APENAS: An Asynchronous Parallel Evolution Based Multi-objective Neural Architecture Search

Mengtao Hu
School of Data Science and Engineering
East China Normal University
Shanghai, China
izrail@163.com

Wei Wang
School of Data Science and Engineering
East China Normal University
Shanghai, China
wwang@dase.ecnu.edu.cn

Li Liu
School of Data Science and Engineering
East China Normal University
Shanghai, China
bran96@163.com

Yao Liu *
School of Data Science and Engineering
East China Normal University
Shanghai, China
liuyao@cc.ecnu.edu.cn

Abstract—Machine learning is widely used in pattern classification, image processing and speech recognition. Neural architecture search (NAS) could reduce the dependence of human experts on machine learning effectively. Due to the high complexity of NAS, the tradeoff between time consumption and classification accuracy is vital. This paper presents APENAS, an asynchronous parallel evolution based multi-objective neural architecture search, using the classification accuracy and the number of parameters as objectives, encoding the network architectures as individuals. To make full use of computing resource, we propose a multi-generation undifferentiated fusion scheme to achieve asynchronous parallel evolution on multiple GPUs or CPUs, which speeds up the process of NAS. Accordingly, we propose an election pool and a buffer pool for two-layer filtration of individuals. The individuals are sorted in the election pool by non-dominated sorting and filtered in the buffer pool by the roulette algorithm to improve the elitism of the Pareto front. APENAS is evaluated on the CIFAR-10 and CIFAR-100 datasets [25]. The experimental results demonstrate that APENAS achieves 90.05% accuracy on CIFAR-10 with only 0.07 million parameters, which is comparable to state of the art. Especially, APENAS has high parallel scalability, achieving 92.5% parallel efficiency on 64 nodes.

Index Terms—automated machine learning, neural architecture search, multi-objective, asynchronous parallel evolution

I. INTRODUCTION

Machine learning is widely used in image recognition [6], speech recognition [5], machine translation [1], etc. However, machine learning requires a lot of human intervention, which is time-consuming, laborious and error-prone. Therefore, it is urgent to improve the automatic learning capabilities of machine learning. Automated Machine Learning (AutoML) attempts to automatically learn these important steps related to data collection, feature engineering, model training, and model evaluation.

Neural architecture search (NAS) is a common method of AutoML to search network architecture. NAS automatically selects network architecture to help those with lacking machine learning background to use machine learning more easily. Recent research shows that the networks searched by NAS has achieved the performance of state of the art [12]. Many different search strategies can be used to explore the space of network architectures, including reinforcement learning (RL), evolution algorithm (EA), and gradient-based optimization, etc.

To construct NAS as a RL problem [17], [18], [23], [26], the generation of network architecture could be regarded as the action of the agent, and the action space is the same as the search space. This method can flexibly generate complex network architectures. However, Real et al. [12] proved that under the same hardware conditions, evolution algorithm could obtain results faster than reinforcement learning, especially in the early stages of search.

Liu et al. [8] proposed a continuous relaxation of the search space to enable gradient-based optimization: instead of fixing a single operation $o_i$ (e.g., convolution or pooling) to be executed at a specific layer, the authors computed a convex combination from a set of operations $o_1, ..., o_m$. This method could search network architecture efficiently. However, the gradient-based optimization hardly balance resource consumption and classification accuracy.

Evolution algorithm updates the population and produces offspring by sampling parents. Evolution algorithm encodes the network structure into a binary string and trains the neural network to obtain the value of fitness function [24]. Lu et al. [20] search network architecture by evolution algorithm to find multi-objective. Because of the design of single GPU, NSGANet lacks scalability. However, evolution algorithm is scalable naturally, which could be run on multiple nodes parallel [21]. In addition, parallel evolution faces the challenge of individual training time differences. To exploit the parallel efficiency and improve the speed of evolution. Therefore, we propose APENAS, an asynchronous parallel evolution based
multi-objective neural architecture search. In this paper, our contributions can be summarized as follows:

- We propose a multi-objective neural architecture search, focusing on both classification accuracy and computing resource, helping the researchers with lacking machine learning background to use machine learning more easily. The method we proposed not only improves the accuracy of training but also reduces resource consumption. The experimental results show that APENAS only spend 0.07 million parameters, achieving the accuracy comparable to state of the art.
- To make full use of computing resource, we propose a multi-generation undifferentiated fusion scheme to achieve asynchronous parallel evolution on multiple GPUs or CPUs, which speeds up the process of network architecture search significantly. The experimental results show that the parallel efficiency of asynchronous parallel evolution achieves 92.5%. Compared with synchronous parallel evolution, the parallel efficiency on 64 nodes improved 11.01%.
- We propose an election pool and a buffer pool for two-layer filtration of individuals. The individuals are sorted in the election pool by non-dominated sorting to improve the elitism of the Pareto front, and filtered in the buffer pool by the roulette algorithm to improve the diversity of the Pareto front.

The paper is organized as follows. Section II presents related work about neural architecture search and multi-objective optimization. Section III describes the details of APENAS. Section IV analyzes the experimental results. Finally, section V presents the conclusion and the future work.

II. RELATED WORK

A. Neural Architecture Search Methods

Zoph and Le [17] utilized a Recurrent Neural Network (RNN) controller in order to sequentially generate architectures, using an autoregressive approach. Subsequently, Zoph et al. [18] proposed NASNet, which designed a convolutional and pooling block that is repeated to construct a network. Cai et al. [3], [4] utilized RL to train a bidirectional Long Short-Term Memory (Bi-LSTM) RNN, widening or deepening each layer. Tan et al. [15] also employed the same RNN as in zoph et al. [18] to sequentially sample architectural parameters, while trying to optimize both the latency, as well as the accuracy of the generated networks on mobile devices.

Liu et al. [8] employed gradient descented in Differentiable Architecture Search (DARTS). Xie et al. [16] further expanded DARTS, by encoding each layer level as a one-hot vector and sampling one vector for each level. Pham et al. [11] utilized RL in order to sample paths within the hyper-graph of NASNet, while alternating between updating the hyper-graph’s weights and the controller’s weights, Bender [2] sampled architectures at random, in an effort to better understand one-shot methods.

Real et al. [12] directly compared their work with Zoph et al. [18]. The experimental results show that both RL and evolution algorithms can produce competent architectures. The evolution algorithm exhibited better any-time performance and requires less time to run. Stanley and Miikkulainen [14] proposed the neuroevolution of augmenting topologies (NEAT), designing neural networks through evolution. Miikkulainen et al. [10] attempted to extend NEAT to deep networks with CoDeepNEAT using a co-evolution approach. Real et al. [13] introduced perhaps the first truly large scale application of a simple evolution algorithm. Their simple EA searches over the same space as NASNet [18] and has shown to have a faster convergence to an accurate network when compared to RL and random search.

We consider an asynchronous parallel evolution based multi-objective neural architecture search, using a multi-generation undifferentiated fusion scheme to make full use of computing resource, which speeds up the process of neural architecture search.

B. Multi-objective optimization

Multi-objective optimization [9] deal with problems with multiple complementary objective functions \( f = f_1 \ldots f_n \), where \( f : E \subseteq \mathbb{R}^n \rightarrow O \subseteq \mathbb{R}^n \) for some \( n \)-dimensional objective space \( O \) and a finite design space \( E \) with dimensionality \( m \). A multi-objective optimization (maximization) can be expressed as equation 1:

\[
\arg\max_{x \in E} f(x)
\]  

It is generally not possible to find a solution that maximizes each objective equally, but instead, there is a tradeoff between Pareto-optimal points represent the best compromises across all objectives; in particular, a Pareto-optimal solution is a point \( x^* \in E \) for which it is not possible to find another point \( x^* \in E \), such that \( f_i(x^*) > f_i(x) \) for all \( i \in n \). Formally, for multi-objective maximization, \( x^* \succeq x^* \) dominates \( x \) if and only if \( f_i(x^*) \geq f_i(x) \) for all \( i \in n \), Pareto-optimal points are not dominated and form the Pareto front, which maps points with the optimal tradeoff in the objective space.

To reduce the error rate and the complexity of the network architecture at the same time, we employ the accuracy and the number of parameters (Params) as multi-objective of APENAS.

