A Practical Incremental Method to Train Deep CTR Models

Yichao Wang, Huifeng Guo, Ruiming Tang, Zhirong Liu, Xiuqiang He
Noah's Ark Lab, Huawei, China.
{wangyichao5,huifeng.guo,tangruiming,liuzhirong,hexiuqiang1}@huawei.com

ABSTRACT

Deep learning models in recommender systems are usually trained in the batch mode, namely iteratively trained on a fixed-size window of training data. Such batch mode training of deep learning models suffers from low training efficiency, which may lead to performance degradation when the model is not produced on time. To tackle this issue, incremental learning is proposed and has received much attention recently. Incremental learning has great potential in recommender systems, as two consecutive window of training data overlap most of the volume. It aims to update the model incrementally with only the newly incoming samples from the timestamp when the model is updated last time, which is much more efficient than the batch mode training. However, most of the incremental learning methods focus on the research area of image recognition where new tasks or classes are learned over time. In this work, we introduce a practical incremental method to train deep CTR models, which consists of three decoupled modules (namely, data, feature and model module). Our method can achieve comparable performance to the conventional batch mode training with much better training efficiency. We conduct extensive experiments on a public benchmark and a private dataset to demonstrate the effectiveness of our proposed method.

KEYWORDS

Incremental Learning, Deep Learning, Recommender System, Click-Through Rate Prediction

ACM Reference Format:
Yichao Wang, Huifeng Guo, Ruiming Tang, Zhirong Liu, Xiuqiang He. 2019. A Practical Incremental Method to Train Deep CTR Models. In *ACM Conference (KDD'20), Aug 22–27, 2020, San Diego, California.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3298689.3347033

1 INTRODUCTION

Internet users can access a huge number of online products and services, therefore it becomes difficult for users to identify what might interest them. To reduce information overload and to satisfy the diverse needs of users, personalized recommender systems are playing an important role in modern society. Accurate personalized recommender systems benefit both demand-side and supply-side including publisher and platform.

Click-Through Rate (CTR) prediction is to estimate the probability that a user will click on a recommended item under a specific context. It plays a crucial role in personalized recommender system, especially in the business of app store and online advertising. Nowadays, deep learning approaches have attracted more and more attention due to the superiority over prediction performance and automated feature exploration. Therefore, many industrial companies deploy deep CTR models in their recommender system, such as Wide & Deep [1] in Google Play, DeepFM [4] and PIN [12] in Huawei AppGallery, DIN [19] and DIEN [18] in Taobao, etc.

However, every coin has two sides. To achieve good performance, Deep CTR models with complicated architectures need to be trained on huge volume of training data for several epochs, therefore they all suffer from low training efficiency. Such low training efficiency (namely, long training time) may lead to performance degrade when the model is not produced on time. We observe such performance degradation when the model stops updating in app recommendation scenarios in Huawei AppGallery, as presented in Figure 1. For instance, if the model stops updating for 5 days, the model performance degrades 0.66% in terms of AUC, which would lead to significant loss of revenue and also user experience. Hence,

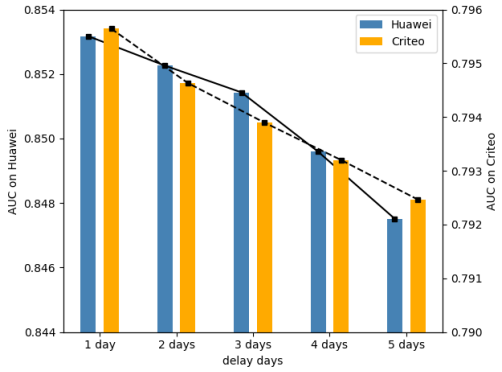

Figure 1: Model performance degrades when the model stops updating for different days. X-axis is different gaps between training set and test set.

as can be observed, how to improve training efficiency of Deep CTR models without hurting its performance is an essential problem in recommender system. Distributed learning and incremental learning are two common paradigms to tackle this problem from different perspectives. Distributed learning requires extra computational resources and distributes training data and the model to multiple nodes to accelerate training. On the other side, incremental learning changes the training procedure from batch mode to increment mode, which utilizes just the most recent data to update

KDD'20, Aug 22–27 2020, San Diego, California

ACM ISBN 978-1-4503-6243-6/19/09…$15.00
https://doi.org/10.1145/3298689.3347033

the current model. However, most deep models in industrial recommender system are trained in the batch mode, where a fixed-size window of training data (usually in a multi-billion scale) is used to train the model iteratively. In this work, we focus on devising incremental method to train deep CTR models, which aims to improve the training efficiency significantly without degrading the model performance.

