CSIT 6000M Project-2 Report: Reproduction and Improvement of xDeepFM

Hexiao Zhang
20932780
hzhangea@connect.ust.hk

Ke Zheng
20882179
kzhengah@connect.ust.hk

1 Introduction

The explosive growth of online platforms, including e-commerce, social media, and online advertising, has made personalized recommendations ubiquitous in our daily lives. Click-through rate (CTR) prediction is crucial in recommendation systems, as it estimates the likelihood that a user will click on a given item. Accurate CTR prediction is key to the success of various online applications, including advertising optimization, content recommendation, and personalized search. Accurate CTR prediction can result in higher click-through rates, better user engagement, and increased revenue for online platforms.

CTR prediction is essentially a binary classification task for structured data. One significant challenge is that features are often highly sparse, with many missing values. Modeling relationships between them is difficult and requires extensive manual feature engineering. With the advent of deep learning, many deep learning models have been proposed, which can automatically extract the interaction between features and achieve better performance than traditional methods. xDeepFM is a famous one of them.

In this assignment, we reproduce the xDeepFM model on the Criteo dataset. Noticing that the Compressed Interaction Network (CIN) doesn't utilize numerical features, we propose a method to encode numerical features as input to CIN. By doing so, the CIN can extract the relationship between numerical and category features. Our experiments demonstrate the effectiveness of this technique, which can improve the model's performance and replace the role of DNN in xDeepFM.

2 Related Work

This section briefly introduces the development of click-through rate (CTR) prediction algorithms and the application of deep learning in this field.

Before the rise of deep learning, CTR prediction was often achieved through manual feature engineering, which has four major drawbacks. First, it requires feature designers to have an in-depth understanding of the business and its users, as well as domain-specific knowledge. Second, in the era of big data, recommendation systems often need to process millions or even billions of data points, making it nearly impossible to manually extract all cross-features. In addition, manual feature engineering cannot generalize across different datasets. Lastly, manual features can usually only extract relationships between pairs of features (2nd order) and struggle to handle higher-order feature interactions. Therefore, implementing automatic feature processing and interaction is an urgent need.

Factorization Machine (FM) [1] maps original features to embeddings and models the correlation between features through the inner product of the embeddings. The weighted sum of the original features is used to obtain the final prediction output. The formula is shown below:

$$y = \sum_{i,j} < V_i, V_j > x_i \cdot x_j \tag{1}$$

where x_i and x_j represent original features, and $< Vi, Vj >$ represents the inner product of the embeddings of the original features. FM can automatically implement pairwise feature interactions and is simple and effective. It is the foundation of many feature interaction models that follow. In theory, FM can be extended to high-order feature interactions, but this can lead to high complexity. Additionally, in recommendation systems, FM maps these zero vector features to non-zero embeddings, which can lead to learning too many useless interaction terms and introducing noise.

With the development of Deep Learning, researchers began to apply the powerful feature-learning ability of neural networks to recommendation systems. [2] and [3] respectively used Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) for rating prediction in recommendation systems. CNN focuses more on the interaction between adjacent features due to the property of convolution. RNN has better modeling performance for sequential feature interactions due to its time characteristics.

The above DNN-based recommendation system models for learning high-order feature interactions ignore low-order feature interaction relationships [4].

