Improving Training Stability for Multitask Ranking Models in Recommender Systems

Jiaxi Tang*
Google Deepmind
Mountain View, California, USA
jiaxit@google.com

Yoel Drori*
Google Research
Tel Aviv, Israel
dyoel@google.com

Daryl Chang*
Google Inc
Mountain View, California, USA
dlchang@google.com

Maheswaran Sathiamoorthy
Google Deepmind
Mountain View, California, USA
nlogn@google.com

Justin Gilmer
Google Deepmind
Mountain View, California, USA
gilmer@google.com

Li Wei
Google Inc
Mountain View, California, USA
liwei@google.com

Xinyang Yi
Google Deepmind
Mountain View, California, USA
xinyang@google.com

Lichan Hong
Google Deepmind
Mountain View, California, USA
lichan@google.com

Ed H. Chi
Google Deepmind
Mountain View, California, USA
edchi@google.com

ABSTRACT

Recommender systems play an important role in many content platforms. While most recommendation research is dedicated to designing better models to improve user experience, we found that research on stabilizing the training for such models is severely under-explored. As recommendation models become larger and more sophisticated, they are more susceptible to training instability issues, *i.e.*, loss divergence, which can make the model unusable, waste significant resources and block model developments. In this paper, we share our findings and best practices we learned for improving the training stability of a real-world multitask ranking model for YouTube recommendations. We show some properties of the model that lead to unstable training and conjecture on the causes. Furthermore, based on our observations of training dynamics near the point of training instability, we hypothesize why existing solutions would fail, and propose a new algorithm to mitigate the limitations of existing solutions. Our experiments on YouTube production dataset show the proposed algorithm can significantly improve training stability while not compromising convergence, comparing with several commonly used baseline methods. We open source our implementation at https://github.com/tensorflow/recommenders/tree/main/tensorflow_recommenders/experimental/optimizers/clippy_adagrad.py.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Retrieval models and ranking**.

KEYWORDS

Recommender System; Optimization; Training Stability

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08.
https://doi.org/10.1145/3580305.3599846

ACM Reference Format:
Jiaxi Tang*, Yoel Drori*, Daryl Chang*, Maheswaran Sathiamoorthy, Justin Gilmer, Li Wei, Xinyang Yi, Lichan Hong, and Ed H. Chi. 2023. Improving Training Stability for Multitask Ranking Models in Recommender Systems. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599846

1 INTRODUCTION

A good recommender system plays a key factor to user experience. It has become a core technology and even a main user interface in many web applications, including YouTube, one of the largest online video platforms in the world. As a result, many components can be incorporated into recommendation models to capture contexts with different modalities and improve recommendation quality, including audio signals [30], video signals [21], user history sequence [5, 28], etc. Besides, the scaling law of recommendation models [3] suggests substantial quality improvements by increasing model capacity in data-rich applications.

As recommendation models become larger and more sophisticated, they are more susceptible to training instability issues [14], *i.e.*, the loss diverges (instead of converging), causing the model to be "broken" and completely useless. In industry, serving such a "broken" model leads to catastrophic user experience (see Section 2.2). Moreover, if we cannot ensure reliable training of recommendation models, a huge amount of resources can be wasted and model development can be blocked. Therefore, we couldn't emphasize more on how essential training stability is. However, very sparse research has been done on the training stability of recommendation models.

On one hand, there's a lack of fundamental understanding of why recommendation models are prone to training instability issues. In particular, we observe that ranking models with multiple objectives are more likely to encounter problems than retrieval models with a single objective (*e.g.* Softmax Cross-Entropy over large output space). In addition to increasing model complexity, we found that simply adding new input features or output tasks can also cause

*Equal contribution to the work.

training unstable. To deal with this problem, people mostly rely on empirical solutions and sometimes on luck (when the problem occurs randomly). Developing a fundamental understanding of what causes the problem would allow people to navigate the process more confidently.

On the other hand, we found that there's a lack of effective approaches to largely mitigate the training instability problem. There are some widely used methods, such as activation clipping [20], gradient clipping [7, 24], learning rate warmup [12, 14], and layer normalization [4]. But in practice, we found that these approaches were ad hoc and couldn't completely prevent training instability in our model. Developing an effective method that can significantly improve model training stability accelerates model improvements by addressing concerns about training problems.

The focus of this paper is to share the lessons learned from addressing the training instability problems experienced by a multi-task ranking model used in production for YouTube recommendations. In Section 2, we show some implications and consequences of unstable model training in real-world recommender systems, suggesting the importance and difficulties of considerably improving model training stability. After introducing some preliminary basics of our model in Section 3, we present some case studies about changes that had led to more training instability problems and provide our understanding on the root cause of the problems. In practice, however, we've found that there's a big gap between knowing the root cause and having an effective solution. Some methods that are supposed to be effective do not work well empirically. Next, in Section 4, we closely examine the training dynamics of our model, which inspired us to propose a more effective approach to overcome the limitations in existing methods. The empirical evidence on a YouTube dataset in Section 5 reveals the effectiveness of the proposed method for improving model training stability, especially when increasing model capacity and using a large learning rate for faster convergence. We hope that these findings can help the community better understand the training instability problem and effectively solve it.

2 BACKGROUND AND RELATED WORK

2.1 Symptoms

Training instability is a model property that measures the unsteadiness of model training. It has a common symptom of *loss divergence* (*a.k.a* loss blow-up). Based on our observations, we further categorize loss divergence into two types: *micro-divergence* and *full divergence*. When a model's loss micro-diverges (see *model-a* in Figure 1 as an example), we can observe a sudden jump in training loss and a sudden drop in training metrics, although the loss may recover to normal (as shown in the example) as training continues. Usually, we don't need to worry too much about this situation, because the recovered model can have a quality on-par with models that don't suffer from loss divergence. However, if a model's loss fully diverges (see *model-b* in Figure 1 as an example), we can see that the training loss becomes very high in a few number of training steps, and all training metrics become extremely bad. For example, the binary classification AUC (the metric we mainly look throughout the paper) drops to 0.5 as shown in Figure 1, suggesting the model becomes completely useless, practically giving random

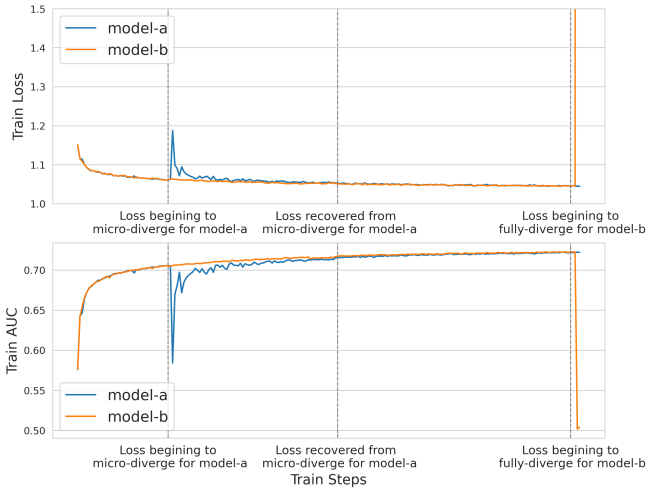


Figure 1: Example of loss divergence in our model and its impact on training loss (top) and AUC (bottom). In this example, model-a's loss micro-diverged then recovered, whereas model-b's loss fully-diverged.

results. What's worse, the fully diverged loss cannot recover to its pre-divergence value as training continues.

2.2 Motivation and Challenges

We motivate the importance of training stability research, especially for recommender systems in industry, from several aspects. First, the problem of loss divergence, once it occurs regularly, can affect almost all types of model development. This includes, but is not limited to:

(1) **Increasing model complexity:** As more modeling techniques are applied and more components are added to the recommendation model (to improve its quality), there's a greater chance that the model will suffer from loss divergence problems. Even simply enlarging the model capacity could put the model in a dangerous state, despite the great benefits in data-rich environments suggested by current scaling laws [3].

