engineering; NAS is built in Google's Cloud to help design deep networks for computer vision applications. All these products significantly reduce their customers' efforts in deploying machine learning tools for real applications.

7.2 Future Works

First of all, as AutoML focuses on how to do machine learning in a special way (Definition 2), the current trends in machine learning can also be seen as future works of AutoML. Examples are human interpretability [143] and privacy [144] in machine learning. In the rest of this section, we focus more on future works that are closely related to the framework proposed in Section 2.2.

7.2.1 Problem setup

How to create features from the data is a fundamental problem not only in machine learning, but also many related areas. For example, scale-invariant feature transform (SIFT) [145] and histograms of oriented gradients (HoG) [146], successfully generalized for many problems and applications in computer vision. Similarly, in natural language processing (NLP), "term frequency-inverse document frequency" (TF-IDF) [147], is easy to calculate and performs well across many NLP tasks.

As what data should be used for the subsequent learning tools heavily depends on application scenarios (Figure 1), there are no general rules or unified models for creating features from the data. Specifically, interactions underneath the given data need to be understood by humans, then features are usually designed based on humans' expertise, e.g., SIFT and TF-IDF. Due to such difficulties, automatically creating features from the data have only became possible for some specific data types. For example, SIFT has been replace by CNN and TF-IDF have been taken over by RNN. More recently, some endeavors have been made for relational data set, i.e., DSM [12] (Table 2). Their success lies on utilizing the common relationships inside the data. Besides, with those automatically generated features, the performance of subsequent learning tools are significantly improved. Thus, one important future work is to automatically creating features from the data.

7.2.2 Techniques

In Section 2.2, we have proposed a framework, where configurations are updated based on alternative iterations between the optimizer and evaluator. However, AutoML can be extremely resource-consuming because of the extensive and complex search space, and the expensive evaluation. For example, 800 GPUs and 28 days are used in [16] for NAS with reinforcement learning to discover the convolutional architecture on CIFAR-10 data set. Thus developing more efficient basic techniques is always desired. Higher efficiency can be achieved by either proposing algorithms for the optimizer, which visit less configurations before reaching a good performance, or designing better methods for the evaluator, which can offer more accurate evaluations but in less time.

One interesting direction is to simultaneously optimize configurations and parameters, such methods have been recently explored in NAS [46], [148] and automated searching

TABLE 8
A brief list of AutoML products in the industrials, and "——" indicated no official announcements are found.

	company	AutoML products	customer
public company	Google	Deployed in Google's Cloud	Disney, ZSL, URBN
	Microsoft	Deployed in Azure	——
	IBM	IBM Watson Studio	——
startup	H2O.ai	H2O AutoML Package	AWS, databricks, IBM, NVIDIA
	Feature Labs	Feature Labs' platform	NASA, MONSANTO, MIT, KOHL'S
	4Paradigm	AutoML platform	Bank of China, PICC, Zhihu

TABLE 9
Some popular open-source research projects on Github (up to Nov. 2018). More stars indicates greater popularity.

Project	stars	Project	stars
TPOT	4326	hyperopt	2302
autokeras	3728	adanet	1802
H2O AutoML	3262	darts	1547
Auto-sklearn	2367	ENAS-pytorch	1297
MOE	1077	Spearmint	1124

step-size for SGD [149], which have shown to be much more efficient than previous state-of-the-arts.

7.2.3 Applications

In this survey, we have focused on the supervised-learning problem. It is also the most considered problem in AutoML, due to the learning performance can be clearly evaluated. However, AutoML can be applied in many other problems in machine learning as well. For example, recently, AutoML approaches have been applied in, e.g., active learning [121], neural network compression [150], and semi-supervised learning (SSL) [151].

While the framework in Section 2.2 and techniques in Sections 4-5 can still be applied in these problems, as different learning tools have to be used, the search space is different. Besides, properties underneath these problems should be further and carefully explored in order to achieve good performance. Taking semi-supervised learning [151] as an example. Existing AutoML for supervised learning can not well address SSL problems. The reason is that SSL introduces new challenges. Specifically, the feature engineering is much harder and the performance is much more sensitive, due to the limited labeled data. Thus, appropriate meta-features (perhaps unsupervised feature is one possible solution) and safeness are crucial for automated SSL to boost the performance upper-bound and lower-bound simultaneously.

