Google

Improving Training Stability for Multitask Ranking Models in Recommender Systems

Jiaxi Tang[*1], Yoel Drori[*2], Daryl Chang[*3], Maheswaran Sathiamoorthy[1],

Justin Gilmer[1], Li Wei[3], Xinyang Yi[1], Lichan Hong[1], Ed H. Chi[1]

[1] Google Deepmind [2] Google Research [3] Google Inc
(* Equal contribution to the work)

Model Training Instability: An increasingly important issue

- Affected many models at YouTube
- Becoming a core challenge in Vision / LLMs / etc

5.1 Training Instability

For the largest model, we observed spikes in the loss roughly 20 times during training, despite the fact that gradient clipping was enabled. Those spikes occurred at highly irregular intervals, sometimes happening late into training, and were not observed when training the smaller models. Due to the cost of training the largest model, we were not able to determine a principled strategy to mitigate these spikes.

Instead, we found that a simple strategy to effectively mitigate the issue: We re-started training from a checkpoint roughly 100 steps before the spike started, and skipped roughly 200–500 data batches, which cover the batches that were seen before and during the spike. With this mitigation, the loss did not spike again at the same point. We do not believe that the spikes were caused by "bad data" per se, because we ran several ablation experiments where we took the batches of data that were surrounding the spike, and then trained on those same data batches starting from a different, earlier checkpoint. In these cases, we did not see a spike. This implies that spikes only occur due to the combination of specific data batches *with* a particular model parameter state. In the future, we plan to study more principled mitigation strategy for loss spikes in very large language models.

changes had clear effects on validation perplexity.

2.5 Training Processes

Here we describe significant training process adjustments that arose during OPT-175B pre-training.

Hardware Failures We faced a significant number of hardware failures in our compute cluster while training OPT-175B. In total, hardware failures contributed to at least 35 manual restarts and the cycling of over 100 hosts over the course of 2 months. During manual restarts, the training run was paused, and a series of diagnostics tests were conducted to detect problematic nodes. Flagged nodes were then cordoned off and training was resumed from the last saved checkpoint. Given the difference between the number of hosts cycled out and the number of manual restarts, we estimate 70+ automatic restarts due to hardware failures.

Loss Divergences Loss divergences were also an issue in our training run. When the loss diverged, we found that lowering the learning rate and restarting from an earlier checkpoint allowed for the job to recover and continue training. We noticed a correlation between loss divergence, our dynamic loss scalar crashing to 0, and the l^2-norm of the activations of the final layer spiking. These observations led us to pick restart points for which our dynamic loss scalar was still in a "healthy" state (≥ 1.0), and after which our activation norms would trend downward instead of growing unboundedly. Our empirical LR schedule is shown in Figure 1. Early in training, we also noticed that lowering gradient clipping from 1.0 to 0.3 helped with stability; see our released logbook for exact details. Figure 2 shows our validation loss with respect to training iterations.

Other Mid-flight Changes We conducted a number of other experimental mid-flight changes to handle loss divergences. These included: switching to vanilla SGD (optimization plateaued quickly, and we reverted back to AdamW); resetting the dynamic loss scalar (this helped recover some but not all divergences); and switching to a newer version of Megatron (this reduced pressure on activation norms and improved throughput).

Training instability is common in LLMs like PaLM (left) and OPT (right).

Highlights of this Work

- We show some implications and consequences of unstable model training.
  - Suggesting the importance and difficulties of the problem in real-world recommender systems.
- We present case studies about model changes that had led to more training instabilities.
  - Suggesting the prevalence of the problem.
- We provide our understanding on the root cause of the problem.
- We propose an effective approach (called *Clippy*) to overcome the problem.
  - By closely examining the training dynamics of a real ranking model at YouTube.
  - *Clippy* is deployed in several ranking models for YouTube recommendations.

