DarwinML: A Graph-based Evolutionary Algorithm for Automated Machine Learning

Fei Qi, Zhaohui Xia, Gaoyang Tang, Hang Yang, Yu Song, Guangru Qian, Xiong An, Chunhuan Lin, Guangming Shi

Abstract—As an emerging field, Automated Machine Learning (AutoML) aims to reduce or eliminate manual operations that require expertise in machine learning. In this paper, a graph-based architecture is employed to represent flexible combinations of ML models, which provides a large searching space compared to tree-based and stacking-based architectures. Based on this, an evolutionary algorithm is proposed to search for the best architecture, where the mutation and heredity operators are the key for architecture evolution. With Bayesian hyper-parameter optimization, the proposed approach can automate the workflow of machine learning. On the PMLB dataset, the proposed approach shows the state-of-the-art performance compared with TPOT, Autostacker, and auto-sklearn. Some of the optimized models are with complex structures which are difficult to obtain in manual design.

I. INTRODUCTION

Various models have been thoroughly investigated by the machine learning (ML) community. In theory, these models are general and applicable to both academia and industry. However, it could be time-consuming to build a solution on a specific ML task, even for a ML expert. To remedy this, Automated ML (AutoML) [1] emerges to minimize the design complexity of a complete ML solution, which usually includes data preprocessing, feature engineering, model selection and ensemble, fine-tuning of hyperparameters, etc.

Research of AutoML starts from hyperparameter optimization, which includes random search [2], evolutionary algorithm (EA) [3], and Bayesian [4] approaches. Automatic feature engineering [5], [6] is another important sub-field in AutoML.

In deep learning, the end-to-end scheme [7] provides a solution for automated learning to a certain degree. However, to train a deep end-to-end neural network, it generally requires large scale labeled dataset, which is unavailable in many practical problems. In addition, Glasmachers has shown other limitations of the end-to-end scheme [8]. At the mean while, neural architecture search approaches have emerged [9], [10] to automate neural network design.

Though deep learning is very powerful, there is still room to apply traditional ML. To achieve the goal of AutoML, Thornton et al. [11], Thakur and Krohn-Grimberghe [12], and Feurer et al. [13] proposed solutions for automating the entire process for traditional ML problems. These solutions are not powerful enough, because they use simple combinations or single model selection. Employing EA for architecture search, Tree-based Pipeline Optimization Tool [14], or TPOT, uses a tree-based representation for model combination; while Autostacker [15] uses the stacking scheme.

According to a comparison between tree-based [16] and graph-based [17] representations for symbolic regression [18], a graph-based representation is much flexible and general than the tree-based one. In addition, the graph-based representation for computation are widely used in modern ML frameworks including TensorFlow [19].

In this paper, we propose a novel AutoML solution called DarwinML based on the EA with tournament selection [20]. DarwinML employs the directed acyclic graph (DAG) for model combination. Compared to existing AutoML methods such as pipeline, the proposed method is with a flexible representation and is highly extensible. In summary, the key contributions of this paper are as follows.

- The adjacency matrix of graph is analyzed and used to represent the architecture composed by a series of traditional ML models.
- Several evolutionary operators have been defined and implemented to generate diverse architectures, which has not been thoroughly investigated in existing works.
- Based on EA, an end-to-end automatic solution called “DarwinML” is proposed to search optimal composition of traditional ML models.

The rest of this paper is organized as follows. Section II reviews literature related to AutoML. Section III introduces the architectural representation for ML model composition. Section IV presents the approach for evolutionary architecture search and optimization. Section V illustrates experimental results on the PMLB [21] dataset. Section VI concludes this work.

II. RELATED WORKS

A. Automatic Machine Learning

The earlier AutoML models were born in the competition called “ChaLearn AutoML Challenge” [22] in 2015. According to a test on 30 datasets, the two top ranking models of the challenge were designed by Intel’s team and Hutter’s team, respectively. Intel’s proprietary solution [1] is a tree-based method. Auto-sklearn [13], the open-source solution from Hutter’s team, won the challenge. Hutter also
co-developed SMAC [23] and Auto-WEKA [11], [24].

