Neural Factorization Machines for Sparse Predictive Analytics*

Xiangnan He
School of Computing
National University of Singapore
Singapore 117417
dcshex@nus.edu.sg

Tat-Seng Chua
School of Computing
National University of Singapore
Singapore 117417
dcscts@nus.edu.sg

ABSTRACT

Many predictive tasks of web applications need to model categorical variables, such as user IDs and demographics like genders and occupations. To apply standard machine learning techniques, these categorical predictors are always converted to a set of binary features via one-hot encoding, making the resultant feature vector highly sparse. To learn from such sparse data effectively, it is crucial to account for the interactions between features.

Factorization Machines (FMs) are a popular solution for efficiently using the second-order feature interactions. However, FM models feature interactions in a linear way, which can be insufficient for capturing the non-linear and complex inherent structure of real-world data. While deep neural networks have recently been applied to learn non-linear feature interactions in industry, such as the *Wide&Deep* by Google and *DeepCross* by Microsoft, the deep structure meanwhile makes them difficult to train.

In this paper, we propose a novel model *Neural Factorization Machine* (NFM) for prediction under sparse settings. NFM seamlessly combines the linearity of FM in modelling second-order feature interactions and the non-linearity of neural network in modelling higher-order feature interactions. Conceptually, NFM is more expressive than FM since FM can be seen as a special case of NFM without hidden layers. Empirical results on two regression tasks show that with one hidden layer only, NFM significantly outperforms FM with a 7.3% relative improvement. Compared to the recent deep learning methods Wide&Deep and DeepCross, our NFM uses a shallower structure but offers better performance, being much easier to train and tune in practice.

CCS CONCEPTS

•Information systems → Information retrieval; Recommender systems; •Computing methodologies → Neural networks; Factorization methods;

KEYWORDS

Factorization Machines, Neural Networks, Deep Learning, Sparse Data, Regression, Recommendation

*NExT research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative.

SIGIR'17, August 7–11, 2017, Shinjuku, Tokyo, Japan

DOI: http://dx.doi.org/10.1145/3077136.3080777

1 INTRODUCTION

Predictive analytics is one of the most important techniques for many information retrieval (IR) and data mining (DM) tasks, ranging from recommendation systems [2, 16], targeted advertising [21], to search ranking [19, 39], visual analysis [35], and event detection [40]. Typically, a predictive task is formulated as estimating a function that maps predictor variables to some target, for example, real valued target for regression and categorical target for classification. Distinct from continuous predictor variables that are naturally found in images and audios, such as raw features, the predictor variables for web applications are mostly discrete and categorical. For example, in online advertising, we need to predict *how likely* (target) a *user* (first predictor variable) of a particular *occupation* (second predictor variable) will click on an *ad* (third predictor variable). To build predictive models with these categorical predictor variables, a common solution is to convert them to a set of binary features (*a.k.a.* feature vector) via one-hot encoding [2, 9, 16, 30, 31]. Thereafter, standard machine learning (ML) techniques such as logistic regression and support vector machines can be applied.

Depending on the possible values of categorical predictor variables, the generated feature vector can be of very high dimension but sparse. To build effective ML models with such sparse data, it is crucial to account for the interactions between features [4, 23, 31]. Many successful solutions in both industry and academia largely rely on manually crafting combinatorial features [9], *i.e.,* constructing new features by combining multiple predictor variables, also known as *cross features*. For example, we can cross variable $occupation = \{banker, doctor\}$ with $gender = \{M, F\}$ and get a new $occupation\_gender = \{banker\_M, banker\_F, doctor\_M, doctor\_F\}$. It is well known that top data scientists are usually masters of crafting combinatorial features, which play a key role in their winning formulas [31]. However, the power of such features comes at a high cost, since it requires heavy engineering efforts and useful domain knowledge to design effective features. Thus these solutions can be difficult to generalize to new problems or domains.

Instead of augmenting feature vectors manually, another solution is to design a ML model to learn feature interactions from raw data automatically. A popular approach is factorization machines (FMs) [27], which embeds features into a latent space and models the interactions between features via inner product of their embedding vectors. While FM has yielded great promise[1] in many prediction tasks [2, 8, 21, 24], we argue that its performance can be limited by its linearity, as well as the modelling of pairwise (*i.e.,* second-order) feature interactions only. Specifically, for real-world data

[1] https://securityintelligence.com/factorization-machines-a-new-way-of-looking-at-machine-learning

that have complex and non-linear underlying structure, FM may not be expressive enough. Although higher-order FMs have been proposed [27], they still belong to the family of linear models and are claimed to be difficult to estimate [28]. Moreover, they are known to have only marginal improvements over FM, which we suspect the reason might be due to the modelling of higher-order interactions in a linear way.

In this work, we propose a novel model for sparse data prediction named *Neural Factorization Machines* (NFMs), which enhances FMs by modelling higher-order and non-linear feature interactions. By devising a new operation in neural network modelling — *Bilinear Interaction* (Bi-Interaction) pooling — we subsume FM under the neural network framework for the first time. Through stacking non-linear layers above the Bi-Interaction layer, we are able to deepen the shallow linear FM, modelling higher-order and non-linear feature interactions effectively to improve FM's expressiveness. In contrast to traditional deep learning methods that simply concatenate [9, 31, 44] or average [16, 36] embedding vectors in the low level, our use of Bi-Interaction pooling encodes more informative feature interactions, greatly facilitating the following "deep" layers to learn meaningful information. We conduct extensive experiments on two public benchmarks for context-aware prediction and personalized tag recommendation. With one hidden layer only, our NFM significantly outperforms FM (the official LibFM [28] implementation) with a 7.3% improvement. Compared to the state-of-the-art deep learning methods — the 3-layer Wide&Deep [9] and 10-layer DeepCross [31] — our 1-layer NFM shows consistent improvements with a much simpler structure and fewer model parameters. Our implementation is available at: https://github.com/hexiangnan/neural_factorization_machine.

The main contributions of this work are summarized as follows.

(1) To the best of our knowledge, we are the first to introduce the Bi-Interaction pooling operation in neural network modelling, and present a new neural network view for FM.
(2) Based on this new view, we develop a novel NFM model to deepen FM under the neural network framework for learning higher-order and non-linear feature interactions.
(3) We conduct extensive experiments on two real-world tasks to study the Bi-Interaction pooling and NFM model, demonstrating the effectiveness of NFM and great promise in using neural networks for prediction under sparse settings.

2 MODELLING FEATURE INTERACTIONS

Due to the large space of combinatorial features, traditional solutions typically rely on manual feature engineering efforts or feature selection techniques like boosted decision trees to select important feature interactions. One limitation of such solutions is that they cannot generalize to combinatorial features that have not appeared in the training data. In recent years, embedding-based methods become increasingly popular, which try to learn feature interactions from raw data [9, 18, 24, 30, 31, 44]. By embedding high-dimensional sparse features into a low-dimensional latent space, the model can generalize to unseen feature combinations. Regardless of domain, we can categorize the approaches into two types: 1) factorization machine-based linear models, and 2) neural network-based non-linear models. In what follows, we shortly recapitulate the two representative techniques.

