Future Generation Computer Systems 132 (2022) 40–50

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

DPLRS: Distributed Population Learning Rate Schedule☆

Jia Wei, Xingjun Zhang *, Zeyu Ji, Zheng Wei, Jingbo Li

School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China

ARTICLE INFO

Article history:
Received 28 June 2021
Received in revised form 16 December 2021
Accepted 5 February 2022
Available online 11 February 2022

Keywords:
Deep learning
Distributed training
Hyperparameter search
Data parallel
Population algorithm

ABSTRACT

Deep neural network models perform very brightly in the field of artificial intelligence, but their success is affected by hyperparameters, and the learning rate schedule is one of the most important hyperparameters, while the search for the learning rate schedule is often time-consuming and computationally resource-intensive. In this paper, we proposed Distributed Population Learning Rate Schedule (DPLRS) based on population joint optimization, which uses distributed data parallel deep neural network training to implement a dynamic learning rate schedule optimization strategy based on the population idea, with almost no loss of test accuracy. DPLRS is able to dynamically refine the learning rate schedule during model training instead of following the usual suboptimal strategy. We conducted experiments on typical AlexNet, VGG16, and ResNet18 using the Tianhe-3 supercomputing prototype. The results illustrate that using DPLRS to dynamically update the learning rate can greatly reduce the searching time of the learning rate schedule and meanwhile, can ensure the close performance with the latest population hyperparameter algorithm. Also, In our experiments, DPLRS lead to 123.85x speedup maximum, which prove the effectiveness and robustness of DPLRS.

1. Introduction

In recent years, deep learning, as an important branch of artificial intelligence, has achieved world-renowned results [1] in many areas such as image recognition [2], natural language processing [3], and complex decision making [4], which often benefit from a refined set of hyperparameters chosen.

Researchers have devoted themselves to improving the final training performance of deep learning (e.g. accuracy of image classification, BLEU scores in natural language processing, etc.) by modifying hyperparameters such as dataset style, model structure, loss function, and optimizer parameters in conjunction with specific application requirements. Sener et al. [5] and Yin et al. [6] have used their respective improved active learning methods to select the optimal range of training data; He et al. [2] and Devlin et al. [7] proposed ResNet and BERT models that caused far-reaching impact in the fields of image classification and natural language processing, respectively; Lin et al. [8] and Liu et al. [9] proposed Focal and SphereFace loss functions on the basis of cross-entropy loss function produced significant improvement on model training results in the fields of target detection and image recognition.

In this context, choosing the appropriate learning rate schedule is a very critical hyperparameter that affects the model training results [10], because most optimizers have difficulty in traversing a non-convex, non-smooth loss function space with numerous local minima and potential saddle points [11–13]. In order to converge to a model with good generalization capability, the learning rate schedule often requires extensive and trivial tuning [10,14]. In addition, the widely used small-batch gradient descent with uncertainty also makes the search for learning rate tables difficult to be defined as a good optimization problem and solved by standard optimization methods. Static or dynamic built-in optimizers or predefined learning rate tables based on Bayesian or massively parallel search are widely used [15,16]. Static optimizers tend to have simpler rules, such as exponential and cosine, which cannot be aligned with nonsmooth loss functions, while dynamic optimizers, such as Adam [17] and Adadelta [18], are extended from convex optimization requiring strong assumptions to guarantee convergence. Therefore, using a variety of learning rate table search methods can get better results than using a fixed optimizer. However, using Bayesian-based search methods requires the constant generation and utilization of prior knowledge, and thus has a significant serial dependence, requiring a lot of time to perform multiple serial searches repeatedly; while large-scale grid and stochastic searches require the use of a large number of computational resources, and during the training process, the information of different nodes generally cannot be shared during the training process. There are recent studies based on the idea of the population for learning rate

☆ This work was supported by the National Natural Science Foundation of China (62172327).

* Corresponding author.
E-mail address: xjzhang@xjtu.edu.cn (X. Zhang).

https://doi.org/10.1016/j.future.2022.02.001
0167-739X/

search, which allows multiple nodes trained independently to exchange parameters and learning rates under specific conditions [16,19,20], but these methods still require a single node to complete the entire model training, with huge time overhead.

Therefore, we propose Distributed Population Learning Rate Schedule (DPLRS) for Deep Learning. This method uses the idea of population algorithm to dynamically and adaptively optimize the learning rate during the training of parallel deep neural networks with distributed data, and the learning rate is adjusted during the distributed training process, and each node only needs to complete 1/n (n is the number of nodes) of the whole dataset, and the model training time is improved nearly n times with the accuracy of the test set similar to the latest PBT [16] algorithm.

The main contributions of this work are summarized as follows:

1. We proposed a new distributed deep learning algorithm DPLRS based on particle swarm algorithm and genetic optimization algorithm for a distributed computing environment.
2. We efficiently couple the learning rate schedule search with the neural network model training process, solving the problems of high computational overhead and serious serial dependency of traditional search algorithms.
3. We achieve a significant reduction in overall training time while achieving comparable training accuracy with the state-of-the-art population search PBT algorithm.
4. We verified the effectiveness and reliability of the algorithms on the Cifar10 and ImageNet datasets using the current classical AlexNet, VGG16 and ResNet18 models in image recognition on the Tianhe-3 supercomputing prototype platform.

2. Related work

This section focuses on the development of deep neural network hyperparameter search algorithm and neural network distributed training strategy related to DPLRS algorithm.

2.1. Hyperparameter search

With the deepening of neural network theory and the accumulation of social production experience, the success of a neural network model suffers from the learning framework composed of model structure, data presentation, model details optimization, and so on. Each component in the learning framework is controlled by many hyperparameters, which will affect the learning process and must be adjusted properly to fully demonstrate the network performance [21–23].

At present, there are two main ways of hyperparameter search, parallel search, and serial search, which make a trade-off between consuming the number of computing resources and obtaining the best results [24–26]. Parallel search starts multiple computing nodes at the same time, uses a lot of computing resources to independently perform several complete neural network training processes, and there is less data interaction between them. And the purpose is to identify a single optimal output from these multiple nodes. Common parallel search algorithms include grid search [27] and stochastic search [28]. Sequential search carries out a small amount of parallel optimization and a large number of sequential search, while each search will use the information obtained from the previous training as much as possible to gradually optimize the hyperparameters, and the results of each search will also be used to guide the subsequent hyperparameter optimization. The common serial search includes manual hyperparameter adjustment and Bayesian search. Sequence optimization generally provides a better solution, but due to the long training process, the time cost of multiple serial searches is often quite high [16].

Currently, most of the automatic hyperparameter adjustment mechanisms adopt the sequential optimization algorithm, which means the training results of each specific hyperparameter search are used as the prior knowledge to inspire the hyperparameters of the subsequent search. Srinivas et al. [29], Bergstra et al. [30] and Snoek et al. [31] have used a Bayesian optimization framework to exploit this knowledge by updating the posterior probabilities of Bayesian models using successfully trained hyperparameters. As noted in the previous section, these sequential algorithms mentioned above will make the whole search process very long, György & Kocsis et al. [32], Sabharwal et al. [33], Springenberg et al. [34], Lisha Liet et al. [35] and Tobias Domhan et al. [36] used the methods of setting the early termination conditions of training, using Intermediate loss values to predict the final performance, and modeling the overall time and data-dependent optimization process to reduce the number of steps required in each training process and better explore the promising hyperparameter space to accelerate the overall hyperparameter search process.

Shah & Ghahramani et al. [37], González et al. [38], Wu & Frazier [39], Patrick Koch [40], Richard Liaw [25], and Daniel Golovin [26] have tried to parallelize Bayesian optimization by training multiple models independently, but these methods still need a large number of serial model updating operations in a single model.

