Adaptive Factorization Network: Learning Adaptive-Order Feature Interactions

Weiyu Cheng, Yanyan Shen[*]**, Linpeng Huang**
Shanghai Jiao Tong University
{weiyu_cheng, shenyy, lphuang}@sjtu.edu.cn

Abstract

Various factorization-based methods have been proposed to leverage second-order, or higher-order cross features for boosting the performance of predictive models. They generally enumerate all the cross features under a predefined maximum order, and then identify useful feature interactions through model training, which suffer from two drawbacks. First, they have to make a trade-off between the expressiveness of higher-order cross features and the computational cost, resulting in suboptimal predictions. Second, enumerating all the cross features, including irrelevant ones, may introduce noisy feature combinations that degrade model performance. In this work, we propose the Adaptive Factorization Network (AFN), a new model that learns arbitrary-order cross features adaptively from data. The core of AFN is a logarithmic transformation layer that converts the power of each feature in a feature combination into the coefficient to be learned. The experimental results on four real datasets demonstrate the superior predictive performance of AFN against the state-of-the-arts.

1 Introduction

Feature engineering is typically recognized as central to successful machine learning tasks, such as recommender systems (Lian et al. 2017), computational advertising (He et al. 2014) and search ranking (Lian and Xie 2016). Except for exploiting raw features, it is usually crucial to find effective transformations of raw features to boost the performance of predictive models. *Cross features* are a major type of feature transformations, where multiplication is performed over sparse raw features to form new features (Cheng et al. 2016). However, handcrafting useful cross features is inevitably expensive and time-consuming, and the results may not generalize to unseen feature interactions. In order to solve this problem, Factorization Machines (FMs) (Rendle 2010; Rendle 2012) are proposed to explicitly model second-order cross features by parameterizing the weight of a cross feature as the inner product of the embedding vectors of the raw features. To be more general, higher-order FMs (HOFMs)

involving higher-order feature combinations were also introduced in the original work (Rendle 2010).

Despite the superior predictive power, there remain two critical questions in FMs/HOFMs to be answered. First, *what is the maximum order of cross features we should consider*? While a larger order enables the modeling of more complex feature interactions and seems to be beneficial, the number of cross features can increase exponentially with the value of the maximum order, resulting in high computational complexity. Some recent works (Blondel et al. 2016) focus on reducing the time complexity of training HOFMs. However, due to the large model size, the time cost of model training and prediction is still high for a large maximum order, which limits the practical usage of higher-order cross features.

Second, *what is the set of useful cross features under the maximum order*? It is important to recognize that not all the features contain useful signals for estimating the target, and different cross features usually have different predictive power. Interactions among irrelevant features can be considered as noises, which have no contribution to the prediction or even degrade model performance. To deal with this problem, Xiao et al. (2017) proposed Attentional Factorization Machines (AFM) to distinguish the importance of factorized interactions by reweighing each cross feature with an attention score (Bahdanau, Cho, and Bengio 2015). The influence of useless cross features can be compromised by assigning lower weights. Nevertheless, applying the attention mechanism over complex feature combinations increases the computational cost significantly. As such, AFM aims at modeling second-order feature interactions only.

In this paper, we argue that existing factorization methods fail to answer the above two questions appropriately. In general, they follow an *enumerating-and-filtering* manner to model feature interactions for prediction. The typical procedure is to predefine the maximum order, enumerate all the cross features within the maximum order, and then filter irrelevant cross features via training. This procedure consists of two major drawbacks. First, predefining a maximum order (which is typically small) restricts model's potential in finding discriminative cross features, because of the trade-off between expressive higher-order cross features and com-

---

putational complexity. Second, considering all the cross features may introduce noises and degrade the prediction performance, since not all the useless cross features can later be filtered out successfully.

To this end, we propose the Adaptive Factorization Network (AFN) to learn arbitrary-order cross features and their weights adaptively from data. The key idea is to encode feature embeddings into a logarithmic space and convert the powers of features into the multiplications with coefficients. The core of AFN is a logarithmic neural transformation layer consisting of multiple vector-wise logarithmic neurons. The purpose of each logarithmic neuron is to automatically learn the powers (i.e., orders) of the features in a possibly useful combination. Upon the logarithmic neural transformation layer, we apply the feedforward neural network to model element-wise feature interactions. Different from FMs/HOFMs, AFN is able to learn useful cross features from data adaptively, and the maximum order can be delivered on the fly. We summarize the major contributions of this paper as follows.

- To the best of our knowledge, we are the first to introduce the logarithmic transformation structure with neural networks to model arbitrary-order feature interactions for prediction.

- Based on the proposed logarithmic transformation layer, we propose the Adaptive Factorization Network (AFN) to learn arbitrary-order cross features and their weights adaptively from data.

- We show that FMs/HOFMs can be interpreted as two specializations of AFN, and the learned orders in AFN allow rescaling feature embeddings in different cross features.

- We conducted extensive experiments on four public datasets. The results demonstrate that the orders of the learned cross features span across a wide range, and our approach achieves superior prediction performance compared with the state-of-the-art methods.