However, to the best of our knowledge, most of the existing incremental learning methods mainly concentrate on image recognition field where new tasks or classes are learned over time. While incremental learning for deep CTR models faces different circumstances from image recognition, such as incoming new features, etc, therefore there is a need to look into this topic seriously. In this paper, we propose a practical incremental method *IncCTR* for deep CTR models. As presented in Figure 2, three decoupled modules are integrated in our model: *Data Module*, *Feature Module* and *Model Module*. Data module mimics the functionality of a reservoir, constructing training data from both historical data and incoming data. Feature module is designed to handle new features from incoming data and initialize both existing features and new features wisely. Model module employs knowledge distillation to fine-tune the model parameters, balancing learning knowledge from the previous model and from incoming data. More specifically, we look into two different choices for the teacher model.

The main contributions of this work are listed as follows:

- We highlight the necessity of incremental learning in recommender system through rigorous offline simulations. We propose a practical incremental method, IncCTR, to train deep CTR models.
- IncCTR consists of data module, feature module and model module, which have the functionality of constructing training data , handling new features and fine-tuning model parameters, respectively.
- We conduct extensive experiments on a public benchmark and a private industrial dataset from Huawei AppGallery to demonstrate that IncCTR is able to achieve comparable performance to the batch mode of training, with significant improvement on training efficiency. Moreover, ablation study of each module in IncCTR are performed.

The rest of the paper is organized as follows. In Section 2, we introduce some preliminaries for better understanding our method and application. We elaborate our incremental learning framework IncCTR and three individual modules in detail in Section 3. In section 4, results of comparison experiments and ablation studies are reported to verify the effectiveness of our proposed framework. Lastly, we draw a conclusion and discuss future work in section 5.

2 PRELIMINARIES

In this section, we introduce some notations, basic knowledge about deep CTR models. Also, the training modes (batch mode and incremental mode) are presented and compared.

2.1 Deep CTR Model

Recently, various deep CTR models are proposed, such as DeepFM [4], Wide & Deep [1], PIN [12], DIN [19], and DIEN [18]. Generally, deep CTR models include three parts: embedding layer, interaction layer, and prediction layer.

2.1.1 Embedding layer. In most CTR prediction tasks, data is collected in a multi-field categorical form [12, 14, 16]. Each data instance is transformed into a high-dimensional sparse (binary) vector via one-hot encoding [5]. For example, the raw data instance (Gender=Male, Height=185, Age=18, Name=Bob) can be represented as:

$$\underbrace{(0,1)}_{\text{Gender=Male}}\ \underbrace{(0,...,1,0,0)}_{\text{Height=185}}\underbrace{(0,1,...,0,0)}_{\text{Age=18}}\underbrace{(1,0,0,..,0)}_{\text{Name=Bob.}}$$

An embedding layer is applied to compress the raw features to low-dimensional vectors before feeding them into neural networks. For a univalent field, (e.g., "Gender=Male"), its embedding is the feature embedding; For a multivalent field (e.g., "Interest=Football, Basketball"), the field embedding is the average of the feature embeddings [2]. More formally, in an instance, each field f_i ($1 \leq i \leq m$) is represented as a low-dimensional vector $e_i \in R^{1\times k}$, where m is the number of fields and k is the embedding size. Therefore, each instance can be represented as an embedding matrix $E = (e_1^\top, e_2^\top, ..., e_m^\top)^\top \in R^{m\times k}$. Assume there are n features, the embeddings of all the features form an embedding table $\mathcal{E} \in R^{n\times k}$.

2.1.2 Interaction and Prediction Layers. The key challenge in CTR prediction is modelling feature interactions. Existing deep CTR models utilize product operation and multi-layer perception (MLP) to model the explicit and implicit feature interactions, respectively. For example, DeepFM [4] adopts Factorization Machine [13] to model the order-2 feature interactions and MLP to model the high-order feature interactions. The method about how to model feature interactions is beyond the scope of this work, those who are interested in such techniques, please refer to [4, 8, 9, 12, 15].

After the interaction layer, the prediction $\hat{y}$ is generated as the probability of the user will click on a specific item within such context. Then, the cross-entropy loss is used as the objective function:

$$\mathcal{L}_{CE}(y,\hat{y}) = -y\log\hat{y} - (1-y)\log(1-\hat{y}). \tag{1}$$

with y as the label.

2.2 Batch Mode V.S. Incremental Mode

In this section, we present and compare two different training modes, namely batch mode and incremental mode.

2.2.1 Training with Batch Mode. The model trained in the batch mode is iteratively learned based on the data from a fixed-size time window. When new data is arriving, the time window slides forward. As shown in Figure 3, "model 0" is trained based on data from day 1 to day 10. Then when new data (namely "day 11") is arriving, a new model (namely "model 1") is trained based on data from day 2 to day 11. For similar procedure, "model 2" is trained on data from day 3 to day 12.

2.2.2 Training with Incremental Mode. With incremental mode, the model is trained based on the existing model and new data. As shown in Figure 3, the "Model 1" is trained based on the existing model "Model 0" (which is trained on data from day 1 to day 10), and data from day 11. Then "Model 1" turns into the existing model.

