Efficient Learning to Learn a Robust CTR Model for Web-scale Online Sponsored Search Advertising

Xin Wang[1], Peng Yang[1], Shaopeng Chen[2], Lin Liu[2], Lian Zhao[2], Jiacheng Guo[2], Mingming Sun[1], Ping Li[1]
[1]Cognitive Computing Lab, Baidu Research
[2]Baidu Sponsored Search (Phoenix Nest)
No.10 Xibeiwang East Road, Beijing 100193, China
10900 NE 8th St. Bellevue, Washington 98004, USA
{wangxin60, pengyang01, chenshaopeng, liulin03, zhaolian, guojiacheng, sunmingming01, liping11}@baidu.com

ABSTRACT

Click-through rate (CTR) prediction is crucial for online sponsored search advertising. Several successful CTR models have been adopted in the industry, including the regularized logistic regression (LR). Nonetheless, the learning process suffers from two limitations: 1) Feature crosses for high-order information may generate trillions of features, which are sparse for online learning examples; 2) Rapid changing of data distribution brings challenges to the accurate learning since the model has to perform a fast adaptation on the new data. Moreover, existing adaptive optimizers are ineffective in handling the sparsity issue for high-dimensional features.

In this paper, we propose to learn an optimizer in a meta-learning scenario, where the optimizer is learned on prior data and can be easily adapted to the new data. We firstly build a low-dimensional feature embedding on prior data to encode the association among features. Then, the gradients on new data can be decomposed into the low-dimensional space, enabling the parameter update smoothed and relieving the sparsity. Note that this technology could be deployed into a distributed system to ensure efficient online learning on the trillions-level parameters. We conduct extensive experiments to evaluate the algorithm in terms of prediction accuracy and actual revenue. Experimental results demonstrate that the proposed framework achieves a promising prediction on the new data. The final online revenue is noticeably improved compared to the baseline. This framework was initially deployed in Baidu Search Ads (a.k.a. *Phoenix Nest*) in 2014 and is currently still being used in certain modules of Baidu's ads systems.

CCS CONCEPTS

• Information systems → Computational advertising; **• Computing methodologies → Transfer learning**.

KEYWORDS

Computational advertising; CTR prediction; Meta-learning

ACM Reference Format:
Xin Wang[1], Peng Yang[1], Shaopeng Chen[2], Lin Liu[2], Lian Zhao[2], Jiacheng Guo[2], Mingming Sun[1], Ping Li[1]. 2021. Efficient Learning to Learn a Robust CTR Model for Web-scale Online Sponsored Search Advertising. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3459637.3481912

1 INTRODUCTION

Online advertising services are crucial components in modern e-commerce business, including sponsored search advertising, display advertising, and social media advertising, etc. Companies including Google, Facebook, Baidu, Amazon, etc. rely quite heavily on the revenues generated by online advertising. In this paper, we share a story about Baidu's online advertising technology, which was deployed in 2014 and is still being currently used in some modules of Baidu's ads system (a.ka. *Phoenix Nest*). In the past a few years, Baidu has published a series work on their ads systems, focusing in 1) general deep learning platforms [54, 61, 62]; 2) approximate near neighbor search algorithms for advertising [11, 44, 65]; and 3) specific topics related to online advertising [13, 14, 57, 58], etc.

In this paper, the presented ads technology was developed in 2014, based on the meta-learning paradigm, for the task of CTR (click-through rate) predictions [3, 10]. An advertiser system would pay an agreed amount per click for its adverts on the publishers' web pages in online advertising. To improve the revenue, a publisher needs to estimate the CTR, which is the number of clicks on a specific ad to the times of this ad shown to the users. The publisher can then choose the advertisement to maximize the income by considering both CTR and pay per click (PPC). Besides advertisement content and search query, nowadays, publishers can choose ads based on the user profile. In this context, CTR for the same ad varies with each individual. In order to obtain reasonable CTR, the publishers train machine learning predictors to distill the knowledge from large-scale datasets. The predictor's performance is crucial to choose advertisements and improve revenue. Thus, the online learning prediction system should have a fast adaptation to capture the ever-changing search trends, web pages, and ad content.

Besides the model parameter, the hyper-parameters should also be updated accordingly in the dynamic scenario. Inappropriate hyper-parameters of optimization, such as learning rate or momentum, may hinder model converging at an acceptable time cost. A possible solution to obtain a suitable optimizer is to build a meta-model. However, models such as [2, 5, 7] are not implementable for online advertising with trillion-level features, which require to be efficient in both time and memory. In addition, to get a low latency during a search, only a small part of filtered ads go into the CTR predicting process. It means that, for a short period, the number of collected samples is relatively small compared with the trillions of features. Therefore, the gradient would be sparse, and the weight can not be updated adequately.

To address the above problems, we explore building an optimizer as a meta-model to predict the change of CTR model parameters. In this way, we avoid hyper-parameters searching for new data. Specifically, the optimizer firstly projects the trillions of features into a representative low-dimensional space. Then we encode the association between original feature space and new space based on prior data. When processing the new data, updates of parameters can be smoothed by propagating upon the association matrix. In this way, the optimizer relieves the sparsity of gradients of new data. More parameters of the CTR model can be updated with less bias on historical experience. The pivot vector of the propagation process can be estimated by gradient descent on a distributed system instead of algebraic computation. Both association estimation and clustering for new space generation can be conducted in parallel. Thus, it is implementable in a web-scale online advertising system.

In 2014, the presented model was deployed as the main platform for Baidu's sponsored search. Both online and offline evaluations showed improvement in prediction accuracy as well as revenue. In summary, the proposed model enjoys the following advantages:

- *Adaptation.* Traditional learning technology can find the optimal parameters and hyper-parameters for the prior data, but the performance may drop significantly in new data. The proposed meta-learner can obtain general knowledge from prior knowledge. We generate the higher-level abstract features to adapt the model into new data in online learning. In this way, the learned predictor can be robust for newly collected data of low-proportion.
- *Feature Richness.* The traditional model addresses the sparsity issue by using regularization terms, which constrain the number of non-zero parameters. The features with strong signals are retained and impact the prediction performance. In this work, we explore to learn the feature similarity with a meta-learner. In this way, features can be updated sufficiently via interacting with similar ones. Thus, richer features can contribute to the CTR prediction with non-zero parameters, and hence improving the performance of the system.
- *Efficiency.* The proposed framework ensures efficiency. The meta-model is **trainable** with gradient descent. It ensures that the parameters of the optimizer can converge quickly. The meta-learner has a **low-dimensional** feature space, and it thus reduces the computational complexity of the meta-learner inference process. Also, the meta-learner can be implemented and optimized in a **parallel** distribution system to speed up the updating process.

Organization. In Section 2, we formally describe the traditional CTR model and its limitation in practice. The details of the proposed solution considering performance and efficiency are described in section 3. The experiment settings and results are present in Section 4. We discuss related work in section 5 and conclude in section 6.

2 BACKGROUND

2.1 Filtering for Low Latency

For each query, it is expensive to estimate the CTR for all relevant advertisement candidates. The search engine adopts a filtering pipeline for ad prioritization and CTR prediction to reduce the computational cost. For each query, the search engine retrieves millions of relevant ad candidates. The advertisements are then selected according to the expanded queries and user's rich profile, grasping only hundreds of the most relevant ads. After that, the CTR prediction and final ranking process are deployed on these prioritized candidates to maximize the revenue. In this way, the required computing resources are reduced, and the search engine can respond with low latency.

