Feature Generation by Convolutional Neural Network for Click-Through Rate Prediction

Bin Liu[1], Ruiming Tang[‡ 1], Yingzhi Chen[★ 2], Jinkai Yu[1], Huifeng Guo[1], Yuzhou Zhang[1]
[1]Noah's Ark Lab, Huawei, Shenzhen, China
[2]Jinan University, Guangzhou, China
[1]{liubin131, tangruiming, yujinkai, huifeng.guo, zhangyuzhou3}@huawei.com
[2]chen_yingzhi@163.com

ABSTRACT

Click-Through Rate prediction is an important task in recommender systems, which aims to estimate the probability of a user to click on a given item. Recently, many deep models have been proposed to learn low-order and high-order feature interactions from original features. However, since useful interactions are always sparse, it is difficult for DNN to learn them effectively under a large number of parameters. In real scenarios, artificial features are able to improve the performance of deep models (such as Wide & Deep Learning), but feature engineering is expensive and requires domain knowledge, making it impractical in different scenarios. Therefore, it is necessary to augment feature space automatically.In this paper, We propose a novel Feature Generation by Convolutional Neural Network (FGCNN) model with two components: *Feature Generation* and *Deep Classifier*. *Feature Generation* leverages the strength of CNN to generate local patterns and recombine them to generate new features. *Deep Classifier* adopts the structure of IPNN to learn interactions from the augmented feature space. Experimental results on three large-scale datasets show that FGCNN significantly outperforms nine state-of-the-art models. Moreover, when applying some state-of-the-art models as *Deep Classifier*, better performance is always achieved, showing the great compatibility of our FGCNN model. This work explores a novel direction for CTR predictions: it is quite useful to reduce the learning difficulties of DNN by automatically identifying important features.

KEYWORDS

CNN, Click-Through Rate Prediction, Feature Generation

ACM Reference Format:
Bin Liu[1], Ruiming Tang[‡ 1], Yingzhi Chen[★ 2], Jinkai Yu[1], Huifeng Guo[1], Yuzhou Zhang[1]. 2019. Feature Generation by Convolutional Neural Network for Click-Through Rate Prediction. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313497

‡: Corresponding author.
★: This work was done when Yingzhi Chen was an intern in Huawei.

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313497

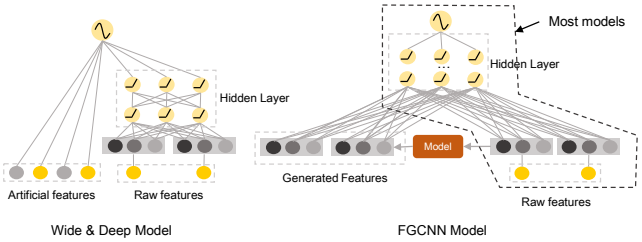

Figure 1: Comparision Between our Model and Wide & Deep Model

1 INTRODUCTION

Click-Through Rate (CTR) is a crucial task for recommender systems, which estimates the probability of a user to click on a given item [9, 23]. In an online advertising application, which is a billion-dollar scenario, the ranking strategy of candidate advertisements is by CTR×bid where "bid" is the profit that the system receives once the advertisement is clicked on. In such applications, the performance of CTR prediction models [9, 23, 40] is one of the core factors determining system's revenue.

The key challenge for CTR prediction tasks is to effectively model feature interactions. Generalized linear models, such as FTRL [21], perform well in practice, but these models lack the ability to learn feature interactions. To overcome the limitation, Factorization Machine [26] and its variants [16] are proposed to model pairwise feature interactions as the inner product of latent vectors and show promising results. Recently, deep neural networks (DNN) have achieved remarkable progress in computer vision [11, 32] and natural language processing [1, 33]. And some deep learning models have been proposed for CTR predictions, such as PIN [23], xDeepFM [19] and etc. Such models feed raw features to a deep neural network to learn feature interactions explicitly or implicitly. Theoretically, DNN is able to learn arbitrary feature interactions from the raw features. However, due to that useful interactions are ususally sparse compared with the combination space of raw features, it is of great difficulties to learn them effectively from a large number of parameters [23, 29].

Observing such difficulties, Wide & Deep Learning [7] leverages feature engineering in the *wide* component to help the learning of *deep* component. With the help of artificial features, the performance of deep component is improved significantly (0.6% improvement on offline AUC and 1% improvement on online CTR). However, feature engineering can be expensive and requires domain knowledge. If we can generate sophisticated feature interactions

automatically by machine learning models, it will be more practical and robust.

Therefore, as shown in Figure 1, we propose a general framework for automatical feature generation. Raw features are input into a machine learning model (represented by the red box in Figure 1) to identify and generate new features [1]. After that, the raw features and the generated new features are combined and fed into a deep neural network. The generated features are able to reduce the learning difficulties of deep models by capturing the sparse but important feature interactions in advance.

The most straightforward way for automatical feature generation is to perform Multi-Layer Perceptron (MLP[2]) and use the hidden neurons as the generated features. However, as mentioned above, due to that useful feature interactions are usually sparse [23], it is rather difficult for MLP to learn such interactions from a huge parameter space. For example, suppose we have four user features: *Name*, *Age*, *Height*, *Gender* to predict whether a user will download an online game. Assume that the feature interaction between *Age* and *Gender* is the only signal that matters, so that an optimal model should identify this and only this feature interaction. When performing MLP with only one hidden layer, the optimal weights associated to the embeddings of *Name* and *Height* should be all 0's, which is fairly difficult to achieve.

As an advanced neural network structure, Convolutional Neural Network (CNN) has achieved great success in the area of computer vision [14] and natural language processing [1]. In CNN, the design of shared weights and pooling mechanism greatly reduces the number of parameters needed to find important local patterns and it will alleviate the optimization difficulties of later MLP structures. Therefore, CNN provides a potentially good solution to realize our idea (identify the sparse but important feature interactions). However, applying CNN directly could result in unsatisfactory performance. In CTR prediction, different arrange orders of original features do not have different meanings. For example, whether the arrangement order of features being *(Name, Age, Height, Gender)* or *(Age, Name, Height, Gender)* does not make any difference to describe the semantics of a sample, which is completely different from the case of images and sentences. If we only use the neighbor patterns extracted by CNN, many useful global feature interactions will be lost. This is also why CNN models do not perform well for CTR prediction task. To overcome this limitation, we perform CNN and MLP, which complement each other, to learn global-local feature interactions for feature generation.

In this paper, we propose a new model for CTR prediction task, namely Feature Generation by Convolutional Neural Network (FGCNN), which consists of two components: *Feature Generation* and *Deep Classifier*. In *Feature Generation*, a CNN+MLP structure is designed to identify and generate new important features from raw features. More specifically, CNN is performed to learn neighbor feature interactions, while MLP is applied to recombine them to extract global feature interactions. After *Feature Generation*, the feature space can be augmented by combining raw features and new features. In *Deep Classifier*, almost all state-of-the-art network structures (such as PIN [23], xDeepFM [19], DeepFM [9]) can be adopted. Therefore, our model has good compatibility with the state-of-the-art models in recommender systems. For the ease of illustration, we will adopt IPNN model [22, 23] as *Deep Classifier* in FGCNN, due to its good trade-off between model complexity and accuracy. Experimental results in three large-scale datasets show that FGCNN significantly outperforms nine state-of-the-art models, demonstrating the effectiveness of FGCNN. While adopting other models in *Deep Classifier*, better performance is always achieved, which shows the usefulness of the generated features. Step-by-step analyses show that each component in FGCNN contributes to the final performance. Compared with traditional CNN structure, our CNN+MLP structure performs better and more stable when the order of raw features changes, which demonstrates the robustness of FGCNN.

