GAAF: Searching Activation Functions for Binary Neural Networks through Genetic Algorithm

Yanfei Li†, Tong Geng†, Samuel Stein†, Ang Li‡, and Huimin Yu†
†Zhejiang University, Hangzhou, Zhejiang, China
‡Pacific Northwest National Laboratory, Richland, WA, USA
aoxue18@zju.edu.cn, tong.geng@pnnl.gov, ang.li@pnnl.gov, yhm2005@zju.edu.cn

Abstract—Binary neural networks (BNNs) show promising utilization in cost and power-restricted domains such as edge devices and mobile systems. This is due to its significantly less computation and storage demand, but at the cost of degraded performance. To close the accuracy gap, in this paper we propose to add a complementary activation function (AF) ahead of the sign based binarization, and rely on the genetic algorithm (GA) to automatically search for the ideal AFs. These AFs can help extract extra information from the input data in the forward pass, while allowing improved gradient approximation in the backward pass. Fifteen novel AFs are identified through our GA-based search, while most of them show improved performance (up to 2.54% on ImageNet) when testing on different datasets and network models. Our method offers a novel approach for designing general and application-specific BNN architecture. Our code is available at http://github.com/flying-Yan/GAAF.

Index Terms—BNN, Genetic Algorithm, Activation Function

I. INTRODUCTION

Binary Neural Networks (BNNs) [1], [2] binarize the full precision inputs and weights of deep neural networks into binary values: \{+1, −1\}. Consequently, the costly floating-point dot-product (or multiply accumulation) computation can be replaced by lightweight bit-wise exclusive-nor (\(\text{xnor}\)) and population-count (\(\text{popc}\)) operations. BNNs show great potential in cost- & power-restricted domains [3]–[7], such as Internet-of-Things (IoT), embedded systems, mobile devices, etc. This is primarily due to the significantly reduced computation requirements, lower power cost, lower storage demand, and improved robustness to input noise [1], [8], [9].

Existing research has shown that, through BNN, significant speedups and memory reduction can be harvested on CPUs [10], GPUs [8], [11], and FPGAs [12]–[14], compared with their DNN counterparts. As a result, BNNs have been adopted in a variety of practical applications such as COVID-19 face-cover detection [3], auto-driving [4], smart agriculture [5], image enhancement [6], 3D objection detection [7], etc.

Though BNNs significantly alleviate the computation cost, they suffer from non-trivial performance degradation largely due to the imperfection of the binarization function and its derivative approximation [15], [16]. Most existing BNN works binarize the inputs and weights into binary (+1/-1) tensors through a deterministic sign function. This sign function also serves as the non-linear activation function of the neural networks. However, the gradient of the sign function is always zero and is non-differentiable at zero. To allow standard backpropagation still able to work, the Straight Through Estimator (STE) [17] is adopted as an approximated gradient of the sign function [1], [2]. STE approximates the gradient as an identity function. To avoid the saturation effect where certain neurons can never get a chance to toggle its binary value, a clip function is placed prior to the sign function. It zeros out the gradient in the backward pass when real-value is outside the range (usually [-1,1]) but making no impact in the forward pass.

The accuracy degradation with the sign activation in training arises from the information loss due to the rigid binarization through sign in the forward pass [16], and the approximation imposed by the STE gradient. Consequently, significant efforts have been devoted in designing better activation functions for BNNs [18]–[21]. Bi-Real Net [18] presented a customized BNN sign function – ApproxSign, which is a differentiable piecewise polynomial function that has a triangle-shaped derivative, acting as a magnitude-aware gradient. BNN+ [19] showed a switch-like activation function for a better approximation of the sign derivative function. Self-Binarizing network [22] even adopted an evolving activation function to progressively reshape the scaled hyperbolic tangent function towards the sign function. Finally, ReActNet [23] generalized the sign and PReLu functions as RSign and RPReLU, allowing shifting and reshaping the distribution of the real-valued feature maps prior to binarization, mitigating the information loss from to the rigid sign activation.

Despite harvesting improved performance, most of these designs are crafted based on human expertise [18]–[23]. Whether they have already been optimal or if better activation functions can be found still remain an open question.

Recently, using machine-learning based search techniques for automatically discovering traditionally human-designed components of deep neural networks exhibits great effectiveness. For example, enhanced accuracy has been showcased with the new activation functions discovered by reinforcement learning [24], with respect to the widely used ReLU activation in DNNs. In EvoNorms, a set of new normalization-activation layers have been found by evolution-based search [25]. DARTS [26] modeled network architecture search (NAS) in a differentiable manner, thus can be efficiently handled by gradient descent.

Motivated by these search approaches, in this paper, we propose to leverage the genetic algorithm (GA) based search algorithms for discovering novel activation functions for BNNs, and perform associated design space exploration. Observing that the complementary activation function is one-to-one mapping (i.e., transforming a tensor under the same shape), we define the structure of the first-order (Type-I) and second-order (Type-
II) activation functions, acting as the templates describing how unary and binary functional operators can be composed to form an activation function. Each of the operators is selected from a list of labeled candidates, and the goal is to find the optimal combination. We encode the choice of each combination as a string of fixed length. We then rely on GA to auto-search the optimal combination. The newly discovered activation functions are evaluated on CIFAR10 and ImageNet with several BNN models, demonstrating improved validation accuracy (up to 2.54% on ImageNet). In total, seven lightweight activation functions are identified showing low computation cost and few parameters. Similar to the clip function, these complementary activation functions only show-up in the training, and do not impact BNN inference efficiency. This paper thus makes the following contributions:

- Our work is the first effort to automatically search complementary activation functions for BNNs, to the best of our knowledge.
- Through GA-based effective searching, we have identified seven activation functions that can show enhanced accuracy on different datasets and network models, with up to 2.54% performance improvement on ImageNet.

