Article

GAIN: A Gated Adaptive Feature Interaction Network for Click-Through Rate Prediction

Yaoxun Liu, Liangli Ma* and Muyuan Wang

College of Electronic Engineering, Naval University of Engineering, Wuhan 430033, China
*Correspondence: maliangl@163.com

Abstract: CTR (Click-Through Rate) prediction has attracted more and more attention from academia and industry for its significant contribution to revenue. In the last decade, learning feature interactions have become a mainstream research direction, and dozens of feature interaction-based models have been proposed for the CTR prediction task. The most common approach for existing models is to enumerate all possible feature interactions or to learn higher-order feature interactions by designing complex models. However, a simple enumeration will introduce meaningless and harmful interactions, and a complex model structure will bring a higher complexity. In this work, we propose a lightweight, yet effective model called the Gated Adaptive feature Interaction Network (GAIN). We devise a novel cross module to drop meaningless feature interactions and preserve informative ones. Our cross module consists of multiple gated units, each of which can independently learn an arbitrary-order feature interaction. We combine the cross module with a deep module into GAIN and conduct comparative experiments with state-of-the-art models on two public datasets to verify its validity. Our experimental results show that GAIN can achieve a comparable or even better performance compared to its competitors. Furthermore, in order to verify the effectiveness of the feature interactions learned by GAIN, we transfer learned interactions to other models, such as Logistic Regression (LR) and Factorization Machines (FM), and find out that their performance can be significantly improved.

Keywords: recommendation system; factorization machines; feature interaction; deep learning

1. Introduction

With the explosion of mobile internet data, almost all Information Technology (IT) enterprises have gathered massive data, such as users’ personal information and their interactive information with the user interface, using their own Apps. These data can be used to learn interests or consumption habits of various users, so as to provide them with personalized recommendations, which has become one of the most critical profit engines. For example, content sharing platforms, such as YouTube and TikTok, increase user stickiness by pushing them videos which they may be interested in, and e-commerce platforms, such as Amazon and Taobao, facilitate transactions through the individual customized recommendations of commodities.

The Click-Through Rate, which is a critical indicator in a recommendation system, determines what content should be displayed and in what order. For those IT enterprises, even a slight improvement of 0.001 can be considered as practically significant [1,2] and will bring billions of dollars in extra revenue. Therefore, the CTR prediction task has always been a hot research spot for academia and industry.

Modeling a feature interaction is one of the most prevailing methods for the CTR prediction task. In an early study [3], feature interaction is also called cross feature, and a combination of different features is regarded as a new feature. However, meaningful cross features rely on the manual engineering of domain experts, which is usually a very time-consuming job. Although FM and its variants [4–6] make it possible to learn 2nd order interactions automatically, learning higher-order interactions remains impractical.
due to its high computational complexity. Inspired by the triumph of deep learning in computer vision [7,8], speech recognition [9,10] and natural language processing [11,12], many researchers resort to deep neural networks (DNNs) to learn high-order feature interactions and obtain decent results [13,14]. Nevertheless, some studies [15–17] point out that DNNs model feature interactions in an implicit way and can hardly learn interactions higher than the 2\text{nd} order. In recent years, integrated models which combine a deep module with a shallow module (here, we use the word “shallow” to distinguish from “deep”) have become the most prevailing solution.

The models that emerged in recent years can be seen as improvements to the above works and can be roughly divided into three categories. Models from the first category try to improve the model performance by ameliorating feature embeddings. All features should be embedded to vectors before interacting with others, and in typical CTR models, each feature usually corresponds to one embedding vector. Juan et al. [6] introduce the concept of a “feature field” and assign multiple vectors to each feature, so a feature can use different embedding vectors to interact with other features from different fields. Yang et al. [18] argue that each feature should use different representations when performing different operations and propose a new embedding method. Assigning multiple embedding vectors to one feature creates too many parameters, and thus affects the model’s efficiency. To solve this problem, Sun et al. [19] propose to use a field matrix between two feature vectors to model their interactions.

The models that form the second category are committed to the exploration of the interaction form. The inner product, outer product and Hadamard product are commonly used interaction operations in CTR models. Some works [20–22] state that these operations are too simple to effectively model feature interactions and propose their own interaction operations to obtain a better expressive effect. A self-attention mechanism [11], which can be regarded as an interaction operation, has been utilized in CTR prediction models [23–25]. In addition, Lang et al. [26] design a model to adaptively explore and optimize the interaction operations according to data.

Models from the third category work on ameliorating model architectures. Integrated models usually consist of a shallow module and deep module. Some models [5,20,27–29] use a single-tower architecture, in which two modules work in sequence, while other models [3,17,30–32] choose a dual-tower architecture, in which two modules work in parallel. Chen et al. [33] point out that insufficient sharing between two modules may limit the model’s expressiveness and effectiveness and devise a bridge module to enhance the layerwise information sharing. Song et al. [34] introduce a neural architecture search (NAS) to CTR models and utilize the learning-to-rank strategy to search the optimal architecture.

Zhu et al. [1] design an open benchmarking framework for CTR prediction models, which provides a fair comparison environment to evaluate the performance of different models. After fine-tuning and a thorough comparison, several interesting observations are worth noting. Firstly, in spite of the diversity of architectures and interaction manners, the differences in model performance are smaller than expected. Secondly, DNNs still serve as a strong baseline, and a plain multi-layer perception (MLP) performs better than complicated deep models [35,36]. Thirdly, the complex design of the architecture decreases the model’s efficiency and does not always lead to an improvement in the model performance. How to make a trade-off between complexity and efficiency is indeed a very challenging task.

In this paper, we propose a lightweight, yet efficient feature interaction model, whose architecture is shown in Figure 1. As the core of our model, the cross (shallow) module is composed of multiple interaction units, which learn feature interactions independently. Each unit preserves a gate for every feature field to decide whether the field will engage in an interaction. Each gate has only two statuses, closed or open, which are randomly initialized and adaptively updated according to the data. For each unit, the order is determined by the number of fields involved in the interaction, i.e., the number of open gates. Therefore, each unit can adaptively learn an arbitrary-order interaction without
assigning an order previously. Theoretically, the learning ability of our cross module is limited by the number of units. However, the experiments demonstrate that the model performance does not always upgrade with the number of units, and excessive units may bring down the prediction accuracy. Instead of considering all possible interactions, our cross module aims to learn useful ones which only take up a small proportion. In addition, our proposed model includes a deep module which is implemented by an MLP. The two modules work in parallel and share the same feature embeddings. A cross module can memorize frequent combinations which are discriminative to the prediction, while a deep module can generalize an interaction to unseen combinations. We name our model the Gated Adaptive feature Interaction Network, abbreviated GAIN. To summarize, we make the following contributions:

1. We devise a novel cross module to learn interactions. The structure of our cross module is quite simple; in contrast to shallow modules in other models which consist of multi-layer or complex connections, our cross module is composed of multiple gated units. Each unit works independently and can take full advantage of GPU parallel computing, which ensures the efficiency of our model.
2. We propose three strategies to learn useful feature interactions. Three strategies have their own characteristics and are suitable for different scenarios.
3. By assembling the cross module and deep module, we propose the Gated Adaptive feature Interaction Network (GAIN) to adaptively learn useful feature interactions.
4. We carry out extensive experiments on two real-world datasets to validate the effectiveness and efficiency of GAIN.

![Figure 1. Architecture of GAIN.](image)

The rest of this paper is organized as follows. Section 2 introduces the related works which are relevant to our proposed model which is described in detail in Section 3. We analyze our model from different aspects in Section 4. The experimental results on three real-world datasets are presented and discussed in Section 5, followed by Section 6, which provides a conclusion of our work.