(2) **Adding more input features or tasks:** Typically, the ranking model in a recommendation system uses many input features for multiple tasks [36]. A combination of predictions on all tasks is used to decide the ranking of a candidate item. We found that both adding new input features and adding new tasks can lead to training instability, although they are common ways to improve model quality.

(3) **Increasing convergence speed:** We have found that hyperparameter tuning that facilitate model convergence (such as increasing the learning rate) can significantly increase the likelihood of loss divergence. This forces model designers to use a smaller learning rate which results in slower convergence.

Second, as training complex models requires large amounts of resources, loss divergence problems, which block the model from

completing their training, waste training resources. Moreover, inadvertently deploying a "broken" model for serving also leads to catastrophic user experience.

Consequently, we've seen many efforts on alleviating this problem from an engineering perspective, such as ensuring model quality before serving. Nevertheless, given that engineering efforts cannot prevent training instability from occurring, it is clear that drastically improving model training stability is the right path to pursue in the long run.

In dealing with the problem of model instability, we experienced the following challenges.

- **Reproducibility**: A model can suffer from loss divergence at any time during training, however, only some of them can be easily reproduced (more discussions in Section 3). Failing to reproduce bad cases makes it hard to understand what happened to the model before loss divergence.
- **Detection:** In practice, it is costly to evaluate model and report results frequently during training, otherwise the training can be significantly slowed down. Since sometimes a micro-divergence can happen and then recover very quickly, it is hard to even detect if any micro-divergence happens during model training without sacrificing training speed.
- **Measurement:** There's little research on quantitative measures of a model's training stability prior to training. To know (1) whether a modeling change will increase the risk of loss divergence, or (2) whether a mitigation can help reduce the risk of loss divergence, one has to rely on empirical evaluations (*i.e.,* train multiple copies of a model and check how many of them have issues), which are time-consuming and resource-intensive.

2.3 Related Work

Model training stability has been an under-explored research area, not only for recommendation models, but also in general machine learning. Fortunately, with the increasing trend of large models [8, 11, 29], stabilizing model training has become an emerging research area and attracts more attention in recent years.

From the perspective of optimization theory, Wu et al. [32] first theoretically predicted the training instability for quadratic models with learning rate and the "sharpness" (measured by the maximum eigenvalue of the loss Hessian) of the loss curvature. For deep neural networks, Cohen et al. [12], Gilmer et al. [14] confirmed that this prediction is still accurate enough.

In terms of techniques, there are some methods widely used in language and vision models, such as activation clipping [20], gradient clipping [24], learning rate warmup [16], and various normalization techniques [4, 18]. In addition, You et al. [34] proposed a new optimizer that achieves a better trade-off between convergence and stability for large batch-size training. Brock et al. [7] developed Adaptive Gradient Clipping to improve the stability of ResNet models [17] without Batch Normalization [18].

However, empirically, we found these approaches not effective enough to completely prevent our model from training instability (See Section 5). This may due to some unique properties of the recommendation model. As will be discussed in the next section, these properties can make multi-task ranking models more susceptible to training instability problems.

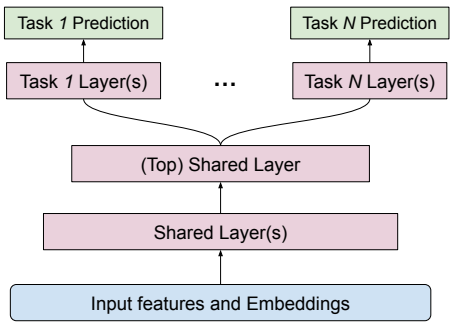


Figure 2: An general illustration of the ranking model used in recommender systems. The model has one or more layers that are (softly or fully) shared by multiple tasks.

3 UNDERSTANDING THE CAUSE OF THE ISSUE

In this section, we first describe the model to be studied in this paper and its characteristics. Then, we share our understanding on the root cause of the training instability problems that happened in our model.

3.1 Model Definition

YouTube's video recommendation system uses multiple candidate generation algorithms to retrieve a few hundred candidates. This is followed by a ranking system which generates a ranked list from these candidates. This paper mainly focuses on the ranking models in YouTube's recommender system. Different from candidate generation models (*a.k.a* retrieval models), which are responsible for filtering out the majority of irrelevant items, ranking models aim to provide a ranked list so that items with the highest utility to users are displayed at the top. Therefore, ranking models use more advanced machine learning techniques with more expensive features to have sufficient model expressiveness for learning the association of features and their relationship with utility.

Figure 2 depicts a general architecture of the ranking model that we want to study throughout the paper. Below we summarize some important features for our ranking model and how it is trained; one can refer to [36] for more details.

- **Multitask:** As shown in Figure 2, the ranking model has multiple tasks that predict multiple labels. These predictions are combined to form the final ranked list of items. Regardless of different modeling choices [9, 22], there are some hidden layers in the middle of the model that are shared by these tasks (either fully shared or softly shared).
- **Sequential training:** The model is trained sequentially, *i.e.,* the training is done over a corpus of data in sequential order from old to new. While unlike pure online learning [6, 25], which visits training data in strictly sequential order, we define a time-based moving window and randomly sample data batches from this window of training data for parallel

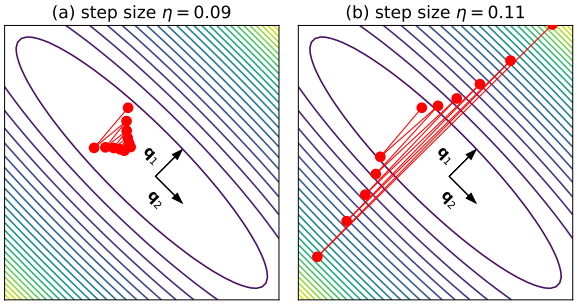


Figure 3: From [12, Figure 2]. Gradient descent on a quadratic model with eigenvalues $\alpha_1 = 20$ and $\alpha_2 = 1$. We can clearly observe training instability problems starting to occur when learning rate $\eta > 2/\alpha_* = 2/\alpha_1 = 0.1$.

training. This training scheme has been widely used and is known to be beneficial for many aspects of recommendation quality [2, 23, 36].

- **Optimization:** Large batch-size training is known to have less noise in gradients and thus optimization is more curvature driven [2, 34]. We adopt a large batch size with a high learning rate for faster convergence. We found Adagrad [13] to be strong in our case, despite many advances in optimizers (e.g., Adam [19], Adafactor [26]).

3.2 Root Cause and Case Studies

Regardless of the types of loss divergence, we believe that the intrinsic cause can be summarized as **"step size being too large when loss curvature is steep"**. Once a model meets both conditions at a given state, a divergence can easily occur. Intuitively, the step size should be conservative at a steep loss surface (measured by the maximum eigenvalue of the loss Hessian) to ensure that loss decreases instead of increases.

For quadratic models, Wu et al. [32] theoretically proves the above argument and suggests

$$2/\eta > \alpha_*$$

to make training stable, where η is the learning rate and α_* is the maximum eigenvalue of the loss Hessian. Cohen et al. [12] gives a nice and straightforward example (in Figure 3) for the proof. For neural networks, this argument still mostly holds [12, 14].

Knowing the root cause of the training instability problem allows us to answer the following research questions:

RQ1: Why do recommendation models in general have worse training stability than models in other domains?

RQ2: Within recommendation models, why do ranking models typically have worse training stability than retrieval models?