This paper is organized as follows. In Section II we discuss the background of BNNs and GA-based search, highlighting our motivation. In Section III we present the proposed searching methodology. In Section IV we show the experiment and evaluation results. We discuss our result in Section V summarize the related work in Section VI and conclude in Section VII.

II. BACKGROUND AND MOTIVATION

A. Binary Neural Networks

BNNs [1], [2] are seen as an alternative approach of neural network compression, in addition to network pruning [27], [28]. Instead of reducing the total number of weights in the network, BNN reduces the bit-width of the inputs and weights to the extreme of a single bit only. That is to say, both weights and inputs are expressed as 1-bit with two possible values: +1 or -1. Compared to traditional DNNs, the 32-bit floating-point multiplication and addition can be replaced by efficient bit operations (i.e., exclusive-nor and population-count), which is extremely hardware-friendly [11], [14], given the one-to-one mapping from a neuron to a logic-bit.

A typical binary convolution layer in BNNs is depicted in Fig. 1. First, the real-valued input$_L$ and weight $w_iL$ of layer $L$ are binarized. Then, a binarized convolution (BConv) is applied on the binary inputs and weights. The outputs of BConv are real-valued, and serve as the inputs for the consecutive layer $L + 1$ (i.e., input$_{L+1}$) after batch-normalization (BN) [29].

BNN uses the sign function in Eq. (1) for binarization, which also serves as the non-linear activation function. Its plot is shown in Fig. 2.

\[
\text{Forward: } x^b = \text{sign}(x) = \begin{cases} 
+1 & x \geq 0 \\
-1 & \text{otherwise} 
\end{cases} \quad (1)
\]

Since the gradient of sign function is always zero and is non-differentiable at 0, the Straight-Through-Estimator (STE) activation function and its gradient.

\[
\text{Backward: } \frac{\partial \text{Loss}}{\partial x} = \frac{\partial \text{Loss}}{\partial x^b} \cdot 1_{|x|<t_{clip}} \quad (2)
\]

where $t_{clip}$ is typically 1. This implies that for values within the range of [-1,+1], the gradient of full-precision values ($\frac{\partial \text{Loss}}{\partial x}$) is equal to the gradient of the binarized values ($\frac{\partial \text{Loss}}{\partial x^b}$). For values outside the range, the gradient keeps zero. The plot is shown in Fig. 3.

Eq. (1) and (2) are achieved in BNN through the combination of Clip(-1,1) (also known as hard tanh) and Sign in Eq. (1). Being followed by Sign, Clip(-1,1) does not impact the forward pass. However, for backward pass, it constraints the range of [-1,1] over the gradient and cuts-off values beyond the range.

As seen in Fig. 2 the sign function is very rigid by fixing the binarization boundary at value 0. In [23], Liu et al. observed that by involving small variations to distributions of feature maps, better performance can be achieved. This is because the output only has two options: +1/-1, a small shift or reshape to the distribution of the input feature maps could result in a completely distinct binary output [23].

With such an observation, they proposed two improving approaches: (i) $R\text{Sign}$, where a channel-wisely learnable bias is imposed to the original sign function, see Eq. (3).

\[
x^b_i = R\text{Sign}(x_i) = \begin{cases} 
+1 & x_i \geq \alpha_i \\
-1 & x_i < \alpha_i 
\end{cases} \quad (3)
\]

where $\alpha_i$ is a learnable channel-wise parameter. Fig. 4 and 5 show the plot of the $R\text{Sign}$ activation function and its gradient.
The challenge then is how to find a decent complementary representation of the solution domain that determines the search space; (ii) a fitness function that estimates the degree of how fit the candidate solution is, usually being the objective function of the optimization problem.

GA typically starts from a population of randomly generated individuals, where the fitness of each individual is evaluated. Then, the parent individuals are stochastically selected from the current population, where the individuals with better fitness scores have a better chance to be selected for reproduction. Based on the selected parent individuals, an offspring is generated by operations such as crossover, mutation. The new offspring is then evaluated. The best individuals among the population alongside the new offspring form the new population, which will be used for the next iteration. Usually the population size is predefined based on the problem settings. The iterative process stops when the population has converged (not producing any offspring that are significantly different from the previous generation) or a satisfactory solution is found.

GA is an appealing approach to obtain high-quality solutions for an optimization problem thanks to its ability to deliver a good solution fast-enough. Please refer to [31] for a detailed review about GA.

Motivation 2: Our purpose is to search a good complementary function, according to an architecture template with its operators selected from a list of candidate functions, the search space is discrete. Since the combination of good operators tends to produce a better solution, it better matches the process of GA. Additionally because of its simplicity and efficiency, we adopt GA in this work for searching the optimal activation function of BNNs.

III. Methodology

We first introduce the proposed encoding strategy and fitness function and then describe our GA-based searching approach in detail.