C. NSGA-II

The NSGA-II algorithm is the modified version of NSGA [22] by Deb and other researchers. NSGA-II [7] is the well-known and frequently-used evolution multi-objective optimization algorithm with non-dominated sorting, crowded distance sorting procedure and simple crowded comparison operator. The schematic diagram of NSGA II algorithm as shown in Fig.1.

APENAS sort classification accuracy and the number of parameters by non-dominated sorting. In the process of non-dominated sorting, fast crowded distance sorting is used to select the non-dominated individuals.
III. APENAS

In this section, we illustrate the details of multi-objective NAS based on non-dominated sorting and asynchronous parallel evolution based on multi-generation undifferentiated fusion scheme. We also introduce the election pool and the buffer pool are employed in APENAS to improve the diversity and elitism of the evolution algorithm.

A. Multi-objective NAS

APENAS employs non-dominated sorting to sort the accuracy and the number of parameters for balancing the tradeoff between time consumption and classification accuracy. We search over the entire architecture of the network instead of repeating the same stage throughout the entire network. The search strategy ensures the flexibility of APENAS.

1) Multi-objective: To reduce the error rate and the complexity of the network architecture at the same time, APENAS chooses the classification accuracy and the number of parameters as the multi-objective. Several metrics can be used to quantify the performance of models. Finally, we adopt the number of parameters in the literature as performance metrics. Many metrics could be used as computational complexity of model, such as inference time, floating point operations (FLOPs), activate nodes. However, FLOPs is influenced by the size of the dataset, and inference time cannot be estimated with the different computing environments. Other metrics only relate to one aspect of computational complexity.

   Particularly, we have to point out that multi-objective optimization is a scientific problem. For objectives that other applications focus on, the method proposed in this paper is still valid.

2) Encoding: In evolution algorithms, a chromosome composed of several genes represents a possible solution to the optimization problem. In APENAS, the whole network architecture composed of several stages. We use $N$ to represent the number of stages, and the $S_i$ represent the number of nodes in $i$th stage ($0 < i <= n, S_i > 1$). The edge between nodes could be encoded by a binary number, represented as $Edge_{i,j}$. $Edge_{i,j} = 0$ represents that node$_i$ and node$_j$ are not connected. Otherwise, node$_i$ is connected to node$_j$. The encoding process of APENAS includes two steps, connecting and checking. In connecting step, we keep the connectivity of graph from node$_0$ to node$_S_i$, but do not keep all the nodes have predecessor node or successor node. In checking step, APENAS guarantees all nodes have predecessor node and successor node. If node $N_i$ has neither predecessor node nor successor node, it will be discarded, as node 4 in Fig. 2 and Fig. 3. If node $N_i$ only owns successor node, the predecessor of node $N_i$ will be pointed to node $N_0$, as node 2 in Fig. 2 and Fig. 3. If node $N_i$ only has predecessor node, the successor of node $N_i$ will be pointed to node $N_6$, as node 3 in Fig. 2 and Fig. 3.

   The encoding space in APENAS is governed by our encoding method as shown in equation 2:

\[
T = \prod_{i=1}^{N}((1 + 2 + ... + (S_i - 1))!)
\]

3) Selection: APENAS selects individuals by tournament method to crossover and mutate. The tournament method takes a certain number of individuals from the population (Bootstrap Sample), and then selects the best one to enter the offspring. This operation is repeated until the new population size equals the original population size. When the better individual is selected by tournament method, crossover and mutation operations performed directly. Since a amount of individuals do not dominate each other in the later period of the parallel evolution, we adopt a random selection method. And APENAS considers roulette algorithm in crossover operation, which guarantees the elite individuals are inherited.

4) Crossover and Mutation: Crossover and mutation operations are employed after a selection operation to ensure the diversity of population. In this paper, stage is used as the
basic unit of crossover and mutation operations. The position of crossover operation is \( C(0 \leq C < S) \) and the position of mutation operation is \( M(0 \leq M < S) \). After the crossover operation, individuals are put into the buffer pool. The individuals in the buffer pool are submitted for training by the roulette algorithm. However, the random bit mutation may cause the network architecture to be disconnected or illegal. Therefore, when the mutation probability is satisfied, the stage is searched again. APENAS only keeps one stage is mutated, guaranteeing the diversity of the population and the legality of the network architecture.