2. Related Works

2.1. Feature Embedding

Real-world CTR datasets usually consist of multiple feature fields, and each field contains multiple features. In order to make full use of the information in the features, raw features should be transformed to expressive representations. In early studies, a common approach is to represent every feature as a one-hot vector. However, one-hot encoding brings sparsity to feature vectors, especially when a feature field contains a large number of features, which would result in an excessive memory assumption and a decrease in model efficiency. In fact, in a dataset with tens of millions of samples, it is very common for
a feature field to contain millions of features. As a result, one-hot encoding is no longer applicable for a nowadays web-scale dataset. Currently, the mainstream approach is feature embedding, i.e., encoding features to low-dimensional dense vectors. Categorical features and numerical features are usually processed separately.

2.1.1. Categorical Feature

For a categorical feature set \([F_1, F_2, \cdots, F_i, \cdots, F_{m'}]\), \(m'\) denotes the number of categorical feature fields, and the \(i_{th}\) feature field \(F_i\) contains \(n_i\) different features. A vocabulary consists of \(n_i\) vectors and is built for \(F_i\), then every feature can correspond to a unique dense vector.

In vector-wise feature interaction models [27,31,32,37,38], every feature interacts with others as a whole; hence, all features need to be embedded to dense vectors with the same length. Other bit-wise feature interaction models [17,39] argue that it is irrational to embed all features into the same length because the numbers of features in different fields vary widely and concatenate all feature embedding vectors to a one-dimension vector. Therefore, in these bit-wise models, the length of the embedding vector for different fields can be arbitrary.

In most models, each feature usually corresponds to only one vector, while in some models, such as Field-aware Factorization Machines (FFM) [6] and Operation-aware Neural Networks (ONN) [18], in order to enhance the model’s expressiveness, each feature corresponds to multiple vectors; hence, different embedding vectors can be used to interact with features from different fields. In Field-matrixed Factorization Machines (FmFM) [19], though each feature corresponds to one vector, a field matrix is proposed to interact with different field pairs, which enhance the expressiveness of the model while maintaining computational space and time efficiency.

2.1.2. Numerical Feature

For a numerical feature set \([F_1, F_2, \cdots, F_i, \cdots, F_{m''}]\), \(m''\) denotes the number of numerical feature fields. The value range of feature field \(F_i\) contains innumerable continuous values which make it impossible to encode the numerical features to dense vectors such as categorical features. The most widely used approach in CTR models is discretization, which aims to discretize numerical features to categorical ones. The value range of \(F_i\) can be partitioned into multiple segments denoted as \([Seg_1, Seg_2, \cdots, Seg_i, \cdots, Seg_{n_i}]\), where \(Seg_i\) denotes the \(i_{th}\) segment, and \(n_i\) denotes the number of segments of field \(F_i\). Each segment corresponds to a unique embedding vector. For \(Seg_i\), all values in it share the same embedding vector \(v_{num_i}\). However, this approach relies too much on partitioning strategies and can easily lead to local optima.

In models such as AutoInt [23], the Adaptive Factorization Network (AFN+) [32] and ContextNet [27], every numerical feature field is assigned a vector, and a feature embedding is represented with the production of its value and the vector of the field it belongs to. For example, the vector set of all numerical feature fields is denoted as \([v_{num_1}^1, v_{num_2}^2, \cdots, v_{num_i}^i, \cdots, v_{num_{m''}}^{m''}]\), and for an arbitrary value \(a\) of the \(i_{th}\) feature, its embedding vector can be represented as \(a \cdot v_{num_i}\).

AutoDis [40] initializes multiple meta-embeddings for every numerical field and assigns every feature with a unique vector through the weight-sum operation of multiple meta-embeddings. The deep learning recommendation model (DLRM) [28] concatenates all numerical features in a sample and transforms it to a vector with equal size to a categorical feature through an MLP.

Instead of encoding numerical features to a vector, some models maintain the original form of a scalar for every feature, e.g., Wide & Deep [3] uses original values, the Deep & Cross Network (DCN) [17] and DCNv2 [39] normalize all values of each field and the DNN for YouTube [14] utilizes several transformations of normalized values.
2.2. High-Order Feature Interaction

The Factorization Machine (FM) [4] assumes that every feature can be represented with a low-dimensional embedding vector; hence, the $2^{nd}$-order feature interaction can be implemented by using the inner product of two vectors. Although the FM can learn the feature interaction automatically, it can hardly learn an interaction higher than the $2^{nd}$ order because a large number of meaningless feature pairs involved in the computation procedure would raise the complexity.

It has been a consensus that learning high-order features interaction is critical for CTR prediction. Models such as Higher-Order Factorization Machines (HOFM) [41], DCN, xDeepFM [31], DCN-v2 and the Field-aware INTeraction Neural Network (FINT) [37] are proposed to model high-order feature interactions by stacking multiple cross layers. These models share three main drawbacks. Firstly, the model capacity is restricted by the number of cross layers which determines the maximum order. Secondly, excessive interactions brought by computation will degrade the model performance. Thirdly, in a stacked structure, feature interactions have to be calculated layer by layer, which will bring down the efficiency.

The Deep Interaction Machine (DeepIM) [38] introduces Newton’s identities and represents an arbitrary-order interaction in the form of an elementary symmetric polynomial, which can be calculated by the power sums of features embedding vectors. Using this trick, the original complex computation of multiple layers is simplified to an algebra problem. However, the trick is a two-edge sword; the symmetry of Newton’s identities makes it hard to differentiate the importance of interactions, which would limit the model’s expressiveness.

2.3. Selection of Useful Feature Interaction

The computation process of those models described in Section 2.2 involves all the possible interactions of bounded degree without distinguishing their importance, which would significantly increase the complexity and harm the performance. With the rise of the attention mechanism, many attention-based models are proposed to cope with this problem. Attentional Factorization Machines (AFM) [42] utilize the attention mechanism to enable interactions to contribute differently to the prediction. To enhance the model’s expressiveness, AutoInt [23] introduces a multi-head self-attention mechanism, which is updated to disentangled self-attention in Disentangled Self-Attentive Neural Networks (DESTINE) [24]. However, attention calculates the similarity of two vectors, and it cannot be used to represent the importance of the feature interaction.

Xue et al. [43] introduce the hashing algorithm to the feature interaction and propose AutoHash, whose core idea is to put $m$ features into $k$ buckets according to a hash function, each feature can be reused and the feature combination in each bucket represents a feature interaction. The hash function is determined by probabilistic distribution $p$, which is a learnable parameter. The authors believe that AutoHash can adaptively learn useful feature interactions in a data-driven way.

An automatic feature interaction selection (AutoFIS) [44] proposes a two-stage algorithm and assigns a gate for every possible interaction to control whether the interaction would be calculated in the next step. In the search stage, the importance of every interaction is represented as an architecture parameter $\alpha$ that can be optimized by gradient descent. Those interactions whose corresponding $\alpha$ equals to zero would be regarded as useless and then be dropped. In the re-train stage, the $\alpha$ of the remaining interactions will serve as attention units and the model will be trained for the second time.

Chen et al. [45] propose a Bayesian higher-order feature interaction selection (BH-FIS) to perform the FIS. First, the outer product and masking techniques are utilized to enumerate all feature interactions. Second, variational inference networks are leveraged to calculate coefficients for all feature interactions. AFN+ [32] represents a feature interaction in the form of a power product of features and projects embedding vectors into a logarithmic space. Logarithmic neurons are devised to learn useful interactions. Each
neuron is independent from others and can learn arbitrary-order interactions adaptively. Although logarithmic transformation can convert the operation of a power product to a weighted sum, it imposes constraints on the values of input because the log function requires positive inputs. AFN+ adopts the method of taking the absolute value of the original input, which not only might lead to a collision but also influence the semantic of embedding vectors.