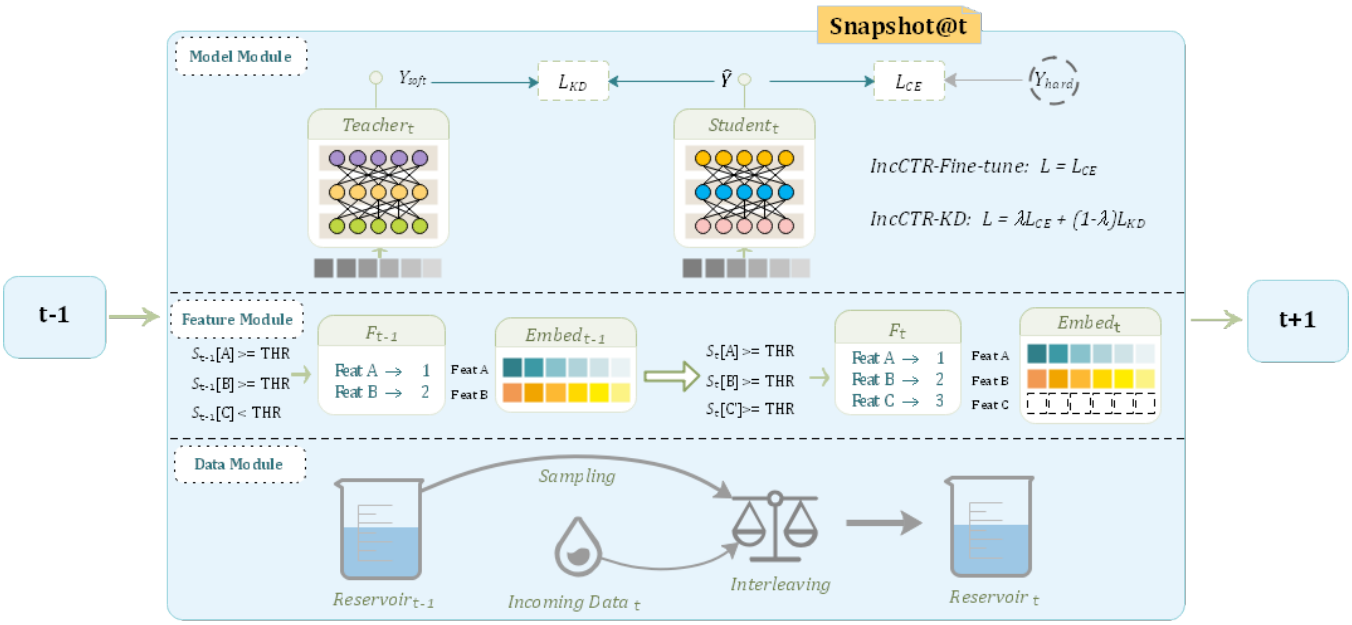

Figure 2: Overview of IncCTR architecture, where t indicates the incremental step.

Consequently, when data from day 12 arrives, "Model 2" is trained based on the "Model 1" and data from day 12.

As can be seen, when training with batch mode, two consecutive time window of training data overlap most of the volume. For instance, data from day 1 to day 10 and data from day 2 to day 11 overlap the part from day 2 to day 10, where 80% of the volume are shared. Under such circumstance, replacing batch mode with incremental mode improves the efficiency significantly, while such replacement is highly possible to retain the performance.

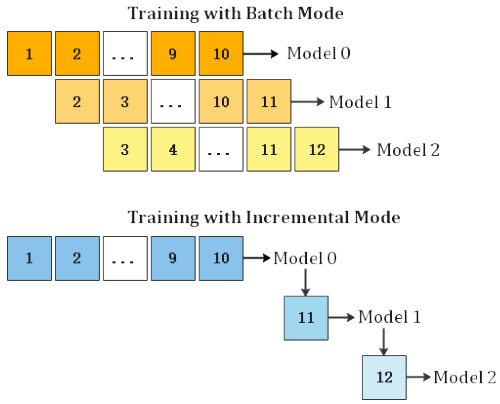

Figure 3: Training with Batch Mode V.S. Incremental Mode. Each block represents one day of training data.

3 METHODOLOGIES

An overview of our incremental learning framework IncCTR is shown in Figure 2. Three modules are designed from the perspective of *feature*, *model* and *data* respectively to balance the trade-off between learning from historical data and incoming data. Specifically, data module serves as a reservoir, constructing training data from both historical data and incoming data. Feature module handles new features from incoming data and initializes both existing features and new features properly. Model module employs knowledge distillation to fine-tune the model parameters.