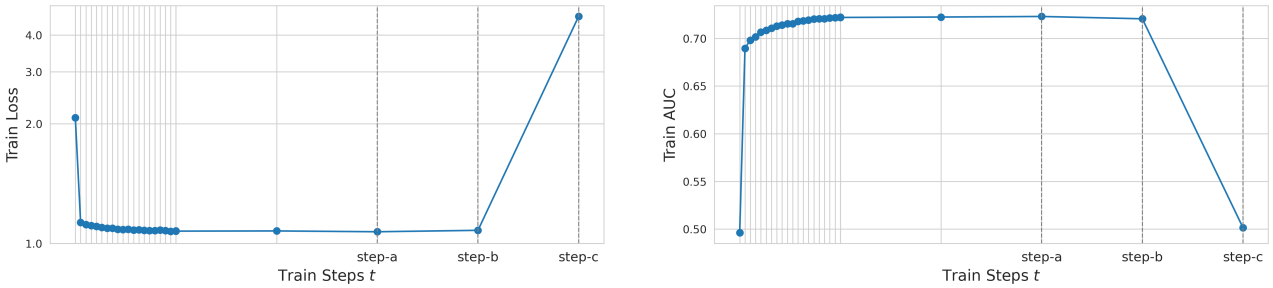
We relate the answers to these questions to the following unique properties of our models. Please refer to some empirical evidence in Section A.1 in Supplementary Material.

- **Data distribution changes (RQ1)**: Compared to models in other domains, recommendation models use several orders of magnitude more input features (hundreds to thousands). What's worse, with sequential training, the distribution of these input features (and labels) keeps changing. We think a steeper loss curvature can occur when data distribution sudden change, which happens regularly. Also, a model with sequential training will never converge as it has to adapt to newly arriving data points with a changed distribution. Thus, a large learning rate is required to make the adaptation efficient enough. In summary, compared to models in other domains that are trained on a fixed dataset, changes in the training data distribution pose greater challenges for stabilizing the training of recommendation models.
- **Larger model size and complexity (RQ2)**: Compared to retrieval models used for candidate generation, ranking models are usually much larger in capacity to accurately measure the utility of candidate items. With the recent developments of ML hardware (*e.g.,* TPUs), we are able to significantly increase the model size for quality improvements [3]. The empirical studies from Gilmer et al. [14] suggested the increased model capacity and complexity is a contributing factor to steeper loss curvature.
- **Multiple objectives vs. Single objective (RQ2)**: Compared to retrieval models which usually has a single objective (e.g. Softmax Cross-Entropy) [33], ranking models often need to optimize for many objectives at the same time [36]. This causes ranking models to suffer from loss divergence much more easily. Because if there are spurious gradients caused by bad predictions from a particular task, the gradients can back-propagate throughout the model, causing the layers that are shared by multiple tasks to behave (slightly) abnormally. But since the layers are shared by different tasks, other tasks tend to predict irregular values afterwards, reinforcing the instability to a nonrecoverable state. In other words, shared layers (as well as embeddings) can be a double-edged sword — they allow transfer learning from different tasks, but can also exacerbate the training instability problem, making ranking models more vulnerable than retrieval models.

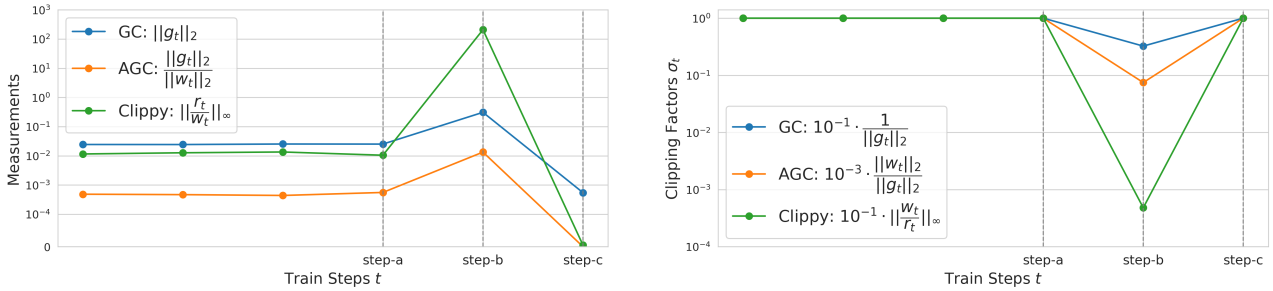
Despite the recent advances in understanding the root causes of divergence issues, we have found a large gap remains between our current understanding on the cause of the issue and having an effective solution. We have tried many temporary fixes. Some examples are: (1) Using even slower learning rate warmup schedule to pass the initial model state where loss curvature is steep [14]. (2) Enlarging the sequential training moving window to make training data distribution changes smoother. These fixes indeed mitigated training instability issues for a while, but when our model became more complex, loss divergence happened again. After trying many ad-hoc fixes, we believe developing a more principled way that can significantly improve model stability is the long-term solution.

4 EFFECTIVE METHOD FOR IMPROVING TRAINING STABILITY

In this section, we first introduce the general direction (Gradient Clipping) for controlling the effective step size while loss curvature



(a) Training loss (left) and AUC (right) at different steps.



(b) (left) Measurements used by different methods to determine clipping factors. (right) The corresponding clipping factors.

Figure 4: (a) We dive into three typical moments in model training: The model was training healthily before step-a. Then at step-b, model's loss aroused and AUC dropped. Finally at step-c, the loss is fully diverged and AUC dropped to 0.5. (b) When checking some statistics from the top hidden layer of the model, we found that GC and AGC failed to provide small enough clipping factor. While Clippy's clipping factor can be 2-orders of magnitude smaller than GC and AGC. Section B in Supplementary Material has the statistics for other layers.

is steep, by presenting some classical methods on this direction, accompanied with notations and denotations. Despite being successful when applied in other domains, we found these classical methods are not effective enough when applied in our model. Based on some observations of training dynamics in our model, we propose a new method and explain why it can be more effective for improving the training stability.

We first describe Adagrad [13], the optimizer used in our model. In Adagrad, model parameters $\boldsymbol{w}_t$ are updated by the rule

$$\begin{aligned} \boldsymbol{G}_t &= \boldsymbol{G}_{t-1} + \boldsymbol{g}_t^2, \\ \boldsymbol{r}_t &= \boldsymbol{g}_t \cdot \boldsymbol{G}_t^{-1/2}, \\ \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \eta_t \cdot \boldsymbol{r}_t, \end{aligned} \tag{1}$$

where η_t denotes the learning rate at step t, $\boldsymbol{g}_t$ is the standard stochastic gradient of the empirical loss with respect to the model parameters and $\boldsymbol{G}_t$, known as "accumulator", is a vector initialized to some small constant value, typically 0.1. In addition, all powers operations are computed element-wise.

As mentioned in Section 3, we desire a more principled approach to control the step size when loss curvature is steep. However, the loss curvature measured by the eigenvalue of loss Hessian is very expensive to compute during training. Fortunately, the first-order gradients $\boldsymbol{g}_t$ can be used as a surrogate for the Hessian (c.f. [35]). Consequently, gradient clipping based algorithms become very popular to improve training stability and used in many large models [8, 11, 29].

Gradient Clipping. Proposed by Pascanu et al. [24], Gradient Clipping (GC) limits the magnitude of gradient (measured by its norm) before applying it to the model. In other words, as gradient magnitude becomes large (loss curvature becomes steeper), Gradient Clipping controls the "effective step size" to stabilize model training.

Formally, Gradient Clipping algorithm clips the gradients $\boldsymbol{g}_t$ (before applying Adagrad update in equation 1) as:

$$\boldsymbol{g} \rightarrow \begin{cases} \lambda \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|} & \text{if } \|\boldsymbol{g}\| \geq \lambda, \\ \boldsymbol{g} & \text{else.} \end{cases} \tag{2}$$

$$\text{Or } \boldsymbol{g} \rightarrow \sigma \cdot \boldsymbol{g}, \text{ where } \sigma = \min\{\frac{\lambda}{\|\boldsymbol{g}\|}, 1.0\}$$

The clipping threshold λ is a hyperparameter that controls the maximum allowable gradient norm $\|\boldsymbol{g}\|$. In other words, if the model gradient $\boldsymbol{g}_t$ has a large magnitude at step t, GC will clip its norm to λ by rescaling gradients with a scalar *clipping factor* $\sigma \in \mathbb{R}^+$. In practice, Frobenius norm (or L_2 norm) $\|.\|_2$ is a common choice for vector norm, and clipping is often applied to each layer independently of the other layers.

