# Field-Embedded Factorization Machines for Click-Through Rate Prediction

A Preprint

**Harshit Pande**[*]
Adobe
`pandeconscious@gmail.com`

## Abstract

Click-through rate (CTR) prediction models are common in many online applications such as digital advertising and recommender systems. Field-Aware Factorization Machine (FFM) and Field-weighted Factorization Machine (FwFM) are state-of-the-art among the shallow models for CTR prediction. Recently, many deep learning-based models have also been proposed. Among deeper models, DeepFM, xDeepFM, AutoInt+, and FiBiNet are state-of-the-art models. The deeper models combine a core architectural component, which learns explicit feature interactions, with a deep neural network (DNN) component. We propose a novel shallow Field-Embedded Factorization Machine (FEFM) and its deep counterpart Deep Field-Embedded Factorization Machine (DeepFEFM). FEFM learns symmetric matrix embeddings for each field pair along with the usual single vector embeddings for each feature. FEFM has significantly lower model complexity than FFM and roughly the same complexity as FwFM. FEFM also has insightful mathematical properties about important fields and field interactions. DeepFEFM combines the FEFM interaction vectors learned by the FEFM component with a DNN and is thus able to learn higher order interactions. We conducted comprehensive experiments over a wide range of hyperparameters on two large publicly available real-world datasets. When comparing test AUC and log loss, the results show that FEFM and DeepFEFM outperform the existing state-of-the-art shallow and deep models for CTR prediction tasks. We have made the code of FEFM and DeepFEFM available in the DeepCTR library(`https://github.com/shenweichen/DeepCTR`).

***Keywords*** factorization machines, recommender systems, feature interactions, deep learning CTR, matrix embeddings

## 1 Introduction

The click-through rate (CTR) prediction models play an important role in the rapidly growing multi-billion dollar online advertising industry [1, 2]. In a CTR prediction system, the main task is to predict the probability of click on an item shown to a user. Displaying items or advertisements of desired preference to a user not only boosts revenue but also enhances user satisfaction. The CTR data is usually multi-categorical and modeling interaction between features across fields leads to improved CTR prediction performance. Because of the high number of categories in such data, the number of unique features can be very high, often in the order of millions. Multiple studies have shown a link between lift in the CTR prediction performance and improvement in online CTR & increase in revenue [3, 4]. The studies show that a small lift in offline CTR performance leads to significant increase in revenue by improving the online CTR.

Shallow factorization machine-based models [5, 6, 7, 8], Bayesian methods [9], tensor factorization methods [10] and tree-based methods [11] have been used to model CTR prediction tasks. Given the breakthrough in deep neural networks [12], incorporation of deep learning for CTR prediction tasks is an active area of research [4, 13, 14, 15, 16, 3, 17, 18, 19, 20, 21, 22]. We discuss both the existing shallow and deep models in further detail in Section 2. A feature may behave differently depending on the field of its interacting feature. We refer to this property as field specificity. The shallow models Field-Aware Factorization Machine (FFM) [7] and Field-weighted Factorization Machine (FwFM)

[*]This work was done when the author was employed at Adobe

[8] use this property for improved performance. In deep learning models, FiBiNet [22] and Field-aware Probabilistic Embedding Neural Network (FPENN) [19] also use this property for higher performance. Neural networks that use layers designed to capture the properties of their respective data domains are successful. The success of convolutional layers [23] for images and Long shot-term memory(LSTM) layers [24] for text is partly due to their ability to exhibit the inductive bias needed for these respective domains. Field specificity is one such property of the CTR tasks that we also use to design the core layers of our proposed models. Another interesting property observed in the existing literature is that at most 3-order or 4-order interactions are needed to give the best CTR prediction performance, while further higher-order interactions lead to degradation in performance. The number of DNN layers is usually less than or equal to 3 and a similar depth is needed for the core layers proposed by different methods such as Deep Cross Network DCN [15], Compressed Interaction Network (CIN) in xDeepFM [21], Squeeze-Excitation network (SENET) module in FibiNet [22], and self-attention modules in AutoInt [20]. With these observations in mind, we also designed an architecture of our deep learning models to capture at most 3 or 4-order interactions, while designing the layers with inductive bias needed to model CTR prediction tasks.

Here we list the key contributions of this paper:

- We first introduce shallow Field Embedded Factorization Machine (FEFM), a novel factorization machine variant, that outperforms the existing state-of-the-art shallow models such as FM, FFM and FwFM on publicly available CTR datasets. We introduce symmetric field pair matrix embeddings to capture field specificity for feature interactions. FEFM has significantly lower model complexity (in terms of number of parameters) than FFM and similar model complexity as FM and FwFM.
- The eigen values of the matrix embeddings of FEFM provide an interpretable way of modeling field pair interactions. We define such field interaction strengths in terms of these eigen values and demonstrate that the fields appearing at the top of the list of strongest interactions are intuitively expected to do so.
- We extend our proposed shallow FEFM to a deeper variant called Deep Field Embedded Factorization Machine (DeepFEFM). DeepFEFM uses field pair matrix embeddings to generate FEFM interaction embeddings, which in combination with feature vector embeddings via skip concatenation layers generate higher order interactions. On doing extensive experiments on publicly available CTR datasets, we show that DeepFEFM consistently outperforms other state-of-the-art deep models such as DeepFM, xDeepFM, AutoInt, and FiBiNet.
- We compare our models and existing state-of-the-art models for multiple embedding dimensions to evaluate not only an overall best performing model but also to study the effect of embedding dimensions on different models. This is in contrast with most of the existing deep learning work, which compares models for only one embedding dimension. Interesting and novel outcomes arise as a result of this study.

The rest of the paper is organized as follows. In Section 2 we provide the mathematical notations, preliminaries and a discussion on the related work . In Section 3, we introduce our proposed models in detail. In Section 4, we present the experimental setup and the results on public data sets. The conclusions are presented in Section 5.

## 2 Preliminaries and Related work

Suppose there are $n$ unique fields and $m$ unique features across all the fields in a CTR prediction system where $F(i)$ is the field to which the $i^{th}$ feature belongs. In practice $n \ll m$. Suppose there are $N$ points in data with $d^{th}$ data point defined as $(y^{(d)}, x^{(d)})$, where $y^{(d)} \in \{1, -1\}$ (1 and $-1$ indicate click and no-click respectively) and $x^{(d)} \in \{0, 1\}^m$ is a multi-hot vector indicating $x_i^{(d)} = 1$ if feature i is active and $x_i^{(d)} = 0$ if feature i is inactive. Irrespective of the modeling technique, it is common to have the click probability modeled as a logistic function $\sigma(\phi) = \frac{1}{1+\exp(-\phi)}$. The optimization problem given in Equation 1 forms the basis of many well-known CTR prediction methods.

$$min_\theta \, \lambda \frac{\|\theta\|_2^2}{2} + \sum_{d=1}^{N} log(1 + exp(-y^{(d)}\phi(\theta, x^{(d)}))) \tag{1}$$

where $\lambda$ is the regularization strength. Different type of parameters may have different regularization strengths. For simplicity we show a single $\lambda$. The difference among methods lies in how each of the methods model the logit function function $\phi$ and what parameters $\theta$ are used in the models. A logistic regression (LR) model [5] serves as a baseline model and its logit function is shown in Equation 2.

$$\phi(\theta, x) = \phi_{LR}(w, x) = w_0 + \sum_{i=1}^{m} w_i x_i \tag{2}$$

The linear parameter $w$ is often also used in factorization machine-variant models (as a linear component) along with the specific parameters of the respective models.