2.2 Data Block in Online Setting

In practice, the datasets used to train the CTR predictor do not follow the i.i.d assumption. Moreover, as millions of new samples appear per day, we cannot save these data permanently with limited resources. Instead, the data used in each iteration of online learning is collected with a sliding window on the data sequence.

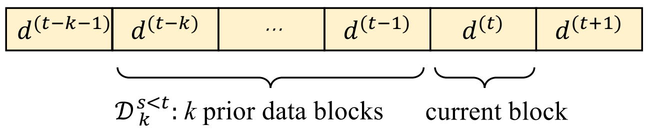

Figure 1: Illustration of online learning. Each block represents the data collected in a period. Each data set used for model learning contains multiple adjacent blocks. As time goes on, the sliding window adds the newly collected data blocks and drops the oldest ones.

As shown in Figure 1, we split the data sequence into several data blocks. At iteration t, block $d^{(t)}$ is the newly collected data that has never been observed before, while block set $\mathcal{D}_k^{(s<t)}$ contains k previous data blocks $d^{(s<t)}$ (including $d^{(t-1)}$, $d^{(t-2)}$, ..., $d^{(t-k)}$), which has been learned in previous trials. Both $\mathcal{D}_k^{(s<t)}$ and $d^{(t)}$ are used in the training process to achieve a trade-off between the current data block and previous ones. After that, the training data is updated by adding a new block and removing the oldest one. With the setting of life-long learning, the model training process could keep pace with the online data evolution.

2.3 Difficulties in CTR Prediction Process

Sparsity. Although billions of initial candidates are generated for a specific query, the number of instances used in CTR modeling is relatively small after filtering described in Section 2.1. On the other hand, rich basic features are extracted from the user profile, query history, ad title, ad context, related page context, search session, and so on. Moreover, trillions of feature crosses are generated by combining the basic features to get higher-order information. As

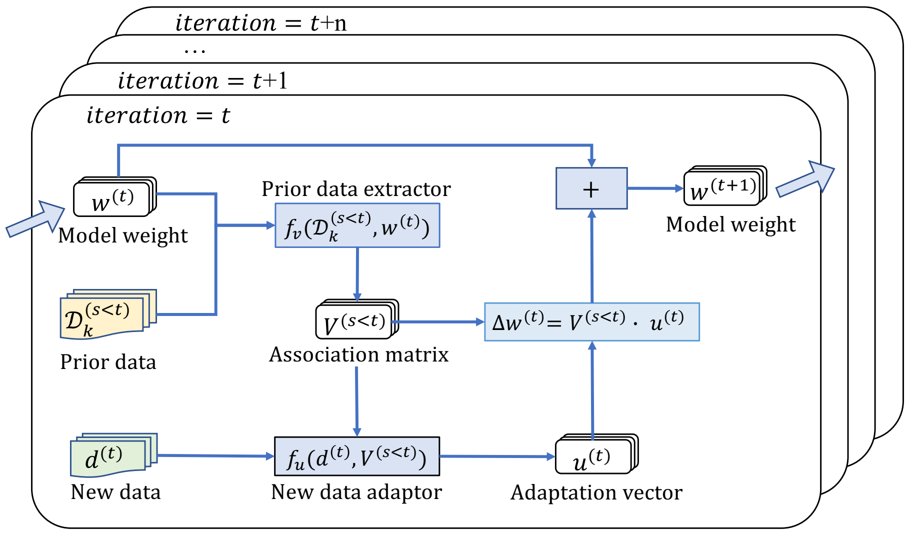

Figure 2: The process of updating parameters of the base CTR model with meta-model. The meta-model is learnt on prior data with extractor f_v and fine-tuned on new data with adaptor f_u. The parameter change of the base model is obtained via a process of gradient propagation. The updated base model parameter is used as the input of the meta-model in the next iteration.

a result, these one-hot features (including both basic features and feature crosses) are relatively sparse with respect to the number of filtered samples.

Data Shifting. The environment of search engines changes dramatically. Users' interests in new products or breaking news can grow explosively. The number of web pages, including advertisement pages, also increase rapidly. If the CTR model cannot be updated frequently, it may fail to capture the search trend. The ever-changing data make optimization very hard. Both model parameter and hyper-parameter should also be tuned with the dataset shift [52]. Hyper-parameter influences the final performance in many ways. For example, an optimization-related hyper-parameter determines the convergence rate and result of the learning process. Therefore, both model parameters and hyper-parameters should be continuously updated to maintain an accurate prediction. There are two major issues during the learning process. Firstly, the learner suffers a cold-start issue. Although new information is critical to future prediction, only a few new samples are observed, leading to a bias towards historical experience. Secondly, the data sparsity may further aggravate the imbalance.

Meta-learning technology is usually a promising solution to deal with the cold-start issue. An optimizer can also be learned as a neural meta-model [2], and has the potential to search for an optimal hyper-parameter for a new data set. However, few works can handle the sparsity problem. Current meta-learning algorithms may suffer in two aspects considering the trillion-level scale of features and corresponding gradients. Firstly, the time cost of the recurrent process is unacceptable. Secondly, the storage of hidden states and linear mapping parameters cost too much ($40 \times$ trillion) memory. This paper aims to adopt meta-learning into the CTR prediction and enable an efficient and robust meta-learning on the high-dimensional and sparse dataset.

3 EFFICIENT ROBUST LEARNING TO LEARN FRAMEWORK

This section describes the Learn To Update (LTU) framework that Baidu deployed in the online system in 2014. Generally, the parameters are updated via gradient descent at every iteration. At the t-th iteration of online learning, the update process can be described as:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta\mathbf{w}^{(t)} \tag{1}$$

Instead of directly setting $\Delta\mathbf{w}^{(t)} = \mathbf{g}^{(t)}$. Here $\mathbf{g}$ is the product of gradient and the learning rate. In the rest of the paper, we denote $\mathbf{g}$ as "gradient" when there's no ambiguity. We explore to learn the parameter change via a trainable optimizer. The contributions of the proposed framework are two folds: Firstly, the optimizer is learned as a meta-model to encode the knowledge of previous data and perform fast adaptation to the new data with a few samples. Secondly, the meta-model learns a low-dimensional space for trillions of features, which enables the gradient smoothed over similar features and mitigates sparsity. The framework is illustrated in Figure 2.

We train a **meta-model** as the optimizer on the prior data. This model encodes the similarity of parameter change patterns among features. When the new data comes, the learned optimizer could be fine-tuned to adjust the similarity of features, enabling a fast adaptation to new data with a few samples and performing stably over the data shift. Besides, we can smooth parameter update by gradient propagation among similar features, which would generate more non-zero parameters to mitigate the sparsity. In this way, more features with non-zero parameters can contribute to the CTR prediction and boost CTR prediction performance. Note that the traditional model addresses the sparsity issue by imposing a regularization term to constrain the parameters. Unlike existing regularization fixing the prior distribution of parameters, the proposed optimizer dynamically learns the association between parameters along data evolution when handling sparsity.

In the rest of this section, we first introduce the problem setting for the meta-learning framework in Section 3.1. Then we describe the parameter-change prediction process to show how to solve the gradient sparsity in Section 3.2. Finally, we discuss the details of implementation in each step to show how to improve the efficiency and accuracy in Section 3.3.

3.1 Problem Setting

As shown in Figure 2, the framework is designed with a meta-learning setting. This subsection introduces the definitions of the functions and shows how datasets are used in different phases.