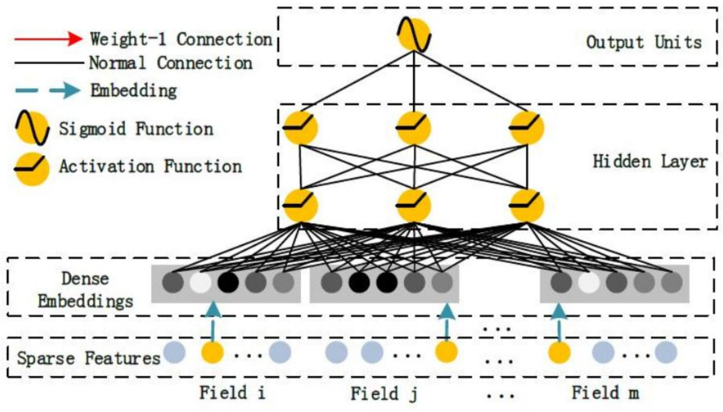


Figure 1: The DNN in DeepFM

DeepFM [5] proposes an end-to-end network structure that combines linear prediction, FM, and DNN. It can learn both low-order (Linear, FM) and high-order (DNN) feature interactions and does not require manual construction of cross-product transformations. However, as a black-box structure, DNN models feature interactions implicitly, resulting in an unpredictable and uncertain network that is difficult to analyze.

To achieve explicit modeling of high-order feature interactions, [6] proposed a Deep & Cross Network (DCN) structure, in which the key module is the Cross Network (CrossNet). The CrossNet uses the output of the embedding layer x_0 and the output of the $(k-1)$-th layer network to calculate the output of the next layer, i.e., the k-th layer network, as follows:

$$x_k = x_0 x_{k-1}^T w_k + b_k + x_{k-1} \tag{2}$$

The CrossNet explicitly models feature interactions and iteratively deepens the interaction degree through multiple network layers.

Another problem of CrossNet and DeepFM is that feature interactions are modeled at the bit level, which means they don't treat the embedding vector for each feature as a whole.

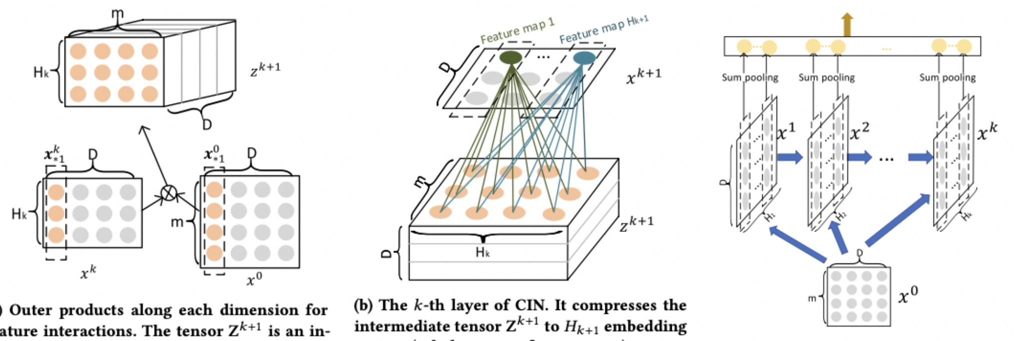
The xDeepFM [7] hybrid network that we reproduce is inspired by DCN and DeepFM. It proposes a Compressed Interaction Network (CIN) to replace the CrossNet in DCN. The CIN explicitly learns feature interactions, and the degree of feature interaction deepens as the network progresses layer by layer, while CIN operates at the vector level. By combining CIN with DNN and linear models, it designs a feature interaction network that utilizes explicit and implicit interactions, low-order and high-order interactions, and vector-level and bit-level interactions.

3 Algorithms

3.1 Problem Definition

The CTR prediction is a binary classification task. The input is feature x, including user features (such as age, gender, language, etc.), contextual features (access time, equipment, access records, etc.) and item features (item category, item history statistics, etc.). The output is designed as the conditional probability P(y|x) of the user performing an operation y given the feature x.

3.2 Compressed Interaction Network



(a) Outer products along each dimension for feature interactions. The tensor Z^{k+1} is an intermediate result for further learning.

(b) The k-th layer of CIN. It compresses the intermediate tensor Z^{k+1} to H_{k+1} embedding vectors (aslo known as *feature maps*).

(c) An overview of the CIN architecture.

Figure 2: The Compressed Interaction Network

The Compressed Interaction Network (CIN) achieves feature interaction at the vector level, and can explicitly reflect the interaction degree of high-order features.