7.2.4 Theory

Optimization theory: As shown in Figure 6, the searching process of AutoML can be considered as a black box optimization problem, where the underneath optimization function is measured by the evaluator. While many theories of convergence have been developed for basic techniques in Section 4, e.g., derivative free optimization [84], [109], gradient descent [122], and greedy search [126], it is still not clear how fast they can identify a good configuration. However, convergence speed is a critical problem for AutoML and matters a lot in the choice of techniques for the optimizer, as the evaluation of one configuration usually requires a model training, which is very expensive.

Learning theory: On this topic, we first care about which type of problems can or cannot be addressed by an AutoML approach. In Section 2.1, we have shown that it is not possible for an AutoML approach to achieve good performance on all learning problems due to No Free Lunch theorems stated in [5] and [6]. However, AutoML attempts to obtain good performance across learning tasks and data sets. Thus, it is interesting to figure out which type of learning problems can AutoML deal with and what assumptions can we made on these learning problems.

After determine whether the problem can be addressed by AutoML, for a specific AutoML approach, it is also important to clarify its generalization ability, i.e., "*how much training data is sufficient? what general bounds can be found to relate the confidence of the learned hypotheses to the amount of training experience and characters of learner's hypothesis space?* [1]". This will help us better design AutoML approaches and understand how these approaches can generalize from seen to unseen data. AdaNet [152] is a pioneering work towards this direction, it analyzed the generalization ability of all possible architectures in a NAS problem and used the derived bound to guide a simple optimization technique to search for architecture.

8 Conclusion

Motivated by the academic dream and industrial needs, the automated machine learning (AutoML) has recently became a hot topic. In this survey, we give a systematical review of existing AutoML approaches. We first define what is the AutoML problem and then introduce a basic framework to show how these approaches are realized. We also provide taxonomies of existing works based on "what" and "how" to automate, which acts as a guidance to design new and use old AutoML approaches. We further discuss how existing works can be organized according to our taxonomies in detail. Finally, we briefly review the history of AutoML and show promising future directions. We hope this survey can act as a good guideline for beginners and show light upon future researches.

Acknowledge

First, we would like to thank Prof. Isabelle Guyon (UPSud, INRIA, University Paris-Saclay), Prof. Hugo Jair Escalante (INAOE), and Mr. Zhengyin Liu (INRIA) for their valuable comments on our draft version. Furthermore, we would also like to thank, Prof. Bogdan Gabrys (UTS), Dr. Bo Han (UTS, RIKEN), Mr. Saket Maheshwary (IIIT, Hyderabad),

Ms. Yaqing Wang (HKUST), Mr. Yihui He (CMU), and Mr. Yi-Qi Hu (NJU) for their suggestion on the public version of the survey.

REFERENCES

[1] T. Mitchell, Machine Learning. Springer, 1997.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no. 7587, p. 484, 2016.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[4] W. Xiong, J. Droppo, X. Huang, F. Seide, M. L. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "Toward human parity in conversational speech recognition," IEEE/ACM Transactions on Audio, Speech and Language Processing, vol. 25, no. 12, pp. 2410–2423, 2017.

[5] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," Neural computation, vol. 8, no. 7, pp. 1341–1390, 1996.

[6] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, Apr. 1997.

[7] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in Advances in Neural Information Processing Systems, 2015, pp. 2962–2970.

[8] L. Kotthoff, C. Thornton, H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA," Journal of Machine Learning Research, vol. 18, no. 1, pp. 826–830, 2017.

[9] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in International Conference on Learning Representations, 2017.

[10] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in European Conference on Computer Vision, 2018.

[11] G. Katz, E. C. R. Shin, and D. Song, "Explorekit: Automatic feature generation and selection," in International Conference on Data Mining, 2016, pp. 979–984.

[12] J. M. Kanter and K. Veeramachaneni, "Deep feature synthesis: Towards automating data science endeavors," in IEEE International Conference on Data Science and Advanced Analytics, 2015, pp. 1–10.

[13] J. Friedman, T. Hastie, and R. Tibshirani, The elements of statistical learning. Springer Series in Statistics, 2001.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[15] L. Xie and A. Yuille, "Genetic CNN," in International Conference on Computer Vision. IEEE, 2017, pp. 1388–1397.

[16] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in International Conference on Learning Representations, 2017.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet: classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.

[21] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 2261–2269.

[22] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," in International Conference on Machine Learning, 2017.

[23] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in Conference on Computer Vision and Pattern Recognition, 2017.