There are n features in the CTR model. For this n-dimensional features space at iteration t, the parameter update variable, denoted as $\Delta\mathbf{w}^{(t)} \in \mathbb{R}^n$, can be computed as:

$$\Delta\mathbf{w}^{(t)} = V^{(s<t)} \cdot \mathbf{u}^{(t)} \tag{2}$$

where $V^{(s<t)} \in \mathbb{R}^{n\times r}$ is the matrix describing the association between the features space and the embedding space, and r is the dimension of the embedding space with $r \ll n$. Specifically, we select recent data block in k-size window for prior data, i.e., $s \in \{t-k, \cdots, t-1\}$. Given the base-model parameter $\mathbf{w}^{(t)}$ and prior data $\mathcal{D}_k^{(s<t)}$, the association matrix is learned by

$$V^{(s<t)} = f_v(\mathcal{D}_k^{(s<t)}, \mathbf{w}^{(t)}), \tag{3}$$

where f_v is the function that actually builds low-dimensional embedding space and encoding the association between spaces.

For $\mathbf{u}^{(t)} \in \mathbb{R}^r$, it is a vector that enables a model adaption into the current inputs. Given the current data $d^{(t)}$ and association matrix $V^{(s<t)}$ learned from prior data, the meta-parameter is learned by

$$\mathbf{u}^{(t)} = f_u(d^{(t)}, V^{(s<t)}), \tag{4}$$

where f_u is an optimizer that exploits $\mathbf{u}^{(t)}$ to adapt association matrix learned from prior data into the current data $d^{(t)}$.

Note that the proposed learning setting is different from the conventional learning algorithm that uses both $\mathcal{D}_k^{(s<t)}$ and $d^{(t)}$ to train the model. In this work, the proposed optimizer separately exploits prior data, and current data in two different stages: (1) the meta-model exploits $\mathcal{D}_k^{(s<t)}$ to train the association matrix $V^{(s<t)}$ and encode the knowledge of prior data; (2) the model performs a fast adaptation into current data $d^{(t)}$ via learning $\mathbf{u}^{(t)}$.

It is worth noting that the functions f_v and f_u can be defined flexibly for different concerns. In this paper, we implement the functions for data adaptation and parameter smoothness.

3.2 Algorithm Schema

Due to the sparse feature input, the gradient $\mathbf{g}^{(t)}$ would be a sparse vector, which results in sparse parameter learning. We next introduce a smoothing adaptation strategy to address the sparsity issue on new data adaptation with only a few samples.

Many correlated features are used during prediction (E.g., city and its weather or basic feature and its feature crosses). Gradients can inherit such a correlation. In gradient descent for LR, for the i-th feature, the feature value $\mathbf{x}_i$ is a factor of the gradient $\mathbf{g}_i$.

$$\mathbf{g}_i(\mathbf{x}, y) = \frac{\partial L(\hat{y}, y)}{\partial \mathbf{w}_i} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\mathbf{w}_i} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} \cdot \mathbf{x}_i \tag{5}$$

$$\mathbf{g}^{(t)} = \sum_{(\mathbf{x},y)\in d^{(t)}} \mathbf{g}(\mathbf{x}, y) \tag{6}$$

where $\hat{y} = \mathbf{w}^\top\mathbf{x}$ is the predicted output, y is the truth label, and $L(\hat{y}, y)$ is the loss function. Obviously, correlated features values $\mathbf{x}_a$ and $\mathbf{x}_b$ lead to correlated $\mathbf{g}_a$ and $\mathbf{g}_b$. Therefore, similarity among gradients is decided by the inherent properties of naturally correlated features. And thus, it is irrelevant to the datasets. Therefore, we can estimate the similarity between gradients on a large prior dataset and use it on a small newly collected dataset.

In other words, when processing new small-scale data, we propagate the sparse gradient $\mathbf{g}^{(t)}$ based on similarity to obtain a smoothed parameter update $\Delta\mathbf{w}^{(t)}$.

Figure 3 illustrates the whole process, which mainly consists of four steps: We first 1) learn a low-dimensional space u for trillions of features based on prior data and then 2) estimate the association scores V between the feature space and the embedding spaces. When new data comes, we 3) fine-tune the low-dimensional space with the new data and 4) calculate $\Delta\mathbf{w}^{(t)}$ according to the feature association matrix and the updated low-dimensional space. The details of the process are described as follows.

Step 1: Construction of Low-dimensional Space. We perform a clustering on the gradient $\mathbf{g}^{(s<t)}$ to learn an r-dimensional ($r \ll n$) space, where $\mathbf{g}^{(s<t)}$ is the gradient of the loss at model parameter $\mathbf{w}^{(t)}$ against the prior data $\mathcal{D}_k^{(s<t)}$. We exploit k-means as the clustering algorithm since its algorithm is data-oriented and can be supported on a distributed system when dealing with large-scale data [59]. The low-dimension space $\mathbf{u}$ could be computed by

$$\mathbf{u}^{(s<t)} = parallel\_kmeans(\mathbf{g}^{(s<t)})$$

Nonetheless, the gradient $\mathbf{g}^{(s<t)}$ reflects the changing trend of the model parameters. This work also exploits other meta-data during parallel clustering to describe detailed aspects of the changing trend to learn an accurate low-dimensional space. Corresponding implementation details are introduce in 3.3.1 and 3.3.2.

Step 2: Estimation for Space Association. We learn the association matrix $V^{(s<t)}$ between original feature spaces and low-dimensional embedding spaces via fuzzy membership function, where each element $v_{i,j}^{(s<t)}$ $(1 \leqslant i \leqslant n, 1 \leqslant j \leqslant r)$ describes how the change of feature parameter i (denoted as $\mathbf{g}_i^{(s<t)}$) is associated to a cluster $\mathbf{u}_j^{(s<t)}$ [34]. Intuitive, feature parameters with similar update behavior in $\mathbf{g}^{(s<t)}$ will associate with similar clusters in the low-dimensional space $\mathbf{u}^{(s<t)}$. Thus, we exploit $\mathbf{g}^{(s<t)}$ and $\mathbf{u}^{(s<t)}$ to build the association matrix $V^{(s<t)}$ as

$$V^{(s<t)} = fuzzy\_membership(\mathbf{g}^{(s<t)}, \mathbf{u}^{(s<t)})$$

According to Eq. (5), the similarity is decided by the natural correlation among features, and thus, it is stable. The similar updating behavior could be observed between two consecutive datasets, i.e., $V^{(t)} \approx V^{(s<t)}$ when s is close to t. Thus, we exploit to learn the $V^{(s<t)}$ as the association embedding that can adapt into the new data at iteration t. We will further discuss the details about the sparse fuzzy membership function with a $O(n * c)$ $(c \ll r)$ time and space complexity in Section 3.3.3.

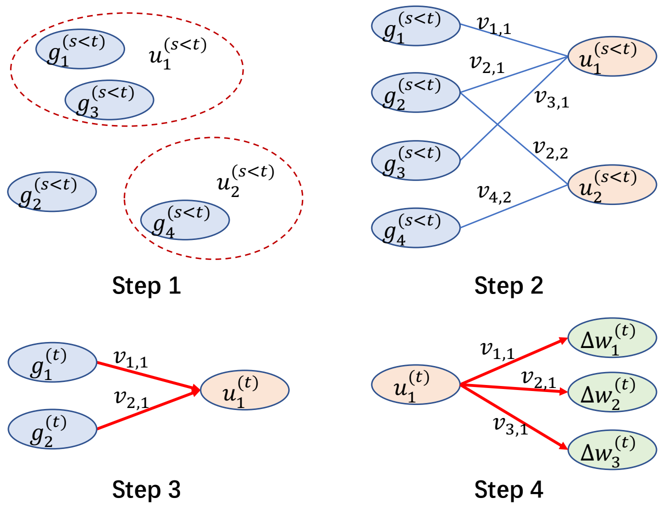

Figure 3: Illustration of meta parameter update process. Step 1 shows the gradient projected into a low-dimensional vector $\mathbf{u}^{(s<t)}$. Step 2 illustrates the matrix $V^{(s<t)}$ that encodes the association between the feature space and the embedding space. Step 3 is the process of estimating $\mathbf{u}^{(s<t)}$ by propagating gradient $\mathbf{g}^{(t)}$. Step 4 describes the computation of parameter update $\Delta\mathbf{w}^{(t)}$ with $V^{(s<t)}$ and $\mathbf{u}^{(t)}$.