3.1 Feature Module

In recommendation and information retrieval scenarios, the feature dimension is usually very high, i.e., in million- or even billion-scale [17]. The occurrence frequency of such large number of features follows long-tailed distribution, where only minor proportion of the features occur frequently and the rest are rarely presented. As observed in [10], half of the features in their model occur only once in the whole training data. The features that are rarely occurred are difficult to be learned well. Therefore, when training with batch mode, features are needed to be categorized to "frequent" or "infrequent" by counting the number of occurrences of each feature. More formally, a feature x with its occurrence $S[x]$ larger than a pre-defined threshold THR (i.e., $S[x] > THR$) is considered as "frequent" and is learned as an individual feature. The rest "infrequent" features are treated as a special dummy feature *Others*. After such processing, each feature is mapped to a unique id by some policy like auto-increment, hash-coding, and etc. We take an auto-increment policy F for simplicity. In batch mode, policy F is constructed from scratch by assigning unique ids to individual

features from the training data in a fixed-size window, where the unique ids increase one by one automatically.

However, training with incremental mode brings additional issue as new features appear when new data comes in. As displayed in Figure 4, every block of new data brings a certain proportion of new features. For instance, as observed from Criteo dataset, the first block of new data imports 12% of new features compared to the set of existing features before this block, while even the 14th block still brings 4% of new features. Therefore, the policy F needs to be updated incrementally when new data comes in. It is possible that a feature x, which is previously treated as *Others*, is considered as a unique feature if its occurrence $S[x]$ is above the threshold THR after new data coming in.

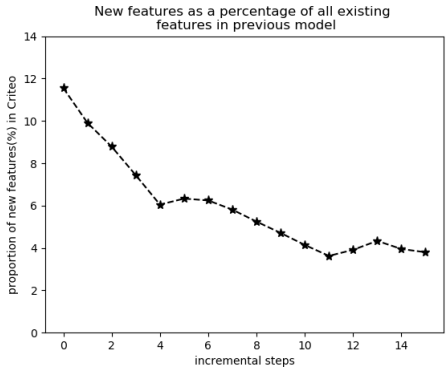

Figure 4: Proportion of new features compared to the set of existing features as blocks of new data comes in, observed from Criteo dataset.

After assigning proper ids to all the features, feature module in IncCTR initializes both existing features and new features. When we start training with batch mode, all the feature embeddings $\mathcal{E}$ are initialized randomly. Whereas, in case of incremental mode, we initialize the embeddings of existing features $\mathcal{E}_{exist}$ and the embeddings of new features $\mathcal{E}_{new}$ separately.

The functionality of feature module (namely, new feature assignment and feature embedding initialization) is presented in Algorithm 1. When new data comes in, we firstly update the occurrences of each feature (line 3) and inherit the existing policy of feature assignment (line 4). If a feature from the new data is new with its occurrence larger than the threshold (line 6), it is added to the policy with id incremented by one (line 7). Feature embeddings are initialized separately, depending on whether a feature is new or not. For an existing feature, it inherits from the embedding of the existing model as its initialization (line 11). Such inheriting transfers the knowledge in historical data to the model that will be trained incrementally. For a new feature, its embedding is randomly initialized as no prior knowledge is available (line 12).

3.2 Model Module

In this section, we introduce the model module in IncCTR, which trains the model properly, such that the model still "remembers" the knowledge from historical data and also makes some "progress" from the new data.

Fine-tune. Besides the embedding of existing features, the network parameters are also inherited from the existing model as warm-start. To fine-tune all the model parameters with incremental mode, we apply some auxiliary tricks to achieve good performance. For instance, we conduct a lower learning rate for $\mathcal{E}_{exist}$ compared to that for $\mathcal{E}_{new}$. The training details of fine-tune are presented in line 19 to line 25 in Algorithm 2. The model is optimized by minimizing cross entropy between prediction and groundtruth. We train the model for a fixed number of epochs, where empirically we set the number to be 1 (line 25).

Algorithm 1 Feature Module: New Feature Assignment and Feature Embedding Initialization

Input: features of coming data X_t; labels of coming data Y_t; existing model M_{t-1}; existing policy of feature assignment F_{t-1}; frequency of existing features S_{t-1}
Output: initialized feature embeddings $\mathcal{E}_t$

1: **Initialize**: feature frequency threshold THR
2: **New Feature Assignment**:
3: $S_t \leftarrow Feature\_Frequency\_Update(S_{t-1}, X_t)$
4: $F_t \leftarrow F_{t-1}$
5: **for** all features f in X_t **do**
6: **if** $f \notin F_t \wedge S_t[f] > THR$ **then**
7: $F_t \leftarrow add(f, |F_t| + 1)$
8: **end if**
9: **end for**
10: **Feature Embedding Initialization**:
11: $\mathcal{E}_{exist} \leftarrow \mathcal{E}_{t-1}$
12: $\mathcal{E}_{new} \leftarrow Random\_Initialize$
13: $\mathcal{E}_t \leftarrow \mathcal{E}_{exist} \cup \mathcal{E}_{new}$
14: **Return**: $\mathcal{E}_t$

Knowledge distillation. Beyond "fine-tune" presented above, we introduce knowledge distillation (KD) method to enhance the knowledge learned from the historical data (namely, to avoid catastrophic forgetting). Hinton et al. in [6] use KD to transfer knowledge from an ensemble of models into a single model for efficient deployment, where KD loss is used to preserve knowledge from the cumbersome model through encouraging the outputs of distilled model to approximate that of cumbersome model. Similarly, the authors of LwF [7] perform KD to learn new task while keeping knowledge on old tasks. Borrowing the similar idea, KD can also be used in incremental learning scenario to learn new knowledge from incoming data while preserving memory on historical data.

