LPFS: Learnable Polarizing Feature Selection for Click-Through Rate Prediction

Yi Guo*
yguocuhk@gmail.com
KuaiShou Technology
Beijing, China

Zhaocheng Liu*
lio.h.zen@gmail.com
KuaiShou Technology
Beijing, China

Jianchao Tan
jianchaotan@kuaishou.com
KuaiShou Technology
Beijing, China

Chao Liao
liaochao@kuaishou.com
KuaiShou Technology
Beijing, China

Sen Yang
senyang.nlpr@gmail.com
KuaiShou Technology
Beijing, China

Lei Yuan
lyuan0388@gmail.com
KuaiShou Technology
Beijing, China

Dongying Kong
kongdongying@kuaishou.com
KuaiShou Technology
Beijing, China

Zhi Chen
chenzhi07@kuaishou.com
KuaiShou Technology
Beijing, China

Ji Liu
ji.liu.uwisc@gmail.com
KuaiShou Technology
Beijing, China

ABSTRACT

In industry, feature selection is a standard but necessary step to search for an optimal set of informative feature fields for efficient and effective training of deep Click-Through Rate (CTR) models. Most previous works measure the importance of feature fields by using their corresponding continuous weights from the model, then remove the feature fields with small weight values. However, removing many features that correspond to small but not exact zero weights will inevitably hurt model performance and not be friendly to hot-start model training. There is also no theoretical guarantee that the magnitude of weights can represent the importance, thus possibly leading to sub-optimal results if using these methods.

To tackle this problem, we propose a novel Learnable Polarizing Feature Selection (LPFS) method using a smoothed-ℓ^0 function in literature. Furthermore, we extend LPFS to LPFS++ by our newly designed smoothed-ℓ^0-liked function to select a more informative subset of features. LPFS and LPFS++ can be used as gates inserted at the input of the deep network to control the active and inactive state of each feature. When training is finished, some gates are **exact zero**, while others are around one, which is particularly favored by the practical hot-start training in the industry, due to no damage to the model performance before and after removing the features corresponding to **exact-zero** gates. Experiments show that our methods outperform others by a clear margin, and have achieved great A/B test results in KuaiShou Technology.

*Both authors contributed equally to this work.

Conference'17, July 2017, Washington, DC, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

KEYWORDS

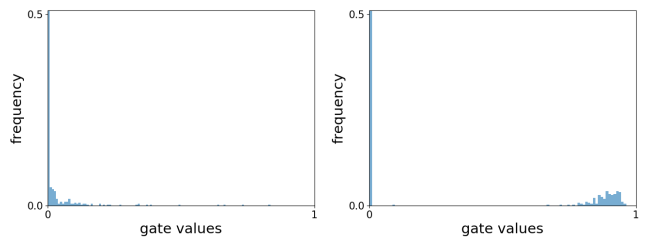
Click-Through Rate Prediction, Feature Selection, Smoothed-L0

ACM Reference Format:
Yi Guo, Zhaocheng Liu, Jianchao Tan, Chao Liao, Sen Yang, Lei Yuan, Dongying Kong, Zhi Chen, and Ji Liu. 2022. LPFS: Learnable Polarizing Feature Selection for Click-Through Rate Prediction. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Click-Through Rate (CTR) prediction, which aims to estimate the probability of a user clicking on an item, has become a crucial task in industrial applications, such as personalized recommendations and online advertising [28, 38, 39]. In recent years, significant progress has been made due to the development of deep learning [8, 10, 16, 31, 35], however, these deep models still require an effective set of feature fields as input. In industry, to accurately characterize user preferences, item characteristics, and contextual environments from different aspects, extensive feature engineering is generally essential for model training, which results in hundreds of feature fields in real-world datasets.

However, we can only feed a subset of feature fields instead of all feature fields into models for effective and efficient training. On the one hand, prior work [1, 30] points out that the subtle dependencies among relevant features and irrelevant features may inflate the error of parameter estimation, resulting in instability of prediction results. On the other hand, the online feature generation process consumes significant computing and storage resources. Therefore how to search for an optimal set of effective feature fields from the real-world dataset is the core concern of both academia and industry, considering both effectiveness and efficiency. Many feature selection methods have emerged in the past few decades. Some methods [2, 9, 18, 19, 21, 36] are proposed to do feature selection for the Logistic Regression (LR) model which is the classical CTR prediction model. For modern deep learning-based CTR prediction models, learning-based feature selection approaches have attracted



(a) Distribution of weights norm for LASSO (b) Distribution of gate values for LPFS

Figure 1: The density distribution for group LASSO with proximal-SGD optimization (left) and for LPFS. The group LASSO method is implemented on the first layer of the network, and proximal SGD is applied. LASSO method outputs a continuous distribution and we need to choose a small threshold to determine whether to remove or keep the features. While our LPFS method outputs a polarized distribution: some gate values are exact zeros, the others are distributed around 1. We can remove the feature fields with exact zero gates, and absorb the non-zero-gates into the embedding or the weight of the first fully connected layer, so as to cause no damage to the model and be friendly to hot-start training. Although there are also exact-zero gates using LASSO, the value of many non-zero gates are very close to zero.

much attention. Specifically, COLD [33] applies the Squeeze-and-Excitation (SE) block [12] to get the importance weights of features and select the most suitable ones. And some prior work [15] proposes to select informative features via LASSO, while FSCD [20] via Gumbel-Softmax-liked sampling method. More recently, UMEC [29] treats feature selection as a constrained optimization problem. Besides, permutation-based feature importance [25] measures the increase in the prediction error of the model after we permuted the feature's values, which also has been widely applied in real-world feature selection. Despite the significant progress made with these methods, some challenges demanding further explorations:

(1) **Getting rid of ad-hoc thresholding**. In general, previous methods output a *continuous distribution* of importance weights for feature fields to represent the feature importance. However, permutation-based, gumbel-softmax-based, or SE-based approaches cannot output **exact-zero** importance weights. They all need to prune from a skewed yet continuous histogram of importance weights, which makes the selection of the pruning threshold critical yet ad-hoc, taking Figure 1 for example.

(2) **Hot-start-friendly**. Empirically, feature selection usually needs a large range of data to train and generate confident feature selection results. To save training cost, the *hot-start* training is widely adopted in practice, which aims to inherit from the trained model and reduce the number of repetitions for training. The methods mentioned above remove many features with small but not exact zero importance weights. This step inevitably causes damage to the model's performance. Even for LASSO with the proximal optimization method, which can output exact zero weights, the remaining weights are very small and very close to zeros, which is unfriendly to hot-start. Last but not least, there is no theoretical guarantee that the magnitude of weights can represent the importance, leading to a suboptimal feature subset selected by these methods.