Before feeding the original feature vector into the CIN, we first pass it through an embedding layer, mapping each feature field to an embedding of uniform length D. To achieve interaction at the vector level, each feature field is unfolded into a matrix $\mathbf{X}^0 \in \mathbb{R}^{m \times D}$, where m is the total number of fields. The output of the k-th layer of the CIN is denoted as $\mathbf{X}^k \in \mathbb{R}^{H_k \times D}$, where H_k represents the number of feature vectors in the k-th layer. The output of each layer is obtained by the interaction between the output of the previous layer and $\mathbf{X}^0$, calculated as follows:

$$\mathbf{X}^k_{h,*} = \sum_{i=1}^{H_{k-1}} \sum_{j=1}^{m} \mathbf{W}^{k,h}_{ij} (\mathbf{X}^{\mathbf{k-1}}_{\mathbf{i},*} \circ \mathbf{X}^0_{j,*}) \tag{3}$$

Here, $\circ$ represents element-wise multiplication. Since each layer's features interact with $\mathbf{X}^0$, the interaction degree can be explicitly seen as deepening layer by layer. Moreover, the operation targets the entire feature vector, which means the feature interaction is at the vector level.

In practice, the above process can be implemented using a convolutional approach. As shown in Figure 2a, first, the outer product of $\mathbf{X}^0$ and $\mathbf{X}^k$ is taken along the feature dimension D to obtain an intermediate tensor representation $\mathbf{Z}^{k+1}$. And then, as shown in Figure 2b, a one-dimensional convolution is performed along the feature dimension D using the convolutional kernel $\mathbf{W}^{k,h}$. This operation is repeated for a total of H_{k+1} times to obtain the output of the $(k+1)$-th layer, $\mathbf{X}^{k+1}$. Finally, as shown in Figure 2c, after calculating the output of each layer of the CIN, sum pooling is performed along the feature dimension D for each hidden layer output, i.e., $p_i^k = \sum_{j=1}^{D} \mathbf{X}^{\mathbf{k}}_{\mathbf{i,j}}$. This yields the pooling vector of the k-th hidden layer, $\mathbf{p}^k = [p_1^k, p_2^k, \cdot, p_{H_k}^k]$. Finally, the hidden layer pooling vectors are concatenated to obtain $\mathbf{p}^+ = [\mathbf{p}^1, \mathbf{p}^2, \cdot, \mathbf{p}^T]$.

3.3 xDeepFM

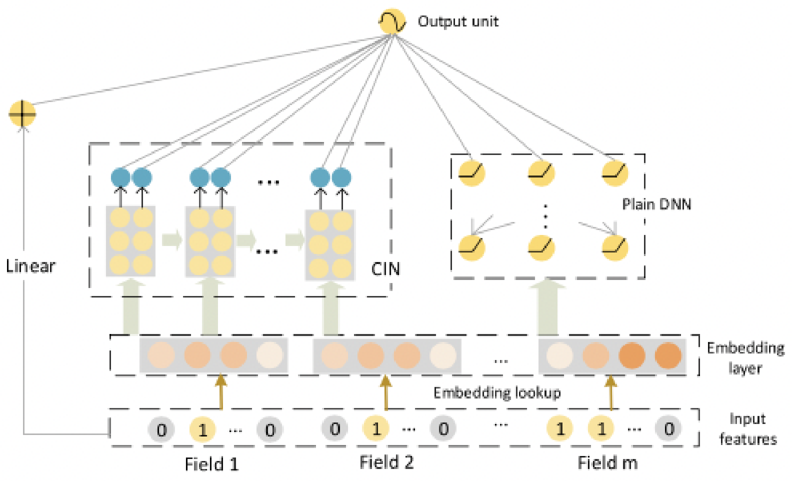


Figure 3: xDeepFM

Considering that the CIN explicitly learns high-order and low-order feature interactions, the DNN implicitly learns high-order feature interactions, and the linear model learns low-order feature interactions, xDeepFM combines all three models.