To summarize, the main contributions of this paper can be highlighted as follows:

- An important direction is identified for CTR prediction: it is both necessary and useful to reduce the optimization difficulties of deep learning models by automatically generating important features in advance.
- We propose a new model-FGCNN for automatical feature generation and classification, which consists of two components: *Feature Generation* and *Deep Classifier*. *Feature Generation* leverages CNN and MLP, which complement each other, to identify the important but sparse features. Moreover, almost all other CTR models can be applied in *Deep Classifier* to learn and predict based on the generated and raw features.
- Experiments on three large-scale datasets demonstrate the overall effectiveness of FGCNN model. When the generated features are used in other models, better performance is always achieved, which shows the great compatibility and robustness of our FGCNN model.

The rest of this paper is organized as follows: Section 2 presents the proposed FGCNN model in detail. Experimental results will be shown and discussed in Section 3. Related works will be introduced in Section 4. Section 5 concludes the paper.

2 FEATURE GENERATION BY CONVOLUTIONAL NEURAL NETWORK MODEL

2.1 Overview

In this section, we will describe the proposed Feature Generation by Convolutional Neural Network (FGCNN) model in detail. The used notations are summarized in Table 1.

As shown in Figure 2, FGCNN model consists of two components: *Feature Generation* and *Deep Classifier*. More specifically, *Feature Generation* focuses on identifying useful local and global patterns to generate new features as a complement to raw features, while *Deep Classifier* learns and predicts based on the augmented feature space through a deep learning model. Besides the two components, we will also formalize *Feature Embedding* of our model. The details of these components are presented in the following subsections.

[1]Here, new features are the feature interactions of the raw features. In the rest of this paper, we may use the term "new features" for the ease of presentation.
[2]MLP is a neural network with several fully connected layers.

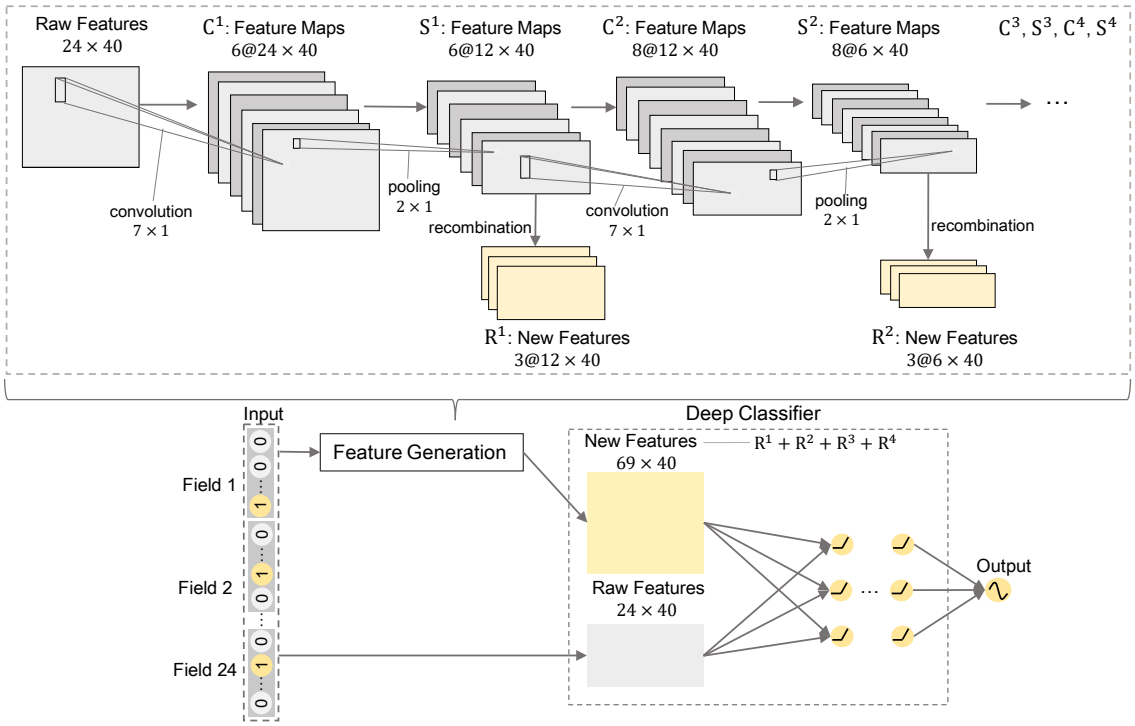

Figure 2: An overview of Feature Generation by Convolutional Neural Network Model (The hyper-parameters in the figure are the best setting of FGCNN on Avazu Dataset)

2.2 Feature Embedding

In most CTR prediction tasks, data is collected in a multi-field categorical form[3] [30, 38], so that each data instance is normally transformed into a high-dimensional sparse (binary) vector via one-hot encoding [13]. For example, (Gender=Male, Height=175, Age=18, Name=Bob) can be represented as:

$$\underbrace{(0,1)}_{\text{Gender=Male}}\ \underbrace{(0,\ldots,1,0,0)}_{\text{Height=175}}\underbrace{(0,1,\ldots,0,0)}_{\text{Age=18}}\underbrace{(1,0,0,..,0)}_{\text{Name=Bob}} \tag{1}$$

An embedding layer is applied upon the raw features' input to compress them to low-dimensional vectors. In our model, if a field is univalent (e.g., "Gender=Male"), its embedding is the feature embedding of the field; if a field is multivalent (e.g., "Interest=Football, Basketball"), the embedding of the field takes the sum of features' embeddings [8].

More formally, in an instance, each field i ($1 \le i \le n_f$) is represented as a low-dimensional vector $e_i \in R^{1\times k}$, where n_f is the number of fields and k is embedding size. Therefore, each instance can be represented as an embedding matrix $E = (e_1^T, e_2^T, ..., e_{n_f}^T)^T$, where $E \in R^{n_f \times k}$. In FGCNN model, the embedding matrix E can be utilized in both *Feature Generation* and *Deep Classifier*. To avoid the inconsistency of gradient direction when updating parameters, we will introduce another embedding matrix $E' \in R^{n_f \times k}$ for *Deep Classifier* while E is used in *Feature Generation*.

[3]Features in numerical form are usually transformed into categorical form by bucketing.

