CowClip: Reducing CTR Prediction Model Training Time from 12 hours to 10 minutes on 1 GPU

Zangwei Zheng[1,†] **Pengtai Xu**[1,†] **Xuan Zou**[2] **Da Tang**[2] **Zhen Li**[2]
Chenguang Xi[2] **Peng Wu**[2] **Leqi Zou**[2] **Yijie Zhu**[2] **Ming Chen**[2]
Xiangzhuo Ding[2] **Fuzhao Xue**[1] **Ziheng Qing**[1] **Youlong Cheng**[2] **Yang You**[1,‡]
[1]Department of Computer Science, National University of Singapore
[2]Bytedance Inc.
{zangwei, f-xue, zihengq, youy}@comp.nus.edu.sg
{zouxuan, pengtai.xu, da.tang, zhen.li1, chenguang.xi, peng.wu, leqi.zou,
yijie.zhu, ming.chen, xiangzhuo.ding, youlong.cheng}@bytedance.com

Abstract

The click-through rate (CTR) prediction task is to predict whether a user will click on the recommended item. As mind-boggling amounts of data are produced online daily, accelerating CTR prediction model training is critical to ensuring an up-to-date model and reducing the training cost. One approach to increase the training speed is to apply large batch training. However, as shown in computer vision and natural language processing tasks, training with a large batch easily suffers from the loss of accuracy. Our experiments show that previous scaling rules fail in the training of CTR prediction neural networks. To tackle this problem, we first theoretically show that different frequencies of ids make it challenging to scale hyperparameters when scaling the batch size. To stabilize the training process in a large batch size setting, we develop the adaptive Column-wise Clipping (CowClip). It enables an easy and effective scaling rule for the embeddings, which keeps the learning rate unchanged and scales the L2 loss. We conduct extensive experiments with four CTR prediction networks on two real-world datasets and successfully scaled 128 times the original batch size without accuracy loss. In particular, for CTR prediction model DeepFM training on the Criteo dataset, our optimization framework enlarges the batch size from 1K to 128K with over 0.1% AUC improvement and reduces training time from 12 hours to 10 minutes on a single V100 GPU. Our code locates at `https://github.com/bytedance/LargeBatchCTR`.

1 Introduction

With the development of the Internet and the e-economy, numerous clicking happens in online shopping [39, 69], video apps [15, 59] and web advertisements [8, 67]. In a typical industrial dataset, the number of click samples has grown up to hundreds of billion [59, 67] and keeps increasing on a daily basis. The click-through rate (CTR) prediction task is to predict whether a user will click on the recommended item. It is a fundamental task in advertising and recommendation systems. Since CTR prediction is a time-sensitive task [67], it is necessary to shorten the time needed for re-training on a massive dataset to maintain an up-to-date CTR prediction model. In addition, given a constant computing budget, decreasing the training time also reduces the training cost, giving rise to a high return-to-investment ratio.

[†]Work done during an internship at Bytedance.
[‡]Yang You is the corresponding author.

Preprint.

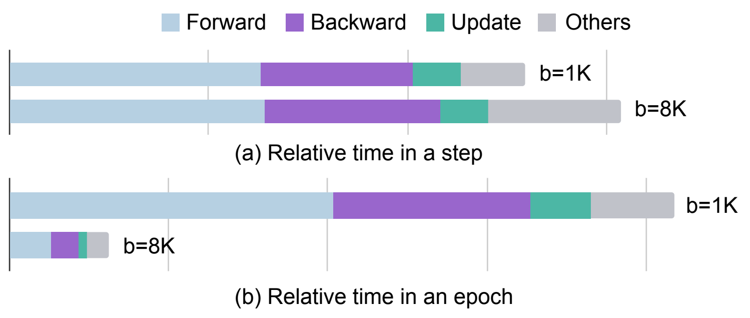


Figure 1: Relative time of training DeepFM model on Criteo dataset with one V100 GPU.

An accurate CTR prediction can directly improve user experience [26] and enhance ads profit [13, 56]. As deep learning achieved remarkable performance in computer vision (CV) [20, 30] and natural language processing (NLP) tasks [10, 53], various neural networks[7, 19, 36, 54, 55, 64] have been proposed for CTR prediction to achieve a better result compared with traditional methods (e.g., logistic regression, factorization machine). Although deep learning benefits the accuracy of CTR prediction, training a neural network requires more computational power.

Recent years have witnessed rapid growth in GPU processing ability [6]. As the powerful hardware greatly benefits the progress in CV and NLP [4, 30], there is also a transition in accelerating CTR prediction model training from multi-node CPU servers[23, 34] to GPU accelerators[7, 46]. Recently, many big companies have developed their own CTR prediction training framework [18, 25, 37, 43, 44, 59, 62, 67, 68] to support a memory and communication efficient system to exploit GPU ability fully. In this trend, CTR prediction systems tend to be synchronously trained due to the accuracy degradation brought by the staleness of gradients in asynchronous training [12, 44].

With the growth of GPU memory and FLOPS, a larger batch size can take better advantage of the parallel processing capability of GPUs. As shown in Figure 1 (a), the time of one forward and backward pass is almost the same when scaling 8 times batch size, indicating GPU with a small batch size is extremely underused. Since the number of training epochs remains the same, large batch training reduces the number of steps and thus significantly shortens the total training time (Figure 1 (b)). In addition, a large batch benefits more in a multi-GPUs setting, where gradients of the large embedding layer need to be exchanged between different GPUs and machines, resulting in high communication costs. To avoid distraction from system optimization in reducing communication costs [44, 59, 67], we focus on designing an accuracy-preserving algorithm for scaling batch size on a single GPU, which can be easily extended for multi-node training.

The challenge of applying large batch training is an accuracy loss when naively increasing the batch size [22]. One possible reason for this phenomenon is that training with a large batch may impair the network's generalization ability [27]. As discussed in Section 2, CTR prediction is more sensitive and cannot bear the accuracy loss. In CV and NLP tasks, equipped with hyperparameter scaling rules [17, 29] and a carefully designed optimization method [60, 61], it is possible to use a large batch size of 32K to train the ResNet-50 [20] and BERT [61] without accuracy loss and reduces the training time of BERT from 3 days to 76 minutes [61]. In this paper, we identified the failure reason behind previous scaling rules on CTR prediction and proposed an effective algorithm and scaling rule for large batch training.

In conclusion, our contributions are as follows:

- To the best of our knowledge, we are the first to investigate the stability of training CTR prediction model in very large batch sizes. we attribute the hardship in scaling the batch size to the difference in id frequencies.
- With rigorous mathematical analysis, we prove that the learning rate for unfrequent features should not be scaled when scaling up the batch size. With CowClip, we can adopt an easy and effective scaling strategy for scaling up the batch size.
- We propose an effective optimization method of adaptive Column-wise Clipping (CowClip) to stabilize the training process of the CTR prediction task. We successfully scale up 128 times batch size for four models on two public datasets. In particular, we train the DeepFM model with 72 times speedup and 0.1% AUC improvement on the Criteo dataset.

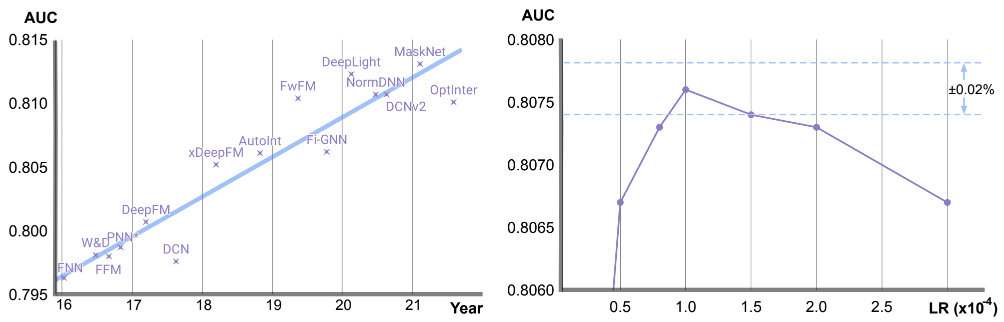


Figure 2: **Left**: Progress on AUC of CTR prediction models on Criteo dataset in the past six years. **Right**: AUC performance of DeepFM on Criteo dataset with different learning rates. The one-year improvement is about 0.3%. The accuracy drops a lot when a slight disturbance is made.

2 Related Work

Sensitiveness of CTR prediction Preserving the CTR prediction model's performance with a large batch size is a great challenge in that the CTR model is very sensitive. In accelerating the training of ResNet-50, an accuracy loss within 1% is tolerable [31, 60, 61]. However, considering the tremendous amounts of clicking happening every day, a 0.1% loss in AUC will cost a company too much to bear. As shown in Figure 2 left, a continuous effort on developing new CTR prediction models only improved the AUC by less than 2% on the Criteo [32] dataset in the past six years., and an improvement in a month of 0.02% is considered significant in Criteo dataset in this paper. In our experiments, a tiny shift in learning rate results in a significant 0.02% drop in model performance at batch size 1K (Figure 2, right), indicating the task is sensitive to hyperparameters.

Large Batch Training Methods To preserve the performance of deep models at a large batch size, we need a good scaling rule and a stable optimization strategy. The scaling rule tells us how to scale the hyperparameters when scaling up the batch size. The two most important hyperparameters when scaling the batch size are learning rate and regularization weight. Based on different assumptions, linear scaling [17] and square root scaling [24, 29] are the two most common scaling rules in the deep learning community. Besides, optimization strategies such as warmup [16], gradient clipping [63] can help stabilize the large batch training process. LARS [60] and LAMB [61] are two optimizers designed for large batch training, which adopt different adaptive learning rates for each layer. Although they achieve good results in CV and NLP tasks, they are ineffective in the CTR prediction task because it is unnecessary to use a layer-wise optimizer with a shallow network (e.g., three or four layers). This paper re-designs the scaling rule and optimization strategy for the embedding layer, which can successfully scale up the batch size for CTR prediction.

Additional related work: CTR prediction networks (Appendix A), works utilizing different frequencies (Appendix C), and properties of CTR prediction input (Appendix B).