B. Asynchronous Parallel Evolution

For ease of description, synchronous serial APENAS and synchronous parallel APENAS as shown in Table I, called APENAS-ss and APENAS-sp, respectively. In APENAS-ss, only one node is used, and all individuals train on this node sequentially. When all individuals in a population are trained, the crossover and mutation operations are executed. In APENAS-sp, the number of nodes is equal to population size, and each individual trained on one node. When all individuals in a population are trained, operations such as selection, crossover, and mutation are performed.

![Fig. 4. Parallel evolution](image)

1) Algorithm Description: The process of parallel evolution is shown in the Fig. 4. The individuals are assigned on each node to evolve parallelly. When any individual on the node is finished, this individual is put into the election pool immediately as shown in Fig. 5.

We refer to Fig. 5 for an illustration as well as Algorithm 1 for pseudo code. APENAS trains individuals on each node. After an individual training, the individual is put into the election pool for non-dominated sorting and crowding distance sorting. Particularly, non-dominated sorting and crowding distance sorting are just to obtain Pareto front not for the selection operator. Subsequently, APENAS only selects two individuals from the election pool for tournament method and APENAS uses random method to avoid a large number of individuals do not dominate each other in the later period of asynchronous parallel evolution. Once the individuals are selected from election pool, APENAS adopts the tournament method to select elite individual. Then the worse individual is discarded, the better individual is put into the buffer pool for preparing training. APENAS considers the buffer pool to save the pre-trained individuals of the offspring. In the training phase, individuals in the buffer pool are submitted for training by roulette algorithm.

![Algorithm 1](image)

2) Election Pool: Once the individual is trained, it will be added to the election pool. All individuals in election pool performed non-dominated sorting and crowding distance sorting. APENAS employs election pool to ensure the efficiency of APENAS. APENAS sets the size of election pool equal to population size to keep more diversity and elite individuals. In Fig.5, we present the election pool in APENAS.

3) Buffer Pool: In order to avoid mismatch the population size of offspring caused by asynchronous parallel evolution,
APENAS adopts the buffer pool to save pre-trained individuals of the offspring. The individuals in the buffer pool are selected for training by roulette algorithm in the training phase. Fig. 5 shows the buffer pool in APENAS.

IV. EXPERIMENTS

In this section, we consider experiments to evaluate the performance of APENAS. Experimental setup and metrics is shown in section IV-A. In section IV-B1, we present the classification accuracy and the number of parameters of the best architecture searched by APENAS. In section IV-B2, we test the transferability of the best architecture learned from CIFAR-10 on CIFAR-100. We also use different number of nodes to evaluate the scalability of APENAS by weak scalability experiments in section IV-B3.

A. Experimental Setup and Metrics

1) Dataset: We use CIFAR-10 for our classification task and CIFAR-100 for evaluating the transferability. We randomly split the original training set of CIFAR-10 and CIFAR-100 by 9:1 to obtain the training and validation sets. The original testing set is only used at the end of the search process to obtain the test accuracy for the models on the final tradeoff front.

2) Platform: We use private workstation with 2 NVIDIA 2080 GPU and Hygon with 600 nodes as experimental platform. In Hygon, each node includes 64 CPUs. Due to resource constraints, we evaluate performance of APENAS on private workstation. We test the scalability by using different nodes on Hygon.

3) The parameters of evolution algorithm: The probabilities of crossover and mutation operations are set to 0.8 and 0.01 respectively. The population size is 10 and the number of generations is 20 in evolution algorithm.

4) The hyper-parameters of NAS: Our experiments consider 3 stages on CIFAR-10. In each stage, the maximum number of nodes is 6. After the first and the second stage, a max-pooling with stride 2 are both placed. After the last stage, a fully-connected layer with 4096 channels is placed. We set the convolution kernel size to 3 * 3 and the padding size to 1 * 1, fixing the stride values to 1. For each generated network architecture, we limit the number of filters in all nodes to 16. We also use standard stochastic gradient descent (SGD) back-propagation algorithm and a multiple step learning rate schedule to train models on our datasets. Our initial learning rate is 0.1. In the architecture search phase, each model trained for 200 epochs, which takes about 4 days on a NVIDIA 2080 GPU implementation in PyTorch [19]. Then the classification error is measured on our validation set.

