An Input-aware Factorization Machine for Sparse Prediction

Yantao Yu , **Zhen Wang** , **Bo Yuan***
Graduate School at Shenzhen, Tsinghua University
{yyt17, z-wang16}@mails.tsinghua.edu.cn, yuanb@sz.tsinghua.edu.cn

Abstract

Factorization machines (FMs) are a class of general predictors working effectively with sparse data, which represent features using factorized parameters and weights. However, the accuracy of FMs can be adversely affected by the fixed representation trained for each feature, as the same feature is usually not equally predictive and useful in different instances. In fact, the inaccurate representation of features may even introduce noise and degrade the overall performance. In this work, we improve FMs by explicitly considering the impact of each individual input upon the representation of features. We propose a novel model named *Input-aware Factorization Machine* (IFM), which learns a unique input-aware factor for the same feature in different instances via a neural network. Comprehensive experiments on three real-world recommendation datasets are used to demonstrate the effectiveness and mechanism of IFM. Empirical results indicate that IFM is significantly better than the standard FM model and consistently outperforms four state-of-the-art deep learning based methods.

1 Introduction

Prediction now plays a crucial role in many personalized systems, such as online advertising [McMahan *et al.*, 2013; Juan *et al.*, 2016] and recommendation [Koren *et al.*, 2009; Cheng *et al.*, 2014]. Typically, the recommendation task is formulated as estimating a function that maps categorical predictor variables ($a.k.a.$ features) to a target. For example, we need to predict *the click probability* (target) that a *user* (first predictor variables) of a particular occupation will click on an *item* (second predictor variables). The first and second predictor variables are usually combined in the form of an instance, e.g., {*young, female, student, pink, skirt*}.

To build predictive models with these categorical predictor variables, it is indispensable to accurately represent them in machine identifiable forms. A common solution is to convert them to a set of binary features ($a.k.a.$ feature vector) via one-hot encoding [Cheng *et al.*, 2016]. Depending on the number of possible values of categorical predictor variables, the generated feature vector can be very high dimensional and sparse. To build an effective model with such sparse data, factorization machines (FMs) were proposed [Rendle, 2010], which learn a one-dimensional weight and a k-dimensional embedding vector as the representation of each feature from sparse data. Owing to its efficient linear training time and high prediction accuracy, FMs have been successfully applied to various applications, from recommendation systems [Rendle *et al.*, 2011] to natural language processing [Petroni *et al.*, 2015]. Despite great promise, FMs produce a single representation for each feature and the same representation of a given feature is shared in different instances to compute its predictive power, which may lead to inferior performance. In this paper, we argue that the impact of each individual input should be given full consideration when creating the representation for each feature.

Many existing studies attempt to improve the prediction accuracy of FMs by focusing on feature interactions. For example, DeepFM [Guo *et al.*, 2017] models high-order feature interactions through a neural network, while AFM [Xiao *et al.*, 2017] enhances FMs by learning the importance of each feature interaction from data via a neural attention network. Nevertheless, these improvements are limited as the uniqueness of each instance is not exploited.

In our work, we propose to improve FMs from a new perspective that tries to refine the representation of features according to different instances. In real-world applications, a feature usually has dissimilar levels of predictive power in different situations. For example, the feature $female$ is apparently crucial for *click probability* in an instance: {*young, female, student, pink, skirt*}. However, in another instance: {*young, female, student, blue, notebook*}, the feature $female$ is relatively less crucial. As such, the same feature on different instances should be assigned different levels of predictive power to better reflect its specific contribution.

In this paper, we present a novel model for prediction tasks under sparsity named *Input-aware Factorization Machine* (IFM), which enhances FMs by explicitly considering the impact of each individual input on the representation of features. It refines the weight and embedding vector of each feature with regard to different instances and adds nonlinearity to the model simultaneously. Specifically, we adopt the idea of end to end memory network [Sukhbaatar *et al.*, 2015]

*Corresponding author

to enable each feature to contribute dissimilarly in different instances. In this way, the representation of each feature is not only related to itself, but also to the instances containing it. In contrast to other deep learning based methods that mainly focus on feature interactions, our use of neural networks for more informative representations of features greatly enhances the expressiveness and interpretability of FMs. Comprehensive experiments show that our IFM features two major advantages: i). it produces better prediction results compared to existing techniques; ii). it provides deeper insights into the role that each feature plays in the prediction task.