3 Method

In CTR prediction, we have the training dataset $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$, where $y \in \{0, 1\}$ denotes whether the user clicked or not. The $\boldsymbol{x}$ contains information about the user, the product and the interaction, which can be categorical or continous. The categorical field is one-hot encoded to be a vector $\boldsymbol{x}_i^{\mathrm{f}_j}$ of d_{f_j} length, where d_{f_j} is the number of possible values (ids) in this field. To represent the frequency of each id, we denote the k-th id in field j as $\mathrm{id}_k^{\mathrm{f}_j}$. The frequency and occurrence probability of the id is:

$$\texttt{count}(\mathrm{id}_k^{\mathrm{f}_j}) = \sum_{i=1}^{N} \delta(\boldsymbol{x}_i^{\mathrm{f}_j}[k] = 1); \quad \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in \boldsymbol{x}) = \frac{\texttt{count}(\mathrm{id}_k^{\mathrm{f}_j})}{N},$$

where $\delta(\cdot)$ equals 1 if the boolean condition holds and 0 otherwise.

Given the predicting network f, the prediction is made from $f(\boldsymbol{x})$. The network and embeddings weights are denoted as w, and the training loss is L. This paper focuses on the Wide/Cross-and-Deep kind of CTR prediction model, as briefly described in Figure 3 right, one of the state-of-the-art networks in CTR prediction [55, 64].

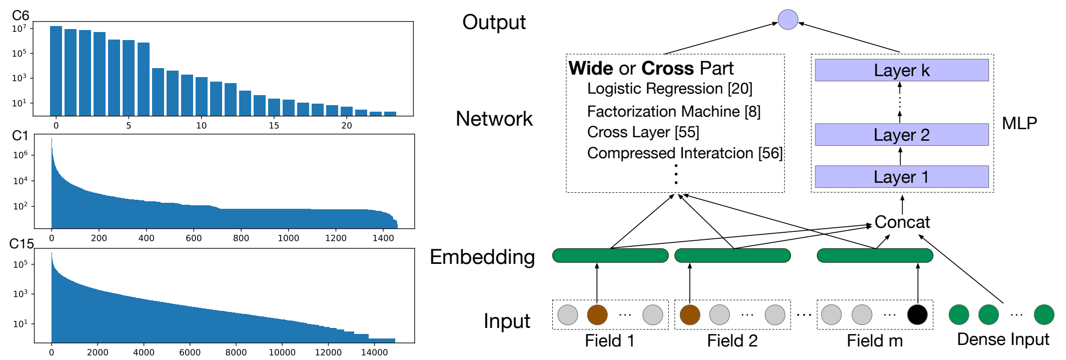


Figure 3: **Left**: Distribution of different ids in three fields of the Criteo dataset. The x-axis is the id number, and the y-axis is in logarithm scale. The total number of samples is 4.13×10^7. **Right**: A simple illustration of a Wide/Cross-and-Deep kind of CTR prediction model. The green data denotes a dense one, while brown input in a categorical field stands for a selected id.

In training the network, we use a batch size of $b = |B|$, where B is a specific batch. The learning rate and L2-regularization weight are denoted as η and λ. The total number of steps in an epoch is $\frac{N}{b}$.

3.1 Failure Cause of Traditional Scaling Rules

When training a neural network, at step t, an optimizer $\texttt{Opt}(\cdot)$ takes the weights and gradients, and output the updated weights. With L2-regularization, the update process is:

$$\boldsymbol{g}_t = \sum_{x \in B_t} \nabla L(w, x) + \frac{\lambda}{2} \cdot \|w\|_2^2; \quad w_{t+1} = \eta \cdot \texttt{Opt}(w_t, \boldsymbol{g}_t).$$

When changing the batch size, the hyper-parameter learning rate η and L2-regularization weight λ should be adjusted for maintaining the same performance as the original batch size.

Square root scaling [24, 29] and linear scaling [17] are two widely used scaling rules in deep learning. The motivation for sqrt scaling is to keep the covariance matrix of the parameters update the same, while for linear scaling, the motivation is to keep the update in a large batch equal to updates from s small batches when scaling s times the batch size (details in Appendix D). The two scaling rules have been shown effective in CV and NLP tasks, and they are shown as follows:

Scaling Rule 1 (Sqrt Scaling) *When scaling batch size from b to $s \cdot b$, do as follows:*

$$\eta \to \sqrt{s} \cdot \eta, \quad \lambda \to \sqrt{s} \cdot \lambda$$

Scaling Rule 2 (Linear Scaling) *When scaling batch size from b to $s \cdot b$, do as follows:*

$$\eta \to s \cdot \eta, \quad \lambda \to \lambda$$

Our first attempt at large batch training of the CTR prediction model is to apply the above classic scaling rules: no scaling, linear scaling [17], and square root scaling [29]. However, as seen in experiments on the Criteo dataset with DeepFM model in Table 1 left, the above rules fail in a CTR prediction model. We claim that the reason for the failure lies in the different frequencies of ids.

The product id of a popular item or ids in fields with a few options (e.g., male and female in gender field) are frequent, while the id of an inactive user seldom appears. In Figure 3 left, we visualize the distribution of different ids' frequencies in three fields. The exponential distribution reveals different frequencies among different ids. For the dense weights (e.g., kernel weights), their gradients appear for each sample while embedding does not have gradients if the corresponding ids do not show up. In CTR prediction, the embedding layers dominate the parameters of the whole network (e.g., 99.9%), and different occurrences of gradients make a great difference from other deep neural networks.

First, we empirically verify our claim by the following experiment. We keep the top three frequent ids in each field and label the rest as a fourth id. In this way, all four ids are very frequent and variations in frequencies are ablated in this modified version of Criteo. As shown in Table 1 right, both scaling rules successfully apply to the modified dataset, which means the traditional scaling rule does not work in CTR prediction due to the presence of infrequent ids.

Table 1: AUC (%) of different scaling strategies on Criteo with DeepFM and a modified version with the first three frequent ids in each field. Previous scaling rules fail on Criteo but work for a revised version. BS denotes batch size.

	Criteo			Criteo (Top 3 frequent ids)		
BS	No Scale	Sqrt Scale	Linear Scale	No Scale	Sqrt Scale	Linear Scale
1024	80.76	80.76	80.76	74.97	74.97	74.97
2048	80.61	80.75	80.77	74.87	74.96	75.01
4096	79.41	80.69	80.65	74.77	74.95	74.95
8192	77.55	80.55	80.46	74.69	74.96	74.96

Next, we provide the theoretical analysis for the failure of sqrt and linear scaling. Different frequencies lead to varying occurrences of ids in batches. Only when the id appears in the batch can the corresponding embedding be updated. They only occur in a small fraction of batches for ids with a low frequency. Suppose we draw the training samples with replacement from the dataset, the probability of an id $\mathrm{id}_k^{\mathrm{f}_j}$ in the batch B is:

$$\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B) = 1 - (1 - \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in \boldsymbol{x}))^b.$$

For frequent ids, and also dense weights whose frequency rate is 1, we have $(1 - \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in \boldsymbol{x}))^b \approx 0$; while for the unfrequent ids, when $p \ll \frac{1}{B}$, we can use binomial approximation and obtain:

$$\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B) \approx \begin{cases} 1 & \mathrm{id}_k^{\mathrm{f}_j} \text{ is frequent} \\ b \cdot \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in \boldsymbol{x}) & \mathrm{id}_k^{\mathrm{f}_j} \text{ is unfrequent} \end{cases}. \tag{1}$$

Now, reconsider the linear scaling motivation for an id's embedding w. Denote the weight update as $\Delta w = w_t - w_{t+1}$. Consider the expected update in a large batch $B' = \bigcup_{i=1}^{s} B_i$ with $b' = |B'| = s \cdot b$, we have

$$\mathbb{E}[\Delta w] = \mathbb{E}[\eta' \cdot \delta(\mathrm{id}_k^{\mathrm{f}_j} \in B') \cdot \frac{1}{b'} \sum_{x \in B'} \nabla L(w, x)] = \eta' \cdot \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B') \cdot \mathbb{E}[\nabla L(w, x)].$$

With the assumption that $\mathbb{E}[\nabla L(w_i, x)] \approx \mathbb{E}[\nabla L(w, x)]$, the expected update in small batches B_i is:

$$\mathbb{E}[\Delta w] = \mathbb{E}[\eta \cdot \sum_{i=1}^{s} \delta(\mathrm{id}_k^{\mathrm{f}_j} \in B_i) \cdot \frac{1}{b} \sum_{x \in B_i} \nabla L(w_i, x)] \approx \eta \cdot s \cdot \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B) \cdot \mathbb{E}[\nabla L(w, x)].$$

For dense weight or embeddings of frequent id, the term $\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B)$ equals 1, making no difference to the original linear scaling rule. However, with an unfrequent id, it shows that the new scaling strategy should be using the same learning rate when scaling the batch size due to the following fact for unfrequent ids:

$$\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B') \approx s \cdot \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B).$$

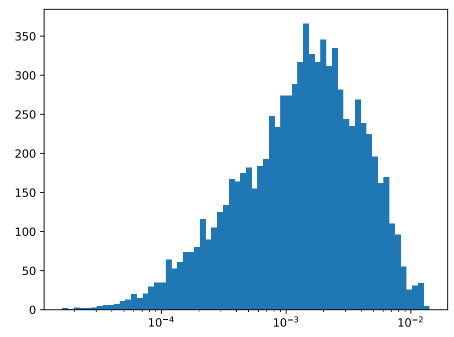
A similar discussion based on sqrt scaling motivation (see Appendix E) shows that under a very strong assumption can we obtain the same conclusion. However, without the assumption, we cannot even choose hyperparameters maintaining the same covariance matrix after scaling the batch size.

When using a relatively small batch size we find most ids satisfied $p < \frac{1}{B}$. Thus, we propose to use no scaling on the whole embedding layers, which suits unfrequent ids. In addition, a smaller learning rate for layers at the bottom leads to a smooth learning process. Experiments show this scaling rule leads to a better result.

After the discussion of learning rate scaling, now let's turn to the L2-regularization weight λ. In CTR prediction, an unsuitable λ can easily lead to overfitting. For the scaling of λ, we first consider the embedding vector w of $\mathrm{id}_k^{\mathrm{f}_j}$, the expected gradient of which in a batch is:

$$\mathbb{E}[\boldsymbol{g}] = \frac{1}{b} \mathbb{E}[\delta(\mathrm{id}_k^{\mathrm{f}_j} \in B) \sum_{x \in B} \nabla L(w, x)] = \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B) \cdot \mathbb{E}[\nabla L(w, x)]. \tag{2}$$

The term $\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B)$ still has no effect with dense weight and embeddings of frequent ids as the probability equals to 1. However, for the unfrequent ids, there is a scaling multiplier before the


(a) Gradients L2-norm.