Agenda

- Background and Motivation
- Understanding the Root Cause
- Proposed Solution – Clippy
- Empirical Studies

1. Background and Motivation

Background: Multitask Ranking Model at YouTube

- **Multitask**
  - Fully shared bottom OR
  - Softly shared bottom (MoE)
- **Sequential Training**
  - Training data visitation in time order
- **Optimization**
  - **Large** batch size + **High** learning rate
  - Optimizer=Adagrad

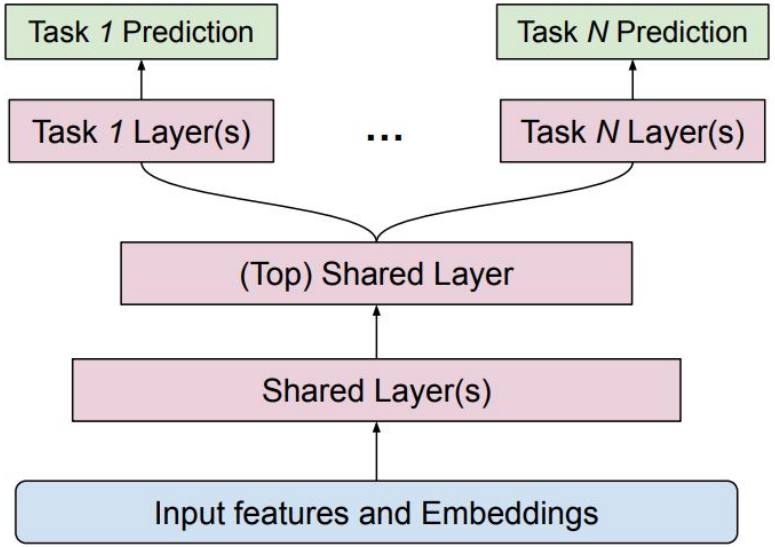
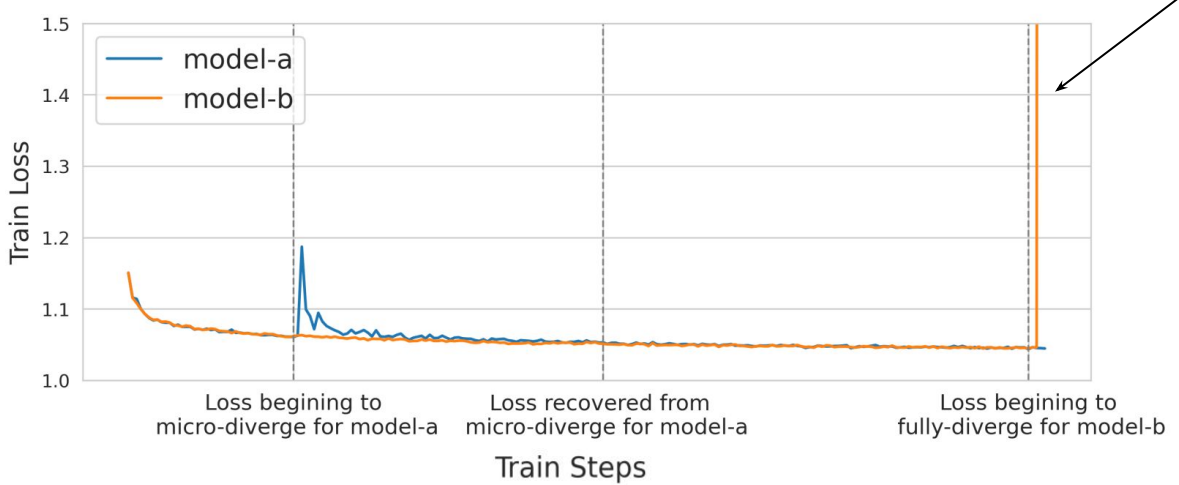


Figure: An general illustration of the ranking model used in recommender systems

Problem: Training Instability

- **Symptom**: *Loss divergence*
  - Training loss shoot up ⟶ All metrics become worse

Symptom:
Loss diverges to large values
⟶ Model becomes useless



- **Two types**:
  - *Micro loss divergence* (model-a): Recoverable
  - *Full loss divergence* (model-b): Non-recoverable ⟶ model become *useless*

Motivation

- **Important problem for Recommender Systems in industry**
  - Training instability can *easily happen*, from
    - increasing model complexity
    - adding more input features or tasks
    - increasing convergence speed
      - E.g., larger learning rate

    } Common types of model development

  - And once training becomes unstable, it can
    - *waste* a lot of training resources
    - *hinder (or even block)* model development
    - negatively affect user experience (if a bad model is served)

Challenges

- **Hard to *reproduce***
  - Same model config does not always have loss divergence.
  - Loss divergence does not always happen at the same time.
- **Hard to *detect***
  - Loss might diverge then recover before metrics are logged.
- **Hard to *measure***
  - No quantitative measurement of model instability.
- **Ad-hoc mitigations can only fix *temporarily***
  - **Engineering**: Automatically rollback; Validate model quality before serving.
  - **Modeling**: Activation clipping; Gradient clipping.
  - However, **none of them** can prevent us from the issue in the long-term.

