An Investigation into Hyperparameter Adjustment and Learning Rate Optimization Algorithm Utilizing Normal Distribution and Greedy Heuristics in Parallel Training

Weixuan Qiao

Software engineering, Harbin University of Science and Technology, Weihai Shandong, 264300, China
Email:2030090319@stu.hrbust.edu.cn

Abstract. In the era of large models, traditional training methods can no longer meet the massive requirements of computing power and data sets. Using distributed training can alleviate this problem to some extent. However, in distributed training, the challenge and complexity of hyperparameter adjustment will increase. In order to solve these challenges, special distributed hyperparameter adjustment algorithms and strategies are needed to find the best hyperparameter combination faster. This paper proposed Learning Rate Search Algorithm (LRSA) to quickly determine the initial value of learning rate, which makes the adjustment of hyperparameter more efficient. This work analyzed the reason why the parallel speed of multi card data decreases is that Using data parallelism at a small batch size can lead to resources being mainly used for communication and related overhead between GPUs, resulting in lower effective utilization of GPUs and an increase in training duration. Furtherly, this paper explored the reasons for the decrease in accuracy under large batch size and used LRSA to improve this situation effectively. This article also proposed an empirical rule to determine the lower bound of batch size. Experimental results indicate that LRSA on several deep learning models demonstrated that they can improve training efficiency and accuracy. For example, LRSA was used in VGG16 to get a suitable learning rate so that the model has the same Accuracy as the smaller batchsize at a faster training speed.

Keywords: Data Parallel, Deep Learning, LRSA

1 Introduction

Artificial intelligence has witnessed remarkable progress in the past few years, particularly in the realm of deep learning, where substantial advancements have been achieved. One of the most significant breakthroughs in deep learning has been the development of large language models like GPT-3.5 [1], It can produce text that resembles that of a human being and do a variety of tasks involving natural language processing with astounding precision. Large language models have shown exceptional performance in various tasks such as language modeling, question-answering, sentiment analysis, vision, and machine translation [2-4]. Some studies used the

P. Kar et al. (eds.), *Proceedings of the 2023 International Conference on Image, Algorithms and Artificial Intelligence (ICIAAI 2023)*, Advances in Computer Science Research 108,
https://doi.org/10.2991/978-94-6463-300-9_16

Theory of mind (ToM) task for testing, and found that the large model headed by GPT3 seems to have a part of the mind [5]. These models have the ability to learn from vast amounts of data, allowing them to capture complex patterns and nuances in language that were previously impossible to achieve. However, the achievements of these models are accompanied by a notable trade-off, which demands substantial computational resources and expansive datasets for effective training.

In order to tackle this challenge, distributed training has become increasingly important. In order to facilitate quicker and more effective training, distributed training divides the training process into smaller pieces and distributes them among several devices, such as GPUs or CPUs. The utilization of multiple devices enables a balanced distribution of computational workload, leading to substantial reductions in training time. Additionally, distributed training allows for larger models to be trained more efficiently, as the computational resources can be scaled up more easily. There are currently three mainstream solutions in distributed training. Discarding all activation tensors in the forward propagation and recompute during backpropagation, such as Gpipe [6]. Keeping activation tensors produced during forward propagation directly in GPU memory, such as PipeDream [7]. Combining the above two methods and dynamically partitioning stages, such as Vpipe [8].

In the context of distributed training, the management of hyperparameters assumes heightened significance due to the partitioning of training data across multiple computing nodes. This means that the complexity and challenge of hyperparameter regulation will also increase, particularly in relation to the learning rate. Over the years, numerous learning rate scheduling algorithms have been proposed to improve the optimization performance of deep neural networks. These algorithms aim to adjust the learning rate dynamically during the training process, thereby facilitating accelerated convergence and improved generalization performance. The evolution of learning rate scheduling algorithms can be divided into several stages. Initially, fixed learning rates were used, which often resulted in suboptimal solutions. Subsequently, dynamic learning rate algorithms were introduced, such as Adagrad [9], RMSProp, and Adam [10], which, based on the gradient of the loss function or other variables, adaptively adjusts the learning rate.