A. Encoding Strategy and Search Space

The gene encoding strategy plays a key role in GA, which essentially dictates the search space. The search space needs to be sufficiently large such that a strong solution can be generated, but not too vast and complex that the majority of the solution space cannot be traversed. The definition of the search space is a trade-off between the capacity (i.e., more likely to incorporate a good solution in the space) and the complexity (i.e., more difficult to find a good solution in the space).

In this paper, we represent the activation function as a graph that transforms an input tensor to an output tensor with the same shape (i.e., one-to-one mapping). To simplify the search process and constrain the size of the search space to a reasonable and computationally feasible size, we fix the graph structure in two patterns: Type-I, representing the first-order architecture, is shown in Figures 8; Type-II, representing the second-order architecture, as shown in Fig. 9. The order here refers to the number of combination operators as discussed later. Based on our observation, higher-order architectures show marginal benefits with substantially larger search space.

![Fig. 4: RSign from [23].](image1)

![Fig. 5: Gradient of RSign.](image2)

![Fig. 6: RPreLU+Clip(-1,1) in [23].](image3)

![Fig. 7: Gradient of RPreLU+Clip](image4)
In Fig. 8 and 9, \( X \) is the input tensor of the complementary activation function, \( Y \) is the output. Note that \( X \) is duplicated for extracting and integrating information from different paths. \( U \) represents a unary operator, which takes one tensor as input and one tensor as the output. The input and output tensors are in the same shape. \( B \) represents a binary operator, which takes two tensors with the same size as the input, and one tensor as output also with the same size. Overall, the mapping from \( X \) to \( Y \) respects to one-to-one mapping.

![Fig. 8: Type-I encoding (1st-order).](image)

![Fig. 9: Type-II encoding (2nd-order).](image)

For Type-I encoding shown in Fig. 8 the replicated \( X \) tensors pass through independent unary operators, and then combined through a binary operator to generate the output tensor \( Y \). As for Type-II encoding in Fig. 9, the three replicated \( X \) tensors pass through their independent unary operators respectively, and then merge gradually through two binary operators to generate the final output tensor \( Y \).

With the fixed graph structure, each candidate solution can be encoded as a fixed length string of values which represent the combination of different operators. Each value in the encoded string represents the index of a corresponding operator. For Type-I, the encoding output is a string with length of 3, representing 2 unary operators and 1 binary operator. The Type-II encoding, in contrast, has 6 values which include 4 unary operators and 2 binary operators. Regarding the format of the string, all unary operators stay on the left and binary operators stay on the right, which essentially yields benefits in the following processing which will be discussed later.

A complete list of unary and binary operators are summarized as follows:

**Unary Operator:** \( x, \sqrt{|x|}, \log(|x|), 1.0/(1 + e^{-x}), e^{(-|x|)}, e^{-x^2}, \sin(x), \cos(x), \tan(x), \arctan(x), \text{erf}(x), \text{erfc}(x), \max(x, 0), \min(x, 0), \alpha, \alpha + x, \alpha \cdot x \)

**Binary Operator:** \( x + y, x - y, x \cdot y, x/y, x/(x + y), \max(x, y), \min(x, y), x/(1 + e^{-y}), e^{(-|x-y|)}, \beta \cdot x + (1 - \beta) \cdot y \)

where \( \alpha \) and \( \beta \) are learnable channel-wise parameters, which can be jointly optimized with other parameters in the neural networks through back-propagation. 0 inUnary operators means the output is always 0, which enables the pruning of redundant operators.

Overall, there are 22 candidate unary operators and 11 candidate binary operators. Therefore, the value of gene-\( U \) thus can go from 0 to 21, where each value corresponds to an operator in the list. The same condition applies to gene-\( B \). Those operators are scalar to scalar which are easy to implement and lower the computation complexity. Using such a fixed graph structure, combinations of different unary and binary operators can form distinct candidates and constitute the search space. Although the search space is drastically shrunk with the fixed graph structure (i.e., architecture), it is still massive: for Type-I, the search space is \( 22 \times 22 \times 11 = 5.3K \); for Type-II, the search space is: \( 22^4 \times 11^2 = 28.3M \).

**B. Fitness Metric**

The fitness metric in GA is to assign a fitness value for each individual in the population, which indicates how good a candidate solution is. The fitness score also determines the ability of an individual to compete with other individuals during selection. The fitness function is designed according to the problem, which is usually the value returned by the objective function of the optimization problem.

In this paper, we focus on searching for promising generalized complementary activation functions for BNNs. The image classification is used to evaluate the fitness, the top-1 validation accuracy will be the fitness value. Since not all well-known real-valued models can be seamlessly applied to BNNs, and given the widely adoption of ResNet structure in BNNs (the shortcuts are essential for rich information flow tackling the information loss from binarization and gradient vanishing/explosion for the BNN latent variables), we use the classical ResNet18 model to perform the evaluation. As each trial assessment of the fitness evaluation requires the training of a BNN model, to be efficient, the BNN model is trained on CIFAR10 dataset. In particular, the top-1 validation accuracy after the BNN version of Resnet18 trained on CIFAR10 for 15 epoches will be assigned to each candidate individual as the fitness value.

**Early Rejection:** Since the candidate activation function can be generated pseudo-randomly, some of them may result in non-convergence during training, we adopt the early rejection strategy. We set a rejection threshold \( T_1 \), whereby when the validation accuracy is less than \( T_1 \) after the first epoch of training, the candidate individual will be discarded without further training, and we move-on to the next step. Otherwise, the training is continued up to 15 epochs. With this early rejection strategy, the search process can be much more efficient.