2   Background

Feature Embeddings.   In many real-world predictive tasks such as CTR prediction, input instances consist of both sparse categorical features and numerical features. By tradition, we represent each input instance as a sparse vector:

$$\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m] \tag{1}$$

where m is the number of *feature fields* (e.g., item brand or user age), $\mathbf{x}_i$ is the representation of the i-th feature field (aka a *feature*), and $\mathbf{x}$ is the concatenation of $\mathbf{x}_i$. Since most categorical features are sparse and high-dimensional, a common practice is to map them into dense vectors (i.e., embeddings) in low-dimensional latent space. Specifically, a categorical feature $\mathbf{x}_i$ is initially a one-hot encoded vector, and we have its embedding $\mathbf{e}_i$ computed as follows:

$$\mathbf{e}_i = \mathbf{V}_i \mathbf{x}_i \tag{2}$$

where $\mathbf{V}_i$ denotes the embedding matrix of field i. For a numerical feature $\mathbf{x}_j$, its representation is a scalar x_j. To capture the interactions between numerical and categorical

features, x_j is also transformed into a dense vector in the same low-dimensional space:

$$\mathbf{e}_j = \mathbf{v}_j x_j \tag{3}$$

where $\mathbf{v}_j$ is the embedding vector for the numerical field j. The resultant collection of feature embeddings $\mathbf{e} = \{\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_m\}$ will be used in FMs or neural networks for prediction (Cheng et al. 2018; Li, Shen, and Zhu 2018).

Factorization Machines.   Factorization Machines (Rendle 2010) (FMs) are proposed to explicitly model second-order feature interactions for high-dimensional data. Formally, the prediction of FMs is made as follows:

$$y = \langle \mathbf{w}, \mathbf{x} \rangle + \sum_{j_2 > j_1}^{m} \langle \mathbf{e}_{j_1}, \mathbf{e}_{j_2} \rangle \tag{4}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product operation. Intuitively, the first term $\langle \mathbf{w}, \mathbf{x} \rangle$ is the linear combination of raw features, and the second term is the sum of pair-wise inner products of feature embeddings. Higher-Order Factorization Machines (HOFMs) were introduced to capture higher-order feature interactions for prediction:

$$y = \langle \mathbf{w}, \mathbf{x} \rangle + \sum_{j_2 > j_1}^{m} \langle \mathbf{e}_{j_1}^{(2)}, \mathbf{e}_{j_2}^{(2)} \rangle + \sum_{j_3 > j_2 > j_1}^{m} \langle \mathbf{e}_{j_1}^{(3)}, \mathbf{e}_{j_2}^{(3)}, \mathbf{e}_{j_3}^{(3)} \rangle + \\ \cdots + \sum_{j_n > \cdots > j_1}^{m} \langle \mathbf{e}_{j_1}^{(n)}, \ldots, \mathbf{e}_{j_n}^{(n)} \rangle \tag{5}$$

where the inner product operation is extended to represent the sum of element-wise products of multiple feature embeddings, and n is the maximum order of cross features. Computing Equation (5) directly takes $O(km^n)$ time, where k is the rank of feature embeddings. Due to the high computational complexity, HOFMs have seldom been applied to real predictive systems (Blondel et al. 2016).

A common limitation of FMs and HOFMs is that they model all feature interactions with the same weight. As not all the cross features are useful, incorporating all of them for prediction may introduce noises and degrade model performance. As described above, some efforts have been devoted to alleviating this problem by exploiting attention mechanisms to assign non-uniform weights on different cross features (Xiao et al. 2017), or by learning weights for only retained cross features (Lian et al. 2018). However, these methods introduce additional costs and are still limited to a preset maximum order n of feature interactions before model training. In practice, n is usually set to a small value to make model size moderate. Such a design hinders the opportunity of finding discriminative higher-order cross features. In this paper, we propose to learn arbitrary-order cross features adaptively from data. Both the maximum order and the set of cross features used for prediction will be identified adaptively through model training, leading to high computational efficiency without sacrificing predictive power.

Logarithmic Neural Network (LNN).   Logarithmic Neural Network (Hines 1996) was initially proposed to approximate unbounded non-linear functions. LNN is composed
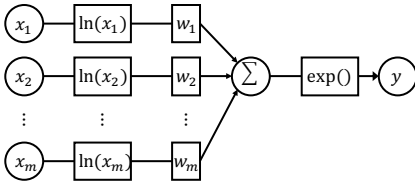
Figure 1: Logarithmic neuron in LNN.

of multiple logarithmic neurons, the structure of which is shown in Figure 1. Formally, a logarithmic neuron can be formulated as:

$$y = \exp(\sum_i w_i \ln x_i) = \prod_i x_i^{w_i} \quad (6)$$

The idea of LNN is to transform input into the logarithmic space, which converts multiplication to addition, division to subtraction, and powers to multiplication by a constant. Although multi-layer perceptrons (MLPs) are known to be universal approximators, they have limited ability in approximating some functions such as multiplication, division, and powers when the input is unbounded (Hines 1996). On the contrary, LNN is able to approximate such functions over the entire input range well.

In this paper, we exploit logarithmic neurons to adaptively learn the powers of each field in cross features from data. We highlight three key differences between LNN and our proposed AFN. 1) The learned power in AFN is applied at a vector-wise level and shared among all feature embeddings in the same field. 2) The inputs to our model are feature embeddings to be learned. Thus we need to use some techniques to keep gradients stable and learn appropriate feature embeddings and combinations for prediction. 3) In AFN, we further apply feed-forward hidden layers upon the learned cross features to enhance the expressiveness of our model.