2.1 Factorization Machines

Factorization machines are originally proposed for collaborative recommendation [27, 30]. Given a real valued feature vector $\mathbf{x} \in \mathbb{R}^n$, FM estimates the target by modelling all interactions between each pair of features via factorized interaction parameters:

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

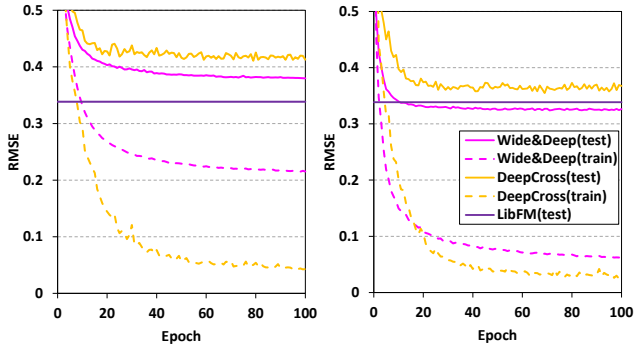
where w_0 is the global bias, w_i models the interaction of the i-th feature to the target. The $\mathbf{v}_i^T \mathbf{v}_j$ term denotes the factorized interaction, where $\mathbf{v}_i \in \mathbb{R}^k$ denotes the embedding vector for feature i, and k is the size of embedding vector, also termed as *number of latent factors* in literature. Note that due to the coefficient $x_i x_j$, only interactions of non-zero features are considered.

One main power of FM stems from its generality — in contrast to matrix factorization (MF) that models the relation of two entities only [17], FM is a general predictor working with any real valued feature vector for supervised learning. Clearly, it enhances linear/logistic regression (LR) using the second-order factorized interactions between features. By specifying input features, Rendle [27] showed that FM can mimic many specific factorization models such as the standard MF, parallel factor analysis, and SVD++ [22], Owing to such genericity, FM has been recognized as one of the most effective embedding methods for sparse data prediction. It has been successfully applied to many predictive tasks, ranging from online advertising [21], microblog retrieval [26], to open relation extraction [25].

2.1.1 ***Expressiveness Limitation of FM.*** Despite effectiveness, we point out that FM still belongs to the family of (multivariate) linear models. In other words, the predicted target $\hat{y}(\mathbf{x})$ is linear *w.r.t.* each model parameter [28]. Formally, for each model parameter $\theta \in \{w_0, \{w_i\}, \{v_{if}\}\}$, we can have $\hat{y}(\mathbf{x}) = g + h\theta$, where g and h are expressions independent of θ. Unfortunately, the underlying structure of real-world data is often highly non-linear and cannot be accurately approximated by linear models [12]. As such, FM may suffer from insufficient representation ability for modelling real data with complex inherent structure and regularities.

In terms of methodology, many variants [18, 21, 24, 38] of FM have been developed. For example, Hong *et al.* [18] proposed Co-FMs to learn from multi-view data; Oentaryo *et al.* [24] encoded prior knowledge of features to FM by designing a hierarchical regularizer; and Lin *et al.* [21] proposed field-aware FM, which learned multiple embedding vectors for a feature to differentiate its interaction with features of different fields. More recently, Xiao *et al.* [38] proposed attentional FM, using an attention network [7, 39] to learn the importance of each feature interaction. However, these variants are all linear extensions of FM and model the second-order feature interactions only. As such, they can suffer from the same expressiveness issue for modelling real-world data.

In this work, we contribute improvements on the expressiveness of FM by endowing it the ability of non-linearity modelling. The idea is to perform non-linear transformation on the latent space



(a) Random initialization (b) FM as pre-training

Figure 1: Training and test error of each epoch of Wide&Deep and DeepCross on Frappe. Random initialization of embeddings leads to poor performance, worse than LibFM (a). Initializing using the embeddings learned by FM improves the performance significantly (b). This reveals optimization difficulties for training deep neural networks.

of the second-order feature interactions; meanwhile, higher-order feature interactions can be captured.

2.2 Deep Neural Networks

In recent five years, deep neural networks (DNNs) have achieved immense success and have been widely used on speech recognition, computer vision and natural language processing. However, the use of DNNs is not as widespread among the IR and DM community. In our view, we think one reason might be that most data of IR and DM tasks are naturally sparse, such as user behaviors, documents/queries and various features converted from categorical variables. Although DNNs have exhibited strong ability to learn patterns from dense data [14], the use of DNNs on sparse data has received less scrutiny, and it is unclear how to employ DNNs for effectively learning feature interactions under sparse settings.

Until very recently, some work [6, 9, 16, 31, 44] started to explore DNNs for some scenarios of sparse predictive analytics. In [16], He *et al.* presented a neural collaborative filtering (NCF) framework to learn interactions between users and items. Later, the NCF framework was extended to model attribute interactions for attribute-aware CF [37]. However, their methods are only applicable to learn interactions between two entities and do not directly support the general setting of supervised learning. Zhang *et al.* [44] developed a FM-supported Neural Network (FNN), which uses the feature embeddings learned by FM to initialize DNNs. Cheng *et al.* [9] proposed Wide&Deep for App recommendation, where the deep part is a multi-layer perceptron (MLP) on the concatenation of feature embedding vectors to learn feature interactions. Shan *et al.* [31] proposed DeepCross for ads prediction, which shares a similar framework with Wide&Deep by replacing the MLP with the state-of-the-art residual network [14].

2.2.1 ***Optimization Difficulties of DNN.*** It is worthwhile to mention the common structure of these neural network-based approaches, *i.e.,* stacking multiple layers above the concatenation of embedding vectors to learn feature interactions. The expectation is that the multiple layers can learn combinatorial features of arbitrary orders in an implicit way [31]. However, we find a key weakness of such an architecture is that simply concatenating feature embedding vectors carries too little information about feature interactions in the low level. An empirical evidence is from He *et al.*'s recent work [16], which shows that simply concatenating user and item embedding vectors leads to very poor results for collaborative filtering. To remedy this, one has to rely on the following deep layers to learn meaningful interaction function. While it is claimed that multiple non-linear layers are able to learn feature interactions well [9, 31], such a deep architecture can be difficult to optimize in practice due to the notorious problems of vanishing/exploding gradients, overfitting, degradation, among others [14].

To demonstrate optimization difficulties of DNNs empirically, we plot the training and test error of each epoch of Wide&Deep and DeepCross on the Frappe data in Figure 1. We use the same architecture and parameters as reported in their papers, where Wide&Deep applies a 3-layer tower structure and DeepCross uses a 10-layer residual network with successively decreasing hidden units. From Figure 1a, we can see that training the two models from scratch leads to a performance much worse than the shallow FM model. For Wide&Deep, the training error is relatively high, which is likely because of the degradation problem [14]. For DeepCross, the very low training error but high test error implies that the model is overfitting. Inspired by FNN [44], we further explore the use of feature embeddings learned by FM to initialize DNNs, which can be seen as a pre-training step. As can be seen from Figure 1b, both models achieve much better performance (over 11% improvements). For Wide&Deep, the degradation problem is well addressed, evidenced by the much lower training error, and the test performance is better than LibFM. However for the 10-layer DeepCross, it still suffers from severe overfitting and underperforms LibFM. These relatively negative results reveal optimization difficulties for training DNNs.

In this work, we present a new paradigm of neural networks for sparse data prediction. Instead of concatenating feature embedding vectors, we propose a new Bi-Interaction operation that models the second-order feature interactions. This results in a much more informative representation in the low level, greatly helping the subsequent non-linear layers to learn higher-order interactions.

3 NEURAL FACTORIZATION MACHINES

We first present the NFM model that unifies the strengths of FMs and neural networks for sparse data modelling. We then discuss the learning procedure and how to employ some useful techniques in neural networks — dropout and batch normalization — for NFM.