Several options are available to design the teacher model when we apply KD in IncCTR. We present two such options.

- **KD-batch.** Outdated model trained with batch mode can be a natural choice for teacher model to distill the incremental model, which can preserve performance on the historical data within a fixed-size window. We refer the KD method with such teacher trained with batch mode as "KD-batch".
- **KD-self.** As training a model with batch mode as teacher model needs extra computation resources, it is more convenient to perform the previous incremental model as the teacher. In such case, the successive incremental model is trained with the supervision of the previous incremental

model. We refer such a design as "KD-Self". Similar idea is employed in *BANs* [3], where a consecutive student model is initialized randomly while taught by the previous teacher model, and the ensemble of multiple student generations was conducted to achieve desirable performance. This work is in image recognition field where all the models are trained in batch mode, which is significantly different from our framework.

When performing KD, we utilize the soft targets Y_{soft} generated by the teacher model on the incoming data. The objective function is formulated as follows:

$$\mathcal{L} = \mathcal{L}_{CE}(Y, \hat{Y}) + \mathcal{L}_{KD}(\hat{Y}, Y_{soft}) + \mathcal{R} \tag{2}$$

$$\mathcal{L}_{KD}(Y, Y_{soft}) = \mathcal{L}_{CE}(\sigma(\frac{Z}{\tau}), \sigma(\frac{Z_{soft}}{\tau})) \tag{3}$$

$$\mathcal{L}_{CE}(Y, \hat{Y}) = \sum_{y_i \in Y} \mathcal{L}_{CE}(y_i, \hat{y}_i) \tag{4}$$

The new objective function combines the standard binary cross-entropy $\mathcal{L}_{CE}(\cdot)$ (where Y and $\hat{Y}$ denote the groundtruth and outputs of the new model respectively) and KD loss $\mathcal{L}_{KD}(\cdot)$. KD loss $\mathcal{L}_{KD}(\cdot)$ is the cross entropy between $\hat{Y}$ and Y_{soft} (where Y_{soft} is the prediction of the teacher model) which are computed based on logits Z and Z_{soft} of two models. Temperature τ is applied to get soft targets and $\mathcal{R}$ is the regularization term. The intuition of loss function in Equation 2 is that the knowledge of the distilled model should be precise to the new data (first term), while it should not be significantly different from the knowledge of the teacher model (second term).

The training details of KD-batch and KD-self are presented in line 3 to line 5 and line 11 to line 17 in Algorithm 2. The difference between KD-batch and KD-self is how the teacher model $Teacher_t$ is trained. Remind that the teacher model in KD-batch is an outdated model trained with batch model while the teacher model in KD-self is the previous incremental model. We will compare their performance empirically in Experiment section. Given features of input data, the incremental model M_t and the teacher model $Teacher_t$ make the predictions, as in line 4 and line 13. Then the incremental model M_t is optimized by minimizing the loss function in Equation 2, as in line 14. The train process terminates when the model is trained with at least one epoch and KD loss stops decreasing, as in line 17.

3.3 Data Module

From data perspective, one straightforward way to tackle the catastrophic forgetting problems is to train the incremental model not only based on the new data, but also based on some selected historical data. We plan to implement a data reservoir to provide proper training data for incremental training. Some proportion of data in the existing reservoir and the new data are interleaved to be the new reservoir. In this module, some problems are needed to look into, such as what and which proportion of data in the existing reservoir should be kept. The implementation of data module is not finished for now and will be a part of future work to accomplish our framework.