Inspired by the study of smoothed-ℓ^0 [7, 23, 24, 34] in compressed sensing, we consider feature selection as an optimization problem under ℓ^0-norm constraint, and propose a novel Learnable Polarizing Feature Selection (LPFS) method to effectively select highly informative features. We insert such differentiable function-based gates between the embedding and the input layer of the network to control the active and inactive state of these features. When training is finished, some gate values are **exact zero**, while others are distributed around one (see Figure 1). Then, we can remove feature fields with zero-gate, and absorb the non-zero gate to the embeddings or the weights of the first fully connected layer in the network. In this way, we can get rid of the threshold choosing, and there is totally no performance impact on the model performance before and after the physical removal, which is also very friendly to hot-start training.

Furthermore, each feature is unique, although maybe sometimes correlated, if a gate of a feature becomes zero accidentally, other features may not fully compensate for it. Then the model needs to have the ability to bring this feature to be active again. However, the derivative of all smoothed-ℓ^0 functions proposed by previous works at $x = 0$ is zero, which makes it impossible for gradient-based optimization methods to make a zero-gate become non-zero again. We are motivated to propose LPFS++ with a newly designed smoothed-ℓ^0-liked function to alleviate this problem.

We conduct extensive experiments to verify the effectiveness of two proposed approaches on both public datasets and large-scale industrial datasets. Experimental results demonstrate that LPFS++ outperforms LPFS, and both of them outperform previous approaches by a significant margin. Our approaches have also been deployed on the offline Kuaishou distributed training platform and have been exploited in different scenarios in Kuaishou to do feature selection for their CTR prediction models and have achieved significant A/B testing results.

We summarize the contributions of the proposed methods below:

- We are pioneers to apply the idea of smoothed-ℓ^0 gate to feature selection for CTR prediction and we propose a novel feature selection method by utilizing the closed-form proximal SGD updating for gate parameters, which outputs a polarized distribution of feature importance scores and is naturally friendly to hot-start training.
- We extend the idea of smoothed-ℓ^0 and propose LPFS++ which is based on a novel smoothed-ℓ^0-liked function and can select features more robustly.
- Experiments show that the both proposed methods outperform other feature selection methods by a clear margin, and they have achieved great benefits in KuaiShou Technology.

2 RELATED WORKS

2.1 Smoothed-ℓ^0 Optimization

The idea of smoothed-ℓ^0 optimization was first proposed in [22, 24] to obtain sparse solutions for under-determined systems of linear equations. The original optimization objective is $\min_x \|x\|_0$

s.t. $Ax = y$, where $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ and $m < n$. This is intractable, and they turn around to optimize $\min_x f_\epsilon(x) = 1 - \exp\left(\frac{-x^2}{2\epsilon^2}\right)$ s.t. $Ax = y$. When ϵ is large, $f_\epsilon(x)$ is smooth, while when ϵ is small enough, the function can approximate ℓ^0-norm indefinitely. So, they solve it iteratively, gradually decaying ϵ. And their experiments show that the method is much faster than ℓ^1-based methods, while achieving the same or better accuracy. Following work [23] studied the convergence properties of the above smoothed-ℓ^0 and find that under some mild constraints, the convergence is guaranteed. Afterwards, various smoothed-ℓ^0 functions had been proposed and studied for compressed sensing, such as $f_\epsilon(x) = \sin\left(\arctan\left(\frac{|x|}{\epsilon}\right)\right)$[32], $f_\epsilon(x) = \tanh\left(\frac{x^2}{2\epsilon^2}\right)$ [37] and $f_\epsilon(x) = \frac{x^2}{x^2+\epsilon^2}$ [34]. All these functions have following important property:

$$\lim_{\epsilon \to 0} g_\epsilon(x) = \begin{cases} 0, & x = 0 \\ 1, & x \neq 0 \end{cases} \tag{1}$$

and can approximate ℓ^0-norm well. In this paper, in addition to directly applying the existing smoothed-ℓ^0 function as gate to control feature selection, we also propose a new designed smoothed-ℓ^0-liked function for feature selection. We use the term "liked" here because what we proposed is an odd function, not an even function like all the above functions.

2.2 Feature Selection

Feature selection is a key component for CTR Prediction and various methods have been proposed in this area. [2, 9, 18, 19, 21, 36] are proposed to do feature selection for the Logistic Regression (LR) model which is the classical CTR prediction model. In the area of deep learning, COLD [33] use Squeeze-and-Excitation(SE) block [12] to measure the importance of features, while FSCD [20] use gumbel-softmax/sigmoid [13]. These two works both output a continuous importance distribution via softmax or sigmoid function, and the importance scores can not become exact zero. LASSO is also a common feature selection method in the industry [15]. With the proximal SGD [27] algorithm, the LASSO method can push a portion of parameters to be exact zeros, but their distribution is also continuous. All these methods require choosing a threshold to truncate the distribution. Recently, [29] use the alternating direction method of multipliers(ADMM) optimization method to select features, but we find it a little hard to compress a very large feature set into very small ones. In addition to this learning-based approach, feature permutation [25] is a common method in the industry. It first trains a model with all the features, and then keeps other features unchanged, permutes the input data along each feature axis randomly one by one, and uses the magnitude of the drop in performance as the metric for the importance of the features. This greedy method is easy to implement and does not require parameter tuning, but it ignores the possible correlation between features, and some features may compensate for each other.

Besides the direct feature selection, there is also some works on the interaction between features, such as AutoCross [36], Autofis [17], Autoint [31]. In essence, we can also regard the cross feature as a common feature, add it to the feature superset, and perform a general feature selection to discover which features should

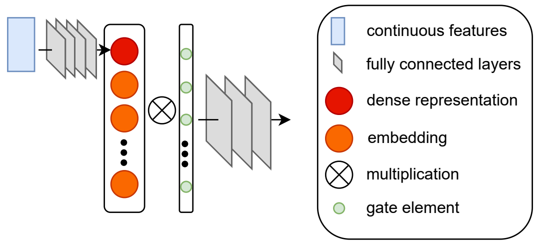


Figure 2: This figure takes DLRM [26] on the Terabyte dataset as an example to show where we insert the gate vector. In this experiment, the 13 continuous features are transformed by a *dense*-MLP to a dense representation, we treat the dense representation as a feature for the subsequent feature selection. Then the dense representation and the 26 categorical embeddings are concatenated as 27 features. We multiply the 27 features by a gate vector with 27 elements, where each element control the liveness of each feature. The multiplication results are the new input to the *top*-MLP in DLRM.

be interacted with. We will study it through an experiment in Section 4.4.