3.1 The NFM Model

Similar to factorization machine, NFM is a general machine learner working with any real valued feature vector. Given a sparse vector $\mathbf{x} \in \mathbb{R}^n$ as input, where a feature value $x_i = 0$ means the i-th feature does not exist in the instance, NFM estimates the target as:

$$\hat{y}_{NFM}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + f(\mathbf{x}), \tag{2}$$

where the first and second terms are the linear regression part similar to that for FM, which models global bias of data and weight

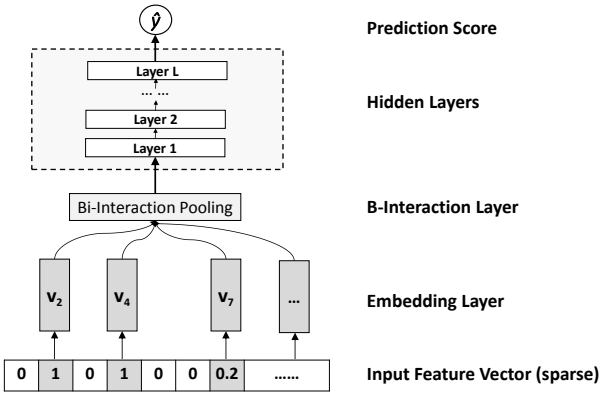


Figure 2: Neural Factorization Machines model (the first-order linear regression part is not shown for clarity).

of features. The third term $f(\mathbf{x})$ is the core component of NFM for modelling feature interactions, which is a multi-layered feed-forward neural network as shown in Figure 2. In what follows, we elaborate the design of $f(\mathbf{x})$ layer by layer.

Embedding Layer. The embedding layer is a fully connected layer that projects each feature to a dense vector representation. Formally, let $\mathbf{v}_i \in \mathbb{R}^k$ be the embedding vector for the i-th feature. After embedding, we obtain a set of embedding vectors $\mathcal{V}_x = \{x_1\mathbf{v}_1, ..., x_n\mathbf{v}_n\}$ to represent the input feature vector $\mathbf{x}$. Owing to sparse representation of $\mathbf{x}$, we only need to include the embedding vectors for non-zero features, *i.e.,* $\mathcal{V}_x = \{x_i\mathbf{v}_i\}$ where $x_i \neq 0$. Note that we have rescaled an embedding vector by its input feature value, rather than simply an embedding table lookup, so as to account for the real valued features [27].

Bi-Interaction Layer. We then feed the embedding set $\mathcal{V}_x$ into the Bi-Interaction layer, which is a pooling operation that converts a set of embedding vectors to one vector:

$$f_{BI}(\mathcal{V}_x) = \sum_{i=1}^{n}\sum_{j=i+1}^{n} x_i\mathbf{v}_i \odot x_j\mathbf{v}_j, \tag{3}$$

where $\odot$ denotes the element-wise product of two vectors, that is, $(\mathbf{v}_i \odot \mathbf{v}_j)_k = v_{ik}v_{jk}$. Clearly, the output of Bi-Interaction pooling is a k-dimension vector that encodes the second-order interactions between features in the embedding space.

It is worth pointing out that our proposal of Bi-Interaction pooling does not introduce extra model parameter, and more importantly, it can be efficiently computed in linear time. This property is the same with average/max pooling and concatenation that are rather simple but commonly used in neural network approaches [16, 31, 36]. To show the linear time complexity of evaluating Bi-Interaction pooling, we reformulate Equation (3) as:

$$f_{BI}(\mathcal{V}_x) = \frac{1}{2}\left[(\sum_{i=1}^{n} x_i\mathbf{v}_i)^2 - \sum_{i=1}^{n}(x_i\mathbf{v}_i)^2\right], \tag{4}$$

where we use the symbol $\mathbf{v}^2$ to denote $\mathbf{v} \odot \mathbf{v}$. By considering the sparsity of $\mathbf{x}$, we can actually perform Bi-Interaction pooling in $O(kN_x)$ time, where N_x denotes the number of non-zero entries in $\mathbf{x}$. This is a very appealing property, meaning that the benefit of Bi-Interaction pooling in modelling pairwise feature interactions does not involve any additional cost.

Hidden Layers. Above the Bi-Interaction pooling layer is a stack of fully connected layers, which are capable of learning higher-order interactions between features [31]. Formally, the definition of fully connected layers is as follows:

$$\begin{aligned} \mathbf{z}_1 &= \sigma_1(\mathbf{W}_1 f_{BI}(\mathcal{V}_x) + \mathbf{b}_1), \\ \mathbf{z}_2 &= \sigma_2(\mathbf{W}_2\mathbf{z}_1 + \mathbf{b}_2), \\ &\cdots\cdots \\ \mathbf{z}_L &= \sigma_L(\mathbf{W}_L\mathbf{z}_{L-1} + \mathbf{b}_L), \end{aligned} \tag{5}$$

where L denotes the number of hidden layers, $\mathbf{W}_l$, $\mathbf{b}_l$ and σ_l denote the weight matrix, bias vector and activation function for the l-th layer, respectively. By specifying non-linear activation functions, such as sigmoid, hyperbolic tangent (tanh), and Rectifier (ReLU), we allow the model to learn higher-order feature interactions in a non-linear way. This is advantageous to existing methods for higher-order interaction learning, such as higher-Order FM [3] and Exponential Machines [23], which only support the learning of higher-order interactions in a linear way. As for the structure of fully connected layers (*i.e.,* size of each layer), one can freely choose tower [9, 16], constant [31], and diamond [44], among others.

Prediction Layer. At last, the output vector of the last hidden layer $\mathbf{z}_L$ is transformed to the final prediction score:

$$f(\mathbf{x}) = \mathbf{h}^T\mathbf{z}_L, \tag{6}$$

where vector $\mathbf{h}$ denotes the neuron weights of the prediction layer.

To summarize, we give the formulation NFM's predictive model as:

$$\begin{aligned} \hat{y}_{NFM}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i \\ + \mathbf{h}^T\sigma_L(\mathbf{W}_L(...\sigma_1(\mathbf{W}_1 f_{BI}(\mathcal{V}_x) + \mathbf{b}_1)...) + \mathbf{b}_L), \end{aligned} \tag{7}$$

with all model parameters $\Theta = \{w_0, \{w_i, \mathbf{v}_i\}, \mathbf{h}, \{\mathbf{W}_l, \mathbf{b}_l\}\}$. Compared to FM, the additional model parameters of NFM are mainly $\{\mathbf{W}_l, \mathbf{b}_l\}$, which are used for learning higher-order interactions between features. In remainder of this subsection, we first show how NFM generalizes FM and discuss the connection of NFM between existing deep learning methods; we then analyze the time complexity of evaluating NFM model.

3.1.1 ***NFM Generalizes FM.*** FM is a shallow and linear model, which can be seen as a special case of NFM with no hidden layer. To show this, we set L to zero and directly project the output of Bi-Interaction pooling to prediction score. We term this simplified model as NFM-0, which is formulated as:

$$\begin{aligned} \hat{y}_{NFM-0} &= w_0 + \sum_{i=1}^{n} w_i x_i + \mathbf{h}^T\sum_{i=1}^{n}\sum_{j=i+1}^{n} x_i\mathbf{v}_i \odot x_j\mathbf{v}_j \\ &= w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n}\sum_{j=i+1}^{n}\sum_{f=1}^{k} h_f v_{if} v_{jf} \cdot x_i x_j. \end{aligned} \tag{8}$$

As can be seen, by fixing $\mathbf{h}$ to a constant vector of $(1, ..., 1)$, we can exactly recover the FM model[2].