Factorization Machine (FM) and variants [6, 7, 8] learn vector embeddings for each feature in the data and model the feature interactions in different ways. Equation 3 serves as a base equation for multiple shallow variants of FM. In practice the summation over large $m$ is not needed as only the number of features in the order of much smaller $n$ are active for one data point.

$$\phi(\theta, x) = \phi_{FMvariant}((v, w), x) = w_0 + \sum_{i=1}^{m} w_i x_i + \sum_{i=1}^{m} \sum_{j=i+1}^{m} Interaction(i, j) \tag{3}$$

$Interaction(i, j)$ models the interaction between the $i^{th}$ feature and the $j^{th}$ feature. The modeling of this interaction term varies for different FM-variants. Suppose the parameter $v_i$ is the vector embedding for the $i^{th}$ feature and $v_j$ for the $j^{th}$ feature. For FM [6], the interaction $Interaction(i, j)$ is modeled as $v_i^T v_j x_i x_j$. Field-Aware Factorization Machine (FFM) [7] outperform FM by incorporating field specificity. For every feature, FFM learns multiple embeddings, one separate embedding per field. The$Interaction(i, j)$ for FFM is $v_{i,F(j)}^T v_{j,F(i)} x_i x_j$. Here the parameter $v_{i,F(j)}$ is the vector embedding for the $i^{th}$ feature when it interacts with a feature from the field $F(j)$. Similarly, the parameter $v_{j,F(i)}$ is the vector embedding for the $j^{th}$ feature when it interacts with a feature from the field $F(i)$. A large number of parameters, leading to a high memory footprint, causes a major hindrance to the use of FFM in real-world production systems. Field-weighted Factorization Machine (FwFM) [8] extends FM by adding another parameter which models field specificity as the strength of field pair interactions. $Interaction(i, j)$ for FwFM is $v_i^T v_j r_{F(i),F(j)} x_i x_j$, where the scalar parameter $r_{F(i),F(j)}$ represents interaction strength between fields $F(i)$ and $F(j)$. FwFM uses only a scalar to model field specificity. There is potential of further improvement by using higher dimensional matrix embeddings for modeling field specificity, which we use in our proposed models.

Now, we discuss the related work in deep learning. The most unifying underlying property of these deep learning methods is the combination of features generated by their respective core layers with feed forward deep neural networks (DNN). The major difference lies in the design of the core layers. Neural factorization machine (NFM) [16], Deep Factorization Machine [3], and Factorization-machine supported Neural Networks (FNN) [13] use the combination of a factorization machine variant along with DNNs. Our proposed DeepFEFM shares some similarities with DeepFM and NFM. The DNN part of DeepFM takes input only the input feature embeddings, but we additionally also input FEFM generated interaction embeddings. Our ablation studies in Section 4 show that these FEFM interaction embeddings are key to the lift in performance achieved by DeepFEFM. NFM generates embeddings fed to a DNN via bi-interaction pooling layer, which are generated as a Hadamard product between feature embeddings. Product-based Neural Networks (PNN) [14], Deep Cross Network (DCN) [15] Deep Interest Network (DNN) [18], extreme deep factorization machine (xDeepFM) [21] use custom-designed core layers along with DNNs. They also combine their core layers with DNNs but their way of generating the intermediate layers is quite different from our method. Wide and Deep models [4], Attentional Factorization Machine (AFM), AutoInt [20], Feature Importance and Bilinear feature Interaction Network (FibiNet) [22] have their core architecture inspired from successful deep learning models from other domains. FPENN [19] estimates the probability distribution of the field-aware embedding as compared to FFM which is a single point estimation. However, for deep learning, they have only compared their method with DeepFM. FiBiNet shares similarity to our models as it also uses pairwise field matrices in their Bilinear-Interaction layer, but they use a combination of inner and outer product, which is different from our FEFM layer used for generating interaction embeddings.

## 3 Proposed Models

We propose a shallow model Field-Embedded Factorization Machines (FEFM) and its extension for deep learning, which we call Deep Field-Embedded Factorization Machine (DeepFEFM)

### 3.1 Field-Embedded Factorization Machines

For each field pair, Field-Embedded Factorization Machines (FEFM) introduces symmetric matrix embeddings along with the usual feature vector embeddings that are present in FM. Like FM, $v_i$ is the vector embedding of the $i^{th}$ feature.

However, unlike FFM, FEFM doesn't explicitly learn field-specific feature embeddings. The learnable symmetric matrix $W_{F(i),F(j)}$ is the embedding for the field pair $F(i)$ and $F(j)$. The interaction between the $i^{th}$ feature and the $j^{th}$ feature is mediated through $W_{F(i),F(j)}$. This modeling is shown in Equation 4.

$$\phi(\theta, x) = \phi_{FEFM}((w, v, W), x) = w_0 + \sum_{i=1}^{m} w_i x_i + \sum_{i=1}^{m} \sum_{j=i+1}^{m} v_i^T W_{F(i),F(j)} v_j x_i x_j \tag{4}$$

where $W_{F(i),F(j)}$ is a $k \times k$ symmetric matrix ($k$ is the dimension of the feature vector embedding space containing feature vectors $v_i$ and $v_j$).

The symmetric property of the learnable matrix $W_{F(i),F(j)}$ is ensured by reparameterizing $W_{F(i),F(j)}$ as $U_{F(i),F(j)} + U^T_{F(i),F(j)}$, where $U^T_{F(i),F(j)}$ is the transpose of the learnable matrix $U_{F(i),F(j)}$. In section 3.1.3, we discuss how the symmetric property of $W_{F(i),F(j)}$ helps it in modeling field pair interaction strengths. Note that $W_{F(i),F(j)}$ can also be interpreted as a vector transformation matrix which transforms a feature embedding when interacting with a specific field. This is advantageous over FFM, which instead explicitly learns field specific-feature embeddings, causing huge number of parameters and a tendency of overfitting. Since both the number of fields ($n$) and the embedding dimensions ($k$) are much smaller than number of features ($m$), FEFM uses less number of parameters while modeling similar field-specificity as in FFM. As can be seen in the Table 1, in practice the number of parameters are dominated by the term $mk$ and the term $\frac{n(n-1)}{2}k^2$ adds only a few additional parameters because both $n \ll m$ and $k \ll m$.

