BGADAM: Boosting based Genetic-Evolutionary ADAM for Neural Network Optimization

Jiyang Bai, Yuxiang Ren, Jiawei Zhang
Florida State University
bai@cs.fsu.edu, yuxiang@ifmlab.org, jiawei@ifmlab.org

***Abstract*—For various optimization methods, gradient descent-based algorithms can achieve outstanding performance and have been widely used in various tasks. Among those commonly used algorithms, ADAM owns many advantages such as fast convergence with both the momentum term and the adaptive learning rate. However, since the loss functions of most deep neural networks are non-convex, ADAM also shares the drawback of getting stuck in local optima easily. To resolve such a problem, the idea of combining genetic algorithm with base learners is introduced to rediscover the best solutions. Nonetheless, from our analysis, the idea of combining genetic algorithm with a batch of base learners still has its shortcomings. The effectiveness of genetic algorithm can hardly be guaranteed if the unit models converge to close or the same solutions. To resolve this problem and further maximize the advantages of genetic algorithm with base learners, we propose to implement the boosting strategy for input model training, which can subsequently improve the effectiveness of genetic algorithm. In this paper, we introduce a novel optimization algorithm, namely Boosting based Genetic ADAM (BGADAM). With both theoretic analysis and empirical experiments, we will show that adding the boosting strategy into the BGADAM model can help models jump out the local optima and converge to better solutions.**

I. INTRODUCTION

Deep learning models have achieved impressive success in many areas, which include the computer vision [18], natural language processing [11], [31] and recently appeared graph neural networks [9], [27]. By owning a large number of variables and non-linear functions (e.g., the Relu function [22]), deep learning models can fit extremely complex data and learn the hidden representations of them. To let the deep learning models achieve outstanding performance, one of the most important procedures is training the models with appropriate optimization algorithms. During the training process of deep learning models, the essence is to search for the global optima of the loss functions. This process can be represented by

$$\min_{\mathbf{w}\in\mathcal{W}} f(\mathbf{X}, \mathbf{y}; \mathbf{w}) \tag{1}$$

Here, $f(\cdot, \cdot; \mathbf{w})$ denotes the loss function with variables $\mathbf{w} \in \mathcal{W}$, where $\mathcal{W}$ is the solution space of variables; $\mathbf{X}$ and $\mathbf{y}$ denote the features and labels of the training data respectively. Up to now, the most commonly used optimization algorithms are based on the gradient descent, for example the stochastic gradient descent (SGD), SGD with momentum [24], AdaGrad [4], RMSPROP [30] and ADAM [15]. Among these algorithms, ADAM is the most widely used one and has been proven to be powerful in various tasks thanks to its fast convergence rate. However, ADAM still has the common drawback of easily converging to the local optima and getting stuck into it, especially when dealing with the non-convex problems [7]. To solve the problem of converging to the local optima, the idea of integrating genetic algorithm and multiple learners have been proposed such as GADAM [37], which can combine the advantages of both the genetic algorithm and the ADAM. During the training process, initially multiple input models are trained simultaneously. From these input models, a batch of (parent) model pairs are selected, and genetic algorithm is implemented to the variables of these parent models after they converge to the local optima. Applying the genetic algorithm can help the trained parent models "jump out" of local optimal points when dealing with the non-convex problem. The "jump out" operation is implemented through reorganizing the variables of the parent models. Even though genetic algorithm with multiple learners has the potential capability of approaching more optimized solutions, the actual effect still can not be guaranteed. If most of the input models, or parent models converge to the close or even the same local optima, the genetic algorithm operation might become meaningless since the output models of the variables reorganizing still locate at the local optima. We denote this phenomenon of failing to jump out of local optima as the "local sticking". To avoid the "local sticking" problems and increase the capability of converging to better solutions, one concrete idea is to let the parent models converge to divergent local optima. In this way, the genetic algorithm is more likely to generate models with variables at different locations and subsequently increase the possibility of converging to better solutions and achieving better performance.

In this paper, we will propose a new optimization algorithm, namely **B**oosting based **G**enetic **ADAM** (BGADAM) to solve the "local sticking" problem mentioned above and further achieve better performance compared with GADAM algorithm. With the support of the boosting strategy, BGADAM can improve the performance of the optimization by differentiating the parent models, and meanwhile guarantee the convergence of the algorithm theoretically. Boosting strategy [6], [38] utilizes the interactions among base learners to accomplish a more effective model training. The interaction is achieved by redistributing the dataset to modify the training set for each input model. In each training iteration, the boosting strategy samples training data from the training dataset with replacement subject to different weights. After the training

Table I
NOTATIONS AND TERMINOLOGIES DEFINITIONS

Notation	Definition
m	Number of samples in training set
$\mathcal{D}_j$	Training set for j_{th} model
$\mathbf{z}$	Weight vector of training samples
M	Input model
$\bar{M}$	Trained input model
$\bar{\mathbf{w}}_i$	Variables of model $\bar{M}_i$
C	Child model
$\mathbf{w}_i$	Variables of child model C_i
N	New generation model
ϵ	Training error rate
g	Number of input models in each generation
K	Number of generations
$G^{(k)}$	k_{th} generation

process of one input model, the weights of all the data samples will be updated according to the classification results of this model. Based on the updated weights, the training set will be sampled under the updated distribution, and newly sampled training data will be assigned to the next input model. Meanwhile, this newly sampled dataset inclines to contain more data misclassified by the previous input model. In this way, by proposing the boosting strategy, different parent models will be trained with different training examples. In other words, the parent models trained by different training set will subsequently converge to divergent local optima with a higher probability. Therefore, BGADAM can effectively diversify the learned parent models, which can further improve the learning performance.

