FLEN: Leveraging Field for Scalable CTR Prediction

Wenqiang Chen
Data Intelligence, Meitu Inc
Xiamen, China
wenqiang.cwq@gmail.com

Lizhang Zhan
Advertisement Recommendation Platform, Tencent Inc.
Shenzhen, China
lizhangzhan@tencent.com

Yuanlong Ci
Data Intelligence, Meitu Inc
Xiamen, China
cyl4@meitu.com

Minghua Yang
Data Intelligence, Meitu Inc
Xiamen, China
jenniyang@meitu.com

Chen Lin*
Xiamen University
Xiamen, China
chenlin@xmu.edu.cn

Dugang Liu*
Shenzhen University
Shenzhen, China
dugang.ldg@gmail.com

ABSTRACT

Click-**T**hrough **R**ate (CTR) prediction systems are usually based on multi-field categorical features, i.e., every feature is categorical and belongs to one and only one field. Modeling feature conjunctions is crucial for CTR prediction accuracy. However, it usually requires a massive number of parameters to explicitly model all feature conjunctions, which is not scalable for real-world production systems.

In this paper, we describe a novel **F**ield-**L**everaged **E**mbedding **N**etwork (**FLEN**) which has been deployed in the commercial recommender systems in Meitu and serves the main traffic. FLEN devises a field-wise bi-interaction pooling technique. By suitably exploiting field information, the field-wise bi-interaction pooling layer captures both inter-field and intra-field feature conjunctions with a small number of model parameters and an acceptable time complexity for industrial applications. We show that some classic shallow CTR models can be regarded as special cases of this technique, i.e., MF, FM and FwFM. We identify a unique challenge in this technique, i.e., the FM module in our model may suffer from the coupled gradient issue, which will damage the performance of the model. To solve this challenge, we develop **Dicefactor**: a novel dropout method to prevent independent latent features from co-adapting.

Extensive experiments, including offline evaluations and online A/B testing on real production systems, demonstrate the effectiveness and efficiency of FLEN against the state-of-the-art models. In particular, compared to the previous version deployed on the system (i.e. NFM), FLEN has obtained 5.19% improvement on CTR with 1/6 of memory usage and computation time.

CCS CONCEPTS

• **Information systems → Computational advertising**;

KEYWORDS

Click-through rate, Inter-field, Intra-field, Dropout

ACM Reference Format:
Wenqiang Chen, Lizhang Zhan, Yuanlong Ci, Minghua Yang, Chen Lin, and Dugang Liu. 2020. FLEN: Leveraging Field for Scalable CTR Prediction. In *Proceedings of the 2nd International Workshop on Deep Learning Practice for High-Dimensional Sparse Data (DLP '20), August 24, 2020, San Diego, California USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3306307.3328180

1 INTRODUCTION

Click-**T**hrough **R**ate (CTR) prediction is the task of predicting the probabilities of users clicking items or advertisements (ads). It is a critical problem in recommender systems and online advertising, which provide substantial revenue for Internet companies. As such, CTR prediction has attracted much attention from both academia and industry communities in the past few years [1, 6, 14, 29, 31].

The data in CTR prediction task is *multi-field categorical data*, i.e., every feature is categorical and belongs to one and only one field. For example, feature "gender=Female" belongs to field "gender", feature "age=24" belongs to field "age" and feature "item category=cosmetics" belongs to field "item category". The value of feature "gender" is either "male" or"female. Feature "age" is discretized to several age groups: "0-18", "18-25", "25-30", and so on. It is well regarded that, feature conjunctions are essential for accurate CTR prediction [3, 4, 7, 8, 11]. An example of informative feature conjunctions is: age group "18-25" combined with gender "female" for item category "cosmetics". It indicates that young girls are more likely to click on cosmetic products.

Modeling sparse feature conjunctions has been continually improved and refined by a number of prior work. Most models follow the Factorization Machines (FM) [19] and its inspired extensions because of its effectiveness and flexibility. For example, FFM [9] and FwFM [16, 30] explicitly model field-aware feature interactions, NFFM [25] combines FFM and MLP to capture operation-aware feature conjunctions with additional parameters, and FPENN [12] estimates the probability distribution of the field-aware embedding rather than using the single point estimation (the maximum a posteriori estimation). Although these models have achieved promising results, *the challenge in real-world online advertising or recommender systems is the strict latency limit at serving time and the scalability for high-dimensionality of features.* We need to predict hundreds of

*Co-corresponding authors

DLP-KDD 2020, August 24, 2020, San Diego, California, USA

ACM ISBN 978-1-4503-6317-4/19/07…$15.00
https://doi.org/10.1145/3306307.3328180

items for each user in less than 10 milliseconds. The model complexity of FFM and FwFM is $O(N^2)$, where N is the number of features. The drawback of directly applying FFM and FwFM in real-world applications is the dramatically increased use of computational resources, because any uniform increase in the number of features will cause a quadratic increase of computation. FFM is also restricted by space complexity, which further weakens its practicality. FFM-based deep models use additional parameters to capture non-linear high-order feature conjunctions. However, increasing model complexity sometimes only marginally improve performance while leading to severe over-fitting problems [8]. In addition, these works usually consume huge memory, resulting in restricted scalability.

In this paper we describe a novel **F**ield-**L**everaged **E**mbedding **N**etwork (FLEN) which has been successfully deployed in the online recommender systems in Meitu, serving the main traffic. FLEN devises a new operation in neural network modeling — Field-wise Bilinear Interaction (FwBI) pooling, to address the restriction of time and space complexity when applying field-aware feature conjunctions in real industrial system. The field-wise bi-interaction pooling technique is based on the observation that features from various fields interact with each other differently. We show that some classic shallow CTR models can be regarded as special cases of this technique, including FM [19], MF [10], and FwFM [16]. The combination of this technique and the traditional MLP layer constitutes our final model. The idea behind our model is to use multiple modules to extract feature interactions at different levels, and finally merge them to get a better representation of the interaction. The parameters in each part are only responsible for a certain level of feature interaction, which helps reduce the parameters of the model and speed up the training of the model.

As noted in previous work [17], FM may cause the coupled gradient issue due to using the same latent vectors in different types of inter-field interactions, i.e. two supposedly independent features are updated in the same direction during the gradient update process. In FLEN, this issue is partly tackled by leveraging field information. We also propose a novel dropout method: Dicefactor to decouple independent features. Dicefactor randomly drops bi-linear paths (i.e. cross-feature edge in FM module of the field-wise bi-interaction pooling layer) to prevent a feature from adapting to other features.