**B. Evolutionary Algorithms**

EA has many variants in implementation such as evolution programming, evolution strategy, genetic algorithm, etc. Encoding problem to different representation that easier to evolve is an efficient way to solution. Genetic Programming (GP), as a special type of EA, encodes computer programs as a set of genes to evolve to the best. In recent years, inspired by its success in generating structured representations [26], more implementations [27], [28] of GP has been studied. One topic is using mathematical functions as coding units to search for appropriate composite function structures. Such as graph-based representations encoded by GP [20], [29], both approaches use GP to evolve a better function to approximate target value.

In AutoML, EAs are used frequently for architecture search. TPOT [30], a later AutoML approach which outperforms previous systems, searches tree-based pipeline of models with GP. TPOT allows dataset to be processed parallelly with multiple preprocessing methods. Such a step can bring diverse features to the following steps. Furthermore, Kordik et al. [31] and de Sa et al. [32] not only use tree-based GP to ensemble ML methods, but also use a grammar to enforce the generation of valid solution. The larger search space and detailed constraints can make solution more a successful method. Recently, Chen et al. proposed Autostacker [15] to combine ML classifiers by stacking [33] and search for the best model using EA. This method largely extends the possibilities of the model combination.

III. ARCHITECTURE REPRESENTATION

A. Graph-based model combination

A set of ML models, \( \mathcal{M} = \{f_1, \cdots, f_M\} \), is chosen to construct the composite model, where \( f_m \) denotes the hypothesis function of the \( m \)-th model. A directed acyclic graph (DAG) is employed to denote the architecture of a combination of models. In the DAG, edges and vertices denote data flow and computational functions, respectively. This is the same to the representation used in TensorFlow [19]. One difference is that the structure of DAG will be changed after evolutionary operations.

Let graph \( \mathcal{G} = \{V, E\} \) be the DAG denoting the combined model, which is composed of a set of vertices \( V = \{v_1, \cdots, v_K\} \) and a set of edges \( E = \{e_{ij}\} \). A vertex \( v_k \) is associated with a computational function from the model set \( \mathcal{M} \). For simplicity, the model \( f_{m_k} \) denotes the function associated with the vertex \( v_k \) in the rest of this paper. An edge \( e_{ij} \) implies that the output from vertex \( v_i \) flows into the vertex \( v_j \).

Let \( z_j \) denote the output of vertex \( v_j \), and \( z_0 \) denote the input of the composite model. Suppose vertex \( v_j \) has \( n \) input vertices, \( v_{i_1}, v_{i_2}, \cdots, v_{i_n} \), the function associated to vertex \( v_j \) is then computed as:

\[
    z_j = f_{m_j} \left( U(z_{i_1}, z_{i_2}, \cdots, z_{i_n}) \right),
\]

where \( U(\cdot) \) is a feature union function. In the proposed approach, vector concatenation is used as the feature union function. The output \( z_k \) of the composite model can be computed by recursively applying (1).

B. Layer

For the convenience of understanding and applying some evolutionary operations, vertices are topologically sorted [34]. For simplicity, the topological order of a vertex is defined as its depth. Vertices with a same depth \( d \) compose a layer \( L_d \):

\[
    L_d = \{v_k \mid d_k = d, \forall v_k \in V\}. \tag{2}
\]

It should be noted that there are no edges between two vertices belong to a same layer. Without loss of generality, vertices \( v_1 \) and \( v_K \) are supposed to be the input and the output of the graph, respectively. Accordingly, \( d_1 \) and \( d_Z \) are with the minimum and maximum depths.

The connection probability of a particular edge \( e_{ij} \) is useful for some evolutionary operations, which is related to the depth of the two layers containing vertices \( v_i \) and \( v_j \):

\[
    p(e_{ij}) = p_0 \exp \left( \gamma(d_i - d_j + 1) \right), \tag{3}
\]

where \( \gamma \) is the decay factor, and \( p_0 \) is an initial connection probability.

