XDL: An Industrial Deep Learning Framework for High-dimensional Sparse Data

Biye Jiang, Chao Deng, Huimin Yi, Zelin Hu, Guorui Zhou, Yang Zheng, Sui Huang, Xinyang Guo, Dongyue Wang, Yue Song, Liqin Zhao, Zhi Wang, Peng Sun, Yu Zhang, Di Zhang, Jinhui Li, Jian Xu, Xiaoqiang Zhu, Kun Gai[*]
Alibaba Inc.
Beijing, China

ABSTRACT

With the rapid growth of data and computing power, deep learning based approaches have become the main solution for many artificial intelligence problems such as image classification, speech recognition and computer vision. Several excellent deep learning (DL) frameworks including Tensorflow, MxNet and PyTorch have been made open-sourced, further accelerating the advance of the community. However, existing DL frameworks are not designed for applications involving high-dimensional sparse data, which exists widely in many successful online businesses such as search engine, recommender systems and online advertising. In these industrial scenarios, deep models are typically trained on large scale datasets with up to billions of sparse features and hundreds of billions of samples, bringing great challenges to DL framework.

In this paper, we introduce a high-performance, large-scale and distributed DL framework named **XDL** which provides an elegant solution to fill the gap between general design of existing DL frameworks and industrial requirements arising from high-dimensional sparse data. Since 2016, XDL has been successfully deployed in Alibaba, serving many productions such as online advertising and recommender system. Running on hundreds of GPU cards in parallel, XDL can train deep models with tens of billions parameters within only several hours. Besides its excellent performance and flexibility, XDL is also friendly to developers. Algorithm scientists in Alibaba can develop and deploy new deep models with only several lines of simple codes. The XDL API and a reference implementation were released as an open-source package under the Apache 2.0 license in December, 2018 and are available at https://github.com/alibaba/x-deeplearning.

KEYWORDS

Deep learning, High-dimension sparse data, XDL

ACM Reference Format:
Biye Jiang, Chao Deng, Huimin Yi, Zelin Hu, Guorui Zhou, Yang Zheng, Sui Huang, Xinyang Guo, Dongyue Wang, Yue Song, Liqin Zhao, Zhi Wang, Peng Sun, Yu Zhang, Di Zhang, Jinhui Li, Jian Xu, Xiaoqiang Zhu, Kun Gai. 2019. XDL: An Industrial Deep Learning Framework for High-dimensional Sparse Data. In *1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data (DLP-KDD'19), August 5, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 9 pages.

1 INTRODUCTION

In the last decade, as one of the most exciting and powerful branches of artificial intelligence, deep learning has generated important breakthroughs in many areas such as speech recognition, computer vision, natural language processing and medical diagnosis. Thanks to the astronomical amount of data generated in many real-world applications, the unprecedented computing power brought by infrastructures such as graphics processing units (GPUs), as well as the sophisticated open-source deep learning frameworks such as TensorFlow [1], MxNet [6], Caffe [18], etc., deep learning technologies have been extensively applied to solve real-world problems.

Although existing deep learning frameworks have achieved great successes in many areas, they are not friendly designed for application involving high-dimensional sparse data. High-dimensional sparse data widely exists in many internet-scale applications such as search engine, recommender systems and online advertising. For example, in our display advertising system, we have petabytes (PB) of log data of user behavior generated everyday. Training samples extracted from this data contain billions of features, while only a few of these dimensions are non-zero for each sample.

Different from applications such as speech recognition, computer vision, and natural language processing where data is *dense*[1], these internet-scale online businesses, where data is *sparse*, pose unique challenges to the deep learning frameworks. First, the problem scale is so large that it cannot be solved on a single node. Parallelization is needed to solve problems of this scale. Second, the data is extremely sparse. Sparsity could cause low efficiency if not handled properly. Third, this data is not nicely formulated. There could be repeated features between samples, causing high pressure on bandwidth. Moreover, in rapid model evolutions, features need to be added or deleted in order to test their relevance to the goal. Existing deep learning frameworks are not friendly designed to resolve these challenges. Such gap makes it difficult for these businesses to fully benefit from the flourishing deep learning technologies.

With the abstraction of our hands-on experience in building deep learning solutions for our online advertising systems, deep models with high-dimensional sparse data often include two parts: *sparse feature learning* and *dense model learning*. Sparse feature learning is

[*] All authors contributed equally to this research.

[1] most dimensions of input feature vectors for these problems are non-zero

DLP-KDD'19 , August 5, 2019, Anchorage, AK, USA

the process to convert massive high-dimensional sparse data into dense features while dense model learning takes care of learning optimal model structures based on the dense features. These two parts also need to be smartly connected so that these two learning tasks can interact with each other to achieve best outcomes.

In this paper, we introduce XDL, which is a high-performance, large-scale, and distributed deep learning framework designed for learning tasks involving high-dimensional sparse data. In the sparse feature learning part, XDL offers a sophisticatedly designed distributed system, with deep optimizations on I/O, data pipeline, communication and GPUs, that provides extreme efficiency and scalability. Users can use embedding dictionary or deep models like CNN/RNN to map sparse item features or image/text features into dense representations. In the dense model learning part, XDL seamlessly adopts any open-source deep learning framework as its backend with a brand new technology named *bridging technology*. With this design, XDL is able to accelerate training on internet-scale learning problems and serve in end-to-end production systems. When evaluating XDL on real-world datasets, we found that it could run at least 5 times faster than the native distributed version of Tensorflow/MxNet.

Since 2016, XDL has been deployed in a series of core businesses in our company such as e-commerce recommendation and online advertising. Running on hundreds of servers, XDL trains a number of models with tens of billions parameters within only several hours. Besides its performance and flexibility, XDL is dedicated to hide complicated engineering details from users. Using XDL, our algorithm scientists can develop and deploy new models with only several lines of simple codes. Such system design brings flexibility and unlocks a number of algorithm innovations.