To demonstrate the effectiveness and efficiency of FLEN, we conduct extensive offline evaluations and online A/B testing. In offline evaluations, FLEN outperforms state-of-the-art methods on both a well-known benchmark and an industrial dataset consisting of historical click records collected in our system. Online A/B testing shows that FLEN enhances the CTR prediction accuracy (i.e. increases CTR by 5.19%) with a fraction of computation resources (i.e. 1/6 memory usage and computation time), compared with the last version of our ranking system (i.e. NFM [8]).

2 RELATED WORK

CTR prediction has been extensively studied in the literature, as online advertising systems have become the financial backbone of most Internet companies. Related literature can be roughly categorized into shallow and deep models.

2.1 Shallow Models

Successful shallow models represent features as latent vectors. For example, matrix factorization (MF) [10] is successfully applied in recommender systems. It factorizes a rating matrix into a product of lower-dimensional sub-matrix, where each sub-matrix is the latent feature space for users and items. Factorization machines (FM) [19] is a well-known model to learn feature interactions. In FM, the effect of feature conjunction is explicitly modeled by inner product of two latent feature vectors (a.k.a. embedding vectors). Many variants have been proposed based on FM. For example, Rendle et al. [18] proposed a context-aware CTR prediction method which factorized a three-way $< user, ad, context >$ tensor. Oentaryo et al. [15] developed hierarchical importance-aware factorization machine to model dynamic impacts of ads.

Field information has been acknowledged as crucial in CTR prediction. A number of recent work has exploited field information. For example, Field-aware Factorization Machines (FFM) [9] represents a feature based on separate latent vectors, depending on the multiplying feature field. GBFM [2] and AFM [24] consider the importance of different field feature interactions. Field-weighted Factorization Machines (FwFM) [16] assigns interaction weights on each field pair.

The field-wise bi-interaction technique of FLEN is inspired by FwFM. It can be viewed as a special case of factorized FwFM in a computationally efficient manner. However, the field-wise bi-interaction technique of FLEN generalizes and ensembles MF, FM and FwFM. Furthermore, shallow models are limited as they focus on modeling linear, low-order feature interactions. FLEN is capable of capturing not only low-order but also high-order, nonlinear interactions.

2.2 Deep Models

An increased interest in designing deep models for CTR prediction has emerged in recent years. The majority of them utilize feature bi-interactions. To name a few, NFM [8] generalizes FM by stacking neural network layers on top of a bi-interaction pooling layer. The architecture of DeepFM [7] resembles with Wide&Deep [3], which also has a shared raw feature input to both its "wide" (i.e. for bi-interaction) and "deep" (i.e. for high-order interaction) components. DCN [22] learns certain bounded-degree feature interactions. xDeepFM [11] improves over DeepFM and DCN by generating feature interactions in an explicit fashion and at the field-wise level. NFFM [25] learns different feature representations for convolutional operations and product operations. However, the space complexity of NFFM and the time complexity of xDeepFM restrict them from applying in industrial systems. FGCNN [13] leverages the strength of CNN to generate local patterns and recombines them to generate new features. Then deep classifier is built upon the augmented feature space. PIN [17] generalizes the kernel product of feature bi-interactions in a net-in-net architecture.

The rest of literature learns the high-order feature interactions in an implicit way, e.g. PNN [17], FNN [26], DeepCrossing [20], and so on. Some tree-based methods [23, 27] combine the power of embedding-based models and tree-based models to boost explainability. One drawback of these approaches is having to break training procedure into multiple stages. A recent work FPENN [12]

also groups feature embedding vectors based on field information in deep neural network structure. It estimates the probability distribution of the field-aware embedding rather than using the single point estimation (the maximum a posteriori estimation) to prevent overfitting. However, as FPENN assigns several latent vectors to each field (i.e. one for a field which is not equivalent as the multiplying field), it requires much more model parameters than FLEN.

3 MODEL

An overview of the model architecture is illustrated in Figure 1. Let $\mathcal{X} = \{\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^s, \cdots\}$ denote the set of samples, $\mathbf{x}^s = [\mathbf{x}_1^s, \cdots, \mathbf{x}_N^s]$ denotes the s-th sample, and $\mathbf{x}_n^s \in \mathcal{R}^{K_n}$ denotes the n-th categorical feature in the sample $\mathbf{x}^s$ (details described in Section 3.1). Suppose FLEN takes $\mathbf{x}^s$ as input. First, all features pass through an embedding layer, which outputs a concatenation of field-wise feature embedding vectors $\mathbf{e}^s = [\mathbf{e}_1^s, \cdots, \mathbf{e}_M^s]$, where $\mathbf{e}_m^s \in \mathcal{R}^{K_e}$ denotes the m-th hierarchical fields (details described in Section 3.2). Then the embedding vectors flow a Field-wise Bi-Interaction pooling layer (FwBI) and an MLP component. The field-wise bi-interaction pooling layer (Sec 3.3) consists of three submodules which capture all single and field-wise feature interactions (degree one or two), and outputs $\mathbf{h}_{FwBI} \in \mathcal{R}^{K_e+1}$. The MLP component captures non-linear, high-order feature interactions (details described in Section 3.5). The output of field-wise bi-interaction pooling layer and output of MLP component are concatenated to feed the last prediction layer (Section 3.6).

We will show that previous CTR prediction models such as MF [10], FM [19] and FwFM [16] can be expressed and generalized under the proposed framework. Alternatively, the FwBI layer can be regarded as a combination of single feature, MF-based inter-field feature interactions and FM-based intra-field feature interactions. The above three parts together with the MLP layer that captures non-linear, high-order feature interactions form our model. The idea behind our model is to use multiple modules to extract feature interactions at different levels, and finally merge them to get a better representation of the interaction. The parameters in each part are only responsible for a certain level of feature interaction, which helps reduce the parameters of the model and speed up the training of the model. Furthermore,the parallel structures of FwBI allows the computational budget be distributed in a distributed environment.

Hereafter, unless stated otherwise, we use lower-case letters for indices, upper-case letters for universal constants, lower-case bold-face letters for vectors and upper-case bold-face letters for matrices, calligraphic letters for sets. We use square brackets to denote elements in a vector or a matrix, e.g. $x[j]$ denotes the j−th elements of $\mathbf{x}$. We will omit superscripts whenever no ambiguity results.

