Incremental Learning from Asynchronously Trained Neural Networks in e-Commerce Search Ranking

Zhipeng Xu
Guoyu Tang
Dadong Miao
xuzhipeng19@jd.com
tangguoyu@jd.com
miaodadong@jd.com
JD.com
Beijing, People's Republic of China

Lin Liu
Sulong Xu
liulin1@jd.com
xusulong@jd.com
JD.com
Beijing, People's Republic of China

Bo Long
Yun Xiao
Yunjiang Jiang
bo.long@jd.com
xiaoyun1@jd.com
yunjiangster@gmail.com
JD.com Silicon Valley R&D Center
Mountain View, CA, USA

ABSTRACT

Asynchronous distributed learning is widely used in industry-scale neural net training. Incremental learning is another popular machine learning procedure used in time-sensitive business domains such as personalized e-commerce search, where millions of new items and user interactions enter the system on a daily basis.

Despite both being effective, time-tested learning paradigms, there has been surprisingly no study on how well they perform when combined in an end-to-end learning framework. In this work, we carry out experiments to examine how these two techniques interact. The first surprising observation is sharp model quality drops across all metrics, when a model is warm-started from parameters saved from an asynchronously trained base model.

We analyze the implementation of the popular parameter server architecture, and make some hypothesis in order to make sense of the observation, as well as provide practical solutions to solve or mitigate the problem. We show that random re-initialization of select groups of weights, as well as proper mixture of old and new data, allow the model to hill-climb back to metrics achieved before restarting, and sometimes exceed earlier peaks. We also discuss how these techniques help with incremental learning that is critical for result freshness in e-commerce search ranking.

ACM Reference Format:
Zhipeng Xu, Guoyu Tang, Dadong Miao, Lin Liu, Sulong Xu, Bo Long, Yun Xiao, and Yunjiang Jiang. 2021. Incremental Learning from Asynchronously Trained Neural Networks in e-Commerce Search Ranking. In *Proceedings of DLP-KDD 2021.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In recent years, deep learning models have gradually taken the center stage of personalized search and recommendation ranking systems. Despite its effectiveness, deep learning model often takes an exorbitant amount of time to train compared to its predecessors, especially when compared with classical methods such as gradient boosted decision trees, or the family of linear methods. On the other hand, newly logged user interaction data comes in on a daily or even hourly basis. Missing out on the latest data can significantly hamper model prediction accuracy and user experience.

In this work, we attempt to overcome both challenges through the use of asynchronously distributed training for the base model and incremental learning for newly arrived data. In the process, however, we discover that asynchrony and incremental learning do not work well together. In fact, naively warm-starting from the asynchronously trained base model invariably causes an initial sharp drop in test metrics, which often never recovers to its starting value. This is demonstrated on both an in-house e-commerce search dataset as well as the public Amazon review dataset.

We provide some preliminary explanation as to why this occurs, as well as the following two effective techniques to mitigate this problem:

- Randomly re-initialize a subset of the model parameters, typically the last few layers of the final DNN network.
- Mix the old and new data during incremental training.

While these two simple methods do not address the symptom directly, they successfully reconcile asynchronous (pre-)training and incremental model update in a large scale industrial setting, bringing about significant gains in both offline and online metrics.

2 RELATED WORK

2.1 Incremental Learning

Incremental learning [9] offers a practical solution to bridge the gap between training time and model freshness. The idea is to train a base model on a large amount of data, followed by incremental updates when new data gradually becomes available. The second phase can typically be executed quickly, due to small data size. The increment step can often piggyback from the near optimal base model and reach new heights, thanks to the newly available data unseen by the base model.

In a fast-paced setting like e-commerce search ranking, where a large fraction of new items replace old ones on a daily basis, training speed is critical to the core business. In fact, training a model from scratch is often considered too inefficient. Thus incremental

DLP-KDD 2021, August 15, 2021, Singapore

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

training, where an earlier version of the model continues to train on newly available data, has become a popular idea.

In practice, however, incremental training of a neural net presents an array of challenges. For instance, it is critical that one stops the base model before it starts to overfit on test dataset (Figure 1), which is common for deep models. A common approach is to monitor some core metric (such as ROC AUC [2]) on a holdout validation dataset, and pick the best model checkpoint accordingly.