Adaptive Gradient Clipping. Empirically, although GC can improve training stability of the model, training stability is extremely sensitive to the choice of the clipping threshold λ, requiring fine-grained tuning for different layers. What's worse, the threshold

λ need to be re-tuned when model structure, batch size, or learning rate is changed.

To overcome this burden, Brock et al. [7] proposed Adaptive Gradient Clipping (AGC). AGC is motivated by the observation that the ratio of the norm of the gradients $\|\boldsymbol{g}_t\|$ to the norm of the model parameters $\|\boldsymbol{w}_t\|$ should not be large, otherwise training is expected to be unstable.

Specifically, the gradients $\boldsymbol{g}$ is clipped by

$$\boldsymbol{g} \rightarrow \begin{cases} \lambda \frac{\|\boldsymbol{w}\|}{\|\boldsymbol{g}\|} \boldsymbol{g} & \text{if } \frac{\|\boldsymbol{g}\|}{\|\boldsymbol{w}\|} \geq \lambda, \\ \boldsymbol{g} & \text{else .} \end{cases} \tag{3}$$

$$\text{Or } \boldsymbol{g} \rightarrow \sigma \cdot \boldsymbol{g}, \text{ where } \sigma = \min\{\lambda \frac{\|\boldsymbol{w}\|}{\|\boldsymbol{g}\|}, 1.0\}$$

Intuitively, if at step t the gradient norm $\|\boldsymbol{g}_t\|$ is greater than a fraction of the parameter norm $\lambda \cdot \|\boldsymbol{w}_t\|$, AGC will clip the gradient norm to $\lambda \|\boldsymbol{w}_t\|$, by rescaling gradients with a scalar *clipping factor* $\sigma \in \mathbb{R}^+$. AGC can be viewed as a special case of GC, where the clipping threshold λ^{GC} is a function of model parameters $\lambda^{\text{GC}} = \lambda^{\text{AGC}} \|\boldsymbol{w}\|$. So when using AGC, we don't need to fine tune λ for different layers, this is where the "adaptiveness" comes from.

4.1 Observations of Training Dynamics

Despite the success of GC and AGC in various domains, we found that they are not effective enough to prevent loss divergence when being applied in our model. To better understand the limitations of GC/AGC and to propose better solutions, we inspect the training of our model without using any gradient clipping based techniques[1].

Figure 4a shows the training loss and AUC for a particular binary classification task. To simplify the illustration, let's look mainly at the 3 most important training steps: step-a, step-b, and step-c[2]. As we can see, this model is training healthily before step-a: the loss is minimized and the AUC has increased rapidly. However, at step-b, the model's training loss started to diverge and AUC began to drop, though relatively unnoticeably. Finally, at step-c, this model was fully diverged with loss become large, and AUC dropped to 0.5.

In Figure 4b(left), we take a closer look at some statistics for the *top shared layer* to understand what happened as loss diverged. The gradient norm $\|\boldsymbol{g}\|_2$ is pretty consistent before step-a when model is healthy. Then it grew to a large value at step-b, suggesting the loss curvature is quite steep at that moment. Since we didn't apply any model stability treatments, the model diverged completely at step-c and the gradient norm $\|\boldsymbol{g}\|_2$ became a small value. This means that all pre-activations (values before applying nonlinear activation) at this layer already reach a state where gradients are extremely small[3], causing the loss divergence to become nonrecoverable.

Knowing what happened, we fabricate how GC/AGC will react in this situation. Figure 4b(left) plots the measurements of $\|\boldsymbol{g}\|_2$ (blue) and $\frac{\|\boldsymbol{g}\|_2}{\|\boldsymbol{w}\|_2}$ (orange) that are used to determine clipping factors in GC and AGC. Not surprisingly, both measurements became larger at step-b. However, the relative scale of change for these measurements are different. $\frac{\|\boldsymbol{g}\|_2}{\|\boldsymbol{w}\|_2}$ (orange) is more sensitive to loss curvature changes than $\|\boldsymbol{g}\|_2$ (blue). The difference in sensitivity of these measurements can result in different *clipping factors* σ, which is the rescaling multiplier to the gradients in different methods. Figure 4b(right) gives the clipping factor σ for GC and AGC when using $\lambda^{\text{GC}} = 10^{-1}$ and $\lambda^{\text{AGC}} = 10^{-3}$ as clipping thresholds[4].

By checking the clipping factors, we hypothesize that the reason behind inefficacy of GC/AGC is that they failed to offer enough constraints on gradients (i.e., failed to provide enough control over the "effective step size") when gradient norm suddenly increases (i.e., loss curvature becomes steep), due to lack of sensitivity. More specifically, **both methods rely on L_2 norm, which is not sensitive to drastic gradient changes in only a few coordinates, especially when layer width is large**.

Algorithm 1 Adagrad with Clippy

1: **Input:** Parameter vector to optimize $\boldsymbol{w}$; objective function $\mathcal{L}$; learning rate schedule η_t.
2: **Input:** Clippy hyperarameters: relative threshold λ_{rel} and absolute threshold λ_{abs}.
3: Initialize parameter vector $\boldsymbol{w}_0$.
4: **for** $t = 0$ to $T-1$ **do**
5: $\quad \boldsymbol{g}_t = \frac{\partial \mathcal{L}(\boldsymbol{w}_t)}{\partial \boldsymbol{w}_t} \rightarrow$ obtain stochastic gradient.
6: $\quad \boldsymbol{G}_t = \boldsymbol{G}_{t-1} + \boldsymbol{g}_t^2 \rightarrow$ update accumulator
7: $\quad \boldsymbol{r}_t = \boldsymbol{g}_t \cdot \boldsymbol{G}_t^{-1/2} \rightarrow$ compute updates
8: $\quad \sigma_t = \min\{1.0, \min(\frac{\lambda_{\text{rel}}|\boldsymbol{w}_t| + \lambda_{\text{abs}}}{\eta_t * |\boldsymbol{r}_t|})\} \rightarrow$ get clipping factor
9: $\quad \boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \sigma_t \boldsymbol{r}_t \rightarrow$ apply rescaled updates
10: **end for**
11: **Return:** $\boldsymbol{w}_T$

4.2 Proposed Solution: Clippy

To alleviate this limitation, we proposed a new algorithm called Clippy. Clippy has two major changes over GC/AGC: First, it uses L_∞ norm instead of L_2 norm to increase its sensitivity to changes in individual coordinates. Second, it clips based on updates $\boldsymbol{r}_t = \boldsymbol{g}_t \cdot \boldsymbol{G}_t^{-1/2}$ instead of gradients $\boldsymbol{g}_t$, since updates are the actual change to model parameters and can be quite different from gradients when using the Adagrad optimizer.

Specifically, Clippy controls

$$\left\| \frac{\boldsymbol{r}_t}{\boldsymbol{w}_t} \right\|_\infty < \lambda, \tag{4}$$

and then rescales updates when the inequality is violated. From Figure 4b, we can see that this measurement has a a more dramatic change at step-b, when loss was diverging. Suppose we use $\lambda^{\text{Clippy}} = 10^{-1}$ as the clipping threshold, Clippy results in 2 orders of magnitude smaller clipping factors σ compared to GC/AGC, thanks to the better sensitivity of the measurement. In other words, we hope Clippy can put **larger constraints on the actual updates when loss curvature is steep even in a few coordinates**.