Table 1: Notations

Parameter	Meaning
k	embedding size
t_f	the total number of one-hot features
n_f	the number of fields
n_c	the number of convolutional layers
n_h	the number of hidden layers
h^i	the height of convolutional kernel in the i-th convolutional layer
h_p	pooling height (width=1) of pooling layers
e_i	the embedding vector for i-th field
E^i	the input of the i-th convolutional layer
m_c^i	the number of feature maps in the i-th convolutional layer
m_r^i	the number of new features' map in the i-th recombination layer
N_i	the number of generated features in the i-th recombination layer ($=n_f/h_p^i m_r^i$)
$C^i_{:,:,j}$	the j-th output feature map of the i-th convolutional layer
$\mathbb{WC}^i_{:,:,j}$	weights for the j-th output feature map of i-th convolutional layer
$S^i_{:,:,j}$	the j-th output feature map of the i-th pooling layer
$\mathbb{WR}^i$	weights for the i-th recombination layer
$\mathbb{BR}^i$	bias for the i-th recombination layer
$R^i_{:,:,j}$	the j-th output map of new features in the i-th recombination layer
W^i	weights for the i-th hidden layer
B^i	biases for the i-th hidden layer
H_i	number of hidden neorons in the i-th hidden layer

2.3 Feature Generation

As stated in Section 1, generating new features from raw features helps improve the performance of deep learning models (as demonstrated by Wide & Deep Learning [7]). To achieve this goal, *Feature Generation* designs a proper neural network structure to identify useful feature interactions then generate new features automatically. As argued in Section 1, using MLP or CNN alone is not able to generate effective feature interactions from raw features, due to the following reasons: *Firstly*, useful feature interactions are always sparse in the combination space of raw features. Therefore, it is difficult for MLP to learn them from a large amount of parameters. *Secondly*, although CNN can alleviates optimization difficulties of

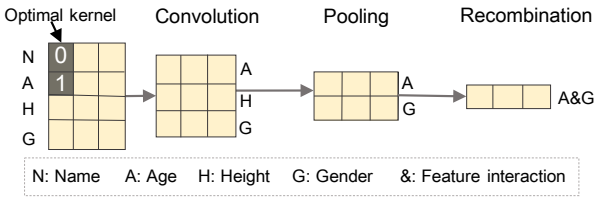

Figure 3: CNN+Recombination structure is able to capture global non-neighbor feature interactions to generate new features. CNN consists of convolutional layer and pooling layer, while Recombination consists of a fully connected layer.

MLP by reducing the number of paramters, it only generates neighbor feature interactions which can lose many useful global feature interactions.

In order to overcome the weakness of applying MLP or CNN alone, as shown in the upper part of Figure 2, we perform CNN and MLP [4] as a complement to each other for feature generation. Figure 3 shows an example of CNN+Recombination structure to capture global feature interactions. As can be observed, CNN learns useful neighbor feature patterns with a limited number of parameters, while recombination layer (which is a fully connected layer) generates global feature interactions based on the neighbor patterns provided by CNN. Therefore, important features can be generated effectively via this neural network structure, which has fewer parameters than directly applying MLP for feature generation.

In the following parts, we detail the CNN+Recombination structure of *Feature Generation*, namely, *Convolutional Layer*, *Pooling layer* and *Recombination layer*.

2.3.1 Convolutional Layer. Each instance is represented as an embedding matrix $E \in R^{n_f \times k}$ via feature embedding, where n_f is the number of fields and k is embedding size. For convenience, reshape the embedding matrix as $E^1 \in R^{n_f \times k \times 1}$ as the input matrix of the first convolutional layer. To capture the neighbor feature interactions, a convolutional layer is obtained by convolving a matrix $\mathbb{WC}^1 \in R^{h^1 \times 1 \times 1 \times m_c^1}$ with non-linear activation functions (where h^1 is the height of the first convolutional weight matrix and m_c^1 is the number of feature maps in the first convolutional layer). Suppose the output of the first convolutional layer is denoted as $C^1 \in R^{n_f \times k \times m_c^1}$, we can formulate the convolutional layer as follows:

$$C^1_{p,q,i} = tanh(\sum_{m=1}^{1} \sum_{j=1}^{h^1} E^1_{p+j-1,q,m} \mathbb{WC}^1_{j,1,1,i}) \tag{2}$$

$$tanh(x) = \frac{exp(x) - exp(-x)}{exp(x) + exp(-x)} \tag{3}$$

where $C^1_{:,:,i}$ denotes the i-th feature map in the first convolutional layer and p, q are the row and column index of the i-th feature map. Notice that the above equation excludes padding which is performed in practice.

2.3.2 Pooling Layer. After the first convolutional layer, a max-pooling layer is applied to seize the most important feature interactions and reduce the number of parameters. We refer h_p as the height of pooling layers (width=1). The output in the first pooling layer is $S^1 \in R^{(n_f/h_p) \times k \times m_c^1}$:

$$S^1_{p,q,i} = max(C^1_{p \cdot h_p,q,i}, \ \dots, C^1_{p \cdot h_p + h_p - 1,q,i}) \tag{4}$$

The pooling result of the i-th pooling layer will be the input for the $(i+1)$-th convolutional layer: $E^{i+1} = S^i$.

2.3.3 Recombination Layer. After the first convolutional layer and pooling layer, $S^1 \in R^{(n_f/h_p) \times k \times m_c^1}$ contains the patterns of neighbor features. Due to the nature of CNN, global non-neighbor feature interactions will be ignored if S^1 is regarded as the generated new features. Therefore, we introduce a fully connected layer to recombine the local neighbor feature patterns and generate important new features. We denote the weight matrix as $\mathbb{WR}^1 \in R^{(n_f/h_p k m_c^1) \times (n_f/h_p k m_r^1)}$ and the bias as $\mathbb{BR}^1 \in R^{(n_f/h_p k m_r^1)}$, where m_c^1 is the number of feature maps in the first convolutional layer and m_r^1 is the number of new features' map in the first recombination Layer. Therefore, in the i-th recombination layer, $n_f/h_p^i m_r^i$ features are generated:

$$R^1 = tanh(S^1 \cdot \mathbb{WR}^1 + \mathbb{BR}^1) \tag{5}$$

2.3.4 Concatenation. New features can be generated by performing CNN+Recombination multiple times. Assume there are n_c convolutional layers, pooling layers and recombination layers and $N_i = n_f/h_p^i m_r^i$ fields of features are generated by i-th round denoted as R^i. The overall new features $\mathcal{R} \in R^{N \times k}$ (where $N = \sum_{i=1}^{n_c} N_i$) generated by *Feature Generation* are formalized as:

$$\mathcal{R} = (R^1, R^2, ..., R^{n_c}) \tag{6}$$

Then, raw features and new features are concatenated as

$$\mathbb{E} = (E'^T, \mathcal{R}^T)^T \tag{7}$$

where E' is the embedding matrix of raw features for *Deep Classifier* (see Section 2.2). Both the raw features and new features are utilized for CTR prediction in *Deep Classifier*, which will be elaborated in the next subsection.

2.4 Deep Classifier

As mentioned above, raw features and new features are concatenated as an augmented embedding matrix $\mathbb{E} \in R^{(N+n_f) \times k}$, where N and n_f are the number of fields of new features and raw features respectively. $\mathbb{E}$ is input into *Deep Classifier*, which aims to learn further interactions between the raw features and new generated features. In this subsection, for the ease of presentation, we adopt IPNN model [23] as the network structure in *Deep Classifier*, due to its good trade-off between model complexity and accuracy. In fact, any advanced network structure can be adopted, which shows the compatibility of FGCNN with the existing works. The compatibility of FGCNN model will be verified empirically in Section 3.

2.4.1 Network Structure. IPNN model [23] combines the learning ability of FM [27] and MLP. It utilizes an FM layer to extract pairwise feature interactions from embedding vectors by inner product operations. After that, the embeddings of input features and the results of FM layer are concatenated and fed to MLP for learning. As evaluated in [23], the performance of IPNN is slightly worse than PIN (the best model in [23]), but IPNN is much more efficient. We will illustrate the network structure of IPNN model.

[4] According to its function, we will call MLP with one hidden layer as recombination layer later.