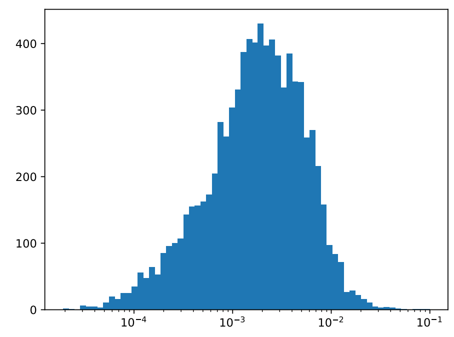

(b) Gradients L2-norm divided by occurrence.

Figure 4: L2-norm distribution of different columns gradients at 1000th step of DeepFM on Criteo dataset. Only columns with existing ids in the batch are shown. The x-axis is the L2 norm value and y-axis is the count of columns.

expectation of the gradient as some ids may not appear in a certain batch. When using an adaptive optimizer such as Adam, this scaling multiplier results in a different behaviour, which is equivalent to adjusting the L2-regularization weight λ as follows (see Appendix F for proof):

$$\frac{\lambda}{\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B)} = \frac{\lambda}{b \cdot \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in \boldsymbol{x})}.$$

Thus, to maintain the same L2-regularization strength, we scale up the λ by n. Combined with the learning rate scaling rule, we have the following one.

Scaling Rule 3 (CowClip Scaling) *When scaling batch size from b to $s \cdot b$, use sqrt scaling for the dense weights, and do as follows for embeddings:*

$$\eta_e \to \eta_e, \quad \lambda \to s \cdot \lambda$$

However, we find directly applying the above rule leads to overfitting due to s times less application of L2-regularization when scaling up batch size by s times. If no additional regularization technique is introduced, L2-regularization should be strengthened further with a large batch size. In the case of an SGD optimizer, we have (details in Appendix D):

$$\eta' \lambda' \approx s \eta \lambda.$$

Hence, we need to further scale up the λ by s times when the learning rate is unchanged. Although the behavior of adaptive optimizers such as Adam is different from SGD, we find a larger λ prevents overfitting. Thus, we have the following scaling rule which can scale up the batch size to 4K without additional optimization strategy.

Scaling Rule 4 (n^2–λ Scaling) *When scaling batch size from b to $s \cdot b$, use sqrt scaling for the dense weights, and do as follows for embeddings:*

$$\eta_e \to \eta_e, \quad \lambda_e \to s^2 \cdot \lambda_e$$

3.2 CowClip Algorithm

Although the Scaling Rule 4 helps us scale to 4 times the original batch size, it fails on a larger batch size. The challenge in choosing the proper learning rate η and L2-weight λ mentioned above impairs the performance for larger batch sizes. To enable large batch training, the gradient norm clipping [63] can smooth the process of training and alleviate the sensitiveness of hyperparameters. Given a clip threshold `clip_t`, gradient norm clipping does follows:

$$\boldsymbol{g} \to \min\{1, \frac{\texttt{clip\_t}}{\|\boldsymbol{g}\|}\} \cdot \boldsymbol{g}$$

Gradient norm clipping smoothes the training process by reducing the norm of a large gradient greater than a threshold. However, it is hard to choose an appropriate threshold for clipping the norm. Besides, as one column of the embedding matrix represents the embedding vector for an id, Figure 4

Algorithm 1 Adaptive Column-wise Clipping(CowClip)

```
Input: CowClip coefficient $r$ and lower-bound $\zeta$, number of steps $T$, batch size $b$, learning rate for
    dense and embedding $\eta, \eta_e$, optimizer $\texttt{Opt}(\cdot)$
 1: for $t \leftarrow 1$ to $T$ do
 2:     Draw $b$ samples $B$ from $\mathcal{D}$
 3:     $\boldsymbol{g}_t, \boldsymbol{g}_t^e \leftarrow \frac{1}{b}\sum_{x \in B} \nabla L(x, w_t, w_t^e)$
 4:     $w_{t+1} \leftarrow \eta \cdot \texttt{Opt}(w_t, \boldsymbol{g}_t)$                // Update dense weights
 5:     for each field and each column in the field do
 6:         $n_{\boldsymbol{g}} \leftarrow \|\boldsymbol{g}_t^e[\mathrm{id}_k^{\mathrm{f}_j}]\|$
 7:         $\texttt{cnt} \leftarrow |\{x \in B | \mathrm{id}_k^{\mathrm{f}_j} \in x\}|$             // Number of occurrence
 8:         $\texttt{clip\_t} \leftarrow \texttt{cnt} \cdot \max\{r \cdot \|w_t^e[\mathrm{id}_k^{\mathrm{f}_j}]\|, \zeta\}$      // Clip norm threshold
 9:         $\boldsymbol{g}_c \leftarrow \min\{1, \frac{\texttt{clip\_t}}{n_{\boldsymbol{g}}}\} \cdot \boldsymbol{g}_t^e[\mathrm{id}_k^{\mathrm{f}_j}]$          // Gradient clipping
10:         $w_t^e[\mathrm{id}_k^{\mathrm{f}_j}] \leftarrow \eta_e \cdot \texttt{Opt}(w_t^e[\mathrm{id}_k^{\mathrm{f}_j}], \boldsymbol{g}_c)$     // Update the id embedding
```

shows that the magnitude of gradients for different columns varies. We denote the column for an id as $w[\mathrm{id}_k^{\mathrm{f}_j}]$. Clipping on the whole embeddings whose gradient norm is dominated by gradients of columns with large gradients impairs the ones with normal but smaller gradients. In addition, according to Equation (2), since we want to clip on $1 \cdot \nabla L(w, x)$, the different frequencies of ids lead to the scaler of $\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B)$ on the expected gradients. The variety of different gradients' norms still exists for the expected gradients $\nabla L(w, x)$ as shown in Figure 4 right.

To tackle the above problems, inspired by LAMB optimizer [61], which normalizes the norm of gradients of each kernel to be proportional to the norm of the kernel weight, we relate the clip threshold with the norm of id embedding vectors. The difference between our clipping method and the gradient norm clipping is three-fold: First, every id embedding vector has a unique clipping threshold for more flexible clipping. Second, the clipping threshold is multiplied by the occurrence number of the id to make sure the bound is based on $1 \cdot \nabla L(w, x)$. Last but not the least, the clipping threshold is calculated by the norm of the id vector in consideration of different magnitudes:

$$\texttt{clip}(\mathrm{id}_k^{\mathrm{f}_j}) = \mathrm{cnt}(\mathrm{id}_k^{\mathrm{f}_j}) \cdot \max\{r \cdot \|w_t^e[\mathrm{id}_k^{\mathrm{f}_j}]\|, \zeta\}$$

where $\texttt{cnt}(\mathrm{id}_k^{\mathrm{f}_j})$ is the number of occurence of the id in a batch.

As the weights grow larger in the training process, the benefit of a threshold proportional to the norm of the weight is that the clipping value adaptively grows with the network. As some unfrequent id embedding vectors become too small due to the continual application of L2-regularization with no id occurrence in steps, we restrict the clipping norm by a lower-bound ζ to avoid a too strong clipping.

The network training with CowClip is summarized in the Algorithm 1. In practice, tensor multiplication instead of for-loop is adopted for less computational overhead. Since CowClip stabilizes the training process, it is possible to use the CowClip scaling 3 rule to 128x batch size, leaving η_e unchanged and linear scaling the λ. Our large batch training framework contains the CowClip on the gradients and the CowClip scaling strategy for adjusting the hyperparameters.

4 Experiment

4.1 Experimental Setting

Datasets We evaluate our algorithms for large batch training and scaling strategy when scaling batch size on two public datasets. **Criteo** [32] is a widely used CTR prediction dataset. It collects 45M records on ad display information, and the corresponding user clicks feedback. There are 13 continuous fields and 26 categorical fields, which are all anonymized to protect users' privacy. Following [19, 64], the data is split into training and test sets by 90%:10%. **Avazu** [5] is another ad click-through dataset containing 32M training samples. It has 24 anonymous categorical fields. According to [64], we split the dataset into training and test sets by 80%:20%.

Implementation Details We use two popular metrics [41] in CTR prediction: AUC (Area Under ROC) and Logloss (Logistic loss). Four CTR prediction models are considered in our experiments: Wide-and-Deep Network (W&D) [7], DeepFM [19], Deep-and-Cross Network (DCN) [54], DCN

Table 2: Performance of different scaling methods on Criteo dataset from 1K to 8K on DeepFM.

	1K (1024)		2K (2048)		4K (4096)		8K (8192)	
	AUC (%)	LogLoss	AUC (%)	LogLoss	AUC (%)	LogLoss	AUC (%)	LogLoss
No Scaling	80.76	0.4438	80.66	0.4456	80.48	0.4518	80.31	0.4530
Sqrt Scaling	80.76	0.4438	80.71	0.4430	80.59	0.4450	80.28	0.4582
Sqrt Scaling*	80.76	0.4438	80.75	0.4444	80.69	0.4449	80.55	0.4547
LR Scaling	80.76	0.4438	80.77	0.4434	80.65	0.4434	80.46	0.4542
n^2–λ Scaling (Ours)	80.76	0.4438	80.86	0.4432	80.90	0.4426	80.73	0.4441
CowClip (Ours)	80.86	0.4430	80.93	0.4427	80.97	0.4422	80.97	0.4425

Table 3: Performance of CowClip methods on Criteo dataset from 1K to 128K on four models.

		baseline	1K	2K	4K	8K	16K	32K	64K	128K
DeepFM [19]	AUC (%)	80.76	80.86	80.93	80.97	80.97	80.94	80.95	80.96	80.90
	Logloss	0.4438	0.4430	0.4427	0.4422	0.4425	0.4424	0.4423	0.4429	0.4430
W&D [7]	AUC (%)	80.75	80.86	80.94	80.96	80.96	80.95	80.94	80.96	80.89
	Logloss	0.4439	0.4430	0.4424	0.4422	0.4425	0.4422	0.4428	0.4429	0.4434
DCN [54]	AUC (%)	80.76	80.86	80.93	80.96	80.97	80.98	80.95	80.99	80.91
	Logloss	0.4438	0.4429	0.4424	0.4422	0.4428	0.4419	0.4426	0.4426	0.4428
DCN v2 [55]	AUC (%)	80.78	80.87	80.94	80.97	80.98	80.97	80.95	80.97	80.89
	Logloss	0.4437	0.4429	0.4425	0.4422	0.4423	0.4420	0.4424	0.4427	0.4427