2 Preliminaries

Factorization machines are proposed to learn feature interactions for sparse data, which combine the advantages of Support Vector Machines (SVMs) with factorization models. FMs enhance linear regression (LR) using the second-order factorized interactions between features. Given a real valued feature vector $\mathbf{x} \in \mathbb{R}^n$ where n denotes the number of features and most of the elements x_i in a vector $\mathbf{x}$ are zero, the FM model estimates the target by modelling the interactions via factorized interaction parameters:

$$\hat{y}_{FM}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \quad (1)$$

where $\mathbf{x}$ is the feature vector of the instance and x_i denotes the i-th dimension of the feature vector while $\hat{y}$ represents the value predicted by the FM model (e.g., the estimated probability of click). w_0 is the global bias and w_i models the weight of the i-th variable. $\mathbf{v}_i$ is a k-dimensional embedding vector of the i-th variable and $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ is the dot product of two vectors of size k, which models the interaction between the i-th and j-th variables. Here, k is the dimension of embedding vectors, which controls the complexity of FMs. Note that the feature interactions can be reformulated [Rendle, 2010] as:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j = \frac{1}{2} \sum_{f=1}^{k} \left[\left(\sum_{j=1}^{n} v_{j,f} x_j \right)^2 - \sum_{j=1}^{n} v_{j,f}^2 x_j^2 \right] \quad (2)$$

where $v_{j,f}$ denotes the f-th element in $\mathbf{v}_j$. The time complexity of Equation 1 is $O(kn^2)$, but with reformulating it drops to linear time complexity $O(kn)$.

It is worth noticing that FMs produce a fixed representation for each feature: the same weight w_i and embedding vector $\mathbf{v}_i$ are shared across all different instances that involve the i-th feature. However, it is not unusual that a certain feature is not equally predictive and useful across different instances. Consequently, due to the lack of flexibility in feature representation, FMs may suffer from the insufficient ability for modelling complex data, which may adversely affect their performance in general.

3 Input-aware Factorization Machine

In this section, we present the details of the proposed IFM model to show how to boost the performance of FMs via the input-aware strategy.

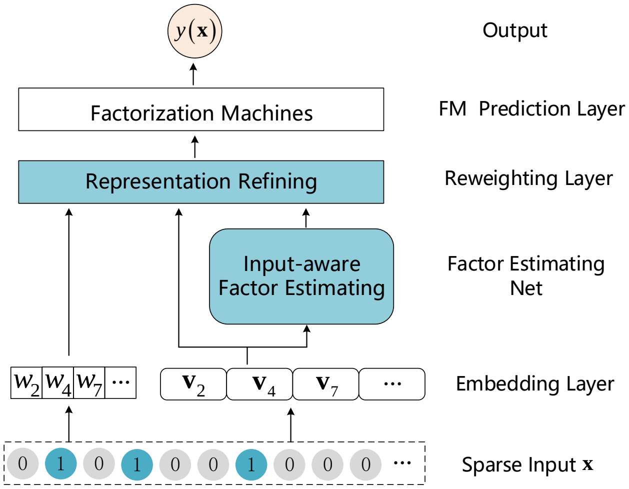


Figure 1: The network architecture of our proposed Input-aware Factorization Machines model

3.1 The IFM Model

Similar to factorization machines, given a sparse feature vector $\mathbf{x} \in \mathbb{R}^n$ as input, where a feature value $x_i = 0$ means that the i-th feature does not exist in the instance, IFM predicts the target as:

$$\hat{y}_{IFM}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_{\mathbf{x},i} x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_{\mathbf{x},i}, \mathbf{v}_{\mathbf{x},j} \rangle x_i x_j \quad (3)$$

where the first and second terms model the global bias of data and weights of features respectively, and the third term captures the feature interaction. Note that, in our method, the second and third terms model each feature's effect on the label in a nonlinear way, as the weight $w_{\mathbf{x},i}$ and embedding vector $\mathbf{v}_{\mathbf{x},i}$ of each feature not only correlate with the i-th feature but also are related to the input vector $\mathbf{x}$. Figure 1 illustrates the network architecture of our proposed IFM model, where the extra components in addition to traditional FMs are marked by dark background.