Our proposed model GAIN is inspired by AutoFIS and AFN+. We assume that only a minority of useful interactions would bring about appreciable influence to the prediction accuracy, and these useful interactions can be learned through training. Unlike AutoFIS, GAIN accomplishes searching and training in one stage. There is no need for it to consider all the possible interactions as useful ones can be learned automatically. Compared with AFN+, GAIN restricts the value of the power exponent of the embedding vectors to 0 or 1 and relaxes the restriction on the value of the inputs.

3. Gated Adaptive Feature Interaction Network

GAIN applies the classic structure in CTR prediction task which consists of embedding layer, cross module, deep module and prediction layer. Cross module and deep module work in parallel and share the same inputs from embedding layer. Outputs of the two modules are passed to the prediction layer. In this section, we will describe these components in detail.

3.1. Embedding Layer

GAIN assembles two fashions of feature interaction. The cross module models explicit interaction at vector-wise level, which requires that all features must be embedded to vectors with same length. The deep module models implicit interaction at bit-wise level, which has no requirements on the length of embedding vectors. Some studies [6,18] illustrate that using different representations for different operations can lead to better performance. However, parameters in embedding layer account for a high proportion of total parameters in the model, and using different embedding vectors in two modules will lead to a significant increase in model’s complexity.

To facilitate interaction with others, all features, including numerical and categorical features, are encoded to dense vectors with equal size. First, numerical features are converted to categorical features through discretization. After that, a vocabulary is randomly initialized for each field, and every feature is represented with a unique vector by means of a look-up table. At last, all feature embedding vectors are assembled to form the input of cross module and deep module, respectively. The input of cross module is represented as a matrix $X_{\text{cross}} \in \mathbb{R}^{d \times m}$, where $d$ and $m$ represent the dimension of embedding vectors and the number of feature fields, respectively. The input of deep module is represented as a one-dimension vector $X_{\text{deep}} = \text{Cat}[v_1, \cdots, v_i, \cdots, v_m]$, where $v_i$ is embedding vector of the $i$th feature.

Uniqueness is the basic requirement of feature embedding; besides that, every embedding vector should carry certain information, i.e., features co-occurring frequently in positive samples should show some similarity in latent representation space. Although randomly initialized vectors can ensure uniqueness, they cannot carry useful information which is deeply buried in data. GAIN can be trained end to end, and the embedding layer is jointly optimized with other components, so as all embedding vectors can be updated according to data.

3.2. Cross Network

The cross module is composed of multiple gated units, each of which can adaptively learn a unique feature interaction. Each unit consists of $n$ gates, and each gate controls whether the corresponding feature engages in the interaction. Every single feature is $1^{st}$ order, and the combination of different features is regarded as the feature interaction and
the number of features in the combinations is the order of interaction. A generalized definition of feature interaction is given as below.

**Definition 1.** Let $\mathcal{F} = [F_1, \cdots, F_i, \cdots, F_m]$ denote the collection of feature fields, the combination of $k$ features $F_{t_1} = \alpha, F_{t_2} = \beta, \cdots, F_{t_k} = k$ is a $k_{th}$-order feature interaction, where $1 \leq t_1, t_2, \cdots, t_k \leq m$, and $F_i$ represents the $i_{th}$ field.

To facilitate the multiplication operation, a definition based on embedding vectors is given as below.

**Definition 2.** Let $\mathbf{V}^{d \times m}$ denote the embedding matrix of an instance in datasets, and $v_i$ denotes the $i_{th}$ row of $\mathbf{V}^{d \times m}$. Then, a $k_{th}$-order feature interaction can be represented as

$$FI^{(k)} = \prod_{i=1}^{m} v_i^{g_i}$$

(1)

where $k = \sum_{i=1}^{m} g_i$ is the interaction order, and $g_i \in \{0, 1\}$ is the gate status.

### 3.2.1. Gated Unit

A gating mechanism is introduced into the cross module, each unit maintains $m$ gates and each gate controls the engagement of one feature field. There are only two statuses for gate, closed or open, which indicates the corresponding feature will be dropped or reserved, respectively. Combination of $m$ gates in a unit can be regarded as a feature interaction. Each unit can learn a unique feature interaction, and expressiveness of our proposed model is determined by the number of units $n$. However, our experimental results demonstrate that model performance does not always improve with the increase in $n$; therefore, finding an appropriate $n$ is crucial to GAIN.

The statuses of gates are initialized randomly and updated dynamically by gradient decent. For convenience of calculation, the statuses of gates in a unit are represented as a multi-hot vector, in which open and closed gates are denoted as 1 and 0, respectively. However, this manner of representation cannot be directly used in training phase because discrete statuses cannot propagate gradients. To overcome this problem, a continuous relaxation technology is introduced to the status of gates. Firstly, we randomly initialize a weight vector for each unit, then a softmax function is utilized to transform the weight vector to a probability distribution, which is called “soft” vector (by contrast, the multi-hot vector is called “hard” vector). Secondly, “soft” vector is transformed to “hard” vector with specific feature selection strategy (which will be described in Section 3.2.2). In “hard” vector, a value of 1 indicates that its corresponding feature will engage in interaction. In the forward phase of training, “hard” vector is used to determine which features will be selected, while in the backward phase, gradients will not be backpropagated to “hard” vector but “soft” vector. In particular, given an initialized weight vector $[0.8, 0.6, 0.5, 0.1, 0.2]$, the “soft” vector $[0.28, 0.23, 0.20, 0.14, 0.15]$ can be easily calculated. By applying a threshold strategy which selects features whose probabilities greater than 0.20, we obtain the “hard” vector $[1, 1, 1, 0, 0]$, which is passed forward to calculate feature interaction. In the backward phase, the gradients are backpropagated to the “soft” vector and the weight vector in sequence, skipping the “hard” vector. Assuming that the weight vector is updated to $[0.9, 0.5, 0.2, 0.1, 0.5]$, the “soft” vector for next minibatch becomes $[0.31, 0.20, 0.15, 0.14, 0.20]$, and the “hard” vector becomes $[1, 1, 0, 0, 1]$. During training, the weights are constantly updated and eventually stabilized, which will help the units to learn meaningful feature interactions. Those features which are more discriminative to prediction will accumulate more gradients and will be more likely to be selected. The data flow of a gated unit is shown in Figure 2.

In a gated unit, engagements of features are determined by “soft” vector, but only those with higher probabilities will have the chance to engage in interaction, while features with
lower probabilities will lose their chance to be selected, which will not only make the “soft” vector lose the meaning of probability but also lead to local optima. Gumbel-softmax [46], a reparameterization trick, is introduced to cope with this problem. Gumbel-softmax trick is implemented with two steps.

Firstly, random numbers from a Gumbel distribution are added to every element of “soft” vector. The “soft” vector can be represented as

\[ S = [s_1, \ldots, s_i, \ldots, s_m]. \]

The random numbers from a Gumbel distribution can be represented as

\[ \varphi_i = -\log(-\log(\mu_i)), \]

where \( \mu_i \in U(0, 1) \) is uniformly distributed random numbers. Then, the new “soft” vector can be represented as

\[ S' = [s_1 + \lambda \varphi_1, \ldots, s_i + \lambda \varphi_i, \ldots, s_m + \lambda \varphi_m], \] (2)

where \( \lambda \) is the coefficient of the Gumbel random numbers, which will be discussed in Section 5.3. Random numbers make all features have opportunities to engage in interaction and can help GAIN escape from local optima.