3 Adaptive Factorization Network

We first elaborate the AFN model that learns adaptive-order feature interactions, including the optimization procedure and its ensemble with deep neural networks. We then make discussions on the learned feature orders in AFN, the model's relation to FMs/HOFMs, and the time complexity. The overall structure of AFN is depicted in Figure 2.

3.1 Model Architecture

Input Layer and Embedding Layer. The input layer of AFN absorbs both sparse categorical features and numerical features. As described in Section 2, all the raw input features are first transformed into embeddings in a shared latent space. Here we introduce two crucial techniques for implementing the embedding layer. First, since we will apply a logarithmic transformation to feature embeddings in the successive layer, we need to keep all the values in the embeddings to be positive. Second, it is advised to add a small positive value ϵ (e.g., 1e-7) to zero embeddings to avoid numerical overflow. After that, the output of the embedding layer is a collection of positive feature embeddings $\mathbf{e} = \{\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_m\}$.
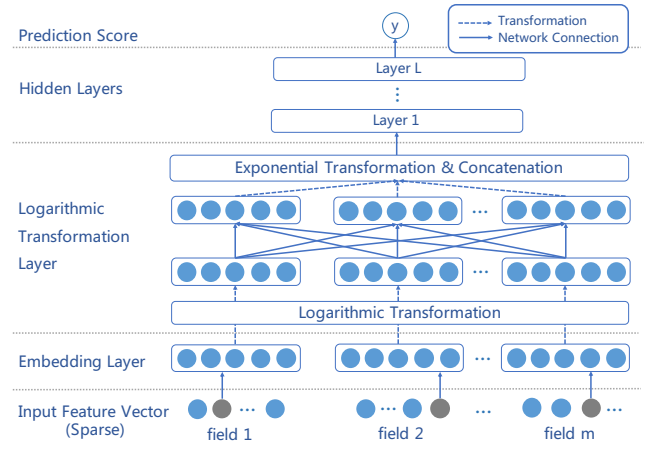

Figure 2: Framework of AFN.

乘法转加法，除法转减法

Logarithmic Transformation Layer The core of AFN is the logarithmic transformation layer which learns the powers (i.e., orders) of each feature field in cross features. This layer consists of multiple vector-wise *logarithmic neurons*. Similar to Equation (6), the output of the j-th vector-wise logarithmic neuron can be formulated as:

$$\mathbf{y}_j = \exp(\sum_{i=1}^m w_{ij} \ln \mathbf{e}_i) = \mathbf{e}_1^{w_{1j}} \odot \mathbf{e}_2^{w_{2j}} \odot ... \odot \mathbf{e}_m^{w_{mj}} \quad (7)$$

where w_{ij} is the coefficient of the j-th neuron on the i-th field. The functions $\ln(\cdot)$ and $\exp(\cdot)$ and the power term w_{ij} are all applied at the element-wise level to the corresponding vectors, and $\odot$ denotes element-wise product operation. The main observation based on Equation (7) is that the output of each logarithmic neuron $\mathbf{y}_j$ is able to represent any cross features. For example, when w_{1j} and w_{2j} are set to 1, and w_{ij} $(2 < i < m)$ are set to 0, we have $\mathbf{y}_j = \mathbf{e}_1 \odot \mathbf{e}_2$, which is a second-order cross feature for the first two raw feature fields. Thus we can use multiple logarithmic neurons to obtain different feature combinations in arbitrary orders as the output of this layer. Note that the elements in the coefficient matrix $\mathbf{W}_{LTL} \in \mathbf{R}^{m \times N}$ (where N denotes the number of logarithmic neurons in the layer) are learnable parameters, and are not necessarily converged to be 0s or 1s.

Feed-forward Hidden Layers and Prediction. Upon the logarithmic transformation layer, we stack several fully-connected hidden layers to combine the formed cross features. We first concatenate all the cross features as input to the feed-forward network:

$$\mathbf{z}_0 = [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N] \quad (8)$$

where N is the number of logarithmic neurons in the preceding layer, and $[\,]$ denotes the concatenation operation. We then feed $\mathbf{z}_0$ into L hidden layers:

$$\mathbf{z}_1 = ReLU(\mathbf{W}_1 \mathbf{z}_0 + \mathbf{b}_1)$$
$$\cdots\cdots \quad (9)$$
$$\mathbf{z}_L = ReLU(\mathbf{W}_L \mathbf{z}_{L-1} + \mathbf{b}_L)$$

where $\mathbf{W}_L$ and $\mathbf{b}_L$ denote the weight matrix and bias vector of the L-th layer, respectively. $ReLU$ is the Rectifier activation function (Nair and Hinton 2010) to capture nonlinear element-wise feature interactions. At last, the output $\mathbf{z}_L$ of the hidden layers is transformed to the final prediction $\hat{y}$:

$$\hat{y} = \mathbf{w}_p^\top \mathbf{z}_L + b_p \tag{10}$$

where $\mathbf{w}_p$ and b_p denote the weight vector and bias term of the prediction layer, respectively.