Embedding Layer

As in FMs, a weight and an embedding vector are randomly initialized for each feature as its representation. Formally, let $\mathbf{v}_i \in \mathbb{R}^k$ be the embedding vector of the i-th feature, where k is the embedding size. Due to the sparsity of $\mathbf{x}$, we only need to include the embedding vectors of non-zero features, *i.e.*, $\mathcal{V}_{\mathbf{x}} = \{\mathbf{v}_i\}$ where $x_i \neq 0$. Finally, we stack all vectors in $\mathcal{V}_{\mathbf{x}}$ into a single $k \times h$ dimensional vector: $\mathbf{V}_{\mathbf{x}} = \left[\mathbf{v}_1^T, \mathbf{v}_2^T ...., \mathbf{v}_h^T\right]$, where h denotes the number[1] of non-zero elements per instance.

Factor Estimating Network

The input-aware strategy is inspired by memory networks [Weston *et al.*, 2014], which are a class of deep learning based models mainly used in question answering (QA) tasks. One of their key ideas is to allow different memory records to con-

[1] In this paper, we assume that h is a fixed value for each dataset.

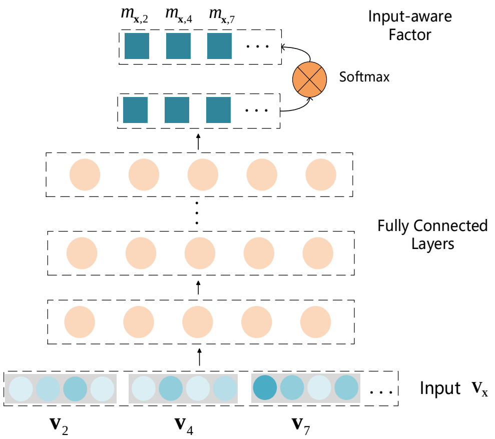


Figure 2: The neural network architecture of Factor Estimating Network

tribute differently to each question. Motivated by this inherent flexibility of memory networks, the input-aware strategy is proposed to extend the functionality of FMs by introducing an input-aware factor $m_{\mathbf{x},i}$ into the feature weight w_i and embedding vector $\mathbf{v}_i$, which allows each feature to contribute dissimilarly in different inputs.

To estimate the input-aware factor $m_{\mathbf{x},i}$, we first feed the stacked vector $\mathbf{V_x}$ into the *Factor Estimating Network* (FEN) to learn a unique vector $\mathbf{U_x}$ for the given input $\mathbf{x}$. Figure 2 shows the architecture of FEN. The network begins with a stack of fully connected layers that are capable of learning unique information from $\mathbf{x}$. Formally, the definition of the fully connected layers is as follows:

$$\begin{aligned} \mathbf{a}_1 &= \sigma_1\left(\mathbf{W}_1\mathbf{V_x} + \mathbf{b}_1\right), \\ &\ldots \\ \mathbf{U_x} = \mathbf{a}_L &= \sigma_{\mathrm{L}}\left(\mathbf{W}_{\mathrm{L}}\mathbf{a}_{L-1} + \mathbf{b}_L\right) \end{aligned} \tag{4}$$

where $\mathbf{W}_l$, $\mathbf{b}_l$, σ_l and $\mathbf{a}_l$ are the weight matrix, bias vector, activation function and the output of the l-th layer, respectively. Next, $\mathbf{U_x}$, which is the output vector of the last hidden layer $\mathbf{a}_L$, is used to learn an input-aware factor $m_{\mathbf{x},i}$ for each feature in the input $\mathbf{x}$:

$$\begin{aligned} &\mathbf{m'_x} = \mathbf{U_x}\mathbf{P}, \mathbf{P} \in \mathbb{R}^{t\times h}, \\ &m_{\mathbf{x},i} = h \times \frac{\exp(m'_{\mathbf{x},d})}{\sum_{j=1}^{h}\exp(m'_{\mathbf{x},j})}, x_i \neq 0 \end{aligned} \tag{5}$$

where $\mathbf{P}$ denotes the weight matrix that transforms $\mathbf{U_x}$ to a h-dimensional vector $\mathbf{m'_x}$, and t denotes the number of neurons in the last hidden layer. Finally, the elements $m'_{\mathbf{x},d}, d \in [1, h]$ in $\mathbf{m'_x}$ are normalized through a softmax function, so that the sum of all elements is equal to h, representing the input-aware factor $m_{\mathbf{x},i}$ of each non-zero feature in $\mathbf{x}$.