3.1 Feature Representation

It is natural to represent categorical features as one-hot or multi-hot vectors. Suppose there are N unique features and each feature $\mathbf{x}_n$ has K_n unique values, we represent each instance as $\mathbf{x} = [\mathbf{x}_1, \cdots, \mathbf{x}_N]$, where $\mathbf{x}_n \in \mathcal{R}^{K_n}$, $\mathbf{x}_n[j] \in \{0, 1\}, j = 1, \cdots, K_n$. $\sum_{j=1}^{K_n} \mathbf{x}_n[j] = k$. Vector $\mathbf{x}_n$ with $k = 1$ refers to one-hot encoding and $k > 1$ refers to multi-hot encoding.

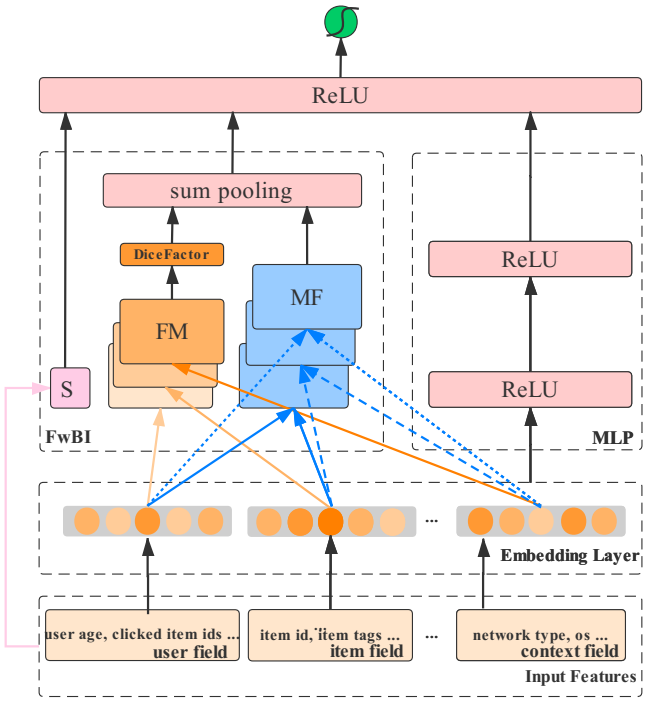


Figure 1: Architecture overview of FLEN

Suppose there are M fields, $F(n)$ denotes the field of feature $\mathbf{x}_n$, we organize the feature representations in a field-wise manner for complexity reduction. Specifically, $\mathbf{x} = concat(\mathbf{x}_1, \cdots, \mathbf{x}_M)$, where $\mathbf{x}_m$ is the concatenation of feature vectors in field m, i.e. $\mathbf{x}_m = concat(\mathbf{x}_n | F(n) = m)$. The organized instance is illustrated in the example below. Note that there is a hierarchical structure of fields. For example, "item tags field" and "item id field" belong to the more general "item field". In practice, inspired by YouTube's work [4], we also classify features according to whether they describe properties of the item or properties of the user/context (namely user field, item field and context field as illustrated in Figure 1) and achieve maximal performance enhancement and complexity reduction. A practically useful aspect of this hierarchical design is that it aligns with the intuition that the output of each hierarchical field is highly inter-correlated.

$$\underbrace{\underbrace{[0,1,0,...,0]}_{\textit{age field}} \ldots \underbrace{[\quad 1,0 \quad]}_{\textit{gender field}}}_{\textit{user field}} \underbrace{\underbrace{[0,1,0,...,0]}_{\textit{item id field}} \underbrace{[0,1,0,1,...,0]}_{\textit{item tags field}}}_{\textit{item field}}$$

3.2 Embedding Layer

Since the feature representations of the categorical features are very sparse and high-dimensional, we employ an embedding procedure to transform them into low dimensional, dense real-value vectors. Firstly, we transform each feature $\mathbf{x}_n$ to $\mathbf{e}_n$.

$$\mathbf{e}_n = \mathbf{V}_n \mathbf{x}_n, \tag{1}$$

where $\mathbf{V}_n \in \mathcal{R}^{K_e \times K_n}$ is an embedding matrix for the corresponding feature that will be optimized together with other parameters in the network. Note that feature size can be various.

Next we apply sum-pooling to $\mathbf{e}_n$ to obtain the field-wise embedding vectors.

$$\mathbf{e}_m = \sum_{n|F(n)=m} \mathbf{e}_n \tag{2}$$

Finally, we concatenate all field-wise embedding vectors to build $\mathbf{e}$, as illustrated in Figure 2.

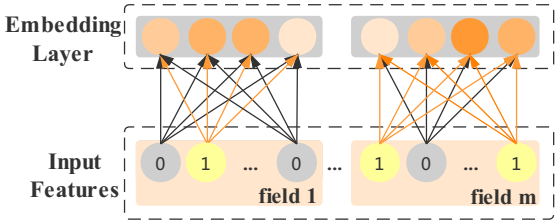


Figure 2: Illustration of embedding layer with $K_e = 4$.

3.3 Field-wise Bi-Interaction Pooling Layer

The field-wise bi-interaction pooling layer learns a mapping $\Phi_{FwBI}(\mathbf{W}_S, \mathbf{R}, \mathbf{W}_{FwBI}) : (\mathcal{R}^N, \mathcal{R}^{MK_e}) \rightarrow \mathcal{R}^{K_e+1}$. There are three submodules in this layer.

The first sub-module (denoted as S) is a linear regression part similar to that of FWFM, which models global bias of data and weight of categorical features. It operates on the categorical vectors, i.e. $\mathbf{h}_S = w_0 + \sum_{i=1}^{N} \sum_{j=1}^{K_i} w_i[j]x_i[j]$ where w_0 is bias term. Note that for ease of description, we let $\mathbf{W}_S$ denote the set of all $w_0 \cup w_i$.

The second sub-module is called the MF module, which focuses on learning inter-field feature interactions between each pair of the hierarchical fields. It first operates element-wise product on all pairs of field-wise embedding vectors, i.e. $\mathbf{h}_{MF} = \sum_{i=1}^{M} \sum_{j=i+1}^{M} \mathbf{e}_i \odot \mathbf{e}_j r[i][j]$, where $r[i][j] \in \mathcal{R}$ is a weight to model the interaction strength between field i and j. We use $\odot$ to denote the element-wise product of two vectors, that is, $(\boldsymbol{e}_i \odot \boldsymbol{e}_j)[k] = \mathbf{e}_i[k]\mathbf{e}_j[k]$.

In real industrial system, the quantity of feature fields is usually 10 or more, for example there are 33 feature fields in our industrial dataset, but the hierarchical field number, M, is usually less than 4 [4, 29] for reducing computation and avoiding overfitting. This hierarchical field manner is inspired by YouTube, according to whether they describe properties of the item or properties of the user/context [4]. As shown in Figure 1, there are 3 MF models to learn field-wise feature interactions for each pair of the hierarchical user, item and context fields, respectively.

