Deep&Cross Network for Software-Intensive System Fault Prediction

Guoxin Xia[1(✉)], Dongjin Yin[2], Yingtao Gan[2], Lanlan Rui[1], and Yu Zhu[1]

[1] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
736070995@qq.com
[2] Beijing TangMIX Technology Co., Ltd., Beijing, China

Abstract. With the development of information technology, the causes of software-intensive system failures become more complicated. This paper analyzes the correlation of various fault factors of software-intensive equipment and uses deep learning model to do fault prediction in complex electronic information system. Experimental results show that neural network model based on feature interaction can get better effect than some other methods.

Keywords: Software-intensive · Fault prediction · Field-aware Factorization Machine · Deep&Cross network

1 Introduction

With the rapid development of software and hardware and the continuous upgrading of various modern equipments, software has become more and more important, and it has more and more functions. Software-intensive systems have gradually become the mainstream form of software systems [1].

At the same time, the requirements of system availability and security become more stringent in such complex systems. What's more, maintenance is becoming more complicated after system error. Software is less reliable than hardware. Once a fault occurs, it not only causes the loss of the corresponding hardware function, but even causes the whole system to fall into paralysis. So it's important to predict which fault factors cause the error and to repair system targetedly based on the prediction.

The complexity of software grows exponentially with increasing demand, and the incidence of software-intensive system disasters due to software failures continues to increase. Some fault prediction methods for software-intensive systems are knowledge-based, mainly includes expert system [2] and fuzzy logic. Some fault prediction methods are based on models, like failure physical model and state space model. However, with the development of machine learning and deep learning theory, data-based fault prediction technology is starting to become a trend.

S. Han et al. (Eds.): AICON 2019, LNICST 287, pp. 28–39, 2019.
https://doi.org/10.1007/978-3-030-22971-9_3

There are 3 main reason for higher fault rate of software-intensive system:

Logical complexity: As code lines increase and modularize, interaction modules increase, it's difficult to control and predict the behavior of these interactions. Studies have shown that more and more failures occur because of abnormal interactions within the system.

Boundary erosion: software interactions and complex software-intensive peripheral hardware systems have become so frequent and complex software boundaries have been eroded by peripherals. The survey revealed that when the interaction between software and hardware fails, part or whole system will fall into trouble.

Longevity: Complex tasks require these systems to operate over long life cycles, during which the system must combat external and internal failure attacks and evolution to adapt to the environment but remain quasi-stable, which requires more attention and may cause system function failure in runtime, such as abnormal runtime data flow.

Longevity of system means more data can be collected. So data-based fault prediction can have a good effect. In this paper, we use Deep&Cross network to predict the fault factors of software-intensive systems. In particular, it makes following contributions: (1) Solving the problem of feature combination under sparse data while keeping different characteristics between different features, so training time and memory consumption will be less and generalization will be better; (2) The cross network layer introduced by DCN can express any high-order feature combination, while each layer retains low-order combination, and the vectorization of parameters also controls the complexity of the model. So logical complexity and boundary erosion of software-intensive systems are considered to predict more accurately.

2 Related Work

Fault prediction problems received a lot of attention from researchers. Some used autoregressive moving average (ARMA) model [3] or its variants to do fault prediction based on time series. And some did the same things by using LSTM (Long Short-Term Memory) [4] or its variants. When the software-intensive systems were not so complicated, some methods analyze and determine the location of defects by the basic tool that recorded running information of target software. These methods are based on the software failure mechanism, using the concept of control flow and data flow, analyzing the application of program slicing and dynamic slicing technology, and drawing on the idea of "software black box" [5, 6].

As complexity of software-intensive systems increases and large amount of data generate, methods based on deep learning play an increasingly important role. The fault prediction process is as shown below (Fig. 1):