In terms of parallel search, Bergstra & Bengio et al. [24] has proved the efficiency of random hyperparameter search. Loshchilov & Hutter et al. [41], Smith et al. [42], and Massé & Ollivier et al. [43] proved the effectiveness of adaptive adjustment of hyperparameters such as the learning rate in the process of training optimization. Lisha Li et al. [44] and Kevin Jamieson et al. [45] formulate hyperparameter optimization as a pure-exploration non-stochastic infinite-armed bandit problem. Liam Li [46] proposed a parallelized SHA algorithm suitable for large-scale distributed environments. Bäck et al. [47], Clune et al. [48], Xue et al. [49], Salustowicz & Schmidhuber [50] used the idea of genetic algorithm and Lamarckian evolutionary algorithm (parameters are inherited, hyperparameters are evolved) to realize population-based parallel search. Jaderberg et al. [16] of Deep Mind proposed the PBT algorithm, which uses standard optimization technology (gradient descent method) to replace the evolutionary model for parameter optimization, and realized an asynchronous hyperparameter search strategy. But the above strategies still need each node to train a complete model and dataset, which have the problems of excessive computing resource overhead and node computing redundancy.

2.2. Parallelism and consistency models

Deep learning currently mainly implements parallel training through model parallelism or data parallelism [51]. Model parallelism divides the deep neural network into multiple submodules and assigns them to multiple computing nodes for collaborative training. And data parallelism distributes the block datasets with multiple computing nodes, and each node independently completes the neural network training tasks on its own datasets and completes the parameter exchange through communication mechanisms such as parameter server or all-reduce algorithm. In most cases, the cost of model parallelism is much greater than that of data parallelism and the speedup is lower. Therefore, we choose data parallelism [52], which is currently more concentrated, as the parallel strategy of DPLRS. In the process of optimizing distributed parallel training, the main consistency models include Bulk Synchronous Parallel (BSP) [53] and

Asynchronous Parallel (ASP) [54] and their variants (SSP, ADSP, etc.). We select the overall synchronization model of most deep learning tasks in the data center environment: BSP ensures that a group of distributed computing nodes can update the model parameters with the same iteration by setting a synchronization fence. After each iteration, the computing node will wait for the synchronization instruction from the master node according to the synchronization fence mechanism, and the master node updates the parameters after obtaining the updated parameters of all computing nodes and sends them to each computing node to enter a new round of iteration. Common BSP model systems include Spark [55] and Mahout [55].

2.3. Evolutionary computing algorithms

Evolutionary algorithms are a class of meta-heuristic optimization methods with a certain level of robustness and generality, which is quite attractive to be applied to system whose model is difficult to build. Both Particle Swarm Optimization(PSO) [56] and Genetic Algorithms(GA) [57] are highly representative evolutionary computing techniques.

The PSO originated from the behavioral study of bird flock predation. It makes use of the sharing of global optimal information (*gbest*) by individuals in the population, so that the movement of the whole population produces an evolutionary process from disorder to order in the problem-solving space. D.V. Yamille et al. [58] applied PSO to the optimization of power systems. A. Adid et al. [59] used PSO for feature selection thus improving the accuracy of vehicle classification tasks.

Genetic algorithms are designed and proposed according to the evolutionary laws of organisms in nature. They are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection. N. A. Al-Madi et al. [60] used genetic algorithms to optimize traffic signal timers, thus reducing congestion periods in urban traffic. H. M. Balaha et al. used genetic algorithms for hyperparameter selection in the COVID-19 segmentation and identification framework to achieve state-of-the-art metrics [61].

Evolutionary algorithms are well suited to black box problems where mathematical optimization models are difficult to build. So the ideas of these two algorithms can be well incorporated into the searching process for learning rate schedules.

3. DPLRS

In this section, we model the core part of the distributed deep learning training process and give the optimization of the model solving process by DPLRS based on the population idea.

3.1. Deep learning model

The most common deep neural network formulation is described as using the model f to optimize a set of parameters θ to obtain the minimum (or maximum) values of the objective function L (including classification, prediction, clustering, regression, etc.). Usually, optimization methods such as stochastic gradient descent are used to iteratively update the trainable parameters θ until it makes the objective function meet a certain threshold or reach a preset number of training times.

In each iteration, the updating of model parameters is affected by learning rate $\boldsymbol{\lambda}$. DPLRS provides a method to optimize the learning rate $\boldsymbol{\lambda}$ for objective function L in distributed data parallel environment. Each round of parameter updating is shown in the following formula (1):

$$\boldsymbol{\theta} = Iteration(\boldsymbol{\theta}|\boldsymbol{\lambda}) \tag{1}$$

The common iterative updating of neural network is shown in formula (2):

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t \nabla \boldsymbol{L}(\boldsymbol{\theta}_t) + \beta(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) \tag{2}$$

λ and β represent the two most typical hyperparameters, learning rate and momentum coefficient, And in the widely used mini-batch gradient descent, (2) above is rewritten as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\lambda_t}{B}\sum_{i=1}^{B} \nabla \boldsymbol{L_i}(\boldsymbol{\theta}_t) + \beta(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) \tag{3}$$

Where B represents the batch size, we can intuitively see from the above equation that adjusting the batch size and modifying the learning rate are equivalent, and the paper [41] demonstrates this experimentally, so we only need to adjust the learning rate to take into account the effects of these two hyperparameters.

The parameters are updated in a chain, and the ideal optimal solution should meet the following requirements:

$$\begin{aligned}\boldsymbol{\theta}^* &= Optimize(\boldsymbol{\theta}|(\lambda_t)_{t=1}^{T}) \\ &= Iteration(Iteration(...Iteration(\boldsymbol{\theta}|\lambda_1)...)|\lambda_{T-1}|\lambda_T)\end{aligned} \tag{4}$$

However, considering the iteration steps T and the calculation cost of each step, it can be computationally expensive to find the optimal parameter group for training, and it usually takes days, weeks, or even months to optimize θ. In addition, this strategy is also very sensitive to the learning rate schedule $(\lambda_t)_{t=1}^{T}$, the wrong choices of learning rate may lead to the bad solution or even the optimization of θ cannot converge. The correct choices of learning rate schedule require the existence or finding of prior knowledge about $\boldsymbol{\lambda}$. Therefore, the usual practice is to let the whole training use constant learning rate $\boldsymbol{\lambda_t}$ or use a simple schedule (such as using the scheduler provided by Pytorch to adjust the learning rate every a few steps). No matter which of the two methods is used, it is necessary to search over multiple possible values of the learning rate $\boldsymbol{\lambda}$.

$$\boldsymbol{\theta}^* = Optimize(\boldsymbol{\theta}|\boldsymbol{\lambda}^*),\ \boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}\epsilon\Lambda^T} L(optmise(\boldsymbol{\theta}|\boldsymbol{\lambda})) \tag{5}$$

3.2. Algorithm description

As an intelligent algorithm for efficient and fast execution of formula(4), the DPLRS is shown in Algorithm 1:

We consider that the entire training cluster is composed of N working nodes. Each node has its own combination of parameters and learning rate $\boldsymbol{\theta_i}, \boldsymbol{\lambda_i}$. Initially, all nodes have the same parameters:

$$\boldsymbol{\theta_i} = \boldsymbol{\theta_0} \tag{6}$$

Then, in the process of iterative training of the deep learning model, on the one hand, BSP is used to update the global parameters regularly; on the other hand, the optimal learning rates are continuously explored according to the specific conditions. After the training, the optimal model is obtained from the cluster.