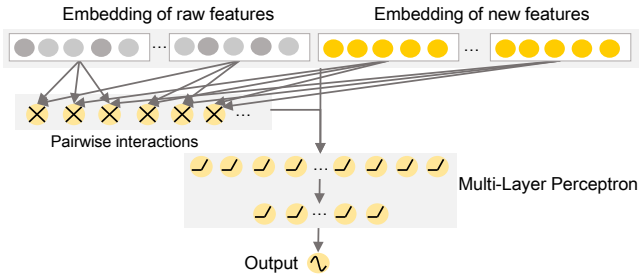

Figure 4: Structure of IPNN Model

As shown in Figure 4, the pariwise feature interactions of augmented embedding matrix $\mathbb{E} \in R^{(N+n_f)\times k}$ are modeled by an FM layer, as follows:

$$R_{fm} = (< \mathbb{E}_1, \mathbb{E}_2 >, ..., < \mathbb{E}_{N+n_f-1}, \mathbb{E}_{N+n_f} >) \tag{8}$$

where $\mathbb{E}_i$ is the embedding of the i-th field, $< a, b >$ means the inner product of a and b. The number of pairwise feature interactions in the FM layer is $\frac{(N+n_f)(N+n_f-1)}{2}$.

After the FM layer, R_{fm} is concatenated with the augmented embedding matrix $\mathbb{E}$, which are fed into MLP with n_h hidden layers to learn implicit feature interactions. We refer the input for the i-th hidden layer as I_i and the output for the i-th hidden layer as O_i. The MLP is formulated as:

$$I_1 = (R_{fm}, flatten(\mathbb{E})) \tag{9}$$

$$O_i = relu(I_i W^i + B^i) \tag{10}$$

$$I_{i+1} = O_i \tag{11}$$

where W^i and B^i are the weight matrix and bias of the i-th hidden layer in MLP. For the last hidden layer (the n_h-th layer), we will make final predictions:

$$\hat{y} = sigmoid(O_{n_h} W^{n_h+1} + b^{n_h+1}) \tag{12}$$

2.4.2 Batch Normalization. Batch Normalization (BN) is proposed in [15] to solve covariant shift and accelerate DNN training. It normalizes activations $w^T x$ using statistics within a mini-batch, as follows:

$$BN(w^T x) = \frac{w^T x - avg_i(w^T x)}{std_i(w^T x)} g + b \tag{13}$$

where g and b scale and shift the normalized values.

In the FGCNN model, Batch Normalization is applied before each activation function to accelerate model training.

2.4.3 Objective Function. The objective function in the FGCNN model is to minimize cross entropy of predicted values and the labels, which is defined as:

$$\mathcal{L}(y, \hat{y}) = -ylog\hat{y} - (1-y)log(1-\hat{y}) \tag{14}$$

where $y \in \{0, 1\}$ is the label and $\hat{y} \in (0, 1)$ is the predicted probability of $y = 1$.

2.5 Complexity Analysis

In this section, we analyse the space and time complexity of FGCNN model. Notations in Table 1 will be reused.

2.5.1 Space Complexity. The space complexity of FGCNN model consists of three parts, namely *Feature Embedding*, *Feature Generation* and *Deep Classifier*. In feature embedding, since there are two matrices (one for *Feature Generation* and one for *Deep Classifier* (as discussed in Section 2.2)), there are $s_0 = 2t_f k$ parameters, where t_f is the total number of one-hot raw features.

In *Feature Generation*, there are $h^i m_c^{i-1} m_c^i$ parameters for convolutional matrix in i-th convolutional layer and $n_f/h_p^i k m_c^i$ neurons in the output of i-th pooling layer. In addition, the number of parameters in i-th recombination layer is $n_f^2/h_p^{2i} k^2 m_c^i m_r^i$, which will generate $N_i = n_f/h_p^i m_r^i$ fields of new features ($n_f/h_p^i = N_i/m_r^i$). Therefore, the total parameters of *Feature Generation* is:

$$\sum_{i=1}^{n_c} h^i m_c^{i-1} m_c^i + n_f^2/h_p^{2i} k^2 m_c^i m_r^i = \tag{15}$$

$$\sum_{i=1}^{n_c} h^i m_c^{i-1} m_c^i + N_i^2 k^2 m_c^i / m_r^i \tag{16}$$

After the *Feature Generation* process, we can get $T = n_f + \sum_{i=1}^{n_c} N_i$ feature embeddings in total. Normally, m_r^i is usually set as $1, 2, 3, 4$ and $m_r^i = m_r^{i-1}$. Meanwhile, h_p is usually set to 2 so that $N_i = N_1/2^{i-1}$. Furthermore, $N_i^2 k^2 >> h_i m_c^{i-1}$ and $\sum_{i=1}^{n_c} m_c^i/m_r^i 2^{2(i-1)}$ is usually a small number. The space complexity s_1 of the feature generating process can be simplified as:

$$s_1 = O(\sum_{i=1}^{n_c} N_1^2/2^{2(i-1)} k^2 m_c^i/m_r^i) = O(N_1^2 k^2) \tag{17}$$

For the first hidden layer of *Deep Classifier*, the number of parameters in the weight matrix is $(T(T-1)/2 + Tk)H_1$ (recall that the input is the raw and new features' embedding and their pairwised product). In the i-th hidden layer, the number of parameters is $H_{i-1}H_i$. Therefore, the space complexity of *Deep Classifier* is:

$$s_2 = O(\frac{T(T-1) + 2Tk}{2} H_1 + \sum_{i=2}^{n_h} H_i H_{i-1}) \tag{18}$$

where Eq. (18) can be simplified as $O(T^2 H_1 + \sum_{i=2}^{n_h} H_i H_{i-1})$ since $Tk < T(T-1)$ usually.

In summary, the overall space complexity of FGCNN is $s_0 + s_1 + s_2 = O(t_f k + N_1^2 k^2 + T^2 H_1 + \sum_{i=1}^{q} H_i H_{i-1})$, which is dominated by the number of one-hot features, the number of generated features, the number of hidden neurons and the embedding size.

2.5.2 Time Complexity. We first analyze the time complexity in *Feature Generation*. In the i-th convolutional layer, the output dimension is $n_f/h_p^{i-1} k m_c^i = N_i h_p / m_r^i k m_c^i$ (recall that $N_i = n_f/h_p^i m_r^i$) and each point is calculated from $h^i m_c^{i-1}$ data points where h^i is the height of convolutional kernels in the i-th convolutional layer. Therefore, the time complexity of the i-th convolutional layer is $N_i h_p k m_c^i h^i m_c^{i-1}/m_r^i$. In the pooling layer, the time complexity of the i-th pooling layer is $n_f/h_p^{i-1} k m_c^i = N_i h_p k m_c^i/m_r^i$. In the i-th recombination layer, the time complexity is $n/h_p^i k m_c^i \cdot N_i k = N_i^2 k^2 m_c^i/m_r^i$ (recall $h_p = 2$ and $N_i = N_1/2^{i-1}$). Therefore, total

Table 2: Dataset Statistics

Dataset	#instances	#features	#fields	pos ratio
Criteo	1×10^8	1×10^6	39	0.5
Avazu	4×10^7	6×10^5	24	0.17
Huawei App Store	2.3×10^8	1.6×10^5	29	0.05

time complexity of the *Feature Generation* is

$$\begin{aligned} t_1 &= \sum_{i=1}^{n_c} N_i h_p k m_c^i h^i m_c^{i-1}/m_r^i + N_i h_p k m_c^i/m_r^i + N_i^2 k^2 m_c^i/m_r^i \\ &= \sum_{i=1}^{n_c} N_1 k m_c^i/(m_r^i 2^{i-1})(2h^i m_c^{i-1} + 2 + N_1/2^{i-1}k) \\ &= O(N_1 k \sum_{i=1}^{n_c} m_c^{i-1} h^i + N_1^2 k^2) \end{aligned}$$