3.2 Optimization

As AFN enhances FMs from the perspective of learning adaptive-order cross features, it can be applied to a variety of prediction tasks, including classification, regression and ranking, where the objective functions should be chosen accordingly. For binary classification tasks producing 0 or 1 target labels, a common objective function is the logarithmic loss:

$$Logloss = -\frac{1}{K}\sum_{i=1}^{K} y_i \log \sigma(\hat{y}_i) + (1-y_i)\log(1-\sigma(\hat{y}_i)) \tag{11}$$

where K is the total number of training instances, and σ denotes the sigmoid function. For regression tasks, we can minimize the mean squared error loss. In this work, we focus on binary classification tasks and optimize the log loss in Equation (11). We employ the Adam optimizer (Kingma and Ba 2015), a variant of Stochastic Gradient Descent that dynamically tunes the learning rate during the training process, thus leading to faster convergence (Sun et al. 2019).

Besides, we perform batch normalization (BN) (Ioffe and Szegedy 2015) on the outputs of logarithmic transformation, exponential transformation, and all the hidden layers with two considerations. First, the feature embeddings $\mathbf{e}$ are usually initialized and optimized to be close to zero. After the logarithmic transformation, the embeddings tend to involve large negative values with a significant variance, which is harmful to the optimization of the parameters in the successive layers. As BN can scale and shift the outputs to normalized values, it is crucial to the training process of AFN. Second, we employ multi-layered neural networks after the logarithmic transformation layer. Performing BN on the outputs of hidden layers helps alleviate the covariance shift problem (Ioffe and Szegedy 2015), leading to faster convergence and better model performance empirically.

3.3 Ensemble AFN with DNNs

Prior works (Cheng et al. 2016; Guo et al. 2017; Lian et al. 2018) have proposed to ensemble the prediction results from cross feature-based models such as FMs with those from raw feature-based neural methods to boost the performance. As AFN adaptively learns cross features for prediction, we can combine it with deep neural networks (DNNs) in a similar manner. Note that neural structures are powerful to capture nonlinear and deep feature interactions at the element-wise level, which can be viewed as a fine-grained feature interactions modeling method to enhance AFN. To enforce the ensemble between AFN and the neural networks, we first train the two models separately. After that, we develop an ensemble model to combine the prediction results from the two trained models, as below:

$$\hat{y}_{ensem} = w_1\hat{y}_{AFN} + w_2\hat{y}_{DNN} + b \tag{12}$$

where $\hat{y}_{AFN}$ and $\hat{y}_{DNN}$ are the predictions made from the trained AFN and DNN respectively, w_1 and w_2 are the corresponding coefficients, and b is a bias term. The ensemble model can be trained by optimizing the log loss similar to Equation (11). We dub this ensembled model as "AFN$^+$". Our ensemble method is a bit different from the one used by DeepFM (Guo et al. 2017) whose feature embeddings are shared among FM and DNN. Here we separate the embedding layers of AFN and DNN to avoid interference. Our primary concern is that unlike DeepFM, the distribution of the embedding values in AFN, which should always keep positive, is far from that in DNN. The separation slightly increases the model complexity but leads to better performance according to our experiments.

3.4 Discussions

Understanding orders in AFN. AFN learns the power (i.e., the orders) of each feature in cross features though the logarithmic transformation layer. As no restriction is enforced on the weight matrix $\mathbf{W}_{LTL}$ of the logarithmic transformation layer, the learned feature orders can be decimals or negative values. In order to understand the feature orders learned by AFN, we borrow some ideas from the field-aware factorization machines (FFMs) (Juan et al. 2016). In FFMs, each feature is associated with m feature embeddings, where m is the number of feature fields. FFMs is distinct from FMs in that each feature employs different embeddings when interacting with features from different fields. The insight of FFMs is to avoid interference among the feature space of different fields. In AFN, the order of each feature can be considered as a scaling factor for the corresponding feature embedding. For example, consider an embedding with values ranging from 0 to 1. An order larger than 1 would shrink embedding values while an order smaller than 1 would do the opposite. By analogy to FFMs, the orders learned by AFN can be utilized to rescale feature embeddings when interacting with other features in different fields.

Relation to FMs and HOFMs. We first show that FMs can be viewed as a special case of AFN. According to Equation (7), the output $\mathbf{y}_j$ from a logarithmic neuron is able to represent any second-order cross features by setting the power w_{ij} of each feature embedding appropriately. Suppose we have enough logarithmic neurons to produce all the second-order cross features, and the successive hidden layers approximate a simple summation function at the element-level. Then AFN can exactly recover FMs. Similarly, when we have enough logarithmic neurons to deliver all the cross features within the maximum order and allow the hidden layers to approximate a summation function, AFN is able to recover HOFMs. Note that in HOFMs (Blondel et al. 2016), feature embeddings in different orders can be either shared or learned separately.

Time complexity analysis. Recall that we use k and m to denote the rank of feature embeddings and the number

Table 1: Statistics of the datasets.

Dataset	#instances	#fields	#features
Criteo	45,840,617	39	2,086,936
Avazu	40,428,967	22	1,544,250
Movielens	2,006,859	3	90,445
Frappe	288,609	10	5,382

of feature fields, respectively. In AFN, a logarithmic neuron in Equation (7) can be computed in $O(km)$ time. Assume we use N logarithmic neurons to obtain cross features. The computational complexity of the logarithmic transformation layer is $O(kmN)$. Considering the additional cost in the hidden layers, the total time complexity of AFN is $O(kmN + n_W)$, where n_W is the total number of weights in the hidden layers. As for HOFMs, supposing n is the maximum order of feature combinations as predefined, it takes $O(km^n)$ time to deliver a prediction which can be reduced to $O(kmn^2)$ with dynamic programming (Blondel et al. 2016). Note that the time complexity of HOFMs is highly correlated with the maximum order n of cross features, while in AFN, both N and n_W are only determined by the model structure due to its adaptive cross features generation manner. The time cost of training AFN with the optimal setting is empirically close to that of CIN (Lian et al. 2018).

