Field-aware Probabilistic Embedding Neural Network for CTR Prediction

Weiwen Liu[1], Ruiming Tang[2], Jiajin Li[1], Jinkai Yu[2], Huifeng Guo[3], Xiuqiang He[4], Shengyu Zhang[1,5]
[1] The Chinese University of Hong Kong, Hong Kong
[2] Noah's Ark Lab, Huawei, Shenzhen, China
[3] Shenzhen Graduate School, Harbin Institute of Technology, Shenzhen, China
[4] Data service center, MIG, Tencent, Shenzhen, China
[5] Tencent Quantum Lab, Tencent, Shenzhen, China
{wwliu,jjli,syzhang}@cse.cuhk.edu.hk,{tangruiming,yujinkai}@huawei.com
huifengguo@yeah.net,xiuqianghe@tencent.com

ABSTRACT

For Click-Through Rate (CTR) prediction, Field-aware Factorization Machines (FFM) have exhibited great effectiveness by considering field information. However, it is also observed that FFM suffers from the overfitting problem in many practical scenarios. In this paper, we propose a Field-aware Probabilistic Embedding Neural Network (FPENN) model with both good generalization ability and high accuracy. FPENN estimates the probability distribution of the field-aware embedding rather than using the single point estimation (the maximum a posteriori estimation) to prevent overfitting. Both low-order and high-order feature interactions are considered to improve the accuracy. FPENN consists of three components, i.e., FPE component, Quadratic component and Deep component. **FPE component** outputs probabilistic embedding to the other two components, where various confidence levels for feature embeddings are incorporated to enhance the robustness and the accuracy. **Quadratic component** is designed for extracting low-order feature interactions, while **Deep component** aims at capturing high-order feature interactions. Experiments are conducted on two benchmark datasets, Avazu and Criteo. The results confirm that our model alleviates the overfitting problem while having a higher accuracy.

CCS CONCEPTS

• **Information systems → Recommender systems**;

KEYWORDS

CTR Prediction; Recommender Systems; Deep Neural Network

This work is done when Huifeng Guo was an intern at Noah's Ark Lab, Huawei.
This work is done when Xiuqiang He worked at Noah's Ark Lab, Huawei.

RecSys '18, October 2–7, 2018, Vancouver, BC, Canada
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5901-6/18/10…$15.00
https://doi.org/10.1145/3240323.3240396

1 INTRODUCTION

The Click-Through Rate (CTR) measures the ratio of clicks to impressions, the prediction of which is a vital task for web search, recommender systems, online advertising, etc. For example, in advertisement, if the CTR is accurately estimated, the recommender system is able to show the user the ads that they prefer to click on. Displaying right ads for users has a direct impact on the revenue as well as user satisfaction.

A key challenge in CTR prediction is feature selection and combination. Shallow models like Logistic Regression (LR) [14] and Factorization Machines (FM) [13], learn a generalized linear function of features, or second-order feature interactions. Though shallow models are generally easy to implement and have low computational cost, they have limited expressive power and lack the ability to extract higher-order feature interactions. Recently, due to the powerful ability of feature representation, different kinds of deep learning techniques are applied to CTR prediction [4, 10, 12, 16–19]. Deep models, however, are biased to high-order interactions and fail to exploit the simple yet effective low-order feature interactions. Several models, e.g., Wide & Deep [3] and DeepFM [6], are proposed to capture both high-order and low-order feature interactions.

All the models mentioned above ignore the underlying *field* information of the features, which has been proved to be effective in CTR prediction, as demonstrated in Field-aware Factorization Machine (FFM) [7]. FFM, as a member of wide models, extends the idea of FM and promotes the concept of the field for CTR prediction, outperforming other models in some world-wide CTR prediction competitions hosted by Avazu and Criteo in 2016 [7]. The basic idea of FFM is that features can be semantically grouped into several fields and the representation of a feature should be different when it interacts with different fields of features.