3 METHODS

3.1 Problem Formulation

For most deep-learning-based CTR prediction models, the input data is unusually collected in continuous and categorical forms. The continuous part can be fed into the network directly, while the categorical part is often pre-processed by mapping each category into a dense representation space [3], also known as embeddings. The embeddings and continuous features are then fed into the deep network. For description simplicity, we only describe the feature selection of categorical features in this paper. Suppose there are N feature fields concatenated as input $\boldsymbol{e} = [\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_N]$, each $\boldsymbol{e}_i, i = 1, 2, ..., N$ is an embedding vector for the i-th field, and we want to select a most informative subset from them. The network can be formulated as a function:

$$y = f(w; \boldsymbol{e}) \tag{2}$$

where w is all the network parameters, y the output of the network. To select features, we can insert a vector-valued function $\boldsymbol{g}(x) = [g(x_1), g(x_2), ..., g(x_N)]$ as gate before the input of the network, where each $x_i, i = 1, 2, ..., N$ can be either a new introduced learnable scalar parameter, or the weights norm from first fully connected layer, or even embedding information in some works [15]. Then the input can be viewed as $\tilde{\boldsymbol{e}} = [g(x_1)\boldsymbol{e}_1, g(x_2)\boldsymbol{e}_2, ..., g(x_N)\boldsymbol{e}_N]$. In this way, the network becomes:

$$y = f(w; \boldsymbol{g}(x)\boldsymbol{e}) \tag{3}$$

As shown in Figure 2, during training, the gates, embeddings, and all other network parameters are trained together, and we also impose some penalties on the gates parameters to implement feature selection. When training terminates, the features with zero-gates can be viewed as irrelevant features and can be removed, while others will be kept. Under this formulation, the key to feature selection becomes how to choose a good gate function. Previous works [15, 20] use lasso or gumble-softmax-trick [13] as the gate function, but these kinds of methods often output a continuous distribution of gates, then remove many feature fields with small but not exact zero gates. We don't think this is a good enough way and is unfriendly to hot-start training.

3.2 LPFS

From the perspective of optimization, feature selection can be essentially formulated as an optimization problem under ℓ^0-norm constraint. Since it's an NP-hard to solve directly, some previous works turn around it by relaxing ℓ^0 to ℓ^1, which is a convex function closest to ℓ^0. Inspired by the research field of smoothed-ℓ^0 optimization, we use the following smoothed-ℓ^0 function [34] as our gate for LPFS:

$$g_\epsilon(x) = \frac{x^2}{x^2+\epsilon} \tag{4}$$

This function has the following roperty:

$$g_\epsilon(x) \begin{cases} = 0, & x = 0 \\ \approx 1, & x \neq 0 \end{cases} \tag{5}$$

The derivative of $g_\epsilon(x)$ w.r.t x is:

$$g'_\epsilon(x) = \frac{2x\epsilon}{(x^2+\epsilon)^2} \tag{6}$$

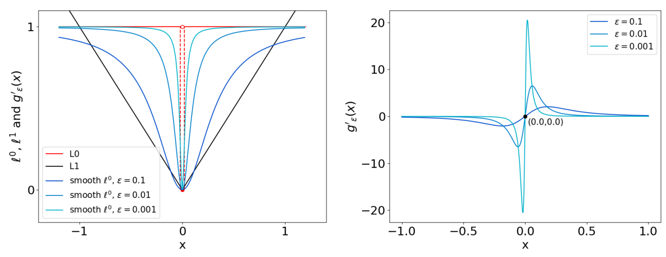
where we choose x here as newly introduced learnable parameters, ϵ is a small positive number. This is a quasi-convex function between ℓ^0 and ℓ^1 function, as shown in Figure 3. From the figure, we can see that when ϵ is large, $g_\epsilon(x)$ smooth and continuously differentiable and that when ϵ decays small enough, $g_\epsilon(x)$ can be infinitely close to ℓ^0. Different from Gumble-softmax/sigmoid with decreasing temperature, function (4) can be exactly zero even when ϵ is not small enough. And different from LASSO with proximal algorithms, function (4) outputs a polarized distribution with some gate values being exactly zeros, while others have large margins from zeros. These are two very nice properties for a hot start, as when we remove features with exactly zero gates, the performance of the model remains unchanged.

3.3 LPFS++

Although LPFS has achieved great performance both in open benchmarks and KuaiShou's online A/B tests (as shown in the experiment section), there is big room for improvement in the task of feature selection. GDP [11] applies the idea of smoothed-ℓ^0 in the field of channel pruning in Computer Vision. But feature selection is much different from channel pruning in the vision in two ways:

1. Channel pruning is essentially searching for the combination of the number of neurons for each layer. It only cares about the number of neurons in a certain layer, rather than which neurons. However, feature selection is different. We need to obtain a subset of features. In addition to caring about how many features this subset contains, it is more important to care about which features.

2. In the industry, user behavior is changing slowly. While mainstream datasets, such as ImageNet [4], are static, user data flows dynamically. This phenomenon requires the model to be more robust to feature selection than to channel pruning.



(a) ℓ^0, ℓ^1 and smoothed-ℓ^0 $g_\epsilon(x)$ (b) smoothed-ℓ^0 g'_ϵ with different ϵ

Figure 3: The graph of ℓ^0, ℓ^1 and smoothed-ℓ^0 $g_\epsilon(x)$ with different ϵ (left) and g'_ϵ (right). $g_\epsilon(x)$ a quasi-convex function between ℓ^0 and ℓ^1 function. When ϵ is large, $g_\epsilon(x)$ is smooth and continuously differentiable; while when ϵ is small enough, $g_\epsilon(x)$ can be infinitely close to ℓ^0. As the ϵ approaching to zero, $g'_\epsilon(x)$ is zero only at the points where $g_\epsilon(x)$ get exact zeros or near ones, otherwise infinity. Note that the $g'_\epsilon(x)|_{x=0.0}$ is exactly zero whatever ϵ is.

Mathematically, what the function (4) needs to be improved for feature selection is that its derivative at $x = 0$ is 0, as easy to see in Eq. (6). In this situation, the derivative of the output y of network w.r.t x, from Eq. (3), is

$$\left.\frac{\partial y}{\partial x}\right|_{x=0} = f'_2(w; \boldsymbol{g}_\epsilon(x)\boldsymbol{e})\boldsymbol{e}g'_\epsilon(x)|_{x=0} \tag{7}$$

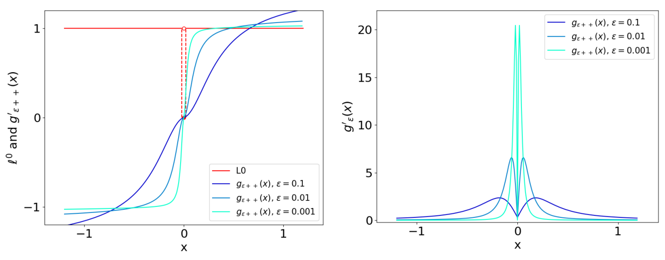
where $f'_2(a;b) = \frac{\partial f}{\partial b}$, the subscript 2 mean partial derivative w.r.t the second term. What the problem Eq. (7) will lead to is that whether some outlier samples or change in user behavior causes a gate value to go to zero accidentally, the corresponding feature will never be resurrected based on gradient-based optimization methods, and it could not be compensated by other features. One solution is to add some fading random noise, but we want to solve this from the gate function itself. We need to construct a gate function that satisfies properties similar to the smoothed-ℓ^0 function, and whose derivative is not zero at $x = 0$. One heuristic optional solution is:

$$g_{\epsilon++}(x) = \begin{cases} \frac{x^2}{x^2+\epsilon} + \alpha\epsilon^{1/\tau}\arctan(x), & x \geq 0 \\ -\frac{x^2}{x^2+\epsilon} + \alpha\epsilon^{1/\tau}\arctan(x), & x < 0 \end{cases} \tag{8}$$