The output unit sums the outputs of the three component units and passes them through a sigmoid function to obtain the binary classification probability.

$$\hat{y} = \sigma(\mathbf{w}_{\mathbf{linear}}^{\mathbf{T}}\mathbf{a} + \mathbf{w}_{\mathbf{dnn}}^{\mathbf{T}}\mathbf{x}_{dnn}^{k} + \mathbf{w}_{\mathbf{cin}}^{\mathbf{T}}\mathbf{p}^{+} + b) \tag{4}$$

3.4 Our Proposed Improvement

The original CIN can only handle category features. These features are first converted into one-hot vectors and then into dense embedding vectors. However, numerical features are ignored. They are taken as input by the DNN and linear model, but not CIN.

We speculate that the reason the CIN layer ignores numerical features is that it is used for feature interactions at the vector level. Category features can be converted into embedding vectors easily, but since numerical features are continuous values, it is difficult to vectorize them.

We propose a simple method for vectorizing numerical features, as shown in Figure 4. We expand the value of numerical features by copying them into a dense feature vector. Then, we concatenate the embedded representation of category features and the obtained dense feature vector together and input them into the CIN layer. This allows the CIN layer to perform more comprehensive high-order feature interactions.

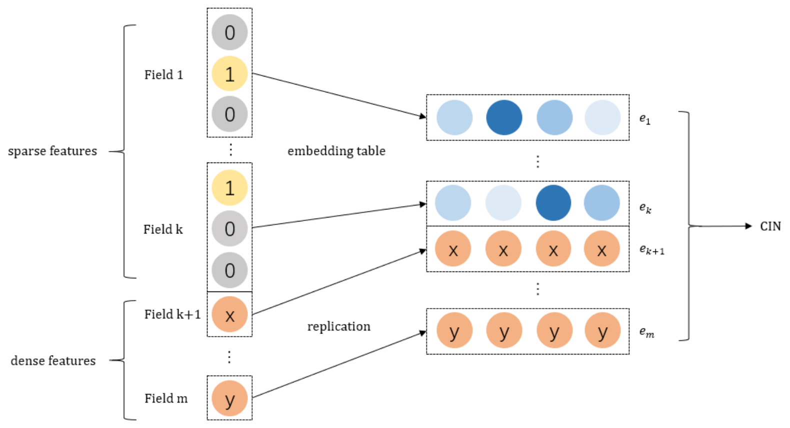


Figure 4: The Input of improved CIN structure

4 Experiments

The reproduction of the xDeepFM model was based on the work of shenweichen[1] and we conducted experiments and further improvements on this foundation.

4.1 The Criteo Dataset

The Criteo Advertising Dataset[2] is a publicly available and classic dataset used for predicting ad click-through rates. In the Criteo dataset, each row represents a sample, where each sample includes:

- a label indicating whether the ad was clicked (1) or not clicked (0);
- 13 numerical features named I1-I13;
- 26 categorical features named C1-C26, where the values of these features are hashed to 32 bits for privacy reasons. When a feature is missing, the field is empty.

Table 1: Samples from the Criteo Dataset

label	I1	I2	I3	...	C23	C24	C25	C26
0	1.0	1	5.0	...	3a171ecb	c5c50484	e8b83407	9727dd16
0	2.0	0	44.0	...	3a171ecb	43f13e8b	e8b83407	731c3655
0	2.0	0	1.0	...	3a171ecb	3b183c5c	Na	Na
0	Na	893	Na	...	3a171ecb	9117a34a	Na	Na

In this project, we used approximately 40% the training set (20 million records) and the size ratio of training set, validation set and test set is 64%:16%:20%.