**C. GA Implementation**

Fig. 10 illustrates the implementation workflow of GA. By defining the encoding strategy and fitness function, the initial population of size \( S \) can be generated. Each individual of the population is evaluated and assigned with a fitness value. The population is then sorted according to the fitness values from high to low. Therefore, the first individual appears to show the best fit. From here, the evolution iteration starts. In each iteration, we first choose two parent individuals and then a new offspring is generated by combining the two parent individuals with crossover and mutation. The new offspring is evaluated to obtain its fitness value.
This fitness value is then compared with the fitness value of the last individual in the population: (i) if the new offspring shows a higher fitness value, the last individual is replaced by the new offspring, then the new population is sorted in descending order based on its fitness values. The next iteration then begins using the new population. (ii) Otherwise, the next iteration starts using the old population. During the evolution process, the total population size keeps unchanged \(S\). Once the population has converged, meaning that no more satisfactory solutions can be generated, the iterative search terminates and a series of optimal solutions are reported.

We describe each process of Fig. 10 in detail.

**Initialization:** To ensure the diversity of the population, individuals in the initial population are generated randomly. For Type-I (Type-II) encoding, the first two (four) genes are randomly sampled from the 22 unary operators; the last one (two) gene is randomly sampled from the 11 binary operators. The fitness value is then evaluated for the generated individual. This process of generation continues until the size of population reaches to \(S\). Then the population is sorted in descending order based on the fitness values.

**Selection:** Selection is a critical step in GA. The convergence rate of GA largely depends on the selection pressure. Selection involves choosing the parent individuals from the population which will be used for reproduction of the new offspring. The selection process is mostly based on fitness. For diversity, each individual should have the chance to be selected. Nevertheless, the theory of “Survival of the Fittest” of GA gives preference to the individuals with higher fitness values; those individuals have a higher chance of being selected for reproduction.

There are several well-known selection techniques, such as elitism, roulette wheel, rank, tournament, Boltzmann, stochastic universal sampling, etc. Each selection technique has its advantages and drawbacks. Taking into consideration the variability and efficiency of the algorithm, three selection techniques – elitism selection, tournament selection, and proportionate selection – are adopted here. In each iteration, one of the three selection techniques is randomly picked for selecting the two parent individuals. We discuss each selection technique:

1) **Elitism selection** chooses the two best individuals based on their fitness values. It always chooses the best individuals in the population for reproduction, which leads to a better chance of reproducing a better individual. However, Elitism may result in early convergence due to being trapped in a local maximum.

2) **Tournament selection** chooses individuals according to their fitness values from a stochastic roulette wheel in pairs. The first individual is randomly selected from the population; the second individual is randomly picked from the right part of the first one in the population. Therefore, the second individual has a lower fitness than the first one. Both the two individuals are chosen at random, which ensures each individual has a chance to be selected.

3) **Proportionate selection** is implemented by describing the fitness value as a probability distribution over the population. Then two random individuals are selected from the distribution. Due to the transformation, the chance of each individual being selected is proportional to its fitness value.

Elitism selection prefers the best individuals whereas tournament selection and proportionate selection stochastically choose individuals. The probabilistic combination of those three selection techniques can take the advantages of each technique, while avoiding their drawbacks, accelerating the convergence speed.

**Crossover:** The crossover step generates new offspring by combining different portions of two or more selected parent individuals. The well-known crossover operators include single-point, two-point, k-point, uniform, partially matched, ordered, precedence preserving, shuffle, etc. In this work, the single-point crossover is adopted due to its simplicity and the relatively simple settings of our GA problem.

Fig. 11 illustrates how crossover is performed between the two parent individuals in this paper. The crossover point is randomly picked. According to its position, both of the parent individuals are split into a left part and a right part. The new offspring is then formed by combining the left part of \(Parent_1\) with the right part of \(Parent_2\), or by combining the left part of \(Parent_2\) with the right part of \(Parent_1\). The new offspring has the same size as its parents.

**Mutation:** Mutation is essential for GA. The key idea is to
insert random genes in offspring to maintain the diversity of the population for avoiding premature convergence. The well-known mutation operators include displacement, simple inversion, and scramble mutation. Displacement mutation displaces a substring of a given individual within itself. Simple inversion mutation reverses the substring between any two specified locations of an individual solution. Scramble mutation places the elements in a specified range of an individual solution in a random order.

In our GA design, we use single point mutation, which is one variant of displacement mutation, as illustrated in Fig. 12. First, a single mutation point is randomly selected from the individual. Then the chosen gene is then replaced by a new randomly generated gene. Based on the mutation point, we figure out whether the mutation refers to operator $U$ or $B$. If $U$, a replacement gene is randomly generated from the 22 unary operators; if $B$, the replacement gene is randomly generated from the 11 binary operators.

Note that the mutation operator is capable of maximizing the variance of the newly generated individuals, which is important in finding the global optimal solution.

IV. EXPERIMENTS

A. GA Setting

We perform the GA-based searching for optimal BNN complementary activation functions. The BNN version of ResNet18 with candidate complementary activation function is trained on CIFAR10. We use the standard pre-processing, the batch size is 128. The Adam optimizer with $\text{betas} = (0.9, 0.999)$ is adopted with a constant learning rate $5\text{e}-3$. After 15 epochs of training, the top-1 validation accuracy is assigned to the candidate individual as its fitness value.