The following part of the paper is organized as follows. In Section II, we will talk about some related works. In Section III, we will cover more details about our proposed BGADAM algorithm, whose effectiveness will be analyzed in Section IV, and performance will be tested with extensive experiments in Section V. Finally, we will conclude this paper in Section VI.

II. RELATED WORKS

Deep Learning Models: Deep learning models have achieved state-of-the-art results in recent years, whose representative examples include feed-forward deep neural networks (DNN) [25], [33], convolutional neural networks (CNN) [18] [19], and recurrent neural networks (RNN) [11], [31]. DNN models mainly consist of several layers of the linear transforms and the non-linear activation functions. The combination of these linear and non-linear transforms between input features and outputs enables the DNN to fit more complex patterns hidden in the training data. Inspired by the great success of DNN models, more attention had been focused on transplanting the deep model ideas onto other types of data such as the image type and the sequential type data. To deal with the image data, CNN has been proposed and already shown outstanding performance on various computer vision tasks; for the sequential data, the RNN model came up. RNN [2], [20] models are connectionist models with the ability to pass information across sequence elements while conducting sequential data one element at a time. Besides, there also exist many other types of deep learning models to tackle different types of data, e.g., the graph neural networks [9], [10], [13], [27], graph convolutional networks [16] and graph attention networks [32] to deal with graph type data; the Generative Adversarial Networks (GAN) [8] to train discriminative models and generative models, etc.

Genetic Algorithm: Genetic algorithm [1], [12], [14], [21], [34] is a family of computational algorithms inspired by creature evolution in the natural environment. It encodes a potential solution to a specific problem on chromosome-like data structures and applies recombination (reorganization) operators to these structures [35]. This process is also called crossover [29]. Besides, genetic algorithm also adds random mutations on the solutions to mimic the natural world's real gene mutation process. In this way, for some models stuck in local optimal points, genetic algorithm can help them jump out by switching specific variables. During the generating process, the above crossover and mutation operations are implemented to a bunch of models, which are called the population set. After that, the models with the worst performance (the fitness score) will be eliminated. Finally, individuals with relatively better performance are retained.

Optimization Algorithms: The main purpose of the optimization algorithms is to closely approach the best solution of the loss functions during the backpropagation. So far most commonly used optimizers are based on the gradient descent [24], such as the stochastic gradient descent (SGD), the AdaGrad [4], the RMSPROP [30] and the ADAM [15]. SGD performs a variable update for each training example $\mathbf{X}[i,:]$ and label $\mathbf{y}[i]$, it can be expressed as:

$$\mathbf{w} = \mathbf{w} - \eta \cdot \nabla_{\mathbf{w}} f(\mathbf{X}[i,:], \mathbf{y}[i]) \tag{2}$$

where η is the learning rate and $\nabla_{\mathbf{w}}$ is the derivative of the loss function regarding variable $\mathbf{w}$. The advantages of SGD include fast speed and getting rid of redundancy [24]. SGD with momentum [23], [28] is a method that helps accelerate SGD in the relevant directions. SGD with momentum updates variables with the following equations:

$$\begin{aligned} \mathbf{v}_t &= \gamma \cdot \mathbf{v}_{t-1} + \eta \cdot \nabla_{\mathbf{w}} f(\mathbf{w}) \\ \mathbf{w}_t &= \mathbf{w}_t - \mathbf{v}_t \end{aligned} \tag{3}$$

where γ is the momentum term weight. The momentum term accelerates variable updates for dimensions whose gradients are in the same direction as historical gradients and decelerates updates for dimensions whose gradients are the reverse. Therefore its convergence process will be faster. However, the unified learning rate for all variables can lead to some problems. If variables have different scales, we should not update them with the same rate. Some variant algorithms have been proposed to solve this problem, such as the AdaGrad and the RMSPROP. AdaGrad [4] adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. RMSPROP [30] is a refined version of the AdaGrad that applies the moving average of the historical gradients to adapt the learning rates.

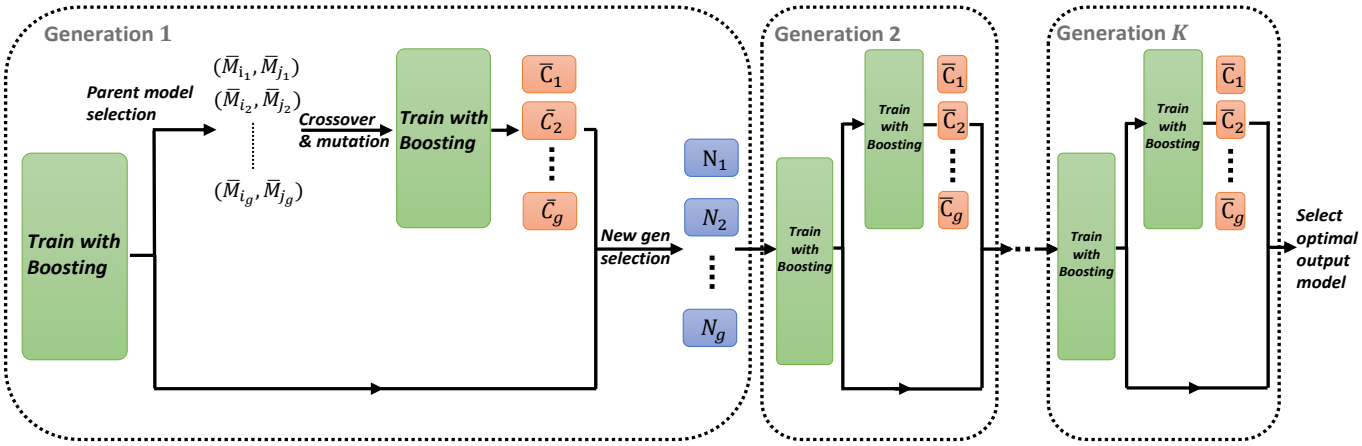


Figure 1. BGADAM structure

ADAM [15] is proposed based on the SGD and the momentum, which computes individual adaptive learning rates for different variables. Similar to the SGD with momentum and the RMSPROP, ADAM records the first-order momentum $\mathbf{m}_t$ and the second-order momentum $\mathbf{v}_t$ of the gradients and computes the bias-corrected version of them respectively.