4.2 Preprocessing

In the preprocessing of the dataset, we fill the missing values by 0 for numerical features and -1 for category features, indicating a new category. Then, the numerical features are normalized to 0-1 using maximum-minimum normalization, and the sparse features are coded as one-hot vectors.

[1]https://github.com/shenweichen/DeepCTR-Torch
[2]https://labs.criteo.com/2014/02/download-kaggle-display-advertising-challenge-dataset/

4.3 Hyperparameters and Training

We follow the settings in [7], setting the dimensions of the embedding vectors to 10. The DNN consists of 2 linear layers, with a hidden layer size of 400, using ReLU as the activation function, and a dropout rate of 0.5. The linear layers are initialized with a normal distribution with a mean of 0 and a variance of 0.0001. The CIN consists of 3 layers, each with 200 dimensions (i.e., extracting 200 interaction features per layer). Additionally, to ensure the reproducibility of the results, we set all random seeds to 17.

Training was performed on an RTX 2080Ti GPU with Ubuntu 20.04. The Python version is 3.9.10 and third-party libraries are documented in requirements.txt.

We use the Adam optimizer with a learning rate of 0.001, batch size of 1024, and L2 regularization coefficient of 0.0001, and train for a total of 10 epochs. In each round of training, we first train and update parameters on the training set and record performance metrics on the training set, then test on the validation set with a batch size of 256.

To evaluate the feature interaction model, we use AUC (Area Under the ROC Curve) and Logloss as performance metrics to measure the accuracy of score prediction. AUC measures the probability that a randomly selected positive sample has a higher predicted score than a negative sample. This evaluation metric focuses only on the ranking order and not on the specific scores, allowing it to effectively measure the model's performance in imbalanced class situations. Logloss uses cross-entropy loss, which calculates the difference between the predicted scores and the true labels for all samples.

After training, we select the model with the highest val_score for testing. Here, we define the val_score as

$$val\_score = val\_auc + \lambda(1 - val\_logloss), \tag{5}$$

where $\lambda = 1$ in our experiment.

4.4 Results

As shown in the training curve in Figure 5, as the training progresses, the AUC on the training set continuously increases, and the LogLoss continuously decreases. On the validation set, the AUC rises, and LogLoss declines in the first two rounds of training. Afterward, the AUC continuously drops, and LogLoss continuously rises, indicating that the model was overfitting.

The results on the test set are AUC = 0.7959 and LogLoss = 0.4555. Compared to the original paper's results [7] with AUC = 0.8052 and LogLoss = 0.4418, our replication results have a slightly lower AUC (by 0.0093) and a higher LogLoss (by 0.0137), but they are still very close to the original results. We speculate that the slightly lower replication performance may be due to: 1) different batch size settings during training, with the original paper setting the batch size to 4096, while we were limited by GPU memory and set the batch size to 1024; and 2) some parameters not specified in the original paper, such as the model initialization method, dataset splitting ratio, and random seed for splitting, making it impossible to be entirely consistent with the original paper; 3) we use a smaller portion of the dataset.

We conducted ablation experiments by removing either the CIN or DNN module from the xDeepFM model for training (keeping the linear layer in the xDeepFM model) and testing on the test set, obtaining the results shown in Table 2.

From the table, we can see that using both CIN and DNN on top of the linear layer yields better results compared to using either CIN or DNN alone. This confirms the effectiveness of xDeepFM in combining CIN and DNN.

To test our improved model, we trained two model structures, one with only CIN and the other with both CIN and DNN (xDeepFM) (keeping the linear layer in the model), and tested the four cases of whether to add numerical feature vectors on the test set. The results are shown in Table 3.