Formally, we present Clippy in Algorithm 1. As can be seen, there are some minor but important changes in line-8 of the algorithm compared to what we described in equation 4.

[1] Note the model we are inspecting here is the model-b in Figure 1

[2] The specific training step numbers are: step-a=198.7k, step-b=198.8k, step-c=198.9k.

[3] One typical example is the dying ReLU where most pre-activations are smaller than zero. It is worth noting that other nonlinear activations also have regions where gradients are close to zero, so can suffer from the same issue.

[4] As described in Section 5.1.3, we obtain these thresholds with grid-search and a even lower threshold can impact convergence.

(1) **Introducing absolute threshold.** In Clippy, we use two hyperparameters: The relative threshold λ_{rel} that is similar to GC/AGC, and another absolute threshold λ_{abs}. With the absolute threshold λ_{abs} introduced, we can avoid aggressive clipping when model parameters are zero (e.g., biases that are initialized to zeros) or have very small values. As will be discussed in Section 4.3.1, this allows Clippy to switch from GC-style to AGC-style during training.
(2) **Considering learning rate.** We have learning rate η_t in the denominator when calculating the clipping factor to account for different learning rate schedules. If the learning rate slowly ramps up, this will loosen the clipping threshold at initial training, avoiding a slow pace of convergence in the initial phases of training.

4.3 Additional Discussions

4.3.1 Relationship with other methods. Clippy has interesting connections with other methods. In gradient clipping based algorithms, if we accumulate the accumulator with original gradients (instead of clipped gradients). Then, we can have a general Adagrad update form with all aforementioned algorithms

$$\begin{aligned} \boldsymbol{r}_t &= \boldsymbol{g}_t \cdot \boldsymbol{G}_t^{-1/2}, \\ \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - (\eta_t \sigma_t) \boldsymbol{r}_t. \end{aligned} \tag{5}$$

That is, different algorithms scale down the learning rate η_t with different choices of clipping factor σ_t. And the choice of clipping factors by different algorithms are summarized in the table below.

Algorithm	σ_t
GC [24]	$\min\{1.0, \lambda \frac{1}{\|\boldsymbol{g}\|_2}\}$
AGC [7]	$\min\{1.0, \lambda \frac{\|\boldsymbol{w}\|_2}{\|\boldsymbol{g}\|_2}\}$
LAMB [34]	$\frac{\phi(\|\boldsymbol{w}_t\|_2)}{\|\boldsymbol{r}_t\|_2}$
Clippy (Ours)	$\min\{1.0, \min(\frac{\lambda_{\text{rel}}\|\boldsymbol{w}_t\|+\lambda_{\text{abs}}}{\eta_t * \|\boldsymbol{r}_t\|})\}$

Clippy is a combination of GC/AGC/LAMB: First of all, Clippy switches from GC-style to AGC-style during training. During initial model training when $|\boldsymbol{w}| \approx 0$, λ_{abs} dominates the clipping threshold $\frac{\lambda_{\text{rel}}|\boldsymbol{w}_t|+\lambda_{\text{abs}}}{\eta_t * |\boldsymbol{r}_t|} \approx \frac{\lambda_{\text{abs}}}{\eta_t * |\boldsymbol{r}_t|}$ and makes Clippy close to GC. In later training, when $\lambda_{\text{rel}}|\boldsymbol{w}| \gg \lambda_{\text{abs}}$, Clippy acts more like AGC $\frac{\lambda_{\text{rel}}|\boldsymbol{w}_t|+\lambda_{\text{abs}}}{\eta_t * |\boldsymbol{r}_t|} \approx \frac{\lambda_{\text{rel}}|\boldsymbol{w}_t|}{\eta_t * |\boldsymbol{r}_t|}$. However, compared to GC/AGC, Clippy relies on updates instead of gradients. Moreover, although both Clippy and LAMB use the updates, Clippy does not completely ignore the update magnitude as in LAMB[5]. Finally, Clippy uses L_∞ instead of L_2 norm to be more sensitive to drastic update changes in a small number of coordinates.

4.3.2 Clip locally or globally. When using Clippy, we clip the update per each layer (*a.k.a* locally) instead of per all model parameters as a whole (*a.k.a* globally), similar to the other methods (like GC/AGC/LAMB). This gives more flexibility on finer-grained control, but results in a biased gradient update. However, in large-batch settings, it can be shown that this bias is small [34].

[5]LAMB updates parameter by $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \frac{\boldsymbol{r}_t}{\|\boldsymbol{r}_t\|_2} \phi(\|\boldsymbol{w}_t\|_2)$, with $\phi(x) = \min\{\max\{x, \gamma_l\}, \gamma_u\}$ bounds the parameter L_2 norm. It uses only the direction of updates and ignores its magnitude.

Table 1: The configuration of each model setting.

Model Name	Non-Embedding Model Parameters	Shared bottom Architecture
`Small`	7.5M	FFN: 512×2
`Large`	57.0M	FFN: 4096×4
`Large+DCN`	68.0M	DCN + LN $\rightarrow 4096 \times 4$

4.3.3 Adapting to other optimizers. One can easily adapt Clippy to optimizers other than Adagrad by using the optimizer-dependent update $\boldsymbol{r}_t$. Empirically, we have also observed clear benefits in training stability when applying Clippy on Adam [19], without compromising convergence. But we leave the theoretical convergence analysis of Clippy to future work.

5 EMPIRICAL STUDIES

Conducted on a YouTube production dataset, experiments in this section are divided into two parts. Firstly, we compare Clippy with other baselines to verify its benefits for improving model stability. Then we show some further analyses for Clippy to better understand its strength.

5.1 Experiment Setup

5.1.1 Model detail. Besides all the model properties that are already covered in Section 3.1, it is worth mentioning that we simplified our ranking model by (1) Only keeping the most important subset of tasks and input features; (2) Using a simple shared bottom structure with several shared hidden layers. Though much simpler than the production model, we found it to be a sufficiently good testbed for studying the training stability problem, as it allows us to train models faster and focus more on research perspectives instead of irrelevant modeling details. The model is built with TensorFlow-2 [1] and is trained using a large batch size of 65k on TPUs.

5.1.2 Evaluation protocol. Unfortunately, there is no reliable metric to quantify the model's training stability. To precisely measure the benefit from better training stability, we vary model complexity as well as learning rates, then check model's offline quality, measured by AUC for binary classification tasks and RMSE for regression tasks. Presumably, a more complex model gives better offline quality but is more likely to suffer from loss divergence issues. So if an algorithm can significantly improve the model's training stability, we should observe better offline metrics when using it. More specifically, we used first $(N-1)$ days of data to sequentially train the model and continuously evaluate the model's performance (AUC or RMSE) on the last day (the N-th day) of data. If the model does not suffer from any loss divergence issues during training, we should observe the evaluation metrics keep becoming better, as the model is adapting to the data distribution closer to the N-th day of data. Whereas if the model's loss diverges during training, either fully-diverge or consistently micro-diverge, the evaluation metrics will be significantly impacted.