[24] S. Huang, X. Li, Z. Cheng, Z. Zhang, and A. G. Hauptmann, "GNAS: A greedy neural architecture search method for multi-attribute learning," in ACM Multimedia, 2018, pp. 2049–2057.

[25] H. Jin, Q. Song, and X. Hu, "Efficient neural architecture search with network morphism," in International Joint Conference on Artificial Intelligence, 2018.

[26] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, "Path-level network transformation for efficient architecture search," in International Conference on Machine Learning, 2018.

[27] M. J. Smith, R. Wedge, and K. Veeramachaneni, "FeatureHub: Towards collaborative data science," in IEEE International Conference on Data Science and Advanced Analytics, 2017, pp. 590–600.

[28] J. Vanschoren, "Understanding machine learning performance with experiment databases," lirias. kuleuven. be, no. May, 2010.

[29] C. Lemke, M. Budka, and B. Gabrys, "Metalearning: a survey of trends and technologies," Artificial Intelligence Review, vol. 44, no. 1, pp. 117–130, 2015.

[30] J. Vanschoren, "Meta learning," 2018. [Online]. Available: https://www.ml4aad.org/wp-content/uploads/2018/09/chapter2-metalearning.pdf

[31] S. J. Pan and Q. Yang, "A survey on transfer learning," IEEE Transactions on Knowledge and Data Engineering, no. 10, pp. 1345–1359, 2009.

[32] M. Feurer and F. Hutter, "Hyperparameter optimization," 2018. [Online]. Available: https://www.ml4aad.org/wp-content/uploads/2018/09/chapter1-hpo.pdf

[33] J.-H. M. Thomas Elsken and F. Hutter, "Neural architecture search," 2018. [Online]. Available: https://www.ml4aad.org/wp-content/uploads/2018/09/chapter3-nas.pdf

[34] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," IEEE Transactions on Image processing, vol. 15, no. 12, pp. 3736–3745, 2006.

[35] M. Guarnieri, "The roots of automation before mechatronics," IEEE Industrial Electronics Magazine, vol. 4, no. 2, pp. 42–43, 2010.

[36] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, Deep learning. MIT Press, 2016.

[37] B. C. Kuo and F. Golnaraghi, Automatic control systems. Prentice-Hall Englewood Cliffs, NJ, 1995.

[38] C. L. Phillips and R. D. Habor, Feedback control systems. Simon & Schuster, 1995.

[39] C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2013, pp. 847–855.

[40] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures," in International Conference Machine Learning, 2013, pp. I–115.

[41] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in International Conference on Machine Learning, 2015, pp. 2113–2122.

[42] E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska, "Automating model search for large scale machine learning," in ACM Symposium on Cloud Computing, 2015, pp. 368–380.

[43] J. N. van Rijn, S. M. Abdulrahman, P. Brazdil, and J. Vanschoren, "Fast algorithm selection using learning curves," in International symposium on intelligent data analysis. Springer, 2015, pp. 298–309.

[44] Y.-F. Li, H. Wang, T. Wei, and W.-W. Tu, "Derivative-free optimization via classification," in AAAI Conference on Artificial Intelligence, vol. 16, 2016, pp. 2286–2292.

[45] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: one-shot model architecture search through hypernetworks," in International Conference on Learning Representations, 2018.

[46] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," arXiv preprint arXiv:1806.09055, 2018.

[47] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," Journal of Machine Learning Research, vol. 13, no. Feb, pp. 281–305, 2012.

[48] A. G. Baydin, B. A. Pearlmutter, A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," Journal of Machine Learning Research, vol. 18, pp. 1–43, 2017.

[49] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," Artificial Intelligence Review, vol. 18, no. 2, pp. 77–95, 2002.

[50] M. G. Smith and L. Bull, "Genetic programming with a genetic algorithm for feature construction and selection," Genetic Programming and Evolvable Machines, vol. 6, no. 3, pp. 265–281, 2005.

[51] B. Tran, B. Xue, and M. Zhang, "Genetic programming for feature construction and selection in classification on high-dimensional data," Memetic Computing, vol. 8, no. 1, pp. 3–15, 2016.

[52] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, and D. Turaga, "Learning feature engineering for classification," in International Joint Conference on Artificial Intelligence, vol. 17, 2017, pp. 2529–2535.

[53] H. J. Escalante, M. Montes, and L. E. Sucar, "Particle swarm model selection," Journal of Machine Learning Research, vol. 10, no. Feb, pp. 405–440, 2009.