Catastrophic forgetting [5] is another common issue with incremental learning, whereby the model loses performance on old data after learning from new data. In e-commmerce search and recommendation, this can manifest itself in data distribution shift due to seasonality or other cyclic factors. Brain-inspired techniques such as regularization [8] and compressed data replay [7] have been successful in overcoming this issue. Other neural network specific remedies such as selective weight adjustment [14] and class rebalancing [3] have also been proposed. The last two techniques are closely related to our proposed methods.

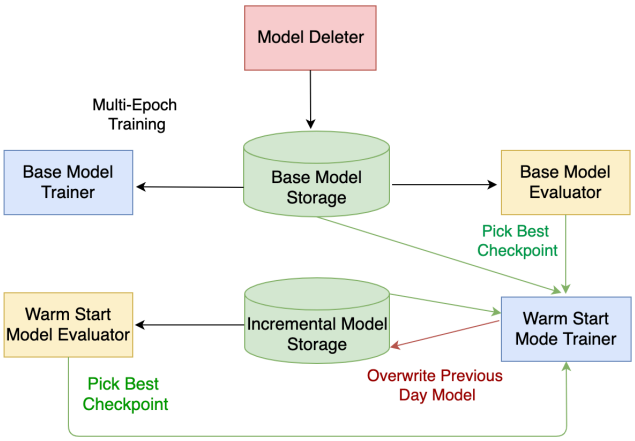


Figure 1: Incremental Training Pipeline. Best model checkpoints are selected according to evaluator metric.

2.2 Asynchronous Distributed Training

While incremental learning reduces the training time on newly acquired data, the training of a high quality base model can still be time-consuming. To address this, various distributed training strategies are often adopted, which can be roughly divided into two camps: synchronous and asynchronous. While deterministic synchronous training (DST) with powerful specialized hardware advances have taken a lead in many domains [4], asynchronous training (AST) still plays an important role in large scale deep learning, especially when the prevailing compute resource comes in the form of heterogeneous CPU clusters.

In a typical industrial setting, CPU-powered machines come in many varieties. Training strategies such as those based on parameter servers are highly tolerant to such device heterogeneity, while producing close to linear speed up in training time.

A popular AST strategy, first proposed in the early 2000 [12], spawns a set of processes called parameter servers to store a centralized copy of all the model parameters. The remaining worker processes run the actual training computation, while exchanging data with the parameter server asynchronously. While this has been empirically demonstrated to work similarly to DST in a variety of settings [6], we show that warm-starting from an asynchronously trained base model can result in sharp validation metrics drop.

Much effort has been devoted to studying performance of AST under various parameter settings [6]. For instance, [1] shows that both larger batch size and more workers cause performance degradation. One obvious reason for lower accuracy under AST, especially when the number of workers is large, is the so-called stale gradient problem: the gradient applied to the centralized weight update may be a few steps behind the latest. [13] shows that careful tuning of learning rate and batch size can mitigate the staleness problem. Other works such as [10], [15], and [11] introduce staleness-aware gradient update algorithms to address this problem more systematically, at the expense of higher memory usage, needed to store the staleness for each weight.

3 ISSUES WITH INCREMENTAL TRAINING

We take 30 days of search log $D = D(30)$ as the basic training data. Our main neural network is an improved version of DIN [16] (attention applied to user historical interactions followed by an MLP), trained under parameter server strategy with 30 workers and lazy ADAM optimizer. Ideally a new model should be trained every day on the latest 30 days. Training a new model from scratch, however, takes too long (3-5 days). Thus it is a natural idea to warm-start the model M_{t+1} from the previous day 30 day model M_{t+0}. Figure 5 illustrates this moving window training schedule.

For optimal model generalization, we always keep the model at the optimal step s with the highest evaluation metric (Session AUC) on a holdout set $D_{t+2}(1)$, 2 days ahead of the training set, in anticipation of the T+1 incremental training data.

Table 1 shows how the number of workers during base model training affects the final converged test AUC; the case of single worker synchronous training takes too long and is thus omitted.