To further determine and optimize the learning rate for achieving better performance of the model, his paper proposes an algorithm called LRSA for quickly determining the learning rate. Furthermore, for better hyperparameter regulation, this paper also explores the reasons for the increase in training time caused by multi card data parallelism and the decrease in accuracy under large batch size.

2 Method

2.1 Dataset Description and Preprocessing

The dataset used in this paper is CIFAR10, which is a benchmark image classification dataset that is commonly employed in the field of computer vision. It comprises of 60,000 color images, each measuring 32 pixels by 32 pixels, broken down into 10 classes, each containing 6,000 images.

This paper firstly converted images into the tensor type. Subsequently, the images are normalized so that the data ranges between [-1,1]. This can make the model easier to learn and improve the performance of the model. Finally, the dataset was transformed using the DataLoader function into a PyTorch dataloader object so that batching could be used during model training and testing to speed up those processes and enhance model performance.

2.2 CNN model

LeNet. LeNet, which debuted in 1998, is a ground-breaking convolutional neural network (CNN) model that excelled at recognizing digit handwriting [11]. Convolutional layers, pooling layers, and fully connected layers make up the majority of LeNet. It is distinguished by alternating between convolutional layers and pooling layers, with a pooling layer for extracting image features coming after each convolutional layer. The fully connected layer is used to classify the features and output the corresponding category probability.

VGG16. The Oxford University Computer Vision Group proposed the VGG16 deep convolutional neural network model in 2014. It is one of the most famous versions of the VGG network, consisting of 16 convolutional layers and fully connected layers. The main feature of VGG16 is the use of a large number of small convolution kernels, following each convolutional layer, there is a pooling procedure, which enables the VGG16 network to efficiently extract features from images. The model has demonstrated exceptional performance in image classification tasks, surpassing previous approaches on the challenging ImageNet dataset and emerging as a critical benchmark model in the field.

2.3 Data Parallelism

Data parallelism is a distributed deep learning training method, Its fundamental concept is to split the training data into various subgroups and give each subset a distinct computational node to calculate, and then summarize the calculated gradients to a central node, and update the model parameters based on the summarized gradients. The main purpose of data parallelism is to accelerate training and improve model performance, such as Parameter Server [12]. Specifically, the steps for data parallelism are as follows: 1) Dividing the training dataset into multiple subsets. 2) Assigning each subset to different computing nodes. 3) At each node, using the same model parameters for training and calculating the gradient corresponding to that subset. 4) Summarizing the gradients calculated by each node onto a central node. 5) Updating model parameters based on the aggregated gradient.

2.4 Hyperparameter regulation

Hyperparameter refer to parameters that need to be specified in advance during model training, for example, learning rate, regularization parameters, etc. The model's performance is significantly impacted by the choice of these hyperparameters, and different hyperparameter settings will lead to different model performance and performance. Therefore, the optimal combination of hyperparameter can be found through the adjustment of hyperparameter, so as to obtain better model performance.

However, in the context of distributed training, hyperparameter tuning poses additional challenges, primarily due to the following reasons:1) Multiple computing nodes. 2) Communication overhead. This is also one of the reasons why multi card data parallelism is actually slower. 3) Model complexity. 4) Long training time. Therefore, this paper proposes a search algorithm LRSA based on normal distribution and greedy thought, which is used for faster, automatic and higher accuracy of the initial learning rate.

2.5 Batchsize

In deep learning, the training data set is usually divided into several batches of equal size, which are subsequently fed to the neural network for training purposes. The size of each batch, referred to as Batchsize, exerts a notable influence on training speed and model stability. Larger Batchsize values tend to expedite the training process and enhance model stability. However, a larger Batchsize also consumes more memory and may cause overfitting problems in the model. Additionally, a bigger Batchsize can result in a reduction in the model's capacity for generalization. On the contrary, a smaller Batchsize can improve the generalization ability of the model, because the data in each batch is more random, which can reduce the risk of overfitting.