4 Experiments

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

RQ1: How do our proposed methods AFN and AFN$^+$ perform against the state-of-the-art methods?

RQ2: How does the performance of AFN vary with different settings of the hyper-parameters?

RQ3: What are the learned feature orders in AFN, and can AFN find useful cross features from data?

4.1 Experimental Settings

Datasets. We conduct experiments with four publicly accessible datasets following previous works (Lian et al. 2018; He and Chua 2017): Criteo[1], Avazu[2], Movielens[3] and Frappe[4]. For each dataset, we randomly split the instances by 8:1:1 for training, validation and test, respectively. The details of the four datasets are summarized in Table 1.

(1) **Criteo**: This is a popular industry benchmarking dataset for CTR prediction, which contains 13 numerical feature fields and 26 categorical feature fields.

(2) **Avazu**: This dataset contains users' click records on mobile advertisements. It has 22 feature fields including user features and advertisement attributes.

(3) **Movielens**: This dataset consists of users' tagging records on movies. We focus on personalized tag recommendation by converting each tagging record (user ID, movie

ID, tag) to a feature vector as input. The target value denotes whether the user has assigned a particular tag to the movie.

(4) **Frappe**: This dataset contains app usage logs from users under different contexts (e.g., daytime, location). We converted each log (user ID, app ID, context features) to a feature vector as input. The target value indicates whether the user has used the app under the context.

Evaluation metrics. We adopt two metrics for performance evaluation: AUC (Area Under the ROC curve) and Logloss (cross entropy). Note that a slight increase in AUC or decrease in Logloss at .001-level is known to be a significant improvement for the tasks such as CTR prediction (Cheng et al. 2016; Guo et al. 2017; Song et al. 2018).

Comparison methods. We compare AFN and AFN$^+$ with four classes of the existing approaches: (i) first-order approaches that linearly sum up raw features; (ii) FM-based methods that consider second-order cross features; (iii) advanced approaches that model high-order feature interactions; (iv) ensemble models that involve a DNN as the counterpart. We briefly describe these methods as follows.

• **Linear Regression (LR)**. It sums up raw features linearly.

• **Wide&Deep** (Cheng et al. 2016). It integrates LR with DNN. Note that we omit the hand-crafted cross features for a fair comparison.

• **FM** (Rendle 2012). FM models second-order cross features with factorization techniques for prediction.

• **HOFM** (Blondel et al. 2016). It is the high-order version of FM.

• **DeepFM** (Guo et al. 2017). It is an ensemble between DNN and FM.

• **AFM** (Xiao et al. 2017). It extends FM via the attention mechanism to distinguish the importance of second-order cross features.

• **CrossNet** (Wang et al. 2017). It explicitly models feature interactions by taking the outer product of input feature vectors.

• **Deep&Cross** (Wang et al. 2017). It is the ensemble between CrossNet and DNN.

• **NFM** (He and Chua 2017). It sums up pairwise Hadamard product of input feature vectors followed by fully connected layers.

• **PNN** (Qu et al. 2019). It models feature interactions by concatenating pairwise inner or outer products of input feature vectors.

• **CIN** (Lian et al. 2018). It produces high-order cross features by computing outer products of feature vectors at different orders.

• **xDeepFM** (Lian et al. 2018). It integrates CIN with DNN.

Implementation details. We implement our methods using Tensorflow[5]. We apply Adam with a learning rate of 0.001 and a mini-batch size of 4096. The default number of logarithmic neurons is set to 1500, 1200, 800 and 600 for Criteo, Avazu, Movielens and Frappe datasets, respectively. We use 3 hidden layers and 400 neurons per layer by default in AFN. To avoid overfitting, we perform early-stopping according to the AUC on the validation set. We set the rank of feature embeddings to 10 in all the models. We use the same
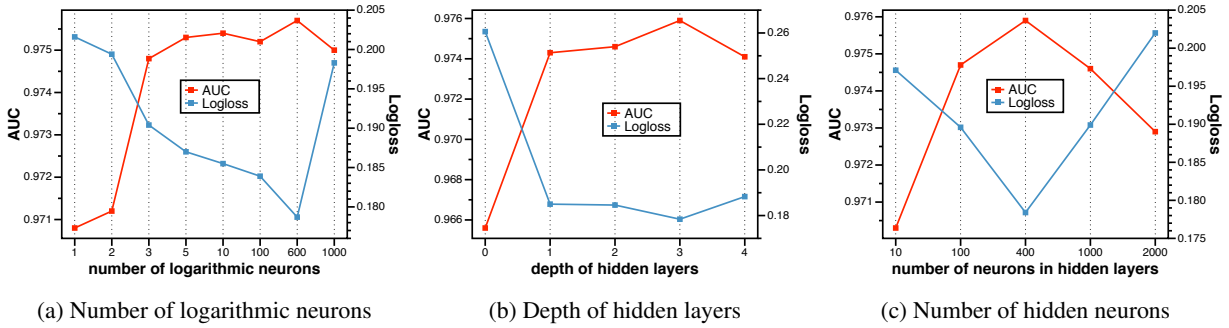
---

[1]http://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/

[2]https://www.kaggle.com/c/avazu-ctr-prediction

[3]https://grouplens.org/datasets/movielens/

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

[5]https://github.com/WeiyuCheng/AFN-AAAI-20

Table 2: Performance comparison.