Table 1: Number of workers for base model training

Worker Count	Session AUC@Step	Time Cost
10	0.8684@0.714m	20h 7m
30	0.8677@0.865m	9h 49m

We now turn to the issue of warmstart fragility. The green **All Params Warmstart** curve in Figure 4 represents the naive strategy of initializing all weights in M_{t+1} from weights in M_{t+0} trained to convergence. Compared to all the other strategies, including training from scratch (**All Params Init**), we see a clear initial drop in test AUC, which hardly recovers for many subsequent steps.

Figure 3 shows a more extreme case of metric drop when the model is warm-started from the peak of an asynchronously trained base model. In this case we used the public Amazon review dataset. Sub-figure 3a shows the test AUC for the base model trained from scratch. The highest recorded AUC occurs at step 6.5m. Sub-figure 3b shows the same AUC metric evaluated on the same test dataset, for a model warm-started from the base model at the step 6.5m. The

sharp AUC drop in the latter is hard to miss, and to the best of our knowledge has not been studied at all in the literature.

To understand this anomaly, note first that at any given point of time, different workers may have different versions of the model parameters. In particular they can be different from the model weights saved to disk. Thus there is no guarantee that the re-initialized model $M_{t+1}(s)$ can exactly reproduce the worker states of $M_{t+0}(s)$. See Figure 2 for an illustration of the weight disagreement phenomenon during model checkpoint saving and warm-start. Thus we hypothesize that the weight disagreement among different workers help prevent over-fitting on holdout set, especially near the point of convergence. During warm-start, however, these weights are forced to coalesce, which could precipitate over-fitting.

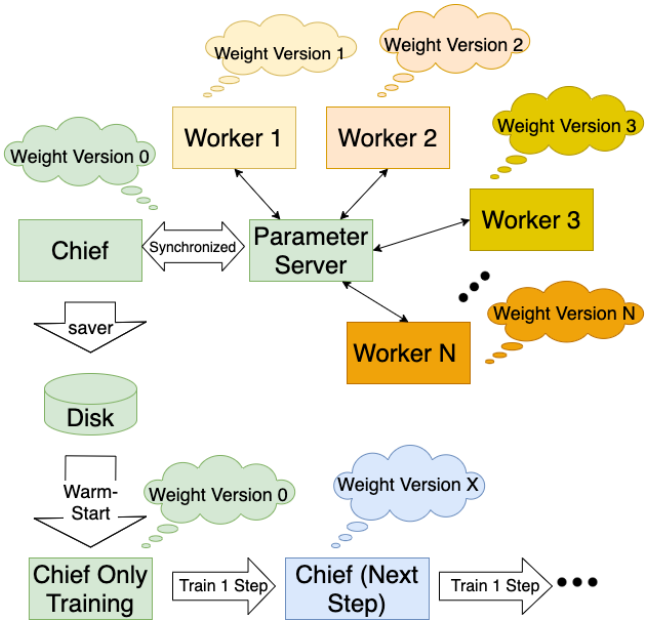


Figure 2: Inconsistent weights on different workers and parameter server

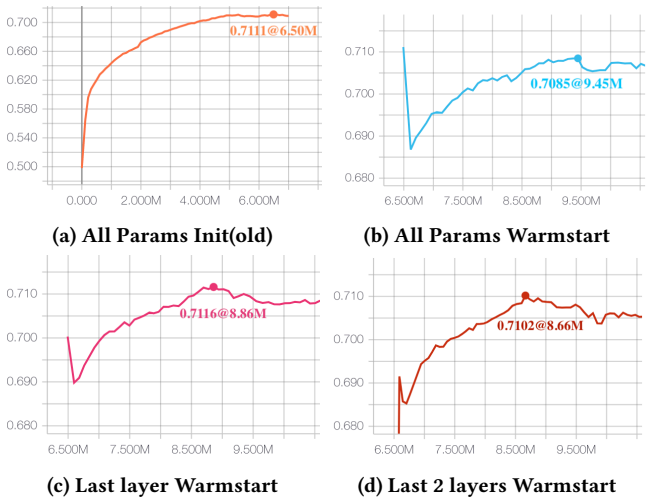
4 PROPOSED METHOD AND EXPERIMENTS

An obvious remedy to the problem of undesired weight synchronization during warm-start is to save not only the weights from the chief worker, but all the workers. This would require much more disk space and significantly reduce worker training efficiency.