Step 3: Adaptation on New Data. With the association matrix $V^{(s<t)}$ from prior data, we next adapt the feature representation into the new data $d^{(t)}$ via learning a scaling vector $\mathbf{u}^{(t)}$. Specifically, $\mathbf{u}^{(t)}$ is fine-tuned via stochastic gradient descent (SGD) to minimize the CTR prediction loss on the data $d^{(t)}$. Given a sample $(\mathbf{x}, y) \in d^{(t)}$, the gradient $\Delta\mathbf{u}$ is formally computed as:

$$\begin{aligned}\Delta\mathbf{u}^\top &= \sum_{(\mathbf{x},y)\in d^{(t)}} \frac{\partial L(\hat{y},y)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \mathbf{w}^\top} \cdot \frac{\partial \mathbf{w}^\top}{\partial \mathbf{u}} \\ &= \sum_{(\mathbf{x},y)\in d^{(t)}} \frac{\partial L(\hat{y},y)}{\partial \hat{y}} \cdot \mathbf{x}^\top \cdot V^{(s<t)} \\ &= \mathbf{g}^{(t)\top} \cdot V^{(s<t)} \end{aligned} \tag{7}$$

where $V^{(s<t)}$ is fixed during the fine-tuning process.

From Eq. (7), we observe that $\Delta\mathbf{u}$ is learned by propagating the gradient $\mathbf{g}^{(t)}$ from the trillions of features into the low-dimensional space according to the association matrix $V^{(s<t)}$. In other word, the gradient on new data $\mathbf{g}^{(t)}$ are projected into the low-dimensional space via the matrix production $\mathbf{g}^{(t)\top} \cdot V^{(s<t)} \in \mathbb{R}^r$. Note that the $\mathbf{u}^{(t)}$ could be optimized in parallel on a distributed system.

Step 4: Calculation of Parameter-Change. Finally, we can compute $\Delta\mathbf{w}^{(t)}$ via the product of the association matrix $V^{(s<t)}$ and the vector $\mathbf{u}^{(t)}$, as present in Eq. (2). Since $\mathbf{u}^{(t)}$ is learned by the gradient descent on new data as in Eq. (7), the product term $\Delta\mathbf{w}^{(t)} = V^{(s<t)} \cdot \mathbf{u}^{(t)}$ could properly approximate the gradient on new data $\mathbf{g}^{(t)}$, and thus could be used to update the model parameter (CTR model) that fits the new data $d^{(t)}$. Note that $\Delta\mathbf{w}^{(t)}$ is approximated gradient via projecting the learned vector $\mathbf{u}^{(t)}$ from a low-dimensional space $\mathbb{R}^d$ back to the original high-dimensional feature space $\mathbb{R}^n$. In such a step, the data knowledge could be shared via propagating the information among the features with similar low-dimensional embedding. The elements in $\mathbf{u}^{(t)}$ can be considered as pivots of the gradient propagation process.

3.3 Implementation Details

In this subsection, we describe the details of algorithm implementation. The proposed implementation aims to improve time and space efficiency without sacrificing accurate performance.

3.3.1 Updating Trend Representation. As mentioned in *Step 1*, the gradient $\mathbf{g}$ on prior data is exploited to generate a low-dimensional space for the trillions of features. Besides the gradient, we introduce two statistic measures, i.e., direction $\mathbf{a}$ and scale size $\mathbf{b}$, to describe parameter updating behavior.

The *direction* $\mathbf{a}_i$ describes whether feature i sufficiently contributes to the prediction. The sign of $\mathbf{a}_i$ is correlated with the sign of $\mathbf{g}_i$, showing the direction of updating. Specifically, the direction $\mathbf{a}_i$ is computed as the log ratio of the number of actual clicked samples to the predicted number of clicks.

$$\begin{aligned} \mathbf{a}_i &= log\frac{\alpha + |D_{click}|}{\alpha + |D_{predict}|} \\ \text{s.t.} \quad D_{click} &= \{(\mathbf{x}, y) \in \mathcal{D} | \mathbf{x}_i = 1, y = 1\} \\ D_{predict} &= \{(\mathbf{x}, y) \in \mathcal{D} | \mathbf{x}_i = 1, \hat{y} > 0.5\} \end{aligned} \tag{8}$$

where the positive scalar $\alpha > 0$ is used for Laplace smoothing [32], which guarantees the denominator and final result larger than zero.

The *scale size* $\mathbf{b}$ is considered as the scale of parameter update. The more related samples observed, the high confidence we have. Therefore, the $\mathbf{b}_i$ can be computed with the number of positive feature observations in the prior data.

$$\begin{aligned} \mathbf{b}_i &= log(\beta + |D_{observed}|) \\ \text{s.t.} \quad D_{observed} &= \{(\mathbf{x}, y) \in \mathcal{D} | \mathbf{x}_i = 1\} \end{aligned} \tag{9}$$

where the positive scalar $\beta > 0$ ensures that $\beta + |D_{observed}|$ is larger than zero.

Finally, the updating behavior of each feature i could be represented as a triplet of updating-related statistics $\{\mathbf{g}_i, \mathbf{a}_i, \mathbf{b}_i\}$ during the clustering process for embedding space generation.

3.3.2 Intra-category Clustering. In the CTR model, each one-hot feature is included in a feature category, e.g., the binary features ‘search query is about laptop or not’ and ‘search query is about a phone or not’ belong to the same category ‘electronic product’. Features from the same category tend to have better comparability and connection. Therefore, we choose to build the low-dimensional embedding space and subsequent steps only for features within the same category. To optimize hundreds of feature categories, we perform feature learning on these categories in parallel to speed up the whole learning process.

3.3.3 Association Estimation and Selection. We introduce the membership function [34] to encode the association between features and clusters. Given the gradient on feature i, denoted as $\mathbf{g}_i$, and the representation of cluster j, denoted as $\mathbf{u}_j$, we aim to minimize the distance between $\mathbf{g}_i$ and $\mathbf{u}_j$ if feature i is associated with cluster j. The objective function aims to learn the membership matrix V so that the weighted distance between feature and cluster is minimized:

$$\begin{aligned} V &= \underset{V}{argmin} \sum_{i=1}^{n} \sum_{j=1}^{r} V_{i,j}^2 \cdot dis(\mathbf{g}_i, \mathbf{u}_j) \\ \text{s.t.} & \sum_{j=1}^{r} V_{i,j} = 1, \quad i \in [1, n] \end{aligned} \tag{10}$$

where $dis(a, b)$ represents for the distance between a and b, and $V_{i,j}$ is (i, j)-element of the matrix V. The solution of Eq. (10) is the normalized reciprocal of distance [45], which is:

$$V_{i,j} = \frac{1/dis(\mathbf{g}_i, \mathbf{u}_j)}{\sum_{j=1}^{r}(1/dis(\mathbf{g}_i, \mathbf{u}_j))} \tag{11}$$

In the online advertising system, it is infeasible to store all $n \times r$ elements in V. Therefore, we only calculate top-c closest clusters for each feature ($c \ll r$) via searching for the c nearest cluster representation $\mathbf{u}$ towards the gradient vector on each feature $\mathbf{g}_i$. In this way, only $n \times c$ elements are computed and stored in memory. The nearest clusters are the most representative ones with the largest association score in V, so we can obtain an accurate approximation of V without losing too much accuracy.