### 3.1.1 Model complexity

Table 1 shows a comparison of the number of parameters used by various models. It also describes why FwFM and FEFM have only slightly higher model complexity as FM, while FFM have a very high model complexity.

| Model | No. of parameters |
|---|---|
| LR | $m + 1$ |
| FMs | $m + mk + 1$ |
| FFMs | $m + m(n-1)k + 1$ |
| FwFMs | $m + mk + \frac{n(n-1)}{2} + 1$ |
| FEFMs | $m + mk + \frac{n(n-1)}{2}k^2 + 1$ |

Table 1: A comparison of the number of parameters used by various models. Here $m$ is the number of unique features, $n$ is the number of unique fields, and $k$ is the dimension of the embedding vectors used to represent features. Since in practice $k << m$ and $n << m$, $mk$ and $m(n-1)k$ terms dominate the number of parameters. FM, FwFM, and FEFM have roughly the same number of parameters for a given $k$ while FFM has a many folds more number of parameters

### 3.1.2 FEFM generalizes other shallow Factorization machines

In equation 4, when $W_{F(i),F(j)} = I$, where $I$ is the identity matrix, FEFM reduces to FM. When $W_{F(i)F(j)} = diag(r_{F(i),F(j)})$, where $diag(a)$ is the diagonal matrix with all entries in the diagonal being $a$ and $r_{F(i),F(j)}$ represents field strength between fields $F_i$ and $F_j$, FEFM reduces to FwFM. Thus FEFM is a powerful algorithm with the ability to generalize FM and FwFM. Since $W_{F(i),F(j)}$ transforms $v_j$ and then a dot product is done between the transformed vector and $v_i$, FEFM also has the ability to implicitly generate field-specific feature vector embeddings as explicitly done by FFM.

### 3.1.3 Field pair matrix embeddings and field interaction strength

In equation 4, $W_{F(i),F(j)} \in \mathbb{R}^{k \times k}$ is a symmetric matrix, where $k$ is the dimension of the feature vector embedding space. By the principal axis theorem, $W_{F(i),F(j)}$ has $k$ real non-increasing eigen values $\lambda_1, \lambda_2, ..., \lambda_k$ and the corresponding orthonormal eigenbasis $u_1, u_2, ..., u_k$ for the space $\mathbb{R}^k$ . For the equation 4 consider the feature interaction term $v_i^T W_{F(i),F(j)} v_j$ which models the interaction between the feature $i$ of the field $F(i)$ and the feature $j$ of the field $F(j)$. The feature embeddings $v_i$ and $v_j$ can be represented as linear combinations of the eigenbasis as $v_i = \sum_{t=1}^{k} b_t u_t$ and $v_j = \sum_{t=1}^{k} c_t u_t$. Thus the feature interaction term can be expressed as shown in the equation 5

$$v_i^T W_{F(i),F(j)} v_j = [\sum_{t=1}^{k} b_t u_t]^T W_{F(i),F(j)} [\sum_{t=1}^{k} c_t u_t] = \sum_{t=1}^{k} \lambda_t b_t c_t \tag{5}$$

where $b_t$ and $c_t$ are the coordinates of the feature emebddings $v_i$ and $v_j$ along the eigenbasis $u_t$. This suggests that the field pair embeddings matrices with eigen values of greater magnitude imply stronger field pair interactions. Thus we define the field pair interaction strength between fields $F(i)$ and $F(j)$ as given in Equation 6.

$$Strength_{F(i),F(j)} = \sqrt{\sum_{t=1}^{k} \lambda_t^2} \tag{6}$$

where $\lambda_t$ is the $t^{th}$ eigen value of the field pair embedding matrix $W_{F(i),F(j)}$. Further, the vector embeddings of strongly interacting features of a field pair tend to get aligned along the eigenbasis corresponding to the strongest eigen values of the field pair matrix embedding. In section 4.8, we study field pair strength properties on real world datasets.

### 3.2 Deep Field-Embedded Factorization Machines

In this section, we describe how we use FEFM to generate feature interaction embeddings which are adapted into a deep learning paradigm to yield Deep Field-Embedded Factorization Machine (DeepFEFM). The modeling of the logit function for DeepFEFM is shown in Equation 7.

$$\phi(\theta, x) = \phi_{DeepFEFM}((v, w, W, W^{deep}), x) = \phi_{FEFM}((w, v, W), x) + \phi_{DNN}((W^{deep}, w^{logit}), v_{tr}) \tag{7}$$

Here $\phi_{FEFM}((w, v, W), x)$ follows the same modeling as described in Equation 4 and $\phi_{DNN}((W^{deep}, w^{logit}), v_{tr})$ follows the usual modeling of a feed forward DNN used by other deep learning models for CTR prediction. Note the difference that for DeepFEFM in Equation 7 the input to the DNN is $v_{tr}$, a concatenation of the FEFM interaction embeddings generated by the FEFM interaction module and the input feature vector embeddings. The input layer to the DNN is defined as:

$$v_{tr} = concat(\{v_{fefm}, v_{active(x,1)}, v_{active(x,2)}, ..., v_{active(x,n)}\}) \tag{8}$$

where $active(x, F)$ is the feature that is active for the data point $x$ for the field $F$. The function $concat(...)$ concatenates all the vectors to create a single vector. And the FEFM interaction embedding $v_{fefm}$ generated by the FEFM interaction module is defined in Equation 9

$$v_{fefm} = concat(\{[v_i^T W_{F(i),F(j)} v_j x_i x_j] \mid (i, j) \in \{1, m\} \times \{1, m\} \wedge i < j\}) \tag{9}$$

Another way to think about the component FEFM embeddings of $v_{fefm}$ is in terms of the interaction of each feature with other features of different fields. The $i^{th}$ feature interacts with the features of the other $n - 1$ fields via the FEFM interaction layer by the operation $v_i^T W_{F(i),F(j)} v_j x_i x_j$. Thus for each feature an FEFM interaction vector embedding of length $n - 1$ is generated. All of the elements of these embeddings are concatenated with deduplication in Equation 9.

The overall architecture of DeepFEFM is shown in Figure 1. The ablation studies in Section 4 show that both FEFM interaction embeddings and FEFM logit terms are important for the success of the proposed architecture.

## 4 Experiments

In this section, we conduct extensive experiments for CTR prediction tasks to answer the following questions:

- **(Q1)** Over a wide range of hyperparameters, how does our proposed shallow model perform as compared to the state-of-the-art shallow models?
- **(Q2)** Over a wide range of hyperparameters, how does our proposed deep model perform as compared to the state-of-the-art deep models?