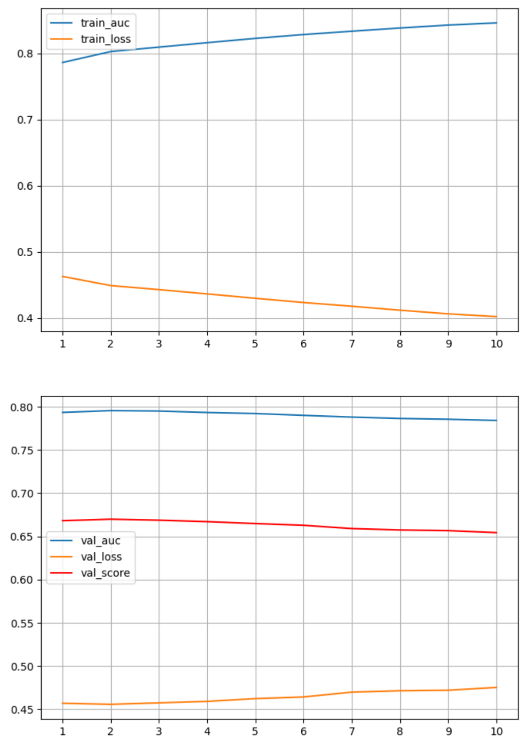


Figure 5: The training curve of the xDeepFM model, with the top figure displaying the changes in AUC and LogLoss on the training set, and the bottom figure showing the changes in AUC, LogLoss, and val_score on the test set. To better display them in the same plot, we divide the val_score by $(1 + \lambda)$.

Table 2: Ablation study

Model	AUC	LogLoss
Linear+CIN	0.7862	0.4635
Linear+DNN	0.7955	0.4561
Linear+CIN+DNN (xDeepFM)	0.7959	0.4555

As can be seen, the performance of the model improves after adding numerical feature vectors, whether using only CIN or both CIN and DNN (xDeepFM). This demonstrates the effectiveness of our strategy, with which the CIN module can perform feature interactions between category and numerical features.

Furthermore, we noticed that after adding numerical feature vectors, the performance of using only CIN is comparable to using both CIN and DNN (xDeepFM). We speculate that the reason is without numerical feature vectors, one of the roles of DNN is to supplement the feature interactions between category and numerical features, which are not learned by CIN. So the role of DNN is no longer significant after adding numerical feature vectors.

Table 3: Results of different structures with and without numerical feature vectors

Model	With Numerical Features	AUC	LogLoss
Linear+CIN	No	0.7862	0.4635
	Yes	0.7974	0.4538
Linear+CIN+DNN (xDeepFM)	No	0.7959	0.4555
	Yes	0.7974	0.4547

5 Conclusions

In this project, we reproduced the xDeepFM model and trained and tested it on the Criteo dataset, achieving replication results similar to those in the original paper. Through ablation experiments, we verified the effectiveness of combining CIN and DNN to perform both explicit and implicit feature interactions. The experiment shows that after adding the numerical feature vectors obtained by copying numerical features into the input of the CIN module, the CIN can achieve better prediction results.

We proposed a relatively simple method for vectorizing numerical features. Better strategies will be left for future work.

6 Division of work

The workload is even for both members. Zheng Ke was responsible for preparing the slide, and Zhang Hexiao was responsible for writing the report. Hexiao proposed the mission of the project and we worked together to complete the proposal, coding, and experiments.

References

[1] Rendle, S. Factorization machines. In *2010 IEEE International conference on data mining*, pages 995–1000. IEEE, 2010.

[2] Liu, Q., F. Yu, S. Wu, et al. A convolutional click prediction model. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 1743–1746. 2015.

[3] Zhang, Y., H. Dai, C. Xu, et al. Sequential click prediction for sponsored search with recurrent neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28. 2014.

[4] Zhang, W., T. Du, J. Wang. Deep learning over multi-field categorical data. In *European conference on information retrieval*, pages 45–57. Springer, 2016.

[5] Guo, H., R. Tang, Y. Ye, et al. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[6] Wang, R., B. Fu, G. Fu, et al. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, pages 1–7. 2017.

[7] Lian, J., X. Zhou, F. Zhang, et al. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1754–1763. 2018.