3.3.4 Weight Initialization for Stability. There are several ways to initialize $\mathbf{w}^{(0)}$. One naive way is to randomly initialize parameters of the model, which, however, leads to the cold-start issue in an online web-scale commercial system. Since the online gradient descent model has been trained on the stream data in previous years, its model parameter could be a good initialization point and directly used in our meta-learning CTR framework. To this end, we use the parameters of the online CTR model as the initial parameters $\mathbf{w}^{(0)}$ in this work.

3.3.5 Decomposition View. Besides propagation, approximate decomposition is another view of understanding the smoothing process. Principal components analysis (PCA) with singular value decomposition (SVD) [48] and alternating least squares (ALS) [27] have achieved a remarkable result in matrix completion. Similarly, in this work, we conduct approximate decomposition to reduce the sparsity on parameter changes and make the $\Delta\mathbf{w}^{(t)}$ smoothing.

4 EXPERIMENTS

4.1 Experimental Settings

4.1.1 Baseline. **GD**: The baseline system is the LR model with L1 regularization, whose weights are updated with gradient descent (GD), which is the previous online CTR model in our search engine. The model is trained with combination of $\mathcal{D}_k^{(s<t)}$ and $d^{(t)}$, and then tested on $d^{(t+1)}$. The hyper-parameters are selected with a grid search.

4.1.2 Parameter Setting. The experiment split data to block by day. We set the window size k to 10 to exploit data of the previous ten days to learn the model. The training set contains billion-level samples in this setting, while the validation set and test set include hundreds of millions of samples. Cluster number r is set to 100 to achieve a balance between performance and efficiency. We set $c = 3$ to obtain the three nearest clustering centers and corresponding coefficient.

4.2 Evaluation

The evaluation is conducted by training the model with data described in 3.1 and testing with data of the next day $d^{(t+1)}$. The experiment result is shown in Table 1. The definitions of the metrics and analysis of the results are as follows.

Table 1: Comparisons: AUC, relative error, and the number of contributing features for GD (baseline) and LTU (proposed).

	AUC	Relative Error	Contributing Features
LR-GD	0.7875	0.06	500 million
LR-LTU	0.7912	0.04	10 billion

4.2.1 AUC. The area under the curve (AUC) is an important evaluation metric for CTR prediction. We use the AUC of receiver operating characteristic (ROC) to evaluate the prediction [12, 20].

As shown in Table 1, the AUC of this work is improved by 0.37% compared with the GD baseline. It is a noticeable improvement for the web-scale system, which demonstrates the effectiveness of our model. The GD baseline is always in the dilemma of timeliness and performance. The fixed hyper-parameter in the GD baseline may degrade the performance. For instance, hyper-parameters for optimization, such as learning rate, affect the convergence during training and may lead to a sub-optimal solution. Nonetheless, grid search for hyper-parameter is time-consuming, especially in a web-scale commercial advertising system.

By contrast, the proposed optimizer predicts parameter changes that fit the new data directly. There are no optimization hyper-parameters that need to tune. In this way, the proposed method achieves a fast adaptation to new data.

4.2.2 Relative Error. To further validate the effectiveness of the proposed framework, we evaluate the relative error between the ground-true CTR and the predicted one. This metric aims to test whether the avenue predicted by the model is close to the real value. We conduct a piece-wise evaluation per sample and then compute the average result over all the instances. The predicted CTR ranges from 0 to 1. This range is then equally divided into n intervals. The i-th interval contains m_i samples with their predicted values in this interval. And the relative error in the i-th piece is denoted as e_i, which can be computed as:

$$e_i = \frac{\sum_{j=1}^{m_i} click_j - \sum_{j=1}^{m_i} predict_j}{\sum_{j=1}^{m_i} predict_j} \tag{12}$$

where $click_j = 1$ if the user actually clicks the sample according to the log, otherwise, $click_j = 0$. And $predict_j$ is the predicted CTR. The overall *error* is averaged the sample number on each interval.

$$error = \frac{1}{n}\sum_{i=1}^{n} e_i \tag{13}$$

We set $n = 1000$ in this experiment. Table 1 shows that the proposed model can reduce the relative error by 33% from 0.06 to 0.04 compared with the GD baseline, demonstrating the effectiveness of our CTR prediction model. Hence, the advertisement prioritization becomes more accurate.

4.2.3 *Contributing Features.* There are trillions of features used in both systems. We define the features with non-zero parameters as contributing features, which influence the regression result. As shown in Table 1, for the GD baseline, the number of contributing features is limited to 500 million due to the regularization penalty.

The prior distributions for the parameters are often introduced to avoid over-fitting with sparse data. E.g., existing regularization technology involves constant prior such as Laplacian prior (L1 regularization) or Gaussian prior (L2 regularization). This work organizes features with their natural categories and involves dynamic prior that changes with data in each category. Each parameter in new data belongs to some fuzzy clustering centers. The membership function becomes the dynamic prior evolved on recent data of online learning iteration. The parameter updates are smoothed based on such a prior, and consequently, the weight sparsity is relieved. Thus, around 10 billion contributing features can provide more comprehensive information to the CTR model, boosting the prediction performance.

4.3 Comparison with Identical Distribution Settings

Conventional settings assume that training and testing data are identically distributed. Thus, there are no subdivisions of training data for meta-learning or other transfer technology. We implement this setting in our framework by learning association embedding V and adaptation vector $\mathbf{u}$ on the same data. It can be viewed as an ablation study that only remains the smoothing mechanism and excludes meta-learning.

Specifically, we learn $V^{(s\leqslant t)}$ on $\mathcal{D}_k^{(s\leqslant t)}$, which is the combination of $\mathcal{D}_k^{(s<t)}$ and $d^{(t)}$. Then, $u^{(s\leqslant t)}$ is estimated on data randomly selected from $\mathcal{D}_k^{(s\leqslant t)}$. The smoothed parameter updates can be computed as $\Delta\mathbf{w} = V^{(s\leqslant t)} \cdot u^{(s\leqslant t)}$. We call this learning setting as the *Identical* group, while the proposed meta-learning setting described in 3.1 as the *Evolving* group.

Note that the optimizer adaptation on small data is similar to the validation step in traditional gradient descent, which chooses hyper-parameter for the optimizer. Therefore, we call the small data used to estimate u as *validation* data and the data $d^{(t+1)}$ used for final evaluation is called *testing* data.

We evaluate the algorithms in two stages. In stage 1, we calculate the AUC on *validation* set with the GD baseline model trained on the training set. In stage 2, we calculate AUC on *testing* data with the model smoothed and adapted on the validation set.

Table 2: The comparison of AUCs on two kind of validation sets. One has identical distribution with the training set. The other has different distribution with the training set.