Nevertheless, a major drawback of FFM is that it is prone to overfitting [7]. It is also observed in our experiments in Section 3: after a certain number of epochs, the performance of FFM becomes significantly weaker on the unseen data as it begins to fit the noise of the training data. As a result, it is difficult to pin down the optimal stopping time, and a slight delay of the stopping time can cause a dramatic decline in prediction accuracy.

To tackle this problem, we present a novel Field-aware Probabilistic Embedding method (FPE). Instead of using the single point estimation (the maximum a posteriori estimation), we propose to estimate a probability distribution of each element of the embedding. For each feature, we embed it into several random latent vectors,

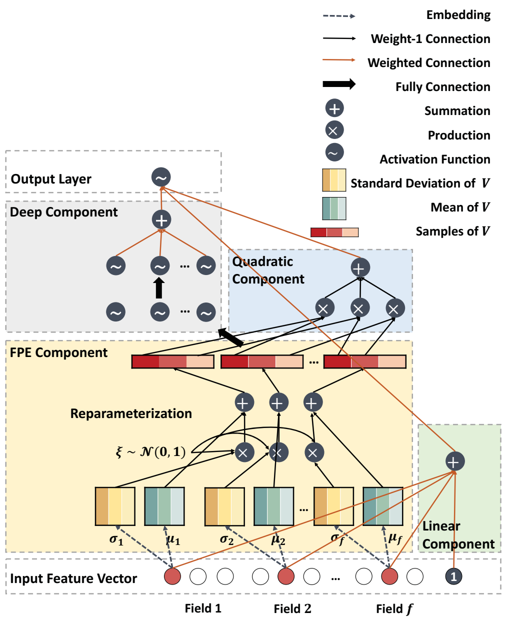


Figure 1: Network Structure of FPENN. The output of FPENN is a combination of three components, i.e., the deep component, the quadratic component and the linear component.

one for interacting with a specific field. Compared to the previous deterministic methods, the probabilistic embedding methods provide prior knowledge on the solution to prevent overfitting, which can be regarded as adding regularization [11]. Moreover, both the first and the second moments of the distribution (i.e., the mean and the variance) are taken into consideration in our probabilistic embedding. If the variance is large, it indicates that the current approximation of the mean is unreliable. To handle the unreliable mean value, we propose to use TS-strategy and UCB-strategy to correct the estimation, balancing between the mean and variance to improve the accuracy. Therefore, our model can not only enhance the robustness of the prediction, but also improve the accuracy.

2 MODEL

2.1 Problem Formulation

Suppose we are given a data set of m instances $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^d$ is a vector of d categorical or continuous features of the i-th given instance, and $y_i \in \{0, 1\}$ is the label indicating whether the user clicked on the item ($y_i = 1$) or not ($y_i = 0$). Categorical features are encoded by one-hot vectors, and continuous features are represented either as the value itself or one-hot vectors after discretization. The CTR prediction task can be formulated as to predict the probability that $y_i = 1$, namely

$$\hat{y} = \mathbb{P}(y_i = 1).$$

2.2 Architecture of FPENN

We aim at an effective and robust CTR prediction model. Our proposed FPENN is a fusion of the linear term ϕ_{LN}, quadratic term ϕ_{QDR} and deep neural net term ϕ_{DNN}, which is defined by

$$\hat{y} = \frac{1}{1 + e^{-z}},$$

where

$$z = w_{\text{LN}} \cdot \phi_{\text{LN}} + w_{\text{QDR}} \cdot \phi_{\text{QDR}} + w_{\text{DNN}} \cdot \phi_{\text{DNN}}.$$

As displayed in Figure 1[1], given a large and sparse input feature vector, the Field-aware Probabilistic Embedding (FPE) component embeds each feature to dense latent vectors. The quadratic (QDR) component and deep (DNN) component share the same input latent vectors given by the FPE component, elaborated as follows.