2. Understanding the Root Cause

We hope to investigate

- **Root cause** of the problem
  - In order to come up with a more principled long-term solution.

- Two **research questions** (omitted, but discussed in the paper)

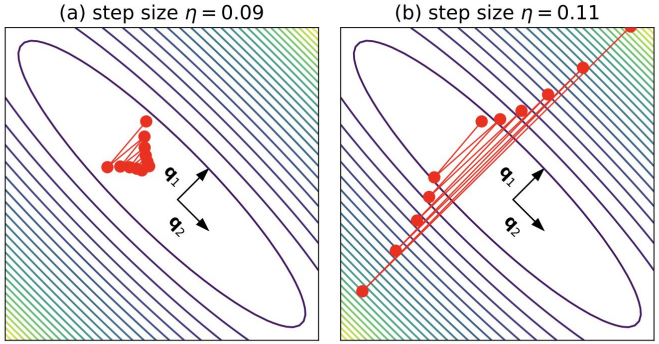
RQ1: *Why do recommendation models frequently suffer from training instability issues?*
  - *Observations: Even small (O(1M) dense params) recommendation models can easily suffer from the issue.*

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

Root cause of the problem

- **Root cause:** *Step size too large when loss curvature is steep.*

- Cohen et al and Gilmer et al showed that we should keep η < 2 / λ to stabilize model training.
  - η = learning rate
  - λ = maximum eigenvalue of the training loss Hessian, which describes the *sharpness of the loss curvature*



(source) 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 $\eta > 2/\alpha* = 2/\alpha 1 = 0.1$.

3. Proposed Solution – Clippy

Background: Gradient Clipping*

Clipped Gradient

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

Limits on gradient norm

Pros

- Easy to implement
- Well-studied

Cons

- Parameter(s) hard to tune

* On the difficulty of training recurrent neural networks, Pascanu et al.

Background: Adaptive Gradient Clipping (AGC)*

Clipped Gradient

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

Limits on the ratio of gradient norm / parameter norm

Pros

- Easy to implement
- Easier to tune

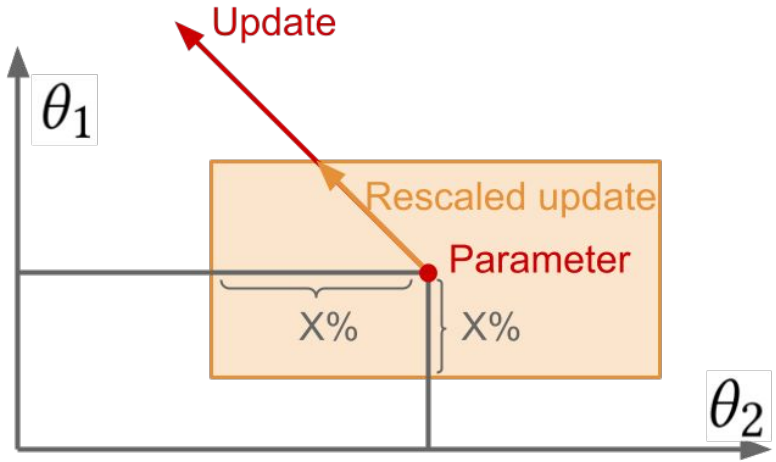
Cons

- Not invariant under translations of θ

* High-performance large-scale image recognition without normalization, Brock et al.

Proposed method (Clippy)

- **Similar idea to AGC, but**
  - **Controls the updates** instead of gradients:
    - Parameter update cannot be too large than the parameter itself.
  - Uses **L-infinity norm** instead of L2 norm:
    - More sensitive to large updates in a few coordinates.



Detailed Algorithm

Algorithm 1 Adagrad with Clippy

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

4. Empirical Studies

Benchmark Setting

- **Simplified version** of prod Multitask Ranking model
  - Subset of inputs (~200 out of ~500) + subset of tasks (5 out of 10) according to importance.
  - Shared bottom w/ several hidden layers.