The third sub-module is called the FM module, which focuses on learning intra-field feature interactions in each FM module. It first computes self element-wise product on each field embedding vectors, i.e. $\mathbf{hf}_m = \mathbf{e}_m \odot \mathbf{e}_m$. Similar operations are also conducted on each feature embedding vectors, i.e. $\mathbf{ht}_m = \sum_{n,F(n)=m} \mathbf{e}_n \odot \mathbf{e}_n$. Finally, the two vectors are merged over all field-wise subtraction of the two intermediate vectors, i.e. $\mathbf{h}_{FM} = \sum_m (\mathbf{hf}_m - \mathbf{ht}_m) r[m][m]$, where $r[m][m] \in \mathcal{R}$ is a weight for each field m, discriminating the importance of each field that contribute to the final prediction.

Note that for ease of description, we let $\mathbf{R}$ denote the set of all $r[i][j] \cup r[m][m]$.

Clearly, the output of MF module and FM module is a K_e-dimension vector that encodes the inter-field and intra-field feature interactions in the embedding space, respectively.

We concatenate the output of S and the sum pooling of MF and FM module, i.e. $\mathbf{h}_{in} = [\mathbf{h}_S, \mathbf{h}_{MF} + \mathbf{h}_{FM}]$. Then $\mathbf{h}_{in}$ is fed to a hidden layer, i.e. $\mathbf{h}_{FwBI} = \sigma(\mathbf{W}_{FwBI}^T \mathbf{h}_{in})$. In practice, we use the ReLU as the active function σ.

By leveraging multiple MF and FM modules to learn both the inter-field and intra-field feature interactions, we end up with more disentangled parameters and therefore with faster training. Furthermore, the parallel structures of FwBI allows the computational budget be distributed in a distributed environment.

It is worth pointing out that the FwBI pooling layer is much more memory-efficient than FFM and furthermore, it can be efficiently computed in $O(MK_eN + K_eM^2)$. In real industrial system, the quantity of feature fields is usually 10 or more, for example there are 33 feature fields in our industrial dataset, but the hierarchical field number, M, is usually less than 4 [4, 29]. There are 3 hierarchical field, i.e. user, item and context, in our industrial dataset. When $M \ll N$, FwBI can be efficiently trained and serverd online in linear time, which is very attractive in industrial systems.

3.4 Relation to Previous CTR Prediction Systems

A variety of shallow models can be expressed and generalized under the field-wise bi-interaction technique. To start with, we set the active function σ as an identify function, the weight matrix $\mathbf{W}_{FwBI}^T = \mathbf{I}$, where $\mathbf{I}$ is a unit matrix. Since the embedding vectors $\mathbf{e}_m, \mathbf{e}_n$ are transformed from the original feature representations, $\mathbf{e}_m = \sum_{n|F(n)=m} \mathbf{e}_n$, $\mathbf{e}_n = \mathbf{V}_n \mathbf{x}_n$, we can re-write the computation in field-wise bi-interaction pooling layer as:

$$\begin{aligned}\Phi_{FwBI} = \mathbf{W}_{FwBI}^T \Big[& \underbrace{w_0 + \sum_{i=1}^{N} \sum_{j=1}^{K_i} w_i[j]x_i[j]}_{S} \\ & \underbrace{, \sum_{i=1}^{M} \sum_{j=i+1}^{M} \Big[\Big(\sum_{n,F(n)=i} \mathbf{V}_i \mathbf{x}_n \Big) \Big(\sum_{n,F(n)=j} \mathbf{V}_j \mathbf{x}_n \Big) r[i][j] \Big]}_{MF} \\ & \underbrace{+ \sum_{m}^{M} \Big[\Big(\sum_{n,F(n)=m} \mathbf{V}_n \mathbf{x}_n \Big)^2 - \Big(\sum_{n,F(n)=m} (\mathbf{V}_n \mathbf{x}_n)^2 \Big) \Big] r[m][m] \Big]}_{FM}\end{aligned} \tag{3}$$

where we use the symbol $(\mathbf{Vx})^2$ to denote $\mathbf{Vx} \odot \mathbf{Vx}$. The first term in Equation 3 corresponds to the first sub-module which is based on single feature values. The second term corresponds to the sum pooling over the second sub-module, which resembles the matrix factorization form when written in feature representations, and the third sub-module, which resembles the factorization machine form.

Clearly, if there are only one field, i.e., $M = 1$ and $r_{m,m} = \frac{1}{2}$, then we can exactly recover the FM model.

$$\begin{aligned}&\Phi_{FwBI} = \Phi_{FM}\\&= \left[w_0 + \sum_{i=1}^{N}\sum_{j=1}^{K_i} w_i[j]x_i[j], \frac{1}{2}[(\sum_{i=1}^{N}\mathbf{V}_i\mathbf{x}_i)^2 - \sum_{i=1}^{N}(\mathbf{V}_i\mathbf{x}_i)^2]\right]\end{aligned} \tag{4}$$

If there are N fields, i.e., $M = N$, there is not any intra-field feature interactions, then we can recover the FwFM model.

$$\begin{aligned}&\Phi_{FwBI} = \Phi_{FwFM}\\&= \left[w_0 + \sum_{i=1}^{M}\sum_{j=1}^{K_i} w_i[j]x_i[j], \sum_{i=1}^{M}\sum_{j>i}^{M}(\mathbf{V}_i\mathbf{x}_i)(\mathbf{V}_j\mathbf{x}_j)r[i][j]\right]\end{aligned} \tag{5}$$

3.5 MLP component

We employ an MLP component to capture non-linear, high-order feature interactions. The input is simply a concatenation of all field-wise embedding vectors, i.e. $\mathbf{h}_0 = concat(\mathbf{e}_1, \cdots, \mathbf{e}_M)$. A stack of fully connected layers is constructed on the input $\mathbf{h}_0$. Formally, the definition of fully connected layers are as follows:

$$\begin{aligned}\mathbf{h}_1 &= \sigma_1(\mathbf{W}_1\mathbf{h}_0 + \mathbf{b}_1),\\\mathbf{h}_2 &= \sigma_2(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2),\\&\cdots\cdots\\\mathbf{h}_L &= \sigma_L(\mathbf{W}_L\mathbf{h}_{L-1} + \mathbf{b}_L),\end{aligned} \tag{6}$$

where L denotes the number of hidden layers, $\mathbf{W}_l$, $\mathbf{b}_l$ and σ_l denote the weight matrix, bias vector and activation function for the l-th layer, respectively. We use ReLU as the active function for each layer. Note that for ease of description, we let $\mathbf{W}_{MLP}$ denote the set of all $\mathbf{W}_l$, and $\mathbf{b}_{MLP}$ denote the set of all $\mathbf{b}_l$.