2.2.1 FPE Component. The Field-aware Probabilistic Embedding (FPE) component is an embedding layer that embeds the sparse input feature vector to dense latent vectors to reduce the dimension. Randomness is introduced to the embedding to enhance the robustness and effectiveness of the prediction. A distribution is learned for each element in the latent vectors. Suppose the features in $\mathbf{x} \in \mathbb{R}^d$ can be grouped into f different fields. For the i-th ($i = 1, \ldots, d$) feature, we embed it into a matrix $\mathbf{V}_i \in \mathbb{R}^{(f-1)\times k}$, where k is the dimension of the latent vector. One latent vector is learned for each of the other field, so there are $f - 1$ latent vectors for each feature. The outcome of our FPE embedding is a tensor $\mathbf{V}$ of dimension $d \times (f - 1) \times k$. Note that the training and the testing procedure are slightly different for FPE component. The probability distribution is learned in the training procedure, while the mean and the variance information of the distribution are combined together to make predictions in the testing procedure.

Training: Unlike deterministic models (which models each feature as a latent vector), our FPE component treats each element of the embedding vector as a distribution (or more specifically, the confidence level of each element is considered). Probability distribution provides prior knowledge of the model to alleviate the overfitting problem, while various confidence levels are utilized to make more accurate predictions.

However, it brings a difficulty for the training of the neural network, because the gradients cannot be back-propagated through the sampling procedure. We circumvent this issue by a *reparameterization trick* [9, 15]: decompose the continuous random variable into two parts, i.e., the deterministic term and the base distribution (e.g., uniform or standard normal distribution). Specifically, in our case, assume that $\mathbf{V}$ follows a diagonal Gaussian distribution, i.e., all the dimensions in $\mathbf{V}$ are independent. Denote vectorization of $\mathbf{V}$ by vec($\mathbf{V}$):

$$\text{vec}(\mathbf{V}) \sim \mathcal{N}(\mu, \text{diag}(\sigma)^2),$$

where $\mu \in \mathbb{R}^{d(f-1)k\times 1}$ is the vector of mean value and $\sigma \in \mathbb{R}^{d(f-1)k\times 1}$ is the vector of standard deviation. The reparameterization can be written as

$$\text{vec}(\mathbf{V})_i = \mu_i + \xi_i * \sigma_i, \quad \xi_i \sim \mathcal{N}(0, 1). \tag{1}$$

After sampling ξ, $\mathbf{V}$ becomes deterministic, and the parameters μ and σ can be learned by the back propagation.

[1]In Figure 1, *Embedding* means latent vectors to be learned; *Weight-1 Connection* means a connection with weight 1; *Fully Connection* means the layer is fully connected; *Summation* means adding all the inputs; *Production* is the inner product of two latent vectors; The white and red circles indicate zero and one in one-hot encoding of the input respectively and each rectangle represents a latent vector.

The training procedure of FPE component is illustrated in Figure 1. Given the input feature vector of f fields, each feature is embedded into two matrices (i.e., *mean of* $\mathbf{V}$ and *standard deviation of* $\mathbf{V}$), both of dimension $(f-1)\times k$. After that, the *samples of* $\mathbf{V}$ (probabilistic embedding) is generated following Eq.(1), preparing for the QDR component and the DNN component.

Testing: The mean and variance information are combined in our testing procedure. The learned variance can be regarded as an indicator of how confident we are in the empirical mean. For example, a high variance indicates a low confidence level of the mean value. To utilize the variance information and avoid the overfitting problem, we propose two strategies obtain the latent vectors $\hat{\mathbf{V}}$, i.e., TS-strategy and UCB-strategy.

TS-strategy: Thompson Sampling (TS) algorithm [2] makes decisions based on parameters drawn from the posterior distribution. Inspired by TS, we use the samples drawn from the distribution learned in the training procedure. This strategy outputs $\hat{\mathbf{V}}$:

$$\mathrm{vec}(\hat{\mathbf{V}})_i \sim \mathcal{N}(\hat{\mu}_i, \hat{\sigma}_i),$$

where $\hat{\mu}_i$ and $\hat{\sigma}_i$ are our estimated mean and variance obtained from the training procedure.