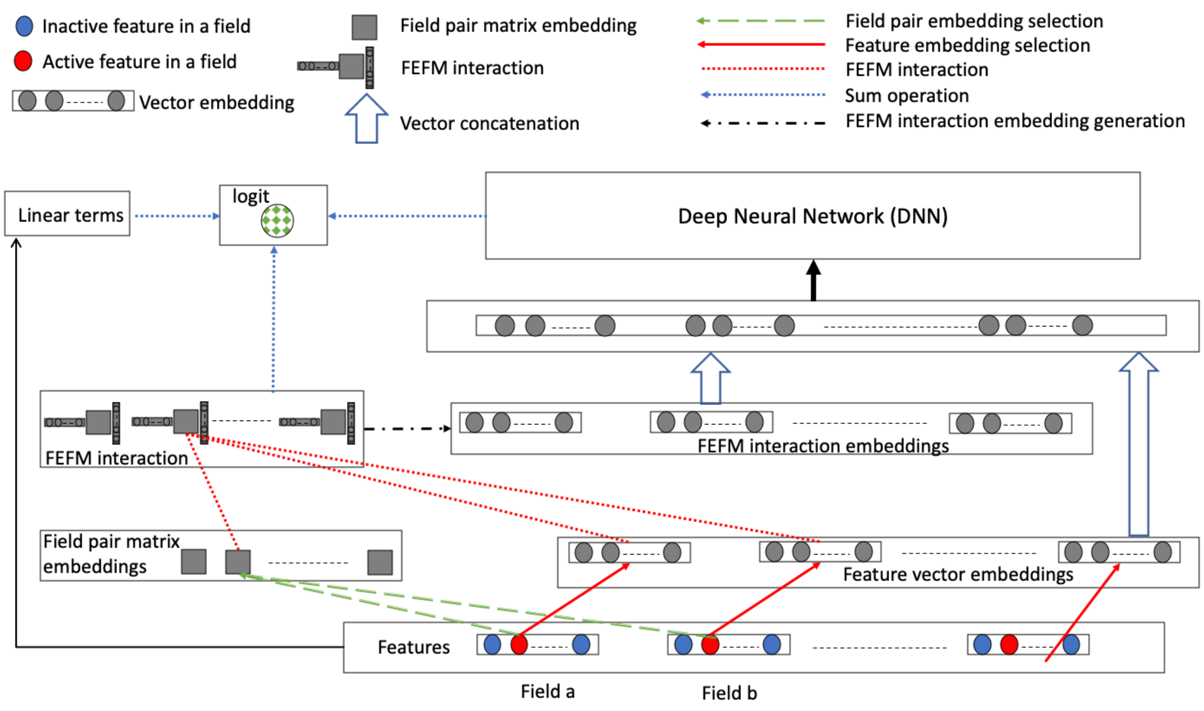

Inactive feature in a field
Active feature in a field
Vector embedding
Field pair matrix embedding
FEFM interaction
Vector concatenation
Field pair embedding selection
Feature embedding selection
FEFM interaction
Sum operation
FEFM interaction embedding generation
Linear terms
logit
Deep Neural Network (DNN)
FEFM interaction
FEFM interaction embeddings
Field pair matrix embeddings
Feature vector embeddings
Features
Field a
Field b

Figure 1: Architecture of Deep Field-Embedded Factorization Machine

- **(Q3)** What is the effect of feature vector embedding dimension on the performance of our proposed models vis-a-vis other state-of-the-art models?
- **(Q4)** How does the validation loss changes with the epochs for our models as compared to the state-of-the-art models?
- **(Q5)**: Which fields and field interactions are important?
- **(Q6)** Ablation studies: How important are various components and design choices of our proposed models?

Before addressing these questions we present fundamental details of experimental setup.

### 4.1 Datasets

We use two publicly available datasets called Avazu [25] and Criteo [26] to compare our proposed models with the existing models. After shuffling, both the datasets are further divided into train, validation, and test sets. Avazu dataset has only categorical features while Criteo has both categorical and numerical features. The numerical features in Criteo dataset are discretized at integer levels and thus treated as categorical features. Discretization of numerical features in Factorization Machines is known to yield better performance [7]. For both Avazu and Criteo datasets, features with frequency of occurrence less than 20 are filtered out as a noise removal step. For Avazu, the id field is removed and the hour field is transformed to the 0-23 range. For categorical features, if a feature is missing in the test and validation set, then a special feature category of unknown feature is created for all the fields. All the transformations are fit on the train set and applied to the train, test and validation sets. For discretization of a numerical feature, if a feature is missing in the test and the validation set, its nearest neighbor from the train set is used as the feature. The processed dataset properties are shown in Table 2.

| **Dataset** | $n$ | $m$ | **train : validation : test** |
|---|---|---|---|
| Avazu | 22 | 254,644 | 25,874,537 : 6,468,635 : 8,085,794 |
| Criteo | 39 | 1,040,123 | 29,337,994 : 7,334,499 : 9,168,124 |

Table 2: The number of fields ($n$), features ($m$) and the size of the train, validation, and test tests for both the datasets after preprocessing.

| Model | Hyperparameters | AUC | Log loss | # params (Mil) |
|---|---|---|---|---|
| LR | $\eta = 0.05, \lambda_1 = 1e-7$ | 0.79416 | 0.45600 | 1.0 |
| FM | $\eta = 0.05, \lambda_1 = 1e-6, \lambda_2 = 1e-5, , k = 48$ | 0.80889 | 0.44269 | 51.0 |
| FFM | $\eta = 0.01, \lambda_1 = 1e-7, \lambda_2 = 1e-7, k = 32$ | 0.81018 | 0.44171 | 1265.8 |
| FwFM | $\eta = 0.05, \lambda_1 = 1e-6, \lambda_2 = 1e-5, \lambda_3 = 1e-8, k = 32$ | 0.81024 | 0.44148 | 34.3 |
| FEFM (proposed) | $\eta = 0.05, \lambda_1 = 1e-6, \lambda_2 = 1e-5, \lambda_3 = 1e-7, k = 16$ | **0.81122** | **0.44053** | 17.9 |

(a) Criteo

| Model | Hyperparameters | AUC | Log loss | # params (Mil) |
|---|---|---|---|---|
| LR | $\eta = 0.05, \lambda_1 = 1e-8$ | 0.76056 | 0.38993 | 0.2 |
| FM | $\eta = 0.05, \lambda_1 = 1e-6, \lambda_2 = 1e-6, k = 96$ | 0.77474 | 0.38249 | 24.7 |
| FFM | $\eta = 0.6, \lambda_1 = 1e-5, \lambda_2 = 1e-5, k = 16$ | 0.77614 | 0.38214 | 85.8 |
| FwFM | $\eta = 0.1, \lambda_1 = 1e-5, \lambda_2 = 1e-7, \lambda_3 = 1e-8, k = 80$ | 0.77680 | 0.38139 | 20.6 |
| FEFM (proposed) | $\eta = 0.02, \lambda_1 = 1e-6, \lambda_2 = 1e-6, \lambda_3 = 1e-8, k = 96$ | **0.77704** | **0.38107** | 26.8 |

(b) Avazu

Table 3: A comparison of the prediction performance of FEFM with the existing state-of-the-art shallow models on the test sets. The best hyperparameters and the total number of parameters for each of the model types are also mentioned. $\lambda_1$, $\lambda_2$, and $\lambda_3$ are the L2 regularization strengths used for the linear parameters, feature embeddings, and field-pair embeddings; $\eta$ is the learning rate, and $k$ is the size of the feature embeddings.