Reweighting Layer

Once the outputs from FEN are obtained, they are used to refine the feature weight w_i and embedding vector $\mathbf{v}_i$ with regard to the current input. The inputs of this layer are the w_i and $\mathbf{v}_i$ of the given input $\mathbf{x}$ and the input-aware factor $m_{\mathbf{x},i}$ from the previous layer. Formally, the definition of the reweighting layer is as follows:

$$\begin{aligned} w_{\mathbf{x},i} &= m_{\mathbf{x},i}w_i \\ \mathbf{v}_{\mathbf{x},i} &= m_{\mathbf{x},i}\mathbf{v}_i \end{aligned} \tag{6}$$

where $w_{\mathbf{x},i}$, $\mathbf{v}_{\mathbf{x},i}$ are the refined representations of features for the specific input $\mathbf{x}$.

FM Prediction Layer

In this layer, the input-aware weight $w_{\mathbf{x},i}$ and embedding vector $\mathbf{v}_{\mathbf{x},i}$ are fed into factorization machines to predict the target as Equation 3. It is worth mentioning that, similar to Equation 2, we can reformulate Equation 3 to reduce the runtime of this layer to linear time complexity $O(kn)$:

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\left\langle\mathbf{v}_{\mathbf{x},i},\mathbf{v}_{\mathbf{x},j}\right\rangle x_ix_j = \frac{1}{2}\sum_{f=1}^{k}\left[\left(\sum_{j=1}^{n}v_{\mathbf{x},j,f}x_j\right)^2 - \sum_{j=1}^{n}v_{\mathbf{x},j,f}^2x_j^2\right] \tag{7}$$

In summary, in IFM, the model parameters are $\Theta = \{w_0, w_i, \mathbf{v}_i, \mathbf{W}_l, \mathbf{b}_l, \mathbf{P}\}$. Compared to the FM model, the additional model parameters of IFM are $\{\mathbf{W}_l, \mathbf{b}_l, \mathbf{P}\}$, which are mainly used for capturing the input-aware factors.

3.2 Relationship with FM and AFM

It is easy to see that, in the IFM model, by fixing all $m_{\mathbf{x},i}$ to 1, $w_{\mathbf{x},i}$ and $\mathbf{v}_{\mathbf{x},i}$ only depend on the i-th feature, and IFM reduces to the original FM model. On the other hand, if we neglect the second term of Equation 1, AFM [Xiao *et al.*, 2017] can be seen as a special case of IFM.

To show this, we compare the formulation of feature interaction in AFM with ours:

$$\begin{aligned} \sum_{i=1}^{n}\sum_{j=i+1}^{n}\left\langle\mathbf{v}_{\mathbf{x},i},\mathbf{v}_{\mathbf{x},j}\right\rangle x_ix_j =& \sum_{i=1}^{n}\sum_{j=i+1}^{n} m_{\mathbf{x},i}m_{\mathbf{x},j}\left\langle\mathbf{v}_i,\mathbf{v}_j\right\rangle x_ix_j \\ =& \mathbf{p}^T\sum_{i=1}^{n}\sum_{j=i+1}^{n} a_{i,j}\left\langle\mathbf{v}_i\odot\mathbf{v}_j\right\rangle x_ix_j \end{aligned} \tag{8}$$

As can be seen from Equation 8, by replacing $m_{\mathbf{x},i}m_{\mathbf{x},j}$ with $a_{i,j}$ and fixing $\mathbf{p}^T$ to a constant vector of $[1, ..., 1]$, we can exactly recover the AFM model.

3.3 Learning

As IFM directly enhances FMs from the perspective of data modelling, it can also be applied to a variety of prediction tasks, including regression, classification and ranking. Different objective functions should be used to customize the IFM model for different tasks. For binary classifications, the loss function is log loss. For regression, a commonly adopted loss function is the squared loss:

$$L_{reg} = \sum_{\mathbf{x}\in\chi}\left(\hat{y}_{IFM}\left(\mathbf{x}\right) - y\left(\mathbf{x}\right)\right)^2 \tag{9}$$

where χ denotes the training data, and $y(\mathbf{x})$, $\hat{y}_{IFM}(\mathbf{x})$ denote the label and target of the input $\mathbf{x}$, respectively. In this work, we mainly focus on the regression task and optimize the squared loss of Equation 3. The optimization for ranking and classification tasks can be done in the same way. To optimize the loss function, we employ adaptive gradient descent (AdaGrad), a universal solver for neural network models. The key to implementing the AdaGrad algorithm is to obtain the derivative of the squared loss L_{reg} $w.r.t.$ each parameter with an adaptive learning rate. In our model, not only can we train all parameters directly through backpropagation, but also we can pretrain the feature weight w_i and embedding vector $\mathbf{v}_i$ through a FM.