UCB-strategy: Since the empirical mean value on the training set is just an estimator of the real mean, a small error is inevitable. Instead of using the empirical mean, we use the Upper Confidence Bound (UCB), which is likely to avoid the suboptimal value by balancing the mean and the variance [1]:

$$\mathrm{vec}(\hat{\mathbf{V}})_i = \hat{\mu}_i + \alpha\hat{\sigma}_i,$$

where α is a hyperparameter (when $\alpha = 0$, it is simplified to the MAP estimate).

2.2.2 Linear Component. The linear component is simply defined by

$$\phi_{\mathrm{LN}}(\mathbf{w}, \mathbf{x}) = \mathbf{w}^\top \mathbf{x},$$

where $\mathbf{x}$ is the input feature vector and $\mathbf{w}$ is the parameter to be learned in the training procedure.

2.2.3 Quadratic Component. The quadratic (QDR) component captures the second-order feature interactions by computing the inner products of latent vectors.

Given embedding $\mathbf{V}$ (the outcome of FPE component) and feature vector $\mathbf{x}$, the quadratic component is formulated as follows.

$$\phi_{\mathrm{QDR}}(\mathbf{V}, \mathbf{x}) = \sum_{j_1=1}^{d}\sum_{j_2=j_1+1}^{d} \mathbf{V}_{j_1,f_2}^\top \mathbf{V}_{j_2,f_1} \cdot x_{j_1}x_{j_2},$$

where f_1 and f_2 denote the fields of j_1 and j_2, respectively.

2.2.4 Deep Component. We deploy a neural network to learn high-order feature interactions, on the basis of feature embeddings generated by FPE component. For each feature, the FPE embedding is an $(f-1)\times k$ matrix. We vectorize all the embeddings and concatenate them together to form the input of a neural network.

We use the standard fully connected neural network, where the output of each hidden layer is defined by

$$\mathbf{x}^{(l+1)} = s(\mathbf{W}^{(l)}\mathbf{x}^{(l)} + \mathbf{b}^{(l)}),$$

where l is the layer number, s is the activation function, $\mathbf{x}^{(l+1)}$ and $\mathbf{x}^{(l)}$ are the output and input of layer l, and $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight and the bias of layer l to be learned. The output layer does not apply the activation function.

3 EXPERIMENTS

In this section, the performance of our proposed FPENN model is evaluated and compared with existing baselines on two benchmark datasets, which are the two mostly used publicly available ones in evaluation of CTR prediction.

For the fairness of our experiments, the batch size is set to 2000, the learning rate is 10^{-3}, activation function is *ReLU* and the optimization method is *Adam* [8] for all the experiments. We randomly sample 80% of the data for training and 20% for testing. We separate 20% of the training set as the validation set. Our grid search is done on the validation set, then the models are retrained on the training plus the validation set to obtain the final results. As for the evaluation criteria, we select the AUC (Area Under Curve) and the Logloss (cross entropy) as our metrics.

3.1 Datasets

We test the models on two public CTR prediction datasets, namely Avazu [2] and Criteo [3].

Avazu dataset contains 4×10^7 instances of 24 fields. To reduce the dimensionality of one-hot encoded features, we group the features that appear less than 20 times to one feature and finally obtain 645,195 features. For Criteo, we select "day 6" to "day 13" out of the one month Criteo data (including 39 fields), with the first 7 days for training and the last day for testing. Since the data is extremely unbalanced, with only 3% of positive samples, we randomly down-sample the negative samples to keep the ratio of positive samples and negative samples as 1:1. As for the 13 numerical fields, we convert them into categories through bucketing and then group the features that appear less than 20 times to one feature. After processing, there exist 1,178,909 features.