Algorithm 2 IncCTR

Input: features of incoming data X_t; labels of incoming data Y_t; existing model M_{t-1}; Teacher Model $Teacher_t$; existing policy of feature assignment F_{t-1}; frequency of existing features S_{t-1}
Output: Model M_t

1: **Initialize**: $L = MaxInt$; ep=0; EPOCH=1
2: $\Theta_t \leftarrow$ {Algorithm 1 $\cup$ network parameters}
3: **if** Train with KD **then**
4: $\quad Y_{soft} \leftarrow Inference(Teacher_t, X_t)$
5: $\quad M_t = Train\_with\_KD(X_t, Y_t, Y_{soft}, \Theta_t)$
6: **else**
7: $\quad M_t = Train\_with\_FT(X_t, Y_t, \Theta_t)$
8: **end if**
9: **Return**: M_t
10:
11: ***Train_with_KD***:
12: **while** *stopping criteria* not satisfied **do**
13: $\quad \hat{Y}_t = Inference(M_t, X_t)$
14: $\quad M_t \leftarrow \arg\min_{\Theta_t} \left(\lambda \mathcal{L}_{CE}(Y_t, \hat{Y}_t) + \mathcal{L}_{KD}(\hat{Y}_t, Y_{soft}) + \mathcal{R}\right)$
15: $\quad$ ep += 1
16: **end while**
17: *stopping criteria*: $\mathcal{L}_{KD}(\hat{Y}_t, Y_{soft})$ increases and ep >= EPOCH
18:
19: ***Train_with_FT***:
20: **while** *stopping criteria* not satisfied **do**
21: $\quad \hat{Y}_t = Inference(M_t, X_t)$
22: $\quad M_t \leftarrow \arg\min_{\Theta_t} \left(\mathcal{L}_{CE}(Y_t, \hat{Y}_t) + \mathcal{R}\right)$
23: $\quad$ ep += 1
24: **end while**
25: *stopping criteria*: ep >= EPOCH

4 EXPERIMENT

In this section, we conduct experiments on a public benchmark and a private dataset, aiming to answer the following research questions:

- **RQ1**: What is the performance of IncCTR compared to training with batch mode?
- **RQ2**: What are the contribution of different modules in IncCTR framework?
- **RQ3**: How efficient is IncCTR compared to training with batch mode?

4.1 Dataset

To evaluate the effectiveness and efficiency of the proposed IncCTR framework, we conduct extensive experiments both on a public benchmark and a private dataset.

- Criteo[1]. This dataset is used to benchmark algorithms for click-through rate (CTR) prediction. It consists of 24 days' consecutive traffic logs from Criteo, including 26 categorical features and 13 numerical features, and the first column as label indicating whether the ad has been clicked or not.

[1] http://labs.criteo.com/downloads/download-terabyte-click-logs/

- HuaweiApp. In order to demonstrate the performance of the proposed method in real industrial tasks, we conduct offline experiments on a commercial dataset. HuaweiApp contains 60 consecutive days of click logs collected from Huawei AppGallery with user consent, consisting of app features, anonymous user features and context features.

For the ease of reproducing our experimental results, we present the details of data processing on Criteo data. In a nutshell, we follow Kaggle Champion[2] and [11], which involving data sampling, discretization and feature filtering. We do not give details for processing HuaweiApp dataset for commercial reasons, but the procedure is similar.

- Data sampling: Considering data imbalance (only 3% samples are positive), similar to [12], we apply negative down sampling to keep the positive ratio close to 50%.
- Discretization: Both categorical and numerical features are existing in Criteo. However the distribution of two kinds of features is quite different intrinsically [11]. In most of recommendation models, numerical features are transformed to categorical features through bucketing or logarithm. Following that, we use logarithm[2] as the discretization method in the formulation:

$$v \leftarrow floor(log(v)^2) \tag{5}$$

- Feature filtering: Infrequent features are usually not very informative and may be noisy, so that it is hard for models to learn such features well. Therefore, features in a certain field appearing less than 20 times are set a dummy feature *Others*, following [11].

4.2 Evaluation Protocols

Evaluation Metrics. We adopt AUC (Area Under Curve) and logloss (cross-entropy) as our evaluation metrics which are widely used in CTR prediction models. Besides, it has been endorsed that an improvement of AUC and logloss at 0.1% can be considered as significant for a CTR prediction model [1, 4, 15].

Baseline. Training model with batch mode is used as the baseline, to validate the effectiveness and efficiency of IncCTR. To further evaluate the impact on delay updating of models, we consider the baseline with different delay days. More specifically, Batch_i ($i = 0, 1, 2, 3, 4, 5$) represents the baseline model with i days' delay.

Implementation Details. As the focus of this work is to compare the effectiveness and efficiency of deep CTR models when training with batch mode and incremental training by IncCTR, we choose a popular deep CTR model DCN [15] for such comparison. Observations on other deep CTR models are similar.

To mimic the training procedure in industrial scenarios, experiments are conducted over consecutive days. When training with batch mode, all the data in the fixed size window (i.e., data in size-w window $[s, s + w)$, where $s \in [0, T - w]$) are utilized. While training with incremental mode, only the data with the coming day (i.e., data in size-1 window $[s, s + 1)$, where $s \in [w, T - 1]$) is available. Warm-start is performed for the incremental models with a model trained with batch mode before the first incremental step. That is to say, we firstly train a warm-started model with batch mode on data in $[0, w)$, then train the first incremental model on the data of w-th day. We set $w = 7, T = 23$ for Criteo dataset and $w = 30, T = 59$ for HuaweiApp dataset.