Overfitting is a perpetual issue in model training and we employ dropout [Srivastava *et al.*, 2014] and L_2 regularization to control the overfitting of FEN. In doing so, the actual objective function to be optimized is:

$$L_{reg=} \sum_{\mathbf{x} \in \chi} \left(\hat{y}_{IFM}\left(\mathbf{x}\right) - y\left(\mathbf{x}\right)\right)^2 + \lambda \|\mathbf{\Phi}\|^2 \tag{10}$$

where λ controls the regularization strength and $\mathbf{\Phi}$ denotes the weight matrix $\{\mathbf{W}, \mathbf{P}\}$ in FEN.

4 Related Work

As extensions of FMs, GBFM [Cheng *et al.*, 2014] selects good features using gradient boosting and models only the interactions between good features. Field-aware FM (FFM) [Juan *et al.*, 2016] enables each variable to have multiple embedding vectors for feature interactions, but its large number of parameters may result in undesirable computational cost. In our work, we aim to boost the performance of FMs by explicitly considering the impact of each individual input on the representation of features, which refines the weight and embedding vector of each feature with regard to different instances.

Meanwhile, an increasing number of researchers are interested in employing deep learning for prediction. Specifically, some models use DNN to explore high-order feature interactions: Wide&Deep [Cheng *et al.*, 2016], employs a Multilayer Perceptron(MLP) on the concatenation of feature embedding vectors to learn feature interactions; DeepCross [Shan *et al.*, 2016] applies a deep residual MLP to learn cross features; PNN [Qu *et al.*, 2016] introduces a product layer between the embedding layer and DNN layers to learn feature interaction; DeepFM [Guo *et al.*, 2017] uses the FM component and the deep component to get 2-order and high-order interaction information; NFM [He and Chua, 2017] extends FMs by using DNN to explore high-order nonlinear feature interactions; xDeepFM [Lian *et al.*, 2018] learns certain bounded-degree feature interactions explicitly and arbitrary low/high-order feature interactions implicitly. Some other models extend FMs by discriminating the importance of feature interactions. For example, AFM [Xiao *et al.*, 2017] uses attention mechanism to weight different feature interactions, and FwFM [Pan *et al.*, 2018] uses mutual information to distinguish the strength of feature interactions.

In all the above models, each feature holds a fixed representation for different instances, instead of an adaptive representation according to each unique instance. Consequently, the performance of these methods may be limited, as the impact of each instance on the representation of features is ignored. By extending FMs with a more accurate and flexible representation of features, IFM is expected to produce superior performance compared to existing methods.

5 Experiments

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

- **(Q1)** How do the key hyper-parameters of IFM (e.g., the dropout ratio and the number of hidden layers) impact its performance?
- **(Q2)** Can the factor estimating network effectively refine the representation of features for different instances?
- **(Q3)** How does IFM perform compared to state-of-the-art methods for sparse prediction?

5.1 Experimental Settings

Datasets. For regression tasks, we evaluate various prediction models on two public datasets: Frappe[2] [Baltrunas *et al.*, 2015] and MovieLens [Harper and Konstan, 2016]. The Frappe dataset has been used for context-aware mobile app recommendation, which contains 96,202 records with 957 users and 4,082 apps. The MovieLens dataset has been used for personalized tag recommendation, which contains 668,953 tag applications of 17,045 users on 23,743 items with 49,657 distinct tags. We follow the data processing details of NFM [He and Chua, 2017] and randomly split instances by 8:1:1 for training, validation and test. For binary classification, we use the Avazu[3] dataset, which was published in the contest of Avazu Click-Through Rate Prediction in 2014. The public dataset is randomly split into training and test sets by 4:1. Meanwhile, we remove the features appearing less than 20 times to reduce dimensionality.

Evaluation Metrics. We use MAE (mean absolute error) and RMSE (root mean square error) for evaluating regression tasks such as recommendation and use AUC (Area Under ROC) and log loss (cross entropy) for binary classification tasks.