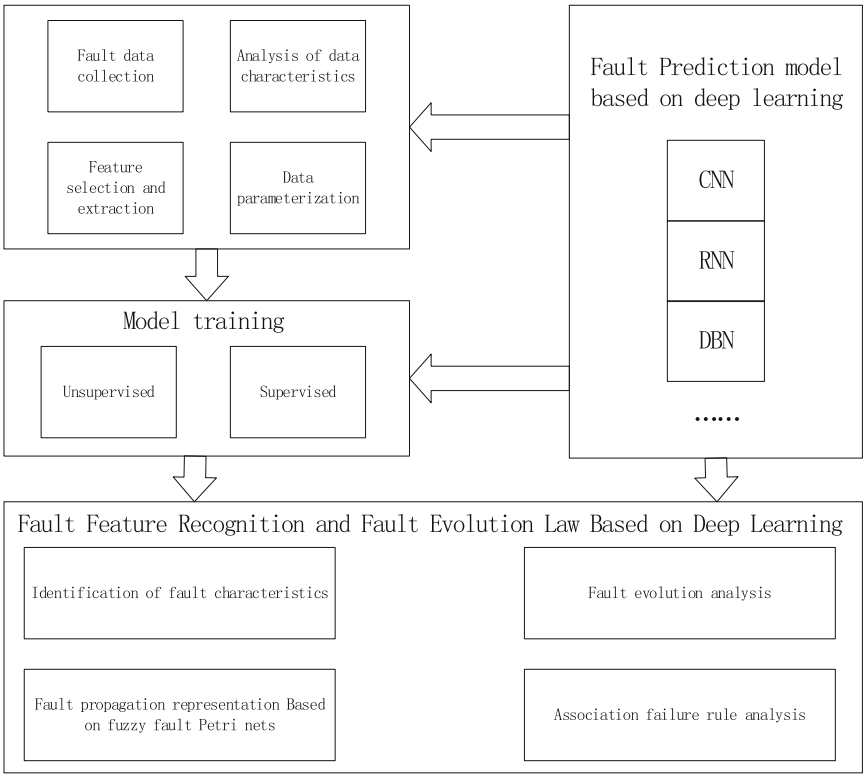

Fig. 1. Fault prediction process.

3 Fault Prediction in a Software-Intensive System

3.1 Fault Characteristics

In a software-intensive system, due to the embedding of lots of software programs which play an equally important role as its hardware system, the failure mode includes not only the software and the hardware but some new problems that hardware and software combination causes [7]. Generally there are three types as follows: (1) The software program is wrong and these errors are passed to the hard through the interaction of software and hardware. And it causes hardware failure; (2) Hardware damage or failure affects the software associated with it, resulting in software errors; (3) There are no separate errors or malfunctions in software and hardware, but when a contact occurs in them, a system failure occurs, or the contact cannot be completed.

There are five typical characteristics in software-intensive systems: hierarchy, relevance, randomness, uncertainty and cross-propagation. These characteristics in software-intensive systems have become more and more obvious with increased software complexity. So expert knowledge system have more limitations and the difficulty of the definition of knowledge is also increasing. In the case of composite faults,

each fault interacts with the system. The fault modeling and theoretical analysis of the composite fault cannot be performed using the certain mode. Deep learning does not rely on prior assumptions and automatically detects interactions.

3.2 Fault Factors

There are a variety of potential failure factors in a software-intensive system. For example, in software-intensive electronic information systems, some fault factors are as follows:

1. Software failure type: including input and output problems, program problems, performance problems, design/logic problems, timing problems, data problems
2. Environment type: temperature, humidity, electromagnetic, current exceed the threshold or change too much; typical misoperations such as mis-opening, mis-closing, etc.
3. Hardware failure type: hardware short circuit, open circuit, parameter problem, hardware damage or looseness.

3.3 Features and Feature Combination

We collected and processed a dataset about software-intensive electronic information system. After filtering, 76 features can be used to do prediction and 27 mainly fault factors exist. Some samples are in Table 1.

Table 1. Fault samples.

Number	Features closely connected to fault factors	Fault factors
1	Speed fluctuation of CPU fan	CPU fan is faulty
2	Server log alarm	Database content is lost
3	CPU temperature is too high for a long time && Computer crash	CPU burned down
4	CPU temperature is in normal && Unable to start server && CPU pin is abnormal	Poor CPU socket contact
5	Abnormal humidity	Hardware deformation
……	……	……

Considering communication and combination in features, we can turn this prediction into a feature combination problem. For example, CPU temperature exceeds eighty degrees may not cause fault. But if CPU have been working for a long time, the situation will be different.