Instead we take an indirect approach, by randomly re-initializing a small subset of the weights from $M_{t+0}(s)$, which has the immediate effect of reducing test AUC back to the starting level of $M_{t+0}(0)$, since the predictions are essentially random. Test AUC climbs back up very quickly (Figure 4), however, since most of the weights were already learned.

Table 2 compares different random re-initialization strategies during incremental warm-start. We record the highest session AUC on a holdout test dataset for each experiment, as well as the number of steps (since warm-starting) taken to reach the value.

- **All Params Init (old)** is essentially the base model without warm-start.



(a) All Params Init(old) **(b) All Params Warmstart**

(c) Last layer Warmstart **(d) Last 2 layers Warmstart**

Figure 3: Test AUC for Amazon Dataset Experiments

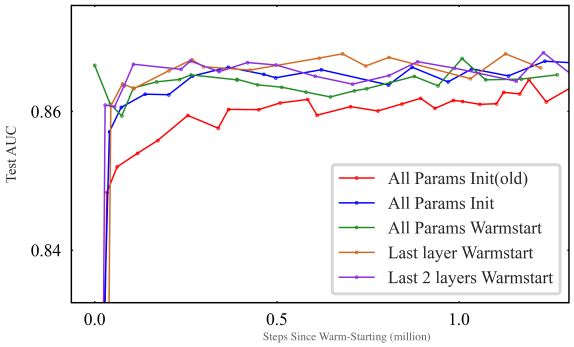


Figure 4: Warm-start parameter initialization strategies. The "All Params Init" curves are shifted horizontally to match the "Warmstart" curves' starting step at 0.

- **All Params Init** uses 29d of old data + 1d of new data.
- **All Params Warmstart** stands for naive warm-start where all parameters of the base model are preserved.
- Finally the last 2 settings, **Last layer Warmstart** and **Last 2 Layers Warmstart**, randomly re-initialize a subset of the MLP layers.

Generally, more randomly initialized layers lead to better test metrics but longer convergence steps. The best setting, randomly initializing the last MLP layer, cuts down training time to a quarter of non-incremental training. This has been confirmed on the public Amazon dataset experiments as well. As show in Figure 3c and 3d, randomly re-initializing the last 1 or 2 layers helps mitigate or eliminate the initial AUC drop, while achieving higher peak AUC than the base model (Figure 3a) or the naive strategy of warm-starting all weights (Figure 3b).

Table 3 shows the effect of mixing old data with the latest day of incremental training data. Data is shuffled uniformly after mixing. Though the base model has seen the old 30d data several times, adding more old data during warm-start gives it more time to recover from damage due to random re-initialization of the last layer.

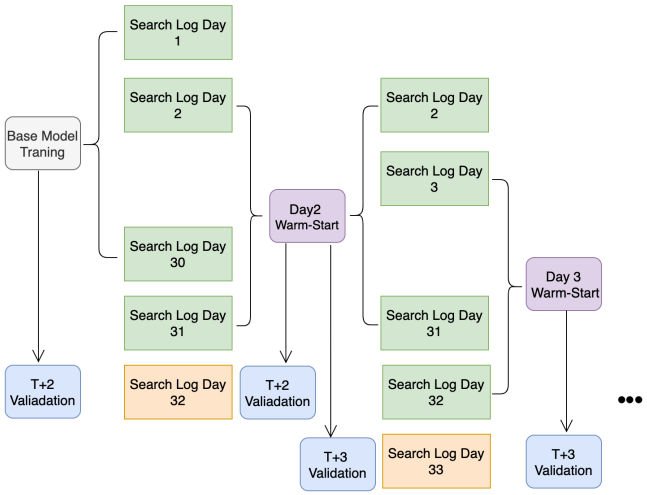


Figure 5: Daily Incremental Training Schedule

With the best combination of random initialization strategy and data mixture ratio, incremental learning achieves parity in session AUC with same day end-to-end trained model (**All Params Init**), or 0.45% gain over previous day model, at only a quarter of their training time.