[54] V. Calcagno and C. de Mazancourt, "glmulti: An R package for easy automated model selection with (generalized) linear models," Journal of Statistical Software, vol. 34, no. i12, 2010.

[55] C. J. Merz, "Dynamical selection of learning algorithms," in Learning from Data. Springer, 1996, pp. 281–290.

[56] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann, "Algorithm selection and scheduling," in International Conference on Principles and Practice of Constraint Programming, 2011, pp. 454–469.

[57] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," Artificial Intelligence, vol. 206, pp. 79–111, 2014.

[58] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, and K. Tierney, "ASlib: A benchmark library for algorithm selection," Artificial Intelligence, vol. 237, pp. 41–58, 2016.

[59] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in International Joint Conference on Artificial Intelligence, 2015.

[60] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," in International Conference on Learning Representations, 2016.

[61] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in Workshop on Automatic Machine Learning, 2016, pp. 58–65.

[62] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, "Pathnet: Evolution channels gradient descent in super neural networks," arXiv preprint arXiv:1701.08734, 2017.

[63] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," arXiv preprint arXiv:1711.04528, 2017.

[64] Z. Zhong, J. Yan, and C.-L. Liu, "Practical block-wise neural network architecture generation," in IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[65] B. Deng, J. Yan, and D. Lin, "Peephole: Predicting network performance before training," arXiv preprint arXiv:1712.03351, 2017.

[66] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in AAAI Conference on Artificial Intelligence, 2018.

[67] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 8, pp. 1798–1828, 2013.

[68] K. Pearson, "On lines and planes of closest fit to systems of points in space," The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, vol. 2, no. 11, pp. 559–572, 1901.

[69] R. Fisher, "The use of multiple measurements in taxonomic problems," Annals of Eugenics, vol. 7, no. 2, pp. 179–188, 1936.

[70] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in International Conference on Machine learning, 2008, pp. 1096–1103.

[71] H. T. Lam, J.-M. Thiebaut, M. Sinn, B. Chen, T. Mai, and O. Alkan, "One button machine for automating feature engineering in relational databases," arXiv preprint arXiv:1706.00327, 2017.

[72] A. Kaul, S. Maheshwary, and V. Pudi, "Autolearnautomated feature generation and selection," in IEEE International Conference on Data Mining, 2017, pp. 217–226.

[73] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in IEEE Conference on Computer Vision and Pattern Recognition, 2010, pp. 2528–2535.

[74] K. Yu, T. Zhang, and Y. Gong, "Nonlinear learning using local coordinate coding," in Advances in Neural Information Processing Systems, 2009, pp. 2223–2231.

[75] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in Advances in Neural Information Processing Systems, 2008, pp. 161–168.

[76] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in International Conference on International Conference on Machine Learning, 2011, pp. 265–272.

[77] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," Mathematical Programming, vol. 45, no. 1-3, pp. 503–528, 1989.

[78] M. M. Salvador, M. Budka, and B. Gabrys, "Automatic composition and optimization of multicomponent predictive systems with an extended auto-weka," IEEE Transactions on Automation Science and Engineering, pp. 1–14, 2018.

[79] R. Negrinho and G. Gordon, "Deeparchitect: Automatically designing and training deep architectures," 2018.

[80] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[81] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in Advances in Neural Information Processing Systems, 2014, pp. 3320–3328.

[82] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[83] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in International Conference on Artificial Intelligence and Statistics, 2011, pp. 315–323.

[84] A. R. Conn, K. Scheinberg, and L. N. Vicente, Introduction to derivative-free optimization. SIAM, 2009, vol. 8.

[85] T. Bck, Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford Univ. Pr, 1998.

[86] H. Vafaie and K. De Jong, "Genetic algorithms as a tool for feature selection in machine learning," in International Conference on Tools with Artificial Intelligence, 1992, pp. 200–203.

[87] W. Tackett, "Genetic programming for feature discovery and image discrimination," in International Conference on Genetic Algorithms, 1993.

[88] W. Siedlecki and J. Sklansky, "A note on genetic algorithms for large-scale feature selection," in Handbook of Pattern Recognition and Computer Vision. World Scientific, 1993, pp. 88–107.

[89] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, "Evaluation of a tree-based pipeline optimization tool for automating data science," in Genetic and Evolutionary Computation Conference. ACM, 2016, pp. 485–492.