After some features are correlated, their correlation with the prediction label will increase. A polynomial model is the most intuitive model that contains a combination of features. In a polynomial model, the combination of the features x_i and x_j is represented by $x_i x_j$. For a second-order polynomial model, the model's expression is as

follows, where n represents the number of features of the sample, x_i is the value of the i-th feature, w_0, w_i and w_{ij} are model parameters.

$$y(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} x_i x_j \tag{1}$$

It's obvious in (1), the number of parameters of the combined feature is $\frac{n \times (n-1)}{2}$. However, after one-hot, features will be very sparse. Samples satisfying that *xi* and *xj* are not all zero are too enough to learn w_{ij} accurately. In FM (Factorization Machine) [8], w_{ij} form a symmetric matrix W which can be expressed as:

$$W = V^T V \tag{2}$$

In other words, $w_{\mathrm{ij}} = \ < v_{\mathrm{i}}, v_j >$. So FM's model equation is:

$$y(x) = w_0 + \sum_{i=1}^{n} w_i x_i \sum_{i=1}^{n} \sum_{j=i+1}^{n} < v_i, v_j > x_i x_j \tag{3}$$

In this equation, V_{i} is the hidden vector of the i-dimensional feature, < , > represents the vector dot product. The length of the hidden vector is k (k << n), which contains k factors describing the feature. The parameter factorization makes the parameters of $x_{\mathrm{h}} x_i$ and the parameters of $x_{\mathrm{i}} x_j$ no longer independent of each other, so we can estimate the quadratic parameters of FM relatively reasonably in the case of sparse samples.

FM's embedding is shown as below (Fig. 2):

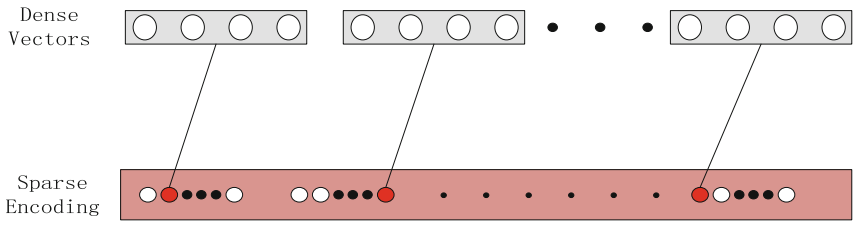

Fig. 2. FM embedding.

By introducing the concept of field, FFM (Field-aware Factorization Machine) [9] attributes features of the same nature to the same field. Taking the CPU temperature as an example, we set multiple levels at first. The three characteristics of 65–70 °C, 70–75 °C or 75–80 °C are all about CPU temperature and can be placed in the same field. In the same way, the data problem may be all the data loss, partial loss, data file virus infection, database operation error, database deadlock, etc. These dozens of features are all data problems, so they can also be put together in a field.

In FFM, feature x_i in each dimension learns a hidden vector v_{i,f_j} for each field f_j of other features. Therefore, the hidden vector is not only related to the feature, but also related to the field. That is to say, the "temperature of CPU is in 70–75 °C" feature uses

different hidden vectors when it is associated with the "program input error" feature and the "current value size" feature.

Assuming that the n features of the sample belong to f fields, then the quadratic term of the FFM has n * f hidden vectors. In the FM model, there is only one hidden vector for feature of each dimension. FM can be seen as a special case of FFM, it is an FFM model that assigns all features to one field.

FFM's model equation is:

$$y(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < v_{i,f_j}, v_{j,f_i} > x_i x_j \tag{4}$$

FFM's embedding is shown as below (Fig. 3):

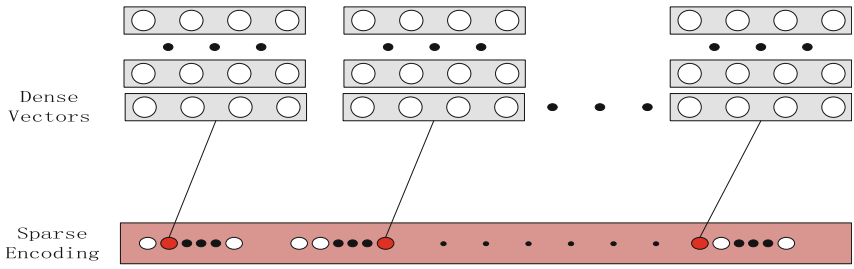

Fig. 3. FFM embedding.