2 RELATED WORK

As the problem scale and complexity are increasing, we are facing more severe challenges for performance and scalability of deep learning frameworks. Traditional solutions such as TensorFlow [1], MxNet [6] and Caffe [18] usually run on single machine and provide limited support for distributed training. Even with multiple GPUs, those single machine solutions can hardly handle the model training with 10^{11} samples and 10^{10} parameters. Therefore, distributed training platform seems a promising solution, and many scholars have made great contributions in this area.

MPI [14] defines a set of interfaces for communication and coordination between machines. Due to its compatibility and scalability, MPI is widely used in parallel computing on distributed memory applications. A lot of machine learning frameworks are built on MPI. Chen et al proposed RABIT [5], an AllReduce library, improving OpenMPI with additional fault-tolerant property. Adam Coates et al proposed COTS HPC technology [3, 9]. They built a cluster of GPU servers with MPI on InfiniBand interconnections. With only 16 machines, this system can efficiently train model with 11 billion parameters. However, the data preprocessing part of these frameworks are oversimplified. Therefore they neither can take full advantage of data characteristics, nor can handle the massive-scale jobs for sparse data. Moreover, fault tolerance is not well-supported in MPI-based systems, thus resulting the availability problem in some industrial scenarios.

Another widely used paradigm is the Parameter Server (PS) which could use key-value store to handle sparse data properly. Dean et al [11] first introduced the PS and used downpour SGD to train large scale deep networks at Google. Petuum [30] introduces bounded-delay model into PS architecture [23], enabling asynchronous computation. Parameter sever is also widely used in industry. DSSTNE [2] is a DL framework designed by Amazon, specifically for large sparse dataset. It provides extremely efficient automatic model-parallel multi-GPU support and 100% Deterministic Execution. Model parallelism and data parallelism are supported at same time. Kunpeng [34] is a PS based distributed learning system at Alibaba, supporting models like sparse Logistic Regression and Multiple Additive Regression Trees.

Those PS based frameworks are usually designed to support traditional models like Logistic Regression or Convolutional neural network, which contains only the sparse part or dense part. It would be difficult for these frameworks to handle models involving both large scale sparse features and customized deep networks, as described in [12, 31, 32].

On the other hand, natural language processing systems such as machine translation [29] and speech recognition [3] use similar SparseNet + DenseNet model structures. However, they usually only have up to millions of words while it is common for us to have billions of item IDs or images as features.

Another active research area is to automatically find optimal run-time configurations for distributed model training. [27] uses reinforcement learning to find optimal device placement. [19, 20] searches optimal configuration for mixed parallelism strategy based on Legion [4], which is a high performance programming system for heterogenous and parallel machines. However, none of those algorithms take sparseness into account or provide solutions to problems with such a scale.

These related works provide lots of insights on building efficient model training systems. By learning their strength and weakness, we develop XDL for supporting industrial problems with high-dimensional sparse data.

3 ARCHITECTURE OF XDL

In this section, we introduce the architecture and the design philosophy of XDL. Before the discussion, it is necessary to describe the motivation of our design.

3.1 Network Characteristic of Deep Models with High-dimensional Sparse Data

In recent years, deep learning based approaches have successfully revolutionized algorithms applied in online businesses and achieved state-of-the-art performance. As a pioneer work, DSSM [17] proposes to model the relevance of query and document with a deep network in the scenario of web search engine. The core architecture of DSSM model follows an Embedding & MLP paradigm: first mapping high-dimensional sparse inputs into low-dimensional embedding space and then fitting the label with multi-layer perceptron (MLP). This Embedding & MLP architecture motivates most of following work to design deep models with high-dimensional sparse input from various applications such as video recommendation, web search and advertising in e-commerce site. Representative networks include Wide & Deep Learning [8], Youtube deep models [10], DeepFM [15], Deep Interest Network (DIN) [32, 33] and CrossMedia [12] etc. A thorough survey about deep models for recommendation systems can be found in [31].

From an algorithmic perspective, Embedding & MLP based network abstracts a family of these industrial models, which typically breaks the learning from sparse data into two steps: i) Representation learning which captures information from high-dimensional sparse input and embeds them into a low-dimensional space, ii) Function fitting which models the relationship between dense embedding representation and supervised label. For simplicity, we refer to network of the first step as **SparseNet** and the second one as **DenseNet** in the rest of this paper. Figure 1 illustrates this kind of abstraction.

In the scenario of industrial applications, new deep models should evolve rapidly to improve the business profit. Hence algorithmic scientists expect the training system to be simple-to-understand and easy-to-use: new models should be designed in script codes in a stand-alone perspective and run in parallel, leaving complex details of distributed training hidden in background. Motivated by these considerations, we design XDL, an industrial deep learning framework aiming to build high-performance system for training deep models with high-dimensional sparse data. We will show how we tackle the challenges arising from high-dimensional sparse data and propose an elegant design which follows the *bridging* strategy.

3.2 Design Philosophy of XDL and Bridging Methodology

From the above, we can see that **SparseNet** and **DenseNet** in the models with high-dimensional sparse data require different system functionalities. **DenseNet** is made up of several dense layers and requires high computation density on local machine. Several DL frameworks including Tensorflow, MxNet and Pytorch have been made open-sourced and such kind of dense networks can be handled well in these frameworks. However, **SparseNet** contains hundreds of billions of features which are generated from the raw samples. Therefore, a successful DL framework that can handle this scenario must have excellent distribution characteristic and enough computational capability of sparse data. The existing DL frameworks mentioned above are not well-designed for networks involving high-dimension sparse data.