(Notice: $m_c^i/(m_r^i 2^i)$ is usually a small number ranging from 2 to 10)

Then, we analyze the time complexity of *Deep Classifier* (taking IPNN as an example). In the FM layer, the time complexity to calculate pairwise inner product is $\frac{T(T-1)k}{2}$. The number of neurons, that are input to the first hidden layer is $T(T-1)/2 + Tk$. Therefore, the time complexity of the first hidden layer is $O((T(T-1)/2 + Tk)H_1) = O(T^2 H_1)$. For the other hidden layers, the time complexity is $O(H_i H_{i-1})$. Therefore, the total time complexity in *Deep Classifier* is

$$t_2 = O(T^2 H_1) + \sum_{i=2}^{n_h} O(H_i H_{i-1}) \tag{19}$$

In summary, the overall time complexity for FGCNN model is $t_1 + t_2 = O(N_1 k \sum_{i=1}^{n_c} m_c^{i-1} h^i + N_1^2 k^2 + T^2 H_1 + \sum_{i=2}^{q} H_i H_{i-1})$ which is dominated by the number of generated features, the number of neurons in the hidden layers, embedding size and convolutional parameters.

3 EXPERIMENTS

In this section, we conduct extensive experiments to answer the following questions.

- **(Q1)** How does FGCNN perform, compared to the state-of-the-art models for CTR prediction task?
- **(Q2)** Can FGCNN improve the performance of other state-of-the-art models by using their structures in *Deep Classifier*?
- **(Q3)** How does each key structure in FGCNN boost the performance of FGCNN?
- **(Q4)** How do the key hyper-parameters of FGCNN (i.e., size of convolutional kernels,number of convolutional layers, number of generated features) impact its performance?
- **(Q5)** How does *Feature Generation* perform when the order of raw features are randomly shuffled?

Note that when studying questions other than **Q2**, we adpot IPNN as the *Deep Classifier* in FGCNN.

3.1 Experimental Setup

3.1.1 Datasets. Experiments are conducted in the following three datasets:

Criteo: Criteo[5] contains one month of click logs with billions of data samples. A small subset of Criteo was published in Criteo Display Advertising Challenge 2013 and FFM was the winning solution [16, 23]. We select "day 6-12" as training set while select "day 13" for evaluation. Due to the enormous data volume and serious class imbalance (i.e., only 3% samples are positive), negative sampling is applied to keep the positive and negative ratio close to 1:1. We convert 13 numerical fields into one-hot features through bucketing, where the features in a certain field appearing less than 20 times are set as a dummy feature "other".

Avazu: Avazu[6] was published in the contest of Avazu Click-Through Rate Prediction, 2014. The public dataset is randomly splitted into training and test sets at 4:1. Meanwhile, we remove the features appearing less than 20 times to reduce dimensionality.

Huawei App Store: In order to evaluate the performance in industrial CTR prediction problems, we conduct experiments on Huawei App Store Dataset. We collect users' click logs from Huawei App Store while logs from 20180617 to 20180623 are used for training and logs of 20180624 are used for test. Negative sampling is applied to reduce data amount and to adjust the ratio of positive class and negative class. The dataset contains app features (e.g., identification, category), user features (e.g., user's behavior history) and context features (e.g., operation time).

In addition, the statistics of the three datasets are summarized in Table 2.

3.1.2 Baselines. We compare nine baseline models in our experiments, including LR [17], GBDT [6], FM [26], FFM [16], CCPM [20], DeepFM [9], xDeepFM [19], IPNN and PIN [23]. Wide & Deep is

Table 3: Parameter Settings

Param	Criteo	Avazu	Huawei App Store
General	bs=2000 opt=Adam lr=1e-3	bs=2000 opt=Adam lr=1e-3	bs=1000 opt=Adam lr=1e-4 l2=1e-6
LR	–	–	–
GBDT	depth=25 #tree=1300	depth=18 #tree=1000	depth=10 #tree=1400
FM	k=20	k=40	k=40
FFM	k=4	k=4	k=12
CCPM	k=20 conv= 7*1 kernel=[256] net=[256*3,1]	k=40 conv: 7*1 kernel = [128] net= [128*3,1]	k=40 conv= 13*1 kernel= [8,16,32,64] net=[512,256,128,1] drop= 0.8
DeepFM	k=20 LN=T net=[700*5,1]	k=40 LN=T net=[500*5,1]	k=40 net=[2048,1024,512,1] drop=0.9
XdeepFM	k=20 net=[400*3,1] CIN=[100*4]	k=40 net=[700*5,1] CIN:[100*2]	k=40 net=[2048,1024,512,1] drop=0.9 CIN:[100*4]
IPNN	k=20 LN=T net=[700*5,1]	k=40 LN=T net=[500*5,1]	k=40 net=[2048,1024, 512,256,128,1] drop=0.7
PIN	k=20 LN=T net=[700*5,1] subnet=[40,5]	k=40 LN=T net=[500*5,1] sub-net=[40,5]	k=40 net=[2048,1024,512,1] drop=0.9 sub-net=[80,10]
FGCNN	k=20 conv=9*1 kernel=[38,40,42,44] new=[3,3,3,3] BN=T ruenet=[4096,2048,1]	k=40 conv=7*1 kernel=[14,16,18,20] new=[3,3,3,3] BN=T net=[4096,2048, 1024,512,1]	k=40 conv=13*1 kernel=[6, 8, 10, 12] new=[2,2,2,2] BN=T net=[2048,1024, 512,256,128,1] drop=0.8

Note: bs=batch size, opt=optimizer, lr=learning rate, l2=l2 regularization on Embedding Layer, k=embedding size, conv=shape of convolutional kernels, kernel=number of convolutional kernels, pool=max-pooling shape, net=MLP structure, sub-net=micro metwork, drop=dropout rate, LN=layer normalization, BN= batch normalization{T=True}, new=number of kernels for new features.

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

[6] http://www.kaggle.com/c/avazu-ctr-prediction

Table 4: Overall Performance

$\star : p < 10^{-2}$ $\star\star : p < 10^{-4}$ **(two tailed t-test)**

	Criteo		Avazu		Huawei App Store	
Model	AUC	Log Loss	AUC(%)	Log Loss	AUC	Log Loss
LR	78.00%	0.5631	76.76%	0.3868	90.12%	0.1371
GBDT	78.62%	0.5560	77.53%	0.3824	92.68%	0.1227
FM	79.09%	0.5500	77.93%	0.3805	93.26%	0.1191
FFM	79.80%	0.5438	78.31%	0.3781	93.58%	0.1170
CCPM	79.55%	0.5469	78.12%	0.3800	93.71%	0.1159
DeepFM	79.91%	0.5423	78.36%	0.3777	93.91%	0.1145
xDeepFM	80.06%	0.5408	78.55%	0.3766	93.91%	0.1146
IPNN	80.13%	0.5399	78.68%	0.3757	93.95%	0.1143
PIN	80.18%[11]	0.5393	78.72%	0.3755	93.91%	0.1146
FGCNN	**80.22%***	**0.5388***	**78.83%****	**0.3746****	**94.07%****	**0.1134****

not compared here because some state-of-the-art models (such as xDeepFM, DeepFM, PIN) have shown better performance in their publications. We use XGBoost [6] and libFFM[7] as the implementation of GBDT and libFFM, respectively. In our experiments, the other baseline models are implemented with Tensorflow[8].