Although it is a piece-wise function, it is also smooth and continuously differentiable. The derivative of $g_\epsilon(x)$ w.r.t x is

$$g'_{\epsilon++}(x) = \frac{2|x|\epsilon}{(x^2+\epsilon)^2} + \frac{\alpha\epsilon^{1/\tau}}{x^2+1} \tag{9}$$

Where x and ϵ have the same meaning as Eq. (4), α is a constant hyper-parameter balancing the two terms. The exponential factor $1/\tau$ is to control the decay rate of $g'_{\epsilon++}(x = 0)$ with respect to ϵ. This variable is pretty robust, thus we have taken $\tau = 2$ in all of previous private experiments in KuaiShou Technology, which worked very well. Also, the arctangent function is not essential, because what we need is just an odd function whose derivative at $x = 0$ is not zero, and whose value tends to be constant for large x. There are many other functions that can satisfy this properties, and might work better, which could be the future work. The graph is shown as Figure 4. Note that, unlike all smoothed-ℓ^0 functions,



(a) ℓ^0 and $g_{\epsilon++}(x)$ (b) $g'_{\epsilon++}$ with different ϵ

Figure 4: The graph of ℓ^0 and$g_{\epsilon++}(x)$ with different ϵ (left) and $g'_{\epsilon++}$ (right). Now $g_{\epsilon++}(x)$ is odd, not even like the smoothed ℓ^0 function. Note that the gradient at $x = 0$ is no longer zero, but a small number decays with ϵ.

Function (8) is an odd function rather than an even function. This is a necessary consequence because the derivative of a continuously differentiable even function at $x = 0$ must be 0. It is fine to use an odd function as the gate, because we can absorb the negative signs of gates values into the corresponding embeddings or the first fully connected layer of the network. That is, in Eq. (3) the equality holds: $\boldsymbol{g}(x)\boldsymbol{e} = abs(\boldsymbol{g}(x))sign(\boldsymbol{g}(x))\boldsymbol{e} = abs(\boldsymbol{g}(x))\boldsymbol{e'}$, where $\boldsymbol{e'} = sign(\boldsymbol{g}(x))\boldsymbol{e}$ is the final embeddings for downstream tasks. Now we have $g'_{\epsilon++}(x=0) = \alpha\epsilon^{1/\tau}$, which is small number decays with ϵ. Note that $g'_{\epsilon++}(x=0) \neq 0$ does not mean $\frac{\partial y}{\partial x}\Big|_{x=0} \neq 0$ because of the existence of $f'_2(w; \boldsymbol{g}_{\epsilon}(x)\boldsymbol{e})$ in Eq. (7). During the offline feature selection, ϵ initializes a large value so that the $g'_{\epsilon++}(x=0)$ is robust enough to outlier samples and the slow change of user behavior. As training goes on, $g'_{\epsilon++}(x=0)$ also decays as ϵ decays, so that the feature superset can be stably divided into non-informative and informative subset when ϵ decays to be small enough.

3.4 Optimization

For both LPFS and LPFS++, we optimize the following objective function:

$$\min_{w,x} \mathcal{F}(w;x) = \mathcal{L}(\hat{y}; f(w; \boldsymbol{g}(x)\boldsymbol{e})) + \lambda \|x\|_1 \tag{10}$$

Where $f(w; \boldsymbol{g}(x)\boldsymbol{e}))$ represents the network, same to Eq. (3); $\hat{y}$ is the ground true(i.e. users click or not click the item); $\mathcal{L}(\cdot;\cdot)$ is the loss function between the ground truth and model prediction; $\|\cdot\|_1$ is ℓ^1-norm; λ is the balance factor. w and $\boldsymbol{e}$ in $\mathcal{L}(\hat{y}; f(w; \boldsymbol{g}(x)\boldsymbol{e}))$ are updated by Adam [14] or Adagrad [6] optimizer, same to baseline. And x is updated by proximal-SGD. It is equivalent to solve $\min_x\{\frac{1}{2\eta}||x - \tilde{x}_t||^2 + \lambda\|x\|_1\}$, whose closed-form solution is:

$$x^{t+1} = \begin{cases} \tilde{x}_t - \lambda\eta, & \tilde{x}_t \geq \lambda\eta \\ 0, & -\lambda\eta < \tilde{x}_t < \lambda\eta, \\ \tilde{x}_t + \lambda\eta, & \tilde{x}_t \leq -\lambda\eta \end{cases} \tag{11}$$

Where η is learning rate for x, $\tilde{x}_t$ is the value of x after one step of updating $\mathcal{L}(\hat{y}; f(w; \boldsymbol{g}(x)\boldsymbol{e}))$ with the Momentum optimizer. The core code for updating x is shown in supplementary material.
It is worth pointing out that we use ℓ^1-norm instead of ℓ^0-norm to regularize x in Eq.(10). In fact, the polarization effect (ℓ^0 property) that we're looking for is derived from smoothed-ℓ^0 gate in Eq. (4), not from ℓ^1 regularization on x, the ℓ^1 regularization is only to penalize x to let smoothed-ℓ^0 gate become polarized like ℓ^0.

Moreover, for ℓ^0 regularization, the problem $min_x\{\frac{1}{2\eta}||x - \tilde{x}_t||^2 + \lambda\|x\|_0\}$ indeed has closed-form solutions:

$$x^{t+1} = \begin{cases} 0, & |\tilde{x}_t| < \sqrt{2\lambda\eta} \\ \tilde{x}_t, & |\tilde{x}_t| > \sqrt{2\lambda\eta} \\ 0 \text{ or } \tilde{x}_t, & |\tilde{x}_t| = \sqrt{2\lambda\eta} \end{cases} \tag{12}$$

but we find it is very sensitive to the initial value of x in Eq. (10) in our experiments. If we initialize x to be greater than $\sqrt{2\lambda\eta}$ accidentally, it can hardly be penalized during the whole training process under the second case in Eq. (12); otherwise it becomes zero very quickly under the first case in Eq. (12). Additionally, in each SGD iteration, $\tilde{x}_t$ is not necessarily optimal so it does not need to be directly updated by Eq. (12) to be zero. While in Eq. (11), x will be penalized by a small step $\lambda\eta$ whatever its initial value is.

4 EXPERIMENTS

In this section, we will demonstrate the superiority of our method through three experiments. We mainly describe the core part of the experiments here, and the training details such as hyper-parameter setting are left at the end of this paper.

4.1 Datasets

To show that our method can filter out the highly informative features, we conducted experiments on a large-scale dataset Criteo AI Labs Ad Terabyte dataset [1] and an industrial dataset in KuaiShou. For Terabyte dataset, it contains 26 categorical features and 13 continuous features. It contains about 4.4 billion click log samples over 24 days. Similar to DLRM [26], for negative samples, we randomly select 12.5% on each day. To be fair, for all the public methods and our method, We pre-train the model by "day 0 ∼ 17", and select features by "day 18 ∼ 22", in which process, all the model parameters, including newly introduced parameters for FSCD [20] and our methods, are trained together for learnable methods. When the feature subsets are selected by these methods, the models are then trained from scratch by "day 0 ∼ 22" using this subset, then evaluated by "day 23" and the best AUC calculated in official DLRM [26] code [2] is reported.