3.6 Prediction Layer

The output vector of the last hidden MLP layer $\mathbf{h}_L$ is concatenated with the output vector of field-wise bi-interaction pooling layer Φ_{FwBI} to form $\mathbf{h}_F = concat(\mathbf{h}_{FwBI}, \mathbf{h}_L)$. The concatenation $\mathbf{h}_F$ goes through one last hidden layer and is transformed to

$$z = \sigma(\mathbf{w}_F^T\mathbf{h}_F), \tag{7}$$

where vector $\mathbf{w}_F$ denotes the neuron weights of the final hidden layer.

Finally, we apply a sigmoid layer to make predictions.

$$\sigma(\mathbf{z}) = \frac{1}{1 + e^{-z}} \tag{8}$$

Our loss function is *negative log-likelihood*, which is defined as follows:

$$\mathbb{L} = -\frac{1}{|\mathcal{X}|}\sum_{s=1}^{|\mathcal{X}|}(y^s log(\Phi(\mathbf{x}^s)) + (1 - y^s)log(1 - \Phi(\mathbf{x}^s))), \tag{9}$$

where $y^s \in \{0, 1\}$ as the label, $\Phi(\mathbf{x}^s)$ is the output of the network, representing the estimated probability of the instance $\mathbf{x}^s$ being clicked. The parameters to learn in our model are represented as $\mathbf{V}, \mathbf{W} = \{\mathbf{W}_S, \mathbf{W}_{FwBI}, \mathbf{W}_{MLP}, \mathbf{w}_F\}, \mathbf{b}_{MLP}, \mathbf{R}$, which are updated via minimizing the total negative log-likelihood using gradient descent.

4 DICEFACTOR: DROPOUT METHOD

As noted in previous work [17], FM may cause the coupled gradient issue because it uses the same latent vectors in different types of inter-field interactions, i.e. two supposedly independent features are updated in the same direction during the gradient update process. It will damage the performance of the model. To solve this challenge, we propose a novel dropout method to decouple independent features, i.e., Dicefactor. Dicefactor is inspired by Dropout [21]. It randomly drops bi-linear paths (i.e. cross-feature edge in the FM module of field-wise bi-interaction pooling layer) to prevent a feature from adapting to other features.

We first reformulate Eq. (4) as:

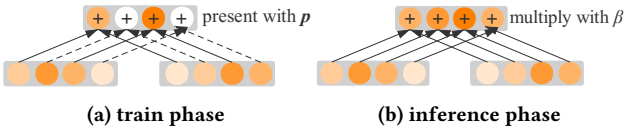
$$\Phi_{FM} = \left[w_0 + \sum_{i=1}^{N}\sum_{j=1}^{K_i} w_i[j]x_i[j], \sum_{i=1}^{M}\sum_{j=1\&i\neq j}^{M}\mathbf{e}_i \odot \mathbf{e}_j\right], \tag{10}$$

where $\mathbf{e}_i$ represents the i-th field's embedding vector. Based on Eq. (10), Fig. 3 shows the expanding structure of the bi-linear interaction in two field embedding vectors. There are K_e bi-linear paths, each bi-linear path connects corresponding elements in two embedding vectors. The key idea of DiceFactor is to randomly drop the bi-linear paths during the training. This partly prevents $\mathbf{e}_i$ from co-adapting to $\mathbf{e}_j$, i.e., $\mathbf{e}_i$ is updated through the direction of $\mathbf{e}_j$.

In our implementation, each factor is retained with a predefined probability β during training. With the DiceFactor, the formulation of bi-linear interaction of FM part in the training becomes:

$$\Phi_{FM} = \left[w_0 + \sum_{i=1}^{N}\sum_{j=1}^{K_i} w_i[j]x_i[j], \sum_{i=1}^{M}\sum_{j=1\&i\neq j}^{M}\mathbf{p}\mathbf{e}_i \odot \mathbf{e}_j\right], \tag{11}$$

where $\mathbf{p} \in \mathcal{R}^{K_e}$ and $p[i] \sim Bernoulli(\beta)$. With the DiceFactor, the network can be seen as a set of 2^{K_e} thinned networks with shared weights. In each iteration, one thinned network is sampled randomly and trained by back-propagation as shown in Fig. 3a.



(a) train phase **(b) inference phase**

Figure 3: Dicefactor keeps a bi-linear path with probability β at train phase. At inference phase, every bi-linear path is kept and the output is multiplied by β.

For inference, instead of explicitly averaging the outputs from all 2^{K_e} thinned networks, we use the approximate "Mean Network" scheme in [21]. As shown in Fig. 3b, each factor term $\mathbf{e}_i \odot \mathbf{e}_j$ is multiplied by β at inference phase:

$$\Phi_{FM} = \left[w_0 + \sum_{i=1}^{N}\sum_{j=1}^{K_i} w_i[j]x_i[j], \sum_{i=1}^{M}\sum_{j=1\&i\neq j}^{M}\beta\mathbf{e}_i \odot \mathbf{e}_j\right] \tag{12}$$

In this way, the output of each neuron at inference phase is the same as the expectation of output of 2^{K_e} different networks at train phase.

5 OFFLINE MODEL EVALUATION

To assess the validity of our CTR prediction models, we run a traditional offline evaluation based on a well-known public benchmark and an industrial dataset.

Avazu[1] The first dataset we adopt is originally used in the Kaggle CTR prediction competition. It contains users' mobile behaviors, i.e. whether a displayed mobile ad is clicked by a user. It has 23 feature fields spanning from user/device features to ad attributes. The data set is collected during a time span of 10 days. We use 9 days of clicks for training and the last 1 day of data for test.

Note that in this paper we do not use another well-known benchmark, i.e., Criteo dataset, because the semantic of its features is undisclosed. We do not known which hierarchical field each feature belongs to.

Meitu The second dataset used to run the experiments is a uniformly generated sample of our internal historical data. As a training set we extract a sample of 2 million items shown during a seven-day time period from 2019-07-26 to 2019-08-01 to users of a photo and video-sharing social networking app, namely Meitu. We collect over 1.5 billion users' records. The test set contains a sample of 1 million items shown the next day, i.e. 2019-08-02. There are around 5 million features (e.g., user age, clicked feed ids, and etc) organized in hierarchical field manner, including "user", "item" and "context". Features used in our system are described in Table 1.

Table 1: Example features in Meitu dataset.