Table 4: Performance of CowClip methods on Avazu dataset from 1K to 128K on four models.

		baseline	1K	2K	4K	8K	16K	32K	64K	128K
DeepFM [19]	AUC (%)	78.84	78.83	78.82	78.90	79.06	79.01	78.82	78.82	78.80
	Logloss	0.3748	0.3751	0.3752	0.3752	0.3740	0.3759	0.3780	0.3781	0.3758
W&D [7]	AUC (%)	78.80	78.80	78.81	78.90	79.06	79.03	78.82	78.81	78.79
	Logloss	0.3752	0.3754	0.3752	0.3752	0.3744	0.3754	0.3782	0.3784	0.3758
DCN [54]	AUC (%)	78.82	78.80	78.81	78.91	79.05	78.97	78.74	78.78	78.79
	Logloss	0.3749	0.3754	0.3752	0.3751	0.3744	0.3760	0.3787	0.3780	0.3758
DCN v2 [55]	AUC (%)	78.84	78.83	78.82	78.89	79.07	78.97	78.80	78.81	78.75
	Logloss	0.3748	0.3750	0.3754	0.3751	0.3742	0.3760	0.3778	0.3779	0.3760

v2 [55](Details in Appendix A). Our implementation is based on Tensorflow [40] and DeepCTR [49] framework. Following the common network setting [19, 64], the dimension of categorical field embedding is 10, the depth of hidden layers for MLP is 3, and the number of neurons is 400 per layer. All models are trained with 10 epochs, and the final model is evaluated at the test set. We use optimizer Adam [28] and an L2-regularization on embedding layers. The base learning rate and L2-regularization weight on batch size 1024 are 10^{-4} and 10^{-5}. Scaling rules are performed based on the 1024 batch size. For CowClip, we use $r = 1$ and tune $\zeta \in \{10^{-5}, 10^{-4}\}$ due to a different initialization weight norm. We also use learning rate warmup [16] and larger initialization weights. More discussion on hyperparameter choice and techniques can be found in the Appendix G. The experiments are conducted on one Tesla V100 GPU.

4.2 Large Batch Training Results

We first compare different scaling strategies with the DeepFM model in Table 2 and Table 10 (in Appendix). Here Sqrt Scaling* is the variant version of Sqrt Scaling used in [19], which does not scale up the L2-regularization. CowClip denotes training with the CowClip method and the CowClip scaling rule 3. As we can see, traditional scaling rules fail to meet the AUC requirement with a loss of over 0.02%. This is consistent with results in [19], where results with 4k batch size are worse than those with 1k. With n^2–λ Scaling rule 4, it can scale batch size to 4K but fails with 8K. When successfully scaling the batch size, there is a performance gain in the AUC, which is also observed in [70]. For our CowClip algorithm, it outperforms the original optimization method by 0.1% AUC on the Criteo dataset at a small batch size. When scaling to a large batch size in Table 3 and Table 4, instead of AUC loss, our algorithm achieves a further performance gain of about 0.1% in the Criteo. CowClip can also scale up to 64x batch size for Avazu. Equipped with CowClip, it can scale all four models to a large batch size with performance improvement, as shown in Table 3 and Table 4.

Table 5: The training time of different methods on Criteo dataset. Last four are trained with CowClip.

			Time (minutes)							
	AUC (%)	Logloss	1K	2K	4K	8K	16K	32K	64K	128K
XDL [2]	80.2	0.452	196	$179^\dagger$	$160^\ddagger$	–	–	–	–	–
FAE [2]	80.2	0.452	122	$116^\dagger$	$104^\ddagger$	–	–	–	–	–
DLRM [45]	79.8	0.456	196	$133^\dagger$	$76^\ddagger$	–	–	–	–	–
Hotline [1]	79.8	0.456	53	$45^\dagger$	$39^\ddagger$	–	–	–	–	–
DeepFM	80.87	0.4428	768	390	204	102	48	27	15	9
W&D	80.86	0.4430	768	390	204	102	48	27	15	10
DCN	80.86	0.4429	768	390	204	102	48	28	17	11
DCN v2	80.87	0.4429	822	408	210	108	60	40	34	30
Speedup (DeepFM)			1x	1.96x	3.76x	7.52x	16.00x	28.44x	51.2x	76.8x

† Trained with 2 GPUs ‡ Trained with 4 GPUs.

Table 6: Ablation study of CowClip on Criteo with DeepFM.

	b = 8K		b = 128K	
	AUC (%)	LogLoss	AUC (%)	LogLoss
Gradient Clipping (GC)	80.63	0.4452	77.24	0.4953
Field-wise GC	80.63	0.4453	80.62	0.4454
Column-wise GC	80.65	0.4095	80.75	0.4432
Adaptive Field-wise GC	80.62	0.4453	77.90	0.4824
Adaptive Column-wise GC	80.97	0.4425	80.90	0.4430

Training with large batches can significantly reduce the training time. As shown in Table 5 (also Table 11 in Appendix), the speedup achieved by scaling up the batch size is almost linear when the batch size is under 16K. We can still accomplish a sublinear speedup when continuing to scale up the batch size and achieve a 76.8 times speed up with 128K batch size on the Criteo dataset. The compared four methods [1, 2, 45] take advantage of different systems, such as reducing the communication and computational cost, so they achieve a much faster training speed with a 1K batch size. However, these methods have a low AUC performance and can only scale up to a 4K batch size due to performance loss. Besides, they scale the batch size by using more GPUs, with 2 and 4 GPUs for 2K and 4K batch sizes, resulting in 2x and 4x cost in the GPU hours. Our method can achieve a much shorter training time within only one GPU resource and obtain a higher AUC score.

4.3 Ablation Study

Next, we show the superiority of CowClip over other clipping method designs with DeepFM on the Criteo dataset at batch size 8K and 128K in Table 6. While naive gradient clipping can help boost the performance with a large batch size, it fails with b=128K.Gradient clipping applied to a smaller unit yields better performance. Field-wise clipping is better than global clipping, and column-wise clipping is better than field-wise clipping. The field-wise adaptive gradient clipping fail to achieve a good result due to the different magnitude of column gradients in a field. CowClip (Adaptive Column-wise GC) outperforms all other methods in both settings. Hyperparameters for these clipping variants and more ablation study into the effectiveness of each component of CowClip can be found in the Appendix I.

5 Conclusion

To accelerate the training of CTR prediction models on one GPU, we have explored large batch training and found that different frequencies hinder the scaling of learning rate and L2-regularization weight when scaling the batch size. Since previous scaling rules used in CV and NLP fail, we propose a novel optimization strategy CowClip with a simple scaling rule to stabilize the training process for large batch training in CTR prediction system. Experiments show that our method successfully scales the batch size to the state-of-the-art number and achieves significant training speedup. Our CowClip algorithm is also applicable to other tasks with a large embedding table such as NLP tasks.

Broader Impact

Accelerating the training speed of the CTR prediction model by large batch training is directly beneficial to the ad-tech and e-commerce practicians. Time and cost are reduced for re-training a model, contributing to faster product development iterations. In addition, the personalized recommendation could be more accurate and up-to-date, which potentially improves the user experience.

References

[1] M. Adnan, “Accelerating input dispatching for deep learning recommendation models training,” Ph.D. dissertation, University of British Columbia, 2021.

[2] M. Adnan, Y. E. Maboud, D. Mahajan, and P. J. Nair, “Accelerating recommendation system training by leveraging popular choices,” in *Proceedings of the VLDB Endowment*, vol. 15, 2021, pp. 127–140.

[3] A. F. Agarap, “Deep learning using rectified linear units (relu),” *arXiv*, vol. abs/1803.08375, 2018.

[4] D. Amodei and D. Hernandez, *AI and compute*, 2018. [Online]. Available: `https://openai.com/blog/ai-and-compute/`.

[5] Avazu, *Avazu click-through rate prediction*, 2015.

[6] T. Baji, “Evolution of the GPU device widely used in AI and massive parallel processing,” in *2018 IEEE 2nd Electron Devices Technology and Manufacturing Conference (EDTM)*, IEEE, 2018, pp. 7–9.

[7] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. B. Aradhye, G. Anderson, G. S. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, “Wide & Deep learning for recommender systems,” in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, 2016.

[8] P. Covington, J. K. Adams, and E. Sargin, “Deep neural networks for YouTube recommendations,” in *Proceedings of the 10th ACM Conference on Recommender Systems*, 2016.

[9] W. Deng, J. Pan, T. Zhou, A. Flores, and G. Lin, “Deeplight: Deep lightweight feature interactions for accelerating CTR predictions in ad serving,” in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” in *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[11] J. C. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, 2010.

[12] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, “Slow and stale gradients can win the race,” *IEEE Journal on Selected Areas in Information Theory*, vol. 2, pp. 1012–1024, 2021.

[13] B. Edelman, M. Ostrovsky, and M. Schwarz, “Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords,” *American economic review*, vol. 97, no. 1, pp. 242–259, 2007.

[14] A. A. Ginart, M. Naumov, D. Mudigere, J. Yang, and J. Y. Zou, “Mixed dimension embeddings with application to memory-efficient recommendation systems,” in *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 2786–2791.

[15] C. Gomez-Uribe and N. Hunt, “The netflix recommender system: Algorithms, business value, and innovation,” *ACM Transactions on Management Information Systems (TMIS)*, vol. 6, 13:1–13:19, 2016.

[16] A. D. Gotmare, N. S. Keskar, C. Xiong, and R. Socher, “A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation,” *arXiv*, vol. abs/1810.13243, 2019.

[17] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, “Accurate, large minibatch sgd: Training imagenet in 1 hour,” *arXiv*, vol. abs/1706.02677, 2017.

[18] H. Guo, W. Guo, Y. Gao, R. Tang, X. He, and W. Liu, “ScaleFreeCTR: Mixcache-based distributed training system for CTR models with huge embedding table,” in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021.