## 4.2 Implementation and Experimental Setup

For FFM we use the LIBFFM implementation provided by the authors of FFM [7]. All the other models are implemented using Keras [27], TensorFlow [28], and DeepCTR [29]. For optimization, we use the AdaGrad optimizer [30]. A batch size of 1024 was used for training. For embeddings at dimensions higher than 32 for Criteo dataset, if the GPU memory is exceeded, a batch size of 512 is used. We discovered that our proposed methods and existing methods show better performance when different L2 regularization is used for the linear parameters, the feature embeddings, and the field-pair embeddings. Thus in our implementation, wherever valid, we provide support for different L2 regularization for these three parameters. All training and inference was done on Tesla K80 GPUs.

## 4.3 Hyperparameter tuning

To ensure fairness, for all the type of models a wide range of hyperparameters are tried to arrive at their corresponding best hyperparameters. For each type of model and embedding dimension, the hyperparameters yielding the lowest log loss on the validation set are chosen as the best hyperparameters. The corresponding best model is then evaluated on the test set and the performance is reported. Early stopping (minimum delta of 0.000005 and patience of 2) based on the log loss on a validation set was also used during the training of the models [27].

## 4.4 Comparison with state-of-the-art Shallow Models (Q1)

For the existing shallow models and the proposed shallow model FEFM, the performance of the best model instances are compared in Table 3. For both the datasets, FEFM achieves better performance than the existing state-of-the-art shallow models for both log loss and AUC metrics. For Criteo, FEFM needs the least number of parameters while for Avazu the number of parameters are slightly higher than those of FM and FwFM.

## 4.5 Comparison with state-of-the-art Deep Models (Q2)

For the existing deep models and the proposed deep model DeepFEFM, the performance of the best model instances are compared in Table 4. For both the datasets, DeepFEFM achieves better performance than the existing state-of-the-art shallow models for both log loss and AUC metrics. FiBiNet is the second in performance and it needs significantly higher number of parameters compared to our proposed DeepFEFM. Another interesting observation is that for Criteo, all the deep models outperform all the shallow models but for Avazu only FiBiNet and DeepFEFM outperform all the shallow models. Interestingly, for Avazu, shallow models FwFM and FEFM (Table 3) are able to outperform many

| Model | Hyperparameters | AUC | Log loss | # params (Mil) |
|---|---|---|---|---|
| DeepFM | $\eta = 0.01, \lambda_1 = 1e-6, \lambda_2 = 1e-5, k = 64$ $\lambda_{deep} = 1e-7, L = 3, dropout = 0.2$ | 0.81235 | 0.43933 | 72.3 |
| xDeepFM | $\eta = 0.01, \lambda_1 = 1e-7, \lambda_2 = 1e-5, \lambda_c = 1e-6, k = 16$ $\lambda_{deep} = 1e-7, L = 3, dropout = 0.2,$ | 0.81214 | 0.43953 | 22.4 |
| AutoInt+ | $\eta = 0.01, \lambda_2 = 1e-5, k = 32, \lambda_{deep} = 0.0$ $L = 2, dropout = 0.0, H = 2, k_a = 64, L_a = 1$ | 0.81216 | 0.43963 | 35.7 |
| FiBiNet | $\eta = 0.01, \lambda_1 = 1e-6, \lambda_2 = 1e-6, k = 64$ $\lambda_{deep} = 0.0, L = 3, dropout = 0.5, r = 3$ | 0.81302 | 0.43871 | 173.1 |
| DeepFEFM (proposed) | $\eta = 0.01, \lambda_1 = 1e-6, \lambda_2 = 1e-5, \lambda_3 = 1e-7, k = 48$ $\lambda_{deep} = 1e-7, L = 3, dropout = 0.2$ | **0.81405** | **0.43778** | 57.6 |

(a) Criteo

| Model | Hyperparameters | AUC | Log loss | # params (Mil) |
|---|---|---|---|---|
| DeepFM | $\eta = 0.01, \lambda_1 = 1e-6, \lambda_2 = 1e-6, k = 64$ $\lambda_{deep} = 1e-7, L = 3, dropout = 0.2$ | 0.77594 | 0.38148 | 20.1 |
| xDeepFM | $\eta = 0.01, \lambda_1 = 1e-6, \lambda_2 = 1e-5, \lambda_c = 1e-5, k = 32$ $\lambda_{deep} = 1e-7, L = 3, dropout = 0.2,$ | 0.77550 | 0.3816 | 12.8 |
| AutoInt+ | $\eta = 0.01, \lambda_2 = 1e-5, k = 80, \lambda_{deep} = 1e-7$ $L = 3, dropout = 0.0, H = 2, k_a = 80, L_a = 1$ | 0.77627 | 0.38153 | 24.4 |
| FiBiNet | $\eta = 0.01, \lambda_1 = 1e-7, \lambda_2 = 1e-6, k = 96$ $\lambda_{deep} = 0.0, L = 3, dropout = 0.5, r = 3$ | 0.77706 | 0.38083 | 76.6 |
| DeepFEFM (proposed) | $\eta = 0.01, \lambda_1 = 1e-7, \lambda_2 = 1e-6, \lambda_3 = 1e-8, k = 96$ $\lambda_{deep} = 1e-7, L = 3, dropout = 0.2$ | **0.77801** | **0.38052** | 31.4 |

(b) Avazu

Table 4: A comparison of the prediction performance of DeepFEFM with the existing state-of-the-art deep models on the test sets. The best hyperparameters and the total number of parameters for each of the model types are also mentioned. $\lambda_1$, $\lambda_2$, and $\lambda_3$ are the L2 regularization strengths used for the linear parameters, feature embeddings, and field-pair embeddings; $\eta$ is the learning rate and $k$ is the size of the feature embeddings; $\lambda_{deep}$ is the L2 regularization strength of the DNN component, $L$ for the number of DNN layers (excluding input layer and logit layer), the width of all the DNN layers is 1024 for all the models; $\lambda_c$ is the L2 regularization strength for the CIN module of xDeepFM; $H$ is the number of self-attention heads, $k_a$ is the attention embedding dimension, $L_a$ is the number of attention layers in AutoInt+; $r$ is the reduction ratio in FiBiNet

other existing deep learning models. This also suggests that deep learning still has a long way to reach the performance level it has reached in images and text models, where deep learning completely outshine shallow learning.