The industrial dataset is collected by Mobile Kuaishou App, we select 9 days for offline features selections, and take 10% of the negative samples at random. All the positive and negative click log samples add up to nearly 1.2 billion over the 9 days. This dataset contains 250 user categorical feature fields, 46 item categorical feature fields, 96 combine categorical feature fields, and 25 continuous features. Similar to the configuration of Terabyte, we pre-train the model by "day 1 ∼ 6", and select features by "day 7 ∼ 8", in which process all the parameters are trained together. When obtaining feature subsets, the models are trained by cold start by "day 1 ∼ 8" using these subsets, then are evaluated by "day 9".

[1]https://labs.criteo.com/2013/12/download-terabyte-click-logs/
[2]https://github.com/facebookresearch/dlrm

4.2 Network Settings

For the public Terabyte dataset, we made some minor modifications to the DLRM [26] to serve as our baseline network. The 13 continuous features in DLRM are transformed by an MLP to a dense representation with the same length as embeddings. Then we treat this dense representation as an embedding, with the same status as the other 26 embedding features. In this way, we have 27 features in the feature superset. Since the 27 features are still very small and the subset of features selected by various methods is similar, we also conducted another experiment with all the crossed features in addition to this direct selection, which will be explained in the following subsections. Same to [29], we set the hidden dimensions of the three-layer MLP prediction model as 256 and 128 for both crossed and non-crossed versions.

For the industrial dataset, every categorical feature is 16 dimension, while the different dense feature has different dimensions, ranging from 1 to 128. Every user feature field and every item feature field is crossed by element-wise multiplication. Then the crossed features, combine features, and continuous features are concatenated as the input for the network. The network contains a share bottom layer, then some auxiliary branches for different auxiliary tasks. We focus on one main branch.

4.3 Terabyte Without Cross Features

In this experiment, we treat the 27 features(including a dense representation transformed by 13 dense features, as described in the last subsection) as a feature superset for feature selection. Each feature is 16 dimensions. These 27 features are multiplied by the corresponding gates, then concatenated as a vector, and fed into the three-layer MLP prediction branch (or called the *top*-MLP in the original paper), as shown in Figure 2. We compared our method with FSCD [20], UMEC [29], feature permutation, and group LASSO. For UMEC, all the hyper-parameters are set as the paper reported; for FSCD, we set all the regularization weights to the same values, for group LASSO, we regularize the weights of the first fully connected layer in the *top*-MLP, and train them by proximal-SGD; for feature permutation, we randomly shuffle the feature to be evaluated in a mini-batch.

The result is shown in Figure 5. When only a small number of features are supposed to be removed, we find in experiments that all these methods select almost the same subsets, so there is little difference in AUC obtained by these methods (look at the upper right corner of Figure 5). While when we want to remove about a half, where the number of combinations becomes much larger($C_{27}^{14} \approx 2 \times 10^7$), our method can cope with this challenge well, so as to select the most informative subset (look at the left half of Figure 5). The feature subset our method selects is attached in the supplementary material for future reference. Although we compare AUC rather than ACC here for UMEC, the ACC for UMEC in our experiment is still higher than the result in the original paper under the same number of remaining features. In the following experiments for a large feature sets, we do not compare with UMEC, partly because of its poor performance here and partly because it is difficult to compress to a very small subset.

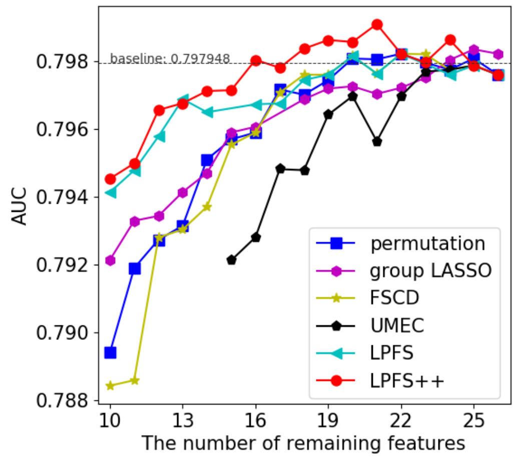


Figure 5: Performance comparison of our method with other methods. Where the abscissa is large(i.e. only a few features are removed), there is little difference between all these methods. We find in experiments that the feature subsets selected by these methods are almost the same. However, where the abscissa is small (i.e. around half of the features are removed), our method has clear superiority over other methods. The baseline for 27 features is best AUC 0.797948, best ACC 0.811079, best Loss 0.423315.

4.4 Terabyte With Cross Features

In this experiment, we take the experiment a step further, partly because 27 features were so few, and partly because we want to show that our method can be easy to be applied to study feature interactions. In this experiment, every two embeddings corresponding to two different feature fields are crossed by element-wise multiplication, then the original features and the crossed features are concatenated as the input of the *top*-MLP of DLRM. So, the total number of features can be viewed as $27 + C_{27}^{2} = 378$, and we treat these 378 features equally to select from this much larger superset. So, if an original feature should be removed, the features that interact with it are not necessarily removed. Except for the cross-feature and the resulting larger number of input units for the first fully connected layer of *top*-MLP of DLRM, other experiment configurations are the same as the last subsection. In this way, we can both select first and second-order features. The result is shown in Figure 6. Compared with previous works [17, 36], our methods can also be used to select higher-order features effectively.

4.5 Industrial Dataset

In this experiment, we apply our methods to a large-scale industrial dataset. All the 392 categorical feature fields (250 users, 46 items, 96 combined) mentioned above are treated equally for the feature selection, although the network does not treat these features equally. Due to the existence of cross features between user and item, even if the same number of features is kept, the computational amount may not be the same. Still, we only care about the feature subsets, rather than the computation cost. Different from the experiment 4.4, we insert gates immediately after the embeddings, rather than after the

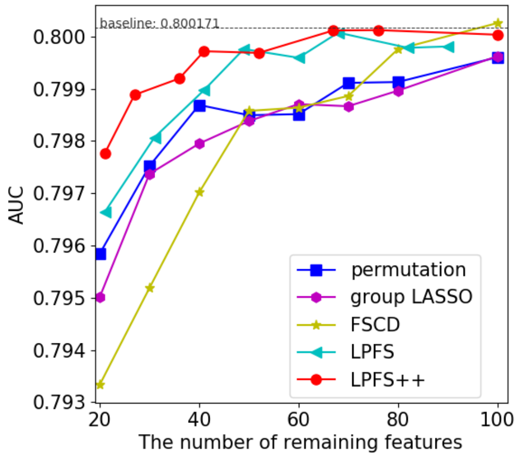


Figure 6: Performance comparison for Terabyte with cross features. Every two different feature embeddings are crossed by element-wise multiplication. Then the original first-order features and the crossed second-order features are concatenated as the input of the *top*-MLP of DLRM. So, the total number of features can be viewed as $27 + C_{27}^{2} = 378$. We can see that our methods, LPFS and LPFS++, have many advantages over other methods, especially when the number of remaining features is small. It can be seen that our methods can also be used to mine higher-order features.

crossed features. So, when a feature is supposed to be removed, all cross-features that interact with it will also be removed. The result is shown in Figure 7. As can be seen from the figure, our method has more obvious advantages in industrial large-scale datasets and challenging complex network structures.