Data	f_v Input	f_u Input	Validation	Testing
Identical	$\mathcal{D}_k^{(s\leqslant t)}$	from $\mathcal{D}_k^{(s\leqslant t)}$	**0.9736**	0.7538
Evolving	$\mathcal{D}_k^{(s<t)}$	$d^{(t)}$	0.7899	**0.7912**

The performances achieved by *Identical* group and *Evolving* group are shown in Table 2. In the first stage, the *Identical* group achieves a higher AUC since the data sets used in this stage are identically distributed. On the contrary, for the *Evolving* group, it is uneasy to predict CTR on the validation set since it is different from the training set in terms of data distribution, which we call a data shift over time. In the second stage, the performance of the *Identical* group falls sharply, while the AUCs on the *Evolving* group are robust on testing data. It is because that the *Evolving* setting learns $V^{(s<t)}$ and $\mathbf{u}^{(t)}$ on different data and avoids over-fitting on a particular one.

4.4 Sensitivity Study

We conduct the sensitivity study regard to the sliding window size and number of clusters in the meta-training phase. Besides, sensitivity to average measures is also tested in this subsection.

4.4.1 *Window Size.* We study the impact of sliding window size in the learning setting. Basically, with the increased size of the sliding window, more adjacent data blocks will be collected for model updating in the meta-train phase. Specifically, we fix the cluster number to 100 and tune the sliding window size from 1 days to 15 days with a step size of 5 days. We show the comparison performance in the left part of Figure 4. We observe that both validation

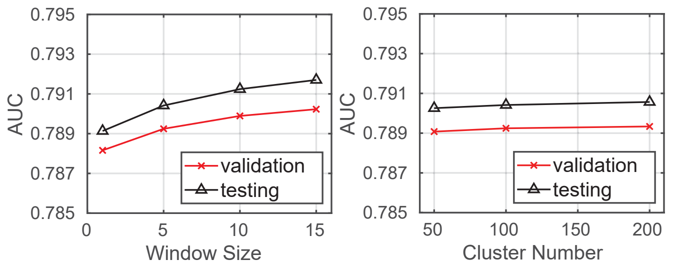

Figure 4: Sensitivity on window size and cluster number.

performance and test performance achieve better performance consistently with various sliding window sizes. The more prior data we use, the more accurate the correlation is encoded. This demonstrates the algorithm that is capable of exploiting large-scale sequential data to maximize the generalization of the predictive model.

4.4.2 Cluster number. In the proposed algorithm, clustering is essential to learn the feature-dependency in the meta-train process. To analyze the impact of clustering in meta-learning, we set the number of clusters using the grid $\{50, 100, 200\}$ while fixing the sliding window size to 5 days. The results in the right part of Figure 4 show that the proposed algorithm can improve the AUC performance with an increased number of clusters. However, further increasing the cluster number brings a higher computational cost. It motivates us to select a proper number of clusters to achieve a balance. Therefore, we set the cluster number to be 100 in the rest of the experiments, since in this setting, the algorithm achieves a high AUC while the computational cost is relatively low.

4.4.3 Average Estimation. Cumulative values such as $|D_{observed}|$ in Eq. (9) should be normalized by the number of samples. Thus, we adopt average values instead of sum values. The moving-average [35] changes smoothly in the online system. Thus, we use the averaged value by default. The AUC result 0.7912 (reported in Table 1) is based on this setting.

On the other hand, the arithmetic mean is the exact reflection of the true data. Therefore, we conduct an experiment with an arithmetic mean and obtain an AUC of 0.7907. We observe that the statistical operation on average has little effect on the performance.

4.5 Online A/B Testing

4.5.1 Click-through Rate. Online A/B testing is conducted to evaluate the influence of the proposed optimizing strategy. Table 3 shows the results of evaluation metrics. The CTR of the proposed model is improved by about 6% compare to the GD baseline. The reason is that the system recommends high CTR ads according to the accurate prediction of the proposed meta-learning framework.

Table 3: The improvements in CTR and ACP of the proposed framework compared with the previous system deployed on different websites/apps. The results are based on our 7-day surveillance of partial online traffic during A/B testing.

	CTR	ACP
Affiliated Websites/Apps	+5.9725%	+0.5725%
Mobile App	+6.3037%	+1.2833%
Overall	+6.0577%	+0.7608%

An accurate CTR predictor encourages the advertiser's system to invest resources in the bid process. Therefore, it leads to adequate price competition and higher click prices. As shown in Table 3, the overall average click price (ACP) increases by 0.76%.

4.5.2 Update Frequency. The update frequency is the time of parameter update with the newly collected dataset per day. It indicates how fast the algorithms adapt to the new dataset. Table 4 shows the max updating frequency of the two models. The GD baseline model can be updated 10 times per day, while the proposed framework can be updated 18 times per day. We observe that the proposed framework's training process converges much faster, and thus the updating frequency improves by 80%. As discussed above, the optimizer is learned automatically and avoids grid search for optimization hyper-parameters. Therefore, the system could fit the new data more quickly and bring improvement to CTR.

Table 4: The max frequency of CTR model updating.

	Max updating frequency
LR-GD	10 times per day
LR-LTU	18 times per day

4.6 System Launching

After the improvement is validated on online A/B testing, the proposed framework is launched on both our mobile app and affiliated websites/apps. Table 5 shows the improvement of various evaluation metrics according to the 7-day monitor on the entire online mobile traffic. Both CTR and ACP improve, showing a positive trend same as the A/B testing.

Table 5: The improvements in CPM, CTR, and ACP of the proposed framework compared with the previous system deployed on different websites/apps. The results are based on our 7-day surveillance of the entire online traffic.

	CPM	CTR	ACP
Affiliated Websites/Apps	+3.9778%	+3.8497%	+1.5826%
Mobile App	+3.3428%	+4.9941%	+3.5823%
Overall	+3.8143%	+4.1629%	+3.8266%

Given the stable page view, Cost Per Mile (CPM) is essential for revenue. The total CPM is increased by 3.8% after the proposed framework launching, indicating a notable revenue increase over sponsored search. This work provides an efficient solution for data-shift and feature sparsity, boosting the CTR prediction performance. And the proposed framework finally improves the revenue of the advertising platform that relies on an accurate CTR.

5 RELATED WORK

5.1 CTR Prediction

CTR prediction has a significant impact on the revenue of a sponsored search. It is one of the core tasks of an online advertising platform. Extensive works have been done made for CTR prediction [33, 43, 47]. Clicking or not can be regarded as the Bernoulli trial so that the probability of click satisfies the binomial distribution. Existing work exploits logistic regression (LR) to predict the ratio between 0 and 1 [39]. The learned parameters intuitively reflect the importance of the features, which enables LR Interpretable and error-traceable. This method is convenient for application because it runs parallelizable and requires less computation overhead. Therefore, LR is the most widely deployed CTR prediction model in the industry.

Feature crosses are introduced to generate non-linear feature combinations to make LR more expressive [40]. However, the feature-dimensional explosion occurs when considering feature cross among various slots (advertisements, users, and queries). Hidden variable based methods, such as Factorization Machine (FM) [38], and Field-aware Factorization Machine (FFM) [28, 56], are proposed to reduce the dimension as well as ease the data sparsity, and they have also been used widely for collaborative ranking and rating predictions [24, 25].

An alternative way to speed up the LR inference process is to induce sparsity of parameters. Approaches such as L_1-normalization, Regularized Dual Averaging (RDA) [53], and Follow the Regularized Leader (FTRL) [33] address the sparsity issue in different ways and achieve remarkable improvements. The proposed framework explores improving the parameters update method with a smoothing mechanism that propagates gradients on a low-dimensional space.

Tree-based models such as Gradient Boost Decision Tree have the advantage in choosing the features with high information gain [16, 17, 29, 30] and have been used for feature crosses generation and selection [22]. In web search, tree methods are also the mainstream for learning to rank [31, 63]. Deep neural networks have been used to extract higher-level abstract features [6], better feature cross [19, 21, 49] or adjust attention on historical features [64].