To explore the effect of model complexities, we consider various model settings summarized in Table 1. Both `Small` and `Large` use simple feed-forward networks as the shared bottom, with two 512 layers and four 4096 layers respectively. `Large+DCN` is built on top

Model Name	Metrics	Methods				
		Naive	GC	AGC	LAMB	Clippy
Small	AUC (higher is better)	*diverged*	71.68 ± 0.13	71.73 ± 0.00	71.56 ± 0.01	<u>71.79</u> ± 0.00
	RMSE (lower is better)		1.058 ± 0.002	1.059 ± 0.003	1.063 ± 0.001	<u>1.056</u> ± 0.000
	Best learning rate		2x	1x	1x	2x
Large	AUC (higher is better)	*diverged*	72.07 ± 0.05	72.09 ± 0.02	72.01 ± 0.09	<u>72.16</u> ± 0.02
	RMSE (lower is better)		1.053 ± 0.003	<u>1.051</u> ± 0.001	1.054 ± 0.002	<u>1.051</u> ± 0.000
	Best learning rate		2x	1x	1x	2x
Large+DCN	AUC (higher is better)	*diverged*	72.27 ± 0.03	72.06 ± 0.08	72.05 ± 0.11	<u>72.37</u> ± 0.01
	RMSE (lower is better)		1.049 ± 0.001	1.051 ± 0.001	1.057 ± 0.001	<u>1.047</u> ± 0.001
	Best learning rate		1x	2x	1x	2x

Table 2: Evaluation of training stability treatments on different model settings. Methods suffering from training instability problems should get worse evaluation metrics. We first find the best learning rate (1x or 2x) for each variant, then repeat the same setting 3 times and report mean and standard deviation. We use <u>underline</u> to denote the best result for each setting.

of Large and further adds more complexity by having DCN-v2 layers [31] on inputs, followed by a standard Layer Normalization [4].

5.1.3 Baselines. We apply Clippy and other baselines to non-embedding model parameters and compare their effectiveness. Below are more details about these baselines and Clippy.

- **Gradient Clipping (GC)** [24]: We used layer-wise (local) gradient clipping with clipping threshold searched from $\lambda^{\text{GC}} \in \{\underline{10^{-1}}, 10^{-2}, 10^{-3}\}$.
- **Adaptive Gradient Clipping (AGC)** [7]: We used the official implementation[6] provided in the paper and searched clipping threshold from $\lambda^{\text{AGC}} \in \{10^{-2}, \underline{10^{-3}}, 10^{-4}\}$.
- **LAMB (adapt to Adagrad)** [34]: LAMB was originally proposed based on Adam [19], while the authors also provided a general form for the clipping which we introduced in Section 4.3.1. We choose $\phi(x) = x$ as in the official implementation[7]. Since LAMB uses parameter L_2 norm $\|\boldsymbol{w}\|_2$ as update magnitude that is different from other methods, we have to scale the learning rates by μ and searched $\mu \in \{10^{-1}, 10^{-2}, \underline{10^{-3}}\}$.
- **Clippy**: Clippy has two hyperparameters λ_{abs} and λ_{rel} so suppose to be more non-trivial for the tunings, but we found simply setting $\lambda_{\text{rel}} = 0.5$ and $\lambda_{\text{abs}} = 10^{-2}$ gives decent performance in our experiments. .

5.2 Overall Performance

Table-2 presents the overall comparison between Clippy and other baselines on different model settings. Though the model is trained on six tasks, due to space limitations, we only present the metrics from two most representative tasks — one binary classification task evaluated with AUC (in percentage) and another regression task evaluated with RMSE. We not only use the original learning rate but also try to double the learning rate and see if any method can benefit from it. After finalizing the best learning rate, we repeat the same setting 3 times with different random seeds and report the mean and standard deviation.

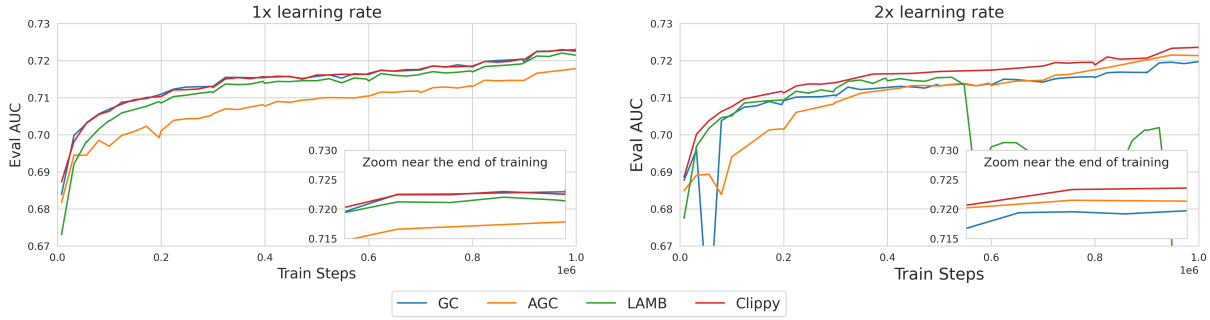
[6]https://github.com/deepmind/deepmind-research/tree/master/nfnets
[7]https://github.com/tensorflow/addons/blob/master/tensorflow_addons/optimizers/lamb.py

Looking at Table 2, we can see the naive method which does not have any treatments on training stability always suffers from loss divergence, even on the Small model. There is a chance for it to survive if we drastically tune down the learning rate (see Section A.1 in Supplementary Material) but we omit its results here as they are bad. GC can survive with 2x learning rate and provide good results on Small and Large model. But in a more complex model with DCN, GC can only use 1x learning rate, otherwise it will suffer from loss divergence issues (see blue line in Figure 5a right). AGC did a reasonable job on Small and Large with 1x learning rate, but became bad with 2x learning rate. On Large+DCN, AGC shows very high variance using either 1x or 2x learning rate (see orange line in Figure 5a), suggesting AGC already reaches its limits on keeping training stable. LAMB successfully trains the model without suffering from training instability problems using 1x learning rate, but the convergence is negatively impacted. On Figure 5a, we found the results from LAMB are always worse than the other methods. We believe this is due to LAMB completely ignoring the update magnitude, causing the convergence at initial training to be very slow when parameter L_2 norm is small. Surprisingly, GC performs the best on all settings among all the baselines, this could be because the model is relatively simple thus tuning the clipping threshold for GC is still trivial.

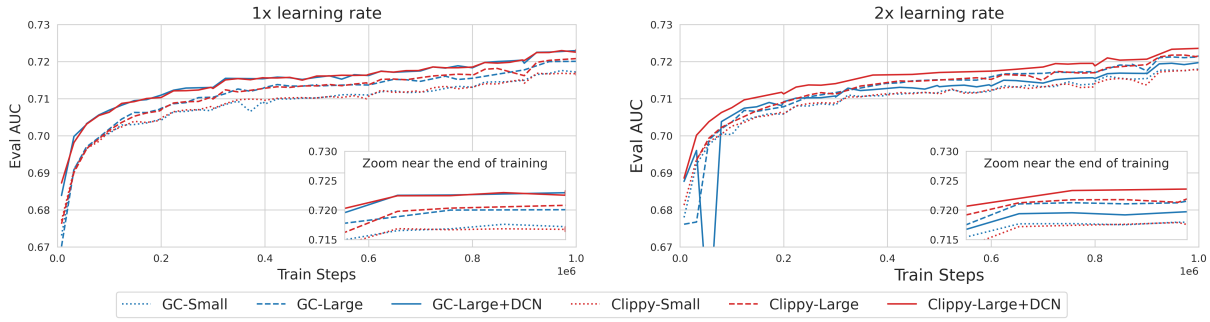
On the last column of Table 2, we can see Clippy handles all model settings with 2x learning rate. More importantly, Clippy doesn't compromise convergence, it has comparable results with GC (*i.e.,* the best baseline) on Small and Large model (see Figure 5b), and having significantly better AUC (Note 0.1% AUC improvement in our model is considered very significant and can lead to live metric gains) and RMSE on Large+DCN model compared to GC. One important finding we want to highlight is that Clippy offers larger gains when the model is more complex and trained with a larger learning rate. On Figure 5b, we can see gap between Clippy and GC is getting larger when using a more complex model with 2x learning rate. So we are not surprised Clippy can help in the production model which is much more complex than Large+DCN.

5.3 Closer Look at Clippy's Clipping Factors

Figure 6 shows Clippy's clipping factor on different layers during training the Large+DCN model. As introduced in Section 4, the



(a) AUC for different methods during the training on `Large+DCN` model.



(b) AUC of Clippy and GC (the best baseline) during the training on different model settings.