Ensemble Learning: Ensemble learning [3], [36], [38] tries to train multiple learners to solve the same problem. In contrast to ordinary learning approaches that attempt to construct one model from the training data, ensemble methods try to construct a set of learners and combine them. The commonly used ensemble methods include boosting and bagging. Boosting [6], [38] generally creates a batch of weak learners, and each learner is trained based on the sampled training data under the distribution decided by the previous learner. In other words, each learner can focus more on the data samples being misclassified by the previous learner. Bagging adopts the bootstrap distribution for generating different base learners. It applies bootstrap sampling [5] to obtain the data subsets for training the base learners.

III. BGADAM LEARNING ARCHITECTURE

In this section, we will introduce more details about the BGADAM optimizer. The architecture of BGADAM is illustrated in Figure 1. In the architecture, the green frame denotes the boosting strategy based ADAM, whose detailed structure is provided in Figure 2. Next, we will provide more detailed descriptions of both the architecture and the involved learning components. The notations and terminologies we have employed in this paper are presented in Table I.

BGADAM adopts the boosting strategy to increase the diversity of the input models, which can further resolve the "local sticking" problem of the parent models. The learning process of BGADAM involves multiple generations, which can be denoted as $G^{(1)}, G^{(2)}, \cdots, G^{(K)}$ respectively (K is the total generation count). In each generation, g input models will be trained along with the boosting based ADAM at first; then a batch of trained input models will be selected to compose the parent model pairs; after that, the child models will evolve according to the genetic algorithm; finally, among both the input models and the generated child models, the same number (e.g., g) of new generation models will be selected as the output models of the current generation and as the input models of the next generation. After a predefined number of iterations (or some converging criteria has been satisfied), the training will stop, and the best new-generation models will become the output model. We will refer to each procedure in the following parts.

A. Training Input Models with Boosting Strategy

In each generation $G^{(k)}, k \in \{1, 2, \ldots, K\}$, g input models $\{M_1, M_2, \ldots, M_g\}$ are trained with boosting based ADAM. More detailed information about the boosting based ADAM training will be covered in Section III-E. The essence of boosting strategy is to train multiple models with collaborating policy [38], then combine these models to induce the final results. For each model, the collaborating is achieved by sampling different sub training sets according to the previous model's performance. Specifically, during the training process of g input models, the same size g training sets $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_g\}$ will be generated. For each training set $\mathcal{D}_j$, its data samples are sampled from the entire training set, under the distribution determined by the performance of model M_{j-1}. For our classification task, the data samples having been misclassified by M_{j-1} will be assigned higher probabilities to be sampled in $\mathcal{D}_j$. In this way, boosting strategy allocates each input model with different training data. Since the different training sets will allow the input model variables to be updated by ADAM differently, these g input models are more likely to converge to different local optimal points. Such a characteristic gives the following genetic algorithm more advantages, and we will discuss this in the later parts. Moreover, boosting strategy not only allows each input model to learn information hidden in data, but also refers to previously trained models. We also call this process the interaction among the input models. After the training process of g input models, $\{\bar{M}_1, \bar{M}_2, \ldots, \bar{M}_g\}$ refers to the trained input models, and then parent model pairs $\{(\bar{M}_{i_1}, \bar{M}_{j_1}), \ldots, (\bar{M}_{i_g}, \bar{M}_{j_g})\}, i, j \in \{1, 2, \ldots, g\}$ will be chosen from them based on special mechanism. Next, the

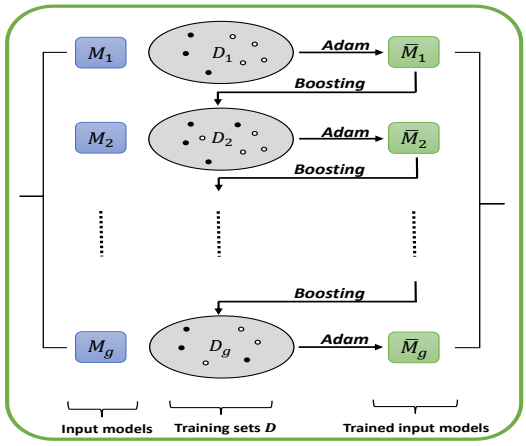


Figure 2. Boosting based ADAM

child models will evolve from them by the genetic algorithm, which essentially helps the parent models jump out of the local minimum and discover global optima to achieve better learning performance.

B. Parent Model Selection and Fitness Evaluation

Similar to the evolutionary laws in the natural world, where individuals with better performance own higher possibilities to become the parents, our proposed BGADAM also inclines to choose input models with better performance to participate in the evolving process. Before the evolving operations, from the input models trained by boosting based ADAM, a batch of parent model pairs will be selected according to the fitness evaluation. The fitness evaluation aims to evaluate each trained input model by computing their fitness scores for the learning setting, and a better fitness score means better performance of the target model. Correspondingly, input models with better fitness scores would be more likely to be selected as the parent models lately. Thus, the function of fitness evaluation is akin to evolutionary laws.