[2]http://https//www.kaggle.com/c/criteo-display-ad-challenge/discussion/10555

4.3 Overall Performance (RQ1)

Table 1 presents the overall performance comparison over consecutive test days. Models trained with batch mode with different delayed days Batch-i ($i \in [1, 5]$) are also compared here. Specifically, Batch-0 represents the model that is tuned on latest data (serving as validation) and then fine-tuned with latest data (serving as train data), namely training process is carried out twice. Batch-0 reaches the performance upper bound of batch mode, however, it is not feasible in practice as it doubles the training time of batch mode. Relative improvement of the best incremental model over the other models in terms of AUC is reported in "Impr." column. From this table, we have the following observations.

- On Criteo dataset, incremental models achieve better effectiveness than baselines trained with batch mode which accelerates training procedure significantly. Specifically, incremental models outperform all the baselines with significant improvement (more than 0.1%) over Batch-1 to Batch-5. Surprisingly, IncCTR achieves comparable performance to Batch-0 (which is an ideal model with upper bound performance).
- On HuaweiApp dataset, consistent results are acquired. Incremental models obtain comparable performance with baselines with huge efficiency improvement. Negligible decrease exists when comparing the performance of IncCTR with Batch-1 which can be ignored in practice. Unsurprisingly, Batch-0 utilizing the entire dataset outperforms the other models. Nonetheless, as stated earlier, Batch-0 is infeasible in practice as it doubles the training time in batch mode.
- On both datasets, severe performance degradation is observed as the delayed period extends, which calls for efficient training method. In industrial scenarios, updating delay of 1-3 days is a common phenomenon if the model is trained with batch mode in the single device which imputes to the enormous data volume, tedious preprocessing, cumbersome model structure and so on. Longer updating delay is also possible in some more complicated settings when multi-task or ensemble are needed. We can see that, with incremental learning method IncCTR, the improvement of performance is quite significant when model updating delay occurs, for instance, AUC improves 0.6% and 0.4% when 5-day delay exists in HuaweiApp dataset and Criteo dataset (as shown in Figure 1).

4.4 Ablation Studies (RQ2)

To validate the contribution of feature module and model module in IncCTR, we do some ablation studies over such two modules.

- Feature Module. In feature module, if the number of occurrences of a new feature from incoming data is above the threshold, we would assign an individual feature id to this feature. To verify the usefulness of the new feature expanding strategy, we conduct experiments to compare the effectiveness of whether expanding the new features during

Table 1: Overall performance comparison between IncCTR and training with batch mode over consecutive days on Criteo and HuaweiApp datasets. Mean AUC and Logloss over consecutive days are reported. The underlined numbers represent the performance of the best variant of IncCTR. Beside performance, average epochs and average train time (sec) for updating a new model are also presented for efficiency comparison.

model	Criteo					HuaweiApp				
	AUC	Logloss	avg epochs	avg time (s)	Impr.	AUC	Logloss	avg epochs	avg time (s)	Impr.
Batch-0	0.7977	0.5438	18	32694.66	0.06%	0.8543	0.0859	17.34	91711.26	-0.14%
Batch-1	0.7956	0.5464	9	16347.33	0.33%	0.8532	0.0861	8.67	45855.63	-0.01%
Batch-2	0.7946	0.5476	9	15907.42	0.45%	0.8523	0.0863	7.7	39697.77	0.09%
Batch-3	0.7939	0.5485	8	14020.88	0.54%	0.8514	0.0866	7.8	44587.14	0.20%
Batch-4	0.7932	0.5492	10	17718.42	0.63%	0.8496	0.0870	7.57	43478.67	0.41%
Batch-5	0.7925	0.5501	9	15853.46	0.72%	0.8475	0.0874	7.73	39030.66	0.66%
IncCTR-Fine-tune	**0.7982**	**0.5431**	1	258.48	0	0.8527	0.0861	1	167.35	0.05%
IncCTR-KD-batch	0.7982	0.5442	1.18	265.32	0	**0.8531**	**0.0861**	1.25	213.75	0
IncCTR-KD-self	0.7981	0.5462	1	261.66	0.01%	0.8530	0.0862	1.20	198.66	0.01%

Table 2: Feature Module: w/o new features V.S. with new features (AUC improvement).

model	Criteo	HuaweiApp
w/o new features	-	-
with new features	+0.08%	+0.055%

incremental learning over the two datasets. As shown in Table 2, the consistent performance degradation demonstrates the necessity of expanding new features. Specifically, because there are more new features per day on Criteo dataset, the strategy of new features expanding has greater impact on this dataset. Therefore, the performance of IncCTR on Criteo dataset drops greater than that on HuaweiApp dataset when new features are not considered. Besides, such decrease becomes more severe as the model keeps incrementally training, as shown in Figure 5, where more than 0.1% performance decrease is observed eventually.