Baselines. We compare IFM with the following competitive methods, some of which are state-of-the-art models for sparse prediction.

- **LibFM** [Rendle, 2012]: This is the official implementation[4] for factorization machines that features stochastic gradient descent.
- **Wide**&**Deep** [Cheng *et al.*, 2016]: The wide part is linear regression and the deep part is a three-layer MLPs with layer sizes 1024, 512 and 256.
- **DeepFM** [Guo *et al.*, 2017]: The FM part is a factorization machine, and the deep part is a three-layer MLP with layer sizes 200, 200 and 200.
- **NFM** [He and Chua, 2017]: We take the implementation of NeuralFM and set the number of layers in the hidden layer to 1 with 512 neurons as the original paper.
- **AFM** [Xiao *et al.*, 2017]: We use the implementation for AttentionFM, where the attention factor is set to 16 and the L_2 regularization of attention net is set to 2 as recommended.

[2] http://baltrunas.info/research-menu/frappe

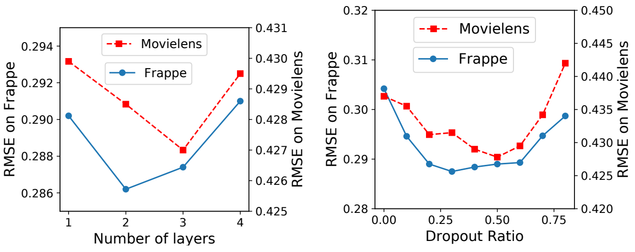
[3] http://www.kaggle.com/c/avazu-ctr-prediction

[4] http://www.libfm.org/

Parameter Settings. We implement our method using Tensorflow[5]. To enable a fair comparison, all methods are learned by optimizing the squared loss (Equation 9) using the Adagrad (Learning rate: 0.01). The batch sizes for Frappe and MovieLens are set to 2048 and 4096, respectively, while the embedding size is set to 256 for all methods. The batch sizes for Avazu is set to 2048 and the embedding size is set to 40. We use a L_2 regularization with λ = 0.00001 for NFM, Wide&Deep, DeepFM and IFM. The default setting for the number of neurons per layer in IFM is 256. For Wide&Deep, AFM, DeepFM and IFM, we find that pre-training their feature embeddings with FM leads to better performance than using random initialization, and so we present their performance with pre-training.

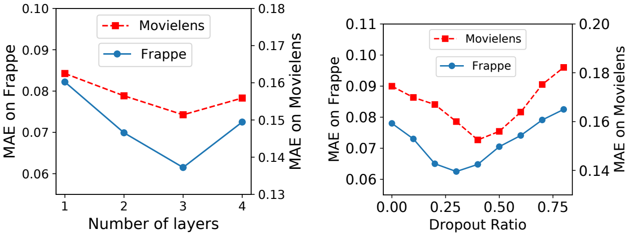
5.2 Hyper-parameter Study (Q1)

We study the impact of hyper-parameters on IFM in this section, including the number of hidden layers, the dropout ratio and activation functions. We conduct experiments on the validation data by holding the settings for the FM prediction layer while varying the settings for FEN.



(a) Number of Layers. (b) Dropout Ratio

Figure 3: Impact of network hyper-parameters (RMSE)



(a) Number of Layers. (b) Dropout Ratio

Figure 4: Impact of network hyper-parameters (MAE)

Depth of Network. Figure 3(a) and Figure 4(a) show the impact of the number of hidden layers in FEN. We can observe that the performance of IFM increases with the increasing of networks at the beginning. However, the model performance starts to degrade when the depth of networks is greater than 2(Frappe) or 3(Movielens). This is caused by overfitting evidenced by the observation that the training error still keeps decreasing.

Dropout ratio. Dropout is an effective regularization technique for neural networks to prevent overfitting. Figure 3(b) and Figure 4(b) show the impact of dropout ratio on FEN. By setting the dropout ratio to a proper value, the performance of IFM on both datasets can be significantly improved. Specifically, the optimal dropout ratios for Frappe and MovieLens are 0.3 and 0.4, respectively. This verifies the usefulness of dropout on the factor estimating network, which also improves the generalization of IFM.

Activation Function. A common practice in deep learning literature is to employ non-linear activation functions on hidden neurons. We thus compare the performance of different activation functions on IFM. As shown in Figure 5, ReLU is more appropriate than others for the two datasets.