In the k_{th} generation $G^{(k)}$, for each trained input model $\bar{M}_i$, we calculate its loss on validation set $\mathcal{V}$ (sampled from the training set) as the fitness score, which can be denoted as follows:

$$\mathbf{l}[i] = \sum_{(\mathbf{X}[j,:],\mathbf{y}[j])\in\mathcal{V}} l(\mathbf{X}[j,:],\mathbf{y}[j];\mathbf{w}_i) \tag{4}$$

where $l(\cdot,\cdot;)$ is the cross-entropy loss function, $\mathbf{w}_i$ denotes the variables of model $\bar{M}_i$, and vector $\mathbf{l} \in \mathbb{R}^g$ contains the computed fitness scores of all these g unit models in the current generation. By calculating the loss of input models on the validation set, we can judge these input models: the less the loss is, the better performance model will have in general. However, directly using the loss terms for parent model selection may not work well because the range of $\mathbf{l}[i]$ might vary in a large scale. So for each $\mathbf{l}[i]$, we utilize the normalized $\hat{\mathbf{l}}[i] = \frac{\mathbf{l}[i]-min(\mathbf{l})}{max(\mathbf{l})-min(\mathbf{l})}$ to calculate the selecting probability for input model $\bar{M}_i$ to become the parent model as follows:

$$\mathbf{p}[i] = \frac{\exp(-\hat{\mathbf{l}}[i])}{\sum_{j=1}^{g}\exp(-\hat{\mathbf{l}}[j])} \tag{5}$$

where $\mathbf{P} \in \mathbb{R}^g$. According to the probability vector $\mathbf{p}$, g different model pairs $\{(\bar{M}_{i_1},\bar{M}_{j_1}),\ldots,(\bar{M}_{i_g},\bar{M}_{j_g})\}$ will be sampled with replacement as parent models.

C. Crossover and Mutation

The genetic laws of individuals' genes are common in the natural world to obtain better offspring. Inspired by the recombination and mutation rules of genes in genetic laws, the genetic algorithm also carries out similar operations, named crossover and mutation separately. The genetic algorithm includes crossover and mutation operations for creating child models generation, where the model variables are treated as the chromosome of gene, respectively.

- **Crossover**: Given a parent model pair $(\bar{M}_{i_n},\bar{M}_{j_n})$ with respective variable vectors $\bar{\mathbf{w}}_{i_n}$ and $\bar{\mathbf{w}}_{j_n}$, crossover generates their child model with variable vector $\mathbf{w}_n$, whose entry $\mathbf{w}_n[h]$ can be represented as

$$\begin{aligned}\mathbf{w}_n[h] = {} & \mathbb{1}(\textbf{rand} \le threshold)\cdot\bar{\mathbf{w}}_{i_n}[h] \\ & + \mathbb{1}(\textbf{rand} > threshold)\cdot\bar{\mathbf{w}}_{j_n}[h]\end{aligned} \tag{6}$$

  In the Equation 6, $\mathbb{1}(\cdot)$ is the indicator function, **rand** is a random number uniformly distributed in $[0,1]$, $threshold = \frac{\mathbf{p}[i_n]}{\mathbf{p}[i_n]+\mathbf{p}[j_n]}$ and $\bar{\mathbf{w}}_{i_n}[h]$, $\bar{\mathbf{w}}_{j_n}[h]$ denotes the h_{th} variable of the parent models $\bar{M}_{i_n}$ and $\bar{M}_{j_n}$ respectively. By applying the evolving rule in Equation 6 to each parent model pairs, the model with relatively better performance would retain more its own variables.
- **Mutation**: Similar to crossover, the mutation can be expressed as

$$\begin{aligned}\mathbf{w}_n[h] = {} & \mathbb{1}(\textbf{rand} \le p_m)\cdot\mathcal{N}(0,0.01) \\ & + \mathbb{1}(\textbf{rand} > p_m)\cdot\mathbf{w}_n[h]\end{aligned} \tag{7}$$

  where $p_m = p\cdot(1-\mathbf{p}[i_n]-\mathbf{p}[j_n])$ and p is a pre-defined value (e.g., $p = 0.01$), $\mathcal{N}(0,0.01)$ is a random number sampled from the normal distribution with 0 mean and 0.01 variance. Equation 7 can make sure the child models of parents with higher $\mathbf{p}$ values to be less likely to mutate. In other words, the mutaton mechanism in Equation 7 inclines to preserve the variables inherited from parents that having better performance.

Formally, via crossover and mutation, we can represent the generated g child models as $\{C_1,C_2,\cdots,C_g\}$, which will be further trained via the boosting based ADAM. Those trained child models can be denoted as $\{\bar{C}_1,\bar{C}_2,\cdots,\bar{C}_g\}$ respectively.