Secondly, a softmax function is leveraged to transform the new “soft” vector to a probability distribution which can be represented as

\[ S = [p_1, p_2, \ldots, p_i, \ldots, p_m], \] (3)

where \( p_i = \frac{e^{s_i / \tau}}{\sum_{j=1}^{m} e^{s_j / \tau}}, \) and \( \tau \) is the temperature parameter, when \( \tau \to 0, \) \( p_i \) becomes a one-hot vector, and when \( \tau \to 1, \) \( p_i \) becomes a uniform distribution.

The significance of the Gumbel-softmax trick is that it not only retains the meaning of probability, that is, features with greater probabilities have greater probabilities of

---

**Figure 2.** Data flow of a gated unit. Solid line and dashed line denote forward phase and backward phase of training, respectively. At the beginning of training, weights of feature fields are randomly initialized. In the forward phase, “soft” vector is calculated using the trick of Gumbel-softmax, and then “hard” vector is calculated using feature selection strategy. After that, the selected features can participate in interaction. In the backward phase, the gradients are backpropagated step by step, skipping the “hard” vector.

Secondly, a softmax function is leveraged to transform the new “soft” vector to a probability distribution which can be represented as

\[ S = [p_1, p_2, \ldots, p_i, \ldots, p_m], \] (3)

where \( p_i = \frac{e^{s_i / \tau}}{\sum_{j=1}^{m} e^{s_j / \tau}}, \) and \( \tau \) is the temperature parameter, when \( \tau \to 0, \) \( p_i \) becomes a one-hot vector, and when \( \tau \to 1, \) \( p_i \) becomes a uniform distribution.

The significance of the Gumbel-softmax trick is that it not only retains the meaning of probability, that is, features with greater probabilities have greater probabilities of
engagement in the interaction, but also makes the features with small probabilities have small chances to stand out. Note that the Gumbel-softmax function is replaced with a softmax function in the inference phase, which means that features are selected based on the probability distribution generated by the latest weight vector.

3.2.2. Feature Interaction Selection

We design three strategies to select features from each gated unit. The first one is top-$k$ strategy, which requires the order $k$ to be specified in advance for each unit. In each unit, features with largest $k$ probabilities will be selected. For convenience of representation, the order of all units can be specified in batches, e.g., dictionary $\{3$:$10, 4$:$20, 5$:$10\}$ defines an allocation scheme of $k$, in which the keys represent the order and the values represent the number of units. With top-$k$ strategy, we can manipulate our model to learn arbitrary high-order feature interactions as our wish. The process of FIS using top-$k$ strategy in a gated unit is described in Algorithm 1.

**Algorithm 1:** FIS using top-$k$ strategy.

**Input:** Dictionary of orders: $D$.

**Output:** Hard vector of gated unit: $H$.

1. $W \leftarrow$ Randomly initialize weights of features;
2. $S \leftarrow$ Compute Gumbel-softmax of $W$;
3. $k \leftarrow$ Get order of unit $u$ from $D$;
4. $index(k) \leftarrow$ indexes of the top-$k$ probabilities in $S$;
5. $H \leftarrow$ Generate multi-hot vector based on $index(k)$.

The second one is threshold strategy, which requires a global threshold $\epsilon$. In each unit, features with probabilities greater than $\epsilon$ will be selected. The order of a unit is determined by the number of selected features, and may be different from the order of others. The orders of all units follow a Gumbel distribution which is relevant to the value of $\epsilon$, and the effect of the $\epsilon$ on the distribution of orders will be discussed in Section 5.3. With threshold strategy, we do not need to assign orders in advance, gated units can adaptively learn feature interactions and the orders can be arbitrary. The process of FIS using top-$k$ strategy in a gated unit is described in Algorithm 2.

**Algorithm 2:** FIS using threshold strategy.

**Input:** Number of feature fields: $m$; Threshold: $\epsilon$.

**Output:** Hard vector of gated unit: $H$.

1. $W \leftarrow$ Randomly initialize weights of features;
2. $S \leftarrow$ Compute Gumbel-softmax of $W$;
3. for $i = 1$ to $m$ do
   if $S[i] > \epsilon$ then
     $H[i] \leftarrow 1$.

The third one is hybrid strategy, which requires both threshold and allocation scheme of orders. Top-$k$ strategy and threshold strategy are applied successively. Take a unit for an example. Firstly, largest $k$ probabilities are coarsely selected. Secondly, probabilities less than threshold $\epsilon$ are filtered out. With hybrid strategy, our proposed model can not only reduce the impact of less important features on model performance but also control the orders within a reasonable range. The process of FIS using hybrid strategy in a gated unit is described in Algorithm 3.
Algorithm 3: FIS using hybrid strategy.

**Input:** Number of feature fields: \( m \); Threshold: \( \varepsilon \); Dictionary of orders: \( D \).

**Output:** Hard vector of gated unit: \( H \).

\[ \mathbb{W} \leftarrow \text{Randomly initialize weights of features}; \]
\[ \mathbb{S} \leftarrow \text{Compute Gumbel-softmax of } \mathbb{W}; \]
\[ k \leftarrow \text{Get order of unit } u \text{ from } D; \]
\[ \text{index}^{(k)} \leftarrow \text{indexes of the top-}k \text{ probabilities in } \mathbb{S}; \]

\[ \text{for } i \text{ in } \text{index}^{(k)} \text{ do} \]
\[ \text{if } \mathbb{S}[i] > \varepsilon \text{ then} \]
\[ H[i] \leftarrow 1. \]

All of the three strategies can be used to adaptively learn useful feature interactions and have different application scenarios. Performance of the top-\( k \) strategy relies heavily on elaborate figuration of allocation scheme of \( k \), which makes it more suitable for specific datasets with some prior knowledge. Threshold strategy which just requires a threshold can be easily applied on unfamiliar or desensitization datasets. Hybrid strategy combines the strengths of the two former strategies and are suitable for a variety of different scenarios.

3.2.3. Multiplication Operation for Feature Interaction

Commonly used multiplication operations for feature interaction include inner product, outer product and Hadamard product. Inner product is widely used for its simplification and efficiency; meanwhile, it is criticized for its expressiveness as the result turns out to be a scalar. Outer product, which turns out to be a matrix, although boosts the expressiveness, brings huge computational complexity. Hadamard product is used in our model, because its result is still a vector, and it can be regarded as a trade-off between expressiveness and efficiency.

Most existing models depict the feature multiplication as a binary operation, hence learning \( k \)-order interaction needs to conduct multiplication operation for \( k - 1 \) times. This may explain why those models stack multiple layers to learn high-order interactions. In order to devise a lightweight but efficient model, it is necessary to generalize the binary operation to a multivariate operation. The definition of multivariate operation of Hadamard product is given as below.

**Definition 3.** Let \( V = [V_1, V_2, \cdots, V_j, \cdots, V_k] \) denote the \( k \) features selected by a gated unit, and \( V_i = [v_{i,1}, v_{i,2}, \cdots, v_{i,d}] \) denotes the \( i \)-th feature, then Hadamard product of these \( k \) features can be represented as

\[
\mathcal{H}P^{(k)} = V_1 \otimes V_2 \otimes \cdots \otimes V_k = [\prod_{i=1}^{k} v_{i,1}, \prod_{i=1}^{k} v_{i,2}, \cdots, \prod_{i=1}^{k} v_{i,d}],
\]

where \( \otimes \) represents Hadamard product, \( k \) and \( d \) represent the interaction order and embedding dimension, respectively.