In order to achieve the above goals, the DPLRS introduces the basic idea of the population (evolutionary) algorithms. The population algorithm draws on the evolutionary characteristics of biological populations in nature, including elementary operations such as genetic coding, population initialization, crossover mutation operators, and gene retention mechanisms. Compared with optimization algorithms such as calculus-based methods and exhaustive methods, population algorithms are a series of mature global optimization algorithms with high robustness and wide applicability [62]. They have the characteristics of self-organization, self-adaptive, and self-learning. They can effectively deal with complex problems (such as NP-hard optimization problems) which are difficult to solve by traditional optimization algorithms without the limitation of the properties of the problem. Inspired by PSO and GA, we designed the genetic process

Algorithm 1 Distributed Population Learning Rate Schedule(DPLRS)

$\boldsymbol{\theta}$ is the initial parameter value of each node
$\boldsymbol{\lambda}$ is the initial learning rate of each node
$\boldsymbol{\theta}^*$ is the optimal model parameter value
p is the evaluation of the model's performance
P is the Population
t means the current training is the t^{th} iteration.
i means the current training is the i^{th} epoch.

for $(\boldsymbol{\theta}, \boldsymbol{\lambda}, p) \in P$ **do**
  while not end of training **do**
    $\boldsymbol{\theta} = Iteration(\boldsymbol{\theta}|\boldsymbol{\lambda})$ ▷ One iteration of DL training using learning rate λ
    $p += L(\boldsymbol{\theta})$ ▷ Evaluate the current model using loss values
    if $readyp(p, P)$ **then**
      if Completed an epoch training **then**
        $\boldsymbol{\theta}_{i+1} = allreduce(\boldsymbol{\theta}_i)/n$ ▷ Allreduce and average model parameters over all nodes
      else
        $\nabla \boldsymbol{L}(\boldsymbol{\theta}_t) = allreduce(\nabla \boldsymbol{L}(\boldsymbol{\theta}_t))/n$ ▷ Allreduce and average model gradients over all nodes
        $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_i \nabla \boldsymbol{L}(\boldsymbol{\theta}_t) + \beta(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})$ ▷ Updates the model parameters
    if $readyhp(p, P)$ **then**
      $\boldsymbol{\lambda_{list}} = allgather([p_i, \lambda_i])$ ▷ Allgather the combination of each node loss value and learning rate
      $\boldsymbol{\lambda}^* = min(\boldsymbol{\lambda_{list}})$ ▷ Select the learning rate with the smallest loss value as the optimal learning rate
      if $\lambda_i != \lambda^*$ **then**
        $\lambda_{i+1} = inherit(\lambda^*)$ ▷ Replace the current learning rate with the optimal learning rate
        $\lambda_{i+1} = mutate(\lambda_{i+1})$ ▷ Generate new learning rate
  $\boldsymbol{\theta}^* = getmaxAccuracy(\boldsymbol{\theta})$ ▷ Obtain the global optimal model parameters
return *best parameter* $\boldsymbol{\theta}^*$

of learning rate ($inherit(\lambda*)$), each node can inherit the optimal learning rate of the entire cluster from the optimal node every time the learning rate is updated. At the same time, we also design mutation operation(mutate(λ)), which expands new learning rates on the basis of the existing optimal learning rate and better explores the search space of the learning rate.

Each node in the cluster is trained in data parallel. Each node utilizes the rest of the node training results by *sum*() and *average*() model parameter gradients at the completion of each batch training and *sum*() and *average*() model parameters at the completion of each epoch training. The Iteration() function is called repeatedly to continuously update the local node parameters. The $readyp(p, P)$ function is used to determine whether the model satisfies the condition of parameter update, and if the result is True, then parameter update is performed for all nodes. And $readyhp(p, P)$ function is used to determine whether the model satisfies the condition of hyperparameter update, and if the result is True, then hyperparameter update of all nodes is performed. When all nodes in the cluster meet the *readyp*() condition (for example, when all nodes have executed several batches After training), the *allreduce*() function is called to synchronously complete the reduction operation of all parameters, and then the result obtained is divided by the number of nodes as the parameters for further training of all working nodes; at the same time, when all nodes conform with *readyhp*() condition (*readyhp*() condition is irrelevant to *readyp*() condition), we call the *allgather*() function to obtain the combination ($[p_i, \boldsymbol{\lambda_i}]$) of the model performance evaluation p_i (such as the cumulative values of the loss function) and the learning rates $\boldsymbol{\lambda_i}$ at current stage(from the last execution of *readyhp*() to this time) of each node and then use the *min*() function to obtain the combination of evaluation and learning rate with the optimal evaluation index (e.g., lowest loss value), and record the optimal learning rate as $\boldsymbol{\lambda}^*$. Then each node will make a judgment, if its own hyperparameters are $\boldsymbol{\lambda}^*$, the hyperparameters remain unchanged, otherwise, the *inherit*() and *mutate*() functions will be executed to update the hyperparameter values. After updating the parameters and learning rates, continue to execute *Iteration*() function. Repeat the above cyclic update process until the model converges.

Inherit() and *Mutate*() operations can be adjusted according to the application requirements. In the deep learning training environment described in this paper, *Iteration*() function is to complete a mini-batch stochastic gradient descent (minibatch SGD), *L*() function represents to calculate the loss values of each training, and accumulate it in the variable p, *Inherit*() function is to inherit the learning rates from the optimal node, *Mutate*() function is to explore new learning rates for gradient based learning by adding disturbance to the optimal learning rate at this stage. The *readyp*() condition and *readyhp*() condition depend on the choices of the update time, which are often triggered when the cumulative gradient (loss function) is large enough to produce significant gradient-based learning. In general, the update time of learning rates and parameters does not have to be the same.

In this paper, the meaning of each function of Algorithm 1 is shown in Table 1.

4. Experimental results and analysis

This section details the application of the DPLRS algorithm to the classification task of typical deep models AlexNet, VGG16 and ResNet18 trained in parallel on distributed synchronous data for the Cifar-10 and ImageNet dataset using the Tianhe-3 supercomputing prototype platform.

4.1. Tianhe-3 supercomputing prototype

The processors used in the Tianhe-3 supercomputing prototype include FT-2000+ (FTP) and MT-2000+ (MTP). FTP contains 64 FTC662 processor cores of armv8 architecture, operating at 2.2–2.4 GHZ, with an on-chip integrated 32MB secondary cache, providing 204.8GB/s access bandwidth and typical operating energy consumption of about 100 W. And the MTP processor, which

Table 1
DPLRS Function description.

Name	Descriptions
Iteration($\boldsymbol{\theta}\|\lambda$)	All nodes iteratively select data locally to perform the forward and backward propagation of mini-batch gradient descent.
readyp(p, P)	Determine if the training status is in either of the following two conditions. First, the gradient update of the model parameters is based on the condition that all nodes of the distributed cluster satisfy the training batchsize = 64(Then each node performs *allreduce*() global gradient synchronization and averaging, then uses the learning rate of each node to update the model parameters); second, when an epoch is completed, all nodes also perform *allreduce*() to synchronize the model parameters once to ensure that the initial weights of each node are the same at the beginning of each epoch training.
allreduce($\boldsymbol{\theta}_i$)	Get the model parameters of each node and sum them up.
allreduce($\nabla \boldsymbol{L}(\boldsymbol{\theta}_t)$)	Get the model parameter gradients of each node and sum them up.
readyhp(p, P)	Determine if each node has completed an epoch training.
allgather($[p_i, \lambda_i]$)	Get the combination of loss value and learning rate of each node, and aggregate the results into a list.
min($\boldsymbol{\lambda}_{\boldsymbol{list}}$)	Obtain the learning rate in the combination with the lowest loss value in $\boldsymbol{\lambda}_{\boldsymbol{list}}$.
inherit(λ^*)	Replace the local learning rate with the best performing learning rate from the previous epoch in the node cluster.
mutate(λ_{i+1})	Multiply the locally updated learning rate by a random perturbation of 0.8 to 1.2 as a new learning rate to expand the learning rate search space.
getmaxAccuracy($\boldsymbol{\theta}$)	Infer the test dataset using model parameters of each node to get the model parameters with the highest accuracy.

Table 2
Basic Situation Of Tianhe-3 Prototype System.