D. New Generation Selection

The selection of the new generation of models also follows the evolutionary law: the best and strongest individuals will survive. The new generation models are selected among both the trained child models and input models. In this way, it is certain that the selected new generation of models will not deteriorate compared with the input models. This idea also conforms to the rule of survival of the fittest in natural selection. For each model in $\{\bar{M}_1,\bar{M}_2,\ldots,\bar{M}_g\}\bigcup\{\bar{C}_1,\bar{C}_2,\ldots,\bar{C}_g\}$, its loss on the validation set will be recorded. Finally, we will select the

Algorithm 1: BGADAM

Input: Input models $\{M_1, M_2, \ldots, M_g\}$; training feature $\mathbf{X}$; training label $\mathbf{y}$
Output: Final model
for $i = 1, 2, \ldots, K$ **do**
  $\mathbf{z} = [\frac{1}{m}, \frac{1}{m}, \ldots, \frac{1}{m}]$ and $\mathcal{D}_1 = (\mathbf{X}, \mathbf{y})$;
  $\mathbf{l} = []$; /* To record loss. */
  for $j = 1, 2, \ldots, g$ **do**
    Train M_j as $\bar{M}_j$ using $\mathcal{D}_j$ dataset;
    $\mathbf{l}[j]$ = The loss of $\bar{M}_j$ on $\mathcal{V}$;
    Update $\mathbf{z}$ by Equation (9) and Equation (10);
    Produce $\mathcal{D}_{j+1}$ by sampling subject to $\mathbf{z}$;
    Compute $\mathbf{p}[j]$ according to Equation (5);
  end
  for $h = 1, 2, \ldots, g$ **do**
    Select $\bar{M}_{i_k}$ and $\bar{M}_{j_k}$ according to $\mathbf{p}$;
    Generate C_k by Equations (6) and (7);
    Train C_h as $\bar{C}_h$ using $\mathcal{D}_h$;
    $\mathbf{l}[h]$ = The loss of $\bar{C}_h$ on $\mathcal{V}$;
  end
  for $l = 1, 2, \ldots, g$ **do**
    Select model with smallest value in $\mathbf{l}$ as N_l;
    Delete the smallest value in $\mathbf{l}$;
  end
end
return N_1 model

top g models with the smallest losses as the new generation models, which can be denoted as $\{N_1, N_2, \ldots, N_g\}$ and they will also serve as the input models for the next generation. Such an iterative model learning and the evolving process will continue until finally converging, and the optimal output model (with the lowest loss on validation set) in the last generation will be selected as the final output model. We will also give the convergence analysis of BGADAM in Section IV-B and briefly mention the convergence results in Section V-C.

E. Boosting Strategy based ADAM

Boosting refers to a family of algorithms that can convert weak learners to strong learners. The core idea of boosting is to correct the mistakes made by previous learners (models) and let the current model focus more on the data examples being misclassified by prior models. So during the training process of the current model, the training set will be different from those for prior models. In this section, we will talk more about the boosting based ADAM learning algorithm adopted in BGADAM.

1) Motivation: Prior to adding the boosting strategy, genetic algorithm is only combined with ADAM to rediscover the best solutions of the models. However, we observe that genetic algorithm cannot really resolve such a problem completely. For instance, assume that after training by ADAM, all input models converge to the same or close local optima of the loss function, which can be eshibited by Figure 3. Under such circumstances, the genetic algorithm (crossover and mutation) can hardly help child models jump out of local optima because the parent models are too similar on the learned model variables. Assume the variables of $\bar{M}_{i_n}$ has $\bar{\mathbf{w}}_{i_n} \in U(\mathbf{o}, \delta) = \{\mathbf{w} | \, \|\mathbf{w} - \mathbf{o}\|_2 \leq \delta\}, \forall i_n \in \{1, 2, \ldots, g\}$, where $\mathbf{o}$ represents a local optima and δ is a default small value. Here, $U(\mathbf{o}, \delta)$ represents a neighborhood region around $\mathbf{o}$ in the variable space. Then for parent model pairs $\bar{M}_{i_n}$ and $\bar{M}_{j_n}$ located in $U(\mathbf{o}, \delta)$, we can compute the variables of their child model C_n as $\mathbf{w}_n[h] = \beta \cdot \bar{\mathbf{w}}_{i_n}[h] + (1 - \beta) \cdot \bar{\mathbf{w}}_{j_n}[h]$, where $\beta \in \{0, 1\}$, which is exactly the crossover operation in Section III-C. Then we have

$$\begin{aligned} \|\mathbf{w}_n - \mathbf{o}\|_2 &\leq \|\mathbf{w}_n - \bar{\mathbf{w}}_{i_n}\|_2 + \|\bar{\mathbf{w}}_{i_n} - \mathbf{o}\|_2 \\ &\leq \|\mathbf{w}_n - \bar{\mathbf{w}}_{i_n}\|_2 + \delta \\ &\leq \|\bar{\mathbf{w}}_{i_n} - \bar{\mathbf{w}}_{j_n}\|_2 + \delta \\ &\leq \|\bar{\mathbf{w}}_{i_n} - \mathbf{o}\|_2 + \|\bar{\mathbf{w}}_{j_n} - \mathbf{o}\|_2 + \delta \\ &\leq 3\delta \end{aligned} \tag{8}$$

and observe that after the crossover, C_n still locates in the neighborhood region of the local optima $\mathbf{o}$. This phenomenon is exactly the "local sticking" situation we have mentioned in the Introduction section. It is true that mutation operation can assist in jumping out of the local optima. However, according to Equation (7) the parent models with a relatively small loss on the validation set will lead to a much lower mutation rate p_m for their child models. Thus mutation cannot solve the local sticking problem thoroughly. On the other hand, boosting can solve this problem by creating different training sets for g input models, which means to let models converge to different local optima. So there will be more gaps and divergence among the learned parent models, which can potentially enhance the advantages of the genetic algorithm and base learners to achieve better solutions.