3.1.3 Evaluation Metrics. The evaluation metrics are **AUC** (Area Under ROC) and **log loss** (cross entropy).

3.1.4 Parameter Settings. Table 3 summerizes the hyper-parameters of each model. For *Criteo* and *Avazu* Datasets, the hyper-parameters of baseline models are set to be the same as in [23]. Notice that when conducting experiments on Criteo and Avazu, we observed that FGCNN uses more parameters in *Deep Classifier* than other models. To make fair comparisons, we also conduct experiments to increase the parameters in MLPs of other deep models. However, all these models cannot achieve better performance than the original settings[9]. The reason could be the overfitting problem where such models simply use embedding of raw features for training but use a complex structure. On the other hand, since our model augments the feature space and enriches the input, more parameters in *Deep Classifier* can boost the performance of our model.

In FGCNN model, *new* is the number of kernels when generating new features. The number of generated features can be calculated as $\sum_{i=1}^{n_c} \#fields/2^i * new_i$.

3.1.5 Significance Test. We repeat the experiments 10 times by changing the random seed for FGCNN and the best baseline model. The two-tailed pairwise t-test is performed to detect significant differences between FGCNN and the best baseline model.

3.2 Overall Performance (Q1)

In this subsection, we compare the performance of different models on the test set. Table 4 summarizes the overall performance of all the compared models on the three datasets, where the underlined numbers are the best results of the baseline models and bold numbers are the best results of all models. We have the following observations:

Firstly, in majority of the cases, non-neural network models perform worse than neural network models. The reason is that deep neural network can learn complex feature interactions much better than the models where no feature interaction is modeled (i.e., LR), or feature interactions are modeled by simple inner product operations (i.e., FM and FFM).

Secondly, FGCNN achieves the best performance among all the models on the three evaluated datasets. It is significantly better than the best baseline models with 0.05%, 0.14% and 0.13% improvements in AUC (0.09%, 0.24% and 0.79% in log loss) on Criteo, Avazu and Huawei App Store datasets, which demonstrates the effectiveness of FGCNN. In fact, a small improvement in offline AUC is likely to lead to a significant increase in online CTR. As reported in [7], compared with LR, Wide & Deep improves offline AUC by 0.275% and the improvement of online CTR is 3.9%. The daily turnover of Huawei App Store is millions of dollars. Therefore, even a few lifts in CTR brings extra millions of dollars each year.

Thirdly, with the help of the generated new features, FGCNN outperforms IPNN by 0.11%, 0.19% and 0.13% in terms of AUC (0.2%, 0.29% and 0.79% in terms of log loss) on Criteo, Avazu and Huawei App Store datasets. It demonstrates that the generated features are very useful and they can effectively reduce the optimization difficulties of traditional DNNs thus leading to better performance.

Fourthly, CCPM, which applies CNN directly, achieves the worst performance among neural network models. Moreover, CCPM performs worse than FFM on Criteo and Avazu datasets. It shows that directly using traditional CNN for CTR prediction task is inadvisable, as CNN is designed to generate neighbor patterns while the arrangement order of feature is usually no meaning in recommendation scenarios. However, in FGCNN, we leverage the strength of CNN to extract local patterns while complementing it with Recombination Layer to extract global feature interactions and generate new features. Therefore, better performance is achieved.

[7] https://github.com/guestwalk/libffm
[8] https://www.tensorflow.org/
[9] Due to the limited pages, we do not show the experimental result in the paper.
[11] According to the experimental recordings, the results of PIN in [23] uses the trick of adaptive embedding size. Here, we use fixed embedding size for all the deep models.

Table 5: Compatibility Study of FGCNN

	Criteo		Avazu		Huawei App Store	
	AUC	Log Loss	AUC	Log Loss	AUC	Log Loss
FM	79.09%	0.5500	77.93%	0.3805	93.26%	0.1191
FGCNN+FM	**79.67%**	**0.5455**	**78.13%**	**0.3794**	**93.66%**	**0.1165**
DNN	79.87%	0.5428	78.30%	0.3778	93.85%	0.1149
FGCNN+DNN	**80.09%**	**0.5402**	**78.55%**	**0.3764**	**94.00%**	**0.1139**
DeepFM	79.91%	0.5423	78.36%	0.3777	93.91%	0.1145
FGCNN+DeepFM	**79.94%**	**0.5421**	**78.44%**	**0.3771**	**93.93%**	**0.1145**
IPNN	80.13%	0.5399	78.68%	0.3757	93.95%	0.1143
FGCNN+IPNN	**80.22%**	**0.5388**	**78.83%**	**0.3746**	**94.07%**	**0.1134**

3.3 Compatibility of FGCNN with Different Models (Q2)

As stated in Section 2.4, *Feature Generation* can augment the original feature space and *Deep Classifier* of FGCNN can adopt any advanced deep neural networks. Therefore, we select several models as *Deep Classifeir* to verify the utility of *Feature Generation*, including non-deep models (FM), deep learning models (DNN, DeepFM, IPNN).

Table 5 summarizes the performance. We have the following observations: Firstly, with the help of the generated new features, the performance of all models are improved, which demonstrates the effectiveness of the generated features and shows the compatibility of FGCNN. Secondly, we observe that when only using raw features, DeepFM always outperforms DNN. But when using the augmented features, FGCNN+DNN outperforms FGCNN+DeepFM. The possible reason is that DeepFM sums up the inner product of input features to the last MLP layer which may cause contradictory gradient updates (compared with MLP) on embeddings.This could be one of the reasons why IPNN (feeding the product into MLP) outperforms DeepFM in all datasets.

In a word, the results show that our FGCNN model can be viewed as a general framework to enhance the existing neural networks by generating new features automatically.

3.4 Effectiveness of FGCNN Variants(Q3)

We conduct experiments to study how each component in FGCNN contributes to the final performance. Each variant is generated by removing or replacing some components in FGCNN, which is described as follows:

- **Removing Raw Features**: In this variant, raw features are not input into *Deep Classifier* and only the generated new features are fed to *Deep Classifier*.
- **Removing New Features**: This variant removes *Feature Generation*. Actually, it is equivalent to IPNN.
- **Applying MLP for *Feature Generation***: *Feature Generation* is replaced by MLP which takes the neurons in each layer as new features. This variant uses the same hidden layers and generates the same number of features in each layer as FGCNN.
- **Removing Recombination Layer**: This variant is to evaluate how *Recombination Layer* complements CNN to capture global feature interactions. The *Recombination Layer* is removed from *Feature Generation* so that the output of pooling layer serves as new features directly. The number of generated new features in each layer keeps the same as FGCNN.

As shown in Table 6, removing any component in FGCNN leads to a drop in performance. We have the following observations:

Firstly, FGCNN with raw features alone or with new generated features alone performs worse than the FGCNN with both of them. This result demonstrates that the generated features are good supplementaries to the original features, which are both crucial.

Secondly, the performance decrease of *Appling MLP for Feature Generation*, compared to FGCNN, shows the ineffectiveness of MLP to identify the sparse but important feature combinations from a large number of parameters. CNN simplifies the learning difficulties by using the shared convolutional kernels, which has much fewer parameters to get the desired combinations. Moreover, MLP recombines the neighbor feature interactions, which is extracted by CNN, to generate global feature interactions.