Specifications		FT-2000+	MT-2000+
Hardware	Nodes	128	512
	Cores in a node	32	32
	Frequency	2.4 GHZ	2.0 GHZ
	Memory	64 GB	16 GB
	Interconnect bandwidth	200 Gbps	200 Gbps
Software	OS	Kylin 4.0-1a OS with kernel v4.4.0	
	File system	Lustre	
	MPI	MPICH v3.2.1	
	Compiler	GCC v4.9.1/v4.9.3	
	Supported libraries	Boost, BLAS, OpenBLAS, Scalapack, etc.	

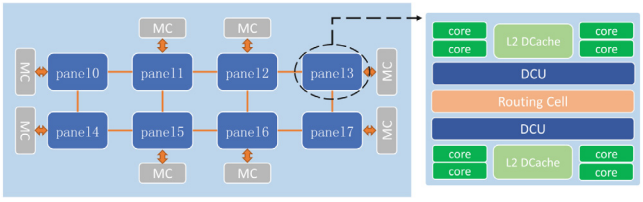

Fig. 1. FT2000+ Processor Architecture [63].

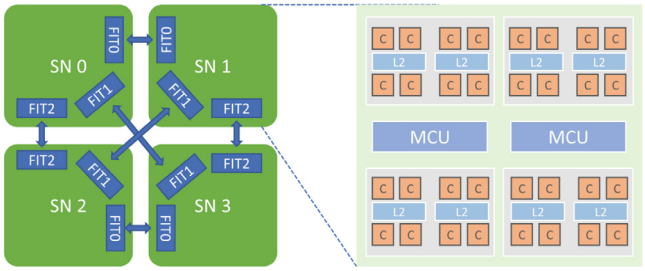

Fig. 2. MT2000+ Processor Architecture [63].

contains a total of 128 custom processor cores, is organized into 4 supernodes with a maximum main frequency of 2.0GHZ and consumes 240 W. The processor architecture of FTP and MTP is shown in the following Figs. 1 and 2 [63].

In the Tianhe-3 supercomputing prototype, as shown in Table 2, both FTP and MTP are divided to use 32 cores as one compute node(Unless otherwise specified, the FTP and MTP described subsequently in this paper denote an FTP node (32 cores) or MTP node (32 cores) in the Tianhe-3 supercomputing prototype, and not the FT-2000+ processor (64 cores) and the MT-2000+ processor (128 cores)), which is probably done to provide more compute nodes for complex computational tasks [30]. The compute nodes are managed and allocated by a batch scheduling system. In FTP, 32 cores share 64G of memory, while in MTP, 32 cores share 16G of memory. They both come with Kylin 4.0-1a operating system with kernel v4.4.0. In addition, the prototype cluster interconnect technology designed and implemented by the National University of Defense Technology provides 200Gbps bi-directional interconnect bandwidth.

4.2. Experimental environment setting

We complete the above training on PyTorch, a widely accepted deep learning training platform, in the environment of the Tianhe-3 supercomputing prototype platform.

During the experiments, up to 128 MTP nodes (4096 processor cores) and 32 FTP nodes (1024 processor cores) were used respectively for AlexNet, VGG16 and ResNet18 networks using the DPLRS algorithm and the latest population algorithm PBT and fixed learning rate strategy on the Cifar-10 and ImageNet dataset, respectively 300 epochs were trained, and each set of experiments was conducted 10 times, and then the mean value of the 10 experiments was taken. The size of the batchsize of each node was always kept as 64.

The source of the advancement of the DPLRS algorithm is the use of a combination of distributed training and population algorithms to optimize the learning rate schedule selection, and the DPLRS algorithm can be combined with many existing learning

rate optimizers. To validate the effectiveness and efficiency of our algorithm, we combined the DPLRS algorithm with the SGDR [41] learning rate optimization algorithm currently used on the WRN (SAM) [64] model, which is the most advanced performer on the Cifar-10, Cifar-100 and Food-101 image recognition datasets. In order to minimize changes to the source code provided in [64] and to keep the experimental environment as close as possible to [64], we used an LTHPC platform environment (64-core Intel(R) Xeon(R) Silver 4216 CPU, 128 GB Memory, 2 NVIDIA GeForce RTX 3090 GPUs). The initial learning rate for all learning rate optimizers is 0.1, momentum is 0.9, total batchsize is 128, and other dynamic optimizer parameters use the torch.optimize default parameters.

The concrete implementation of the underlying communication code uses the "MPI" framework for the Tianhe-3 supercomputing prototype platform and the "GLOO" framework for the GPU platform, both provided by the *torch.distributed* package.

4.3. Experimental results

4.3.1. DPLRS on the cifar-10 dataset

As shown in Table 3, Table 4, Table 5 below, the use of DPLRS optimization algorithm has excellent performance in AlexNet, VGG16 and ResNet18 models in training Cifar-10 dataset, and significantly improves the training speed while ensuring that the final model test set classification accuracy is comparable to that of the PBT algorithm while using the same number of working nodes and training the same number of epochs.

The time spent in training AlexNet, VGG16 and ResNet18 networks by DPLRS and PBT in the experiments using MTP is shown in Table 3, the 8, 16, 32, 64, and 128 nodes were achieved at max 7.63x, 15.18x, 30.23x, 59.71x, and 123.85x speedups, respectively, compared to the baseline PBT model.

As shown in Table 5, the DPLRS had the highest test set classification accuracy both in AlexNet and ResNet18 models, and achieved comparable results with PBT in the VGG16 model in our experiments. Both DPLRS and PBT performed better than static learning rate training in all three models. The experimental results show that we can ensure the effectiveness of DPLRS well by making each node use the results of the rest nodes at proper scenarios for the update of the model parameters and the model parameter gradients.

To fully validate the effectiveness of the algorithm, we conducted similar experiments in using FTP, which showed that the training speed of 8, 16, and 32 nodes in AlexNet, VGG16, and ResNet18 networks was improved by 7.52x, 15.33x, and 30.40x, respectively, compared to the baseline PBT model.

Although training the model using fixed epochs is the most common training configuration, the data-parallel approach essentially expands the batchsize of each Iteration and thus requires more iterations to achieve the same convergence as the non-data-parallel algorithm [65]. Therefore, in order to compare DPLRS and PBT more fairly, we counted the time required to reach the same loss value for different node configurations of the two algorithms separately.

The experimental results are shown in Table 6. The experimental results show that DPLRS achieves the highest 59.72x, 72.40x and 62.23x speedup ratios compared to PBT on AlxeNet, VGG16 and ResNet18 models, respectively, when reaching the same loss value.

In the process of training DNN models using the DPLRS algorithm, the training dataset was first performed with data enhancement operations in the pre-processing stage, thus ensuring good robustness of the DPLRS algorithm. To demonstrate the robustness of the DPLRS algorithm, our experiments were performed with random image rotation, flip and resize perturbations

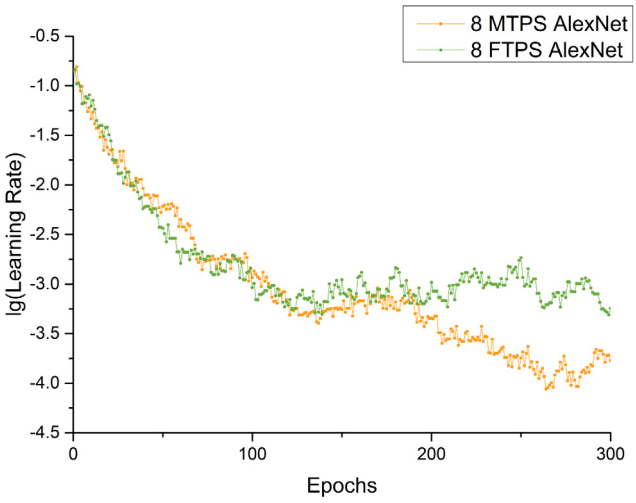

Fig. 3. Learning Rate during training of AlexNet.

on the test set samples when calculating the test set accuracy, as shown in Table 7. AlexNet, VGG16 and ResNet18 all achieved comparable accuracy to the original test set when tested using the noisy dataset in. The experimental results provide good evidence of the robustness of the DPLRS algorithm.