Model Class	Model	Criteo AUC	Criteo Logloss	Avazu AUC	Avazu Logloss	Movielens AUC	Movielens Logloss	Frappe AUC	Frappe Logloss
First-Order	LR	0.7858	0.4636	0.7313	0.4065	0.9215	0.3080	0.9329	0.2860
Second-Order	FM	0.7933	0.4574	0.7496	0.3740	0.9388	0.2797	0.9641	0.2143
	AFM	0.7953	0.4554	0.7454	0.3766	0.9295	0.2836	0.9639	0.2294
High-Order	CrossNet	0.7915	0.4585	0.7498	0.3756	0.9323	0.2929	0.9393	0.2835
	HOFM	0.7960	0.4551	0.7516	0.3756	0.9410	0.3088	0.9709	0.2141
	NFM	0.7968	0.4537	0.7531	0.3761	0.9441	0.3004	0.9727	0.2079
	PNN	0.8026	0.4509	0.7526	0.3737	0.9469	0.2792	0.9735	0.2012
	CIN	0.8042	0.4472	**0.7533**	0.3756	**0.9494**	**0.2600**	0.9704	0.2342
	AFN	**0.8061**	**0.4458**	0.7512	**0.3731**	0.9477	0.2753	**0.9759**	**0.1784**
Ensembled	Deep&Cross	0.8059	0.4463	0.7550	0.3721	0.9419	0.2791	0.9402	0.2808
	Wide&Deep	0.8062	0.4453	0.7529	0.3744	0.9381	0.3310	0.9728	0.2038
	DeepFM	0.8025	0.4501	0.7535	0.3742	0.9424	0.3131	0.9719	0.2108
	xDeepFM	0.8070	**0.4443**	0.7535	0.3737	0.9448	0.2717	0.9738	0.2098
	AFN$^+$	**0.8074**	0.4451	**0.7555**	**0.3718**	**0.9500**	**0.2585**	**0.9783**	**0.1762**

(a) Number of logarithmic neurons   (b) Depth of hidden layers   (c) Number of hidden neurons

Figure 3: Effects of hyperparameters on the performance of AFN.

neural network structure (i.e., 3 layers, 400-400-400) for all the approaches that involve DNN for a fair comparison. The maximum order in HOFM is set to 3. All the other hyperparameters are tuned on the validation set. For each empirical result, we run the experiments for 3 times and report the average value.

4.2 Performance Comparison (RQ1)

Comparing with individual models. We first compare AFN with various individual models involving first-order, second-order and high-order feature interactions. The results are shown in Table 2. We have three important observations. First, AFN yields the best or competing performance over all the datasets. For Criteo and Frappe, AFN outperforms the second-best model CIN by a large margin, i.e., on average, the increase on AUC and the decrease on log loss are 0.0037 and 0.0286, respectively. For Movielens, AFN achieves the second-best performance, and for Avazu, it achieves the best log loss with a moderate AUC. Regarding the good performance of simpler models on Movielens and Avazu, we conjecture that the predictions on these two datasets rely more on lower-order cross features, and the advantages of AFN are thus restricted. Note that Movielens only contains three feature fields, and the benefit of find-

ing useful higher-order cross features can be marginal. Second, AFN consistently outperforms FMs and HOFMs on all the datasets, which verifies that learning adaptive-order cross features can bring better predictive performance than modeling fixed-order feature interactions. Third, the models that utilize higher-order feature interactions generally outperform those based on lower-order cross features, especially when the number of feature fields is large. This is consistent with the intuition that higher-order feature interactions have stronger predictive power.

Comparing with integrated models. AFN$^+$ integrates AFN and DNN to exploit both explicit cross features and implicit element-wise feature interactions for prediction. We compare AFN$^+$ with several state-of-the-art ensemble models. As shown in Table 2, AFN$^+$ achieves the best performance on the four datasets. On average, AFN outperforms xDeepFM, which integrates CIN and DNN, by achieving 0.003 and 0.012 improvements on AUC and log loss, respectively. This demonstrates that the adaptive-order cross features learned by AFN are quite different from the implicit feature interactions modeled by DNN, thus improving the performance gain significantly when combining two different types of feature interactions for prediction.