\( \mathcal{H}P^{(k)} \), the output of a gated unit, remains a \( d \)-dimension vector. All units’ outputs are concatenated to form a one-dimension vector. Subsequently, a full-connected layer transforms the long vector to a short one to be the output of the cross module, which is represented as

\[
\text{CrossNet}(X_{\text{cross}}) = \text{Linear}([\mathcal{H}P^{(k)}_1, \cdots, \mathcal{H}P^{(k)}_i, \cdots, \mathcal{H}P^{(k)}_n]),
\]

where \( n \) is the number of gated units.
3.3. Deep Network

Comparing with image in CV and text in NLP, the data in CTR field lack the spatial or temporal relationship. Those modern DNNs models which excel in CV or NLP have not shown any superiority in CTR prediction task. As a result, we choose plain MLP as the deep module of our model, which is represented as

$$MLP(X_{\text{deep}}) = \text{Linear}(\text{MLPBlock}(\cdots \text{MLPBlock}(X_{\text{deep}})))$$, \hspace{1cm} (6)

$$\text{MLPBlock}(x) = \text{Dropout}(\text{relu}(\text{BN}(\text{Linear}(x))))$$.

3.4. Modules Combination

The cross module and deep module are used to learn explicit and implicit interactions, respectively, and both of them are informative to the prediction. The outputs of the two modules are concatenated as a vector, and then fed into the prediction layer, which is represented as

$$\hat{y} = \text{Sigmoid}(\text{Linear}(\text{Cat}[\text{CrossNet}(X_{\text{cross}}), \text{MLP}(X_{\text{deep}})]))$$. \hspace{1cm} (7)

CTR prediction task is a binary classification problem, which can be trained with binary cross entropy (Logloss). The loss function of our model is represented as

$$L(y, \hat{y}) = \frac{1}{N} \sum (-y \cdot \log(\hat{y}) - (1 - y) \cdot \log(1 - \hat{y})) + \lambda \| \Theta \|$$, \hspace{1cm} (8)

where $y$ is the ground truth, $N$ denotes the size of training set, $\lambda$ denotes the regularization term and $\| \Theta \|$ denotes the set of parameters in the model.

4. Model Analysis

4.1. Power Exponent of Feature

The Hadamard product, which calculates the production of all elements in the same position, can be generalized to the power product, then Equation (4) can be modified as

$$\mathcal{H}P^{(k)} = V_1 \otimes V_2 \otimes \cdots \otimes V_k = [\prod_{i=1}^{k} v_{1,1}^{w_{1,i}} \prod_{i=1}^{k} v_{1,2}^{w_{1,i}} \cdots \prod_{i=1}^{k} v_{d,1}^{w_{1,i}}]$$, \hspace{1cm} (9)

where $w$ represents the power exponent of the feature. Each feature has its own $w$, and all elements of a feature vector share the same $w$.

AFN+ considers the power exponent of a feature vector as a scaling operation. Hence, $w$ is regarded as a scaling factor and can be any value, even a decimal or negative number. Although it seems that introducing power enhances the model’s expressiveness, the power operation requires all inputs to be positive, which severely restricts the embedding representation of the features. The latent space of the embedding vector is compressed to $1/d^2$ of its original size, which will weaken the information carried by the embedding vectors and result in a downgrading of the model performance.

Our proposed model simplifies Equation (9) by restricting the range of the power exponent to $\{0, 1\}$. The value 1 indicates that the feature will be multiplied, while 0 indicates the opposite. The simplification makes the model of great interpretability.

4.2. Complexity Analysis of Cross Network

Our proposed model is mainly inspired by AFN+; hence, we take AFN+ as the comparison target.

**Time Complexity.** Recall that we use $m$ to denote the number of feature fields, $d$ to denote the dimension of the embedding, $k$ to denote the number of the selected features in the gated unit and $n$ to denote the number of gated units. The overall time complexity of the cross module in GAIN is $O(nkd + nm)$, while that of AFN+ is $O(lmd)$, where $l$ denotes
the number of logarithmic neurons. Because \( n \ll l \) and \( k \ll m \), it makes GAIN train faster than AFN+.

**Space Complexity.** For the ease of comparison, the parameters for the batch norm and bias are omitted. The total number of parameters in the cross module of GAIN is \( nm + nd \), while that of AFN+ is \( lm + ldh + (h - 1)t^2 + t \), where \( h \) and \( t \) denote the number of hidden layers and the neurons in each layer, respectively. Because \( n \ll l \) and the MLP in AFN+ is replaced by a single linear layer in GAIN, the space complexity of the cross module in GAIN is far less than in AFN+.

5. Experiments

Extensive experiments on two real-world datasets are conducted in this section to answer the following research questions:

- **RQ1:** How does GAIN perform compared with those state-of-the-art models?
- **RQ2:** How do hyperparameters influence the model performance?
- **RQ3:** Can GAIN learn useful feature interactions?

5.1. Experiment Setup

Zhu et al. [1] propose an open benchmarking framework for CTR prediction models, which adopts the same data partition, preprocessing and evaluation protocols to make models compete in a fair environment. It is worth noting that, after elaborate tuning, all models perform better than their reported result. Our proposed GAIN is also implemented under this framework.

**Datasets.** Comparative experiments are conducted on two public datasets, Criteo and Avazu, which both come from the Kaggle competition and are widely used in the CTR prediction task. Criteo consists of 39 features, including 26 categorical features and 13 numerical features, 45.8 million samples in total, and the proportion of positive samples is about 25.6%. Avazu consists of 22 categorical features, 36 million samples in total, and the proportion of positive samples is about 17%. The hyperparameters tuning experiments are conducted on Frappe, which consists of 10 categorical features, 288 thousand samples in total, and the proportion of positive samples is about 33.4%.

**Baseline models.** To make a fair comparison, we choose those models which perform best under the benchmarking framework as the baseline. Each model has been fine-tuned and outperforms its reported result by a large margin.

- **Wide & Deep** [3]. Wide & Deep assembles two components in parallel. The wide component is composed of a linear combination of raw features and manually designed features. The deep component is implemented by a plain MLP.
- **IPNN** [5]. IPNN assembles two components in serial. The inner product of the feature embedding vectors are fed to the DNN.
- **DeepFM** [30]. DeepFM replaces the wide component of Wide & Deep with FM which can automatically learn 2\textit{nd}-order interactions.
- **DCN** [17]. DCN stacks multiple cross layers to learn high-order interactions. Due to its high computational complexity, DCN can hardly learn higher interactions than the 2\textit{nd} order.
- **xDeepFM** [31]. xDeepFM improves the DCN to learn truly high-order interactions.
- **AFN+** [32]. AFN+ exploits numerous logarithmic neurons to adaptively learn different high-order interactions.

Although the ONN [18] is demonstrated to significantly improve the prediction accuracy, its high time complexity and memory occupation make it hardly train in our GPU (a single GeForce RTX 3090 with 24G ram). Hence, this model is excluded from our experiments.

**Data preprocessing.** Criteo and Avazu are randomly split into 8:1:1 as the training set, validation set and test set, respectively, while Frappe is split into 7:2:1. In Criteo, the numerical features are discretized to the categorical features, applying log transformation which transforms \( x \) to \( \lfloor \log^2(x) \rfloor \) if \( x > 2 \). The minimum count of features is set to 10,
i.e., infrequent features which appear less than 10 times are replaced with “OOV”. In Avazu, the id field is dropped, and the timestamp field is transformed into three features: hour, weekday and weekend. The minimum count is set to 2.

**Evaluation metrics.** Two metrics, AUC and Logloss, which were widely used in the binary classification task, are adopted to evaluate the model performance. Adam is chosen to be the optimization algorithm, and the learning rate is initialized to 0.001.

**Implementation details.** With respect to the memory limitation of the GPU, the embedding dimension is set to 16, and the mini-batch size is set to 5000 for Criteo and 10,000 for Avazu. All baseline models are fine-tuned with specific hyperparameters for a fair comparison. More details can be found in our GitHub (https://github.com/YaoxunLiu/GAIN, accessed on 18 July 2022).