2) Boosting Strategy: The overall framework of boosting based ADAM is shown in Figure 2. To explicitly explain the boosting strategy, first, we have to introduce the weight $\mathbf{z} \in \mathbb{R}^m$ for all the training instances (here, m denotes the size of the total training set). For each input model's training, one sub training set is sampled subject to the current weight vector $\mathbf{z}$. In other words, the value of $\mathbf{z}[i]$ represents the probability that i_{th} training instance will be sampled. Initially, $\mathbf{z}[i] = \frac{1}{m}, \forall i \in \{1, 2, \cdots, m\}$, i.e., all the instances will be sampled with an equivalent chance. We train the first input model M_1 by ADAM with a sub training set sampled from the entire training set subject to the weight vector $\mathbf{z}$. After this training process, we can denote the trained first model as $\bar{M}_1$ and record its prediction results on the entire training set to update $\mathbf{z}$ by

$$\mathbf{z}[i] = \mathbf{z}[i] \times \begin{cases} \exp(-\alpha_j) & \text{if } \bar{M}_j(\mathbf{X}[i,:]) = \mathbf{y}[i]; \\ \exp(\alpha_j) & \text{if } \bar{M}_j(\mathbf{X}[i,:]) \neq \mathbf{y}[i]. \end{cases} \tag{9}$$

and

$$\mathbf{z}[i] = \mathbf{z}[i] / sum(\mathbf{z}) \tag{10}$$

where $\bar{M}_j(\mathbf{X}[i,:])$ denotes the output of model $\bar{M}_j$ on instance $\mathbf{X}[i,:]$, $\alpha_j = \frac{1}{2} log(\frac{1-\epsilon_j}{\epsilon_j})$ and $\mathbf{y}[i]$ represents the true label of i_{th} training example. Here, for the first input model $\bar{M}_1$, $j = 1$

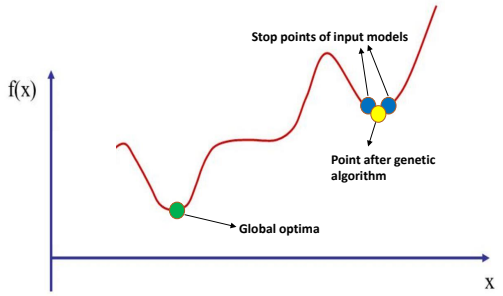


Figure 3. Local sticking situation

and $\epsilon_1 = P_{(\mathbf{X}[i,:],\mathbf{y}[i])\sim\mathcal{X}}(\bar{M}_1(\mathbf{X}(i,:)) \neq \mathbf{y}[i])$. What need to be mentioned is that $\epsilon_j < 0.5$ should be satisfied because we deliberately design the weight of training sample $\mathbf{X}[i,:]$ having $\bar{M}_1(\mathbf{X}[i,:]) \neq \mathbf{y}[i]$ to increase, so α_1 will be larger than 0. In this way, sample $\mathbf{X}[i,:]$ will be more likely to appear in next sub training set and the next model will focus more on correctly classifying it. The Equation (10) is to regulate $\mathbf{z}$ as a probability distribution. When training the next model by ADAM, another sub training set will be sampled based on the new weight vector $\mathbf{z}$. By using the weight vector $\mathbf{z}$, the boosting strategy successfully assigns entire training set with different distributions (redistribution of the training set).

The boosting strategy can also be regarded as the interaction among models. By applying boosting we achieve the interaction among models through updating the sub training set for each input model. What is more, this interactive training essentially makes these g input models diverse. Since different training sets will lead to various loss functions and gradients, g input models are more likely to converge to other local optimal points. This characteristic can give genetic algorithm more advantages, and that is the core idea we implement the boosting strategy into the BGADAM.

IV. THEORETIC ANALYSIS

In this section, we will show that the proposed BGADAM algorithm can guarantee the learning effectiveness; meanwhile, keep converging when the number of generations increases.

A. Effectiveness Guarantee

According to [6], it has been proven that the error ϵ_i of the trained input model $\bar{M}_i$ can be bounded by

$$2^g \prod_{i=1}^{g} \sqrt{\epsilon_i(1-\epsilon_i)} \leq \exp(-2\sum_{i=1}^{g}\gamma_i^2) \tag{11}$$

where $\gamma_i = 0.5 - \epsilon_i$. Let ϵ_{min} be the error of input model with the smallest loss, we have

$$2^g \epsilon_{min}^g \leq 2^g(\epsilon_{min}(1-\epsilon_{min}))^{\frac{g}{2}} \leq 2^g \prod_{i=1}^{g} \sqrt{\epsilon_i(1-\epsilon_i)} \tag{12}$$

Therefore $\epsilon_{min} \leq \frac{1}{2}\exp(-2\sum_{i=1}^{g}\gamma_i^2/g)$. The error of the model we finally select as the output model in the last generation has a relatively tight upper bound.

B. Convergence Analysis

In our proposed method, we effectively integrate boosting based ADAM learning algorithm into BGADAM as shown in Figure 1 and Algorithm 1. For each generation $G^{(k)}$, while training g input models, we no longer use the same training set for every model; instead, the boosting based learning algorithm shown in Figure 2 is added to the training process. For M_i, the training set $\mathcal{D}_i$ (denoted by a gray oval) is applied. A remark to be added here, the training process of the child models also applies the boosting strategy as indicated in the algorithm architecture. With the growth of the k, we will show that the loss of input models can finally converge.