2.6 Learning Rate

In deep learning, the Learning Rate is a crucial hyperparameter that regulates the step size of the model each time the weight is changed. Although it could make the model unstable, a higher learning rate results in a faster weight update for the model. A lower learning rate results in a longer weight update for the model, and it may cause the model to converge too slowly or fall into a local optimal solution.

2.7 LRSA

As the model parameters and training set size increase, the cost of manually and empirically adjusting the learning rate also increases. Therefore, it is expected that more effective algorithms can achieve a better learning rate (higher accuracy, better generalization ability) with as little cost as possible. This paper proposes a lr adjustment algorithm Learning Rate Search Algorithm (LRSA) based on normal distribution and greedy thought shown in Algorithm 1.

According to the formula for gradient descent $w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in B} \nabla l(x, w_t)$, When increasing the batch size by n times, the number of parameter updates is equivalent to a decrease of n times. Therefore, a reasonable approach is to also increase lr by n times and use it as the starting point for searching. At the same time, since the position near the starting point has a greater probability, it is a better lr.

$$S = Lr \times n \tag{1}$$

$$\mathrm{d}(x) = \frac{k}{\sqrt{2\pi}} \mathrm{e}^{\left(-\frac{x^2}{2}\right)} \tag{2}$$

$$Lr^{'} = Lr \pm \mathrm{d}(s - Lr) \times dis \tag{3}$$

The meaning of each letter is as follows: S: starting point, obtained from the expansion multiple n of the original learning rate LR and batchsize, d (x): step coefficient based on normal distribution, which is larger near the starting point S and smaller away from the starting point S, where k is a numerical coefficient that can be

manually adjusted (default is 0.5) lr: the updated learning rate dis: basic step (default is 0.01)

Algorithm 1: Learning Rate Search Algorithm(LRSA)

Input： batch_size, new_batch_size,Lr
(Optional parameters: depth_max, error_max, k, distance)
Output： $Lr^{'}$

function init(batch_size, new_batch_size, lr)
    Initialization start point $S = Lr \times n$
    Set search parameters(such as: depth_max error_max…)
end function

function d(lr,k=0.5)
    return Step coefficient $\frac{k}{\sqrt{2\pi}} e^{\left(-\frac{(lr)^2}{2}\right)}$
end function

function check(lr)
    global accuracy_max
    accuracy = execute() #get accuracy from model
    if accuracy > accuracy_max:
        accuracy_max = accuracy
        if (accuracy_single - accuracy) / accuracy_single < error_max:
            exit()
        return True
    return False
end function

function Lrsa(Lr,cnt)
    if the end condition is met, exit.
    else:
        get Predicted Learning Rate $Lr^{'} = Lr \pm \mathrm{d}(s - Lr) \times dis$
        check($Lr^{'}$)
        if return True, continue searching Lrsa($Lr^{'}, cnt + 1$)
        else: Return to the previous layer
end function

2.8 Sharp Minimum Caused by Excessive Learning Rate

Although increasing batch size can greatly accelerate training speed, it also has some negative effects. According to research [13], whereas small batch sizes tend to converge to flat minimums, which are better at generalization, high batch sizes tend to converge to sharp minimums shown in Fig. 1. Therefore, blindly increasing the batch size is not advisable, it is necessary to choose a more moderate batch size. Therefore, finding a suitable lower bound for batch size has become a valuable issue.