[90] R. S. Olson and J. H. Moore, "TPOT: A tree-based pipeline optimization tool for automating machine learning," in Workshop on Automatic Machine Learning, 2016, pp. 66–74.

[91] S.-W. Lin, K.-C. Ying, S.-C. Chen, and Z.-J. Lee, "Particle swarm optimization for parameter determination and feature selection of support vector machines," Expert Systems with Applications, vol. 35, no. 4, pp. 1817–1824, 2008.

[92] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor, "Particle swarm optimization for hyper-parameter selection in deep neural networks," in Genetic and Evolutionary Computation Conference. ACM, 2017, pp. 481–488.

[93] X. Yao, "Evolving artificial neural networks," Proceedings of the IEEE, vol. 87, no. 9, pp. 1423–1447, 1999.

[94] C. Zhang, H. Shao, and Y. Li, "Particle swarm optimisation for evolving artificial neural network," in IEEE International Conference on Systems, Man, and Cybernetics, vol. 4, 2000, pp. 2487–2490.

[95] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," Evolutionary Computation, vol. 10, no. 2, pp. 99–127, 2002.

[96] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in International Conference on Machine Learning, 2017.

[97] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," arXiv preprint arXiv:1802.01548, 2018.

[98] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in International Conference on Learning Representations, 2018.

[99] K. Swersky, J. Snoek, and R. P. Adams, "Freeze-thaw bayesian optimization," arXiv preprint arXiv:1406.3896, 2014.

[100] T. Nickson, M. A. Osborne, S. Reece, and S. J. Roberts, "Automated machine learning on big data using stochastic algorithm tuning," arXiv preprint arXiv:1407.7969, 2014.

[101] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets," in International Conference on Artificial Intelligence and Statistics, 2016.

[102] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in Advances in neural information processing systems, 2012, pp. 2951–2959.

[103] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, "Scalable gaussian process-based transfer surrogates for hyperparameter optimization," Machine Learning, vol. 107, no. 1, pp. 43–78, 2018.

[104] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in Advances in Neural Information Processing Systems, 2011, pp. 2546–2554.

[105] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in International Conference on Learning and Intelligent Optimization, 2011, pp. 507–523.

[106] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams, "Scalable bayesian optimization using deep neural networks," in International Conference on Machine Learning, 2015, pp. 2171–2180.

[107] K. Kandasamy, K. R. Vysyaraju, W. Neiswanger, B. Paria, C. R. Collins, J. Schneider, B. Póczos, and E. P. Xing, "Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly," ArXiv, vol. abs/1903.06694, 2019.

[108] Y.-Q. Hu, H. Qian, and Y. Yu, "Sequential classification-based optimization for direct policy search," in AAAI Conference on Artificial Intelligence, 2017, pp. 2029–2035.

[109] T. B. Hashimoto, S. Yadlowsky, and J. C. Duchi, "Derivative free optimization via repeated classification," International Conference on Artificial Intelligence and Statistics, 2018.

[110] R. Munos, "Optimistic optimization of a deterministic function without the knowledge of its smoothness," in Advances in Neural Information Processing Systems, 2011, pp. 783–791.

[111] M. Valko, A. Carpentier, and R. Munos, "Stochastic simultaneous optimistic optimization," in International Conference on Machine Learning, 2013, pp. 19–27.

[112] C. Andrieu, N. De Freitas, and A. Doucet, "Sequential mcmc for bayesian model selection," in IEEE Signal Processing Workshop on Higher-Order Statistics, 1999, pp. 130–134.

[113] K. Swersky, J. Snoek, and R. P. Adams, "Multi-task bayesian optimization," in Advances in Neural Information Processing Systems, 2013, pp. 2004–2012.

[114] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. Cox, "Hyperopt: a python library for model selection and hyperparameter optimization," Computational Science & Discovery, vol. 8, no. 1, p. 014008, 2015.

[115] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 1998.

[116] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Faster discovery of neural architectures by searching for paths in a large model," in International Conference on Learning Representations, 2018.

[117] J.-M. Pérez-Rúa, M. Baccouche, and S. Pateux, "Efficient progressive neural architecture search," in British Machine Vision Conference, 2018.

[118] A. György and L. Kocsis, "Efficient multi-start strategies for local search algorithms," Journal of Artificial Intelligence Research, vol. 41, pp. 407–444, 2011.

[119] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," Journal of Machine Learning Research, vol. 18, no. 1, pp. 6765–6816, 2017.