(a) Individual feature order

(b) Cross feature order

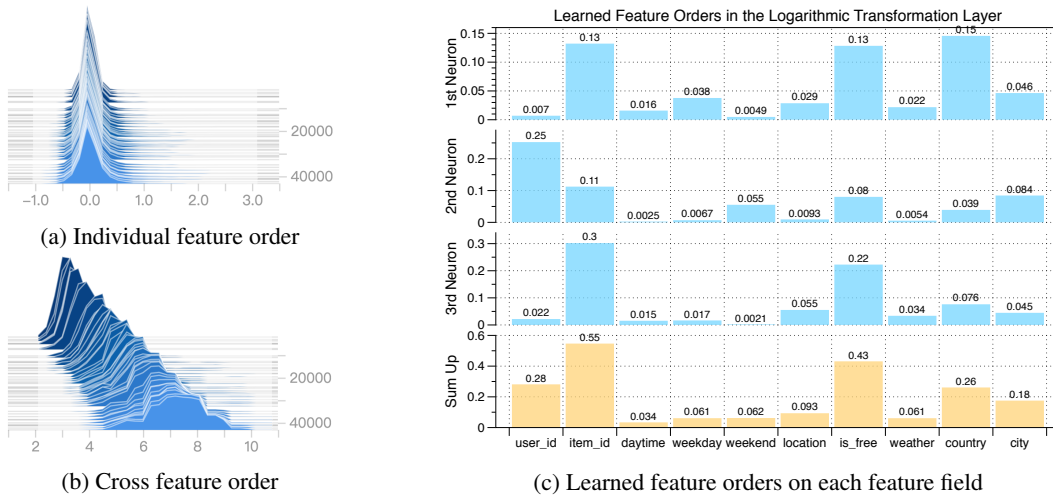(c) Learned feature orders on each feature field

Figure 4: Quantitative analysis and case study. (a)&(b): Distribution of the learned feature orders (x-axis) over training steps (y-axis) on Criteo dataset. (c) A case study on Frappe dataset, where we set the number of logarithmic neurons to 3.

4.3 Hyperparameter Investigation (RQ2)

We now study the effect of three hyperparameters on the performance of AFN. We only provide the results on Frappe as the results on the other three datasets are similar.

Number of logarithmic neurons. Figure 3a provides the results over different numbers of logarithmic neurons in the logarithmic transformation layer. We can see that the performance of AFN shows an increasing trend, followed by a decreasing trend when the number of neurons becomes larger. This indicates that an appropriate number of logarithmic neurons should be employed to make a trade-off between expressiveness and generalization to achieve optimal performance. Surprisingly, the advantage of AFN is stable even when the number of logarithmic neurons is less than 5. This result demonstrates that finding a small number of discriminative cross features is vital to the prediction accuracy and AFN is effective to find these critical cross features.

Depth of hidden layers. Figure 3b shows the effect of the depth of hidden layers. We observe that stacking hidden layers upon the learned adaptive-order cross features is beneficial in improving the model performance. However, it is worth noticing that the performance of AFN is not highly dependent on the number of hidden layers. When the depth is set to 0, and the prediction is made by a weighted sum over the learned cross features, AFN can still achieve fairly good results. This demonstrates the effectiveness of the logarithmic transformation layer in learning discriminative cross features.

Number of neurons in hidden layers. As shown in Figure 3c, the performance of AFN first grows with the number of neurons. This is because more parameters bring better expressiveness to the model. The performance starts to degrade when the parameter size of the hidden layers exceeds 600, which is caused by overfitting as the training loss keeps on decreasing afterward.

4.4 Quantitative Analysis and Case Study (RQ3)

Learned feature orders. We now investigate the learned feature orders in the logarithmic transformation layer of AFN. Figure 4 shows the variation of feature orders during the whole training procedure on Criteo Dataset. From Figure 4a, we can see that the orders of individual feature field are typically centered around zero and within the range of $[-1, 1]$. This is quite different from the typical factorization-based methods where individual feature orders are either 0 or 1. The relaxation in the learned feature orders allows the original feature embeddings to be rescaled when composing different cross features. We also provide the order distribution of cross features in Figure 4b, where the order of a cross feature is computed by the sum of absolute values of the constituent features orders. We can see that the learned cross feature orders are gradually optimized during the training process. The final cross feature orders spread over a wide range (from 4 to 10), instead of being fixed to a predefined value (e.g., 2) as in much of the factorization-based work.

Case study. To get a deeper understanding of the cross features learned by AFN, we conduct a case study on Frappe Dataset, where the description of each feature field is available. For illustration purpose, we limit the number of logarithmic neurons to be 3. Figure 4c provides the absolute values of individual feature orders on each neuron and the summation. From the figure, we can approximately infer that three cross features *(item_id, is_free, country)*, *(user_id, item_id)*, and *(item_id, is_free)* are learned in the respective logarithmic neurons. Moreover, by summing up feature orders in three neurons, the most discriminative feature fields are found to be *item_id*, *is_free*, and *user_id*. This is reasonable because user and item identities are the most commonly used features in collaborative filtering, and *is_free*, denoting whether a user has paid for a mobile app, is a strong indicator of users' preferences towards apps.

5 Conclusion

In this paper, we introduced the Adaptive Factorization Network (AFN), which learns arbitrary-order feature interactions adaptively from data. Instead of explicitly modeling all the cross features within a fixed maximum order, AFN is able to generate discriminative cross features and the weights of the corresponding features automatically. The key idea is to transform feature embeddings into a logarithmic space and treat the power of each feature in a feature combination as the coefficient to be learned. Feedforward neural networks are further applied to combine the learned cross features for prediction. We also showed that AFN can generalize FMs and HOFMs with computational efficiency. Extensive experiments on four real-world datasets demonstrate the superior predictive performance of AFN compared with the state-of-the-art methods.

Acknowledgments

References

[Bahdanau, Cho, and Bengio 2015] Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR'15, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[Blondel et al. 2016] Blondel, M.; Fujino, A.; Ueda, N.; and Ishihata, M. 2016. Higher-order factorization machines. In *NIPS'16, December 5-10, 2016, Barcelona, Spain*, 3351–3359.