Like most BNNs, the first and last layer of BNN models used in this paper should always be left in full-precision. This is because the first layer extracts the initial information from the input images, whereas the last layer immediately dictates the final prediction – both require a higher capacity in order to show good performance.

The early rejection threshold $T_1$ is empirically set as 11% in our fitness evaluation. With the early rejection strategy, the compute resources can be allocated focusing on the most promising candidates. To further enhance searching efficiency, the threshold can be slowly increased as the search continues, so that it can early reject more unqualified candidate individuals and preserve promising solutions. This is based on the intuition that as time goes on, the descendants of the GA tend to be more well-behaved than in the early stage of the search. Increasing the early rejection threshold thus improves the search efficiency.

As the searching goes on, the threshold $T_1$ increases to be 25%, 35%, 40%. The size of population $S$ is set to be 30.

B. Activation Functions found by GA Search

| Type   | Function                        | Graph |
|--------|---------------------------------|-------|
| RSign  | $\begin{cases} +1 & x \geq \alpha \\ -1 & x < \alpha \end{cases}$ |       |
| RPreLU | $\begin{cases} x - \gamma + \zeta & x \geq \gamma \\ \beta \ast (x - \gamma) + \zeta & x < \gamma \end{cases}$ |       |
| AF1    | $\sin(x) - \cos(x)$             |       |
| AF2    | $\sin(x) + \cos(x)$             |       |
| AF3    | $\max(x, 0) + \sin(x)$          |       |
| AF4    | $\beta \ast \cos(x) + (1 - \beta) \ast x$ |       |
| AF5    | $\min(x, 0) + \sin(x)$          |       |
| AF6    | $\beta \ast \text{erf}(x) + (1 - \beta) \ast \max(x, 0)$ |       |
| AF7    | $\exp(-x^2) - \sin(x)$          |       |
| AF8    | $\cos(x) + \text{atan}(x)$      |       |
| AF9    | $\beta \ast \cos(x) + (1 - \beta) \ast \text{atan}(x)$ |       |
| AF10   | $\cos(x) - \text{atan}(x)$      |       |
| AF11   | $\cos(\text{atan}(x)/2) + x$    |       |
| AF12   | $\beta \ast \cos(x + \alpha) + (1 - \beta) \ast x$ |       |
| AF13   | $\cos(\text{atan}(x)) + x$      |       |
| AF14   | $\cos(\text{erf}(x)) - x$       |       |
| AF15   | $\cos(-x) + x$                  |       |

TABLE I: Top discovered activation functions and graphs

Through the GA-based searching, we identified multiple new complementary activation functions for BNNs, as listed in Table I. Specifically, the first two are the hand-crafted RSign...
and **RPReLU** from [23] discussed in Section 2.1, shown for comparison purposes. The 10 activation functions from AF1 to AF10 are obtained in the GA search with Type-I encoding. The last 5 AFs from AF11 to AF15 are obtained in the GA search with Type-II encoding. We show the math expression and the graph in Table I as well. Looking at the table, we draw the following observations:

- These obtained top performing AFs are all relatively simple in their expressions. This is potentially because complicated function increases difficulties in optimization.
- Counterintuitively, trigonometric functions such as \(\sin\) and \(\cos\) exist in almost all the AFs (except AF6). This observation indicates that incorporating periodic functions can extract more information from the inputs and enhances the representativeness of the activation function.
- For the binary operator \(B\), the linear combination such as \(x + y, x - y, \beta x + (1 - \beta) y\) exist more often than other complex binary operators. This might because alternative binary operators may lead to instant and significant jitters/pulses, or discrete gradients, which jeopardizes the stability of the training.
- For Type-II encoding (Fig. 9), all the searched AFs (AF11-AF15) are having their \(U_A\) gene being \(x\) or \(-x\). This may imply that although the unary operator can help in extracting information, the raw input \(x\) contains the most useful information and should always be kept as the residual in the combination.

### C. Testing the Searched Activation Functions

To showcase the performance and generalization of the obtained novel complementary activation functions, we test them on CIFAR10 and ImageNet with different BNN models. As mentioned before, the first and last layer of BNN models are left in full-precision. For each model, we evaluate each AF in Table I. The original BNN is used as the baseline. The two hand-crafted AFs (RSign and RPReLU) from [23] are used for comparison. All models are trained using the same hyperparameters except the activation functions. The Adam optimizer [32] with \(betas = (0.9, 0.999)\) is adopted for the training.

**CIFAR10**: We tested all the AFs in Table I on CIFAR10 with 3 widely used BNN models: ResNet18, NIN-Net and ResNet34. We use standard pre-processing. All models are trained until 300 epochs. The initial learning rate is 5e-3, and is shrunk by a factor of 0.2 at the 80th, 150th, 200th, 240th, and 270th epoch respectively. The batch size is 128. The final validation accuracy is shown in Table I.