4 Model Structure

DCN (deep & cross network) [10] model's structure is shown in Fig. 4. In this prediction problem, feature crossing is an important step, but many network structure only learns the secondary crossover at most. The Logistic Regression model uses the original artificial crossover feature, and Product-based Neural Network [11] uses the product method to do the second-order crossover. Neural Factorization Machines [12] and Attention Neural Factorization Machines [13] also use Bi-interaction to learn the second-order intersection of features. For higher-order feature intersections, others only let deep section learn. But DCN can express any high-order combination while each layer retains a low-order combination by cross layer network.

At first, we make data cleaning for our features collected. After that, we do FFM embedding for high dimensional sparse features and contact them with dense feature. So after embedding and stacking layer, the features can be shown as a vector like:

$$X_0 = [x_{embed,1}^T, \ldots, x_{embed,k}^T, x_{dense}^T] \tag{5}$$

After embedding and stacking layer, the network is divided into two ways. One way is the traditional DNN structure as follows

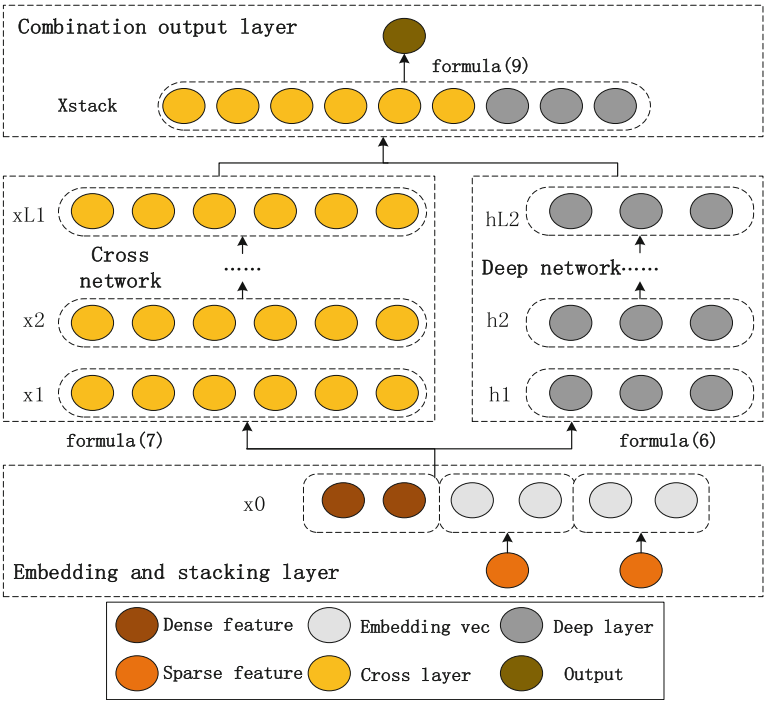

Fig. 4. Structure of Deep&Cross network.

$$h_{l+1} = Relu(w_l h_l + b_l) \tag{6}$$

When vector dimension of X0 is d and there are m units in each layer of DNN, the parameters that need to be learned in total are d * m + m * (m + 1) * ($L2$ − 1). Another way is DCN's core, cross network. Assuming that the network has L1 layers, the relationship between each layer and the previous layer can be expressed by the following relationship:

$$x_{l+1} = x_l x_l^T w_l + b_l + X_l = f(x_l, w_l, b_l) + x_l \tag{7}$$

In above, f is the function to be fitted, and x_l is the network input of the previous layer. The parameters that need to be learned are w_l and b_l. Since the dimension of x_l is d, and x_{l+1} is also d-dimension. The parameters w_l and b_l which need to be learned are also d-dimension vectors. Therefore, each layer has 2 * d parameters (w and b) that need to be learned.

After deep layer network and cross layer network, we use sigmoid function to output result:

$$x_{\text{stack}} = Concat(x_{l1}, h_{l2}) \tag{8}$$

$$\text{p} = \text{sigmoid}(W_{\text{logit}} x_{\text{stack}} + b_{\text{logit}}) \tag{9}$$

5 Experiments and Results

We do experiments based on a large number of test data from daily operations for complex electronic information systems.

After data preprocessing, 76 features are chosen for training and 27 fault factors exist. Fault data feature description are shown in the following Table 2. Discrete variables are converted to one-hot format.

Table 2. Fault data feature description.