Figure 5: Evaluation AUC vs. Training steps for different methods in different model settings.

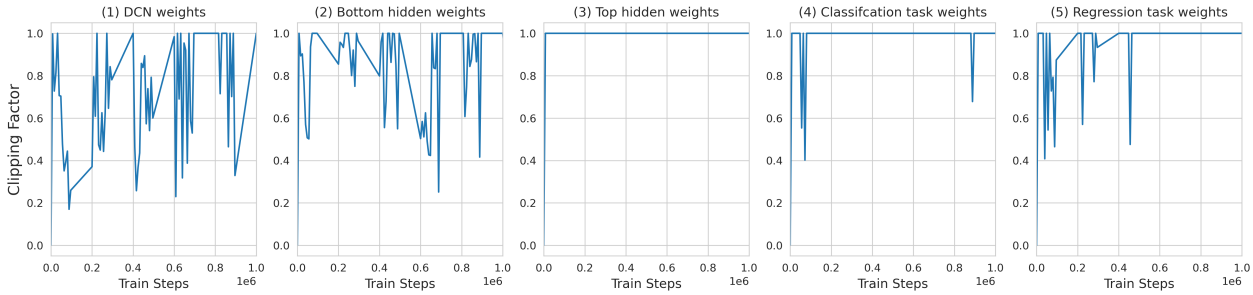


Figure 6: Clippy's clipping factor on different layers during training the `Large+DCN` model.

clipping factor $\sigma \in (0.0, 1.0]$. A smaller clipping factor indicates more clipping is done to scale down the learning rate. Since the clipping is applied per layer, we plot the clipping factor for several layers, including the weights in (1) DCN layer, in (2) top and (3) bottom hidden layer of shared bottom, and in the output layers of (4) binary classification task and (5) regression task. It is interesting to see more clipping is done on the bottom layers of the model. We think this intuitively makes sense, because bottom layers usually have smaller parameter norm so Clippy's clipping threshold will also be smaller. On the other hand, this could potentially benefit training stability because we know a small change in the bottom layer weights can lead to a large difference in model outputs.

6 CONCLUSION AND FUTURE WORK

In this paper, we present the training instability problem that happen recurrently in ranking models at YouTube. We show the importance and challenges of mitigating this problem in the long run. To better understand the issue, we dive deep into the problem, trying to know the root cause of why conventional methods did not work well in our case. With our understandings, we propose a new clipping based method called Clippy, which has a nice relationship with existing methods but alleviates the limitations of them. From empirical studies on the YouTube production dataset, we found Clippy showed significantly better strength on improving model training stability than other baselines.

Clippy showed significant improvements on training stability in multiple ranking models for YouTube recommendations. It is productionized in some large and complex models. More importantly, it unblocks several ongoing modeling developments and alleviates us from training instability problems that happened recurrently.

As for future work, we hope to theoretically justify the effectiveness of Clippy and provide convergence guarantees. In addition, we hope evolution-based AutoML algorithms [10, 27] can be applied here to do a better job than the human-designed clipping methods.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning.. In *OSDI*. Savannah, GA, USA.

[2] Rohan Anil, Sandra Gadanho, Da Huang, Nijith Jacob, Zhuoshu Li, Dong Lin, Todd Phillips, Cristina Pop, Kevin Regan, Gil I Shamir, et al. 2022. On the Factory Floor: ML Engineering for Industrial-Scale Ads Recommendation Models. *arXiv preprint arXiv:2209.05310* (2022).

[3] Newsha Ardalani, Carole-Jean Wu, Zeliang Chen, Bhargav Bhushanam, and Adnan Aziz. 2022. Understanding Scaling Laws for Recommendation Models. *arXiv preprint arXiv:2208.08489* (2022).

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[5] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *International Conference on Web Search and Data Mining*. ACM, 46–54.

[6] Léon Bottou and Yann Cun. 2003. Large scale online learning. *Advances in neural information processing systems* 16 (2003).

[7] Andy Brock, Soham De, Samuel L Smith, and Karen Simonyan. 2021. High-performance large-scale image recognition without normalization. In *International Conference on Machine Learning*. PMLR, 1059–1071.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[9] Rich Caruana. 1997. Multitask learning. *Machine learning* 28, 1 (1997), 41–75.

[10] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Yao Liu, Kaiyuan Wang, Cho-Jui Hsieh, Yifeng Lu, and Quoc V Le. 2022. Evolved Optimizer for Vision. In *First Conference on Automated Machine Learning (Late-Breaking Workshop)*.

[11] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).

[12] Jeremy M Cohen, Simran Kaur, Yuanzhi Li, J Zico Kolter, and Ameet Talwalkar. 2021. Gradient descent on neural networks typically occurs at the edge of stability. *arXiv preprint arXiv:2103.00065* (2021).

[13] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[14] Justin Gilmer, Behrooz Ghorbani, Ankush Garg, Sneha Kudugunta, Behnam Neyshabur, David Cardoze, George Dahl, Zachary Nado, and Orhan Firat. 2021. A Loss Curvature Perspective on Training Instability in Deep Learning. *arXiv preprint arXiv:2110.04369* (2021).

[15] Justin M. Gilmer, George E. Dahl, and Zachary Nado. 2021. *init2winit: a JAX codebase for initialization, optimization, and tuning research.* http://github.com/google/init2winit

[16] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

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

[19] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[20] Alex Krizhevsky and Geoff Hinton. 2010. Convolutional deep belief networks on cifar-10. *Unpublished manuscript* 40, 7 (2010), 1–9.

[21] Hyodong Lee, Joonseok Lee, Joe Yue-Hei Ng, and Paul Natsev. 2020. Large scale video representation learning via relational graph clustering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6807–6816.

[22] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. 2018. Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1930–1939.

[23] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1222–1230.

[24] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*. PMLR, 1310–1318.

[25] Shai Shalev-Shwartz et al. 2012. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning* 4, 2 (2012), 107–194.

[26] Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*. PMLR, 4596–4604.

[27] David R So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. 2021. Primer: Searching for efficient transformers for language modeling. *arXiv preprint arXiv:2109.08668* (2021).

[28] Jiaxi Tang, Francois Belletti, Sagar Jain, Minmin Chen, Alex Beutel, Can Xu, and Ed H. Chi. 2019. Towards neural mixture recommender for long range dependent user sequences. In *The World Wide Web Conference*. 1782–1793.

[29] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239* (2022).

[30] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. *Advances in neural information processing systems* 26 (2013).

[31] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. 2021. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the Web Conference 2021*. 1785–1797.

[32] Lei Wu, Chao Ma, and Weinan E. 2018. How sgd selects the global minima in over-parameterized learning: A dynamical stability perspective. *Advances in Neural Information Processing Systems* 31.

[33] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. 2019. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 269–277.

[34] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962* (2019).

[35] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. 2020. Why Gradient Clipping Accelerates Training: A Theoretical Justification for Adaptivity. In *International Conference on Learning Representations*.

[36] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending what video to watch next: a multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 43–51.

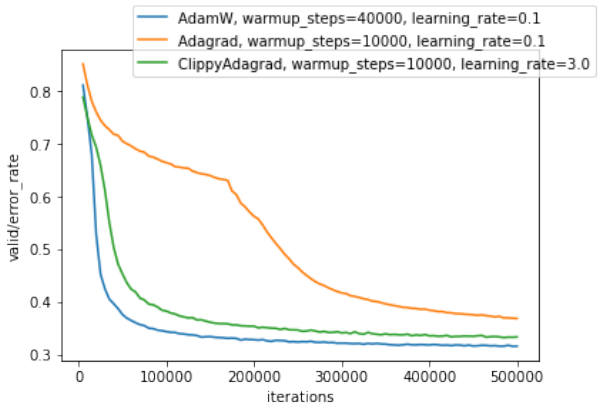


Figure 7: A comparison of AdamW, Adagrad and Adagrad with Clippy on the task for English to German translation.