To verify the reliability of the DPLRS algorithm on large datasets, we extend similar experiments to the ImageNet dataset, where the highest 122.9x speedup ratio compared to PBT can be obtained using DPLRS on the ImageNet dataset. Details of the experimental results are shown in Appendix A.

4.3.2. DPLRS with SGDR

As shown in Tables 8 and 9, our DPLRS with SGDR achieves a better performance than the SGDR algorithm in [64] on the Cifar-10 and Cifar-100 datasets by up to 0.17% and 0.36% model accuracy improvement, while ensuring that the training speed-up ratio is consistent with the original DPLRS. Experimental results show that if we use the same initial learning rate and batchsize settings as the static method, the dynamic optimizer yields poor results.

4.4. Experimental analysis

This section models the final classification accuracy improvement of the model and the speed improvement of the population algorithm from DPLRS, respectively, and combines the experimental results to fully demonstrate the effectiveness and reliability of DPLRS.

4.4.1. Accuracy improvement analysis

The improvement of the DPLRS algorithm for model classification results mainly stems from the ability to dynamically and adaptively adjust the learning rate during training, so that the learning rate during training is always in the best part of the sampling range and is adjusted over time. As shown in Fig. 3 below, during the 300 rounds of training the AlexNet network(8 Nodes) using the DPLRS algorithm to adjust the learning rate, the learning rate can automatically decay with the training process, which is very consistent with previous experience in training neural networks (see Fig. 4).

When the learning rate is large, the model will accelerate learning, making it easier to approach the local or global optimal solution, but at the same time there will be large fluctuations, and when it is close to convergence, even the value of the loss function hovers around the minimum value, always difficult to

Table 3
Training Time of AlexNet, VGG16 and ResNet18 Models(mins) on Cifar-10 dataset.

	8 MTPs	16 MTPs	32 MTPs	64 MTPs	128 MTPs	8 FTPs	16 FTPs	32 FTPs
PBT-Alex	15163.32	15314.44	15265.53	15313.13	16508.27	13414.42	13409.01	13431.62
DPLRS-Alex	1995.15	1026.03	522.36	256.47	**133.29**	1785.55	887.70	455.07
PBT-VGG16	32407.41	32426.18	32752.02	32492.08	32925.31	28283.53	28522.15	28425.66
DPLRS-VGG16	4257.16	2136.65	1083.42	546.14	**277.86**	3765.54	1860.71	935.07
PBT-Res18	70108.91	69592.29	69709.30	70491.15	70224.28	60772.19	60768.75	60747.86
DPLRS-Res18	8952.02	4398.54	2265.16	1134.25	**570.07**	7796.13	3904.41	1951.20

Table 4
Average One Epoch Training Time of AlexNet, VGG16 and ResNet18 Models(mins) on Cifar-10 dataset.

	8 MTPs	16 MTPs	32 MTPs	64 MTPs	128 MTPs	8 FTPs	16 FTPs	32 FTPs
PBT-Alex	50.54	51.05	50.88	51.04	55.02	44.71	44.69	44.77
DPLRS-Alex	6.65	3.42	1.7412	0.85	**0.44**	5.95	2.96	1.52
PBT-VGG16	108.02	108.08	109.17	108.30	109.75	94.27	95.07	94.75
DPLRS-VGG16	14.19	7.12	3.61	1.82	**0.93**	12.55	6.20	3.12
PBT-Res18	233.69	231.97	232.36	234.97	234.08	202.57	202.56	202.49
DPLRS-Res18	29.84	14.66	7.55	3.78	**1.90**	25.99	13.01	6.50

Table 5
Training Accuracy of AlexNet, VGG16 and ResNet18 Models on Cifar-10 dataset.

	AlexNet	VGG16	ResNet18
DPLRS	**0.8474**	0.864	**0.8744**
PBT	0.8458	**0.8645**	0.8738
Static	0.8272	0.8504	0.8551

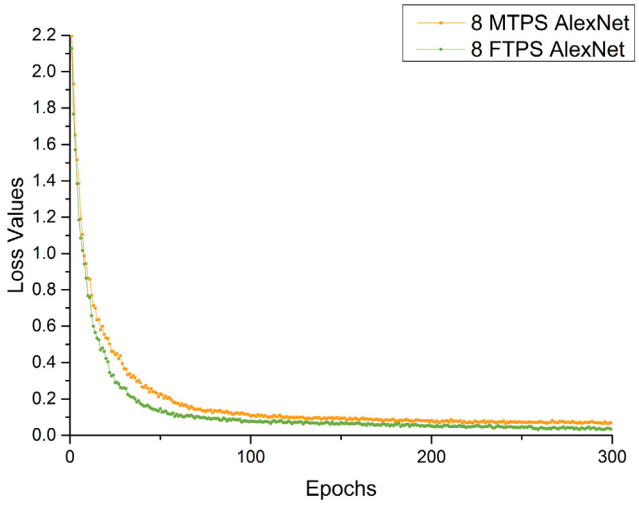

Fig. 4. Loss Values during training of AlexNet.

reach the optimum, so the introduction of learning rate decay, first large and then small learning rate can improve the convergence effect of the model. Compared with the static learning rate adjustment methods of decaying the learning rate in specified rounds and introducing exponential decay, DPLRS can select the optimal learning rate in real time according to the training state of each node, and the mutation operation can play the role of simulated annealing to make the learning rate selection more likely to go beyond the local optimum and tend to the global optimum.

4.4.2. Acceleration efficiency analysis

The outperformance of DPLRS over PBT near-node number (N) is mainly due to the fact that DPLRS relaxes the condition of population learning rate update.

We make all nodes use only 1/N of the total data as their local training dataset, and each node updates its learning rate using the DPLRS algorithm after completing an epoch of training using its own learning rate and local data, which means that for DPLRS, although each node uses the same model at the initial moment of each epoch, the model we use to discriminate between good and bad learning rates is trained by each node using the same number of different local data of each node, while the PBT algorithm requires that the models used for comparison are obtained from the same models starting with the exact same data and trained for the same number of iterations, so this restricts PBT from using data parallelism, and only the same models with same datasets can be trained at different learning rates at the same time. So that Although PBT also performs learning rate update after completing the training of one epoch, the time to train one epoch for PBT is about N times that of DPLRS (because PBT trains one epoch with N times more data than DPLRS), which is the reason why DPLRS is N times faster than PBT for the same number of epochs trained.

We can relax the learning rate update based on mini-batch SGD stochastic gradient descent is valid on the premise that each time the mini-batch data used for training satisfies the assumption of independent identical distribution, which means that although the models we compare each time are trained with different data (of the same size), these data satisfy independent identical distribution, so the training results can reflect the good or bad learning rate, which was confirmed in our experiments.

In the process of distributed deep learning training using mini-batch stochastic gradient descent, the number of nodes involved in training is N, the total number of training epochs is n, the Size of the dataset is S, the Batchsize per node is B, the training time of each node per batch is μ, the communication time between nodes is φ. According to the Algorithm 1 (DPLRS) and the trigger condition of each function in Table 1 the total single training time T_{DPLRS} can be obtained as:

$$T_{\mathrm{DPLRS}} = \frac{nS}{NB} \times (\mu + \varphi_{DPLRS}) + n\varphi_{DPLRS} \tag{7}$$

Meanwhile, the total training time of the PBT algorithm is:

$$T_{\mathrm{PBT}} = \frac{nS}{B} \times \mu + 2n\varphi_{PBT} \tag{8}$$

We can find that the PBT training time is independent of the number of nodes N by Eq. (8). No matter how many nodes we use, each node processes the same amount of data (always the whole training dataset). The training time for each epoch depends on how long it takes for the slowest node in the cluster to finish training. Our FTP cluster or MTP cluster are both homogeneous, which means that the training time of each node is also almost the same.