The BGADAM algorithm will also converge in a finite number of generations, along with the loss of the input models decreasing continuously. For the i_{th} input model $M_i^{(k)}$ in the generation $G^{(k)}$, the variables learned by boosting based ADAM will converge after training, which means $\bar{\mathbf{loss}}_i^{(k)} \leq \mathbf{loss}_i^{(k)}$, where $\mathbf{loss}_i^{(k)}$ and $\bar{\mathbf{loss}}_i^{(k)}$ denote the introduced loss of the model M_i and $\bar{M}_i$ before and after training respectively in generation $G^{(k)}$. Since in the k_{th} generation, the top g models are selected among both the trained input models and the generated child models as the output models, the performance of those chosen g models will not be worse than the input models. In other words, $\mathbf{loss}_i^{(k+1)} \leq \bar{\mathbf{loss}}_i^{(k)}, \forall i \in \{1,2,\ldots,g\}$. Therefore we have

$$\mathbf{loss}^{(k+1)} = \sum_{i=1}^{g}\mathbf{loss}_i^{(k+1)} \leq \sum_{i=1}^{g}\bar{\mathbf{loss}}_i^{(k)} \leq \sum_{i=1}^{g}\mathbf{loss}_i^{(k)} = \mathbf{loss}^{(k)} \tag{13}$$

With the training process going on, the loss of models in each generation will continuously decrease when K goes up, and the BGADAM will finally converge. The convergence results will be exhibited in Section V-C.

V. NUMERICAL EXPERIMENTS

To test the effectiveness and advantages of the proposed BGADAM algorithm, extensive experiments have been conducted on real-world datasets. In this section, we will first describe the datasets we have used in the experiment, and then introduce the experimental settings in detail. Finally, we will exhibit the experimental results together with detailed descriptions and give the parameter sensitivity analysis.

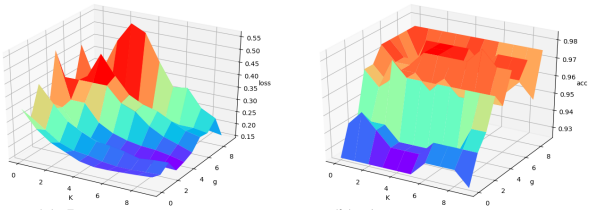
A. Dataset Description

- **ORL Dataset**: The ORL [26] dataset consists of face images of 40 people, each person has ten images. Each image is in size of 112×92.
- **MNIST Dataset**: The MNIST [19] dataset includes 60,000 training samples and 10,000 testing samples, where each sample is a 28×28 image of hand-written numbers from 0 to 9.
- **CIFAR-10 Dataset**: The CIFAR-10 [17] dataset consists of 60000 32×32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset has no augmentation operation.

Table II
EXPERIMENT RESULTS ON ALL DATASETS

Comparison Methods	Datasets					
	ORL-7*		MNIST		CIFAR-10	
	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
BGADAM	0.967	**0.0947**	**0.9917**	**0.045**	**0.6358**	1.2346
GADAM	**0.975**	0.1658	0.9911	0.076	0.6189	2.4067
ADAM	0.958	0.466	0.9905	0.0458	0.6103	**1.1742**
RMSProp	0.9417	0.2819	0.9877	0.0464	0.5978	1.1807
AdaGrad	0.9333	1.763	0.7988	1.5457	0.3292	1.6892

* ORL-7 denotes 3 images per person as test set, the rest 7 images as training set and 4 of 7 as validation set.



(a) Loss on test set (b) Accuracy on test set

Figure 4. The impact of g and K

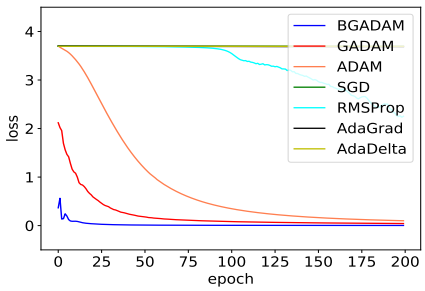


Figure 5. Convergence results of comparison algorithms

B. Experiment Settings

In this part, we will introduce the experiment settings, which covers the detailed experiment setup, comparison methods, and the evaluation metrics.

1) Experiment Setup: We use the convolutional neural network (CNN) structure models as the base model (input model). The CNN model we have built is based on the LeNet-5 [19], which has seven layers. For different datasets, the CNN models have different settings: for the ORL dataset, we use two convolutional layers with 16 and 36 feature maps of 5×5 kernels and 2×2 max-pooling layers, and a fully connected layer with 1024 neurons; for the MNIST dataset, we use LeNet-5 structure in CNN model; for the CIFAR-10 dataset, we apply three convolutional layers with 64, 128, 256 kernels respectively, and a fully connected layer having 1024 neurons. All the experiments apply Relu [22] activation function, and 0.5 dropout rate on fully connected layers. For each training process, the training batch involves 128 samples and the number of epochs satisfies traversing the entire training set around 200 times. The initialization of variables is random numbers sampled from Normal Distribution with 0 means and 0.01 standard variance. For different datasets we use different g and K in the BGADAM, and we will analyze their influence later.

2) Comparison Methods and Evaluation Metrics: To show the advantages of the BGADAM algorithm, we compare it with the most commonly used optimization algorithms, including ADAM [15], RMSProp [30], AdaGrad [4] and GADAM [37] respectively.The input model in our experiments is based on the CNN structures.

To measure the performance of the comparison methods, different metrics have been applied in this paper. We calculate both the accuracy of prediction and test loss achieved by the models trained with these different optimization algorithms on the test set.

C. Experimental Results

In this section, firstly, we will exhibit results on all the datasets, then show the convergence and hyper-parameter analysis results of the BGADAM algorithm.