For better understanding the challenges of training deep models with high-dimensional sparse data, let us look in detail of the scales of SparseNet and DenseNet. The scale of our dataset is shown at first. Table 1 shows typical volume of production data used in our display advertising system for one day.

#Feature	#sample per day	avg.#ID/sample	Disk space
1 Billion	1.5 Billion	5000	17TB

Table 1: typical problem scale

Table 2 shows network parameters statistic of some models used in our daily tasks. Obviously, it is the SparseNet that contributes most of difficulties for these models. DenseNet follows the traditional setting that existing deep learning frameworks hold. Besides, SparseNet needs to handle the practical I/O issue of input data with terabytes volume as well as complex and heavy parallelism issue. These challenges make the training of SparseNet to be quite different and critical from that of DenseNet.

Models (SparseNet)	Input Dimension	Network Parameter	Output Dimension
MLP	1 Billion	18 Billion	1440
DIN	1 Billion	18 Billion	33438
CrossMedia	250 Million	5 Billion	1464
Models (DenseNet)	**Input Dimension**	**Network Parameter**	**Output Dimension**
MLP	1440	1.2 Million	1
DIN	33438	1.7 Million	1
CrossMedia	1464	1 Million	1

Table 2: Network statistic for Multiple Layer Perception, Deep Interest Network [32] and Crossmedia [12] models.

Above all, we designed XDL whose architecture is shown in Figure 2. Follow the divide-and-conquer strategy, we propose a brand-new bridging architecture, in which the training system is bridged with two main sub-systems:

- **Advanced Model Server (AMS)**. AMS offers a sophisticatedly designed distributed system that provides extreme efficiency and scalability for training SparseNet. AMS handles rapidly evolving representation learning algorithms with large scale sparse inputs. Both embedding dictionary and models like CNN/RNN are supported to map large sparse inputs into dense vectors.
- **Backend Worker (BW)**. BW follows a common deep learning setting to learn from low-dimensional inputs and allows adopting any open-source DL frameworks as its backend. With such flexibility, our algorithmic scientists can easily try out new model structures involving DNN [17], attention mechanism [32] or Gated Recurrent Units [33] to model the complicated and evolving user behavior.

In the forward pass, the backed workers first read data from I/O module, and then send feature ID requests to the AMS. The AMS will compute the embedding vectors according to the IDs and send them back to the backend workers. The workers will also receive the updated dense net model parameters from last iteration. After collecting all the dense input vectors and the model parameters, the workers will perform sum-pooling on the input vectors and feed the result into the dense net. In backward pass, the data flow is reversed and the gradients are computed by the workers. AMS will then collect the gradients for both dense net and the sparse net. Parameter updates will be performed at the server side using solvers like Adam [22] or Momentum SGD. More details about the implementation and optimization of AMS modules will be described in next section.

XDL has high performance and good scalability, so it is capable of supporting industrial productions. Starting in 2016, XDL is used in training CTR prediction model in target advertising department. Running on hundreds of machines, XDL trains a family of

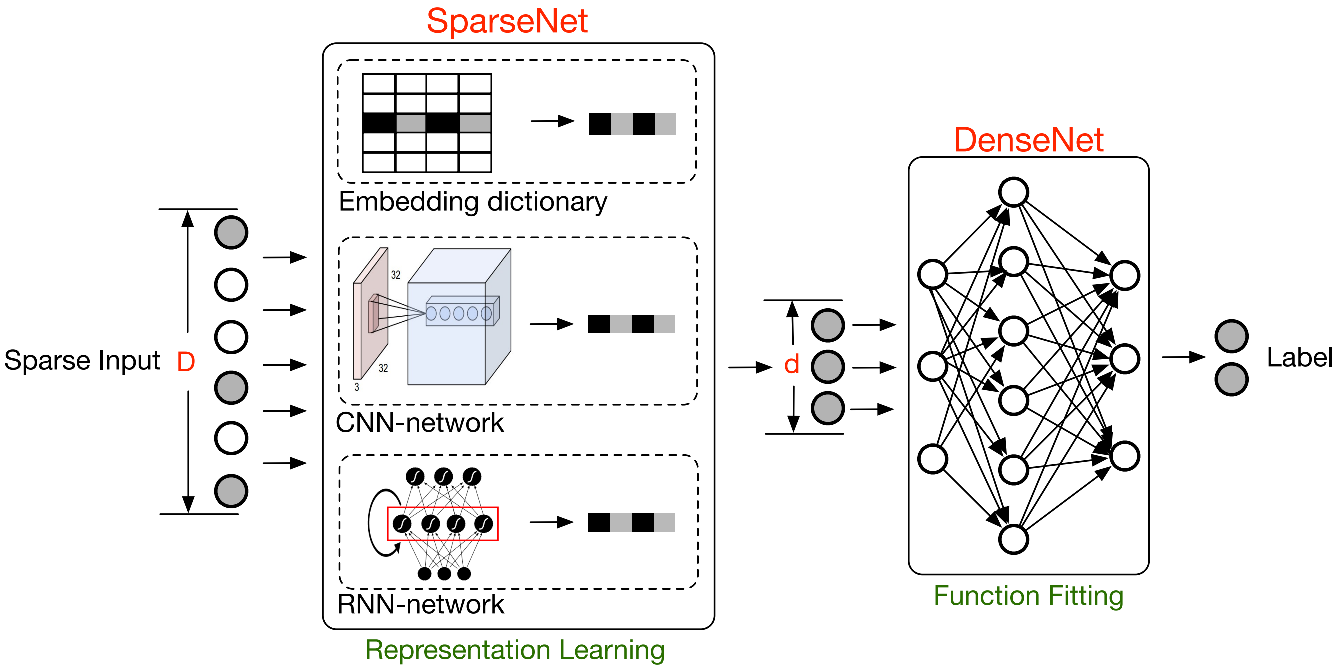


Figure 1: Abstraction of Embedding & MLP network architecture, which mostly consists of SparseNet and DenseNet. SparseNet maps original sparse input (with dimensionality of D where D scales up to billions) into low-dimensional representation (with dimensionality of d which traditionally scales up to thousands). DenseNet learns the function that fits data.