- Model Module. Fine-tune and knowledge distillation are applied in the model module. Compared with fine-tune, two KD methods (KD-batch and KD-self) achieve similar performance on Criteo and slight better performance on HuaweiApp dataset. This may because of too few new features emerging in new data. On average, only about 6% new features arising each day in Criteo dataset, while similar phenomenon happens in HuaweiApp dataset. Comparing two KD methods, their performance are very close to each other, which suggests that KD-self is a better choice in practice as no extra computational resource is needed for training the teacher model.

4.5 Efficiency (RQ3)

Average epochs and training time (sec) of different models are summarized in Table 1. When training with batch mode, one epoch means going through over all data within the fixed-size window, while training with incremental mode, it means going through the incoming data. Tremendous advantage on efficiency is revealed, as we are able to get 60x and 270x improvement over average training time on Criteo and HuaweiApp dataset respectively, which is extremely helpful in practice.

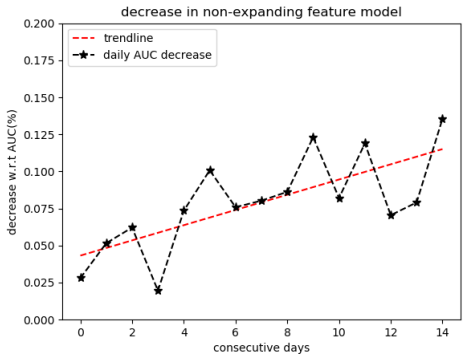

Figure 5: Performance decrease when new features are not expanded in Feature Module of IncCTR, over consecutive days on Criteo dataset.

5 CONCLUSION

In this paper, we propose the IncCTR, which is a practical incremental method to train deep CTR models. IncCTR includes data module, feature module and model module. Specifically, we propose new features expanding and initialization strategies in feature module, and propose several training algorithms in model module. Comprehensive experiments are conducted to demonstrate the effectiveness of the two modules in incCTR. Compared with conventional batch mode training, our method can achieve comparable performance in terms of AUC but with much better training efficiency, which is extremely helpful in practice.

There are two interesting directions for future study. One is investigating the novel approach to utilize history data to guarantee the stability of incremental training, such as reservoir. The other one is how to update (add or remove) the features incrementally and efficiently in the production environment of recommender system.

REFERENCES

[1] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, et al. 2016. Wide & Deep Learning for Recommender Systems. In *DLRS@RecSys*. ACM, 7–10.

[2] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. ACM, 191–198.

[3] Tommaso Furlanello, Zachary C. Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. 2018. Born Again Neural Networks. arXiv:stat.ML/1805.04770

[4] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. Deepfm: a factorization-machine based neural network for ctr prediction. In *IJCAI*. 1725–1731.

[5] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, and Stuart Bowers. 2014. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Eighth International Workshop on Data Mining for Online Advertising*. 1–9.

[6] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *CoRR* abs/1503.02531 (2015). http://arxiv.org/abs/1503.02531

[7] Zhizhong Li and Derek Hoiem. 2016. Learning Without Forgetting. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV (Lecture Notes in Computer Science)*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.), Vol. 9908. Springer, 614–629.

[8] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. *arXiv preprint arXiv:1803.05170* (2018).

[9] Bin Liu, Ruiming Tang, Yingzhi Chen, Jinkai Yu, Huifeng Guo, and Yuzhou Zhang. 2019. Feature Generation by Convolutional Neural Network for Click-Through Rate Prediction. In *The World Wide Web Conference, San Francisco, CA, USA, May 13-17*. ACM, 1119–1129.

[10] H. Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. 2013. Ad click prediction: a view from the trenches. In *ACM SIGKDD*. https://doi.org/10.1145/2487575.2488200

[11] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-Based Neural Networks for User Response Prediction. In *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*. ACM, 1149–1154.

[12] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiuqiang He. 2019. Product-based Neural Networks for User Response Prediction over Multi-field Categorical Data. *ACM Trans. Inf. Syst.* 37, 1 (2019), 5:1–5:35.

[13] Steffen Rendle. 2010. Factorization machines. In *ICDM*. IEEE, 995–1000.

[14] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *ADKDD*. ACM, 12.

[15] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 12:1–12:7.

[16] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep learning over multi-field categorical data. In *European conference on information retrieval*. Springer, 45–57.

[17] Weijie Zhao, Deping Xie, Ronglai Jia, Yulei Qian, Ruiquan Ding, Mingming Sun, and Ping Li. 2020. Distributed Hierarchical GPU Parameter Server for Massive Scale Deep Learning Ads Systems. *CoRR* abs/2003.05622 (2020). https://arxiv.org/abs/2003.05622

[18] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2018. Deep Interest Evolution Network for Click-Through Rate Prediction. arXiv:stat.ML/1809.03672

[19] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *SIGKDD*. ACM, 1059–1068.