5.2. Performance Comparison (RQ1)

The comparison between GAIN and the baseline models is shown in Table 1. In addition to the Logloss and AUC, we also introduce two metrics, the FLOPs and number of parameters to, respectively, represent the complexity and size of the models. Each model is independently run five times on each dataset with the random seed varying from 2018 to 2022, and the average performance (with their standard deviation in parentheses) is shown in the two leftmost columns. Because the parameters in the embedding layer occupy the vast majority of the parameters in each model, and all the models use the same embedding method, we exclude the embedding layer when counting the parameters.

| Model          | Criteo          | Avazu           |
|----------------|-----------------|-----------------|
|                | Logloss         | AUC             | FLOPs | Parameters |
| Wide & Deep    | 0.4380(3.9E-4)  | 0.8138(3.2E-4)  | 4.63 M | 32.86 M    |
| IPNN           | 0.4382(5.1E-4)  | 0.8137(6.3E-4)  | 2.39 M | 2.36 M     |
| DeepFM         | 0.4380(6.5E-5)  | 0.8139(5.5E-5)  | 4.63 M | 32.86 M    |
| DCN            | 0.4384(2.1E-4)  | 0.8131(2.6E-4)  | 4.63 M | 4.62 M     |
| xDeepFM        | 0.4382(9.5E-5)  | 0.8136(8.4E-5)  | 11.35 M| 33.27 M    |
| AFN+           | 0.4386(9.6E-4)  | 0.8130(6.2E-4)  | 27.72 M| 26.88 M    |
| GAIN (top-k)   | 0.4379(4.8E-4)  | 0.8140(5.2E-4)  | 1.96 M | 1.95 M     |
| GAIN (threshold)| 0.4377(3.0E-4)  | 0.8142(3.7E-4)  | 1.96 M | 1.95 M     |
| GAIN (hybrid)  | 0.4376(2.8E-4)  | 0.8144(1.2E-4)  | 1.96 M | 1.95 M     |

**Model Effectiveness.** The experimental results show that our proposed GAIN outperforms all baselines on Criteo. Concretely, comparing with the best baseline model DeepFM, GAIN with the hybrid strategy increases the AUC by 0.049% and decreases the Logloss by 0.069%. On Avazu, GAIN achieves comparable results to the best baseline xDeepFM. All-
though GAIN does not outperform its competitors by a large margin, its model complexity is much lower. Comparing with the best performing baseline model on Criteo, the FLOPs of GAIN on Criteo are 42.3% of that of DeepFM, and the number of parameters of GAIN is only 5.9% of that of DeepFM. A similar situation also occurs in Avazu.

Three strategies for GAIN all behave well on both datasets. The threshold strategy can be easily applied on different datasets because it only relies on one hyperparameter, the threshold $\epsilon$; however, the upper bound of the model performance is limited because the order distribution generated by the threshold cannot guarantee to contain the optimal schema. Theoretically, the top-$k$ strategy can outperform other strategies, yet the huge search space brought by countless combinations of units and orders makes it difficult to find out the optimal schema. By combining the two strategies, the hybrid strategy eliminates their drawbacks and performs best on both datasets. Comparing with the best performing baseline models, the standard deviation of the performance of GAIN is larger. We attribute it to the weight vector in the gated units of GAIN. At the beginning of training, the weights of all features are randomly initialized in each unit, which will affect the selection of the features. Therefore, a different random seed may result in different feature interactions and various model performances. In future work, we will improve our model to reduce the effect of the random seed.

**Model Efficiency.** Efficiency is as important as effectiveness when considering the actual deployment of a model. The memory occupation and time consumption of different models in the training phase are shown in Table 2, and we can observe that the memory occupation of most models is not very different, and GAIN runs faster than the other models. Comparing with the training time, the inference time is more important, especially when the model is asked to respond to users’ requests with low latency. We devise an experiment to compare the training and inference time among different models. For a fair comparison, we set the number of batches to 100 and the batch size to 2000. The experiments for the training and inference time are conducted separately and independently, and the GPU will be warmed-up previously. The results are shown in Figure 3. Apparently, GAIN runs faster both in the training and inference phase. In order to verify GAIN’s efficiency in an online recommendation task, we set the batch size to 1 and perform inference operations on the GPU or CPU alone. Each experiment is conducted 300 times to calculate the average inference time. We also conduct the same experiments on other models. The results in Figure 4 demonstrate the efficiency of GAIN.

**Table 2.** Efficiency comparison. The “Memory” columns record GPU’s memory occupation of different models, while the “Time/Epoch” columns record average time consumption per epoch.
Figure 3. Training and inference time for 100 batches (batch size = 2000) of different models. (a) Comparison on Criteo. (b) Comparison on Avazu.

Figure 4. Average inference time of GPU and CPU (batch size = 1) of different models. (a) Comparison on Criteo. (b) Comparison on Avazu.

5.3. Hyperparameter Investigation (RQ2)

In this section, we investigate the effectiveness of three hyperparameters. All the experiments are conducted on Frappe because the time consumption per epoch of the model on this dataset is much less than that of Criteo or Avazu due to its mini size, while the effectiveness of the hyperparameters is similar.

Threshold $\epsilon$. The threshold $\epsilon$ is the most important hyperparameter of the threshold strategy because it determines the distribution of orders. In addition, $\lambda$ in Equation (2) also plays an important part. The influences of $\epsilon$ and $\lambda$ are investigated through a Monte Carlo simulation and shown in Figure 5. Figure 5a provides the relationship curve of $\epsilon$ and the mean of orders in different $\lambda$. Apparently, the mean of orders decreases with the increasing $\epsilon$. The curve of red and blue represents $\lambda = 0.1$ and $\lambda = 1$, respectively. We set $\lambda$ to 0.1 for two reasons. First, the steep curve of $\lambda = 0.1$ compresses the threshold $\epsilon$ to a smaller range. Second, the relationship between the mean of orders and $\epsilon$ is almost linear in the condition of $\lambda = 0.1$. The experiments prove that setting $\lambda$ to 0.1 can narrow the search space of $\epsilon$, and hence significantly improve the searching efficiency. We simulate the effect of different $\epsilon$ on the order distribution of 100 gated units, and the result of the simulation is shown in Figure 3b, from which we have two observations. First, the orders follow the Gumbel distribution. Second, the orders’ distribution becomes more concentrated when $\epsilon$ increases. For Frappe, which consists of 10 features, useful high-order interactions are mostly concentrated in the range $[3, 7]$. Therefore, we set the mean of orders to 5 and preset the value range of $\epsilon$ to $[0.0943, 0.0963]$ according to the red curve in Figure 5a. After enumerating all possible values in the range, we finally find out the best threshold $\epsilon = 0.0957$. 
Number of gated units $n$. Each gated unit can adaptively learn an arbitrary-order interaction. However, more units cannot always bring a better performance. Extensive experiments are conducted with a different number of units $n$ ranging from 1 to 1000, and the result is shown in Figure 6, in which the fold line of red and blue represents the AUC and Logloss, respectively. We can observe that the model performance fluctuates severely when $n \leq 80$, then gradually stabilizes when $80 < n \leq 500$ and finally decays when $n > 500$. Considering more units bring a high computational complexity to the model, we set $n$ to 40 for Frappe to make a trade-off between complexity and expressiveness.

Order $k$. The top-$k$ strategy requires an allocation of gated units. Because it is an allocation problem, countless possible combinations make it difficult to find out the best distribution scheme. To simplify this problem, firstly, we only take a single order into consideration. We conduct experiments on all possible mappings of $k$ and $n$ and consider the values of the Logloss and AUC as 2D coordinates, which are plotted in Figure 5. We use different sizes of dots to represent different numbers of units and use different colors to represent different $k$. The x-axis and y-axis represent the Logloss and AUC, respectively. It is worth noting that the x-axis is in a reverse direction; hence, the dots closer to the upper right corner indicate a better performance.