[19] H. Guo, R. Tang, Y. Ye, Z. Li, X. He, and Z. Dong, “DeepFM: An end-to-end wide & deep learning framework for CTR prediction,” *arXiv*, vol. abs/1804.04950, 2018.

[20] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[21] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[22] X. He, F. Xue, X. Ren, and Y. You, “Large-scale deep learning optimizations: A comprehensive survey,” *arXiv*, vol. abs/2111.00856, 2021.

[23] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing, “More effective distributed ml via a stale synchronous parallel parameter server,” *Advances in neural information processing systems*, vol. 2013, pp. 1223–1231, 2013.

[24] E. Hoffer, I. Hubara, and D. Soudry, “Train longer, generalize better: Closing the generalization gap in large batch training of neural networks,” *arXiv*, vol. abs/1705.08741, 2017.

[25] B. Jiang, C. Deng, H. Yi, Z. Hu, G. Zhou, Y. Zheng, S. Huang, X. Guo, D. Wang, Y. Song, L. Zhao, Z. Wang, P. Sun, Y. Zhang, D. Zhang, J. Li, J. Xu, X. Zhu, and K. Gai, “XDL: An industrial deep learning framework for high-dimensional sparse data,” in *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, 2019.

[26] E. Kaasinen, V. Roto, K. Roloff, K. Väänänen-Vainio-Mattila, T. Vainio, W. Maehr, D. Joshi, and S. Shrestha, “User experience of mobile internet: Analysis and recommendations,” *International Journal of Mobile Human Computer Interaction (IJMHCI)*, vol. 1, no. 4, pp. 4–23, 2009.

[27] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, “On large-batch training for deep learning: Generalization gap and sharp minima,” *arXiv*, vol. abs/1609.04836, 2017.

[28] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv*, vol. abs/1412.6980, 2015.

[29] A. Krizhevsky, “One weird trick for parallelizing convolutional neural networks,” *arXiv*, vol. abs/1404.5997, 2014.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, pp. 84–90, 2012.

[31] S. Kumar, V. Bitorff, D. Chen, C.-H. Chou, B. A. Hechtman, H. Lee, N. Kumar, P. Mattson, S. Wang, T. Wang, Y. Xu, and Z. Zhou, “Scale MLPerf-0.6 models on google tpu-v3 pods,” *arXiv*, vol. abs/1909.09756, 2019.

[32] C. Labs, *Display advertising challenge*, 2014.

[33] M. Li, Z. Liu, A. Smola, and Y.-X. Wang, “DiFacto: Distributed factorization machines,” in *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, 2016.

[34] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, “Scaling distributed machine learning with the parameter server,” in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014, pp. 583–598.

[35] Z. Li, Z. Cui, S. Wu, X. Zhang, and L. Wang, “Fi-GNN: Modeling feature interactions via graph neural networks for CTR prediction,” 2019.

[36] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G.-z. Sun, “xDeepFM: Combining explicit and implicit feature interactions for recommender systems,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[37] X. Lian, B. Yuan, X. Zhu, Y. Wang, Y. He, H. Wu, L. Sun, H. Lyu, C. Liu, X. Dong, Y. Liao, M. Luo, C. Zhang, J. Xie, H. Li, L. Chen, R. Huang, J. Lin, C. Shu, X.-B. Qiu, Z. Liu, D. Kong, L. Yuan, H.-b. Yu, S. Yang, C. Zhang, and J. Liu, “Persia: An open, hybrid system scaling deep learning-based recommenders up to 100 trillion parameters,” *arXiv*, vol. abs/2111.05897, 2021.

[38] F. Lyu, X. Tang, H. Guo, R. Tang, X. He, R. Zhang, and X. Liu, “Memorize, factorize, or be naïve: Learning optimal feature interaction methods for CTR prediction,” *arXiv*, vol. abs/2108.01265, 2021.

[39] Y. Ma, B. Narayanaswamy, H. Lin, and H. Ding, “Temporal-contextual recommendation in real-time,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[40] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[41] P. Mattson, C. Cheng, C. A. Coleman, G. F. Diamos, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, D. M. Brooks, D. Chen, D. Dutta, U. Gupta, K. M. Hazelwood, A. Hock, X. Huang, B. Jia, D. Kang, D. Kanter, N. Kumar, J. Liao, G. Ma, D. Narayanan, T. Oguntebi, G. Pekhimenko, L. Pentecost, V. J. Reddi, T. Robie, T. S. John, C.-J. Wu, L. Xu, C. Young, and M. A. Zaharia, “MLPerf training benchmark,” *arXiv*, vol. abs/1910.01500, 2020.

[42] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica, “Ad click prediction: A view from the trenches,” in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013.

[43] X. Miao, H. Zhang, Y. Shi, X. Nie, Z. Yang, Y. Tao, and B. Cui, “Het: Scaling out huge embedding model training via cache-enabled distributed framework,” in *Proceedings of the VLDB Endowment*, vol. 15, 2021, pp. 312–320.

[44] D. Mudigere, Y. Hao, J. Huang, Z. Jia, A. Tulloch, S. Sridharan, X. Liu, M. Ozdal, J. Nie, J. Park, L. Luo, J. A. Yang, L. Gao, D. Ivchenko, A. Basant, Y. Hu, J. Yang, E. K. Ardestani, X. Wang, R. Komuravelli, C.-H. Chu, S. Yilmaz, H. Li, J. Qian, Z. Feng, Y.-A. Ma, J. Yang, E. Wen, H. Li, L. Yang, C. Sun, W. Zhao, D. Melts, K. Dhulipala, K. G. Kishore, T. Graf, A. Eisenman, K. K. Matam, A. Gangidi, G. J. Chen, M. Krishnan, A. Nayak, K. Nair, B. Muthiah, M. khorashadi, P. Bhattacharya, P. Lapukhov, M. Naumov, A. S. Mathews, L. Qiao, M. Smelyanskiy, B. Jia, and V. Rao, “Software-hardware co-design for fast and scalable training of deep learning recommendation models,” *arXiv*, vol. abs/2104.05158, 2021.

[45] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, “Deep learning recommendation model for personalization and recommendation systems,” *arXiv*, vol. abs/1906.00091, 2019.

[46] E. Oldridge, J. Perez, B. Frederickson, N. Koumchatzky, M. Lee, Z. Wang, L. Wu, F. Yu, R. Zamora, O. Yilmaz, A. M. Gunny, V. P. Nguyen, and S. Lee, “Merlin: A GPU accelerated recommendation framework,” in *1st International Workshop on Industrial Recommendation Systems (IRS)*, 2020.

[47] Y. Qu, H. Cai, K. Ren, W. Zhang, Y. Yu, Y. Wen, and J. Wang, “Product-based neural networks for user response prediction,” in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 2016, pp. 1149–1154.

[48] S. Rendle, “Factorization machines,” in *2010 IEEE International Conference on Data Mining (ICDM)*, 2010, pp. 995–1000.

[49] W. Shen, *Deepctr: Easy-to-use,modular and extendible package of deep-learning based CTR models*, `https://github.com/shenweichen/deepctr`, 2017.

[50] W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, and J. Tang, “AutoInt: Automatic feature interaction learning via self-attentive neural networks,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019.

[51] Soroush, *What’s the effect of scaling a loss function in deep learning?* Cross Validated, URL:https://stats.stackexchange.com/q/395443 (version: 2019-09-07). [Online]. Available: `https://stats.stackexchange.com/q/395443`.

[52] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[53] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Conference on Neural Information Processing Systems (NIPS)*, 2014.

[54] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & Cross network for ad click predictions," in *Proceedings of the ADKDD'17*, 2017.

[55] R. Wang, R. Shivanna, D. Z. Cheng, S. Jain, D. Lin, L. Hong, and E. H. Chi, "DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems," in *Proceedings of the Web Conference 2021*, 2021.

[56] X. Wang, "A survey of online advertising click-through rate prediction models," in *2020 IEEE International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*, vol. 1, 2020, pp. 516–521.

[57] Z. Wang, Q. She, and J. Zhang, "Masknet: Introducing feature-wise multiplication to CTR ranking models by instance-guided mask," *arXiv*, vol. abs/2102.07619, 2021.

[58] Z. Wang, Q. She, P. Zhang, and J. Zhang, "Correct normalization matters: Understanding the effect of normalization on deep neural network models for click-through rate prediction," *arXiv*, vol. abs/2006.12753, 2020.

[59] M. Xie, K. Ren, Y. Lu, G. Yang, Q. Xu, B. Wu, J. Lin, H. Ao, W. Xu, and J. Shu, "Kraken: Memory-efficient continual learning for large-scale real-time recommendations," in *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2020.

[60] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," *arXiv*, vol. abs/1708.03888, 2017.

[61] Y. You, J. Li, S. J. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large batch optimization for deep learning: Training bert in 76 minutes," *arXiv*, vol. abs/1904.00962, 2020.

[62] B. Zhang, L. Luo, X. Liu, J. Li, Z. Chen, W. Zhang, X. Wei, Y. Hao, M. Tsang, W.-J. Wang, Y. Liu, H. Li, Y. Badr, J. Park, J. Yang, D. Mudigere, and E. Wen, "DHEN: A deep and hierarchical ensemble network for large-scale click-through rate prediction," *arXiv*, vol. abs/2203.11014, 2022.

[63] J. Zhang, T. He, S. Sra, and A. Jadbabaie, "Why gradient clipping accelerates training: A theoretical justification for adaptivity," *arXiv*, vol. abs/1905.11881, 2020.

[64] J. Zhang, T. Huang, and Z. Zhang, "FAT-DeepFFM: Field attentive deep field-aware factorization machine," in *ICDM*, 2019.

[65] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.

[66] W. Zhang, T. Du, and J. Wang, "Deep learning over multi-field categorical data – a case study on user response prediction," in *European Conference on Information Retrieval (ECIR)*, 2016.

[67] W. Zhao, J. Zhang, D. Xie, Y. Qian, R. Jia, and P. Li, "AIBox: CTR prediction model training on a single node," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019.

[68] W. Zhao, D. Xie, R. Jia, Y. Qian, R. Ding, M. Sun, and P. Li, "Distributed hierarchical GPU parameter server for massive scale deep learning ads systems," *arXiv*, vol. abs/2003.05622, 2020.

[69] G. Zhou, N. Mou, Y. Fan, Q. Pi, W. Bian, C. Zhou, X. Zhu, and K. Gai, "Deep interest evolution network for click-through rate prediction," *arXiv*, vol. abs/1809.03672, 2019.