## 4.6 Effect of Embedding Dimension on Performance (Q3)

Most of the existing deep learning work on CTR models report comparison at a single embedding dimension and don't the compare performance of their models with existing models for different embedding dimensions. However, earlier work on shallow models such as FFM [7] does perform a thorough comparison across embedding dimensions. We found that different models behave differently at different dimensions and thus we compare the performance of both shallow (Figure 2 and 3) and deep models (Figure 4 and 5). It is interesting to observe that the behavior changes not only across different model types but also across the datasets. For Criteo, FEFM outperforms all the other shallow models at all dimensions. For Avazu, at lower dimensions FFM outperforms other models but is soon overtaken by both FwFM and FEFM. At the extremities, FEFM outperforms FwFM while for the moderate dimensions, their performance is competitive. For deep learning models on Criteo, DeepFEFM outperforms the other deep models for all dimensions. At lower dimensions, xDeepFM and AutoInt+ are second to DeepFEFM, but xDeepFM and AutoInt+ tend to overfit at higher dimensions. For Avazu, the difference in performance is narrow at lower dimensions and it widens at higher dimensions. At very low dimensions AutoInt+ performs better but overfit at moderate and higher dimensions. FiBiNet performs the best at moderate dimensions but not at the extremities. At lower dimensions, FibiNet is the weakest of all in performance. Also xDeepFM and AutoInt+ encounter a quick drop in performance at higher dimensions while FiBiNet and DeepFEFM continue to perform better at higher dimensions. While for the moderate dimensions, FiBiNet is better, at the extremities, DeepFEFM outperforms FiBiNet.

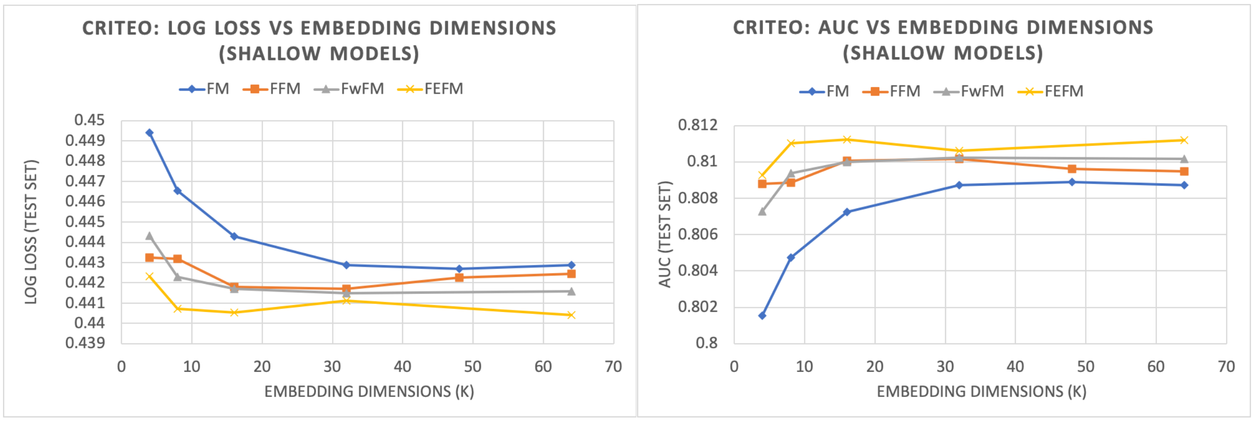


Figure 2: Criteo test set: Log loss and AUC comparison with state-of-the-art shallow models

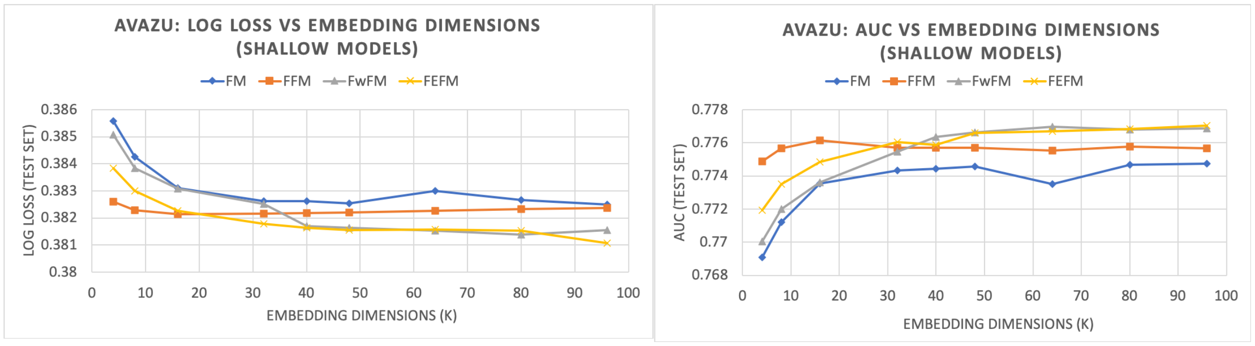


Figure 3: Avazu test set: Log loss and AUC comparison with state-of-the-art shallow models

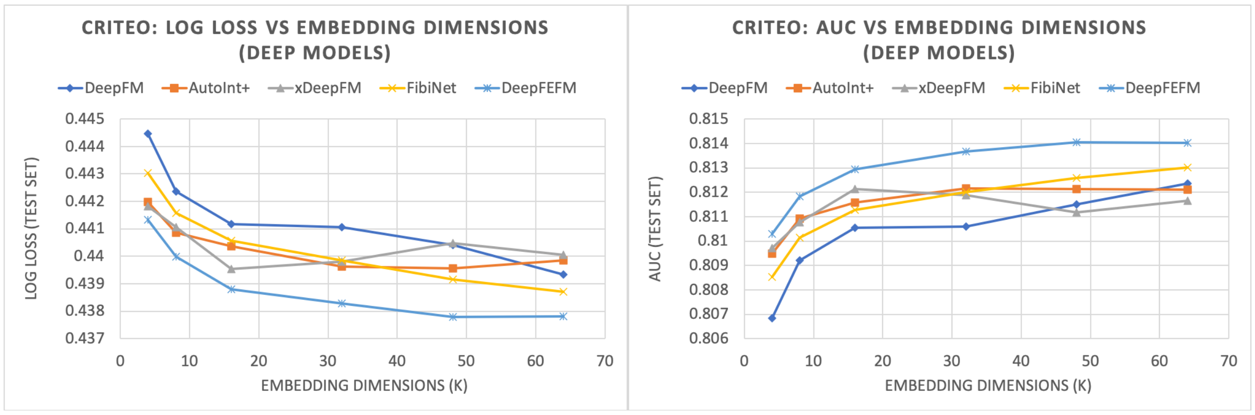


Figure 4: Criteo test set: Log loss and AUC comparison with state-of-the-art deep models