The proposed framework is orthogonal to these approaches. No matter how the features are generated or how the parameters are updated in existing approaches, the proposed meta-learner can smooth the weight changes by propagation through low-dimensional space learned on prior data.

5.2 Meta-learning

This paper is inspired by many works of various meta-learning directions. Wistuba et al. introduced the idea of predicting hyper-parameter configurations with a differentiable function [50]. However, the assumption that the new task will be similar to one of the existing tasks is impractical in the setting of online learning, which motivates us to build a more robust meta-learner. It is efficient to learn the hyper-parameter with gradient descent over a differentiable loss function on the streaming dataset. Several research works focus on speeding up the meta-learning process by reducing the dimension of hyper-parameters [8, 26]. Inspired by previous works, we reduce the dimension of meta-learner with cluster representation. The idea of building simplified models to simulate complex processes is widely used in hyper-parameter searching, which is known as building surrogate models with Bayesian optimization [15, 51]. Different from these works, we use a space mapping function to directly estimate the parameter change and avoid complex hyper-parameter searching.

Meta-features are used to reflect the statistical distribution of the input tasks. Helpful meta-features may include class distribution [23] and inter-class dependency [1]. The meta-features used in this work encode the properties of the datasets in two ways. Firstly, the directions and step sizes are estimated statistically via clustering the tuples of training data. Secondly, the inter-feature dependency could be learned from the subspace of cluster centers. Such higher-level representations achieve invariance among datasets. The general knowledge can be transferred with the representations such as sparse coding [37], low-rank representation [42], or activation in stacked denoising auto-encoders [4, 18]. This work exploits parallel k-means to get robust subspace and uses the fuzzy membership function of clustering to capture the inter-feature dependency in the subspace.

A major category of meta-learning research most related to this work is transfer learning. Transfer learning initializes the new model with parameters learned on the historical data [9, 36, 46]. Meta-learning technologies improve parameter optimization on neural networks. Recent works address this problem called "learning to optimize" [2, 5]. These works propose a functional block to update the optimizer automatically. Daniel et al. learned step sizes by training a controller [7]. Schmidhuber used the parameters of RNN as additional inputs and optimized both the base learner and meta learner simultaneously [41]. In this work, the meta-learner aims to learn the dependency between the features, while the base learner exploits the inter-feature dependency (e.g., cluster) to reformulate the feature embedding for CTR prediction. The features embedding used in this work are more interpretable than neural models, and the feature generation process is flexible in considering the timeliness and robustness.

Reinforcement learning with meta-gradient also has the potential to adapt to changes in the search environment [55]. However, the historical click log cannot provide the label of all newly generated exploration samples, and online exploration could bring a huge risk for a commercial advertising platform.

6 CONCLUSION

Online advertising and CTR prediction tasks are vital for Baidu as well as all major search engines in the world. In recent years, Baidu has published a series of papers, including the GPU parameter server framework (Paddlebox) for training CTR models [61, 62] and the algorithms for approximate near neighbor search (ANNS) and maximum inner product search (MIPS) [11, 44, 60, 65] for improving the recall quality/speed in the early stage of the pipeline and more.

In this paper, we describe the CTR training framework, which was based on meta-learning and was deployed in 2014 in Baidu's ads system. The work has inspired the development of many new CTR algorithms at Baidu. Even to date, it is still used in some modules of Baidu's advertisement system. We believe the work will be interesting to the researchers and practitioners in CTR predictions and web search in general.

In summary, the developed meta-learning framework is for learning to learn a web-scale CTR model for online sponsored search advertising. The meta-model encodes feature association on prior data and uses it to estimates the smoothed gradients on new data. It can be used as an optimizer trained with online streaming data of evolving distribution, and thus it is insensitive for data shift. The proposed algorithm propagates the gradient according to the correlation among features and enables the knowledge transferring among the related features. Both offline and online experiments show improvements in AUC. Without a grid search for hyper-parameters, the possible update frequency has increased by 80%. Practical strategies such as parallelization and weight copying are implemented to ensure efficiency and stability during model switching, respectively. In 2014, after the system was initially deployed online as the CTR prediction framework of the search engine, it brought a significant improvement by 3.8143% of final revenue for the advertising platform of wireless searching.

REFERENCES

[1] Shawkat Ali and Kate A Smith. 2006. On learning algorithm selection for classification. *Applied Soft Computing* 6, 2 (2006), 119–138.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems (NIPS)*. Barcelona, Spain, 3981–3989.

[3] Andrei Broder. 2002. A taxonomy of web search. *SIGIR Forum* 36, 2 (2002), 3–10.

[4] Minmin Chen, Zhixiang Eddie Xu, Kilian Q. Weinberger, and Fei Sha. 2012. Marginalized Denoising Autoencoders for Domain Adaptation. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*. Edinburgh, Scotland, UK.

[5] Yutian Chen, Matthew W. Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Timothy P. Lillicrap, Matthew Botvinick, and Nando de Freitas. 2017. Learning to Learn without Gradient Descent by Gradient Descent. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Sydney, Australia, 748–756.

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS@RecSys)*. Boston, MA, 7–10.

[7] Christian Daniel, Jonathan Taylor, and Sebastian Nowozin. 2016. Learning Step Size Controllers for Robust Neural Network Training. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*. Phoenix, AZ, 1519–1525.

[8] Alex Guimarães Cardoso de Sá, Walter José G. S. Pinto, Luiz Otávio Vilas Boas Oliveira, and Gisele L. Pappa. 2017. RECIPE: A Grammar-Based Framework for Automatically Evolving Classification Pipelines. In *Proceedings of the 20th European Conference on Genetic Programming (EuroGP)*. Amsterdam, The Netherlands, 246–261.

[9] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. 2014. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In *Proceedings of the 31th International Conference on Machine Learning (ICML)*. Beijing, China, 647–655.

[10] Daniel C. Fain and Jan O. Pedersen. 2006. Sponsored search: A brief history. *Bulletin of the American Society for Information Science and Technology* 32, 2 (2006), 12–13.

[11] Miao Fan, Jiacheng Guo, Shuai Zhu, Shuo Miao, Mingming Sun, and Ping Li. 2019. MOBIUS: Towards the Next Generation of Query-Ad Matching in Baidu's Sponsored Search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. Anchorage, AK, 2509–2517.

[12] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.

[13] Hongliang Fei, Shulong Tan, Pengju Guo, Wenbo Zhang, Hongfang Zhang, and Ping Li. 2020. Sample Optimization For Display Advertising. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM)*. Virtual Event, Ireland, 2017–2020.

[14] Hongliang Fei, Jingyuan Zhang, Xingxuan Zhou, Junhao Zhao, Xinyang Qi, and Ping Li. 2021. GemNN: Gating-enhanced Multi-task Neural Networks with Feature Interaction Learning for CTR Prediction. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. Virtual Event, Canada, 2166–2171.

[15] Matthias Feurer, Benjamin Letham, and Eytan Bakshy. 2018. Scalable meta-learning for Bayesian optimization. *arXiv preprint arXiv:1802.02219* (2018).

[16] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[17] Jerome H. Friedman, Trevor J. Hastie, and Robert Tibshirani. 2000. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics* 28, 2 (2000), 337–407.

[18] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*. Bellevue, WA, 513–520.

[19] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*. Melbourne, Australia, 1725–1731.

[20] James A Hanley and Barbara J McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143, 1 (1982), 29–36.