[70] J. Zhu, J. Liu, S. Yang, Q. Zhang, and X. He, "Open benchmarking for click-through rate prediction," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021.

Appendix

A Wide/Cross stream of CTR prediction models

Deep learning has been widely used by the community to boost CTR prediction performance. CTR prediction networks [7, 9, 19, 35, 36, 38, 47, 50, 54, 55, 57, 58, 64–66] (Figure 2 left) have outperformed traditional methods such as Logistic Regression (LR) [42] and Factorization Machine (FM) [48]. A thread of work started from [7, 54] focused on designing a two-stream network. After the W&D model [7], there are many designs on the wide or cross-stream. After one-hot encoding of every categorical field, the input is embedded into a dense vector in the network. The wide or cross-stream serves to model the feature interactions explicitly. For instance, the LR model is a first-order predictor, FM models the second-order interactions, and n cross layers [54] models n-th-order interactions. The first two methods are called "wide" for only one layer is used. The other stream is a feed-forward neural network to compensate for the ability to learn higher-order interactions. The details of the four models in our experiments are as follows.

For the W&D model, the wide part is logistic regression. Denote the output of the network as $\hat{y}$, the predicted probability of clicking is obtained by $\texttt{sigmoid}(\hat{y})$. After one-hot encoding of input x, the vector is a n-dimensional $\boldsymbol{x}$. The logistic regression considers each feature independently and lets the DNN captures high-order relations in the data.

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i \boldsymbol{x}_i.$$

The DeepFM model [19] adopts a factorization machine for the wide part, which is also the cross-stream for it explicitly models the second-order relationship between different features. After embedding each column into a d-dimensional vector $\boldsymbol{v}$, the factorization machine models $\hat{y}$ as follows:

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i \boldsymbol{x}_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle \boldsymbol{x}_i \boldsymbol{x}_j.$$

For DCN [54], it introduces the cross-layer to automatically learns a high-order interaction. With L cross layers, it can model interactions from 2 to $L+1$ order. Denote the $\boldsymbol{x}_0$ as input vector and $\boldsymbol{x}_\ell$ as the output of the ℓ-th layer. The ℓ-th cross-layer does as follows:

$$\boldsymbol{x}_{\ell+1} = \boldsymbol{x}_0 \boldsymbol{x}_\ell^\top \boldsymbol{w}_\ell + \boldsymbol{b}_\ell + \boldsymbol{x}_\ell.$$

The DCN-v2 [55] proposed a new cross-layer to model high-order interaction. Specifically, with $\boldsymbol{W}_\ell \in \mathbb{R}^{d \times d}$, the ℓ-th layer is:

$$\boldsymbol{x}_{\ell+1} = \boldsymbol{x}_0 \odot (\boldsymbol{W}_\ell \boldsymbol{x}_\ell + \boldsymbol{b}_\ell) + \boldsymbol{x}_\ell.$$

B Properties of CTR Prediction Input

The input of the CTR prediction model is high-dimensional, sparse, and frequent-unbalanced. As we have discussed the frequency in the main paper, we focus on the fact that the input feature space for CTR prediction is high-dimensional and sparse, which is an essential difference between the CTR prediction model and other deep learning models.

Table 7: Number of parameters for different layers in our experimental settings.

	Network				Embedding (Dataset)	
Name	W&D	DeepFM	DCN	DCNv2	Criteo	Avazu
#Params	0.431M	0.431M	0.433M	0.655M	372M	104M

A typical industrial CTR prediction model [59, 67, 69] has a high-dimensional input space with 10^8 to 10^{12} dimensions after one-hot encoding of the categorical features. At the same time, a single clicking log may contain only hundreds of non-zero entries. As a result, when we create the embedding for each feature, the whole embedding layer can be extremely large, and the parameters of the CTR prediction model are dominated (e.g., 99.9% [14, 43]) by the embedding part instead of the deep network part. In Table 7, the number of parameters in the embedding layer also overwhelms one of the dense networks.

C Related Work on Frequency in CTR prediction

Many works have shed light on the importance of different frequencies in embeddings to accelerate or improve the accuracy of CTR prediction models. [2, 43] finds that most access to the embedding layer happens in a small fraction of embeddings. They both propose to cache the most frequent embeddings to reduce the communication bottleneck. [14] proposed a mixed dimension embedding where the embedding vector's dimension scales with its query frequency to reduce the memory cost. To improve the performance of the CTR prediction models, [69] shows a regularization by filtering out ids with a low occurrence frequency. [33] proposes to apply stronger regularization on more frequent ids for better generalization performance. In this paper, we show that the different frequencies of ids result in the failure of previous scaling rules when scaling up the batch size.

D Traditional Scaling Rules Derivation

The deduction of scaling rules is mostly based on the SGD optimizer. However, many experimental results [22, 24, 61] show they are also effective when applied to adaptive optimizers such as Adam. We consider the case batch size is scaled from B to B', where $b' = |B'| = s|B| = sb$. For a given big batch B'_t, the corresponding s small batches are $\{B_{t,i}\}_{i=1}^{s}$.

Linear Scaling For Linear Scaling, when scaling up s times the batch size, the motivation is to keep the update by a big batch equivalent to the update made within s small batches. Denote the update made by SGD update as $\Delta w = w_t - w_{t+1}$, for the small batches. We have the expected update:

$$E[\Delta w] = \eta \sum_{i=1}^{s} \mathbb{E}[\frac{1}{b} \cdot \sum_{x \in B_{t,i}} \nabla L(w_{t,i}, x)] = \eta \sum_{i=1}^{s} \mathbb{E}[\nabla L(w_{t,i}, x)],$$

while for the big batch:

$$\mathbb{E}[\Delta w] = \eta' \, \mathbb{E}[\frac{1}{b'} \cdot \sum_{x \in B'_t} \nabla L(w_t, x)] = \eta' \, \mathbb{E}[\nabla L(w_t, x)]. \tag{3}$$

Under the assumption that $\mathbb{E}[\nabla L(w_{t,i}, x)] \approx \mathbb{E}[\nabla L(w_t, x)]$ for $i = 1$ to s, to make the two update equal, we need to scale the learning rate $\eta' \to s\eta$.

Sqrt Scaling For Sqrt Scaling, its motivation is to keep the covariance matrix of the parameters updates Δw the same. The derivation comes from ,[24] and we only consider the case that samples are randomly drawn from a dataset with replacement. According to [24], with minibatch gradient denoted as $\hat{\boldsymbol{g}} = \frac{1}{b} \sum_{x \in B} \boldsymbol{g}_x$, we have

$$\mathtt{cov}(\hat{\boldsymbol{g}}, \hat{\boldsymbol{g}}) = (\frac{1}{b} - \frac{1}{N}) \frac{1}{N} \sum_{i=1}^{N} \hat{\boldsymbol{g}}_i \hat{\boldsymbol{g}}_i^{\top}.$$

Since $\frac{1}{N}$ is small, the update of SGD has covariance:

$$\mathtt{cov}(\Delta w, \Delta w) = \mathtt{cov}(\eta \hat{\boldsymbol{g}}, \eta \hat{\boldsymbol{g}}) \approx \frac{\eta^2}{b \cdot N} \sum_{i=1}^{N} \hat{\boldsymbol{g}}_i \hat{\boldsymbol{g}}_i^{\top}. \tag{4}$$

When scaling $b \to sb$, to keep the covariance matrix to be the same, we need to scale $\eta \to \sqrt{s}\eta$.

Scaling the L2-regularization weight We follow the discussion in [29] to scale the L2-regularization weight with batch size. Consider the case only L2-regularization is applied, under a large batch we have:

$$w_{t+1} = w_t(1 - \eta'\lambda' w),$$

while for small batches:

$$w_{t+1} = w_t(1 - \eta\lambda w)^s = w_t(1 - s\eta\lambda w).$$

To make the L2-regularization strength at the same level, when scaling the batch size, we have the equation:

$$\eta'\lambda' \approx s\eta\lambda.$$

Therefore, considering the learning rate scaling in linear scaling and sqrt scaling, it is easy to calculate the scaling strategy for the λ.

E Sqrt Scaling Motivation with Different Frequencies

In Section 3.1, we have discussed the scaling rules for columns with different frequencies. Here, we consider the sqrt scaling rule. The derivation is inspired by [24]. Similar to Equation 2, we have

$$\boldsymbol{g} = \frac{1}{b}\delta(\mathrm{id}_k^{f_j} \in B)\sum_{x \in B} \nabla L(w, x) = \delta(\mathrm{id}_k^{f_j} \in B)\hat{\boldsymbol{g}}.$$

With Equation 4, since mini-batches are uncorrelated, the covariance is

$$\begin{aligned}
\mathtt{cov}(\Delta\boldsymbol{g}, \Delta\boldsymbol{g}) &= \mathbb{E}[\boldsymbol{g}\boldsymbol{g}^\top] - \mathbb{E}[\boldsymbol{g}]\,\mathbb{E}[\boldsymbol{g}^\top] \\
&= \frac{1}{b^2}\sum_{i=1}^{b}\sum_{j=1}^{b}\mathbb{E}[\delta(\mathrm{id}_k^{f_j} \in B)\delta(\mathrm{id}_k^{f_j} \in B')]\hat{\boldsymbol{g}}_i\hat{\boldsymbol{g}}_j - \mathrm{P}(\mathrm{id}_k^{f_j} \in B)^2\,\mathbb{E}[\hat{\boldsymbol{g}}]\,\mathbb{E}[\hat{\boldsymbol{g}}^\top] \\
&= \mathrm{P}(\mathrm{id}_k^{f_j} \in B)\,\mathbb{E}[\hat{\boldsymbol{g}}\hat{\boldsymbol{g}}^\top] - \mathrm{P}(\mathrm{id}_k^{f_j} \in B)^2\,\mathbb{E}[\hat{\boldsymbol{g}}]\,\mathbb{E}[\hat{\boldsymbol{g}}^\top].
\end{aligned}$$

As we can see, the frequency differently scales the two parts in covariance. It is impossible to correct to the original behavior by multiplying a scaler to the learning rate. Only under a strong assumption that $\mathbb{E}[\hat{\boldsymbol{g}}\hat{\boldsymbol{g}}^\top] \ll \mathbb{E}[\hat{\boldsymbol{g}}]\,\mathbb{E}[\hat{\boldsymbol{g}}^\top]$ can we make the following approximation:

$$\mathtt{cov}(\Delta\boldsymbol{g}, \Delta\boldsymbol{g}) \approx \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B)\mathtt{cov}(\Delta\hat{\boldsymbol{g}}, \Delta\hat{\boldsymbol{g}}).$$

The assumption may not hold in practice, which adds to the problem's difficulty. Under the hypothesis, we have the covariance matrix for the Δw is:

$$\mathtt{cov}(\Delta w, \Delta w) = \mathtt{cov}(\eta\hat{\boldsymbol{g}}, \eta\hat{\boldsymbol{g}}) \approx \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B) \cdot \frac{\eta^2}{b \cdot N}\sum_{i=1}^{N}\hat{\boldsymbol{g}}_i\hat{\boldsymbol{g}}_i^\top.$$

For dense weight and frequent ids, with $\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B) \approx 1$, it is the same to the sqrt scaling. However, for unfrequent ids, the covariance matrix becomes:

$$\mathtt{cov}(\Delta w, \Delta w) = \mathtt{cov}(\eta\hat{\boldsymbol{g}}, \eta\hat{\boldsymbol{g}}) \approx \frac{\eta^2}{\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in \boldsymbol{x}) \cdot N}\sum_{i=1}^{N}\hat{\boldsymbol{g}}_i\hat{\boldsymbol{g}}_i^\top.$$

Hence, there is no need to scale the learning rate to keep the same covariance matrix.