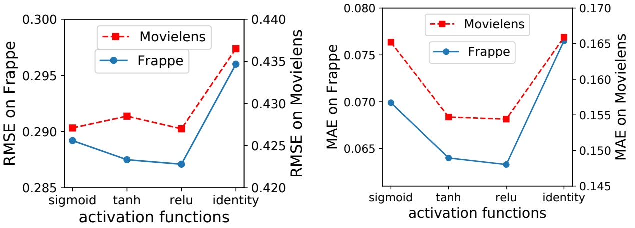
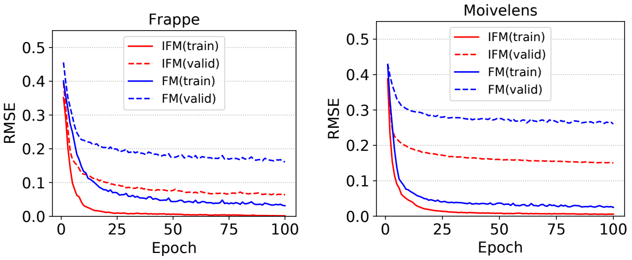


Figure 5: Impact of activation functions on IFM performance



(a) Frappe (b) Movielens

Figure 6: The comparison of convergence behaviors of IFM and FM

5.3 Impact of the Factor Estimating Network (Q2)

The factor estimating network in IFM plays a pivotal role in refining the representations of features for different instances. We first compare IFM with FM to demonstrate the importance of the factor estimating network. Figure 6 shows that IFM generally converges faster than FM. On Frappe and MovieLens, both the training and validation errors of IFM are much lower than those of FM, indicating that IFM can better fit the data and lead to more accurate prediction. This justifies the rationality of IFM's design of learning a unique weight for the same feature in different instances via a neural network, which is the key contribution of this work.

Mechanism Analysis

To better demonstrate the effect of FEN, we select three test instances from Frappe. Table 1 shows the difference in feature representation with respect to different instances. We can see that, in IFM, the same predictor (e.g., x_3 and x_9) may have significantly different degrees of importance in different instances, as evidenced by the diverse input-aware factor

[5] Codes are available at https://github.com/gulyfish/Input-aware-Factorization-Machine

values. By contrast, FM assigns each feature a fixed level of predictive power in three instances, resulting in systematically larger prediction errors.

Table 1: The changes of representation $w.r.t.$ the same features in different instances from Frappe

N	Method	x_3	x_9	$\cdots$	$\hat{y} - y\,(\mathrm{x})$
1	FM	w_3, v_3	w_9, v_9	$\cdots$	0.22
	IFM	**3.12**$*\,(w_3, \mathrm{v}_3)$	**0.15**$*\,(w_9, \mathrm{v}_9)$	$\cdots$	**0.09**
2	FM	w_3, v_3	w_9, v_9	$\cdots$	0.34
	IFM	**0.67**$*\,(w_3, \mathrm{v}_3)$	**0.05**$*\,(w_9, \mathrm{v}_9)$	$\cdots$	**0.16**
3	FM	w_3, v_3	w_9, v_9	$\cdots$	0.14
	IFM	**2.29**$*\,(w_3, \mathrm{v}_3)$	**1.19**$*\,(w_9, \mathrm{v}_9)$	$\cdots$	**0.07**

Performance Analysis of Model Components
Note that, in IFM and other baseline models, each extra model component has its own unique functionality: DNN models high-order feature interactions; attention tries to produce a weight for each feature interaction; FEN focuses on the adaptive representation of features. To provide more insights into the effect of each type of component, we first train all models without using the extra components (i.e., simulating the standard FM model). We then freeze feature embeddings, and train the extra components only. To enable a fair comparison, the hyper-parameters of each model are carefully tuned for best possible performance, and upon convergence, the relative improvements (RI) of each model on Frappe and MovieLens are listed in Table 2. It is clear that with the help of FEN, IFM achieves 35% and 22% improvement on the two datasets, respectively. Although the DNN and attention components also bring noticeable benefits to the FM model, the importance of incorporating adaptive feature representation is clearly highlighted.