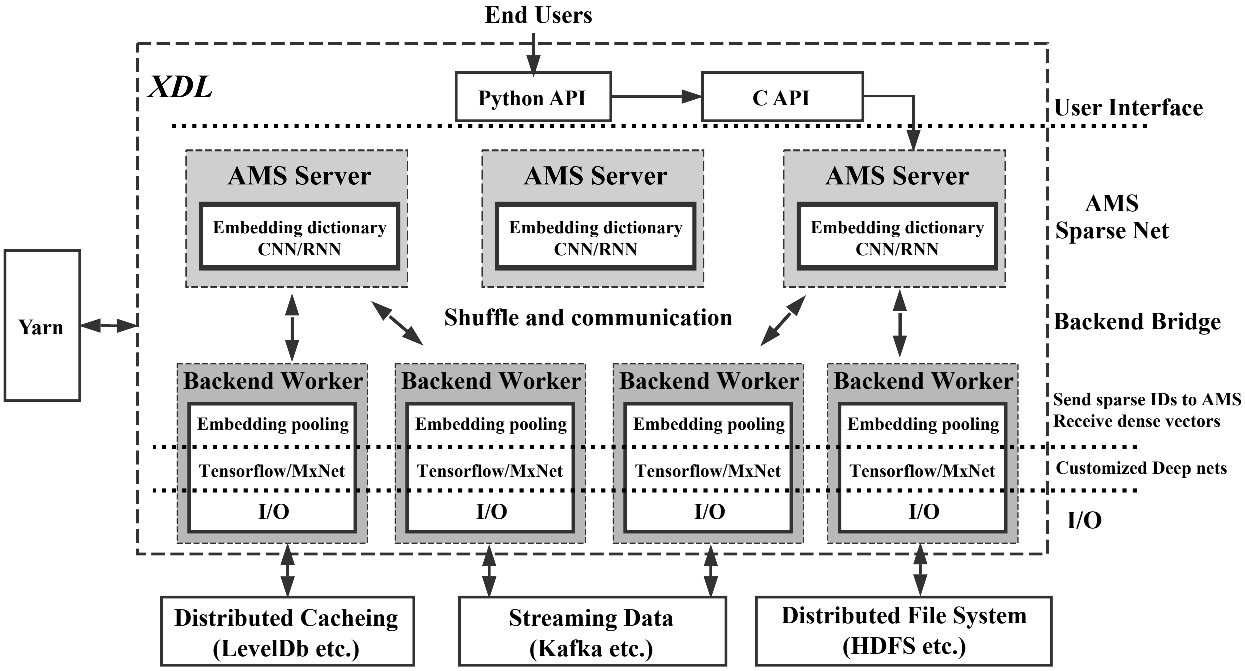


Figure 2: Architecture of XDL. The server and worker don't need to have the same rank. And they could be deployed on the same physical node to improve communication efficiency.

billion-parameter DNN models, supporting businesses in hundreds of scenarios.

In order to facilitate deployment on various computing platforms, XDL can be scheduled by multiple resource management platform, like Yarn, and provides data I/O interfaces to various data storage systems, like HDFS and Kafka. Good compatibility of XDL enables rapid upgrades in business applications.

3.3 Advanced Model Server

Learning the architecture design from Parameter server, we designed Advanced Model Server (AMS), which is a distributed system managing the training of SparseNet and DenseNet. AMS is responsible for storing and updating all the parameters, including SparseNet and DenseNet. Therefore, parameter placement becomes a critical challenge for a high-performance design. With consideration of stability and availability of online business, fault tolerant should also be well concerned.

Parameter Placement

As both sparse parameters and dense parameters are stored on AMS, parameter placement algorithm should handle the different characteristics of these two kinds of parameters. On one hand, sparse parameters have heavy memory usage, for the dimension of sparse features could be up to tens of billions. However, in each mini-batch, only a few feature IDs are requested by workers, thus indicating that sparse parameters have low I/O pressure. Moreover, the memory usage of sparse parameters is changeable along with the training process because the samples will bring new IDs into the sparse parameters continually. On the other hand, dense parameters require low memory usage but have heavy pressure on I/O because workers requested the whole dense parameters in each mini-batch.

According to the above analysis, our key-value storage uses hash-map as low-level data structure for sparse parameters in AMS, which also exists in traditional parameter servers. So we can divide the hash table bucket averagely on each server, relieving the memory storage pressure on servers. Unfortunately, this process will cause a huge increase of request numbers from workers. To address this issue, we union the worker requests on worker-side before the embedding dictionary lookup. This optimization gives us a big speedup in practice. As for dense parameters, the I/O pressure of each parameter will be computed and separated averagely on each server within the memory constraints.

Fault Tolerant

With the rapid growth of sample amount and model complexity, off-line training takes more and more time for a specific deep model. When some roles in the DL system fails, re-training from the very beginning is very expensive in view of time and resources. Moreover, in some online learning scenarios, stability and availability are very important. Therefore, fault tolerant is carefully designed in XDL. AMS is made up of a scheduler and several servers. Servers keep synchronized with scheduler using regular heart beat. During the training process, the snapshot of the whole models will be stored in a certain place. If any server fails for some reason, the scheduler will notice the abnormal state and set the whole AMS system to be not ready until the server is restarted by yarn or any other scheduling system. If AMS is ready, the scheduler will notice the servers to restore from the latest snapshot and continue the training process.