F Effect of Loss Scaling

In training the network, the gradient of weights w on one sample x concerning the training loss L is $\nabla_w L(w, x)$. When training with batch size b, we take the average of gradients on the samples and thus have the expected gradients as follows:

$$\mathbb{E}[G] = \mathbb{E}[\frac{1}{b}\sum_{i=1}^{b}\nabla_w L(w, x_i)] = \mathbb{E}[\nabla_w L(w, x)].$$

Table 8: hyperparameters for square root scaling, linear scaling and empirical scaling.

	Sqrt Scaling		**Linear Scaling**		**Empirical Scaling**		
Batch Size	LR	L2	LR	L2	LR (Embed)	L2	LR (Dense)
1K (1024)	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}
2K (2048)	$\sqrt{2} \times 10^{-4}$	$\sqrt{2} \times 10^{-4}$	2×10^{-4}	1×10^{-4}	1×10^{-4}	4×10^{-4}	2×10^{-4}
4K (4096)	2×10^{-4}	2×10^{-4}	4×10^{-4}	1×10^{-4}	1×10^{-4}	1.6×10^{-3}	4×10^{-4}
8K (8192)	$2\sqrt{2} \times 10^{-4}$	$2\sqrt{2} \times 10^{-4}$	8×10^{-4}	1×10^{-4}	1×10^{-4}	$\underline{1.28 \times 10^{-2}}$	8×10^{-4}

Table 9: hyperparameters for CowClip scaling on Criteo and Avazu dataset.

	Criteo				**Avazu**			
Batch Size	LR (Embed)	L2	LR (Dense)	(r, ζ)	LR (Embed)	L2	LR (Dense)	(r, ζ)
1K (1024)	1×10^{-4}	1×10^{-4}	8×10^{-4}	$(1, 10^{-5})$	1×10^{-4}	1×10^{-4}	1×10^{-4}	$(10, 10^{-3})$
2K (2048)	1×10^{-4}	2×10^{-4}	$8\sqrt{2} \times 10^{-2}$	$(1, 10^{-5})$	1×10^{-4}	2×10^{-4}	$\sqrt{2} \times 10^{-4}$	$(10, 10^{-3})$
4K (4096)	1×10^{-4}	4×10^{-3}	16×10^{-4}	$(1, 10^{-5})$	1×10^{-4}	4×10^{-3}	2×10^{-4}	$(1, 10^{-4})$
8K (8192)	1×10^{-4}	8×10^{-4}	$16\sqrt{2} \times 10^{-4}$	$(1, 10^{-5})$	1×10^{-4}	8×10^{-4}	$2\sqrt{2} \times 10^{-4}$	$(1, 10^{-4})$
16K (16384)	1×10^{-4}	1.6×10^{-3}	32×10^{-4}	$(1, 10^{-5})$	1×10^{-4}	1.6×10^{-3}	4×10^{-4}	$(1, 10^{-4})$
32K (32768)	1×10^{-4}	3.2×10^{-3}	$32\sqrt{2} \times 10^{-4}$	$(1, 10^{-5})$	1×10^{-4}	3.2×10^{-3}	$4\sqrt{2} \times 10^{-4}$	$(1, 10^{-4})$
64K (65536)	1×10^{-4}	6.4×10^{-3}	64×10^{-4}	$(1, 10^{-5})$	1×10^{-4}	6.4×10^{-3}	8×10^{-4}	$(1, 10^{-4})$
128K (131072)	1×10^{-4}	1.28×10^{-2}	$64\sqrt{2} \times 10^{-4}$	$(1, 10^{-5})$	1×10^{-4}	$\underline{9.6 \times 10^{-3}}$	$\underline{16 \times 10^{-4}}$	$(1, 10^{-4})$

As discussed in Equation 2, in training with weights of different frequencies, there is a case a constant multiplier is applied to the expected gradients. We consider the case gradients are scaled by c and show different behaviors with SGD and Adam with L2-regularization following discussion in [51]:

$$\mathbb{E}[G] = c \cdot \mathbb{E}[\nabla_w L(w, x)].$$

First, for SGD optimizer with learning rate η and L2-regularization weight λ, the expected update is:

$$\mathbb{E}[\Delta w] = \mathbb{E}[w_t - w_{t+1}] = \eta \cdot (\mathbb{E}[\nabla_{w_t} L(w_t, x)] - \frac{\lambda}{2} w_t).$$

When the gradients are scaled by c, we have

$$\mathbb{E}[\Delta w] = \eta \cdot (c \cdot \mathbb{E}[\nabla_{w_t} L(w_t, x)] - \frac{\lambda}{2} w_t) = c\eta \cdot (\cdot \mathbb{E}[\nabla_{w_t} L(w_t, x)] - \frac{\lambda}{2c} w_t).$$

So the effect is using a new learning rate of $c\eta$ and new L2-regularization weight of $\frac{\lambda}{c}$.

For Adam optimizer with (β_1, β_2) without L2-regularization, the expected update is:

$$\mathbb{E}[\Delta w] = \eta \, \mathbb{E}[\frac{m_t}{\sqrt{v_t + \epsilon}}]. \tag{5}$$

If we omit the ϵ term and bias correction for simplicity, when the gradients are scaled by c, the momentum term m_t has:

$$\mathbb{E}[m_t] = \beta_1 \, \mathbb{E}[m_t] + (1 - \beta_1) \cdot c \cdot \mathbb{E}[\nabla_{w_t} L(w_t, x)] = c \cdot \sum_{i=1}^{t} (1 - \beta_1) \beta_1^{i-1} \, \mathbb{E}[\nabla_{w_i} L(w_i, x)].$$

which is c times the original $\mathbb{E}[m_t]$. The similar deduction finds $v_t \to c^2 v_t$. Thus,

$$\mathbb{E}[\Delta w] = \frac{c \cdot m_t}{\sqrt{c^2 \cdot v_t + \epsilon}} \approx \frac{m_t}{\sqrt{v_t + \epsilon}},$$

so the behaviour of Adam without regularization is not changed. However, with L2-regularization, we have:

$$\mathbb{E}[m_t] = c \cdot \sum_{i=1}^{t} (1 - \beta_1) \beta_1^{i-1} (\mathbb{E}[\nabla_{w_i} L(w_i, x)] + \frac{\lambda}{2c} w_t),$$

and the same deduction applies to v^2. This shows that when the gradients are scaled by c, it is equivalent to using a new L2-regularization weight of $\frac{\lambda}{c}$.

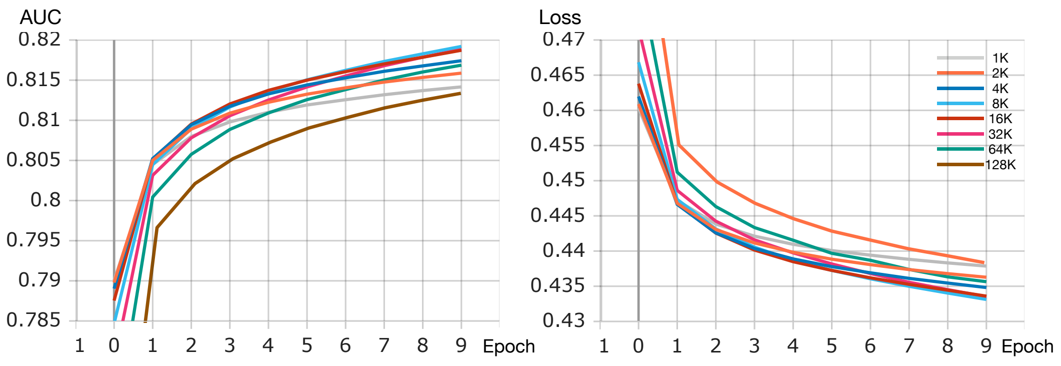


Figure 5: Training AUC (left) and Loss (right) at different epochs with different batch sizes during the training.

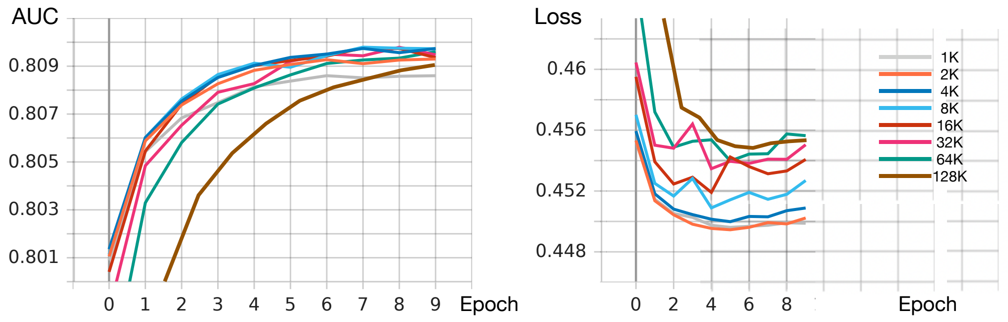


Figure 6: Test AUC (left) and Loss (right) at different epochs with different batch sizes during the training.