4.6 Ablation Studies and Experimental Analysis

The influence of $\epsilon's$ decaying rate on performance has been studied by [11] for channel pruning, and similar to it, the decaying rate of ϵ in (4) and (8) is not much important for offline feature selection, and our experiments found that the final value of ϵ between $1e-4$ and $1e-8$ has no significant effect on the performance of the model. The key component of the gate function (8) for LPFS++ is the second arctangent function part and its balancing factor α. For simplicity, we will focus on α.

By Function (9), since the derivative at $x = 0$ is $g'_{\epsilon++}(x = 0) = \alpha\epsilon^{1/\tau}$, we should fix the schedule of ϵ first before studying α. In the industrial experiments, we decay ϵ by 0.986 every 500 steps, and the minimal value is $1.0e-4$. As shown in Figure 8, in general, the larger the α value, the more informative the selected feature subset. This is to be expected, a larger value of α results in a larger value of the derivative $g'_{\epsilon++}(x = 0) = \alpha\epsilon^{1/\tau}$ and thus a greater fault tolerance of the model. We also find in experiments that a larger α will result in a smaller number of features to be removed. when other hyper-parameters kept unchanged. Therefore, in order to select roughly the same number of features, when increasing the value of α, the value of λ in Function (10) must also be increased

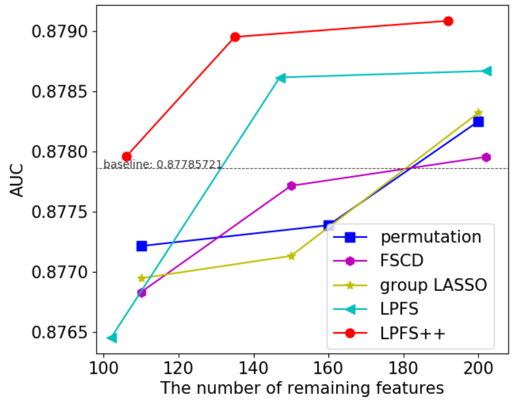


Figure 7: Performance comparison of our methods with other methods for KuaiShou industrial dataset. The total number of feature field is 392 (250 user, 46 item, 96 combine), and we treat all these features equally. In the network, every user feature and every item feature is interacted by element-wise multiplication, and there are five auxiliary tasks to help improve the performance of the main task. Our LPFS++ has more obvious advantages in industrial large-scale datasets and challenging network structure.

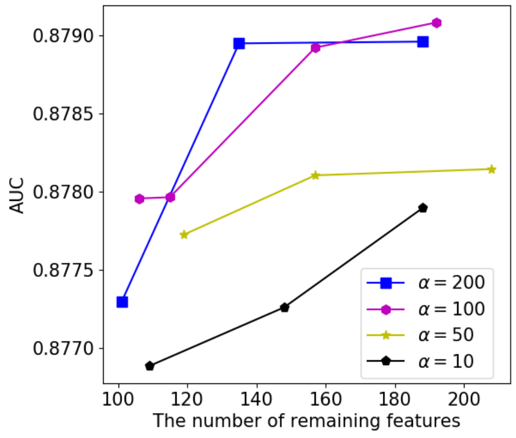


Figure 8: Ablation study for different α. We can see that in general, the larger the α value, the more informative the selected feature subset.

accordingly, as shown in Table 1. This is also to be expected, because a larger value of α makes the model more resistant to compression. Figure 9 shows the number of features with zero-gate changes over the training step, and it contains a wealth of interesting information. First of all, let's look at the beginning of the training, where zero-gates first appear (i.e. the ordinate of the curve becomes non-zero for the first time). In our experiments, all the x in Function (4) and (8) are initialized to 1.0. So, at the beginning of training, the first case (i.e. $\tilde{x}_t \geq 2\lambda\eta$) in (11) dominates, and x will be subtracted by $2\lambda\eta$ at each step. Then as the training goes on, x becomes zero gradually, and the speed at which it becomes zero is related to the value of $2\lambda\eta$. In Figure 9, we tune the α and λ so that the final number of

α	10	50	100	200
λ	0.004	0.0066	0.013	0.02
# feats	109	119	106	101

Table 1: The table shows the relationship between α and λ when about the same number of features are wanted to be kept. "# feats" means the number of remaining features. We can see that when we increase α, the λ should also be increase if we want to remain about the same number of features.

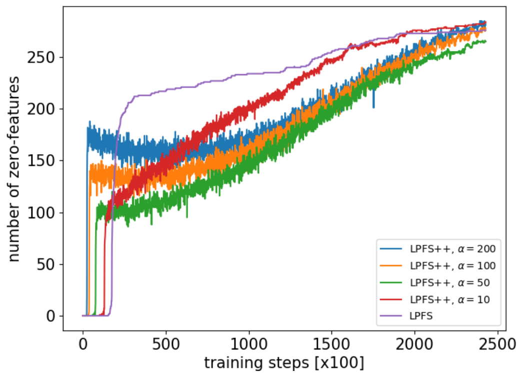


Figure 9: The number of features with zero-gate changes over training step. The X-axis represents training steps (we record every 100 steps, so the total number of training steps is about 250000), while the Y-axis represents the number of features with zero-gate.

removed features is about the same. And it shows that zero gates occur earlier in the model with larger λ and α. Second, in the middle of training, as the value of α increases, the oscillating degree of the curve also increases. This is also as expected, since increasing α increases the $g'_{\epsilon++}(x=0) = \alpha\epsilon^{1/\tau}$, giving more features a chance to revive.

Third, near the end of the training, all the curves gradually stop oscillating, because $g'_{\epsilon++}(x=0) = \alpha\epsilon^{1/\tau}$ and the learning rate is decaying.

4.7 Online Experiments

The proposed LPFS and LPFS++ have been successfully deployed on the offline Kuaishou distributed training platform and exploited by dozens of advertising scenarios in Kuaishou. To show the effectiveness of our proposed methods, we choose our first launch as an example, in which nothing changed except for the input feature fields. The dataset is collected from KuaiShou APP and the feature superset contains 332 feature fields due to the extensive feature engineering, from which we select 177 features for subsequent experiments. We found that the feature subsets selected across different runs are highly overlapping, which shows that our method is robust. We first conduct an offline experiment on the dataset which is sampled from 14 days online logs. Our model relatively outperforms permutation feature importance and the previous online feature set by 0.209% and 0.304% in offline AUC, respectively. We conducted an online A/B test for two weeks in July 2021. Compared with the previous state-of-the-art method used online, our method increased the cumulative revenue [5] by 1.058% and the social welfare [5] by 1.046%, during the A/B test of 10% traffic.