As shown in Table I, all the 15 AFs obtained by the GA-based search attain performance improvement over the BNN baseline on ResNet18. Specifically, the top 5 functions (AF12, AF6, AF11, AF3, AF5) improve accuracy by 0.71%, 0.63%, 0.63%, 0.6%, 0.59%, respectively. For NIN-Net, 14 out of the 15 AFs show performance improvement. The top 5 functions (AF6, AF10, AF12, AF14, AF4) improve performance by 1.45%, 1.39%, 1.22%, 1.19%, 1.01%. Finally, for ResNet34, 14 out of the 15 functions outperform the original BNN architecture. The top 5 functions (AF13, AF11, AF14, AF9, AF4) improve accuracy by 1.2%, 1.08%, 0.93%, 0.75%, 0.68% respectively. The best improvement over the baseline BNN on CIFAR10 is attained by AF6 on NIN-Net, showcasing 1.45% accuracy enhancement. Overall for CIFAR10, most of the GA-searched AFs (except AF3, AF7, AF11 and AF15) match or outperform the baseline BNN, RSign and RPReLU on all of the three network models (ResNet18, NIN-Net and ResNet34). We did not observe significant performance improvement using RSign and RPReLU.

**ImageNet**: For ImageNet, two different network models are utilized: ResNet18 and NIN-E. NIN-E is an enhanced implementation of NIN-Net proposed in [33], which enlarges the kernel size of the first four mlpconv layers from \(1 \times 1\) to \(3 \times 3\), and increases the number of output channels of the first two mlpconv layers from 96 to 128.

We use the standard pre-processing technique: all images are resized to \(256 \times 256\) and randomly cropped to \(224 \times 224\) for training. The validation dataset uses center crop only. We set the initial learning rate as 5e-3, which is gradually decreased during training by a factor of 5 at the 25th, 35th, 40th, and 45th epoch. Each model is trained over 50 epochs. The batch size is 64. The validation accuracy is listed in Table III.

For ResNet18, 11 out of the 15 discovered AFs improve the top-1 accuracy. The best behaving functions (AF3, AF5, AF12, AF1, AF2) harvest 1.782%, 1.428%, 1.23%, 1.114%, 0.984% performance improvement respectively, over the BNN baseline. For NIN-E, all the 15 AFs improve the validation accuracy. The promising functions (AF9, AF4, AF6, AF1, AF3) improve the top-1 accuracy by 2.54%, 2.502%, 2.132%, 2.008%, 1.95%, respectively. The most significant improvement is attained by AF9 on NIN-E, showcasing as much as 2.54% accuracy improvement over the BNN baseline.

Overall, except AF5, AF8, AF9, AF10 and AF11, all the remaining AFs (10 out of 15) outperform the BNN baseline, RSign, and RPReLU approaches. Again, we did not observe performance improvements by using RSign and RPReLU on ResNet18. For NIN-E, RSign and RPReLU bring nearly 0.25% and 0.54% top-1 accuracy improvement.

### Table II: Validation Accuracy on CIFAR10.

| AF       | ResNet18 | NIN-Net | ResNet34 |
|----------|----------|---------|----------|
| BNN baseline | 90.69    | 86.03   | 91.00    |
| RSign [23]  | 90.64    | 86.36   | 91.10    |
| RPReLU [23] | 90.79    | 86.34   | 90.66    |
| AF1       | 91.20    | 86.87   | 91.45    |
| AF2       | 91.02    | 86.77   | 91.39    |
| AF3       | 91.29    | 86.55   | 91.08    |
| AF4       | 91.16    | 87.04   | 91.68    |
| AF5       | 91.28    | 86.36   | 91.26    |
| AF6       | 91.32    | 87.48   | 91.39    |
| AF7       | 90.75    | 86.09   | 90.98    |
| AF8       | 91.06    | 87.03   | 91.16    |
| AF9       | 90.93    | 86.96   | 91.75    |
| AF10      | 91.24    | 87.42   | 91.48    |
| AF11      | 91.22    | 86.46   | 92.08    |
| AF12      | 91.40    | 87.25   | 91.92    |
| AF13      | 90.90    | 86.56   | 92.20    |
| AF14      | 91.15    | 87.22   | 91.93    |
| AF15      | 91.14    | 86.15   | 91.58    |
which partially explains the better performance. The $\alpha$ activation functions stand out showing performance improvement. This is obvious with the monotonic feature is still conserved with our searched complementary design and eliminate runtime cost. We argue that this valuable Sign shown that the Fig. 15 shows the complete activation functions (i.e., AF + functions and Fig. 14 visualizes their gradients. Additionally, the training. Fig. 13 illustrates these complementary activation functions and successfully generalize such improvement to a different space, we propose to rely on genetic algorithm for identifying a complementary activation function prior to the sign-based network structure 

\begin{table}
\centering
\begin{tabular}{|c|c|c|}
\hline
AF & ResNet18 (Top-1/5) & NIN-E (Top-1/5) \\
\hline
BNN baseline [1] & 54.284 / 77.670 & 49.730 / 74.144 \\
RSign [23] & 54.292 / 77.618 & 49.980 / 74.514 \\
RPReLU [25] & 53.028 / 76.850 & 50.286 / 74.252 \\
AF1 & 55.398 / 78.558 & 51.738 / 75.290 \\
AF2 & 55.268 / 78.394 & 51.514 / 75.520 \\
AF3 & 56.066 / 79.035 & 51.680 / 75.752 \\
AF4 & 55.064 / 78.106 & 51.232 / 76.186 \\
AF5 & 55.712 / 78.454 & 50.210 / 74.494 \\
AF6 & 54.384 / 77.636 & 51.862 / 75.518 \\
AF7 & 54.624 / 77.752 & 50.502 / 74.514 \\
AF8 & 54.194 / 77.640 & 49.918 / 74.272 \\
AF9 & 53.414 / 77.960 & 52.270 / 75.822 \\
AF10 & 53.004 / 76.454 & 50.834 / 75.216 \\
AF11 & 54.025 / 77.836 & 50.722 / 74.768 \\
AF12 & 55.514 / 78.556 & 51.558 / 75.308 \\
AF13 & 54.390 / 77.728 & 50.714 / 74.742 \\
AF14 & 54.488 / 77.976 & 50.324 / 74.466 \\
AF15 & 55.092 / 78.246 & 50.374 / 74.500 \\
\hline
\end{tabular}
\caption{Validation Accuracy on ImageNet.}
\end{table}