[21] Xiangnan He and Tat-Seng Chua. 2017. Neural Factorization Machines for Sparse Predictive Analytics. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. Shinjuku, Tokyo, Japan, 355–364.

[22] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. 2014. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising (ADKDD)*. New York City, New York, 5:1–5:9.

[23] Tin Kam Ho and Mitra Basu. 2002. Complexity measures of supervised classification problems. *IEEE transactions on pattern analysis and machine intelligence* 24, 3 (2002), 289–300.

[24] Jun Hu and Ping Li. 2017. Decoupled Collaborative Ranking. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*. Perth, Australia, 1321–1329.

[25] Jun Hu and Ping Li. 2018. Collaborative Filtering via Additive Ordinal Regression. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM)*. ACM, Marina Del Rey, CA, 243–251.

[26] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2014. An Efficient Approach for Assessing Hyperparameter Importance. In *Proceedings of the 31th International Conference on Machine Learning (ICML)*. Beijing, China, 754–762.

[27] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. 2013. Low-rank matrix completion using alternating minimization. In *Proceedings of the Symposium on Theory of Computing Conference (STOC)*. Palo Alto, CA, 665–674.

[28] Yu-Chin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware Factorization Machines for CTR Prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys)*. Boston, MA, 43–50.

[29] Ping Li. 2009. ABC-boost: adaptive base class boost for multi-class classification. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*. Montreal, Canada, 625–632.

[30] Ping Li. 2010. Robust LogitBoost and Adaptive Base Class (ABC) LogitBoost. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence (UAI)*. Catalina Island, CA, 302–311.

[31] Ping Li, Christopher J. C. Burges, and Qiang Wu. 2007. McRank: Learning to Rank Using Multiple Classification and Gradient Boosting. In *Advances in Neural Information Processing Systems (NIPS)*. Vancouver, Canada, 897–904.

[32] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval.* Cambridge University Press.

[33] 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 *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. Chicago, IL, 1222–1230.

[34] Swarup Medasani, Jaeseok Kim, and Raghu Krishnapuram. 1998. An overview of membership function generation techniques for pattern recognition. *Int. J. Approx. Reason.* 19, 3-4 (1998), 391–417.

[35] John J Murphy. 1999. *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications.* Penguin.

[36] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Trans. Knowl. Data Eng.* 22, 10 (2010), 1345–1359.

[37] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. 2007. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML)*. Corvallis, OR, 759–766.

[38] Steffen Rendle. 2010. Factorization Machines. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM)*. Sydney, Australia, 995–1000.

[39] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*. Banff, Alberta, Canada, 521–530.

[40] Rómer Rosales, Haibin Cheng, and Eren Manavoglu. 2012. Post-click conversion modeling and analysis for non-guaranteed delivery display advertising. In *Proceedings of the Fifth International Conference on Web Search and Web Data Mining (WSDM)*. Seattle, WA, 293–302.

[41] Jürgen Schmidhuber. 1992. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation* 4, 1 (1992), 131–139.

[42] Ming Shao, Carlos Castillo, Zhenghong Gu, and Yun Fu. 2012. Low-Rank Transfer Subspace Learning. In *Proceedings of the 12th IEEE International Conference on Data Mining (ICDM)*. Brussels, Belgium, 1104–1109.

[43] Maad Shatnawi and Nader Mohamed. 2012. Statistical techniques for online personalized advertising: a survey. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*. Riva, Trento, Italy, 680–687.

[44] Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. 2020. Fast Item Ranking under Neural Network based Measures. In *Proceedings of the Thirteenth ACM International Conference on Web Search and Data Mining (WSDM)*. Houston, TX.

[45] Sergios Theodoridis, Konstantinos Koutroumbas, and ST Koutroumbas. 2003. Clustering Algorithms III: schemes based on function optimization. *Pattern Recognition. Elsevier Academic Press, USA* (2003), 489–544.

[46] Sebastian Thrun and Lorien Y. Pratt. 1998. Learning to Learn: Introduction and Overview. In *Learning to Learn*. Springer, 3–17.

[47] Looja Tuladhar and Manish Satyapal Gupta. 2014. Click through rate prediction system and method. US Patent 8,738,436.

[48] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. 2003. Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis*. Springer, 91–109.

[49] 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, 12:1–12:7.

[50] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2015. Learning hyperparameter optimization initializations. In *Proceedings of the 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. Campus des Cordeliers, Paris, France, 1–10.

[51] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2018. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning* 107, 1 (2018), 43–78.

[52] David H Wolpert and William G Macready. 1997. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* 1, 1 (1997), 67–82.

[53] Lin Xiao. 2009. Dual Averaging Method for Regularized Stochastic Learning and Online Optimization. (2009), 2116–2124.

[54] Zhiqiang Xu, Dong Li, Weijie Zhao, Xing Shen, Tianbo Huang, Xiaoyun Li, and Ping Li. 2021. Agile and Accurate CTR Prediction Model Training for Massive-Scale Online Advertising Systems. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. Virtual Event, China, 2404–2409.

[55] Zhongwen Xu, Hado van Hasselt, and David Silver. 2018. Meta-Gradient Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*. Montréal, Canada, 2402–2413.

[56] Peng Yan, Xiaocong Zhou, and Yitao Duan. 2015. E-Commerce Item Recommendation Based on Field-aware Factorization Machine. In *Proceedings of the 2015 International ACM Recommender Systems Challenge (RecSys Challenge)*. Vienna, Austria, 2:1–2:4.

[57] Tan Yu, Yi Yang, Yi Li, Xiaodong Chen, Mingming Sun, and Ping Li. 2020. Combo-Attention Network for Baidu Video Advertising. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*. Virtual Event, CA, 2474–2482.

[58] Tan Yu, Yi Yang, Yi Li, Lin Liu, Hongliang Fei, and Ping Li. 2021. Heterogeneous Attention Network for Effective and Efficient Cross-modal Retrieval. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. Virtual Event, Canada, 1146–1156.

[59] Weizhong Zhao, Huifang Ma, and Qing He. 2009. Parallel K-Means Clustering Based on MapReduce. In *Proceedings of the First International Conference on Cloud Computing (CloudCom)*. Beijing, China, 674–679.

[60] Weijie Zhao, Shulong Tan, and Ping Li. 2020. SONG: Approximate Nearest Neighbor Search on GPU. In *Proceedings of the 36th IEEE International Conference on Data Engineering (ICDE)*. Dallas, TX, 1033–1044.

[61] 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. In *Proceedings of the 3rd Conference on Machine Learning and Systems (MLSys)*. Austin, TX.

[62] Weijie Zhao, Jingyuan Zhang, Deping Xie, Yulei Qian, Ronglai Jia, and Ping Li. 2019. AIBox: CTR Prediction Model Training on a Single Node. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*. Beijing, China, 319–328.

[63] Zhaohui Zheng, Hongyuan Zha, Tong Zhang, Olivier Chapelle, Keke Chen, and Gordon Sun. 2007. A General Boosting Method and its Application to Learning Ranking Functions for Web Search. In *Advances in Neural Information Processing Systems (NIPS)*. Vancouver, Canada, 1697–1704.

[64] Guorui Zhou, Xiaoqiang Zhu, Chengru 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. London, UK, 1059–1068.

[65] Zhixin Zhou, Shulong Tan, Zhaozhuo Xu, and Ping Li. 2019. Möbius Transformation for Fast Inner Product Search on Graph. In *Advances in Neural Information Processing Systems (NeurIPS)*. Vancouver, Canada, 8216–8227.