  → allows us to focus more on research perspectives instead of irrelevant modeling details.

- **Baselines**
  - **GC** [1]: Layer-wise gradient clipping, clips ||g||.
  - **AGC** [2]: Adaptive layer-wise gradient clipping, clips ||g|| / ||w||.
  - **LAMB** [3]: Adapted to Adagrad, L2-normalize update and scale it by ||w||.

- **Variants**
  - **Models**: `Small` / `Large` / `Large+DCN`[4]
  - **Learning rate**: 1x or 2x

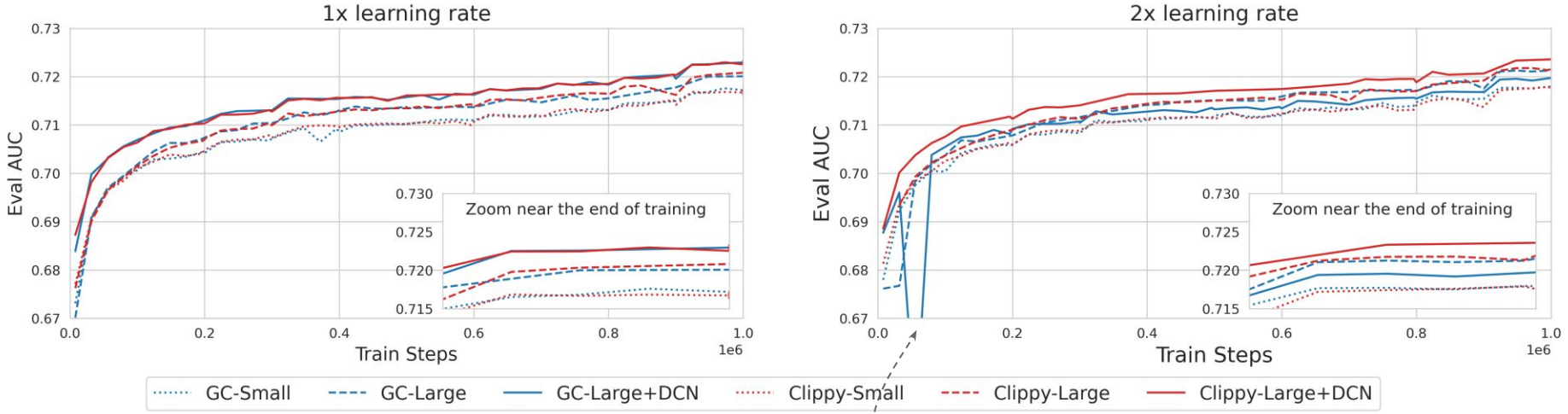
[1] On the difficulty of training recurrent neural networks, Pascanu et al.
[2] High-performance large-scale image recognition without normalization, Brock et al.
[3] Large Batch Optimization For Deep Learning: Training Bert In 76 Minutes, You et al.
[4] DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems, Wang et al.

Empirical Effectiveness

- **Under large learning rates:**
  Clippy prevents loss divergence.
- **Under small learning rates:**
  Clippy does not hurt performance.

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	71.79 ± 0.00
	RMSE (lower is better)		1.058 ± 0.002	1.059 ± 0.003	1.063 ± 0.001	1.056 ± 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	72.16 ± 0.02
	RMSE (lower is better)		1.053 ± 0.003	1.051 ± 0.001	1.054 ± 0.002	1.051 ± 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	72.37 ± 0.01
	RMSE (lower is better)		1.049 ± 0.001	1.051 ± 0.001	1.057 ± 0.001	1.047 ± 0.001
	Best learning rate		1x	2x	1x	2x

Empirical Effectiveness (con't)



With 1x LR

- **Clippy** ~= **GC** on all model variants

With 2x LR

- **Clippy** ~= **GC** on `Small` / `Large`
- **Clippy** > **GC** on `Large+DCN`
- **GC** suffered from *recovering from micro-divergence issues*

Thanks!

DOI:
https://doi.org/10.1145/3580305.3599846

Arxiv:
https://arxiv.org/abs/2302.09178

Code:
https://github.com/tensorflow/recommenders/blob/main/tensorflow_recommenders/experimental/optimizers/clippy_adagrad.py