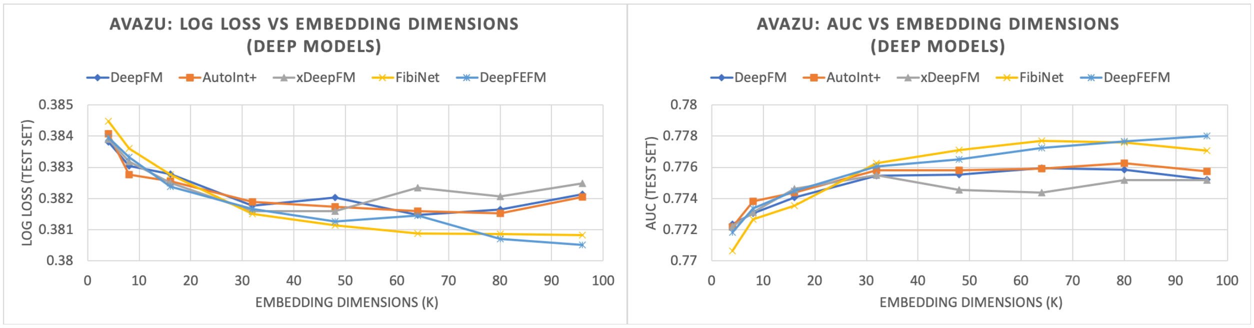


Figure 5: Avazu test set: Log loss and AUC comparison with state-of-the-art deep models

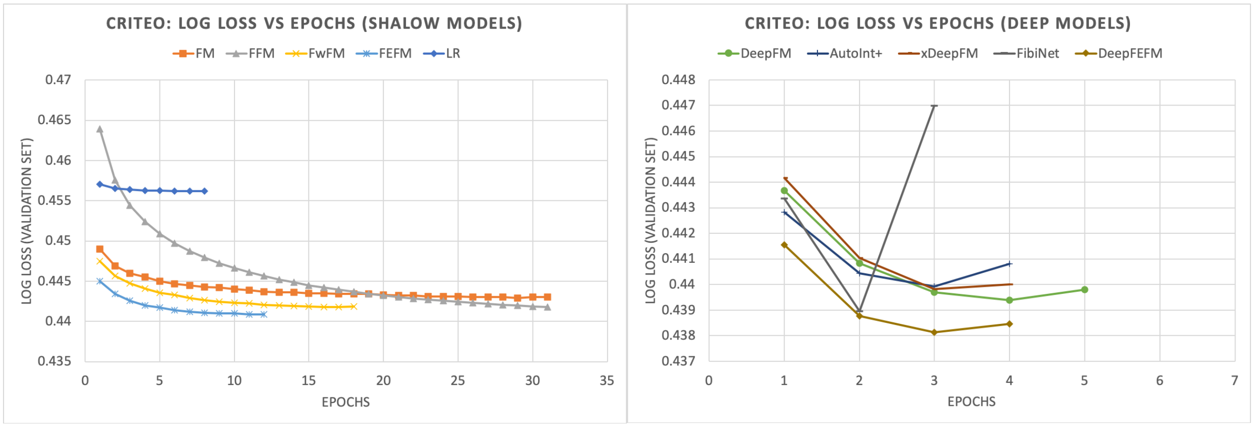


Figure 6: Criteo validation set: Log loss for the shallow and deep models at different epochs.

## 4.7 Convergence of training (Q4)

Figure 6 and 7 show how the log loss on the validation set changes with each epoch for the best models for each type for both the datasets. We can see that overall for all the models, convergence is faster for Avazu as compared to Criteo. For shallow FM-variants, FEFM and FwFM converge faster than FFM and FM. Deep models in general converge quickly, with FiBiNet being the fastest to converge, which is expected because FiBiNet uses very high number of parameters compared to the other models.

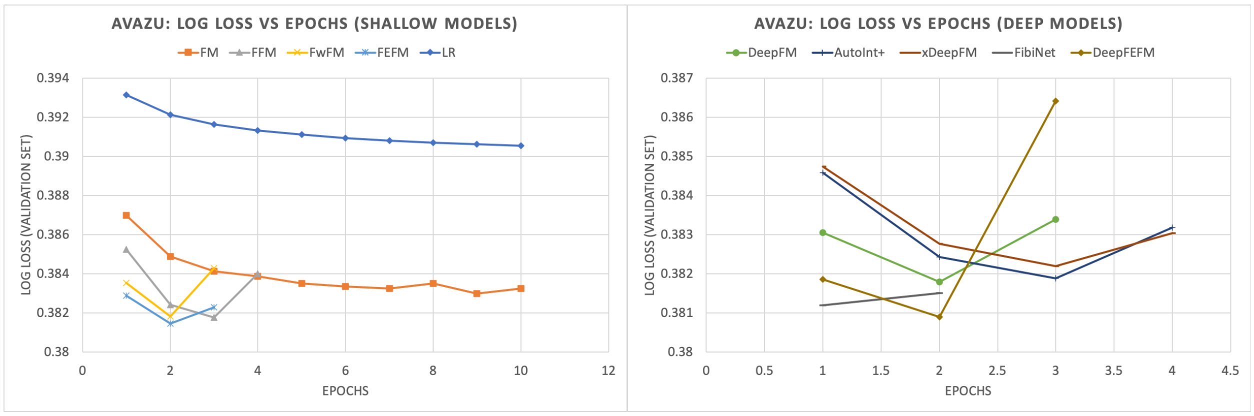
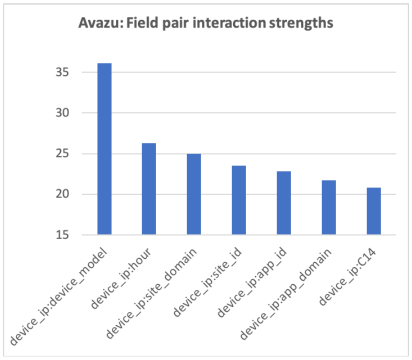


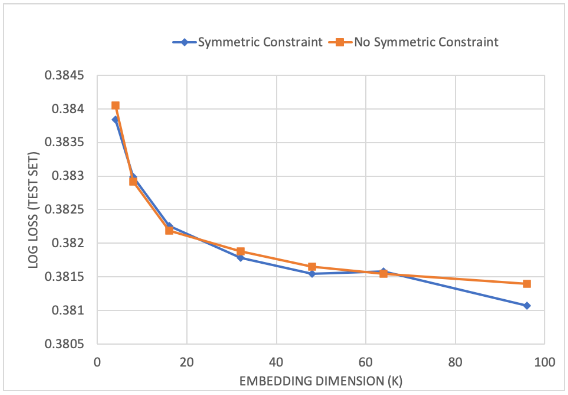
Figure 7: Avazu validation set: Log loss for the shallow and deep models at different epochs.