5) Parallel efficiency metrics: The experiments use $Para_{eff}$ to evaluate the acceleration effect and the scalability of APENAS. Let $T_s$ be the running time of the APENAS-sp, and $T_p$ be the running time of the APENAS-sp, then parallel efficiency is defined as equation 3:

$$Para_{eff} = \frac{T_s}{T_p}$$

The experiments use $Para_{eff}$ to evaluate the acceleration effect and the scalability of APENAS. Let $T_s$ be the running time of the APENAS-sp, and $T_a$ be the running time of APENAS, then parallel efficiency is defined as equation 4:

$$Para_{eff} = \frac{T_s}{T_a}$$

B. Results Analysis

We use the experimental setup on CIFAR-10 as explained in section IV-A.

1) Network Architecture: We choose network architecture as shown in Fig. 6 from Pareto set. The second individual was chosen with the highest classification accuracy from the parato front. The individual achieves test error of 9.95% with 0.07 millions of parameters on the CIFAR-10 testing set. In Table II, we summarize the performance of the chosen network architecture and state of the art, training on the entire official CIFAR-10 dataset. In this table, the first block presents state of the art architectures designed by human experts. The second block presents NAS methods that design the entire network. Impressively, APENAS only spends 0.07 million parameters, which achieving accuracy supers to other manual approaches and comparable with other NAS. The number of parameters we used are reduced by 48x compared to other NAS.

2) Transferability: To study the transferability of chosen network architecture as shown in Fig. 6(a), we consider using CIFAR-100 dataset to evaluate the transferability of the found architecture by APENAS. We use the same setup on CIFAR-10 dataset as explained in section IV-A. The entire process of APENAS takes about 3 hours on a single NVIDIA 2080 GPU. Table III shows that the architecture learned on CIFAR-10 is suitable for CIFAR-100, and could be compared with the architecture generated by manual design.

3) Evaluation of Asynchronous Parallel Evolution: In order to demonstrate the scalability of APENAS, we search architecture on CIFAR-10 by different nodes. The orange line in Fig.7 shows the parallel efficiency of APENAS. It is a slow upward trend with the increasing number of nodes. When 64 nodes are used, the parallel efficiency $Para_{eff}$ achieves 92.5%. The experimental results show that the APENAS has a good
acceleration effect. To demonstrate the parallel efficiency of asynchronization, we design APENAS-sp to be compared with APENAS. The blue line in Fig.7 shows the parallel efficiency of APENAS-sp. It is a more quick upward trend than the orange line with the increasing number of nodes. When the number of nodes is 64, the parallel efficiency \( \text{Para}_{eff}^{\text{async}} \) is 81.49%.

By comparing the parallel efficiency of APENAS and APENAS-sp, we find that our method improves the evolution efficiency on different nodes. In summary, the experiments on Hygon show that asynchronous parallel evolution could achieve 92.5% parallel efficiency on 64 nodes, which improves 11.01% compared with synchronous parallel evolution.

V. CONCLUSION

This paper presents APENAS, an asynchronous parallel evolution based multi-objective NAS, which find the solution with good accuracy and a very small number of parameters compared to state of the art. Especially, APENAS shows the high scalability on multiple GPUs or CPUs. It helps the researchers with lacking machine learning experience to utilize machine learning easily. The experiments on CIFAR-10 show
that APENAS could find network architecture with 90.05% accuracy and 0.07 million parameters. The experimental results on hygon show that APENAS achieves 92.5% parallel efficiency on 64 nodes.

There are several directions to improve APENAS further. For example, a more fine-grained structure could be considered as the basic unit of network architecture search to improve classification accuracy. It will also be considered to change the methods of crossover and mutation operations during the search process.

ACKNOWLEDGMENT

This work is supported by the National Key Research and Development Program of China (No.2020YFA0607902).

REFERENCES