4.8 Complexity Analysis

As complexity analysis is important for industrial recommender systems, we do a simple complexity analysis in this section. Suppose there are total N features, we need to initialize a N-dimensional learnable vector $\boldsymbol{x}$ in equation (3), which consumes $4N$ bytes storage. Then the intermediate variables include $g(\boldsymbol{x})$, as well as the gradient and the momentum of $\boldsymbol{x}$ during the optimization process, approximately occupying a total of $4 \times 4N$ bytes memory, which is negligible compared with the storage of the whole network and the embeddings. Moreover, we found in many experiments that the increase of the training time in each iteration is also negligible.

In our experiments, if the training of deep CTR model with the feature superset is cold-started, it needs to be pre-trained for a while until almost converged, using T_1 time, and we then load this pre-trained checkpoint to perform feature superset pruning to obtain the best feature subset, using T_2 time. Then we find that T_2 is usually about $\frac{1}{4}T_1$ in our all experiments. Of course, if we do feature selection based on an online model checkpoint, this pre-training step can be omitted for this case.

4.9 Experiment details

In order to facilitate readers to reproduce and use our method, we describe our experiment in detail.

In all the experiments, x in Function (8) and (4) is initialized to 1.0, while ϵ 0.1. It can be calculated that the initial value of Function (8) and (4) is 0.909 and $0.909 + 0.785\alpha\epsilon$. When we get a pre-trained model, in order to ensure that there is no sudden change in the performance of the model before and after inserting the gates, we divide all gate functions by their initial value, so that the initial value of the gate function is 1.0. Besides, if we only do feature selection for categorical features, rather than continuous features, the magnitude of categorical feature embeddings will change quickly compared with that of continuous representations, and this phenomenon is especially evident for LPFS++. So, we find that it would be better to divide the categorical feature embedding by an overall number, which can be the root mean square of gate values. This number does not participate in gradient backpropagation, and can be updated every 100 steps for example, then keep fixed after enough training steps. We found that this step resulted in a slight improvement in LPFS++ performance.

For Terabyte, we did not shuffle the dataset, and we train in chronological order to sense changes in user behavior to show the robustness of LPFS++. We use Adagrad optimizer for the model parameters (not including the gate parameters), and the batch size is 512, and the learning rate is 0.01. We decay ϵ by 0.9978 every 100 steps, and the minimum value is $1.e-5$. We use proximal-SGD with momentum to train x in Function (8) and (4), the initial learning rate is 0.01 for LPFS++, 0.005 for LPFS and decayed by 0.9991 every 100 steps, but not exceed $5e-4$. Different α and λ values are combined to tune the parameters to obtain feature subsets of different sizes.

For the industrial dataset, we also use Adagrad optimizer for the model parameters (not including the gate parameters), and the batch size is 1024, learning rate is initial as 0.01 then exponentially rises to 0.1 very slowly and stays constant. We decay ϵ by 0.986 every 500 steps, with minimal value $1e-4$.

5 CONCLUSION, LIMITATION AND FUTURE WORK

In this paper, we adopted the idea of smoothed-ℓ^0 to feature selection and proposed a new smoothed-ℓ^0-liked function to select features more effectively and robustly. Both LPFS and LPFS++ show superiority over other methods, and LPFS++ achieves state-of-the-art performance. LPFS and LPFS++ have played an important role as efficient feature selection plugin tools for recommendation scenarios in Kuaishou Technology.

The main limitation for function (8), from the perspective of dimensional analysis, is that it is not scale-invariant. The x in the arctangent trigonometric function arctan means that x must be dimensionless. For function (4), ϵ has the dimension of x squared. If we want to construct a gate function that is just a dimensionless coefficient, then the derivative must have the dimension of the form $\sim \frac{1}{x}$ or $\sim \frac{x}{\epsilon}$. When $x = 0$ and ϵ is small enough, the $\sim \frac{1}{x}$ tends to be infinity, $\sim \frac{x}{\epsilon}$ tends to be strictly zero, and it is impossible to have finite non-zero derivative values at $x = 0$. Thus we have to assume that both x and ϵ are dimensionless, and the consequence that function (8) is not scale-invariant is inevitable. Obviously, the solution is not unique to the problem, and there are many smoothed-ℓ^0-liked functions that satisfy the derivative being non-zero and decaying as ϵ decaying. Other than function (8), we didn't try any other similar property functions yet.

In fact, LPFS++ can be easily extended to online feature selection. What we want is that some features can be active or inactive dynamically as user behavior changes, and then predict whether a user will click on an item using active features in real-time. Our LPFS++ can meet well with this requirement. Specially, we can maintain a superset of features to train and select features simultaneously, so that all the model parameters and gate parameters are trained together online. Then, the features in the active status and their corresponding sub-networks can become effective on a specialized inference platform in real-time. Online feature selection enables the model to capture the temporal changes in user behaviors and select the optimal feature subset dynamically and adaptively in real-time. This involves both algorithmic, training, and inference engineering improvements, which are left as future work.

A APPENDIX

A.1 Feature mask for Terabyte dataset

We list the feature subsets that we selected by LPFS++ for reference. In the following table 2, the "#" column is the number of remaining features, the "mask" column indicates the feature subsets selection. In the mask, "0" means the corresponding feature should be removed, while "1" means kept. The length of every mask is 27, where the first value represents the 13 continuous features, and the following 26 values represent the 26 categorical feature fields. The order of the feature fields is the same as the official code of DLRM [26].

#	mask	#	mask
26	111110111111111111111111111	17	111100010111110101010101011
25	111100111111111111111111111	16	111100010111110001010101011
24	111100111111111111011111111	15	111100010011110101000101011
23	111100011111111111011111111	14	111100010011110001000101011
22	111100011111111101011111111	13	111100010011110001000001011
21	111100110111110101011111111	12	111100010011110000000001011
20	111100010111110101011111111	11	111100010101110000000001001
19	111100010111110101011101111	10	101100010100110000010101000
18	111100010111110101011101011		

Table 2: Feature mask for Terabyte dataset.