C. Layer blocks

The adjacency matrix, \( A \), of a graph, \( \mathcal{G} \), is a \( K \times K \) matrix with elements given by:

\[
    A_{ij} = \begin{cases} 
    1 & \text{if } e_{ij} \in E, \\ 
    0 & \text{else} \end{cases}, \tag{4}
\]

where one indicates an edge from vertex \( v_i \) to vertex \( v_j \). As the graph is directed acyclic and topologically sorted, the adjacency matrix is always upper triangular. As shown in Fig. 1, the diagonal of the adjacency matrix are square blocks full of zeros:

\[
    A_{ij}^{(d)} = 0 \quad \forall v_i, v_j \in L_d, \tag{5}
\]

where \( A^{(d)} \) is a \( |L_d| \times |L_d| \) block corresponds to the layer \( L_d \). The rest of the upper-triangle in the adjacency matrix can be split into several blocks \( A_{ij}^{(d,d')} \) according to the fragmenting of layers, where \( v_i \in L_d \) and \( v_j \in L_d' \). The size of the block \( A^{(d,d')} \) is \( |L_d| \times |L_d'| \).

The connection probability can be visualized by mapping onto the adjacency matrix, as shown in Fig. 1. According
is the parameter endowed by the graph, considering blocks $A$ individual architecture will also be called an following sub-sections. Following the convention in EA, an algorithm. These operations are designed to generate diverse heredity, and keep best, constitute the main part of the algorithm. In Fig. 2, Four operations, which are the random, mutation, selection [35] is employed to explore for architectures in the complex configuration space. The work flow is illustrated in Fig. 2. Four operations, which are the random, mutation, heredity operation are produced from (3), each block $A^{(d,d')}$ has a same connection probability which can be calculated as:

$$p(e_{ij}|v_i \in L_d, v_j \in L_{d'}) = p_0 \exp \left( \gamma (d - d' + 1) \right).$$

With above intuitive observation, changing layer $L_d$ should consider blocks $A^{(d)}$, $A^{(d,\cdot)}$, and $A^{(\cdot,d)}$, which form a cross-shaped area as shown in Fig. 1.

IV. EVOLUTION OF ARCHITECTURES

Our goal is to find the architecture that performs best on a given ML task. In this paper, only classification tasks are considered and evaluated. Let $X = \{x_1, x_2, \cdots, x_N\}$ and $Y = \{y_1, y_2, \cdots, y_N\}$ be the features and labels of a dataset, respectively. The optimization goal can then be formally expressed as:

$$G^*, \Theta^* = \arg \min_{G, \Theta} \ell(G, \Theta, X, Y) + \alpha c(G),$$

where $G$ denotes the architecture represented by a DAG, and $\Theta$ is the parameter endowed by the graph, $\ell(\cdot)$ is the loss function of defined by the classification task, $c(\cdot)$ measures the complexity of the graph, and $\alpha$ is a coefficient to trade-off between loss and complexity. With the complexity term, the objective function prefers a simple architecture. In implementation, the loss function includes a regularization term and some cross-measures the complexity of the graph, and $\alpha$ is a coefficient to trade-off between loss and complexity. With the complexity term, the objective function prefers a simple architecture. In implementation, the loss function includes a regularization term and some cross-validation strategies to avoid over-fitting, and the complexity counts the number of vertices and edges in the graph.

A. Search Algorithm

The evolutionary search algorithm based on tournament selection [35] is employed to explore for architectures in the complex configuration space. The work flow is illustrated in Fig. 2. Four operations, which are the random, mutation, heredity, and keep best, constitute the main part of the algorithm. These operations are designed to generate diverse architectures. Details of these operations will be explained in following sub-sections. Following the convention in EA, an architecture will also be called an individual in the following.

In initialization, the first generation is build up by individuals according to the convention in EA, generated by the random operation.

In the second generations, individuals evolve by applying all four operations. Firstly, the keep best operator is applied to get top 15% best individual set from first generation. Then, tournament selection [35] picks the top model from a randomly chosen sub-group in previous generation and keep best set. A number of top models are selected by repeating random grouping and tournament selection. Secondly, the mutation and heredity operations are applied to these promising individuals, and new individuals are created for the current generation. Then, the top 15% best individuals are inherent back in the population to ensure models with good performance are reserved. After that, individuals are generated by the random operation to fulfill the current generation. Finally, fitness of all individuals, except the ones inherent from keep best, are evaluated after a training procedure.