[Cheng et al. 2016] Cheng, H.; Koc, L.; Harmsen, J.; Shaked, T.; Chandra, T.; Aradhye, H.; Anderson, G.; Corrado, G.; Chai, W.; Ispir, M.; Anil, R.; Haque, Z.; Hong, L.; Jain, V.; Liu, X.; and Shah, H. 2016. Wide & deep learning for recommender systems. In *DLRS@RecSys'16, Boston, MA, USA, September 15, 2016*, 7–10.

[Cheng et al. 2018] Cheng, W.; Shen, Y.; Zhu, Y.; and Huang, L. 2018. DELF: A dual-embedding based deep latent factor model for recommendation. In *IJCAI'18, July 13-19, 2018, Stockholm, Sweden*, 3329–3335.

[Guo et al. 2017] Guo, H.; Tang, R.; Ye, Y.; Li, Z.; and He, X. 2017. Deepfm: A factorization-machine based neural network for CTR prediction. In *IJCAI'17, Melbourne, Australia, August 19-25, 2017*, 1725–1731.

[He and Chua 2017] He, X., and Chua, T. 2017. Neural factorization machines for sparse predictive analytics. In *SIGIR'17, Shinjuku, Tokyo, Japan, August 7-11, 2017*, 355–364.

[He et al. 2014] He, X.; Pan, J.; Jin, O.; Xu, T.; Liu, B.; Xu, T.; Shi, Y.; Atallah, A.; Herbrich, R.; Bowers, S.; and Candela, J. Q. 2014. Practical lessons from predicting clicks on ads at facebook. In *ADKDD'14, August 24, 2014, New York City, New York, USA*, 5:1–5:9.

[Hines 1996] Hines, J. W. 1996. A logarithmic neural network architecture for unbounded non-linear function approximation. In *ICNN'96*, volume 2, 1245–1250 vol.2.

[Ioffe and Szegedy 2015] Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML'15, Lille, France, 6-11 July 2015*, 448–456.

[Juan et al. 2016] Juan, Y.; Zhuang, Y.; Chin, W.; and Lin, C. 2016. Field-aware factorization machines for CTR prediction. In *RecSys'16, MA, USA, September 15-19, 2016*, 43–50.

[Kingma and Ba 2015] Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. In *ICLR'15, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[Li, Shen, and Zhu 2018] Li, R.; Shen, Y.; and Zhu, Y. 2018. Next point-of-interest recommendation with temporal and multi-level context attention. In *ICDM'18, Singapore, November 17-20, 2018*, 1110–1115.

[Lian and Xie 2016] Lian, J., and Xie, X. 2016. Cross-device user matching based on massive browse logs: The runner-up solution for the 2016 cikm cup. *arXiv preprint arXiv:1610.03928*.

[Lian et al. 2017] Lian, J.; Zhang, F.; Hou, M.; Wang, H.; Xie, X.; and Sun, G. 2017. Practical lessons for job recommendations in the cold-start scenario. In *Proceedings of the Recommender Systems Challenge 2017*, RecSys Challenge '17, 4:1–4:6. New York, NY, USA: ACM.

[Lian et al. 2018] Lian, J.; Zhou, X.; Zhang, F.; Chen, Z.; Xie, X.; and Sun, G. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *KDD'18, London, UK, August 19-23, 2018*, 1754–1763.

[Nair and Hinton 2010] Nair, V., and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML'10, June 21-24, 2010, Haifa, Israel*, 807–814.

[Qu et al. 2019] Qu, Y.; Fang, B.; Zhang, W.; Tang, R.; Niu, M.; Guo, H.; Yu, Y.; and He, X. 2019. Product-based neural networks for user response prediction over multi-field categorical data. *ACM Trans. Inf. Syst.* 37(1):5:1–5:35.

[Rendle 2010] Rendle, S. 2010. Factorization machines. In *ICDM'10, Sydney, Australia, 14-17 December 2010*, 995–1000.

[Rendle 2012] Rendle, S. 2012. Factorization machines with libfm. *ACM TIST* 3(3):57:1–57:22.

[Song et al. 2018] Song, W.; Shi, C.; Xiao, Z.; Duan, Z.; Xu, Y.; Zhang, M.; and Tang, J. 2018. Autoint: Automatic feature interaction learning via self-attentive neural networks. *arXiv preprint arXiv:1810.11921*.

[Sun et al. 2019] Sun, H.; Shao, Y.; Jiang, J.; Cui, B.; Lei, K.; Xu, Y.; and Wang, J. 2019. Sparse gradient compression for distributed SGD. In *DASFAA'19, Chiang Mai, Thailand, April 22-25, 2019, Proceedings, Part II*, 139–155.

[Wang et al. 2017] Wang, R.; Fu, B.; Fu, G.; and Wang, M. 2017. Deep & cross network for ad click predictions. In *AD-KDD'17, Halifax, NS, Canada, August 13 - 17, 2017*, 12:1–12:7.

[Xiao et al. 2017] Xiao, J.; Ye, H.; He, X.; Zhang, H.; Wu, F.; and Chua, T. 2017. Attentional factorization machines: Learning the weight of feature interactions via attention networks. In *IJCAI'17, Melbourne, Australia, August 19-25, 2017*, 3119–3125.