Meanwhile, as the workers are responsible for reading samples, the reading state of each worker is also stored in AMS as a special parameter. When workers fail and get restarted, their reading states can be recovered by pulling the parameter from AMS. With this effort, no samples are lost or read redundantly during the training process by workers.

Asynchronized Update

XDL supports both synchronized mode and asynchronized mode. For synchronized mode, in each iteration, the workers pull parameters and push updates exactly once. And each iteration won't finish until all AMS receive updates from all the workers and apply the averaging gradient on the parameters. In asynchronized mode, there is no such a concept of iteration. The workers operate independently. Therefore each worker can move on to the next step as soon as it push its updates to the AMS, without waiting for other workers to finish their updates. Also, in the server side, updates from the workers are applied in a lock-free style. We use this approach since updates for the embedding table is very sparse and conflict rarely happens. For the dense DNN, though each update may not be successfully applied on the entire model, there is always only one version of the model globally. Therefore the lock-free style update can be seen as adding a mask on the model to randomly drop part of the gradients. In practice, we found that asynchronized mode can achieve much bigger system throughput, with very little model performance lost.

4 SYSTEM IMPLEMENTATION AND OPTIMIZATION

4.1 I/O

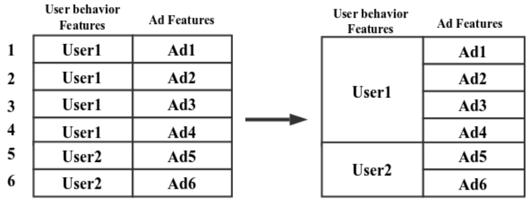
In production scenario, the inputs of our training system, such as samples and initial model, are usually stored in different locations. For example, some of the samples are streamed from an online queuing system, while others are retrieved from an offline distributed file system. The goal of I/O module design is to maximize the input bandwidth usage from data sources, such as HDFS, Kafka, etc.

It is well known that I/O bandwidth in distributed ML system has become one of the biggest bottlenecks. Since we use existing DL frameworks as computational backends, how to swallow data faster becomes a more critical problem. Therefore, we implement various optimizations to maximize the I/O throughput.

First, there exists a lot of repeated features between different samples. These repetitions come from the very beginning of our data source: user behavior. One user clicking on two different links will produce two different samples containing repeated features of the same user. Figure 3 illustrates the structure of repetitions. Compression could be done to optimize storage, communication and computation efficiency.

Hierarchical sample compression:

XDL takes full advantage of the fact that ID repetition exists between samples. In preprocessing phase, the raw samples are organized into multi-prefix trees, which drastically reduces the storage space and the communication cost. In training phase, a number of



	User behavior Features	Ad Features
1	User1	Ad1
2	User1	Ad2
3	User1	Ad3
4	User1	Ad4
5	User2	Ad5
6	User2	Ad6

User behavior Features	Ad Features
User1	Ad1
	Ad2
	Ad3
	Ad4
User2	Ad5
	Ad6

Figure 3: repetition

prefix trees are grouped into mini-batches of user-defined batch size. This compression also brings benefits for computation as it reduces duplicated embedding vector computation. If the total sample size of prefix trees exceed batch size, the last prefix tree will be split.

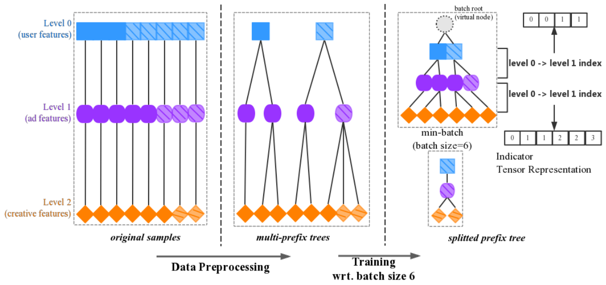


Figure 4: I/O data and computation compression. In the data-preprocessing stage, multi-prefix trees are produced for mini-batch generation of training stage.

An example of hierarchical sample compression is shown in figure 4. The raw samples are organized into two different 3-layer prefix trees. The first, second, and third layer of the trees represents user feature, ad feature and creative feature respectively. In training phase, the last prefix-tree is split with respect to batch size of 6. Two auxiliary tensors named indicator are constructed in order to indicate the relationship between adjacent layers.

4.2 Work Flow Pipeline

Work flow pipeline is a common technology to speed-up the program running process. Making the stages in the program overlapped in timeline with multi-thread technologies, considerable running time can be saved. There are also several kinds of work flow pipelines implemented in XDL.

In order to maximize the I/O throughput, we pipeline the process with threads. A complete training iteration is divided into three stages: 1) reading samples and group them into mini-batches 2) pre-fetching the parameter indexes in mini-batches from key-value store or input data for calculation on servers; 3) pull model parameters and do forward/backward propagation. Pipeline exploits thread-parallelism by overlapping the execution of the three stages.

As shown in Figure 5, jobs on these three stages are scheduled to three different thread pools respectively. And these three thread pools coordinate through a lock-free queue. If GPU is used, data in mini-batches and contexts are copied onto GPUs asynchronously using cudaMemcpyAsync() function in different streams.