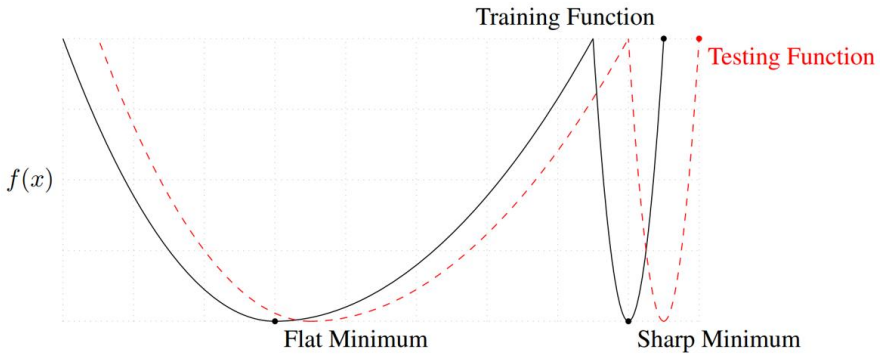

Fig. 1. An illustration of Flat and Sharp Minima. The loss function's value is shown on the Y-axis, while the parameters' values are shown on the X-axis (Photo/Picture credit: Original).

3 Results and Discussion

3.1 The Investigation and Discussion of Multi Card Data Parallelism Technology

Usually, data parallelism can significantly accelerate training, but in the above experiments shown in Table 1, it was found that after data parallelism, the training time actually increased significantly. This may be due to the following reasons: 1) Communication bottleneck: When multi card data is parallel, each card needs to send the calculation results to other cards for gradient updates. If the communication speed is slow, communication time may become a bottleneck, leading to an increase in training duration. 2) Improper adjustment of hyperparameter: when multi card data is parallel, the optimal value of hyperparameter may change. If the hyperparameter is set improperly, the training duration may increase.

Table 1. Testing VGG16 on RTX3090 with different number(DP: Data Parallel,SC: Single Card)

Type	Memory-Usage_1	Memory-Usage_2	Peak Volatile GPU-Util_1	Peak Volatile GPU-Util_2	Accuracy	Training Time
DP	7858MiB/ 24576MiB	2510MiB/ 24576MiB	58%	50%	79%	329.27s
SC	2930MiB/ 24576MiB	/	66%	/	80%	143.11s

The possibility exists that the employed batch size is inadequate, thereby engendering a considerable extent of communication overhead and GPU-switching overhead. Consequently, such inefficiencies contribute to a notable escalation in the duration required for training. In order to substantiate this conjecture, a series of experiments were undertaken shown in Table 2.

Table 2. Testing VGG16 on two RTX3090 sheets with different batch sizes

Batch Size	Memory-Usage_1	Memory-Usage_2	Peak Volatile GPU-Util_1	Peak Volatile GPU-Util_2	Accuracy	Training Time
128	3212MiB / 24576MiB	2898MiB / 24576MiB	66%	55%	78%	167.9s
256	3542MiB / 24576MiB	3452MiB / 24576MiB	73%	66%	74%	98.9s
512	4862MiB / 24576MiB	4700MiB / 24576MiB	80%	74%	71%	67.5s
1024	7332MiB / 24576MiB	7184MiB / 24576MiB	85%	81%	63%	61.49s
2048	12392MiB / 24576MiB	12454MiB / 24576MiB	91%	88%	57%	59.52s
4096	22678MiB / 24576MiB	22652MiB / 24576MiB	100%	100%	40%	61.68s

From the experimental data, it can be seen that:

1) When the batch size reaches 256, its training time is significantly lower than that of a single card. When the batch size is 2048, the training duration reaches its minimum of 59.525s.
2) As the batch size continues to increase, although the training duration continues to decrease, the degree of the decrease continues to decrease.
3) And when the batch size reaches 4096, its training duration becomes longer compared to the smaller batch size of 2048, and its Peak Volatile GPU Util_ 1, 2 achieved 100%
4) As the batch size increases, accuracy continues to decrease

The possible reasons for the above phenomena are as follows: The possible reason for 2) is that communication and other expenses are gradually increasing, and it is about to reach the cost boundary of distributed training; The possible reason for 3) is that the batch size is too large and memory is tight, resulting in one communication being split into multiple times, which increases the training time compared to situations where memory is relatively loose $w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in B} \nabla l(x, w_t)$, where n is the batch size and n is the learning rate [9].When the batch size increases, it is equivalent to fewer parameter updates, so the model is likely to be in an underfitting state, leading to a decrease in accuracy. Therefore, this situation can be improved by increasing the gleaning rate and increasing the number of epochs.