In the subsequent generation, $i$-th generation applied by mutation and heredity operation are produced from $(i - 1)$-th generation and top 15% best individuals in $(i - 2)$-th generation.

In every generation, the percentage of individuals generated from random, heredity, and mutation operations are about 30%, 40%, and 30%, respectively. A higher heredity probability shows a better performance in tasks fit for complex architectures. Individuals with invalid graph structures are simply dropped no matter how the individual is generated. The rules will be explained in Implementation Details. In addition, the search algorithm prefers simple architectures according to the fitness in (7).

The evolution stops when a predefined duration or number of population has been reached. The final output is the individual with the highest fitness.

B. Evolutionary Operations

1) Random Operation: The random operation is designed to randomly sample individuals in the configuration or searching space. The pseudo code of the random operation is shown in Algorithm 1. In the algorithm, $K$ and $D$ are the numbers of vertices and layers of the DAG to be generated as an individual. The constant $C$ serves as an upper bound to control the size of DAG. The sizes of layers, $l_d$, should follow a constraint that $\sum_d l_d = K$, where $l_1 = l_D = 1$ applies since the DAG is always implemented as single-in-single-output. The predefined threshold $\rho$ controls the density of edges.

2) Mutation Operations: Three mutation operations are designed to provide flexible way to vary individual architectures for the purpose of traversing to better individuals.

a) Vertex mutation: This operation is designed to replace one vertex with another ML model. Its implementation is as straightforward as shown by Algorithm 2. A real example of this operation is shown in Fig. 3, where the vertex model “SVC” is replaced by a “ridge classifier”.

b) Edge mutation: This operation, as given by Algorithm 3, is defined to flip an edge connection in the DAG.

c) Layer mutation: Local structures could affect the performance of the architecture. Layers are naturally such local structures. Thus, the layer mutation is introduced to change the DAG at a scale larger than vertex and edge. The pseudo
Random Operation
Mutation Operation
Heredity Operation
Keep Best
Evolutionary Algorithm
...
Initial Graph
Envoled Graph
First Generation
Last Generation
Tournament Selection

Fig. 2: DarwinML framework. Evolutionary Algorithm (EA) using gene operations do searching job in the form of decoded matrices, and do training to get fitness in the form of encoded graph. After training, EA save best 15% models to next generation. EA keep searching and training until the population is enough. The best model can be chosen in population by fitness as well as loss function in machine learning.

Algorithm 1 Random Operation
1: $K \leftarrow \text{randint} \triangleright$ Number of vertices ($1 < K < C$)
2: Randomly choose $K$ functions from the model set $M$ and assign them to each vertex.
3: $D \leftarrow \text{randint}() \triangleright$ Number of layers ($D < K$).
4: $l_2, \ldots, l_{D-1} \leftarrow \text{randint}() \triangleright$ Size of each layer.
5: $A \leftarrow 0 \triangleright$ Initialize the adjacency matrix.
6: for $1 < d < d' < D$ do
7: for $v_i \in L_d, v_j \in L_{d'}$ do
8: $p_{ij} \leftarrow \text{rand}()$ according to (6). $\triangleright$ $0 \leq p_{ij} \leq 1$.
9: $A_{ij}^{(d,d')} \leftarrow 1$ If $p_{ij} > \rho \triangleright$ Determine edge $e_{ij}$.
10: end for
11: end for
12: Perform topological sort.

Algorithm 2 Vertex Mutation
1: $k \leftarrow \text{randint}() \triangleright$ Select vertex $v_k$ to change.
2: $m'_k \leftarrow \text{randint}() \triangleright$ Choose a ML model from $M$.
3: $f_{m_k} \leftarrow f_{m'_k} \triangleright$ Replace the model for vertex $v_k$.