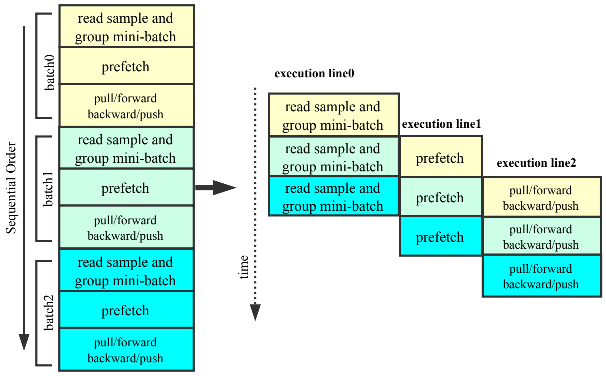

Sequential Order
batch0
read sample and group mini-batch
prefetch
pull/forward backward/push
batch1
read sample and group mini-batch
prefetch
pull/forward backward/push
batch2
read sample and group mini-batch
prefetch
pull/forward backward/push
execution line0
execution line1
execution line2
time
read sample and group mini-batch
read sample and group mini-batch
read sample and group mini-batch
prefetch
prefetch
prefetch
pull/forward backward/push
pull/forward backward/push
pull/forward backward/push

Figure 5: The training iteration consists of three stages, the pipelining is used to maximize the parallelism

Besides the pipeline in I/O implementation, a work flow pipeline between SparseNet and DenseNet is designed to further improve the performance of XDL. In this situation, as the input of DenseNet is the output of SparseNet, the result of SparseNet can be precomputed before the previous training process of DenseNet ends. Assuming that the training time of SparseNet and DenseNet are T_{sparse} and T_{dense} respectively, the total runtime will be reduced from $T_{sparse} + T_{dense}$ to $\max\{T_{sparse}, T_{dense}\}$. In this scenario, the efficiency of training process is significantly improved with only little lost of model performance because when the output of SparseNet is prefetched, there may exist some sparse IDs which are not updated. So this kind of pipeline is often adapted when there is few common sparse IDs between the neighbor mini-batches.

XDL also allows algorithm scientists to build pipelines that connect any stage in the training process with only a few lines of codes. A deep model can be divided into any number of stages. Algorithm scientists have variety of choices with consideration of performance and accuracy to build pipelines in the training process.

4.3 Optimization for Advanced Model Server

In order to improve the efficiency of AMS, optimizations are adopted in various aspects.

To speedup the K-V query in the embedding process, the power of GPU is utilized in XDL. GPU has two advantages. First, the memory bandwidth of GPU is higher (700GBps for Nvidia P100, 100GBps for Intel E5-2699 v4). Second, it runs in parallel on thousands of cores (3584cores for Nvidia P100). Therefore, we can implement much faster embedding dictionary lookup on GPU. The disadvantage of GPU is that GRAM size is limited. To handle this problem, only the index of embedding dictionary will be prefetched into GRAM.

As AMS plays a role of parameter server in most scenarios, there exists a huge amount of network communications between AMS and backend workers. The amount of network communications can be extremely huge when the degree of parallelism is high. In this situation, the run time consumed during network communication becomes a key part of the total run time. XDL adopts **Seastar**[28] as its communication library and many effective optimizations are made based on the Seastar framework. The technologies such as zero-copy and CPU-binding are involved to guarantee that network communication won't become the bottleneck of the whole training process.

In deep learning, batch size is a very important hyper-parameter. Some models show good performance when small batch size is used while some models prefer a large one. This requires that the DL frameworks should be capable of handling different sample batch sizes. As we mentioned before, CPU-binding is an important technology in AMS. Each thread in AMS is bounded with a CPU core so the thread switching time between different cores is saved. When the sample batch size is small, this CPU-binding mechanism enables high performance of computation on AMS. However, if the sample batch size is large, there will be too many IDs to be handled by one single thread. In this situation, AMS has a self-adaptive mechanism in which an extra thread pool is created for the purpose of dealing with the large number of IDs. With this optimization, AMS shows excellent performance on both small and large sample batch size.

4.4 Online Learning with XDL

Online learning methodology is widely used in industry recently for its ability to capture the changes of customer behaviors in real time. An effective online learning model is very valuable in many business scenarios. XDL provides a series of mechanisms for online leaning.

Feature Entry Filter

As we mentioned before, the memory usage of sparse parameters is changeable along with the training process because the samples will bring new IDs into the sparse parameters. Thus the model storage is continuously increasing on AMS. Therefore, the scale of IDs must be controlled within a certain level. To address this issue, XDL provides several feature entry filter mechanisms such as probability-based filter and counting-bloom filter. In this situation, the low-frequency IDs will be dropped before the training process as they contribute much less to the models. Meanwhile, the scale and the memory storage of the online models are well controlled, which guarantees the stability of the online system.

Incremental Model Export

With the continuous training process, when the model storage reaches hundreds of GBs or even higher, the full model export will become extremely expensive with consideration of time and computational resources. So XDL provides the incremental model export for the online training and inference systems. Each incremental model only requires a small amount of storage and is very easy to deployed on the online system.

Feature Expire

With the same purpose with **Feature Entry Filter**, feature expire mechanism also controls of the size of model. During the online training process, the features which are not updated for a long time would become useless and could be deleted. XDL allows algorithm experts to write their own feature expire functions and customize the feature expire plan.

4.5 User Interface

XDL provides python APIs for algorithm scientists to develop new deep learning models. These APIs let users focus on model itself without concerning about I/O efficiency, distribution strategy, system consistency and low-level communication. Coding and debugging on a parallelized model could be almost the same as on a serial one. With details on parallelization and optimization being hidden, it becomes very easy for users to deploy newly developed models onto massive GPU clusters in sandbox and production environments. Most of the time, scientists only need to modify the dense net structure to test new ideas. Users have full control on backend workers by running user-defined codes. It is also very convenient to modify the data flow and the sparse net components to try out more aggressive algorithm innovations.

5 XDL ECOSYSTEM

Besides XDL, Alibaba also open-sourced **XDL Algorithm Solution** and **Blaze Inference Engine** as follows, which comprise an ecosystem together with XDL framework. All these XDL-based tools are available at https://github.com/alibaba/x-deeplearning.