Thirdly, removing the Recombination Layer will limit the generated features as neighbor feature interactions. Since the arrangement order of raw features has no actual meanings in the CTR prediction task, the restriction can lead to losing important non-neighbor feature interactions thus resulting in worse performance.

3.5 Hyper-parameter Investigation (Q4)

Our FGCNN model has several key hyper-parameters, i.e., the height of convolutional kernels, number of convolutional kernels, number of convolutional layers, and the number of kernels for generating new features. In this subsection, to study the impact of these hyper-parameters, we investigate how FGCNN model works by changing one hyper-parameter while fixing the others on Criteo and Huawei App Store datasets.

3.5.1 Height of Convolutional Kernels. The height of convolutional kernels controls the perception range of convolutional layers. The larger the height is, the more features are involved in the neighbor patterns, but more parameters need to be optimized. To investigate its impact, we increase the height from 2 to the number of fields of a dataset. As shown in the top of Figure 5, the performance generally ascends first and then descends as the height of convolutional kernels increases[10].

The results show that as more features are involved in the convolutional kernels, higher-order feature interactions can be learned

[10]Due to that *Feature Generation* and *Deep Classifier* interrelate with each other, the curve has some fluctuations.

Table 6: Performance of Different FGCNN Variants

	Criteo		Avazu		Huawei App Store	
Method	AUC	Log Loss	AUC	Log Loss	AUC	Log Loss
FGCNN	**80.22%**	**0.5388**	**78.83%**	**0.3746**	**94.07%**	**0.1134**
Removing Raw Features	80.21%	0.5390	78.66%	0.3757	94.01%	0.1138
Removing New Features	80.13%	0.5399	78.68%	0.3757	93.95%	0.1143
Applying MLP for Feature Generation	80.12%	0.5402	78.58%	0.3761	94.04%	0.1135
Removing Recombination Layer	80.11%	0.5403	78.74%	0.3753	94.04%	0.1135

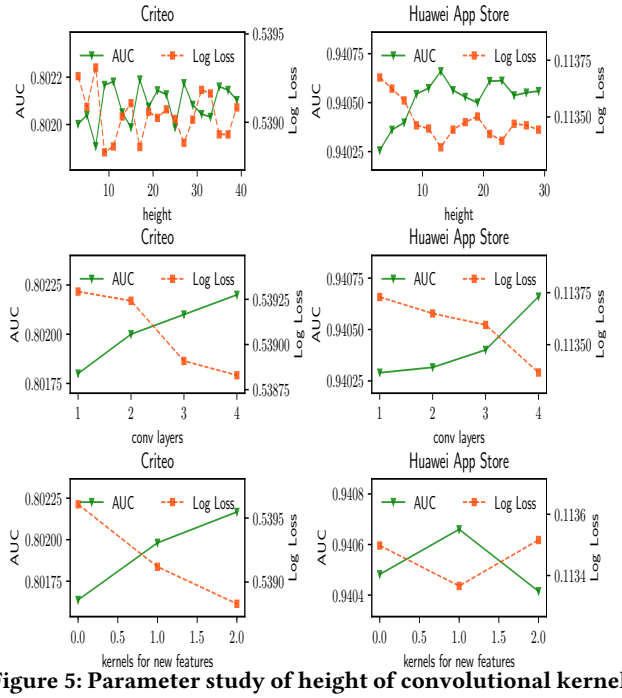

Figure 5: Parameter study of height of convolutional kernel, number of convolutional layers and number of kernels for new features (from top to bottom)

so that the performance increases. However, due to that useful feature interactions are usually sparse, larger heights can cause more difficulties to learn them effectively, leading to a decrease in performance. This observation is consistent with the finding in Section 3.4, i.e., the performance decreases in Appling MLP for Feature Generation.

3.5.2 Number of Convolutional Layers. As shown in the middle of Figure 5, as the number of convolutional layers increases, the performance of FGCNN is improved. Notice that more layers usually lead to higher-order feature interactions. Therefore, the result also shows the effectiveness of high-order feature interactions.

3.5.3 Number of Kernels for Generating New Features. We study how the number of generated features affects the performance of FGCNN. We use the same number of kernels for new features in different *Recombination Layers*. As can be observed in the bottom of Figure 5, the performance is gradually improved with more features generated. The results verify our research idea that it is useful to identify the sparse but important feature interactions first, which can effectively reduce the learning difficulties of DNNs. However, the useful feature interactions can be sparse and limited. If too many features are generated, the extra new features are noisy which will increase the learning difficulties of MLP, leading to the decrease in the performance.

3.6 Effect of Shuffling Order of Raw Features (Q5)

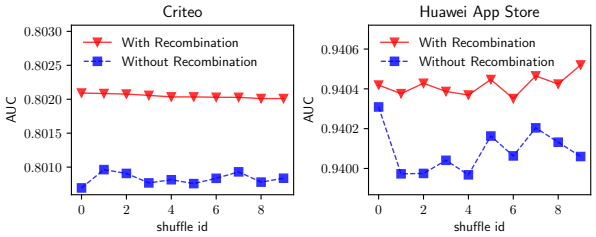

Figure 6: Shuffling the order of raw features

As mentioned before, CNN is designed to capture local neighbor feature patterns so that it is sensitive to the arrangement order of raw features. In our FGCNN model, the design of Recombination Layer is to learn global feature interactions based on CNN's extracted local patterns. Intuitively, our model should have more stable performance than traditional CNN's structure if the order of raw features is shuffled. Therefore, to verify it, we compare the performance of two cases: with/without Recombination Layer. The arrangement order of raw features is shuffled many times at random where the two compared cases are performed for the same shuffled arrangement order.

As shown in Figure 6, the case with Recombination Layer achieves better and more stable performance than that of without Recombination Layer. It demonstrates that with the help of Recombination Layer, FGCNN can greatly reduce the side effects of changing the arrangement order of raw features, which also demonstrates the robustness of our model.

4 RELATED WORK

Click-Through Rate Prediction is normally formulated as a binary classification problem [5, 28]. In this section, we will introduce two important categories of models in Click-Through Rate predictions, namely **shallow models** and **deep learning models**.

4.1 Shallow Models for CTR Prediction

Due to the robustness and efficiency, Logistic Regression (LR) models [18, 25], such as FTRL [21] are widely used in CTR prediction. To learn feature interactions, a common practice is to manually design pairwise feature interactions in its feature space [10, 31]. Poly-2 [4] models all pairwise feature interactions to avoid feature engineering. Factorization Machine (FM) [26] introduces low-dimensional vectors for each feature and models feature interactions through

inner product of feature vectors. FM improves the ability of modelling feature interactions when data is sparse. FFM [16] enables each feature to have multiple latent vectors to perform interactions with features from different fields. LR, Poly-2 and FM variants are widely used in CTR prediction in industry.

4.2 Deep Learning for CTR Prediction

Deep learning has achieved great success in different areas, such as computer vision [11, 14, 32, 36], natural language processing [1, 2, 24, 33] etc. In order to leverage deep Learning for CTR prediction, several models are proposed [34, 37]. FNN [39] is proposed in [39], which uses FM to pre-train the embedding of raw features and then feeds the embeddings to several fully connected layers. Some models adopt DNN to improve FM, such as Attentional FM [35], Neural FM [12].