G Additional Implementation Details

In this paper, "K" means $\times 1024$, so 1K means 1024 batch size. The detailed learning rate, L2-regularization weights, and other hyperparameters are listed in Table 8 and Table 9, square roots are round to four decimal places in practice. The hyperparameter r is not sensitive, so we set it directly to 1, and the choice of ζ is related to the initialization weight in the next paragraph. LR denotes the learning rate η, and L2 denotes the L2-regularization weight λ (no L2-regularization is imposed on dense weights). All activation functions are ReLU [3], and dropout [52] is not used as we do not see its improvement. For DCN and DCNv2, the number of cross-layer is 3, and we only adopt the cross-layer form from the DCNv2 paper. The CowClip method is performed on id vector embeddings (columns), but is not applied to LR method for DeepFM and W&D, whose biases can be viewed as a 1-dimension embedding. For the continuous field, they do not involve in the wide or cross stream and are directly sent into the DNN stream.

One technique to train with CowClip is that as the learning process of embedding becomes more smooth and stable, we can fix the learning rate for the embeddings and scale up the learning rate for the dense layer until the training process diverges for better performance. As the batch size grows beyond a threshold, the proposed scaling rule may face accuracy loss (8K for empirical scaling rules and 128K for the CowClip scaling rule in the Avazu dataset). In that case, we do a little hyperparameter fine-tuning by scaling some of the hyperparameters to twice or half of their supposed value. Basically, we increase the L2 λ when the network is overfitting and increase ζ when the network is underfitting. These values are underlined in the table.

For large batch training with CowClip, there are two additional techniques. Warmup on learning rate has been widely used in training CV and NLP networks [16]. It helps the network start smoothly for a more stable training process. We find warmup on the learning rate of the embeddings has little improvement, so we only apply one-epoch learning rate warmup to the dense weights of the CTR prediction model. Weight initialization is also important for a good starting state. We use Kaiming initialization [21] for all dense weights. The original initialization for embeddings is $w \sim \mathcal{N}(0, \sigma), \sigma = 10^{-4}$. With a dimension of d, the initial weight norm is $\sqrt{d} \cdot \sigma$. To allow for a

Table 10: Performance of different scaling methods on Avazu dataset from 1K to 8K on DeepFM.

	1K (1024)		2K (2048)		4K (4096)		8K (8192)	
	AUC (%)	LogLoss	AUC (%)	LogLoss	AUC (%)	LogLoss	AUC (%)	LogLoss
No Scaling	78.84	0.3748	78.79	0.3775	77.69	0.3952	75.85	0.4411
Sqrt Scaling	78.84	0.3748	78.88	0.3761	77.78	0.3926	76.23	0.4299
Sqrt Scaling*	78.84	0.3748	78.88	0.3759	77.98	0.3976	76.23	0.4140
LR Scaling	78.84	0.3748	78.78	0.3763	77.72	0.3883	76.69	0.4043
n^2–λ Scaling (Ours)	78.84	0.3748	78.84	0.3754	78.26	0.3815	77.24	0.3912
CowClip (Ours)	78.83	0.3748	78.82	0.3752	78.90	0.3752	79.06	0.3740

Table 11: The training time of different methods on Avazu dataset. . Last four are trained with CowClip.

			Time (minutes)							
	AUC (%)	Logloss	1K	2K	4K	8K	16K	32K	64K	128K
XDL [2]	75.8	0.390	108	84	74	–	–	–	–	–
FAE [2]	77.8	0.391	72	62	61	–	–	–	–	–
DLRM [45]	76.6	0.387	163	141	54	–	–	–	–	–
Hotline [1]	76.8	0.386	70	28	24	–	–	–	–	–
DeepFM	78.84	0.3748	210	108	54	30	17	10	6.7	4.8
W&D	78.80	0.3750	210	108	54	30	17	10	6.7	5.0
DCN	78.82	0.3749	210	108	54	30	18	11	7.2	5.7
DCN v2	78.84	0.3748	234	126	66	37	25	19	18.5	19.5
Speedup (DeepFM)			1x	1.94x	3.89x	7.00x	12.3x	21x	31.3x	43.7x

† Trained with 2 GPUs ‡ Trained with 4 GPUs.

greater gradient norm bound in CowClip, we use a larger initial weight by setting the σ to 10^{-2} for training with CowClip. To avoid a too strong clipping value, we lower bound the clipping value with 10^{-4}, which is the original initial weight norm, and in Criteo 10^{-5}, which yields a better result.

H Additional Experimental Results

We run our experiments with three random seeds (1234, 1235, 1236), and the deviation for AUC is less than 0.012%. The training time comparison and speedup for the Avazu dataset are shown in Table 11. The training and testing AUC and loss curve at different epochs with different batch sizes are shown in Figure 5 and Figure 6 respectively.

I More Ablation Study

More ablation study is shown in Table 12. The first two rows verify the effectiveness of our scaling rule. Warmup on the dense weights is critical when the batch size is very large, while large initialization weights prevent the network from underfitting when the batch size is not that large.

We decide the hyperparameters for variants of gradient clipping in Table 6 as follows, which may be helpful if a simple version of gradient clipping is adopted for a complex system. First, we run the experiment and log out the gradients of interested units (i.e., global, field, column). For gradient clipping, the upper bound is 25, and after searching in $\{25, 20, 10, 1\}$, we find the performance is not sensitive to the clipping value. For field-wise and column-wise, we search in $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$ and $\{10^{-5}, 10^{-4}, 10^{-3}\}$ respectively.

For the gradient clipping with constant value `clip_t` for the global embedding or a field, note that when scaling the batch size, we also need to scale this value. Take the field-wise gradient clipping for example. Consider scaling the batch size from b to $s \cdot b$. The scaling rule for the embedding layers is as follows. First, consider the case all ids are frequent, then doubling the batch size doubles the occurrence of these ids in the batch. Thus, the gradients are also doubled. This indicates a linear scaling on the gradient clipping value.

Table 12: More Ablation study of CowClip on Criteo with DeepFM.

	b = 8K		b = 128K	
	AUC (%)	LogLoss	AUC (%)	LogLoss
CowClip w./ Linear Scale on Dense	diverge	diverge	diverge	diverge
CowClip w./ Empirical Scale	80.85	0.4430	79.83	0.4539
CowClip w.o. ζ	80.96	0.4426	80.88	0.4438
CowClip w.o. warmup	80.97	0.4422	80.52	0.4463
CowClip w.o. large init weight	80.92	0.4432	80.90	0.4431
CowClip	80.97	0.4425	80.90	0.4430

However, if all the ids are unfrequent, considering the process of merging s small batches into a big batch. For a specific id, the probability that it occurs in two of s batches is small. Thus, with the assumption that no colliding ids occur in the s small batches, we have the gradient g' for the large batch B' (g for the small batch B):

$$\boldsymbol{g'} = \sqrt{\sum_{i=1}^{s} \boldsymbol{g}_s} = \sqrt{s}\boldsymbol{g}.$$

This indicates we should use a square root scaling for the gradient clipping value. In practice, although both frequent and unfrequent ids exist, we find the sparse one dominates the gradients. We find that the scaling in the norm of gradients is approximately $\sqrt{s}$ when scaling the batch size and suggest square root scaling on gradient clipping value on the embedding layer is a better choice.

J Discussion and Future Work

With CowClip, we can scale the batch size of the CTR prediction model to 128 times larger than the original size on one GPU. Despite the great power of our method, there are more works to be done for large batch CTR prediction training, which we leave as future works.

First, as mentioned in the introduction, many works have been devoted to designing a sound system for a multi-node CTR prediction model. Due to the computational resource, we verified our method on a single GPU setting. Although it seems straightforward to integrate our approach into a multi-GPU training setting, system optimization is still needed for fast distributed training. It is also interesting to know how much can our method accelerate the training in a communication and memory-efficient multi-node CTR prediction system.

Second, when scaling to a very large batch size (e.g., 256K or even larger), the AUC still drops even with CowClip. One possible reason for this is that as the batch size grows, the assumption most ids are unfrequent may not be held. One possible way to deal with the problem is to design an id-wise scaling strategy, which may not be computational-efficient. Another possibility lies in the loss of generalization ability of models trained at a large batch, as found and discussed in CV and NLP areas. Since our experiments are conducted on only one GPU, a larger batch size is needed when scaling to a multi-GPU setting.

In addition, our experiments use the setting of adam optimizer and L2-regularization on all weights. In this setting, every weight and embeddings, along with their optimization states in the optimizer, are updated in each step. In some modern CTR prediction systems, a 'lazy' optimizer (e.g., adagrad [11], lazy-adam) updates the state of embeddings only when the corresponding id appears in the batch, and an L2-regularization only imposed on these ids is used for fast computation. In addition, sparse representation of embedding matrix also makes a difference to the optimization process (e.g., sparse optimizer update of tensor in Tensorflow). The clipping strategy should be modified to suit these methods. Combining our method with these variants of optimization strategies is also an interesting problem for practical deployment.

Our experiments follow previous work and train the network with 10 epochs. In reality, when the training dataset is huge, it is unaffordable or too slow to do multi-epoch training. In this case, one-epoch training is the choice to train an update-to-date CTR prediction model. As shown in the training curve in Figure 6, although we achieve better results when finishing the training, the AUC at the first epoch drops compared to the small-batch setting. We also find that to maintain the first epoch

AUC value, a much smaller L2-regularization should be adopted. The reason may be that in the first training epoch, overfitting is not likely to occur and thus requires a weaker regularization. We believe an investigation into one-epoch large batch training will be valuable work.

Apart from CTR prediction, there are other tasks with a large embedding table, such as NLP tasks. For instance, in the Chinese embedding table, an unbalanced-frequency exists among different characters. Even if the frequencies of different ids are not as varied as those in CTR prediction, a simplified version of CowClip (e.g., remove the occurrence count) may help to stabilize the training of models in these tasks.

With the growth of hardware and modern CTR prediction systems, we believe a trend is to adopt a larger and larger batch size for fast CTR prediction model training. To use a large batch size, apart from a robust system, a suitable algorithm is also needed to maintain the performance. As there are many works on large batch training in CV and NLP, few works discuss the problem of large batch training in the CTR prediction model. We think it is worthwhile to investigate this problem.