A ADDITIONAL EMPIRICAL STUDIES

A.1 Empirical Evidence for Section 3.2

In Table 3, we show more empirical evidence on how the unique properties of multitask ranking models in the recommendation domain can affect training stability. All the experiments are performed on the `Large+DCN` model without any treatments on training stability. We used 0.4x learning rate to make the training at the edge of instability and to see the benefits from other changes on improving training stability.

As a result, we can see that the loss of the model kept diverging if no change is applied: 5 out of 5 runs has loss fully diverged. However, if we reduce the model size (by switching to `Small+DCN` model) or remove DCN-v2 layers (by switching to `Large` model), there will be some surviving cases. Moreover, if we remove a subset of input features or one or more output tasks, we can also observe their benefits on training stability. Due to the large training cost for each trial, we cannot offer more data points, but we hope these results can support for the hypotheses and claims in Section 3.2.

A.2 A Transformer-based model

In this section, we report the performance of Clippy in an additional setting. We based our experiment on init2winit[8]'s [15] 'translate_wmt' dataset with the default 'xformer_translate' model, containing six encoder and six decoder layers for the task of English to German translation. We compared the default AdamW optimizer to Adagrad and to Adagrad with Clippy. For both AdamW and Adagrad optimizers we tested the learning rates [0.01, 0.03, 0.1, 0.3, 1.0], while for Adagrad with Clippy we used learning rate in [0.1, 0.3, 1.0, 3.0, 10.0, 30.0] and set $\lambda^{\text{GC}} = 0.1$, $\lambda^{\text{AGC}} = 10^{-3}$. All experiments were executed twice for 500k steps, once a with warm-up period 10k steps and a second time with a 40k step warm-up period.

When the warm-up period was 10k steps, AdamW diverged when the learning rate was 0.1 and diverged for learning rate equal to 1.0 when the warm-up period was at 40k steps. Adagrad diverged for learning rate equal to 0.3 in both cases. On the other hand, Adagrad with Clippy did not diverge for any of our experiments. In a few of our earlier trials, Adagrad with Clippy did start to slowly diverge, however, that divergence was transitory and the model later recovered.

[8] https://github.com/google/init2winit

The baseline AdamW optimizer attained the best result with a learning rate of 0.1 and 40k warm-up period, reaching a validation error rate of 31.6%. Adagrad's best run was with a learning rate of 0.1 and 10k warm-up period, reaching a validation error rate of 36.9%, while Adagrad with Clippy's best run was with learning rate of 3.0 and 10k warm-up steps, reaching an error rate of 33.4% on the validation set. See Figure 7 for the validation error rate throughout the training process.

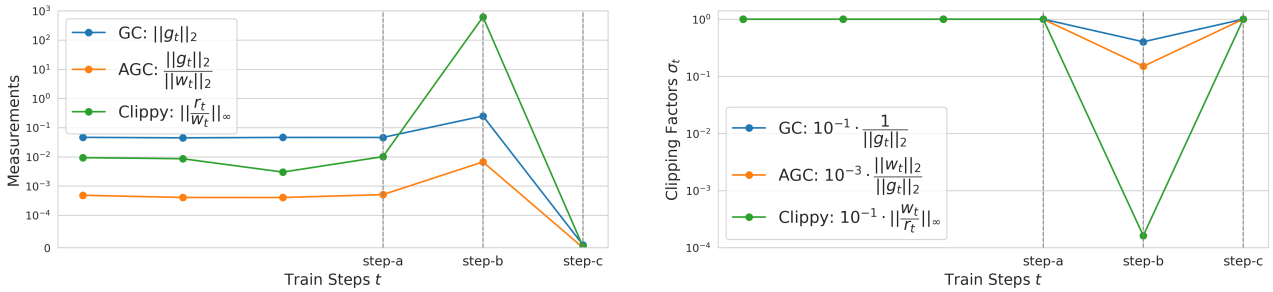
Note that although Adagrad with Clippy did not reach the performance of AdamW, which is considered state-of-the-art for this task, it did show significant improvement over the Adagrad implementation. Furthermore, we did not attempt to tune any of the model's parameters beyond what is reported above, opening the way to further improvements.

B STATISTICS FROM OTHER LAYERS

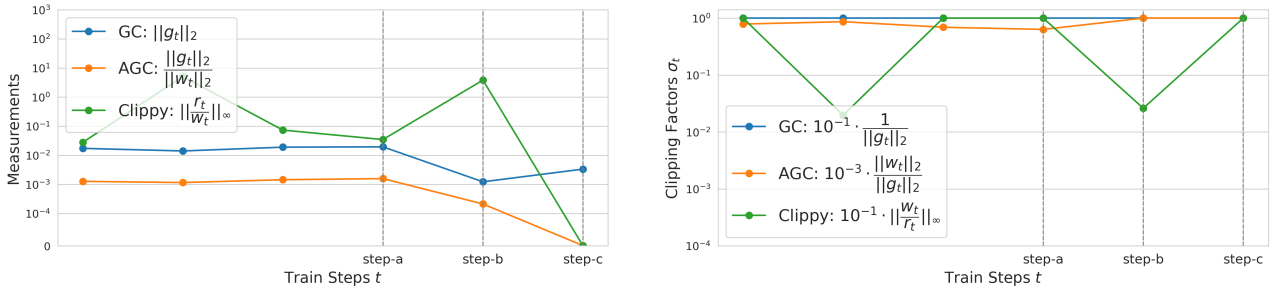
Besides the top hidden layer weights presented in Figure 4b, we also show statistics from other representative layers in Figure 8. From the figure, we can see other layers behave similarly as the top hidden layer at step-b, except for the binary classification layer. From Figure 8b, we found measurements used by GC/AGC even failed to capture the sudden changes in model parameters at step-b, resulting in no clipping applied at step-b for the binary classification layer weights by GC/AGC.

Direction	Specific Change	#Diverged	#Tried	Divergence Ratio
N/a	Clean	5	5	100%
Model size	Smaller model size	1	3	33%
Model complexity	Remove DCN	2	3	66%
Input features	Remove subset of input features	2	3	66%
Output tasks	Remove one task	2	3	66%
	Remove two tasks	0	3	0%

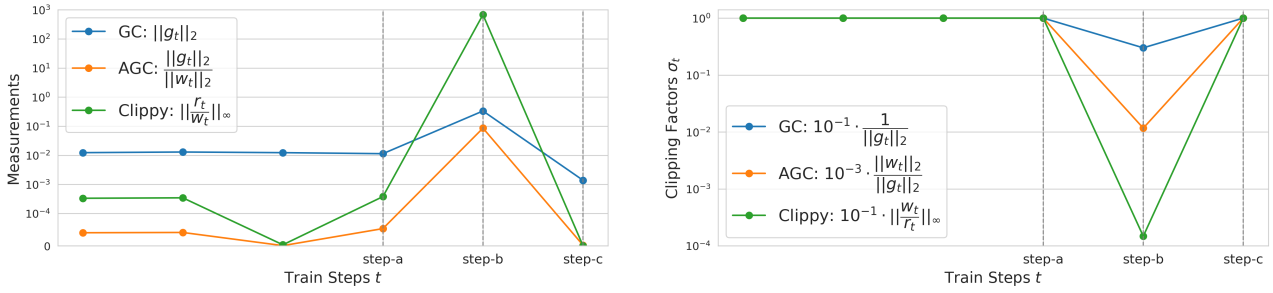
Table 3: The impact of different model properties to training instability. `Large+DCN` model is used without any treatments on training stability and with 0.4x learning rate.



(a) Bottom hidden layer weights.



(b) Classification task layer weights.



(c) Regression task layer weights.

Figure 8: (left) Measurements used by different methods to determine clipping factors. (right) The corresponding clipping factors.