3.2 Experimental Results

To verify the effectiveness our model, we select LR, FM, FFM, CCPM [10] and DeepFM as our baselines and compare their performances on different datasets. LR, FM, and FFM are widely used in industry. CCPM tries to solve the CTR prediction problem using the convolutional neural network. DeepFM is a representative neural network model that combines FM and deep learning.

As for the parameter setting, the parameters are selected on the validation set to maximize the accuracy by grid search for all the models. For FPENN, the latent factor $k = 10$, the network structure is $[700\times3]$, $\alpha = 10^{-4}$ in UCB strategy, the dropout rate is 0.2 (i.e., nodes are dropped out with probability 0.2), and the weight of l_2 norm is 10^{-8} on Avazu. The latent factor $k = 4$, the deep network is $[700\times5]$, $\alpha = 10^{-3}$, the dropout rate is 0.1, and the weight of l_2 norm is 10^{-8} on Criteo. We tested both the TS and the UCB strategy of FPENN. Generally, FPENN with TS has lower accuracy than with UCB, since the prediction is made according to the drawn samples from the distribution for TS, where the excessive randomness may hurt the performance. Therefore for the remaining part of the paper, we only consider the UCB strategy.

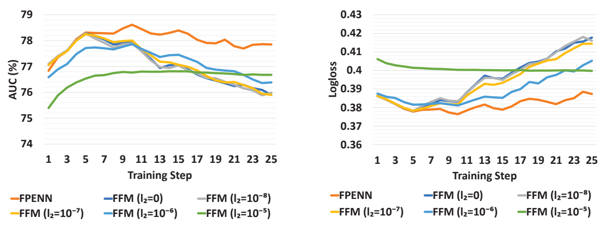
[2]https://www.kaggle.com/c/avazu-ctr-prediction

[3]http://labs.criteo.com/2013/12/download-terabyte-click-logs/

3.2.1 Generalization Ability. We first study the generalization ability of FPENN compared to FFM. Generalization measures the ability to predict the previously unseen data. The result is shown in Figure 2, from which we can see that the performance of our method (FPENN) drops very little as we increase training steps, while FFM under different settings drop significantly.

As mentioned in Section 1, FFM suffers from the overfitting problem, even with the regularizer. We record the AUC and Logloss value every 1/5 epoch out of 5 epochs, as shown in Figure 2. For FFM, the weight of the l_2 regularizer is tuned. When the weight is small, e.g., 0, 10^{-8} and 10^{-7}, the accuracy of FFM drops abruptly, degrading from 78.28% to about 75.9% in terms of AUC (decaying by 3.04%) and increasing from 0.3782 to about 0.41 in terms of Logloss (increasing by 8.40%). When the weight is large (namely 10^{-5}), though the curve is stable, the accuracy is low, with the highest AUC of 76.81% (the lowest Logloss of 0.3998). In contrast, the AUC of FPENN stays high and only declines from 78.61% to 77.70%, decreasing by 1.15%, while the Logloss increases only by 1.67%. The reason is that FPENN has the ability to learn the underlying distribution of the parameters.

The probabilistic model provides prior knowledge for training to prevent overfitting, since the probability assumption can be regarded as adding regularization to the model. As a result, the validation curve is more stable while the accuracy remains higher for FPENN, which is of great value for industrial applications. This indicates that no matter when to stop training, the accuracy does not drop significantly compared to FFM.



(a) AUC (%) **(b) Logloss**

Figure 2: Validation Curve of FPENN and FFM.