XDL Algorithm Solution includes several effective models which are successfully adopted in the advertising business of Alibaba, such as Deep Interest Network (DIN) [32], Deep Interest Evolution Network (DIEN) [33], Tree-based Deep Match (TDM) [35], etc.

Blaze Inference Engine is a high-performance inference engine that enables massive real-time computation for online advertising business, leveraging techniques like kernel fusion and low-precision inference.

XDL ecosystem sets up an example and a guideline for advertising industry. More and more companies have adopted XDL as their off-line or online deep learning tools.

6 EVALUATION

In this section, we provide detailed evaluation of the optimization strategies described in previous section. The experiments are conducted on nodes interconnected with 25Gbps ethernet. Each node equips with 2 Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz (96 cores) and 512GB RAM. In compiling, we use GCC 4.8.5 with flag -O2 -fopenmp -mavx2.

In our practice, when choosing the backend, MxNet or TensorFlow performs almost the same as they have equivalent performance for dense net computation. In the following evaluations, we choose Tensorflow as XDL's computation backend.

6.1 Dataset

In our experiment, we train a DNN model for Click Through Rate (CTR) prediction problem using an open dataset published by [25]. This dataset is sampled with 1% ratio from Taobao's e-commerce recommending system. Statistics of this dataset is shown in Table 3.

The open dataset includes three aspects: user's interest, Ad's characteristic and context. Each of them is represented by high-dimensional sparse IDs. Using XDL's python interface, we build the

Statistics	#Users	#Items	#Pageviews	#Clicks
Counts	0.4M	4.3M	84M	3.4M

Table 3: Benchmark dataset statistics (M = Millions)

model based on the SparseNet + DenseNet abstraction. In SparseNet, we use embedding dictionary to turn each ID into a dense vector with dimension of 18. It follows by a grouped sum-pooling to reduce these vectors into 23 vectors. In DenseNet, we use a 3-layer fully-connected DNN, which contains about 100K parameters. (In this DNN, There are 23*18 nodes in input layer, 200 nodes in first layer, 80 nodes in second layer, and 1 node in output layer.) The model is trained using the clicked label to estimate the CTR.

6.2 Subsystem Evaluation

In order to understand the performance of XDL, we will evaluate the optimization strategies one by one to see how much benefit we can get from each of them. The following experiments run in the same environment using the open dataset described above. All the experiments run on 200 workers (except scalability experiment) with 80 AMS in CPU mode. We allocate 8 CPU cores for each worker, and each AMS will occupy all 96 CPU cores in one machine. The XDL runs in asynchronized mode and the updates are lock-free, as described in Section 4.3

Hierarchical sample compression:

The hierarchical sample compression in I/O module brings great speedup to both communication and computation. As on averaged, each user id can repeat 57 times in the open dataset, it means the compression ratio for the common features shared by those users could be 57. In order to evaluate how much performance gain we can obtain by using sample compression, we resample the dataset to construct new datasets with different averaged compression ratio. We then compare the XDL throughput when running on those datasets. The results can be seen in Figure 6.

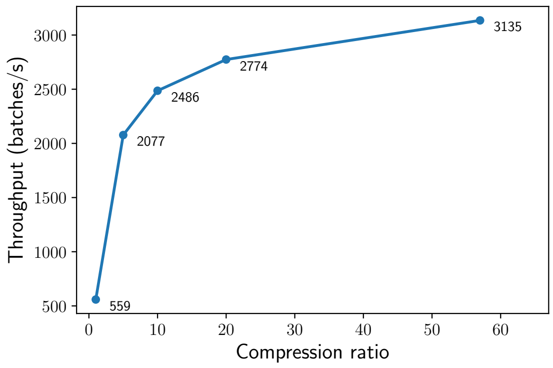


Figure 6: Compression ratio v.s. Throughput. Runs on 200 workers, batch size is 5000.

Large batch size:

Using large batch size will have fewer iterations and therefore it can drastically reduce the overhead of data-copy and communication. Even though Keskar et al [21] found that training with large-batch can result in sharp minima and hurt generalization. Several techniques [13, 16] have been proposed to mitigate the problem. We therefore tend to use larger batch size to improve the throughput. The throughput of AMS under different batch size is shown in Figure 7.

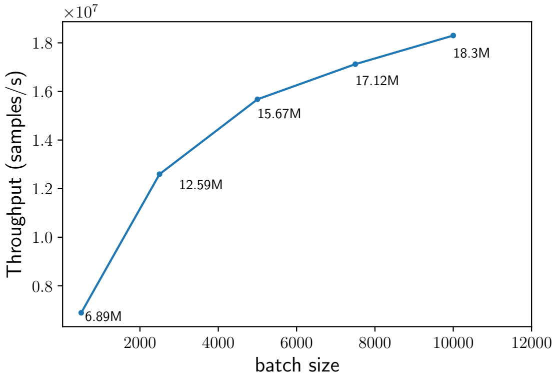


Figure 7: Batch size v.s. Throughput. Compression ratio = 57, using 200 workers.

Scalability:

We also test the scalability of XDL and compare it to native TensorFlow on various number of workers, with 80 AMS. Again, each worker uses 8 CPU cores and each AMS uses 96 CPU cores. Both XDL and Tensorflow use lock-free style asynchronized update. Results are shown in Figure 8. XDL consistently preforms better due to its optimizations for communication and repeated features. As we fix the number of AMS, we achieve almost linear scalability when running in the asynchronized mode.