Field	Feature	Dimemsionality	Type	AverageNonzero Ids per Instance
User Field	gender	2	one-hot	1
	age	~ 10	one-hot	1
	clicked_feed_ids	$\sim 10^7$	multi-hot	$\sim 10^2$
	liked_feed_ids	$\sim 10^5$	multi-hot	$\sim 10^2$
Item Field	feed_id	$\sim 10^7$	one-hot	1
	tags	$\sim 10^3$	multi-hot	$\sim 10^1$
	clicked_rate	10	one-hot	10
	liked_rate	10	one-hot	10
	author_id	$\sim 10^7$	one-hot	1
	author_gender	2	one-hot	1
	author_age	~ 10	one-hot	1
	author_clicked_rate	10	one-hot	10
	author_liked_rate	10	one-hot	10
Context Field	network_type	~ 4	one-hot	1
	time	~ 10	one-hot	1
	brand	~ 15	one-hot	1

The statistics of the data sets are summarized in Table 2.

5.1 Competitors

We compare FLEN with 7 state-of-the-art models.
(1) **FFM** [9]: a shallow model in which the prediction is aggregated over inner products of feature vectors. It represents a feature by several separate vectors, depending on the multiplying feature field.
(2) **FwFM** [16]: a shallow model which also explicitly aggregates over feature products. Interaction weights are assigned for each field pair.
(3) **DCN** [22]: a deep model that takes the outer product of feature vectors at bit-wise level to a feed-forward neural network.
(4) **DeepFM** [7]: a deep model that consists of a wide component that models factorization machine and a deep component.
(5) **NFM** [8]: a deep model which stacks MLP on top of a bi-interaction pooling layer.
(6) **xDeepFM** [11]: a deep model that explicitly generates features with a compressed interaction network.
(7) **NFFM**[25]: a deep model which learns different feature representations for convolutional operations and product operations.

Table 2: Statistics of evaluation data sets.

Data	#Samples	#Fields	#Features (Sparse)
Avazu	40,428,967	23	1,544,488
Meitu	1,508,149,301	33	5,476,029

All methods are implemented in TensorFlow[2]. We use an embedding dimension of 32 and batch size of 512 for all compared methods. Hidden units d' are set to 64, 32. We use AdaGrad [5] to optimize all deep neural network-based models. DCN has two interaction layers, following by two feed-forward layers. We use one hidden layer of size 200 on top of Bi-Interaction layer for NFM as recommended by their paper. We use 2 layers with size $(64, 32)$ in the MLP component of FLEN. All experiments are executed on one NVIDIA TITAN Xp Card with 128G memory.

5.2 Evaluation Metrics

We use two commonly adopted evaluation metrics.

AUC Area Under the ROC Curve (AUC) measures the probability that a CTR predictor will assign a higher score to a randomly chosen positive item than a randomly chosen negative item. A higher AUC indicates a better performance.

Logloss Since all models attempt to minimize the *Logloss* defined by Equation 9, we use it as a straightforward metric.

It is now generally accepted that increase in terms of AUC and Logloss at 0.001-level is significant [3, 7, 22].

5.3 Comparative Performance

We report the AUC and Logloss performance of different models in Table 3. We distinguish the original FLEN (denoted as FLEN) and the model with Dicefactor implementation (denoted as FLEN+D). We can see that on both datasets, FLEN has achieved the best performance in terms of AUC and Logloss. We point out that FLEN has impressively boosted the AUC performance of the best competitor (i.e. NFFM) by 0.002 on the industrial dataset, which validates the superiority of FLEN on large-scale CTR systems. We also observe that Dicefactor further significantly enhances AUC performance of FLEN on both datasets.

Furthermore, we can find an interesting observation: leveraging field information makes modeling feature interactions more precisely. This observation is derived from the fact that by exploiting field information, FLEN, NFFM and xDeepFM perform better than NFM, DeepFM and DCN do on the Avazu and Meitu datasets. This phenomenon can be found in more literature, including FFM [9] and FwFM [16].

5.4 Memory Consumption and Running Time

To illustrate the scalability of FLEN, we first compare the model complexity and actual parameter size on Avazu dataset of each

[1] https://www.kaggle.com/c/avazu-ctr-prediction

[2] Codes are available at https://github.com/aimetrics/jarvis

Table 3: AUC and Logloss performance of different models

Model	Avazu		Meitu	
	AUC	Logloss	AUC	Logloss
FFM	0.7400	0.3994	0.6300	0.5625
FwFM	0.7406	0.3988	0.6306	0.5621
DCN	0.7421	0.3981	0.6337	0.5606
DeepFM	0.7438	0.3982	0.6329	0.5612
NFM	0.7449	0.3973	0.6359	0.5596
xDeepFM	0.7509	0.3947	0.6440	0.5576
NFFM	0.7513	0.3945	0.6443	0.5565
FLEN	**0.7519**	**0.3944**	**0.6463**	**0.5558**
FLEN+D	**0.7528**	**0.3944**	**0.6475**	**0.5554**

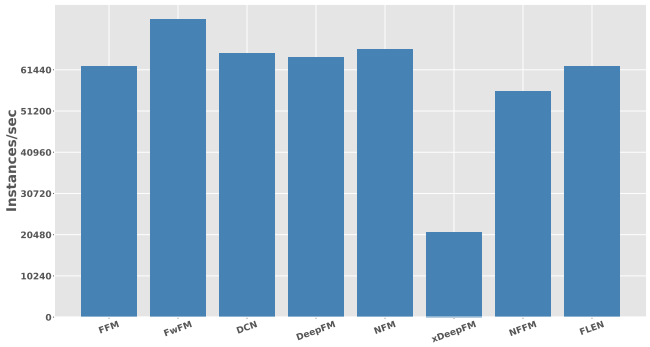
method in Table 4 . For a fair comparison, in computing parameter size, we assume the number of deep layers denoted as $H = 3$, the number of hidden layers denoted as $L = 3$ and embedding size $K_e = 32$. We do not take into account parameters for the feed-forward neural network. The number of features is N=1,544,448 and the number of fields is $M = 23$ on Avazu dataset. We can see that FLEN is one of the models that make use of the smallest number of parameters.