Compared with training from scratch (**All Params Init** old and new), which takes more than 31 hours, incremental training with randomly initialized last layer only takes 8 hours, making daily model refresh possible. Indeed, it takes another 1 hour to perform offline evaluation, online validation, and model reload. As depicted in Figure 5, to decide whether to push the model to the online search system, we evaluate the performance of current day's model and previous model using the same test set at $T+2$ where T represents the last date of the base model's training set.

As shown in Table 4, over a period of 7 days of online A/B experiments, user conversion rate (UCVR) and sales proceeds (GMV) are both significantly improved over the base static model, thanks to the freshness of incremental training data. Here UCVR refers to the number of purchases made per user. GMV stands for Gross Merchandise Value, which is the total sales revenue among the users in the experimental bucket. Both of these online metrics are computed on a daily basis and averaged over the duration of the experiment. Even 0.1% relative improvement in these metrics translates directly to tens of millions of dollars in revenue, once deployed to full production traffic.

Table 2: Random re-initialization strategy for warm-start

Init Strategy	Session AUC@Step	Time Cost
All Params Init (old)	0.8639@2.936m	32h 26m
All Params Init	0.8683@2.848m	31h 8m
All Params Warmstart	0.8676@0.998m	10h 58m
last layer Warmstart	**0.8684@0.714m**	**8h 7m**
last 2 layers Warmstart	0.8684@1.231m	14h 23m

Table 3: Different mixture of training data

Model	Training Data	Session AUC@Step	Time Cost
random last layer (DIN)	**new 1d+old 29d**	**0.8684@714.4k**	**8h 7m**
	new 1d+old 6d	0.8652@127.9k	1h 47m
	new 1d+old 4d	0.8659@208.5k	2h 21m
	new 1d+old 2d	0.8641@335.8k	3h 54m

Table 4: Gains in online AB test

Metrics	Relative Gains	p value
GMV	0.59%	8.41e-2
UCVR	0.43%	1.51e-2

5 CONCLUSION

We make the novel observation that warm-starting from an asynchronously trained base model suffers from severe initial drop in test metrics. We present some plausible explanation for this strange phenomenon and provide two simple remedies to this practically important situation. The random partial weight re-initialization and data mixing strategies mitigate or eliminate the initial drop and help reduce training time to a quarter of conventional model refresh, while maintaining or exceeding earlier evaluation metrics, satisfying the requirement for daily model update.

REFERENCES

[1] O. Bhardwaj and G. Cong. Inefficiency of stochastic gradient descent with larger mini-batches (and more learners). 2016.
[2] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.
[3] F. M. Castro, M. J. Marin-Jimenez, N. Guil, C. Schmid, and K. Alahari. End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.
[4] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.
[5] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.
[6] S. Gupta, W. Zhang, and F. Wang. Model accuracy and runtime tradeoff in distributed deep learning: A systematic study. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 171–180, 2016.
[7] T. L. Hayes, K. Kafle, R. Shrestha, M. Acharya, and C. Kanan. Remind your neural network to prevent catastrophic forgetting. In *European Conference on Computer Vision*, pages 466–483. Springer, 2020.
[8] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.
[9] V. Losing, B. Hammer, and H. Wersing. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, 275:1261–1274, 2018.
[10] H. B. McMahan and M. Streeter. Delay-tolerant algorithms for asynchronous distributed online learning. 2014.
[11] A. Odena. Faster asynchronous sgd. *arXiv preprint arXiv:1601.04033*, 2016.
[12] A. Smola and S. Narayanamurthy. An architecture for parallel topic models. *Proceedings of the VLDB Endowment*, 3(1-2):703–710, 2010.
[13] A. Srinivasan, A. Jain, and P. Barekatain. An analysis of the delayed gradients problem in asynchronous sgd. 2018.
[14] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu. Large scale incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
[15] W. Zhang, S. Gupta, X. Lian, and J. Liu. Staleness-aware async-sgd for distributed deep learning. *arXiv preprint arXiv:1511.05950*, 2015.
[16] G. Zhou, X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, and K. Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1059–1068, 2018.