Besides simply allocating all units to a specific $k$ order, we can also make a combination of different $k$. In order to restrict the total number of units to 40, we choose small dots from the upper right zone to form a new distribution scheme, e.g., \{2':10, '3':20, '4':10\}, which consists of ten $2_{nd}$-order units, twenty $3_{rd}$-order units and ten $4_{th}$-order units. The performance of this scheme is plotted with a red star. From Figure 7, we observe that although simple schemes such as \{6':40\} or \{5':40\} bring decent results, a deliberate scheme can significantly improve the performance.
The feature combinations learned by gated units are saved in the form of a “hard” vector in which the selected features are represented by the value 1. The application of a “hard” vector not only brings good interpretability but also enables the learned feature interactions to be easily transferred to other models. In order to verify the effectiveness of learned feature interactions, we transfer these “hard” vectors to conventional CTR models, such as LR and FM. These learned interactions are regarded as manually designed features and high-order interactions in LR and FM, respectively. Experiments are conducted on the two original models and their enhanced versions, and the performance comparison is shown in Table 3. We can observe that feature interactions learned by GAIN can help to significantly improve the performance of LR and FM on both of the datasets. Due to the constraints of the theory and the structure of the model itself, the enhanced model cannot make the performance reach the SOTA level; however, the improved performance still demonstrates the effectiveness of the learned feature interactions.

Table 3. Performance of transferring interactions selected by GAIN to LR and FM.

| Model   | Logloss | AUC  | Logloss | AUC  |
|---------|---------|------|---------|------|
| LR      | 0.4567  | 0.7934 | 0.3815  | 0.7775 |
| FM      | 0.4431  | 0.8086 | 0.3754  | 0.7887 |
| LR(GAIN)| 0.4424  | 0.8051 | 0.3801  | 0.7808 |
| FM(GAIN)| 0.4418  | 0.8093 | 0.3746  | 0.7899 |

6. Conclusions and Future Work

In this work, we propose a novel model, the Gated Adaptive feature Interaction Network (GAIN), which can adaptively learn arbitrary-order feature interactions. Our model consists of four parts: (a) an embedding layer transforms raw inputs to dense vectors with equal length; (b) a cross module explicitly learns feature interactions using multiple gated units; (c) a deep module implicitly learns feature interactions using an MLP; and (d) a prediction layer concatenates the outputs of a cross and deep module and makes the prediction. As the core of the model, the cross module is composed of multiple gated units, each of which can independently learn arbitrary-order interactions in a data-driven manner. Extensive experiments were conducted on two public datasets, and the experimental results demonstrate that our model outperforms the baselines with lower model complexity, which verifies the effectiveness and efficiency of our model. In addition, feature interactions learned by our model can be transferred to LR and FM to improve their performance, proving that the feature interactions learned by the model are indeed useful. For future work, we will further optimize the model structure to reduce the impact of randomly initialized weights on model performance. Moreover, we intend to reduce our model’s complexity to deploy it on devices with limited computing resources.
Author Contributions: Conceptualization, Y.L. and L.M.; methodology and implementation, Y.L.; writing—original draft preparation, Y.L.; writing—review and editing, Y.L., L.M. and M.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The Criteo dataset supporting this study was obtained from https://www.kaggle.com/c/criteo-display-ad-challenge/data. The Avazu dataset was obtained from https://www.kaggle.com/c/avazu-ctr-prediction/data. The Frappe dataset was obtained from https://www.baltrunas.info/context-aware, accessed on 18 July 2022.

Acknowledgments: The authors would like to thank the anonymous reviewers for their constructive comments and suggestions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zhu, J.; Liu, J.; Yang, S.; Zhang, Q.; He, X. Open Benchmarking for Click-Through Rate Prediction. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, Queensland, Australia, 1–5 November 2021; pp. 2759–2769.
2. Ling, X.; Deng, W.; Gu, C.; Zhou, H.; Li, C.; Sun, F. Model ensemble for click prediction in bing search ads. In Proceedings of the 26th International Conference on World Wide Web Companion, Perth, Australia, 3–7 April 2017; pp. 689–698.
3. Cheng, H.-T.; Koc, L.; Harmsen, J.; Shaked, T.; Chandra, T.; Aradhye, H.; Anderson, G.; Corrado, G.; Chai, W.; Ispir, M. Wide & deep learning for recommender systems. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, New York, NY, USA, 15 September 2016; pp. 7–10.
4. Rendle, S. Factorization machines. In Proceedings of the 2010 IEEE International Conference on Data Mining, Sydney, Australia, 14–17 December 2010; pp. 905–1000.
5. Qu, Y.; Cai, H.; Ren, K.; Zhang, W.; Yu, Y.; Wen, Y.; Wang, J. Product-based neural networks for user response prediction. In Proceedings of the 2016 IEEE 16th International Conference on Data Mining, Barcelona, Spain, 12–15 December 2016; pp. 1149–1154.
6. Juan, Y.; Zhuang, Y.; Chin, W.-S.; Lin, C.-J. Field-aware factorization machines for CTR prediction. In Proceedings of the 10th ACM Conference on Recomender Systems, Boston, MA, USA, 15–19 September 2016; pp. 43–50.
7. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.
8. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. 2012, 25, 1097–1105. [CrossRef]
9. Graves, A.; Mohamed, A.; Hinton, G. Speech recognition with deep recurrent neural networks. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 6645–6649.
10. Tang, H.; Lu, L.; Kong, L.; Gimpel, K.; Livescu, K.; Dyer, C.; Smith, N.A.; Renals, S. End-to-end neural segmental models for speech recognition. IEEE J. Sel. Top. Signal Process. 2017, 11, 1254–1264. [CrossRef]
11. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is all you need. Adv. Neural Inf. Process. Syst. 2017, 30.
12. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv 2018, arXiv:1810.04805.
13. Gupta, U.; Wu, C.-J.; Wang, X.; Naumov, M.; Reagen, B.; Brooks, D.; Cottel, B.; Hazelwood, K.; Hempstead, M.; Jia, B. The architectural implications of facebook’s dnn-based personalized recommendation. In Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture, Seoul, Korea, 27 February–3 March 2021; pp. 488–501.
14. Covington, P.; Adams, J.; Sargin, E. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, 15–19 September 2016; pp. 191–198.
15. Beutel, A.; Covington, P.; Jain, S.; Xu, C.; Li, J.; Gatto, V.; Chi, E.H. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, Marina Del Rey, CA, USA, 5–9 February 2018; pp. 46–54.
16. Qin, Z.; Yan, L.; Zhuang, H.; Tay, Y.; Pasumarthi, R.K.; Wang, X.; Bendersky, M.; Najork, M. Are neural rankers still outperformed by gradient boosted decision trees? In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 3–7 May 2021.
17. Wang, R.; Fu, B.; Fu, G.; Wang, M. Deep & cross network for ad click predictions. In Proceedings of the ADKDD’17, Halifax, NS, Canada, 14 August 2017; pp. 1–7.
18. Yang, Y.; Xu, B.; Shen, S.; Shen, F.; Zhao, J. Operation-aware neural networks for user response prediction. Neural Netw. 2020, 121, 161–168. [CrossRef] [PubMed]
19. Sun, Y.; Pan, J.; Zhang, A.; Flores, A. FM2: Field-matrixed factorization machines for recommender systems. In Proceedings of the Web Conference 2021, New York, NY, USA, 19–23 April 2021; pp. 2828–2837.

20. Zou, D.; Zhang, L.; Mao, J.; Sheng, W. Feature Interaction based Neural Network for Click-Through Rate Prediction. arXiv 2020, arXiv:2006.05312.