V. DISCUSSION

Based on our evaluation with the three distinct network models on CIFAR10 and ImageNet, seven (i.e., AF1, AF2, AF4, AF6, AF12, AF13, AF14) out of the 15 investigated activation functions stand out showing performance improvement and successfully generalize such improvement to a different dataset. These seven searched activation functions are listed below. Compared to STE, the gradients of these functions are much smoother, rendering a more stable training process. Compared to STE, the gradients of these functions are much smoother, rendering a more stable training process, which partially explains the better performance.

- AF1: $\sin(x) - \cos(x)$
- AF2: $\sin(x) + \cos(x)$
- AF4: $\beta \cdot \cos(x) + (1 - \beta) \cdot x$
- AF6: $\beta \cdot \cos(x) + (1 - \beta) \cdot \max(x, 0)$
- AF12: $\beta \cdot \cos(x + \alpha) + (1 - \beta) \cdot x$
- AF13: $\cos(\text{atan}(x)) + x$
- AF14: $\cos(\text{erf}(x)) - x$

The $\alpha$ and $\beta$ here are channel-wise parameters learned during the training. Fig. 13 illustrates these complementary activation functions and Fig. 14 visualizes their gradients. Additionally, Fig. 15 shows the complete activation functions (i.e., AF + Cos(1.1)). Fig. 16 further depicts their gradients.

Regarding the impact of computation efficiency, we argue that these activation functions are very lightweight scalar functions. Additionally, existing works [8, 13] have already shown that the batchnorm layer can be integrated with the Sign layer to form a very efficient threshold-comparison layer during BNN inference, which can greatly simplify hardware design and eliminate runtime cost. We argue that this valuable feature is still conserved with our searched complementary activation functions. This is obvious with the monotonic functions (AF3-AF15):

$$A\text{Sign}(x) = \begin{cases} 
+1 & AF(x) \geq 0 \\
-1 & AF(x) < 0 \\
+1 & x \geq \tau \\
-1 & x < \tau
\end{cases}$$

For the two periodic functions (AF1 and AF2), the integration of batchnorm, activation, and sign can be achieved by piece-wise threshold operation.

Regarding the extra memory space cost, there are no additional parameters in AF1, AF2, AF13 and AF14. Other AFs may contain one or two parameters, but only in channel-wise, which is almost negligible comparing to the size of the weights.

Please note that although in this work we adopt GA-based searching to automatically find new complementary functions for BNNs, this approach can be applied to other network models such as quantized neural networks, transformers, CNNs, or for domain-specific architectural search.

VI. RELATED WORK

A. Binary Neural Networks

BNNs demonstrate substantially advanced memory and computational efficiency at the cost of certain accuracy degradation. Since the proposal of BNNs [1], [2], there have been significant efforts committed to closing the accuracy gap between BNNs and the full-precision DNNs. These include adding learnable scaling factors (also known as gain term) [33, 36], adopting multiple basis [33, 35, 37], designing BNN-oriented network structure [16, 18, 23, 38], and reforming BNN training methodologies [22, 23, 36, 39, 41]. More detailed information can be found in the BNN surveys [42, 43].

B. Automatic Search

Hand-crafted activation functions heavily rely on human expertise and prior network architecture design experience. Due to such limitations, the use of automatic searching algorithms has received more attention in recent years. Various searching strategies have been proposed, including Bayesian optimization [44, 45], evolutionary approaches [46, 48], reinforcement learning (RL) [49, 51], and gradient-based methodology [26, 52, 53], etc.

Particularly for activation function search, Ramachandran et al. [24] identified a new activation function called Switch through a combination of exhaustive search and RL, showcasing improved performance than widely used ReLU function. In [25], researchers co-design the normalization and activation functions as a single computation graph. After evolving the structure from low-level primitives, a novel structure called EvoNorm is identified that can work better on image classification, instance segmentation, image synthesis, etc.

VII. CONCLUSION

In this paper we focus on closing the accuracy gap between BNNs and their DNN counterparts by augmenting a complementary activation function prior to the sign-based binarization. In order to efficiently search in the large design space, we propose to rely on genetic algorithm for identifying novel AFs presenting improved and generalizable performance. Evaluations on different network models and datasets show that the auto-searched AFs can harvest considerable accuracy improvement (up to 2.54% on ImageNet).
REFERENCES

[1] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830, 2016.

[2] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830, 2016.

[3] Nael Fasfous, Manoj-Rohit Vempalarapu, Alexander Frickenstein, Lukas Frickenstein, and Walter Stechele. Binaryconv: Binary neural network-based covid-19 face-mask wear and positioning predictor on edge devices. arXiv preprint arXiv:2102.03456, 2021.

[4] Gang Chen, Haitao Meng, Yucheng Liang, and Kai Huang. Gpu-accelerated real-time stereo estimation with binary neural network. IEEE Transactions on Parallel and Distributed Systems, 31(12):2896–2907, 2020.