A.2 Core code

```
import torch

def lpfs_pp(x, epsilon, alpha=100, tao=2, init_val=1.0):
    '''
    The gate function for LPFS++ (equation 8) in paper.
    '''
    g1 = x*x/(x*x + epsilon)
    g2 = alpha * epsilon**(1.0/tao) * torch.atan(x)
    g = torch.where(x > 0, g2+g1, g2-g1)/init_val
    return g

def train(model, optimizer_model, optimizer_gate, lam, train_dataloader, criterion):
    '''
    model: the model we want to train and select features (the lpfs_pp function is behind the input features)
    optimizer_model: the optimizer for all the original model parameters, not including gate parameters
    optimizer_gate: the optimizer for gate parameters
    lam: the coefficient for the L1 regulation of x (equation 10) in paper.
    train_dataloader: dataloader for trianing set.
    criterion: original training loss function.
    '''
    p = optimizer_gate.param_groups[0]["params"][0]  # get gate parameters, i.e. x in paper
    for data, label in train_dataloader:
        optimizer_gate.zero_grad()  # clear grad
        optimizer_model.zero_grad()  # clear grad
        output = model(data)  # forward the model
        loss = criterion(output, label)  # compute the loss
        loss.back_forward()  # back-propagation
        optimizer_model.step()
        optimizer_gate.step()
        # the following code is for the proximal-L1 algorithm (equation 11) in paper
        thr = lam*lr # lr is the original learning rate.
        in1 = p.data > thr
        in2 = p.data < -thr
        in3 = ~(in1 | in2)
        p.data[in1] -= thr
        p.data[in2] += thr
        p.data[in3] = 0.0
```

Figure 10: Our LPFS++ can be implemented within several lines of code using PyTorch. To demonstrate the reproducibility, we list the core code of LPFS++ and the proximal-SGD updating here.

REFERENCES

[1] Hyojin Bahng, Sanghyuk Chun, Sangdoo Yun, Jaegul Choo, and Seong Joon Oh. 2020. Learning de-biased representations with biased representations. In *International Conference on Machine Learning*. PMLR, 528–539.

[2] Olivier Chapelle, Eren Manavoglu, and Romer Rosales. 2014. Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 4 (2014), 1–34.

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

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[5] Chao Du, Zhifeng Gao, Shuo Yuan, Lining Gao, Ziyan Li, Yifan Zeng, Xiaoqiang Zhu, Jian Xu, Kun Gai, and Kuang-Chih Lee. 2021. Exploration in Online Advertising Systems with Deep Uncertainty-Aware Learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2792–2801.

[6] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12, 7 (2011).

[7] Armin Eftekhari, Massoud Babaie-Zadeh, Christian Jutten, and Hamid Abrishami Moghaddam. 2009. Robust-SL0 for stable sparse representation in noisy settings. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 3433–3436.

[8] 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*. 2166–2171.

[9] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2010. A note on the group lasso and a sparse group lasso. *arXiv preprint arXiv:1001.0736* (2010).

[10] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv preprint arXiv:1703.04247* (2017).

[11] Yi Guo, Huan Yuan, Jianchao Tan, Zhangyang Wang, Sen Yang, and Ji Liu. 2021. Gdp: Stabilized neural network pruning via gates with differentiable polarization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5239–5250.

[12] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7132–7141.

[13] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Yifeng Li, Chih-Yu Chen, and Wyeth W Wasserman. 2016. Deep feature selection: theory and application to identify enhancers and promoters. *Journal of Computational Biology* 23, 5 (2016), 322–336.

[16] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1754–1763.

[17] Bin Liu, Chenxu Zhu, Guilin Li, Weinan Zhang, Jincai Lai, Ruiming Tang, Xiuqiang He, Zhenguo Li, and Yong Yu. 2020. Autofis: Automatic feature interaction selection in factorization models for click-through rate prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2636–2645.

[18] Qiang Liu, Zhaocheng Liu, Haoli Zhang, Yuntian Chen, and Jun Zhu. 2021. Mining Cross Features for Financial Credit Risk Assessment. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 1069–1078.

[19] Zhaocheng Liu, Qiang Liu, Haoli Zhang, and Yuntian Chen. 2020. DNN2LR: Interpretation-inspired Feature Crossing for Real-world Tabular Data. *arXiv preprint arXiv:2008.09775* (2020).

[20] Xu Ma, Pengjie Wang, Hui Zhao, Shaoguo Liu, Chuhan Zhao, Wei Lin, Kuang-Chih Lee, Jian Xu, and Bo Zheng. 2021. Towards a Better Tradeoff between Effectiveness and Efficiency in Pre-Ranking: A Learnable Feature Selection based Approach. *arXiv preprint arXiv:2105.07706* (2021).

[21] Lukas Meier, Sara Van De Geer, and Peter Bühlmann. 2008. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 70, 1 (2008), 53–71.

[22] G Hosein Mohimani, Massoud Babaie-Zadeh, and Christian Jutten. 2007. Fast sparse representation based on smoothed ℓ_0 norm. In *International Conference on Independent Component Analysis and Signal Separation*. Springer, 389–396.

[23] H Mohimani, M Babaie-Zadeh, M Gorodnitsky, and C Jutten. 2010. Sparse recovery using smoothed L0 norm (SL0): convergence analysis. *arXiv preprint cs.IT/1001.5073* (2010).

[24] Hosein Mohimani, Massoud Babaie-Zadeh, and Christian Jutten. 2008. A fast approach for overcomplete sparse decomposition based on smoothed ℓ^0 norm. *IEEE Transactions on Signal Processing* 57, 1 (2008), 289–301.

[25] Christoph Molnar. 2020. *Interpretable machine learning*. Lulu. com.

[26] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[27] Atsushi Nitanda. 2014. Stochastic proximal gradient descent with acceleration techniques. *Advances in Neural Information Processing Systems* 27 (2014).

[28] 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*. 521–530.

[29] Jiayi Shen, Haotao Wang, Shupeng Gui, Jianchao Tan, Zhangyang Wang, and Ji Liu. 2020. UMEC: Unified model and embedding compression for efficient recommendation systems. In *International Conference on Learning Representations*.

[30] Zheyan Shen, Peng Cui, Tong Zhang, and Kun Kunag. 2020. Stable learning via sample reweighting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5692–5699.

[31] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2019. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1161–1170.

[32] Linyu Wang, Junyan Wang, Jianhong Xiang, and Huihui Yue. 2019. A re-weighted smoothed-norm regularized sparse reconstructed algorithm for linear inverse problems. *Journal of Physics Communications* 3, 7 (2019), 075004.

[33] Zhe Wang, Liqin Zhao, Biye Jiang, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2020. Cold: Towards the next generation of pre-ranking system. *arXiv preprint arXiv:2007.16122* (2020).

[34] Jianhong Xiang, Huihui Yue, Xiangjun Yin, and Linyu Wang. 2019. A New Smoothed L0 Regularization Approach for Sparse Signal Recovery. *Mathematical Problems in Engineering* 2019 (2019).

[35] Feng Yu, Zhaocheng Liu, Qiang Liu, Haoli Zhang, Shu Wu, and Liang Wang. 2020. Deep Interaction Machine: A Simple but Effective Model for High-order Feature Interactions. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2285–2288.

[36] Luo Yuanfei, Wang Mengshuo, Zhou Hao, Yao Quanming, Tu Weiwei, Chen Yuqiang, Yang Qiang, and Dai Wenyuan. 2019. AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications. *ACM* (2019).

[37] Ruizhen Zhao, Wanjuan Lin, Hao Li, and S Hu. 2012. Reconstruction algorithm for compressive sensing based on smoothed L0 norm and revised newton method. *Journal of Computer-Aided Design and Computer Graphics* 24, 4 (2012), 478–484.

[38] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 5941–5948.

[39] 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 *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1059–1068.