[2]Note that for NFM-0, a trainable $\mathbf{h}$ can not improve the expressiveness of FM, since its impact on prediction can be absorbed into feature embeddings.

It is worth pointing out that, to our knowledge, this is the first time FM has been expressed under the neural network framework. While a recent work by Blondel *et al.* [4] has unified FM and Polynomial network via kernelization, their kernel view of FM only provides a new training algorithm and does not provide insight for improving the FM model. Our new view of FM is highly instructive and provides more insight for improving FM. Particularly, we allow the use of various neural network techniques on FM to improve its learning and generalization ability. For example, we can use dropout [33] — a well-known technique in deep learning community to prevent overfitting — on the Bi-Interaction layer as a way to regularize FM, which we find can be even more effective than the conventional L_2 regularization (see Figure 3 of Section 4).

3.1.2 ***Relation to Wide&Deep and DeepCross.*** NFM has a similar multi-layered neural architecture with several existing deep learning solutions [9, 16, 31]. The key difference is in the Bi-Interaction pooling component, which is uniquely used in NFM. Specifically, if we replace the Bi-Interaction pooling with concatenation and apply a tower-structure MLP (or residual units [14]) to hidden layers, we can recover the Wide&Deep (or DeepCross) model. An obvious limitation of the concatenation operation is that it does not account for any interaction between features. As such, these deep learning approaches have to rely entirely on the following deep layers to learn meaningful feature interactions, which unfortunately can be difficult to train in practice (*cf.* Section 2.2.1). Our use of Bi-Interaction pooling captures second-order feature interactions in the low level, which is more informative than the concatenation operation. This greatly facilitates the subsequent hidden layers of NFM to learn useful higher-order feature interactions in a much easier way.

3.1.3 ***Time Complexity Analysis.*** We have shown in Equation (4) that Bi-Interaction pooling can be efficiently computed in $O(kN_x)$ time, which is the same as FM. Then the additional costs are caused by the hidden layers. For hidden layer l, the matrix-vector multiplication is the main operation which can be done in $O(d_{l-1}d_l)$, where d_l denotes the dimension of the l-th hidden layer and $d_0 = k$. The prediction layer only involves inner product of two vectors, for which the complexity is $O(d_L)$. As such, the overall time complexity for evaluating a NFM model is $O(kN_x + \sum_{l=1}^{L} d_{l-1}d_l)$, which is the same as that of Wide&Deep and DeepCross.

3.2 Learning

NFM can be applied to a variety of prediction tasks, including regression, classification and ranking. To estimate model parameters of NFM, we need to specify an objective function to optimize. For regression, a commonly used objective function is the squared loss:

$$L_{reg} = \sum_{\mathbf{x} \in \mathcal{X}} (\hat{y}(\mathbf{x}) - y(\mathbf{x}))^2, \tag{9}$$

where $\mathcal{X}$ denotes the set of instances for training, and $y(\mathbf{x})$ denotes the target of instance $\mathbf{x}$. The regularization terms are optional and omitted here, since we found that some techniques in neural network modelling such as dropout can well prevent NFM from overfitting. For classification task, we can optimize the hinge loss or log loss [16]. For ranking task, we can optimize pairwise personalized ranking loss [29, 37] or contrastive max-margin loss [43]. In this work, we focus on the regression task and optimize the squared loss of Equation (9). The optimization for ranking/classification tasks can be done in the same way.

Stochastic gradient descent (SGD) is a universal solver for optimizing neural network models. It iteratively updates the parameters until convergence. In each time, it randomly selects a training instance $\mathbf{x}$, updating each model parameter towards the direction of its negative gradient:

$$\theta = \theta - \eta \cdot 2(\hat{y}(\mathbf{x}) - y(\mathbf{x})) \frac{d\hat{y}(\mathbf{x})}{d\theta}, \tag{10}$$

where $\theta \in \Theta$ is a trainable model parameter, and $\eta > 0$ is the learning rate that controls the step size of gradient descent. As NFM is a multi-layered neural network model, the gradient of $\hat{y}(\mathbf{x})$ *w.r.t.* to each model parameter can be derived using the chain rule. Here we give only the differentiation of the Bi-Interaction pooling layer, since other layers are just standard operations in neural network modelling and have been widely implemented in ML toolkits like TensorFlow and Keras.

$$\frac{df_{BI}(\mathcal{V}_x)}{d\mathbf{v}_i} = (\sum_{j=1}^{n} x_j \mathbf{v}_j) x_i - x_i^2 \mathbf{v}_i = \sum_{j=1, j \neq i}^{n} x_i x_j \mathbf{v}_j. \tag{11}$$

As such, for end-to-end neural methods, upon plugging in the Bi-Interaction pooling layer, they can still be learned end-to-end. To leverage the vectorization and parallelism speedup of modern computing platforms, mini-batch SGD is more widely used in practice, which samples a batch of training instances and updates model parameters based on the batch. In our implementation, we use mini-batch Adagrad [10] as the optimizer, rather than the vanilla SGD. Its main advantage is that the learning rate can be self adapted during the training phase, which eases the pain of choosing a proper learning rate and leads to faster convergence than the vanilla SGD.

3.2.1 ***Dropout.*** While neural network models have strong representation ability, they are also easy to overfit the training data. Dropout [33] is a regularization technique for neural networks to prevent overfitting. The idea is to randomly drop neurons (along with their connections) of the neural network during training. That is, in each parameter update, only part of the model parameters that contributes to the prediction of $\hat{y}(\mathbf{x})$ will be updated. Through this process, it can prevent complex co-adaptations of neurons on training data. It is important to note that in the testing phase, dropout must be disabled and the whole network is used for estimating $\hat{y}(\mathbf{x})$. As such, dropout can also be seen as performing model averaging with smaller neural networks [33].

In NFM, to avoid feature embeddings co-adapt to each other and overfit the data, we propose to adopt dropout on the Bi-Interaction layer. Specifically, after obtaining $f_{BI}(\mathcal{V}_x)$ which is a k-dimensional vector of latent factors, we randomly drop ρ percent of latent factors, where ρ is termed as the dropout ratio. Since NFM with no hidden layer degrades to the FM model, it can be seen as a new way to regularize FM. Moreover, we also apply dropout on each hidden layer of NFM to prevent the learning of higher-order feature interactions from co-adaptations and overfitting.

3.2.2 ***Batch Normalization.*** One difficulty of training multi-layered neural networks is caused by the fact of covariance shift [20]. It means that the distribution of each layer's inputs changes during

training, as the parameters of the previous layers change. As a result, the later layer needs to adapt to these changes (which are often noisy) when updating its parameters, which adversely slows down the training. To address the problem, Ioffe and Szegedy [20] proposed *batch normalization* (BN), which normalizes layer inputs to a zero-mean unit-variance Gaussian distribution for each training mini-batch. It has been shown that BN leads to faster convergence and better performance in several computer vision tasks [14, 42].