1) Results on All Datasets: In Table II, we show the performance of BGADAM algorithm on several datasets compard with other baseline algorithms. The results of BGADAM in the table is with $g = 5, K = 5$. To let the total training iterations of BGADAM be identical to comparison methods (e.g., ADAM), we set the training iterations in each generation of BGADAM as the training iterations of ADAM dividing K. In this way, the total number of training iterations of BGADAM is equal to ADAM, meanwhile the overfitting problem can be potentially inhibited. From the table, we can see that the BGADAM achieves better overall performance on the test set. For the accuracy of prediction, BGADAM achieves the best results on most of the datasets, especially on the CIFAR-10 dataset. The test accuracy achieved by BGADAM is 0.6358, which is at least 2.5% larger than the accuracy obtained by ADAM, GADAM and RMSPROP, and the advantages are much more significant compared with AdaGrad; for the loss on the test set, the advantages of BGADAM are much more obvious: the BGADAM's results are less than GADAM and ADAM by almost 50 percent on all datasets. Especially for the ORL and CIFAR-10 datasets, the test loss of BGADAM is only half of GADAM. The advantages are much more significant when comparing to other algorithms such as RMSPROP and AdaGrad. The overall results demonstrate that our proposed BGADAM method does improve the performance of both genetic algorithm and ADAM by adopting the boosting strategy.

2) Convergence and Hyper-Parameter Analysis: BGADAM algorithm can converge in a finite number of generations,

Table III
HYPER-PARAMETER ANALYSIS

Different Situations	ORL-7	
	Test loss	Test acc
$K = 1, g = 10$	0.3935	0.975
$K = 10, g = 10$	0.088	0.9833

which is shown in Figure 5. The result is on the ORL-7 dataset. Due to the limited space, we will only show the analysis on the ORL dataset in this part. The hyper-parameters g and K involved in the training process may affect the convergence and final results of BGADAM. Thus we also analyze their influence. We run the experiments with $g \in \{1, 2, \ldots, 10\}$ and $K \in \{1, 2, \ldots, 10\}$ on the ORL dataset to illustrate their influence. The results are shown in Figure 4. We can notice that the test loss decreases when K increases in terms of different g numbers, and conversely, the accuracy on the test set increases. The trend can also be seen in Table III. We also find that the loss function does not change when K and g increase from 5 to 10, respectively. In other words, the proposed BGADAM algorithm can converge to an ideal solution in 5 generations with the input models size of 5 generally. That is also why we apply the setting of $g = 5$ and $K = 5$ when carrying out the final results of BGADAM. For the experiments on the other datasets, we also check the influence of K and verify specific values of K to get the final model.

VI. CONCLUSION

In this paper, we have introduced a new hybrid optimization algorithm, namely BGADAM. By combining ADAM, genetic algorithm, and boosting strategy together, BGADAM can maximize the advantages of each part by utilizing their characteristics to efficiently jump out of local optima and prevent the "local sticking" phenomenon, then further converge to better solutions. We have carried out extensive experiments on real-world datasets, and the results show that our proposed BGADAM algorithm outperforms previous optimization methods, especially for training deep neural networks.

REFERENCES

[1] Xiaodong Cui, Wei Zhang, Zoltán Tüske, and Michael Picheny. Evolutionary stochastic gradient descent for optimization of deep neural networks. In *NIPS*, 2018.

[2] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network nearning for speech recognition and related applications: An overview. In *ICASSP*, 2013.

[3] Thomas G Dietterich et al. Ensemble learning. *The handbook of brain theory and neural networks*, 2002.

[4] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

[5] Bradley Efron and R.J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, 1993.

[6] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *ICML*, 1996.

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[9] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *IJCNN*, 2005.

[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

[11] Sepp Hochreiter and Jrgen Schmidhuber. Long short-term memory. *Neural Computation*, 1994.

[12] John H. Holland. Genetic algorithms. *Science American*, 1992.

[13] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *Advances in Neural Information Processing Systems*, 2018.

[14] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. In *arXiv preprint arXiv:1711.09846*, 2017.

[15] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[16] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[17] Alex Krizhevsky. Learning multiple layers of features from tiny images. In *CRC Press*, 2009.

[18] Alex Krizhevsky and Ilya Sutskever Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[19] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-base learning applied to document recognition. In *IEEE*, 1998.

[20] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. In *preprint arXiv:1506.00019*, 2015.

[21] Ilya Loshchilov and Frank Hutter. Cma-es for hyperparameter optimization of deep neural networks. In *ICLR Workshop*, 2016.

[22] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[23] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks : The Official Journal of the International Neural Network Society*, 1999.

[24] Sebastian Ruder. An overview of gradient descent optimization algorithms. In *arXiv:1609.04747v2*, 2017.

[25] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *AISTATS*, 2009.

[26] Ferdinando Samaria and Andy Harter. Parameterisation of a stochastic model for human face identification. In *IEEE Workshop on Applications of Computer Vision*, 1994.

[27] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. In *IEEE Transactions on Neural Networks*, 2009.

[28] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[29] Dirk Thierens and David Goldberg. Convergence models of genetic algorithm selection schemes. In *ICPPSN*, 1994.

[30] Tijmen Tieleman and Geoffrey E. Hinton. Leture 6.5 rmsprop,coursera: Neural networks for machine learning. In *Tehcnical report*, 2012.

[31] Mikolov Tom, Karafit Martin, Burget Luk, ernock Jan, and Khudanpur Sanjeev. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[33] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008.

[34] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 1994.

[35] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 1997.

[36] Cha Zhang and Yunqian Ma. *Ensemble machine learning: methods and applications*. Springer, 2012.

[37] Jiawei Zhang, Limeng Cui, and Fisher Gouza. Gadam: Genetic-evolutionary adam for deep neural network optimization. In *arXiv:1805.07500*, 2018.

[38] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. CRC Press, 2012.