Number	Raw feature	Discrete variable	Continuous variable	Example
1	Code length of each module		✓	1737
2	The extent to which the hardware reaches the specification	✓		[1, 0, 0, ……]
3	CPU temperature		✓	76 (°C)
4	CPU occupancy		✓	43 (%)
5	Memory occupancy		✓	66 (%)
6	Degree of constraint by size of main memory or storage availability	✓		[1, 0, 0, ……]
7	Satisfactoriness of system response time	✓		[1, 0, 0, ……]
8	Stability of hardware and system support	✓		[1, 0, 0, ……]
9	System runtime		✓	24276 (s)
10	Syslog level	✓		[1, 0, 0, ……]
11	Syslog refresh speed		✓	
……	……	……	……	……

We compared DCN method with other models, for example, DNN, SVM, NFM (Neural Factorization Machines) and FDES (fault diagnosis expert system). The goal is to predict fault factors and evaluation standard is precision, recall, F1-measure, accuracy and AUC (Area Under roc Curve). The calculation method of these indicators is the same as two classification problem by thinking of all incorrect categories as a negative sample. For example, when a test set is all predicted, there will be some samples predicted to be other classes while they are actually from class1, and some samples that are not actually from class1, predicted to be Class1, which leads to the following result:

Table 3. Classification situation.

	The predicted result is Class1 (Positive)	The predicted result is non-Class1 (Negative)
The prediction result is true (True)	Class1_TP: The number of samples that are predicted as Class1 and are actually Class	Class1_ TN: The number of samples that are not actually Class1 and are also predicted as other classes (non-Class1)
The prediction result is false (False)	Class1_ FP: The number of samples that are predicted as Class1 but are not actually Class	Class1_ FN: The number of samples that are actually Class1 but are predicted to be other classes (non-Class1)

According to the above Table 3, we can calculate:

$$\text{Class1}_{\text{Precision}} = \frac{\text{Class1\_Tp}}{\text{Class1\_TP} + \text{Class1\_FP}} \tag{10}$$

$$\text{Class1}_{\text{Recall}} = \frac{\text{Class1\_TP}}{\text{Class1\_TP} + \text{Class1\_FN}} \tag{11}$$

$$\text{Class1}_{\text{F1Score}} = \frac{2 * (\text{Class1\_Precision} * \text{Class1\_Recall})}{\text{Class1\_Precision} + \text{Class1\_Recall}} \tag{12}$$

$$\text{Class1}_{\text{Accuracy}} = \frac{\text{Class1\_TP} + \text{Class1\_TN}}{\text{count(Samples)}} \tag{13}$$

Then final precision, recall, F1Score, accuracy can be calculated by taking the average of all classes' corresponding values.

We mainly used TensorFlow and python package scikit-learn to implement most of the models and have adjusted parameters to get the best performance for each single model. The results are as follow (ROC curve means receiver operating characteristic curve):

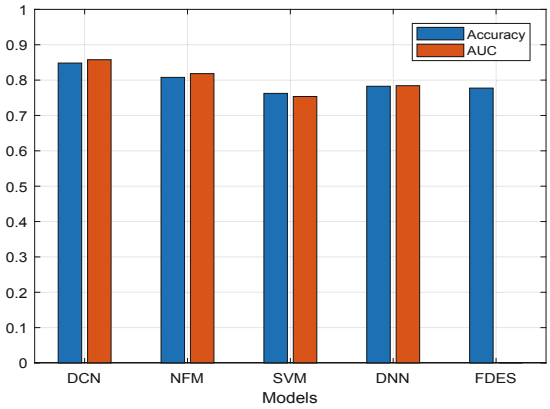

Fig. 5. Accuracy and AUC.

Table 4. Three evaluating indicators of 5 models.

Models	Indicators		
	Precision	Recall	F1Score
DCN	0.843	0.852	0.847
NFM	0.812	0.809	0.810
SVM	0.774	0.778	0.776
DNN	0.792	0.786	0.789
FDES	0.776	0.772	0.774

For FDES, we calculate only accuracy but not AUC, because FDES gives a result without probability. As we can see in Fig. 5 and Table 4. Deep&Cross network model gets the highest accuracy, AUC and F1Score. Obviously, multi-order intersection of features has significant effect. However, other models can't learn multi-order intersection of features effectively because of the lack of cross network structure. When we use FM embedding to replace FFM embedding in DCN model, the accuracy and AUC will drop a little bit. It means FFM embedding learned a more suitable hidden vector than FM embedding by distinguishing feature field.