Formally, let the input vector to a layer be $\mathbf{x}_i \in \mathbb{R}^d$ and all input vectors to the layer of the mini-batch be $\mathcal{B} = \{\mathbf{x}_i\}$, then BN normalizes $\mathbf{x}_i$ as:

$$BN(\mathbf{x}_i) = \gamma \odot (\frac{\mathbf{x}_i - \mu_B}{\sigma_B}) + \beta, \tag{12}$$

where $\mu_B = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \mathbf{x}_i$ denotes the mini-batch mean, $\sigma_B^2 = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}(\mathbf{x}_i - \mu_B)^2$ denotes the mini-batch variance, and γ and β are trainable parameters (vectors) to scale and shift the normalized value to restore the representation power of the network. Note that in testing, BN also needs to be applied, where μ_B and σ_B are estimated from the whole training data.

In NFM, to avoid the update of feature embeddings changing the input distribution to hidden layers or prediction layer, we perform BN on the output of the Bi-Interaction pooling. For each successive hidden layer, the BN is also applied.

4 EXPERIMENTS

As the key contributions of this work are on Bi-Interaction pooling in neural network modelling and the design of NFM for sparse data prediction, we conduct experiments to answer the following research questions:

- **RQ1** Can Bi-Interaction pooling effectively capture the second-order feature interactions? How does dropout and batch normalization work for Bi-Interaction pooling?
- **RQ2** Are hidden layers of NFM useful for capturing higher-order interactions between features and improving the expressiveness of FM?
- **RQ3** How does NFM perform as compared to higher-order FM and the state-of-the-art deep learning methods Wide&Deep and DeepCross?

In what follows, we first present the experimental settings, followed by answering the above research questions one by one.

4.1 Experimental Settings

4.1.1 ***Datasets.*** We experimented with two publicly accessible datasets: Frappe[3] and MovieLens[4]:

1. Frappe. Frappé is a context-aware app discovery tool. This dataset is constructed by Baltrunas *et al.* [1]. It contains 96, 203 app usage logs of users under different contexts. Besides user ID and app ID, each log contains 8 context variables, including weather, city and daytime (*e.g.,* morning or afternoon). We converted each log (*i.e.,* user ID, app ID and all context variables) to a feature vector using one-hot encoding, resulting in 5, 382 features in total. A target value of 1 means the user has used the app under the context.

[3]http://baltrunas.info/research-menu/frappe
[4]http://grouplens.org/datasets/movielens/latest

Table 1: Statistics of the evaluation datasets.

Dataset	Instance#	Feature#	User#	Item#
Frappe	288,609	5, 382	957	4,082
MovieLens	2,006,859	90, 445	17,045	23,743

2. MovieLens. This is the *Full* version of the latest MovieLens data published by GroupLens [13]. As this work concerns higher-order interactions between features, we study the task of personalized tag recommendation rather than collaborative filtering [16] that considers the second-order interactions only. The tagging part of the data includes 668, 953 tag applications of 17, 045 users on 23, 743 items with 49, 657 distinct tags. We converted each tag application (*i.e.,* user ID, movie ID and tag) to a feature vector, resulting in 90, 445 features in total. A target value of 1 means the user has assigned the tag to the movie.

As both original datasets contain positive instances only (*i.e.,* all instances have target value 1), we sampled two negative instances to pair with one positive instance to ensure the generalization of the predictive model. For each log of Frappe, we randomly sampled two apps that the user has not used in the context; for each tag application of MovieLens, we randomly sampled two tags that the user has not assigned to the movie. Each negative instance is assigned to a target value of -1. Table 1 summarizes the statistics of the final evaluation datasets.

4.1.2 ***Evaluation Protocols.*** We randomly split the dataset into training (70%), validation (20%), and test (10%) sets. The validation set was used for tuning hyper-parameters and the final performance comparison was conducted on the test set. The study of NFM properties (*i.e.,* the answering of RQ1 and RQ2) was performed on the validation set, which can also reflect how we choose the optimal hyper-parameters. To evaluate the performance, we adopted *root mean square error* (RMSE), where a lower RMSE score indicates a better performance. Note that RMSE has been widely used for evaluating regression tasks such as recommendation with explicit ratings [5, 30] and click-through rate prediction [24]. We rounded up the prediction of each model to 1 or -1 if it was out of the range. The one-sample paired t-test was performed to judge the statistical significance where necessary.

4.1.3 ***Baselines.*** We implemented NFM using TensorFlow[5]. We compared with the following competitive embedding-based models that are specifically designed for sparse data prediction:

- **LibFM** [28]. This is the official implementation[6] of FM released by Rendle. It has shown strong performance for personalized tag recommendation and context-aware prediction [30]. We used the SGD learner for a fair comparison with other methods which were all optimized with SGD (or its variants).
- **HOFM**. This is the TensorFlow implementation[7] of higher-order FM, as described in [27]. We experimented with order size 3, since the MovieLens data concerns the ternary relationship between users, movies and tags.
- **Wide&Deep** [9]. As introduced in Section 2.2, the deep part first concatenates feature embeddings, followed by a MLP to model

[5]Codes are available at https://github.com/hexiangnan/neural_factorization_machine
[6]http://www.libfm.org/
[7]https://github.com/geffy/tffm

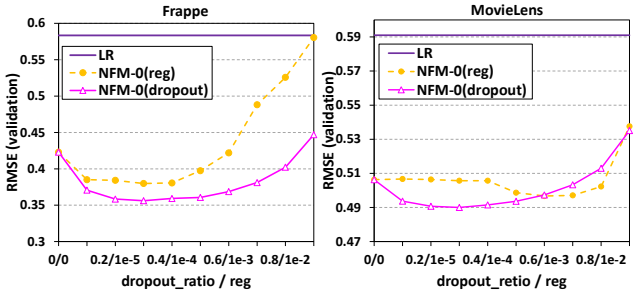


Figure 3: Validation error of NFM-0 *w.r.t.* dropout on the Bi-Interaction layer and L_2 regularization on embeddings.

feature interactions. As the structure of a DNN is difficult to be fully tuned, we used the same structure as reported in their paper, which has three layers with size 1024, 512 and 256, respectively. While the wide part (which is a linear regression model) is subjected to design to incorporate cross features, we used the raw features only for a fair comparison with FM and NFM.

- **DeepCross** [31]. It applies a multi-layered residual network on the concatenation of feature embeddings for learning feature interactions. We used the same structure as reported in their paper, which stacks 5 residual units (each unit has two layers) with the hidden dimension 512, 512, 256, 128 and 64, respectively.

4.1.4 ***Parameter Settings.*** To fairly compare models' capability, we learned all models by optimizing the square loss (Equation (9)). The learning rate was searched in $[0.005, 0.01, 0.02, 0.05]$ for all methods. To prevent overfitting, we tuned the L_2 regularization for linear models LibFM and HOFM in $[1e^{-6}, 5e^{-6}, 1e^{-5}, ..., 1e^{-1}]$, and the dropout ratio for neural network models Wide&Deep, DeepCross and NFM in $[0, 0.1, 0.2, ..., 0.9]$. Note that we found that dropout can well regularize the hidden layers of Wide&Deep and NFM; however it did not work well for the residual units of DeepCross. Besides LibFM that optimized FM with the vanilla SGD, all other methods were optimized with mini-batch Adagrad [10], where the batch size was set to 128 for Frappe and 4096 for MovieLens. Note that the batch size was selected by considering both training time and convergence rate, as a larger batch size usually led to faster training per epoch but slower convergence. For all methods, the early stopping strategy was performed, where we stopped training if the RMSE on validation set increased for 4 successive epochs. Without special mention, we show the results of embedding size 64, and more results of larger embedding sizes are shown in Section 4.4.