Table 2: Relative improvements (RI) of extra components on Frappe and MovieLens

Method	Component	Frappe	MovieLens
		RI	RI
FM	-	-	-
DeepFM	DNN	20%	12%
NFM	DNN	21%	10%
AFM	Attention	12%	9%
IFM	**FEN**	**35%**	**22%**

5.4 Performance Comparison (Q3)

In this section, we compare IFM with several strong baselines, and the results are summarized in Table 3 and Table 4, from which we have the following key observations:

- First, IFM consistently achieves the best results (minimum prediction errors) on three datasets with large performance margins over state-of-the-art methods. This demonstrates overall the effectiveness of IFM and further justifies the importance of input-aware feature representation.
- Second, Wide&Deep, DeepFM and NFM are consistently better than the FM model, which is attributed to the feature extraction by DNN for modelling higher-order feature interactions. However, the superiority of IFM demonstrates that a shallow model with accurate representation of features can achieve even better performance than deep learning methods. This points to a promising direction for future research: developing more effective methods for learning accurate feature representations.

Table 3: Performance comparison for regression task on Frappe and MovieLens (embedding size: 256)

Method	Frappe		MovieLens	
	MAE	RMSE	MAE	RMSE
LibFM	0.1611	0.3352	0.2604	0.4706
Wide&Deep	0.1304	0.3222	0.2385	0.4502
DeepFM	0.0891	0.3149	0.1856	0.4485
NFM	0.0912	0.3074	0.2141	0.4413
AFM	0.1280	0.3085	0.2220	0.4379
IFM	**0.0625***	**0.2872***	**0.1522***	**0.4273***

* indicates that the improvement of IFM over all other methods is statistically significant (α=0.05).

Table 4: Performance comparison for binary classification task on Avazu (embedding size: 40)

	FM	AFM	NFM	DeepFM	IFM
AUC(%)	76.20	77.82	77.98	78.22	**78.50**
Logloss	0.3912	0.3821	0.3798	0.3782	**0.3771**

6 Conclusion

In this paper, we presented an effective predictive model named *Input-aware Factorization Machine* (IFM) for sparse datasets. It aims to enhance traditional FMs by purposefully learning more flexible and accurate representation of features for different instances with the help of a factor estimating network. The major advantage is that it provides a principled mechanism to better reflect the predictive power of each feature in specific situation, while keeping the linear time complexity as the original FMs. Extensive experiments on three real-world benchmark datasets show that the accurate representation of features can yield better performance than capturing high-order feature interactions. In terms of four popular performance metrics, IFM significantly outperforms the classical FM and state-of-the-art deep learning based approaches such as Wide&Deep, AFM, NFM and DeepFM on regression and classification tasks.

References

[Baltrunas *et al.*, 2015] Linas Baltrunas, Karen Church, Alexandros Karatzoglou, and Nuria Oliver. Frappe: Understanding the usage and perception of mobile app recommendations in-the-wild. *CoRR, abs/1505.03014*, 2015.

[Cheng *et al.*, 2014] Chen Cheng, Fen Xia, Tong Zhang, Irwin King, and Michael R Lyu. Gradient boosting factorization machines. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 265–272. ACM, 2014.

[Cheng *et al.*, 2016] Heng Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10. ACM, 2016.

[Guo *et al.*, 2017] 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 26th International Joint Conference on Artificial Intelligence*, pages 1725–1731, 2017.

[Harper and Konstan, 2016] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):19, 2016.

[He and Chua, 2017] 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. ACM, 2017.

[Juan *et al.*, 2016] 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.

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

[Lian *et al.*, 2018] 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. ACM, 2018.

[McMahan *et al.*, 2013] 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. ACM, 2013.

[Pan *et al.*, 2018] 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 on World Wide Web*, pages 1349–1357. International World Wide Web Conferences Steering Committee, 2018.

[Petroni *et al.*, 2015] Fabio Petroni, Luciano Del Corro, and Rainer Gemulla. Core: Context-aware open relation extraction with factorization machines. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1763–1773, 2015.

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

[Rendle *et al.*, 2011] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 635–644. ACM, 2011.

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

[Rendle, 2012] Steffen Rendle. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.*, 3(3):57:1–57:22, May 2012.

[Shan *et al.*, 2016] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 255–262. ACM, 2016.

[Srivastava *et al.*, 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[Sukhbaatar *et al.*, 2015] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

[Weston *et al.*, 2014] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[Xiao *et al.*, 2017] 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. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3119–3125, 2017.