[120] R. Gaudel and M. Sebag, "Feature selection as a one-player game," in International Conference on Machine Learning, 2010, pp. 359–366.

[121] M. Fang, Y. Li, and T. Cohn, "Learning how to active learn: A deep reinforcement learning approach," in Conference on Empirical Methods in Natural Language Processing, 2017, pp. 595–605.

[122] Y. Bengio, "Gradient-based optimization of hyperparameters," Neural Computation, vol. 12, no. 8, pp. 1889–1900, 2000.

[123] F. Pedregosa, "Hyperparameter optimization with approximate gradient," in International Conference on Machine Learning, 2016, pp. 737–746.

[124] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, "Forward and reverse gradient-based hyperparameter optimization," in International Conference on Machine Learning, 2017, pp. 1165–1173.

[125] J. A. Tropp, "Greed is good: Algorithmic results for sparse approximation," IEEE Transactions on Information theory, vol. 50, no. 10, pp. 2231–2242, 2004.

[126] F. Bach et al., "Learning with submodular functions: A convex optimization perspective," Foundations and Trends® in Machine Learning, vol. 6, no. 2-3, pp. 145–373, 2013.

[127] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in Advances in Neural Information Processing Systems, 2018, pp. 2546–2554.

[128] Y.-Q. Hu, Y. Yu, W.-W. Tu, Q. Yang, Y. Chen, and W. Dai, "Multi-fidelity automatic hyper-parameter tuning via transfer series expansion," in AAAI Conference on Artificial Intelligence, vol. 16, 2019, pp. 2286–2292.

[129] O. Maron and A. W. Moore, "Hoeffding races: Accelerating model selection search for classification and function approximation," in Advances in Neural Information Processing Systems, 1994, pp. 59–66.

[130] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with bayesian neural networks," in International Conference on Learning Representations, 2016.

[131] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in Advances in Neural Information Processing Systems, 2007, pp. 153–160.

[132] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in International Conference on Machine Learning, 2013, pp. 1139–1147.

[133] K. Eggensperger, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Efficient benchmarking of hyperparameter optimizers via surrogates," in AAAI Conference on Artificial Intelligence, 2015.

[134] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," in International Conference on Learning Representations, 2018.

[135] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine Learning, vol. 8, no. 3-4, pp. 229–256, 1992.

[136] "ICML-2014 AutoML workshop," 2014. [Online]. Available: https://sites.google.com/site/automlwsicml14/

[137] "ICML-2015 AutoML workshop," 2015. [Online]. Available: https://sites.google.com/site/automlwsicml15/

[138] "ICML-2016 AutoML workshop," 2016. [Online]. Available: https://sites.google.com/site/automl2016/

[139] "ICML-2017 AutoML workshop," 2017. [Online]. Available: https://sites.google.com/site/automl2017icml/

[140] "ICML-2018 AutoML workshop," 2018. [Online]. Available: https://www.4paradigm.com/competition/pakdd2018

[141] "PAKDD 2018 data competition: Automatic machine learning challenge 2018," 2018. [Online]. Available: https://sites.google.com/site/automl2018icml/

[142] "NIPS 2018 challenge: AutoML for lifelong machine learning," 2018. [Online]. Available: https://sites.google.com/site/automl2018icml/

[143] "ICML-2018 workshop on human interpretability in machine learning," 2018. [Online]. Available: https://sites.google.com/view/whi2018/home

[144] "ICML-2018 workshop on privacy in machine learning and artificial intelligence," 2018. [Online]. Available: https://pimlai.github.io/pimlai18/

[145] D. G. Lowe, "Object recognition from local scale-invariant features," in IEEE International Conference on Computer Vision, vol. 2, 1999, pp. 1150–1157.

[146] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, 2005, pp. 886–893.

[147] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," Communications of the ACM, vol. 18, no. 11, pp. 613–620, 1975.

[148] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, and K. Simonyan, "Population based training of neural networks," arXiv preprint arXiv:1711.09846, 2017.

[149] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood, "Online learning rate adaptation with hypergradient descent," arXiv preprint arXiv:1703.04782, 2017.

[150] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in European Conference on Computer Vision, 2018, pp. 784–800.

[151] Y.-F. Li, H. Wang, T. Wei, and W.-W. Tu, "Towards automated semi-supervised learning," in AAAI Conference on Artificial Intelligence, 2019.

[152] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang, "AdaNet: Adaptive structural learning of artificial neural networks," in International Conference on Machine Learning, 2017, pp. 874–883.