When the model has a large number of layers and depth and a complex structure, the node computation overhead will be much larger than the communication overhead. The computation and

Table 6
Training Time of AlexNet, VGG16 and ResNet18 Models(mins) on Cifar-10 dataset(end at same loss).

	8 MTPs	16 MTPs	32 MTPs	64 MTPs	128 MTPs	8 FTPs	16 FTPs	32 FTPs
PBT-Alex	7214.20	7303.24	7261.56	7277.24	7833.17	6528.3462	6582.93	6522.15
DPLRS-Alex	1190.35	656.64	363.56	203.124	**131.15**	1059.43	568.83	313.21
PBT-VGG16	10370.37	10162.96	9858.07	10350.97	10557.99	9050.56	8960.05	9228.85
DPLRS-VGG16	1416.78	738.42	402.16	230.68	**145.82**	1253.17	714.51	448.83
PBT-Res18	20097.55	19717.81	19843.91	20113.43	20943.51	17421.30	17461.82	17434.63
DPLRS-Res18	3289.56	1710.15	942.17	538.99	**336.57**	2754.63	1489.92	799.73

Table 7
Training Accuracy of AlexNet, VGG16 and ResNet18 Models on Cifar-10 dataset with noise.

	AlexNet	VGG16	ResNet18
DPLRS	0.8479	0.8628	0.8736

Table 8
Training Accuracy of Cifar-10 Dataset.

	2 nodes	4 nodes	8 nodes
SGDR	97.02	97.05	**97.16**
ADAM	79.12	80.62	**82.81**
DPLRS	97.12	97.21	**97.33**

Table 9
Training Accuracy of Cifar-100 Dataset.

	2 nodes	4 nodes	8 nodes
SGDR	83.00	83.03	**83.18**
ADAM	38.12	**43.12**	38.75
DPLRS	83.14	83.36	**83.54**

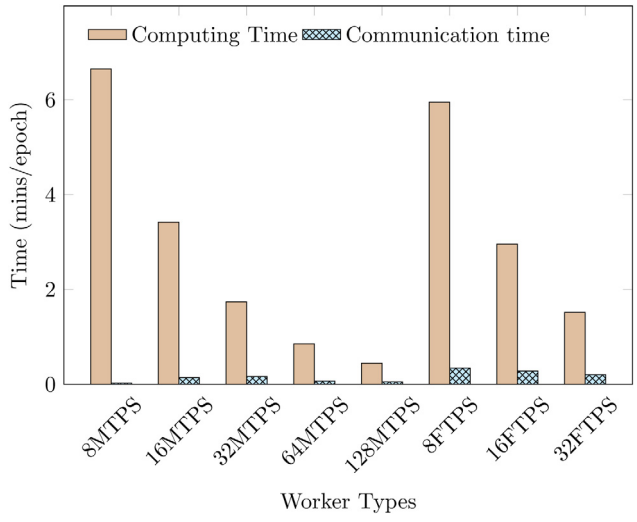

Fig. 5. Computation and communication time using the AlexNet model on the Cifar-10 dataset.

communication time per epoch during the training of AlexNet model with different numbers of nodes are shown in Fig. 5(see Appendix B for the detailed experimental results) , T_{DPLRS}/T_{PBT} will tend to be 1/n, which means that the algorithm proposed in this paper will be n times faster than the PBT algorithm.

There exist studies parallelize population algorithms, which may reduce the training time of PBT, but it can be a promising approach needs further exploration.

If the termination condition of our training model is that the model loss value reaches a certain threshold, the value of n in T_{DPLRS} is different from the value of n in T_{PBT} at this point, and the exact acceleration ratio depends on the actual situation of the different n values in the two algorithms.

4.4.3. Algorithm effectiveness analysis

The theoretical guarantee of using population algorithms in deep neural network training is indeed a fundamental problem. These papers [56,66,67] emphasize that the population algorithm is highly dependent on stochastic processes, which makes it difficult to guarantee convergence. At the same time, the paper [10] points out that the setting of the learning rate schedule requires a large number of trial-and-error iterations, and is hard to directly formulate the search of the learning rate schedule as a well-posed optimization problem and address it through standard optimization. Despite the lack of a theoretical proof, we demonstrate the effectiveness of our algorithm through extensive experiments. Theoretical proofs are not the subject of this paper and we will explore the convergence to guarantee of population algorithms such as DPLRS in more depth in subsequent research.

5. Conclusion

We propose a distributed deep learning DPLRS algorithm based on the joint optimization of particle swarm algorithm and genetic algorithm based on the population idea.

DPLRS utilizes data parallelism based on BSP synchronization protocol, and jointly borrows the genetic operation designed by particle swarm algorithm and the mutation operation designed by genetic algorithm to automatically and periodically adjust the learning rate according to the model's performance during the training process. Experimental results using different strategies on several typical deep neural network models show that the DPLRS algorithm can improve the training time by *a* factor of nearly *n* (*n* is the number of training nodes) while maintaining nearly the same final test set accuracy as the state-of-the-art population algorithm.

In the future, we will try to use more advanced population algorithms in combination with more hyperparameter optimization. And, we will continue to explore theoretical guarantees for the incorporation of population algorithms with deep neural network learning rate schedule.

CRediT authorship contribution statement

Jia Wei: Conceptualization, Methodology, software, Investigation, Writing – original draft, Validation. **Xingjun Zhang:** Resources, Funding acquisition, Project administration, Supervision. **Zeyu Ji:** Data curation, Validation. **Zheng Wei:** Formal analysis. **Jingbo Li:** Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to thank the anonymous reviewers, whose insightful comments greatly improved the quality of this paper. This work was supported by the National Natural Science Foundation of China (62172327).

Table 10
DPLRS compared to PBT speedup ratio on ImageNet dataset.

	8 MTPs	16 MTPs	32 MTPs	64 MTPs	128 MTPs	8 FTPs	16 FTPs	32 FTPs
AlexNet	7.567	15.16	29.25	59.54	**122.9**	7.387	15.29	29.98
VGG16	7.497	15.10.	30.18	58.68	119.6	7.577	15.12	29.84
Res50	7.895	16.09	30.92	62.89	121.88	7.880	15.83	31.36

Table 11
DPLRS total training time and communication time.

	8 MTPs	16 MTPs	32 MTPs	64 MTPs	128 MTPs	8 FTPs	16 FTPs	32 FTPs	DataSet
AlexNet-TOTAL	6.651	3.422	1.7412	0.8549	0.4443	5.95	2.959	1.5169	Cifar-10
AlexNet-Comm	0.2432	0.1413	0.1634	0.6618	0.0512	0.3434	0.1776	0.1167	Cifar-10
VGG16-TOTAL	14.19	7.125	3.6114	1.820	0.9262	12.55	6.200	3.1169	Cifar-10
VGG16-Comm	0.3896	0.3208	0.1759	0.1005	0.0773	0.5513	0.2236	0.2035	Cifar-10
Res50-TOTAL	29.84	14.66	7.55	3.7821	1.901	25.98	13.01	6.5040	Cifar-10
Res50-Comm	0.4641	0.2834	0.3034	0.1828	0.1014	0.5317	0.3287	0.1295	Cifar-10
AlexNet-TOTAL	824.22	415.21	203.31	102.32	52.54	753.09	364.36	180.07	ImageNet
AlexNet-Comm	58.61	38.31	18.27	9.96	6.13	43.46	31.02	14.57	ImageNet
VGG16-TOTAL	1758	864.4	426.5	219.1	109.9	1588	769.7	370.9	ImageNet
VGG16-Comm	49.00	36.52	20.73	12.03	9.456	69.81	43.46	24.21	ImageNet
Res50-TOTAL	3253	1597	822.9	397.9	199.9	2832	1418	708.5	ImageNet
Res50-Comm	95.52	58.14	50.06	30.04.15	14.74	58.93	34.88	15.27	ImageNet

Appendix A. DPLRS training results on ImageNet dataset

We conducted 300 epoch experiments on ImageNet dataset using AlexNet, VGG16 and ResNet50 models respectively in order to verify the effectiveness of DPLRS algorithm on large dataset, we also used the experimental settings of minimum 8 and maximum 128 MTP nodes and minimum 8 and maximum 32 FTP nodes, and the speedup of DPLRS over PBT algorithm under different experimental settings are shown in Table 10 below.