4.2 Study of Bi-Interaction Pooling (RQ1)

We empirically study the Bi-Interaction pooling operation. To avoid other components (*e.g.,* hidden layers) affecting the analysis, we study the NFM-0 model that directly projects the output of Bi-Interaction pooling to prediction score with no hidden layer. As discussed in Section 3.1.1, NFM-0 is identical to FM as the trainable $\mathbf{h}$ does not impact model's expressiveness. We first compare dropout with traditional L_2 regularization for preventing model overfitting, and then explore the impact of batch normalization.

4.2.1 ***Dropout Improves Generalization.*** Figure 3 shows the validation error of NFM-0 *w.r.t.* dropout ratio on the Bi-Interaction

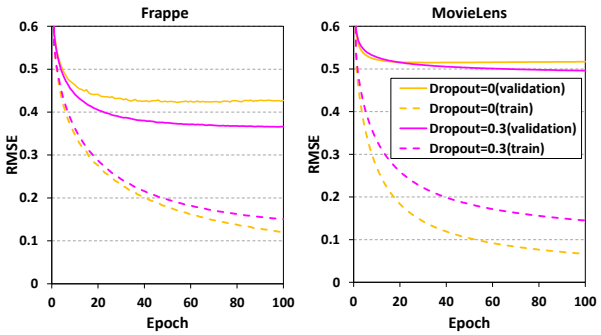


Figure 4: Training and validation error of each epoch of NFM-0 with and without dropout on Bi-Interaction layer.

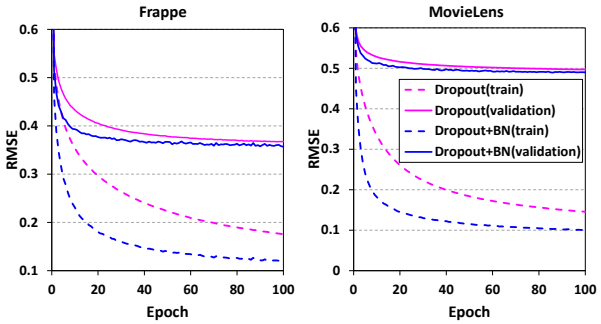


Figure 5: Training and validation error of each epoch of NFM-0 with and without BN on the Bi-Interaction layer.

layer and L_2 regularization on feature embeddings. The performance of linear regression (LR) is also shown for benchmarking the performance of prediction that does not consider feature interactions. First, LR leads to very poor performance, highlighting the importance of modelling interactions between sparse features for prediction. Second, we see that both L_2 regularization and dropout can well prevent overfitting and improve NFM-0's generalization to unseen data. Between the two strategies, dropout offers better performance. Specifically, on Frappe, using a dropout ratio of 0.3 leads to a lowest validation error of 0.3562, which is significantly better than that of L_2 regularization 0.3799. One reason might be that enforcing L_2 regularization only suppresses the values of parameters in each update numerically, while using dropout can be seen as ensembling multiple sub-models [33], which can be more effective. Considering the genericity of FM that subsumes many factorization models, we believe this is a new interesting finding, meaning that dropout can also be an effective strategy to address overfitting of linear latent-factor models.

To be more clear about the effect of dropout, we show the training and validation error of each epoch of NFM-0 with and without dropout in Figure 4. Both datasets show that with a dropout ratio of 0.3, although the training error is higher, the validation error becomes lower. This demonstrates the ability of dropout in preventing overfitting and as such, better generalization can be achieved.

4.2.2 ***Batch Normalization Speeds up Training.*** Figure 5 shows the training and validation error of each epoch of NFM-0 with and without BN on the Bi-Interaction layer. The dropout is

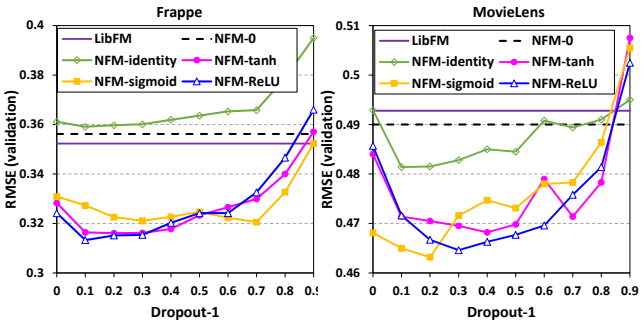


Figure 6: Validation error of LibFM, NFM-0 and NFM with different activation functions on the first hidden layer.

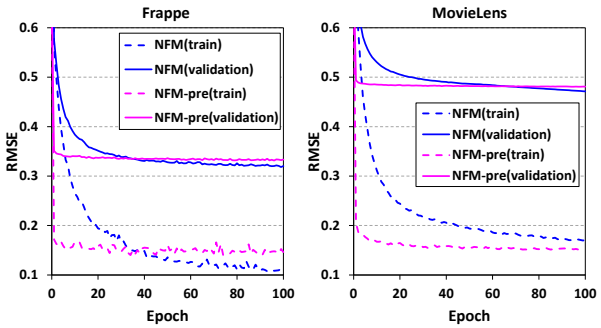


Figure 7: Training and validation error of each epoch of NFM-1 with and without pre-training.

enabled with a ratio of 0.3, and the learning rate is set to 0.02. Focusing on the training error, we can see that BN leads to a faster convergence; on Frappe, when BN is applied, the training error of epoch 20 is even lower than that of epoch 60 without BN; and the validation error indicates that the lower training error is not overfitting — in fact, [14, 20] showed that by addressing the internal covariate shift with BN, the model's generalization ability can be improved. Our result also verifies this point, where using BN leads to slight improvement (although the improvement is not statistically significant). Furthermore, we notice that BN makes the learning less stable, as evidenced by the larger performance fluctuation of blue lines. This is caused by our use of dropout and BN together, as randomly dropping neurons can change the input distribution normalized by BN. It is an interesting direction to effectively combine BN and dropout.

4.3 Impact of Hidden Layers (RQ2)

The hidden layers of NFM play a pivotal role in capturing higher-order interactions between features. To explore the impact, we first add one hidden layer above the Bi-Interaction layer and slightly overuse the name NFM to indicate this specific model. To ensure the same model capability with NFM-0, we set the size of hidden layer the same as the embedding size.

Figure 6 shows the validation error of NFM *w.r.t.* different activation functions and dropout ratios for the hidden layer. The performance of LibFM and NFM-0 are also shown for benchmarking purposes. First and foremost, we observe that by using non-linear activations, NFM's performance is improved with a large margin — compared to NFM-0 which has a similar performance with LibFM, the relative improvement is 11.3% and 5.2% for Frappe and MovieLens, respectively. This highlights the importance of modelling higher-order feature interactions for quality prediction. Among the different non-linear activation functions, there is no obvious winner. Second, when we use the identity function as the activation function, *i.e.,* the hidden layer performs a linear transformation, NFM does not perform that well. This provides evidence to the necessity of learning higher-order feature interactions with non-linear functions.