Table 4: Model complexity and parameter size in Avazu dataset for different models

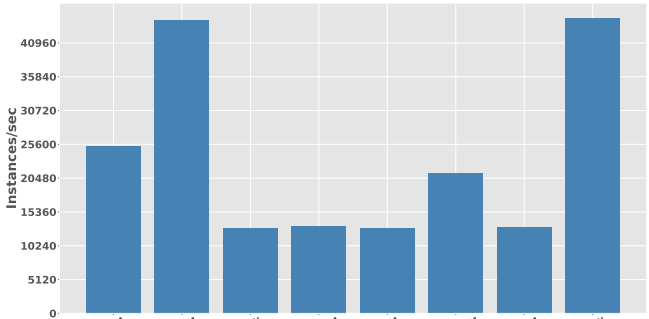
Model	Model Complexity	Parameter Size
FFM	$O(NMK_e)$	1.14×10^9
FwFM	$O(NK_e + M^2)$	4.94×10^7
DCN	$O(NK_e + MK_eL + MK_eH)$	4.94×10^7
DeepFM	$O(NK_e + MK_eH)$	4.94×10^7
NFM	$O(NK_e + K_eH)$	4.94×10^7
xDeepFM	$O(NK_e + MH^2L + MK_eH)$	4.94×10^7
NFFM	$O(NMK_e)$	1.14×10^9
FLEN	$O(NK_e + M^2 + MK_eH)$	4.94×10^7

For a detailed study, we report the number of instances being processed by different models per second on the two datasets. As shown in Figure 4, FLEN operates on the most instances on Meitu dataset. On Avazu dataset, FLEN is comparable with other state-of-the-art methods. But the high efficiency makes FLEN applicable in real industrial systems to handle large scale and high-dimensional data.

We keep track of AUC and Logloss during the training process after each training "epoch" (i.e., 5,000 iterations through all the training data on Avazu and 20,000 iterations on Meitu). As shown in Figure 5, FLEN obtains the best AUC (i.e. highest) and Logloss (i.e. lowest) on both datasets in each iteration. Furthermore, we point out that although NFFM (which is the best competitor) is close with FLEN, FLEN achieves faster convergence towards optimization. For example, FLEN has a sharper increase of AUC and more steep decrease of Logloss on Meitu dataset. Thus FLEN requires less training time than NFFM, which is desirable in real-world production systems.



(a) Avazu



(b) Meitu

Figure 4: Number of instances per second processed by different models

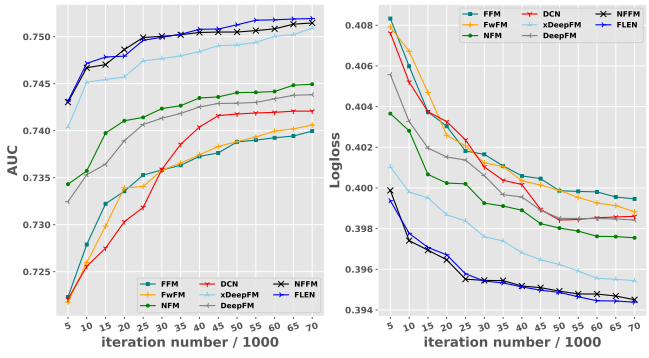
5.5 Impact of Parameters

We analyze the impact of an important parameter of Dicefactor, the probability β to keep bi-linear paths. We empirically find that a small value $\beta \in (0, 0.5)$ leads to poor performance. Henceforth, we set $\beta = 0.5, 0.6, 0.7, 0.8, 0.9, 1.0$ respectively and report the AUC and Logloss results. As shown in Figure 6, performance of FLEN is affected by β, the keep probability. The best parameter settings for AUC and Logloss are consistent. For example, best AUC and Logloss are both obtained at $\beta = 0.7$ on Avazu dataset. On Meitu dataset, the best $\beta = 0.8$.

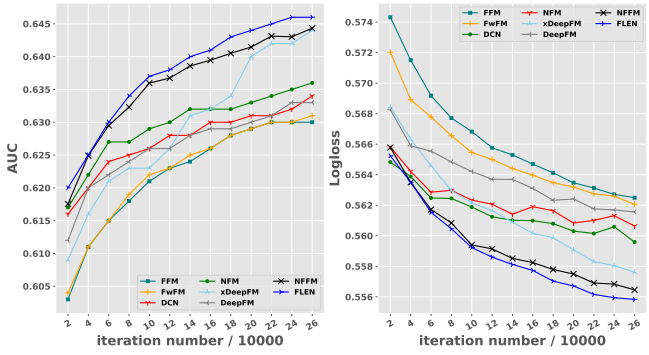
6 ONLINE EVALUATION

To measure the impact that FLEN has on users we conduct an online evaluation through a 7-day A/B testing on Meitu app. We split 10% of the incoming traffic as experiment group, the rest as control group. The items of the control group in the A/B testing period are provided by the previous version of online ranking system, which is based on NFM. The items offered to the experiment group are items that are predicted by FLEN , i.e. we deliver top 12 items with highest prediction score by FLEN.

We report the CTR different during the A/B testing period in Figure 7. We observe stable and significant increase of CTR during the A/B testing period. The minimal CTR increase is above 4.9%. The mean CTR improvement over seven days is 5.195% with variance 0.282%.



(a) Avazu



(b) Meitu

Figure 5: AUC and Logloss change in training time.

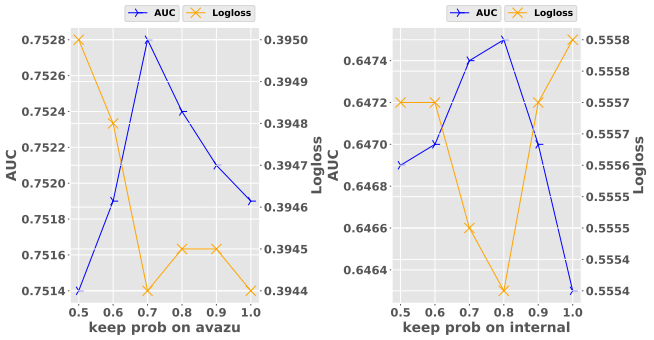


Figure 6: AUC and logloss with different keep probability of dicefactor on two datasets.

7 CONCLUSION

In this paper, we describe the FLEN model that is deployed in the online recommender systems in Meitu, serving the main traffic. FLEN has obtained significant performance increase by exploiting field information with an acceptable memory usage and computing latency in the real-time serving system. As future work, we plan to explore the usage of the attention mechanism to attend to important field embedding. Furthermore, we are interested in extending FLEN to multi-task learning, i.e. predict conversions-rate and click-through-rate simultaneously.