As for training time, DCN with FFM takes about 12476 s and DCN with FM takes about 11382 s. There is not much difference in training time between the two methods. NFM takes 16232 s, DNN takes 17286 s and SVM takes 28109 s. FM and FFM make feature space small, so models can converge quickly. FFM is slower than FM due to diversity of feature combinations.

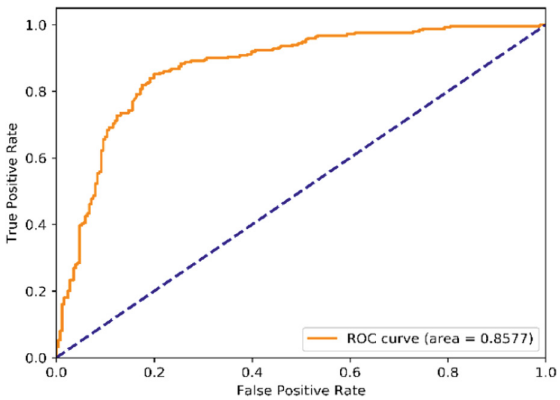

Fig. 6. ROC curve of DCN model.

Figure 6 shows the ROC curve of DCN model. An ideal ROC curve has a shape which covers the maximum area. From the experimental results above, we can see that DCN has a better effect than other models.

6 Summary

Software-intensive systems' failure mode are very complex. Fault characteristics from software and hardware are closely related. So mining associations between features is important to get a great result. DCN can express any high-order feature combination, while each layer retains low-order combination, and the vectorization of parameters also controls the complexity of the model. So it's suitable for prediction the failure point of software-intensive systems.

Acknowledgement. The work presented in this paper was supported by the National Natural Science Foundation of China (61302078, 61372108), 863 Program (2011AA01A102), National S&T Major Project (2011ZX 03005-004-02), and China Postdoctoral Science Foundation funded Project 2017M610827.

References

1. Shu, S., Wang, Y., Wang, Y.: A research of architecture-based reliability with fault propagation for software-intensive systems. In: Reliability and Maintainability Symposium, pp. 1–6. IEEE (2016)
2. Chen, W.B., Liu, X.L., He, C.J., et al.: Knowledge base design for fault diagnosis expert system based on production rule. In: Asia-Pacific Conference on Information Processing, APCIP 2009, pp. 117–119. IEEE (2009)
3. Baptista, M., Sankararaman, S., Medeiros, I.P.D., et al.: Forecasting fault events for predictive maintenance using data-driven techniques and ARMA modeling. Comput. Ind. Eng. **115**, 41–53 (2018)
4. Zhang, K., Xu, J., Min, M.R., et al.: Automated IT system failure prediction: a deep learning approach. In: IEEE International Conference on Big Data. IEEE (2017)
5. Mohan, K.K., Verma, A.K., Srividya, A.: Software reliability estimation through black box and white box testing at prototype level. In: International Conference on Reliability. IEEE (2011)
6. Li, N., Li, Z., et al.: Predicting software black-box defects using stacked generalization. In: Sixth International Conference on Digital Information Management. IEEE (2011)
7. Li, L., Lu, M.: Analyzing failure mechanism for complex software-intensive systems. In: IEEE International Symposium on Software Reliability Engineering Workshops. IEEE Computer Society, pp. 57–58 (2015)
8. Guo, H., Tang, R., Ye, Y., et al.: DeepFM: a factorization-machine based neural network for CTR prediction. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, pp. 1725–1731 (2017)
9. Juan, Y., Zhuang, Y., Chin, W.S., et al.: Field-aware factorization machines for CTR prediction. In: ACM Conference on Recommender Systems, pp. 43–50. ACM (2016)
10. Wang, R., Fu, B., Fu, G., et al.: Deep & Cross Network for Ad Click Predictions. In: Proceedings of the ADKDD 2017, pp. 1–7 (2017)
11. Qu, Y., Cai, H., Ren, K., et al.: Product-based neural networks for user response prediction. In: 16th International Conference on Data Mining, pp. 1149–1154 (2016)

12. He, X., Chua, T.: Neural factorization machines for sparse predictive analytics. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 355–364 (2017)
13. Xiao, J., Ye, H., He, X., et al.: Attentional factorization machines: learning the weight of feature interactions via attention networks. In: 26th International Joint Conference on Artificial Intelligence, IJCAI 2017, pp. 3119–3125 (2017)