To see whether a deeper NFM can further improve the performance, we stack more ReLU layers above the Bi-Interaction layer. As it is computationally expensive to tune the size and dropout ratio for each hidden layer separately, we use the same setting for all layers and tune them the same way as NFM-1. As can be seen from Table 2, when we stack more layers, the performance is not further improved, and best performance is when we use one hidden layer only. We have also explored other designs for hidden layers, such as the tower structure and residual units, however, the performance is still not improved. We think the reason is because the Bi-Interaction layer has encoded informative second-order feature interactions, and based on which, a simple non-linear function is sufficient to capture higher-order interactions. To verify this, we replaced the Bi-Interaction layer with concatenation (which leads to the same architecture as Wide&Deep), and found that the performance can be gradually improved with more hidden layers (up to three); however, the best performance achievable is still inferior to that of NFM-1. This demonstrates the value of using a more informative operation for low-level layers, which can ease the burden of higher-level layers for learning meaningful information. As a result, a deep structure becomes not necessarily required.

Table 2: NFM *w.r.t.* different number of hidden layers.

Methods	Frappe	MovieLens
NFM-0	0.3562	0.4901
NFM-1	**0.3133**	**0.4646**
NFM-2	0.3193	0.4681
NFM-3	0.3219	0.4752
NFM-4	0.3202	0.4703

4.3.1 ***Pre-training Speeds up Training.*** It is known that parameter initialization can greatly affect the convergence and performance of DNNs [11, 16], since gradient-based methods can only find local optima for DNNs. As have been shown in Section 2.2.1, initializing with feature embeddings learned by FM can significantly enhance Wide&Deep and DeepCross. Now the question arises, how does pre-training impact NFM?

Figure 7 shows the state of each epoch of NFM-1 with and without pre-training. First, we can see that by using FM embeddings as pre-training, NFM exhibits extremely fast convergence — on both datasets, with 5 epochs only, the performance is on par with 40 epochs of NFM that is trained from scratch (with BN enabled). Second, we find that pre-training does not improve NFM's final performance, and a random initialization can achieve a result that is slightly better than that with pre-training. This demonstrates the robustness of NFM, which is relatively insensitive to parameter initialization. In contrast to the huge impact of pre-training on Wide&Deep and DeepCross (*cf.* Figure 1) that improves both their convergence and final performance, we draw the conclusion that

Table 3: Test error and number of trainable parameters for different methods on latent factors 128 and 256. M denotes "million"; $*$ and $**$ denote the statistical significance for $p < 0.05$ and $p < 0.01$, respectively, compared to the best baseline.

	Frappe				MovieLens			
	Factors=128		Factors=256		Factors=128		Factors=256	
Method	**Param#**	**RMSE**	**Param#**	**RMSE**	**Param#**	**RMSE**	**Param#**	**RMSE**
LibFM [28]	0.69M	0.3437	1.38M	0.3385	11.67M	0.4793	23.24M	0.4735
HOFM	1.38M	0.3405	2.76M	0.3331	23.24M	0.4752	46.40M	0.4636
Wide&Deep [9]	2.66M	0.3621	4.66M	0.3661	12.72M	0.5323	24.69M	0.5313
Wide&Deep (pre-train)	2.66M	0.3311	4.66M	0.3246	12.72M	0.4595	24.69M	0.4512
DeepCross [31]	4.47M	0.4025	8.93M	0.4071	12.71M	0.5885	25.42M	0.5907
DeepCross (pre-train)	4.47M	0.3388	8.93M	0.3548	12.71M	0.5084	25.42M	0.5130
NFM	**0.71M**	**0.3127****	**1.45M**	**0.3095****	**11.68M**	**0.4557***	**23.31M**	**0.4443***

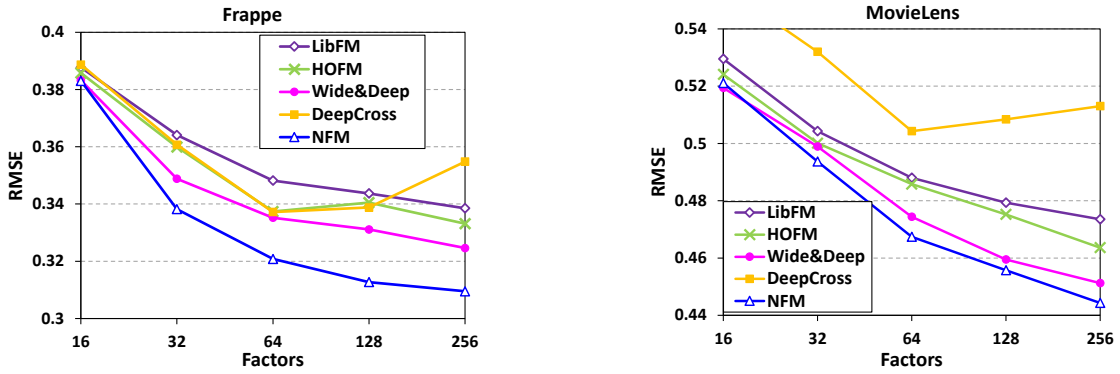


Figure 8: Performance comparison on the test set *w.r.t.* different embedding sizes. LibFM, HOFM and HOFM are trained from random initialization; Wide&Deep and DeepCross are pre-trained with FM feature embeddings.

NFM is much easier to train and optimize, which is due largely to the informative and effective Bi-Interaction pooling operation.

4.4 Performance Comparison (RQ3)

We now compare with state-of-the-art methods. For NFM, we use one hidden layer with ReLU as the activation function, since the baselines DeepCross and Wide&Deep also choose ReLU in their original papers. Note that the most important hyper-parameter for NFM is the dropout ratio, which we use 0.5 for the Bi-Interaction layer and tune the value for the hidden layer.

Figure 8 plots the test RMSE *w.r.t.* different number of latent factors (*i.e.,* embedding sizes), where Wide&Deep and DeepCross are pre-trained with FM to better explore the two methods. Table 3 shows the concrete scores obtained on factors 128 and 256, and the number of model parameters of each method. The scores of Wide&Deep and DeepCross without pre-training are also shown. We have the following three key observations.

First and foremost, NFM consistently achieves the best performance on both datasets with the fewest model parameters besides FM. This demonstrates the effectiveness and rationality of NFM in modelling higher-order and non-linear feature interactions for prediction with sparse data. The performance is followed by Wide&Deep, which uses a 3-layer MLP to learn feature interactions. We have also tried deeper layers for Wide&Deep, however the performance has not been improved. This further verifies the utility of using the informative Bi-Interaction pooling in the low level.

Second, we observe that HOFM shows slight improvement over FM with 1.45% and 1.04% average improvement on Frappe and MovieLens, respectively. This sheds light on the limitation of FM that models only the second-order feature interactions, and thus the usefulness of modelling higher-order interactions. Meanwhile, the large performance gap between HOFM and NFM reflects the value of modelling higher-order interactions in a non-linear way, since HOFM models higher-order interactions linearly and uses much more parameters than NFM.

Lastly, the relatively weak performance of DeepCross reveals that deeper learnings are not always better, as DeepCross is the deepest method among all baselines that utilizes a 10-layer network. On Frappe, DeepCross only achieves a comparable performance with the shallow FM model, while it underperforms FM significantly on MovieLens. We believe that the reasons are due to optimization difficulties and overfitting (as evidenced by the worse performance on factors 128 and 256).

5 CONCLUSION AND FUTURE WORK

In this work, we proposed a novel neural network model NFM, which brings together the effectiveness of linear factorization machines with the strong representation ability of non-linear neural networks for sparse predictive analytics. The key of NFM's architecture is the newly proposed Bi-Interaction operation, based on which we allow a neural network model to learn more informative feature interactions at the lower level. Extensive experiments on two real-world datasets show that with one hidden layer only, NFM significantly outperforms FM, higher-order FM, and state-of-the-art deep learning approaches Wide&Deep and DeepCross.