The above three models for experiments on ImageNet and Cifar-10 were obtained by fine-tuning the models based on those provided by the torchvision.models() package, respectively.The results of testing on the ImageNet dataset using a completely different test set of images to the training set provide good evidence of the algorithm's generalization performance. The experimental results also show that the DPLRS algorithm can also obtain high speed-up ratios on large datasets.

Appendix B. DPLRS computing and communication overhead

To demonstrate that when the model is complex, as described in Section 4.4.2, the communication time is a low percentage of the total time during each Epoch training. We counted the total time and communication time during the training of AlexNet, VGG16 and ResNet (using ResNet18 on the Cifar-10 dataset and ResNet50 on the ImageNet dataset) models on the Cifar-10 and ImageNet datasets, respectively, and the results are shown in Table 11.As the number of nodes increases, the number of times each Epoch performs communication is greatly reduced, resulting in less overall communication time

References

[1] M. Li, Y. Liu, D. Qian, et al., The deep learning compiler: A comprehensive survey, IEEE Trans. Parallel Distrib. Syst. (2020) 708–727.
[2] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings Of The IEEE Conference On Computer Vision And Pattern Recognition, 2016, pp. 770–778.
[3] M. Elbattah, J. L. Guerin, R. Carette, et al., Generative modeling of synthetic eye-tracking data: NLP-based approach with recurrent neural networks, in: Proceedings Of The 12th International Joint Conference On Computational Intelligence, 2020, pp. 479–484.
[4] T. Gegovska, R. Koker, T. Cakar, Green supplier selection using fuzzy multiple-criteria decision-making methods and artificial neural networks, Comput. Intell. Neurosci. (2020) 1–26.
[5] O. Sener, S. Savarese, Active learning for convolutional neural networks: A core-set approach, in: Proceedings Of The International Conference On Learning Representations, 2018.
[6] T. Yin, N. Liu, H. Sun, Self-paced active learning for deep CNNs via effective loss function, Neurocomputing (2021) 1–8.
[7] J. Devlin, M.-W Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, North Am. Chapter Assoc. Comput. Linguist. (2019) 4171–4186.
[8] T.Y. Lin, R. Goyal, et al., Focal loss for dense object detection, in: Proceedings Of The IEEE International Conference On Computer Vision, 2017, pp. 2980–2988.
[9] W. Liu, Y. Wen, Z. Yu, et al., Sphereface: Deep hypersphere embedding for face recognition, in: Proceedings Of The IEEE Conference On Computer Vision And Pattern Recognition, 2017, pp. 212–220.
[10] Y. Jin, T. Zhou, L. Zhao, et al., AutoLRS: Automatic learning-rate schedule by Bayesian optimization on the fly, in: Proceedings Of The International Conference On Learning Representations, 2021.
[11] K. Kawaguchi, Deep learning without poor local minima, in: Proceedings Of The Conference On Neural Information Processing Systems, 2016, pp. 586–594.
[12] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[13] C. Jin, R. Ge, P. Netrapalli, et al., How to escape saddle points efficiently, in: Proceedings Of The International Conference On Machine Learning, 2017, pp. 1724–1732.
[14] Z. Li, S. Arora, An exponential learning rate schedule for deep learning, in: Proceedings Of The International Conference On Learning Representations, 2020.
[15] A.H. Victoria, G. Maragatham, Automatic tuning of hyperparameters using Bayesian optimization, Evol. Syst. (2021) 217–223.
[16] M. Jaderberg, V. Dalibard, S. Osindero, et al., Population based training of neural networks, 2017, arXiv preprint arXiv:1711.09846.
[17] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Proceedings Of The International Conference On Learning Representations, 2015.
[18] M.D. Zeiler, Adadelta: an adaptive learning rate method, 2012, arXiv preprint arXiv:1212.5701.
[19] R. Esteban, A. Alok, H. Yanping, L. Quoc V., Regularized evolution for image classifier architecture search, in: Proceedings Of The AAAI Conference On Artificial Intelligence, 2019, pp. 4780–4789.
[20] C. Edoardo, M. Vashisht, S.F. Petroski, et al., Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents, in: Proceedings Of The 32nd International Conference On Neural Information Processing Systems, 2018, pp. 5032–5043.
[21] A. Kaplunovich, Y. Yesha, Automatic tuning of hyperparameters for neural networks in serverless cloud, in: Proceedings Of The IEEE International Conference On Big Data, 2020.
[22] N. Giladi, M.S. Nacson, E. Hoffer, D. Soudry, At Stability's Edge: How to adjust hyperparameters to preserve minima selection in asynchronous training of neural networks? in: Proceedings Of The International Conference On Learning Representations, 2020.
[23] Y. Zhou, S. Cahya, S.A. Combs, et al., Exploring tunable hyperparameters for deep neural networks with industrial ADME data sets, J. Chem. Inf. Model. (2018) 1005–1016.
[24] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization., J. Mach. Learn. Res. (2012) 281–305.
[25] R. Liaw, E. Liang, R. Nishihara, et al., Tune: A research platform for distributed model selection and training, 2018, arXiv preprint arXiv:1807.05118.