Wide & Deep Learning [7] jointly trains a wide model and a deep model where the wide model leverages the effectiveness of feature engineering and the deep model learns implicit feature interactions. Despite the usefulness of wide component, feature engineering is expensive and requires expertise. To avoid feature engineering, DeepFM [9] introduces FM layer (order-2 feature interaction) as a wide component and uses a deep component to learn implicit feature interactions. Different from DeepFM, IPNN [23] (also known as PNN in [22]) feeds both the result of FM layer and embeddings of raw features into MLP and results in comparable performance. Rather than using inner product to model pairwise feature interactions as DeepFM and IPNN, PIN [23] uses a Micro-Network to learn complex feature interaction for each feature pair. xDeepFM [19] proposes a novel Compressed Interaction Network (CIN) to explicitly generate feature interactions at the vector-wise level.

There are several models which use CNN for CTR Prediction. CCPM [20] applies multiple convolutional layers to explore neighbor feature dependencies. CCPM performs convolutions on the neighbor fields in a certain alignment. Due to that the order of features has no actual meaning in CTR predictions, CCPM can only learn limited feature interactions between neighbor features. In [3], it is shown that features' arrangement order has a great impact on the final performance of CNN based models. Therefore, the authors propose to generate a set of suitable feature sequences to provide different local information for convolutional layers. However, the key weakness of CNN is not solved.

In this paper, we propose FGCNN model, which splits the CTR prediction task into two stages: *Feature Generation* and *Deep Classifier*. *Feature Generation* augments the original feature space by generating new features while most state-of-the-art models can be adopted in *Deep Classifier* to learn and prediction based on the augmented feature space. Different from traditional CNN models for CTR Prediction [3, 20], FGCNN can leverage the strength of CNN to extract local information and it greatly alleviates the weakness of CNN by introducing the *Recombination Layer* to recombine information from different local patterns learned by CNN to generating new features.

5 CONCLUSION

In this paper, we propose a FGCNN model for CTR prediction which aims to reduce the learning difficulties of DNN models by identifying important features in advance. The model consists of two components: *Feature Generation* and *Deep Classifier*. *Feature Generation* leverages the strength of CNN to identify useful local patterns and it alleviates the weakness of CNN by introducing a *Recombination Layer* to generate global new features from the recombination of the local patterns. In *Deep Classifier*, most existing deep models can be applied on the augmented feature space. Extensive experiments are conducted on three large-scale datasets where the results show that FGCNN outperforms nine state-of-the-art models. Moreover, when applying other models in *Deep Classifier*, compared with the original model without *Feature Generation*, better performance is always achieved, which demonstrates the effectiveness of the generated features. Step-by-step experiments show that each component in FGCNN contributes to the final performance. Furthermore, compared with the traditional CNN structure, our CNN+Recombination structure in *Feature Generation* always performs better and more stable when shuffling the arrangement order of raw features. This work explores a novel direction for CTR prediction that it is effective to automatically generate important features first rather than feeding the raw embeddings to deep learning models directly.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *Computer Science* (2014).

[2] Antoine Bordes, Y-Lan Boureau, and Jason Weston. 2016. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683* (2016).

[3] Patrick PK Chan, Xian Hu, Lili Zhao, Daniel S Yeung, Dapeng Liu, and Lei Xiao. 2018. Convolutional Neural Networks based Click-Through Rate Prediction with Multiple Feature Sequences.. In *IJCAI*. 2007–2013.

[4] Yin Wen Chang, Cho Jui Hsieh, Kai Wei Chang, Michael Ringgaard, and Chih Jen Lin. 2010. Training and Testing Low-degree Polynomial Data Mappings via Linear SVM. *Journal of Machine Learning Research* 11, 11 (2010), 1471–1490.

[5] Junxuan Chen, Baigui Sun, Hao Li, Hongtao Lu, and Xian Sheng Hua. 2016. Deep CTR Prediction in Display Advertising. (2016), 811–820.

[6] Tianqi Chen and Carlos Guestrin. 2016. XGBoost:A Scalable Tree Boosting System. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785–794.

[7] Hengtze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Deepak Chandra, Hrishi Aradhye, Glen Anderson, Greg S Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & Deep Learning for Recommender Systems. *conference on recommender systems* (2016), 7–10.

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

[9] 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).

[10] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, Xiuqiang He, and Zhenhua Dong. 2018. DeepFM: An End-to-End Wide & Deep Learning Framework for CTR Prediction. (2018).

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. (2015), 770–778.

[12] Xiangnan He and TatSeng Chua. 2017. Neural Factorization Machines for Sparse Predictive Analytics. (2017), 355–364.

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

[14] Gao Huang, Zhuang Liu, Van Der Maaten Laurens, and Kilian Q Weinberger. 2016. Densely Connected Convolutional Networks. (2016), 2261–2269.

[15] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[16] Yuchin Juan, Yong Zhuang, Wei Sheng Chin, and Chih Jen Lin. 2016. Field-aware Factorization Machines for CTR Prediction. In *ACM Conference on Recommender Systems*. 43–50.

[17] Kuang Chih Lee, Burkay Orten, Ali Dasdan, and Wentong Li. 2012. Estimating conversion rate in display advertising from past erformance data. In *Acm Sigkdd International Conference on Knowledge Discovery & Data Mining*. 768–776.

[18] Kuang Chih Lee, Burkay Orten, Ali Dasdan, and Wentong Li. 2012. Estimating conversion rate in display advertising from past erformance data. In *Acm Sigkdd International Conference on Knowledge Discovery & Data Mining*. 768–776.

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

[20] Qiang Liu, Feng Yu, Shu Wu, and Liang Wang. 2015. A convolutional click prediction model. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 1743–1746.

[21] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, 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*. ACM, 1222–1230.

[22] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 1149–1154.

[23] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiuqiang He. 2018. Product-based Neural Networks for User Response Prediction over Multi-field Categorical Data. *arXiv preprint arXiv:1807.00311* (2018).

[24] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. 2017. Learning to Generate Reviews and Discovering Sentiment. (2017).

[25] Kan Ren, Weinan Zhang, Yifei Rong, Haifeng Zhang, Yong Yu, and Jun Wang. 2016. User Response Learning for Directly Optimizing Campaign Performance in Display Advertising. (2016), 679–688.

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

[27] Steffen Rendle. 2011. Factorization Machines. In *IEEE International Conference on Data Mining*. 995–1000.

[28] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting clicks:estimating the click-through rate for new ads. In *International Conference on World Wide Web*. 521–530.

[29] Shai Shalevshwartz, Ohad Shamir, and Shaked Shammah. 2017. Failures of Gradient-Based Deep Learning. (2017).

[30] Ying Shan, T. Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and J. C. Mao. 2016. Deep Crossing:Web-Scale Modeling without Manually Crafted Combinatorial Features. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 255–262.

[31] Guangzhong Sun, Guangzhong Sun, Guangzhong Sun, Guangzhong Sun, Guangzhong Sun, and Guangzhong Sun. 2017. Practical Lessons for Job Recommendations in the Cold-Start Scenario. In *Recommender Systems Challenge*. 4.

[32] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. 2016. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. (2016).

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. (2017).

[34] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. ACM, 12.

[35] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. 2017. Attentional factorization machines: Learning the weight of feature interactions via attention networks. *arXiv preprint arXiv:1708.04617* (2017).

[36] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. 2017. Aggregated Residual Transformations for Deep Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*. 5987–5995.

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

[38] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep Learning over Multi-field Categorical Data. (2016).

[39] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep Learning over Multi-field Categorical Data: A Case Study on User Response Prediction. (2016).

[40] 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*. ACM, 1059–1068.