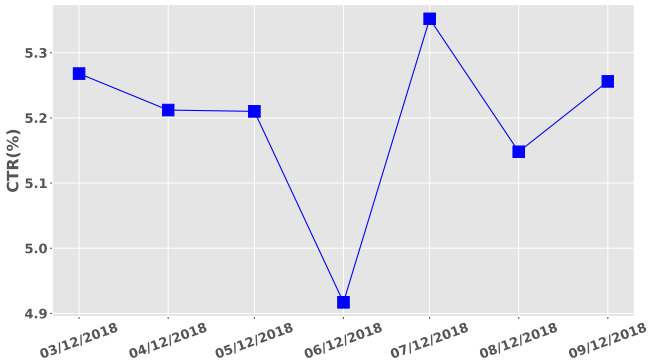


Figure 7: CTR increase during our online A/B testing period.

ACKNOWLEDGMENTS

The authors thank the Runquan Xie for the valuable comments, which are beneficial to the authors' thoughts on recommender systems and the revision of the paper. The authors also thank Haoxuan Huang for his discussions and help in the extensive experiments at Meitu. Chen Lin is supported by the National Natural Science Foundation of China Nos. 61972328. Dugang Liu is supported by the National Natural Science Foundation of China Nos. 61872249, 61836005 and 61672358.

REFERENCES

[1] Chapelle, O.; Manavoglu, E.; and Rosales, R. 2015. Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5(4):61.

[2] Cheng, C.; Xia, F.; Zhang, T.; King, I.; and Lyu, M. R. 2014. Gradient boosting factorization machines. In *Proceedings of the 8th ACM Conference on Recommender systems*, 265–272. ACM.

[3] Cheng, H.-T.; Koc, L.; Harmsen, J.; Shaked, T.; Chandra, T.; Aradhye, H.; Anderson, G.; Corrado, G.; Chai, W.; Ispir, M.; et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, 7–10. ACM.

[4] Covington, P.; Adams, J.; and Sargin, E. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, 191–198. ACM.

[5] Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12(Jul):2121–2159.

[6] Graepel, T.; Candela, J. Q.; Borchert, T.; and Herbrich, R. 2010. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. Omnipress.

[7] Guo, H.; Tang, R.; Ye, Y.; Li, Z.; and He, X. 2017. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*.

[8] He, X., and Chua, T.-S. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, 355–364. ACM.

[9] Juan, Y.; Lefortier, D.; and Chapelle, O. 2017. Field-aware factorization machines in a real-world online advertising system. In *Proceedings of the 26th International Conference on World Wide Web Companion*, 680–688. International World Wide Web Conferences Steering Committee.

[10] Koren, Y.; Bell, R.; and Volinsky, C. 2009. Matrix factorization techniques for recommender systems. *Computer* 42(8):30–37.

[11] Lian, J.; Zhou, X.; Zhang, F.; Chen, Z.; Xie, X.; and Sun, G. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. *arXiv preprint arXiv:1803.05170*.

[12] Liu, W.; Tang, R.; Li, J.; Yu, J.; Guo, H.; He, X.; and Zhang, S. 2018. Field-aware probabilistic embedding neural network for ctr prediction. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, 412–416. New York, NY, USA: ACM.

[13] Liu, B.; Tang, R.; Chen, Y.; Yu, J.; Guo, H.; and Zhang, Y. 2019. Feature generation by convolutional neural network for click-through rate prediction. In *The World Wide Web Conference*, WWW '19, 1119–1129. New York, NY, USA: ACM.

[14] McMahan, H. B.; Holt, G.; Sculley, D.; Young, M.; Ebner, D.; Grady, J.; Nie, L.; Phillips, T.; Davydov, E.; Golovin, D.; et al. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 1222–1230. ACM.

[15] Oentaryo, R. J.; Lim, E.-P.; Low, J.-W.; Lo, D.; and Finegold, M. 2014. Predicting response in mobile advertising with hierarchical importance-aware factorization machine. In *Proceedings of the 7th ACM international conference on Web search and data mining*, 123–132. ACM.

[16] Pan, J.; Xu, J.; Ruiz, A. L.; Zhao, W.; Pan, S.; Sun, Y.; and Lu, Q. 2018. Field-weighted factorization machines for click-through rate prediction in display advertising. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, 1349–1357. International World Wide Web Conferences Steering Committee.

[17] Qu, Y.; Fang, B.; Zhang, W.; Tang, R.; Niu, M.; Guo, H.; Yu, Y.; and He, X. 2018. Product-based neural networks for user response prediction over multi-field categorical data. *ACM Transactions on Information Systems (TOIS)* 37(1):5.

[18] Rendle, S.; Gantner, Z.; Freudenthaler, C.; and Schmidt-Thieme, L. 2011. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, 635–644. ACM.

[19] Rendle, S. 2010. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, 995–1000. IEEE.

[20] Shan, Y.; Hoens, T. R.; Jiao, J.; Wang, H.; Yu, D.; and Mao, J. 2016. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 255–262. ACM.

[21] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

[22] Wang, R.; Fu, B.; Fu, G.; and Wang, M. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, 12. ACM.

[23] Wang, X.; He, X.; Feng, F.; Nie, L.; and Chua, T.-S. 2018. Tem: Tree-enhanced embedding model for explainable recommendation. In *Proceedings of the 2018 World Wide Web Conference*, 1543–1552. International World Wide Web Conferences Steering Committee.

[24] Xiao, J.; Ye, H.; He, X.; Zhang, H.; Wu, F.; and Chua, T.-S. 2017. Attentional factorization machines: Learning the weight of feature interactions via attention networks. *arXiv preprint arXiv:1708.04617*.

[25] Yang, Y.; Xu, B.; Shen, F.; and Zhao, J. 2019. Operation-aware neural networks for user response prediction. *arXiv preprint arXiv:1904.12579*.

[26] Zhang, W.; Du, T.; and Wang, J. 2016. Deep learning over multi-field categorical data. In *European conference on information retrieval*, 45–57. Springer.

[27] Zhu, J.; Shan, Y.; Mao, J.; Yu, D.; Rahmanian, H.; and Zhang, Y. 2017. Deep embedding forest: Forest-based serving with deep embedding features. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1703–1711. ACM.

[28] Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818–2826.

[29] Liu, X.; Xue, W.; Xiao, L.; and Zhang, B. 2017. Pbodl: Parallel bayesian online deep learning for click-through rate prediction in tencent advertising system. *arXiv preprint arXiv:1707.00802*.

[30] Deng, W.; Pan, J.; Zhou, T.; Flores, A.; and Lin, G. 2020. A sparse deep factorization machine for efficient ctr prediction. *arXiv preprint arXiv:2002.06987*.

[31] Zhou, G.; Zhu, X.; Song, C.; Fan, Y.; Zhu, H.; Ma, X.; Yan, Y.; Jin, J.; Li, H.; and Gai, K. 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.