The work represents the first step towards bridging the gap between linear models and deep learning. Linear models, such as various factorization methods, have shown to be effective for many IR and DM tasks and are easy to interpret. However, their limited expressiveness may hinder the performance when modelling real-world data with complex inherent patterns. While deep learning models have exhibited great expressive power and yielded immense success on speech processing and computer vision, their performance is still unsatisfactory for IR tasks, such as collaborative filtering [16]. In our view, one reason is that most data of IR and DM tasks are naturally sparse; and to date, there still lacks effective deep learning solutions for prediction with sparse data. By connecting neural networks with FM — one of the most powerful linear models for supervised learning — we are able to design a simple yet effective deep learning solution for sparse data prediction.

With recent developments on GPU platforms, it is not technically difficult to build very deep models with hundreds or even thousands of layers [14]. However, deeper models do not necessarily lead to better results, since deeper models are less transparent and more difficult to optimize and tune. As such, we expect future research on deep learning for IR should focus more on designing better neural components or architectures for specific tasks, rather than relying on deeper models for minor improvements. Our proposed Bi-Interaction pooling is an effective neural component for sparse feature modelling, reducing the demand for deeper structure for quality prediction. In future, we will improve the efficiency of NFM by resorting to hashing techniques [32, 41] to make it more suitable for large-scale applications and study its performance for other IR tasks, such as search ranking and targeted advertising. While this work endows FM with non-linearities from predictive model perspective, another viable solution for incorporating non-linearities is to extend the objective function with regularizers like the graph Laplacian [15, 34]. Lastly, we are interested in exploring the Bi-Interaction pooling for recurrent neural networks (RNNs) for sequential data modelling.

REFERENCES

[1] L. Baltrunas, K. Church, A. Karatzoglou, and N. Oliver. Frappe: Understanding the usage and perception of mobile app recommendations in-the-wild. *CoRR*, abs/1505.03014, 2015.

[2] I. Bayer, X. He, B. Kanagal, and S. Rendle. A generic coordinate descent framework for learning from implicit feedback. In *WWW*, 2017.

[3] M. Blondel, A. Fujino, N. Ueda, and M. Ishihata. Higher-order factorization machines. In *NIPS*, 2016.

[4] M. Blondel, M. Ishihata, A. Fujino, and N. Ueda. Polynomial networks and factorization machines: New insights and efficient training algorithms. In *ICML*, 2016.

[5] D. Cao, X. He, L. Nie, X. Wei, X. Hu, S. Wu, and T.-S. Chua. Cross-platform app recommendation by jointly modeling ratings and texts. *ACM TOIS*, 2017.

[6] J. Chen, B. Sun, H. Li, H. Lu, and X.-S. Hua. Deep ctr prediction in display advertising. In *MM*, 2016.

[7] J. Chen, H. Zhang, X. He, L. Nie, W. Liu, and T.-S. Chua. Attentive collaborative filtering: Multimedia recommendation with feature- and item-level attention. In *SIGIR*, 2017.

[8] T. Chen, X. He, and M.-Y. Kan. Context-aware image tweets modelling and recommendation. In *MM*, 2016.

[9] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide & deep learning for recommender systems. In *DLRS*, 2016.

[10] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

[11] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 2010.

[12] M. Genzel and G. Kutyniok. A mathematical framework for feature selection from real-world data with non-linear observations. *arXiv preprint arXiv:1608.08852*, 2016.

[13] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[15] X. He, M. Gao, M.-Y. Kan, Y. Liu, and K. Sugiyama. Predicting the popularity of web 2.0 items based on user comments. In *SIGIR*, 2014.

[16] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. In *WWW*, 2017.

[17] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*, 2016.

[18] L. Hong, A. S. Doumith, and B. D. Davison. Co-factorization machines: Modeling user interests and predicting individual decisions in twitter. In *WSDM*, 2013.

[19] R. Hong, Y. Yang, M. Wang, and X.-S. Hua. Learning visual semantic relationships for efficient visual retrieval. *IEEE Transactions on Big Data*, 2015.

[20] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[21] Y. Juan, Y. Zhuang, W.-S. Chin, and C.-J. Lin. Field-aware factorization machines for ctr prediction. In *RecSys*, 2016.

[22] Y. Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *KDD*, 2008.

[23] A. Novikov, M. Trofimov, and I. Oseledets. Exponential machines. In *ICLR Workshop*, 2017.

[24] R. J. Oentaryo, E.-P. Lim, J.-W. Low, D. Lo, and M. Finegold. Predicting response in mobile advertising with hierarchical importance-aware factorization machine. In *WSDM*, 2014.

[25] F. Petroni, L. Del Corro, and R. Gemulla. Core: Context-aware open relation extraction with factorization machines. In *EMNLP*, 2015.

[26] R. Qiang, F. Liang, and J. Yang. Exploiting ranking factorization machines for microblog retrieval. In *CIKM*, 2013.

[27] S. Rendle. Factorization machines. In *ICDM*, 2010.

[28] S. Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology*, 2012.

[29] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, 2009.

[30] S. Rendle, Z. Gantner, C. Freudenthaler, and L. Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *SIGIR*, 2011.

[31] Y. Shan, T. R. Hoens, J. Jiao, H. Wang, D. Yu, and J. Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *KDD*, 2016.

[32] F. Shen, Y. Mu, Y. Yang, W. Liu, L. Liu, J. Song, and H. T. Shen. Classification by retrieval: Binarizing data and classifier. In *SIGIR*, 2017.

[33] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

[34] M. Wang, W. Fu, S. Hao, D. Tao, and X. Wu. Scalable semi-supervised learning by efficient anchor graph regularization. *IEEE Transaction on Knowledge and Data Engineering*, 2016.

[35] M. Wang, X. Liu, and X. Wu. Visual classification by l1-hypergraph modeling. *IEEE Transaction on Knowledge and Data Engineering*, 2015.

[36] P. Wang, J. Guo, Y. Lan, J. Xu, S. Wan, and X. Cheng. Learning hierarchical representation model for nextbasket recommendation. In *SIGIR*, 2015.

[37] X. Wang, X. He, L. Nie, and T.-S. Chua. Item silk road: Recommending items from information domains to social users. In *SIGIR*, 2017.

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

[39] C. Xiong, J. Callan, and T.-Y. Liu. Learning to attend and to rank with word-entity duets. In *SIGIR*, 2017.

[40] C. Zhang, G. Zhou, Q. Yuan, H. Zhuang, Y. Zheng, L. Kaplan, S. Wang, and J. Han. Geoburst: Real-time local event detection in geo-tagged tweet streams. In *SIGIR*, 2016.

[41] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, and T.-S. Chua. Discrete collaborative filtering. In *SIGIR*, 2016.

[42] H. Zhang, M. Wang, R. Hong, and T.-S. Chua. Play and rewind: Optimizing binary representations of videos by self-supervised temporal hashing. In *MM*, 2016.

[43] H. Zhang, Z.-J. Zha, Y. Yang, S. Yan, Y. Gao, and T.-S. Chua. Attribute-augmented semantic hierarchy: Towards bridging semantic gap and intention gap in image retrieval. In *MM*, 2013.

[44] W. Zhang, T. Du, and J. Wang. Deep learning over multi-field categorical data. In *ECIR*, 2016.