Algorithm 3 Edge Mutation
1: $i,j \leftarrow \text{randint}() \triangleright$ Select two vertices $v_i$ and $v_j$.
2: $A_{ij} \leftarrow 1 - A_{ij} \triangleright$ Flip the connection of edge $e_{ij}$.
3: Perform topological sort.

Fig. 3: An example of vertex mutation.

code is illustrated in Algorithm 4. An example is shown in Fig. 4, where a layer composed by three vertices, which are the “KNeighbors Regressor”, “SVC”, and “Ridge Classifier”, was inserted to the DAG. The dashed edge was automatically removed after insertion. Please note that edges crossing layers are permitted but not available in this example.

3) Heredity Operation: Heredity operation is an enhancement on dealing with local structures. This operation provides an opportunity for an individual to replace a bad layer with a good one. In this sense, heredity plays a very important role to inherent and broadcast the good local structures to the whole population. Different to previous operations, heredity requires two good individuals to perform the operation, as given by Algorithm 5.

One example is illustrated in Fig. 5, the layer with one vertex, “Ridge Classifier”, has been inserted after the removal of the layer containing two vertices, “Normalizer” and “Robust Scaler”. With the heredity operation, the balanced accuracy of the architecture increased to 0.778 from 0.723. The example shows the effectiveness of this operation.

4) Keep Best: This operation is used to keep the best individual in previous generation to avoid losing the best one
in following generations. In implementation, 15% individuals with top performances are treated as best ones and kept from one generation to the next.

C. Implementation Details

1) Graph Validation: As the edges are generated by the evolutionary operations according to a random distribution, there are possibilities that the generated individual is not a valid DAG. So validation check should be performed after each operation, and the invalid individual is dropped. There is a maximum retrieval number for these operations. So the percentages of individuals from each operation may differ from the configuration. One rule for validation check is that vertices in a DAG should have both input and output edges except the first and last vertex. Another rule may be applied to promote the ratio of individuals with good performance, which is that classifier/regression vertices should not directly connect to another classifier/regression vertex alone. Also, there are other constraints like maximum vertex number will be shown in the experiment section.

2) Model Mix: After building a DAG with machine learning models, we have some problems while the graph ensembles machine learning methods different from the task. If the task is classifier type and there is regression model in DAG, we calculate root-mean-square error between regression prediction and classifier label as regression loss function to train model. If unsupervised model in the graph, we do unsupervised learning on dataset and output a vector as feature to next vertex. For example, the output of k-means is a one-of-k coded vector, indicating the cluster index. So the output of k-means can be used by its following vertex for model combination (stacking).

3) Hyperparameter Optimization: After searching models by the EA with tournament selection, BHO [4] is applied to the top five individuals in the final population. Due to the limitation of computing resources, hyperparameter optimization is very expensive to apply to the whole population. The individuals find by algorithm is competitive, and hyperparameter optimization could further improve their performances.

V. Experiments

To show the performance of DarwinML, its robustness and accuracy were evaluated. Results were compared with random
Datasets #Instances #Features #Classes
monk1 556 6 2
parity5 32 5 2
parity5+5 1124 10 2
pima 768 8 2
prnn crabs 200 7 2
allhypo 3770 29 3
spect 267 22 2
vehicle 846 18 5
wine-recognition 286 9 2
cars 392 8 3
Hill Valley with noise 1212 100 2
ecoli 327 7 8
heart-h 3772 29 2

TABLE I: Datasets used in experiments

A. Datasets of PMLB

DarwinML was tested on the same datasets used in Autostacker [15], where 15 datasets were selected from PMLB [21]. PMLB is a benchmark dataset that include hundreds of public datasets which mainly resources from OpenML [36] and UCI [37]. These 15 datasets present various domains in PMLB that can test DarwinML’s classification performance on both binary and multiple-class tasks. Table I shows the detail of these datasets. For each dataset, samples were shuffled and divided into training (80%) and testing (20%) sets. The training set uses cross validation.