3.2.2 Accuracy. Next we validate the effectiveness of FPENN. We conducted t-test between FPENN and the baselines. The p-values on both datasets are less than 10^{-6}, which validate that our model significantly outperforms the others. As observed in Figure 1, our proposed FPENN has the highest accuracy, which achieves AUC of 78.61% and 79.97% on Avazu and Criteo respectively (0.3764 and 0.5417 in terms of Logloss). LR ranks the lowest on both datasets, since the model is basically a generalized linear model and only considers linear feature interactions. The AUC of FM increases over 1% compared with LR on both datasets, but it still lacks expressive power. CCPM outperforms FM but is not as good as FFM, it is because CNN-based methods are biased to the interactions between neighboring features. FFM and DeepFM have better performance, because they extend FM by considering field information and deep component respectively. As the best model, FPENN outperforms LR by 2.71% and 2.53% in terms of AUC (3.16% and 3.80% in terms of Logloss) on Avazu and Criteo respectively. Compared with the second best model, FPENN gives a rise of 0.32% and 0.11% in terms of AUC (0.34% and 0.11% in terms of Logloss). The reason is that our model captures both low-order and high-order feature interactions with field information. Moreover, the model estimates different confidence levels for the learned parameters, where parameters with low confidence levels can be corrected by combining the mean and the variance information of the distribution.

It should be noted that our improvement over DeepFM is of similar amount as previous improvements (DeepFM over FFM, FFM over CCPM, and CCPM over FM). In many industrial scenarios, the improvement is also of a similar scale; for example, the well-known Wide & Deep model improves 0.2% of AUC from the Wide model [3]. However, this seemingly small improvements of AUC can bring millions of dollars in revenue.

Table 1: Accuracy of Baselines and FPENN

	Avazu		Criteo	
	AUC (%)	Logloss	AUC (%)	Logloss
LR	76.53	0.3883	78.00	0.5631
FM	77.97	0.3802	79.09	0.5500
FFM	78.25	0.3782	79.77	0.5440
DeepFM	78.36	0.3777	79.91	0.5423
FPENN	**78.61**	**0.3764**	**79.97**	**0.5417**

3.2.3 Real-world Feasibility. We also test the real-world feasibility in a real-world commercial system. When applying a CTR prediction algorithm in a commercial system, it involves two phases, i.e., the offline training and online prediction. For the offline training, FPENN trains billions of data within one day on a GPU cluster. For the online prediction, FPENN predicts the CTR values for 200-300 Apps within 50-100 milliseconds, which satisfies the latency requirement of the commercial App market run by our commercial partner.

4 CONCLUSION

In this work, we propose an FPENN model for CTR prediction with both good accuracy and generalization ability. The FPE component learns a distribution for each element in the field-aware feature representation vectors to provide additional statistical information for prediction. Compared with the single point estimation of the deterministic models, FPENN has the ability to avoid overfitting by adding prior knowledge on the solution. Moreover, the variance is incorporated as an indicator of the confidence level of the empirical mean value. The Quadratic component learns low-order feature interactions while the Deep component models high-order feature interactions, which takes advantages of both shallow and deep models. The experiment results on two standard benchmark datasets demonstrate that our proposed model can achieve higher accuracy and significantly alleviate the overfitting problem of FFM.

ACKNOWLEDGMENTS

This work is partially sponsored by Huawei Innovation Research Program.

REFERENCES

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2-3 (2002), 235–256.

[2] Olivier Chapelle and Lihong Li. 2011. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*. 2249–2257.

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

[4] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 191–198.

[5] Google. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/ Software available from tensorflow.org.

[6] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence* (2017).

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

[8] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980

[9] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[10] Qiang Liu, Feng Yu, Shu Wu, and Liang Wang. 2015. A convolutional click prediction model. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 1743–1746.

[11] Kevin P. Murphy. 2012. *Machine learning - a probabilistic perspective.* MIT Press.

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

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

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

[15] Francisco R Ruiz, Michalis Titsias RC AUEB, and David Blei. 2016. The generalized reparameterization gradient. In *Advances in Neural Information Processing Systems*. 460–468.

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

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

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

[19] Guorui Zhou, Chengru Song, Xiaoqiang Zhu, Xiao Ma, Yanghui Yan, Xingya Dai, Han Zhu, Junqi Jin, Han Li, and Kun Gai. 2017. Deep interest network for click-through rate prediction. *arXiv preprint arXiv:1706.06978* (2017).