[26] D. Golovin, B. Solnik, S. Moitra, et al., Google vizier: A service for black-box optimization, in: Proceedings Of The 23rd ACM SIGKDD International Conference On Knowledge Discovery And Data Mining, 2017, pp. 1487–1495.
[27] Y. Sun, S. Ding, Z. Zhang, W. Jia, An improved grid search algorithm to optimize SVR for prediction, Soft Comput. (2021) 5633–5644.
[28] H. Fu, G. Wu, J. Liu, Y. Xu, More efficient stochastic local search for satisfiability, Appl. Intell. (2021) 3996–4015.
[29] N. Srinivas, A. Krause, S.M. Kakade, M. Seeger, Gaussian process optimization in the bandit setting: No regret and experimental design, in: Proceedings Of The International Conference On Machine Learning, 2010, pp. 1015–1022.
[30] J. Bergstra, R. Bardenet, B. Bengio, Algorithms for hyper-parameter optimization, in: Proceedings Of The 25th International Conference On Neural Information Processing Systems, 2011.
[31] J. Snoek, O. Rippel, K. Swersky, et al., Scalable bayesian optimization using deep neural networks, in: Proceedings Of The International Conference On Machine Learning, 2015, pp. 2171–2180.
[32] A. György, L. Kocsis, Efficient multi-start strategies for local search algorithms, J. Artif. Intell. Res. (2011) 407–444.
[33] A. Sabharwal, H. Samulowitz, G. Tesauro, Selecting near-optimal learners via incremental data allocation, in: Proceedings Of The AAAI Conference On Artificial Intelligence, 2016, pp. 2007–2015.
[34] J.T. Springenberg, A. Klein, S. Falkner, F. Hutter, Bayesian optimization with robust Bayesian neural networks, in: Proceedings Of The 30th International Conference On Neural Information Processing Systems, 2016, pp. 4141–4149.
[35] J. Bergstra, B. Komer, C. Eliasmith, et al., Hyperopt: a python library for model selection and hyperparameter optimization, Comput. Sci. Discov. (2015).
[36] T. Domhan, J.T. Springenberg, F. Hutter, Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves, in: Proceedings Of The Twenty-Fourth International Joint Conference On Artificial Intelligence, 2015.
[37] A. Shah, Z. Ghahramani, Parallel predictive entropy search for batch global optimization of expensive objective functions, in: Proceedings Of The Conference On Neural Information Processing Systems, 2015, pp. 3330–3338.
[38] J. González, Z. Dai, P. Hennig, N. Lawrence, Batch Bayesian optimization via local penalization, in: Proceedings Of The International Conference On Artificial Intelligence And Statistics, 2016, pp. 648–657.
[39] J. Wu, P.I. Frazier, The parallel knowledge gradient method for batch Bayesian optimization, Proc. Conf. Neural Inf. Process. Syst. (2016) 3126–3134.
[40] P. Koch, O. Golovidov, S. Gardner, et al., Autotune: A derivative-free optimization framework for hyperparameter tuning, in: Proceedings Of The 24th ACM SIGKDD International Conference On Knowledge Discovery & Data Mining, 2018, pp. 443–452.
[41] I. Loshchilov, F. Hutter, Sgdr: Stochastic gradient descent with warm restarts, in: Proceedings Of The International Conference On Learning Representations, 2017.
[42] L.N. Smith, Cyclical learning rates for training neural networks, in: Proceedings Of The IEEE Winter Conference On Applications Of Computer Vision, 2017, pp. 464–472.
[43] P. Massé, Y. Ollivier, Speed learning on the fly, 2015, arXiv preprint arXiv:1511.02540.
[44] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: A novel bandit-based approach to hyperparameter optimization, J. Mach. Learn. Res. (2017) 6765–6816, JMLR.org.
[45] K. Jamieson, A. Talwalkar, Non-stochastic best arm identification and hyperparameter optimization, in: Proceedings Of The International Conference On Artificial Intelligence And Statistics, 2016, pp. 240–248.
[46] L. Li, K. Jamieson, A. Rostamizadeh, et al., Massively parallel hyperparameter tuning, in: Proceedings Of The Conference On Machine Learning And Systems, 2018.
[47] T. Bäck, An overview of parameter control methods by self-adaptation in evolutionary algorithms, Fund. Inform. (1998) 51–66.
[48] J. Clune, D. Misevic, C. Ofria, et al., Natural selection fails to optimize mutation rates for long-term adaptation on rugged fitness landscapes, PLoS Comput. Biol. (2008).
[49] B. Xue, M. Zhang, W.N. Browne, X. Yao, A survey on evolutionary computation approaches to feature selection, IEEE Trans. Evol. Comput. (2015) 606–626.
[50] R. Salustowicz, J. Schmidhuber, Probabilistic incremental program evolution: Stochastic search through program space, in: Proceedings Of The European Conference On Machine Learning, 1997, pp. 213–220.
[51] J. Dean, G. Corrado, R. Monga, et al., Large scale distributed deep networks, in: Proceedings Of The Conference On Neural Information Processing Systems, 2012, pp. 1223–1231.
[52] Z. Ji, X. Zhang, Z. Fu, et al., Performance-awareness based dynamic batch size SGD for distributed deep learning framework, J. Comput. Res. Dev. (2019) 2396–2409.
[53] F. Loulergue, F. Gava, D. Billiet, Bulk synchronous parallel ML: modular implementation and performance prediction, in: Proceedings Of The International Conference On Computational Science, 2005, pp. 1046–1054.
[54] E.P. Xing, Q. Ho, P. Xie, D. Wei, Strategies and principles of distributed machine learning on big data, Engineering (2016) 179–195.
[55] M. Li, D.G. Andersen, J.W. Park, et al., Scaling distributed machine learning with the parameter server, in: Proceedings Of The 11th {USENIX} Symposium On Operating Systems Design And Implementation, 2014, pp. 583–598.
[56] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings Of International Conference On Neural Networks, 1995, pp. 1942–1948.
[57] D.E. Golberg, Genetic algorithms in search, optimization, and machine learning, Addion Wesley (1989) 36.
[58] D.V. Yamille, V.G. Kumar, M. Salman, et al., Particle swarm optimization: basic concepts, variants and applications in power systems, IEEE Trans. Evol. Comput. (2008) 171–195.
[59] A. Adi, S. Ammar, I. Talha, et al., A particle swarm optimization based deep learning model for vehicle classification, Comput. Syst. Sci. Eng. (2022) 223–235.
[60] N.A. Al-Madi, A.A. Hnaif, Optimizing traffic signals in smart cities based on genetic algorithm, Comput. Syst. Sci. Eng. (2022) 65–74, http://dx.doi.org/10.32604/csse.2022.016730.
[61] H.M. Balaha, H.A. Ali, Hybrid COVID-19 segmentation and recognition framework using deep learning and genetic algorithms, Artif. Intell. Med. (2021) 102–156, http://dx.doi.org/10.1016/j.artmed.2021.102156.
[62] M. Gong, L. Jiao, D.D. Yang, W.P. Ma, Evolutionary multi-objective optimization algorithms, J. Softw. (2009).
[63] X. You, H. Yang, Z. Luan, et al., Performance evaluation and analysis of linear algebra kernels in the prototype tianhe-3 cluster, in: Proceedings Of The Asian Conference On Supercomputing Frontiers, 2019, pp. 86–105.
[64] P. Foret, A. Kleiner, H. Mobahi, B. Neyshabur, Sharpness-aware minimization for efficiently improving generalization, in: Proceedings Of The International Conference On Learning Representations, 2021.
[65] E. Hoffer, I. Hubara, D. Soudry, Train longer, generalize better: closing the generalization gap in large batch training of neural networks, in: Proceedings Of The Conference On Neural Information Processing Systems, 2017, pp. 1731–1741.
[66] Xin-She Yang, Suash Deb, Cuckoo search via Lévy flights, in: World Congress On Nature & Biologically Inspired Computing, NaBIC 2009, 9-11 December 2009, Coimbatore, India, IEEE, 2009, pp. 210–214, http://dx.doi.org/10.1109/NABIC.2009.5393690.
[67] Z. Shao X, J. Qian, An optimizing method based on autonomous animats : Fish-swarm algorithm, Syst. Eng. Theory Pract. 22 (2002).

Jia Wei received the B.E. degree from the School of Information science and technology, NorthWest University, Xi'an, China, in 2019. He is currently a Ph.D. candidate with the Computer science and technology School, Xi'an Jiaotong University. His research interests include computer architecture, high performance computing, and deep learning.

Xingjun Zhang received his Ph.D. degree in Computer Architecture from Xi'an Jiaotong University, China, in 2003. From Jan. 2004 to Dec. 2005, he was Postdoctoral Fellow at the Computer School of Beihang University, China. From Feb. 2006 to Jan. 2009, he was Research Fellow in the Department of Electronic Engineering of Aston University, United Kingdom. He is now a Full Professor and the Dean of the School of Computer Science & Technology, Xi'an Jiaotong University. His research interests include high performance computing, big data storage system and machine learning acceleration.

Zeyu Ji received the B.S. degree from the School of Information Engineering, Zhengzhou University, Zhengzhou, China, in 2008, and the M.S. degree from the Polytechnic University of Tours, Tours, France. He is currently a Ph.D. candidate with Xi'an Jiaotong University, Xi'an, China. His main research interests include computer architecture, high performance computing, and deep learning.

Zheng Wei received the B.S. and M.S degrees from the school of Communication Engineering from Xidian University, Xi'an, China, in 2013 and 2016, respectively. He is currently pursuing a Ph.D. degree with Xi'an Jiaotong University, Xi'an, China. His research interests include computer architecture, deep compression algorithms, and hardware accelerators for deep learning.

Jingbo Li is currently pursuing a Ph.D. degree with Xi'an Jiaotong University, Xi'an, China. His main research interests include computer architecture, job scheduling, and high performance computing.