As listed in Table II, two tests with different sizes of population, which have 120 and 400 individuals to be evaluated in total, were run to compare the performance with different search spaces. The corresponding numbers of generation were set to 10 and 20, respectively. Results from the two population sizes were named “DML120” and “DML400”, respectively. In addition, ranges of numbers of vertices and layers were set to constrain the search space, as done in Kordik et al. [31] and de Sa et al. [32]. The number of retrials is the upper limit of the failures in trying to apply evolutionary operators to a graph. If this limit is reached, the individual will be dropped according to sub-section IV-C1. The max training time is the upper limit of the training time can be used for one graph. If time is out, graph will be dropped as a failure graph. The parameters $p_0$ and $\gamma$ in (3) are set to 0.3 and 1, respectively. The parameter $C$ in random operation is set to 10.

Results named “DML400+BHO” are obtained by applying BHO [4] to the top five best graphs in the population with 400 individuals. With Bayesian optimization, we tuned up to 40 parameter sets. According to Autostacker [15] and TPOT [30], the balanced accuracy [38] is employed to measure the performance for a fair comparison. On each dataset, DarwinML were repeated 10 times with random initialization, the mean and variances of the balanced accuracy are calculated. Results of other AutoML methods are collected from the paper of Autostacker [15], which is produced with a same setting.

B. Comparison

Results were listed in Table III. Random forest is chosen to be the baseline. Being EA based AutoML models, TPOT [30] and Autostacker [15] evolved for 100 and 3 generations, respectively. Both models include hyperparameter optimization, which improves the performance dramatically. In testing DarwinML, hyperparameter optimization was separated for a detailed comparison. DML400 achieves better performance when compared with random forest on 14 datasets. Its accuracy is comparable or superior to TPOT on 11 datasets, and is comparable or superior to Autostacker and auto-sklearn 9 datasets. Although other models except random forest include hyperparameter optimization, DML400 is better than them because DarwinML’s inherent mechanism provides flexible model combination which enlarges the architectural search space. Taking BHO as a post-processing step on DML400, DarwinML far exceeds other models in mean balanced accuracy on most datasets.

Fig. 6 shows the scatter plot of accuracy of each individual on the dataset allhypo, which is one run of DML400. In the plot, the color of each point represents how it is generated. Some individuals run out of the training time and were dropped. Their loss were set to infinity and were excluded.
In this paper, an evolutionary algorithm is proposed to search for the best architecture composed of traditional machine learning models with a graph-based representation. Based on the representation, the random, mutation, and heredity operators are defined and implemented. Evolutionary algorithm is then employed to optimize the architecture. Diversity makes the success of EAs. The evolutionary operations proposed in the paper enables the search of diverse architectures. With Bayesian hyperparameter optimization applied to the best results of the proposed evolutionary method, the proposed approach demonstrates the state-of-the-art performance on the PMLB dataset compared to TPOT, auto-stacker, and autosklearn.

Though the proposed approach is implemented and tested on the set of traditional machine learning models, there are no inherent limitations that it applies only on these models. In the future, we plan to generalize the approach to neural architecture search by extending the ML models with building blocks of neural networks to get better performance on large scale datasets.

### REFERENCES

[1] I. Guyon, I. Chaabane, H. J. Escalante, S. Escalera, D. Jajetic, J. R. Lloyd, N. Macià, B. Ray, L. Romaszko, M. Sebag, A. Statnikov, S. Treguer, and E. Viegas, “A brief Review of the ChaLearn AutoML Challenge: Any-time Any-dataset Learning without Human Intervention,” in Proceedings of the Workshop on Automatic Machine Learning, ser. PMLR, F. Hutter, L. Kothoff, and J. Vanschoren, Eds., vol. 64, New York, New York, USA, 2016, pp. 21–30.

[2] J. Bergstra and Y. Bengio, “Random Search for Hyper-Parameter Optimization,” J. Mach. Learn. Res., vol. 13, no. Feb, pp. 281–305, 2012.

[3] F. Friedrichs and C. Igel, “Evolutionary tuning of multiple SVM parameters,” Neurocomputing, vol. 64, pp. 107–117, Mar. 2005.

[4] E. Brochu, V. M. Cora, and N. de Freitas, “A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning,” arXiv:1012.2599, Dec. 2010.