3.2 Investigation of the Influence of Batch Size in Accuracy

Related experimental results are presented in Table 3, Table 4 and Table 5. Although increasing the number of epochs can slightly improve accuracy, it quickly enters a bottleneck, which is likely caused by low learning rate. Meanwhile, Hoffer's research [14] demonstrated that the decrease in performance of large batch sizes is due to insufficient training time and is not essentially a problem with batch size.

Table 3. The impact of different epochs on Accuracy on two RTX3090 sheetswith batch size=2048 and learning rate=0.001

epoch	Accuracy	Training Time
10	56%	57.63s
20	61%	114.64s
30	61%	175.88s
40	61%	224.13s

Table 4. The impact of different learning rate on Accuracy on two RTX3090 sheets with batch size=2048 and epoch=10

learning rate	Accuracy	Training Time
0.001	56%	57.63s
0.005	68%	64.46s
0.01	73%	60.8s
0.015	77%	64.07s
0.02	78%	60.90s
0.025	78%	60.02s
0.03	79%	58.64s
0.04	80%	59.22s
0.06	80%	60.99s

Table 5. The impact of different learning rate on Loss on single RTX3090 with batch size=2048 and epoch=10

learning rate	loss
0.0001	2.177
0.001	1.074
0.01	0.173
0.03	0.132
0.04	0.128
0.06	0.169
0.1	0.394
1	nan

After continuously improving the learning rate, accuracy has shown a significant increase. When the learning rate is between 0.4 and 0.6, the accuracy achieved by the model is already the same as that of a single card. The loss will progressively drop and then climb. When the learning rate is too high, a gradient explosion event will occur. Therefore, the optimal learning rate should be selected from the area with the smallest loss.

3.3 Investigation of the Lower Bound of Batchsize

On multiple models, the results are roughly presented shown in Table 6, Table 7, Table 8 and Table 9: when the Average Peak volatile GPU-Util is similar to the Peak volatile GPU-Util of a single card, its training time is also similar to a single card.

Table 6. Testing the training time of VGG16 under different GPU-Util on two RTX3090 (data parallel)

Batch Size	Peak Volatile GPU-Util_1	Peak Volatile GPU-Util_2	Average Peak Volatile GPU-Util	Training Time
128	66%	55%	60.50%	167.9s
144	67%	60%	63.50%	144.32s
160	69%	62%	65.50%	131.85s

Table 7. Testing the training time of VGG16 under different GPU-Util on single RTX3090

Batch Size	Memory-Usage	Peak Volatile GPU-Util	Training Time
64	2930MiB/24576MiB	66%	143.11s

Table 8. Test the training time of LeNet under different GPU-Util on two RTX3090 (data parallel)

Batch Size	Peak Volatile GPU-Util_1	Peak Volatile GPU-Util_2	Average Peak Volatile GPU-Util	Training Time
64	6%	4%	5%	153.46s
80	6%	5%	5.50%	141.26s
96	6%	5%	5.50%	137.3s

Table 9. Test the training time of LeNet under different GPU-Util on single RTX3090

Batch Size	Memory-Usage	Peak Volatile GPU-Util	Training Time
64	2086MiB / 24576MiB	6%	132.98s

Henceforth, it is plausible to employ the GPU-Util metric as a criterion for determining an appropriate batch size selection. Specifically, it is advisable to opt for a batch size that surpasses the lower threshold when the Average Peak Volatile GPU-Util exceeds the Peak Volatile GPU-Util observed on a single card. Subsequently, additional adjustments can be made to refine the chosen batch size accordingly.