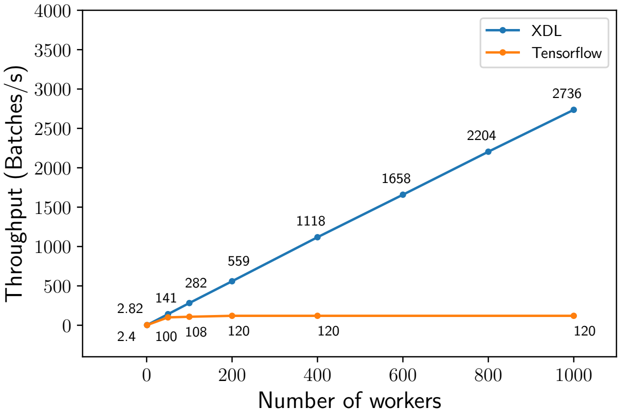


Figure 8: Number of workers v.s. Throughput. Batch size = 5000, compression ratio = 57, using 80 nodes as AMS.

7 CONCLUSION

We have introduced a deep learning framework named XDL, which is very powerful for high-dimensional sparse dataset with the SparseNet + DenseNet abstraction. With high-dimension oriented design and system optimization, XDL can be much faster than native Tensorflow on real-world dataset. Our algorithmic scientists can easily develop new models using the flexible API of XDL to support the rapid evolving business in Alibaba.

With the open-source process of XDL ecosystem, XDL can be further developed in the future. First, advanced acceleration techniques like TVM [7], deep gradient compression [24] and mixed precision training [26] are being tested and will be deployed soon. Second, as an open solution for parameter storage and updating, AMS shall have more flexible data flow and model structure to meet the requirements from the newest models such as Memory Network and Dynamic Computation Graphs.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In OSDI, volume 16, pages 265–283, 2016.

[2] AMAZON. Dsstne, 2016. URL https://github.com/amzn/amazon-dsstne.

[3] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In International Conference on Machine Learning, pages 173–182, 2016.

[4] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. Legion: Expressing locality and independence with logical regions. In High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for, pages 1–11. IEEE, 2012.

[5] Tianqi Chen, Ignacio Cano, and Tianyi Zhou. Rabit: A reliable allreduce and broadcast interface. Transfer, 3:2, 2015.

[6] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274, 2015.

[7] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Haichen Shen, Eddie Yan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Tvm: End-to-end optimization stack for deep learning. arXiv preprint arXiv:1802.04799, 2018.

[8] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, pages 7–10. ACM, 2016.

[9] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with systems. In International Conference on Machine Learning, pages 1337–1345, 2013.

[10] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems, pages 191–198. ACM, 2016.

[11] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In Advances in neural information processing systems, pages 1223–1231, 2012.

[12] Tiezheng Ge, Liqin Zhao, Guorui Zhou, Keyu Chen, Shuying Liu, Huiming Yi, Zelin Hu, Bochao Liu, Peng Sun, Haoyu Liu, et al. Image matters: Jointly train advertising ctr model with image representation of ad and user behavior. arXiv preprint arXiv:1711.06505, 2017.

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

[14] Richard L Graham, Galen M Shipman, Brian W Barrett, Ralph H Castain, George Bosilca, and Andrew Lumsdaine. : A high-performance, heterogeneous mpi. In Cluster Computing, 2006 IEEE International Conference on, pages 1–9. IEEE, 2006.

[15] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for ctr prediction. arXiv preprint arXiv:1703.04247, 2017.

[16] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In Advances in Neural Information Processing Systems, pages 1731–1741, 2017.

[17] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In Proceedings of the 22nd ACM international conference on Conference on information & knowledge management, pages 2333–2338. ACM, 2013.

[18] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia, pages 675–678. ACM, 2014.

[19] Zhihao Jia, Sina Lin, Charles R Qi, and Alex Aiken. Exploring hidden dimensions in parallelizing convolutional neural networks. arXiv preprint arXiv:1802.04924, 2018.

[20] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. arXiv preprint arXiv:1807.05358, 2018.

[21] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836, 2016.

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[23] Mu Li. Scaling distributed machine learning with system and algorithm co-design. PhD thesis, Intel, 2017.

[24] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. arXiv preprint arXiv:1712.01887, 2017.

[25] Xiao Ma, Liqin Zhao, Guan Huang, Zhi Wang, Zelin Hu, Xiaoqiang Zhu, and Kun Gai. Entire space multi-task model: An effective approach for estimating post-click conversion rate. arXiv preprint arXiv:1804.07931, 2018.

[26] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaev, Ganesh Venkatesh, et al. Mixed precision training. arXiv preprint arXiv:1710.03740, 2017.

[27] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. arXiv preprint arXiv:1706.04972, 2017.

[28] Scylla. Seastar, 2015. URL https://github.com/scylladb/seastar.

[29] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144, 2016.

[30] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: A new platform for distributed machine learning on big data. IEEE Transactions on Big Data, 1 (2):49–67, 2015.

[31] Shuai Zhang, Lina Yao, and Aixin Sun. Deep learning based recommender system: A survey and new perspectives. arXiv preprint arXiv:1707.07435, 2017.

[32] Guorui Zhou, Chengru Song, Xiaoqiang Zhu, Xiao Ma, Yanghui Yan, Xingya Dai, Han Zhu, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. arXiv preprint arXiv:1706.06978, 2017.

[33] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. Deep interest evolution network for click-through rate prediction. arXiv preprint arXiv:1809.03672, 2018.

[34] Jun Zhou, Xiaolong Li, Peilin Zhao, Chaochao Chen, Longfei Li, Xinxing Yang, Qing Cui, Jin Yu, Xu Chen, Yi Ding, et al. Kunpeng: Parameter server based distributed learning systems and its applications in alibaba and ant financial. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1693–1702. ACM, 2017.

[35] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. Learning tree-based deep model for recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1079–1088. ACM, 2018.