### 4.8 Study of field interaction strength (Q5)

In Section 3.1.3, we discussed how FEFM provides insights into the interaction strengths of the field pairs via eigen values of the field pair embedding matrices. The bar charts in Figure 8a show the top 7 field pairs arranged in decreasing order of their interaction strengths as per Equation 6. For Avazu dataset, it can be found that the top-most field pairs include interaction between “device_ip" and other fields such as “device_model", “hour", “site_domain", etc. This also makes intuitive sense since “device_ip" is a very important field for CTR prediction. Since the field names are completely anonymized for Criteo dataset, we have not presented the plots. We found that the interaction between the anonymized field “C24" and multiple other fields is very important.



(a) Avazu: Top 7 field pairs reverse sorted by their interaction strengths



(b) Log loss on Avazu test dataset for symmetric vs non-symmetric matrix embeddings in FEFM

Figure 8: Avazu: Field pair strengths and the effect of symmetric constraint on matrix embeddings

| Model | Criteo AUC | Criteo LogLoss | Avazu AUC | Avazu LogLoss |
|---|---|---|---|---|
| No Ablation | 0.81405 | 0.43778 | 0.77801 | 0.38052 |
| Ablation1 | 0.81381 | 0.43795 | 0.77689 | 0.38113 |
| Ablation2 | 0.81386 | 0.43793 | 0.77768 | 0.38078 |
| Ablation3 | 0.81363 | 0.43832 | 0.77814 | 0.3806 |
| Ablation4 | 0.81259 | 0.43919 | 0.77656 | 0.38121 |

Table 5: Comparison of the prediction performance of DeepFEFM with ablation of its various components. **Ablation1**: removed the FEFM logit terms from the final logit, **Ablation2**: removed the linear terms from the final logit, **Ablation3**: removed the feature vector embeddings from the DNN input, **Ablation4**: removed the FEFM interaction embeddings from the DNN input.

### 4.9 Ablation Studies (Q6)

We ablate different components of our models to answer the following specific questions:

- **What is the effect of the symmetric constraint on field pair matrix embeddings?** In Section 3.1.3 and Section 4.8 we discussed the rationale and field interaction insights on public datasets in regard to the symmetric property of the field pair matrix embeddings. In Figure 8b we compare the performance of FEFM with and without the symmetric constraint on the field pair matrix embeddings. We see that at lower embedding dimensions, the performance is similar for both the cases. At higher dimensions, the symmetric constraint gives slightly better performance, suggesting a mild regularizing effect when the number of parameters are high. In either case, it is desirable to have the symmetric constraint for its mathematical properties that explain the important fields and their interactions.
- **Which components of the DeepFEFM architecture are the most important?** We compare the performance of different ablated architectures of DeepFEFM in Table 5. Removal of FEFM interaction embeddings from the proposed DeepFEFM architecture leads to the biggest drop in the performance for both the datasets. Removal of other layers have a minor effect on Criteo but for Avazu, removal of FEFM logit terms leads to the second biggest drop in performance. This leads us to conclude that FEFM interaction embeddings and FEFM logit

terms are the two most important components of our architectures. These are also our novel contributions to the CTR prediction models.

## 5 Conclusion

In this paper, we first proposed a shallow model called Field-Embedded Factorization Machine (FEFM) for click-through rate prediction. FEFM is a novel variant of Factorization Machines (FM) that leverages field-specificity in feature interaction by introducing learnable symmetric matrix embeddings for each field pair. The eigen values of the field pair embeddings represent interaction strengths between the fields. We then extended the shallow model to a deep learning model DeepFEFM using the FEFM interaction embeddings. By conducting extensive experiments over a wide range of hyperparameters, we showed that both our proposed models consistently outperform existing state-of-the-art models in their respective shallow and deep categories. By conducting ablation studies, we also showed that FEFM logit terms and FEFM interaction embeddings are critical to the boost in prediction performance.

## Acknowledgment

We would like to thank Abhishek Veeraraghavan, Kunal Kumar Jain, Priyanka Patel, Tathagata Sengupta, and Umang Moorarka for their valuable feedback for the successful completion of this work.

## References

[1] PricewaterhouseCoopers. Iab internet advertising revenue report. `https://www.iab.com/wp-content/uploads/2019/05/Full-Year-2018-IAB-Internet-Advertising-Revenue-Report.pdf`, 2019.

[2] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230, 2013.

[3] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 1725–1731, 2017.

[4] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[5] Matthew Richardson, Ewa Dominowska, and Robert Ragno. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*, pages 521–530. ACM, 2007.

[6] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.

[7] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 43–50. ACM, 2016.

[8] Junwei Pan, Jian Xu, Alfonso Lobos Ruiz, Wenliang Zhao, Shengjun Pan, Yu Sun, and Quan Lu. Field-weighted factorization machines for click-through rate prediction in display advertising. In *Proceedings of the 2018 World Wide Web Conference*, pages 1349–1357. International World Wide Web Conferences Steering Committee, 2018.

[9] Thore Graepel, Joaquin Quinonero Candela, Thomas Borchert, and Ralf Herbrich. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. Omnipress, 2010.

[10] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[11] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pages 1–9, 2014.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[13] Weinan Zhang, Tianming Du, and Jun Wang. Deep learning over multi-field categorical data. In *European conference on information retrieval*, pages 45–57. Springer, 2016.

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

[15] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, pages 1–7. 2017.

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

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

[18] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1059–1068, 2018.

[19] Weiwen Liu, Ruiming Tang, Jiajin Li, Jinkai Yu, Huifeng Guo, Xiuqiang He, and Shengyu Zhang. Field-aware probabilistic embedding neural network for ctr prediction. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 412–416, 2018.

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

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

[22] Tongwen Huang, Zhiqi Zhang, and Junlin Zhang. Fibinet: combining feature importance and bilinear feature interaction for click-through rate prediction. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 169–177, 2019.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[25] Avazu Inc. Click-through rate prediction challenge. `https://www.kaggle.com/c/avazu-ctr-prediction`, 2015.

[26] Criteo Labs. Display advertising challenge. `https://www.kaggle.com/c/criteo-display-ad-challenge`, 2014.

[27] François Chollet et al. Keras. `https://keras.io`, 2015.

[28] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[29] Weichen Shen et al. Deepctr. `https://github.com/shenweichen/DeepCTR`, 2017.

[30] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.