3.4 Investigation of Dynamically Adjusting the Initial Value of Learning Rate (LRSA)

From Table 10, it can be seen that LRSA quickly selected a better LR. Due to the maximum accuracy error between the selection and a single card not exceeding 1%, LRSA will continue to try to search for better values, and ultimately obtain a LR=0.03236 that is equivalent to a single card accuracy.

Table 10. Test the Accuracy and Loss of VGG16 using LRSA （error_max=0.01, depth_max=3）on two RTX3090

Predicted Learning Rate	Accuracy	Loss	Training Time
0.032	79%	0.144	65.29s
0.03399	79%	0.138	64.46s
0.03598	79%	0.136	64.8s
0.03798	78%	0.162	64.07s
0.03421	79%	0.139	63.25s
0.03236	80%	0.121	62.43s

4 Conclusion

With the increasing application of large models in the field of AI, the demand for computing power and data sets is also increasing. Through parallel processing, distributed training systems can quickly train large models and can efficiently utilize computing resources. However, tuning of hyperparameters in distributed training systems is more difficult. In order to deal with this problem, this paper proposes a dynamic learning rate search algorithm to obtain a better initial value. In the meanwhile, this paper also explored the reasons for the increase in training time caused by multi card data parallelism, explored the reasons for the decrease in accuracy under large batch size, and proposed a method for determining the lower bound of batchsize. This article verifies the above reasons in LeNet and VGG16 and uses LRSA in VGG16 to get a suitable learning rate so that the model has the same Accuracy as the smaller batchsize at a faster training speed. In the future, it is planned to combine LRSA with other dynamic learning rate adjustment algorithms such as Adam, so that the model can achieve higher accuracy and training speed.

References

1. Brown, T., Mann, B., Ryder, N., et al.: Language models are few-shot learners. Advances in neural information processing systems, 33: 1877-1901 (2020).
2. Liu, Z., Lin, Y., Cao, Y., et al.: Swin transformer: Hierarchical vision transformer using shifted windows. Proceedings of the IEEE/CVF international conference on computer vision. 10012-10022 (2021).
3. Raffel, C., Shazeer, N., Roberts, A., et al.: Exploring the limits of transfer learning with a unified text-to-text transformer. The Journal of Machine Learning Research, 21(1): 5485-5551 (2020).
4. Fedus, W., Zoph, B., Shazeer, N.: Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. The Journal of Machine Learning Research, 23(1): 5232-5270 (2022).
5. Kosinski, M.: Theory of mind may have spontaneously emerged in large language models. arXiv preprint arXiv:2302.02083 (2023).

6. Huang, Y., Cheng, Y., Bapna, A., et al.: Gpipe: Efficient training of giant neural networks using pipeline parallelism. Advances in neural information processing systems, 32 (2019).
7. Narayanan, D., Harlap, A., Phanishayee, A., et al.: PipeDream: Generalized pipeline parallelism for DNN training. Proceedings of the 27th ACM Symposium on Operating Systems Principles, 1-15 (2019).
8. Zhao, S., Li, F., Chen, X., et al.: v pipe: A virtualized acceleration system for achieving efficient and scalable pipeline parallel dnn training. IEEE Transactions on Parallel and Distributed Systems, 33(3): 489-506 (2021).
9. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7) (2011_.
10. Kingma, D. P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).
11. Le, Y., Bottou, L., Bengio, Y., et al.: Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11): 2278-2324 (1998).
12. Li, M., Andersen, D. G., Park, J. W., et al.: Scaling distributed machine learning with the parameter server. 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14). 583-598 (2014).
13. Keskar, N. S., Mudigere, D., Nocedal, J., et al. On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836 (2016).
14. Hoffer, E., Hubara, I., Soudry, D.: Train longer, generalize better: closing the generalization gap in large batch training of neural networks, Advances in Neural Information Processing Systems. 1731-1741 (2017).