[5] Chun-Hsian Huang. An fpga-based hardware/software design using binarized neural networks for agricultural applications: A case study. IEEE Access, 9:26523–26531, 2021.

[6] Yinglan Ma, Hongyu Xiong, Zhe Hu, and Lizhuang Ma. Efficient super resolution using binarized neural network. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pages 0–0, 2019.

[7] Chao Ma, Yulan Guo, Yinjie Lei, and Wei An. Binary volumetric convolutional neural networks for 3-d object recognition. IEEE Transactions on Instrumentation and Measurement, 68(1):38–48, 2018.

[8] Ang Li, Tong Geng, Tianqi Wang, Martin Herboldt, Shuaiwen Leon Song, and Kevin Barker. Bstc: A novel binarized-soft-tensor-core design for accelerating bit-based approximated neural nets. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–30, 2019.

[9] Angus Galloway, Graham W Taylor, and Medhat Moussa. Attack on binarized neural networks. arXiv preprint arXiv:1711.00449, 2017.

[10] Yuwei Hu, Jidong Zhai, Dinghua Li, Yifan Gong, Yuhao Zhu, Wei Liu, Lei Su, and Jiangming Jin. Bitflow: Exploiting vector parallelism for binary neural networks on cpu. In 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 244–253. IEEE, 2018.

[11] Ang Li and Simon Su. Accelerating binarized neural networks via bit-tensor-cores in turing gpus. IEEE Transactions on Parallel and Distributed Systems, 32(7):1878–1891, 2020.

[12] Tong Geng, Tianqi Wang, Chunshu Wu, Chen Yang, Wei Wu, Ang Li, and Martin C Herboldt. O3bnn: An out-of-order architecture for high-performance binarized neural network inference with fine-grained pruning. In Proceedings of the ACM International Conference on Supercomputing, pages 461–472, 2019.

[13] Tong Geng, Tianqi Wang, Chunshu Wu, Chen Yang, Shuaiwen Leon Song, Ang Li, and Martin Herboldt. Lp-bnn: Ultra-low-latency bnn inference with layer parallelism. In 2019 IEEE 30th International Conference
on Application-specific Systems, Architectures and Processors (ASAP), volume 2160, pages 9–16. IEEE, 2019.

[14] Tong Geng, Ang Li, Tianqi Wang, Chunshu Wu, Yanfei Li, Runbin Shi, Wei Wu, and Martin Herfordt. O3bnn-r: An out-of-order architecture for high-performance and regularized bnn inference. IEEE Transactions on parallel and distributed systems, 32(1):199–213, 2020.

[15] Alexander G Anderson and Cory P Berg. The high-dimensional geometry of binary neural networks. arXiv preprint arXiv:1705.07199, 2017.

[16] Joseph Bethge, Christian Bartz, Haojin Yang, Ying Chen, and Christoph Meinel. Meliusnet: An improved network architecture for binary neural networks. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pages 1439–1448, 2021.

[17] Yuwen Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432, 2013.

[18] Zechun Liu, Baoyuan Wu, Wenhao Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In Proceedings of the European conference on computer vision (ECCV), pages 722–737, 2018.

[19] Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, and Vahid Partovi Nia. Bnn+: Improved binary network training. 2018.

[20] Chunlei Liu, Wenrui Ding, Xin Xia, Baochang Zhang, Jiaxin Gu, Jianzhong Hu, Rongrong Ji, and Dui Doermann. Circulant binary convolutional networks: Enhancing the performance of 1-bit cnns with circulant back propagation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2691–2699, 2019.

[21] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2250–2259, 2020.

[22] Fayez Lahoud, Radhakrishna Achanta, Pablo Marquez-Neila, and Sabine Stüstrunk. Self-binarizing networks. arXiv preprint arXiv:1902.00730, 2019.

[23] Zechun Liu, Zhiqiang Shen, Marius Savvides, and Kwang-Ting Cheng. Reactnet: Precise binary neural network with generalized activation functions. In European Conference on Computer Vision, pages 143–159. Springer, 2020.

[24] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. arXiv preprint arXiv:1710.05941, 2017.

[25] Hanxiao Liu, Andrew Brock, Karen Simonyan, and Quoc V Le. Evolving normalization-activation layers. arXiv preprint arXiv:2004.02967, 2020.

[26] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055, 2018.

[27] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00493, 2015.

[28] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Franklin, and John Guttag. What is the state of neural network pruning? arXiv preprint arXiv:2003.03033, 2020.

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning, pages 448–456. PMLR, 2015.

[30] David E Goldberg and John Henry Holland. Genetic algorithms and machine learning. 1988.

[31] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. Multimedia Tools and Applications, pages 1–36, 2020.

[32] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[33] Wei Tang, Gang Hua, and Liang Wang. Designing an accurate binary neural network with high accuracy? In Thirty-First AAAI conference on artificial intelligence, 2017.

[34] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160, 2016.

[35] Zechun Liu, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In Advances in Neural Information Processing Systems, pages 345–353, 2017.

[36] Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, and Vahid Partovi Nia. Bnn+: Improved binary network training. arXiv preprint arXiv:1812.11800, 2018.

[37] Mohammad Ghassenzadeh, Mohammad Samragh, and Farinaz Koushanfar. Rbnet: Residual binarized neural network. In 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pages 57–64. IEEE, 2018.