21. Huang, T.; Zhang, Z.; Zhang, J. FiBiNET: Combining Feature Importance and Bilinear feature Interaction for Click-Through Rate Prediction. In Proceedings of the 13th ACM Conference on Recommender Systems. RecSys ’19: Thirteenth ACM Conference on Recommender Systems, Copenhagen, Denmark, 16–20 September 2019; Bogers, T., Said, A., Brusilovsky, P., Tikk, D., Eds.; ACM: New York, NY, USA, 2019; pp. 169–177, ISBN 9781450362436.

22. Wu, S.; Yu, F.; Yu, X.; Liu, Q.; Wang, L.; Tan, T.; Shao, J.; Huang, F. TFNet: Multi-Semantic Feature Interaction for CTR Prediction. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR ’20: The 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, China, 25–30 July 2020; Huang, J., Chang, Y., Cheng, X., Kamps, J., Murdock, V., Wen, J.–R., Liu, Y., Eds.; ACM: New York, NY, USA, 2020; pp. 1885–1888, ISBN 9781450380164.

23. Song, W.; Shi, C.; Xiao, Z.; Duan, Z.; Xu, Y.; Zhang, M.; Tang, J. AutoInt: Automatic feature interaction learning via self-attentive neural networks. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management, Beijing, China, 3–7 November 2019; pp. 1161–1170.

24. Xu, Y.; Zhu, Y.; Yu, F.; Liu, Q.; Wu, S. Disentangled self-attentive neural networks for click-through rate prediction. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, Virtual Event, Australia, 1–5 November 2021; pp. 3553–3557.

25. Chen, Z.; Zhong, F.; Chen, Z.; Zhang, X.; Pless, R.; Cheng, X. DCAPP: Deep Cross Attentional Product Network for User Response Prediction. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management. CIKM ’21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, QLD, Australia, 1–5 November 2021; Demartini, G., Zuccon, G., Culpepper, J.S., Huang, Z., Tong, H., Eds.; ACM: New York, NY, USA, 2021; pp. 221–230, ISBN 9781450384469.

26. Lang, L.; Zhu, Z.; Liu, X.; Zhao, J.; Xu, J.; Shan, M. Architecture and Operation Adaptive Network for Online Recommendations. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. KDD ’21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, 14–18 August 2021; Zhu, F., Chin Ooi, B., Miao, C., Wang, H., Skrypnik, I., Hsu, W., Chawla, S., Eds.; ACM: New York, NY, USA, 2021; pp. 3139–3149, ISBN 9781450383325.

27. Wang, Z.; She, Q.; Zhang, P.; Zang, J. ContextNet: A Click-Through Rate Prediction Framework Using Contextual information to Refine Feature Embedding. arXiv 2021, arXiv:2107.12025.

28. Naumov, M.; Mudigere, D.; Shi, H.-J.M.; Huang, J.; Sundaraman, N.; Park, J.; Wang, X.; Gupta, U.; Wu, C.-J.; Azzolini, A.G. Deep learning recommendation model for personalization and recommendation systems. arXiv 2019, arXiv:1906.00091.

29. Yu, R.; Ye, Y.; Liu, Q.; Wang, Z.; Yang, C.; Hu, Y.; Chen, E. XCrossNet: Feature Structure-Oriented Learning for Click-Through Rate Prediction. In Advances in Knowledge Discovery and Data Mining; Karlapalem, K., Cheng, H., Ramakrishnan, N., Agrawal, R.K., Reddy, P.K., Srivastava, J., Chakraborty, T., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 436–447. ISBN 978-3-030-75764-9.

30. Guo, H.; Tang, R.; Ye, Y.; Li, Z.; He, X. DeepFM: A factorization-machine based neural network for CTR prediction. arXiv 2017, arXiv:1703.04247.

31. Lian, J.; Zhou, X.; Zhang, F.; Chen, Z.; Xie, X.; Sun, G. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018; pp. 1754–1763.

32. Cheng, W.; Shen, Y.; Huang, L. Adaptive factorization network: Learning adaptive-order feature interactions. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; pp. 3609–3616.

33. Chen, B.; Wang, Y.; Liu, Z.; Tang, R.; Guo, W.; Zheng, H.; Yao, W.; Zhang, M.; He, X. Enhancing Explicit and Implicit Feature Interactions via Information Sharing for Parallel Deep CTR Models. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management. CIKM ’21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, QLD, Australia, 1–5 November 2021; Demartini, G., Zuccon, G., Culpepper, J.S., Huang, Z., Tong, H., Eds.; ACM: New York, NY, USA, 2021; pp. 3553–3557. ISBN 9781450384469.

34. Song, Q.; Cheng, D.; Zhou, H.; Yang, J.; Tian, Y.; Hu, X. Towards Automated Neural Interaction Discovery for Click-Through Rate Prediction. In Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. KDD ‘20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, 6–10 July 2020; Gupta, R., Liu, Y., Shah, M., Rajan, S., Tang, J., Prakash, B.A., Eds.; ACM: New York, NY, USA, 2020; pp. 945–955, ISBN 9781450379984.

35. Liu, B.; Tang, R.; Chen, Y.; Yu, J.; Guo, H.; Zhang, Y. Feature generation by convolutional neural network for click-through rate prediction. In Proceedings of the World Wide Web Conference, San Francisco, CA, USA, 13–17 May 2019; pp. 1119–1129.

36. Li, Z.; Cui, Z.; Wu, S.; Zhang, X.; Wang, L. Fi-gnn: Modeling feature interactions via graph neural networks for ctr prediction. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management, Lingotto, Italy, 22–26 October 2018; pp. 539–548.
38. Yu, F.; Liu, Z.; Liu, Q.; Zhang, H.; Wu, S.; Wang, L. Deep interaction machine: A simple but effective model for high-order feature interactions. In Proceedings of the 29th ACM International Conference on Information & Knowledge Management, Virtual Event, Ireland, 19–23 October 2020; pp. 2285–2288.

39. Wang, R.; Shivanna, R.; Cheng, D.; Jain, S.; Lin, D.; Hong, L.; Chi, E. DCN v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In Proceedings of the Web Conference 2021, New York, NY, USA, 19–23 April 2021; pp. 1785–1797.

40. Guo, H.; Chen, B.; Tang, R.; Zhang, W.; Li, Z.; He, X. An embedding learning framework for numerical features in ctr prediction. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, Virtual Event, Singapore, 14–18 August 2021; pp. 2910–2918.

41. Blondel, M.; Fujino, A.; Ueda, N.; Ishihata, M. Higher-order factorization machines. Adv. Neural Inf. Process. Syst. 2016, 29.

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

43. Xue, N.; Liu, B.; Guo, H.; Tang, R.; Zhou, F.; Zafeiriou, S.P.; Zhang, Y.; Wang, J.; Li, Z. AutoHash: Learning Higher-order Feature Interactions for Deep CTR Prediction. IEEE Trans. Knowl. Data Eng. 2020, 34, 2653–2666. [CrossRef]

44. Liu, B.; Zhu, C.; Li, G.; Zhang, W.; Lai, J.; Tang, R.; He, X.; Li, Z.; Yu, Y. AutoFis: Automatic feature interaction selection in factorization models for click-through rate prediction. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Virtual Event, CA, USA, 23–27 August 2020; pp. 2636–2645.

45. Chen, Y.; Wang, Y.; Ren, P.; Wang, M.; de Rijke, M. Bayesian feature interaction selection for factorization machines. Artif. Intell. 2022, 302, 103589. [CrossRef]

46. Jang, E.; Gu, S.; Poole, B. Categorical reparameterization with gumbel-softmax. arXiv 2016, arXiv:1611.01144.