Machine Learning with Graphs (MLG)

Homework 3 Report

吳承霖 成功大學電機所己組 N26090693

1 INTRODUCTION

Recommendation system is used different data mining techniques to generate meaningful suggestions to individual user or the group of users for items or products or elements that might interest them. This is the era of the web and online service; lots of data are available on the internet. On the online service, where the number of choices are overwhelming, so there is need of the information filtering on the service. Although many different approaches to recommender systems have been developed in past few years, the interest in this area is still high because growing demand of practical applications, which can deal with personalized recommendation and deal with large amount of overloaded data.

As a user has large amount of choices from these overloaded information, user only know the some of his/her relevant information, rest of relevant information he/she don't know. So to navigate the users according to their taste or preference the recommendation system comes into the picture.

In this homework, we use two datasets: LON-A and NYC-R for training and examining different recommendation system methods. LON-A is a dataset including tourist attractions information and tourists' user information in London; NYC-R is a dataset about restaurants and in New York City. Both datasets have many data columns which are valuable for predicting user-item interaction, we will further transform these data columns into features of model.

In this report, section 2 describes the methodology of all recommendation system methods including non-NN based approaches, NN based methods and our proposed NN method. Section 3 will illustrates the process of dataset handling, experiments of different sets of hyperparameters, and analysis of experiment results.

2 METHODOLOGY

In this section, we will briefly describe the architecture and math equations of all method we tried to predict interactions between users and items in datasets LON-A and NYC-R.

2.1 Baseline

Our baseline simply uses average rating and bias of user u and bias of item i to predict interaction between users and items.The equation of prediction is:

$$b_{ui} = (\mu + b_u + b_i)$$

Where b_{ui} is the predicted interaction from user u to item i, μ is average interaction score, b_u and b_i are bias for user u and item i respectively. The process of computing μ, b_u and b_i can be ML training, the optimization of training is:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - (\mu + b_u + b_i))^2 + \lambda(b_u^2 + b_i^2)$$

2.2 User-based CF using cosine similarity

User-based collaborative filtering using cosine similarity, or UCF-s in short, using top K similar users' rating as reference of predicting rating from user u to item i. UCF-s use cosine as method to calculate similarity between users, the similarity equation between user u and user v is:

$$sim(x_u, x_v) = cos(x_u, x_v) = \frac{x_u \bullet x_v}{|x_u||x_v|}$$

The equation of how prediction works is:

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} \text{sim}(i,j) \cdot r_{uj}}{\sum_{j \in N_u^k(i)} \text{sim}(i,j)}$$

Where $\widehat{r}_{ui}$ is our predicted interaction from user u to item i, k is the number of most similar users we want to reference.

2.3 User-based CF using pearson correlation

User-based collaborative filtering using pearson correlation, or UCF-p in short, using top K similar users' rating as reference of predicting rating from user u to item v. UCF-s use cosine as method to calculate similarity between users, the similarity equation is:

$$sim(u,\ v)\ =\ \frac{\sum_{i \in Suv}(r_{ui}-\overline{r_u})(r_{vi}-\overline{r_v})}{\sqrt{\sum_{i \in Suv}(r_{ui}-\overline{r_u})^2}\sqrt{\sum_{i \in Suv}(r_{vi}-\overline{r_v})^2}}$$

The equation of how prediction works is:

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} \text{sim}(i,j) \cdot r_{uj}}{\sum_{j \in N_u^k(i)} \text{sim}(i,j)}$$

Where $\widehat{r}_{ui}$ is our predicted interaction from user u to item i, k is the number of most similar users we want to reference.

2.4 Item-based CF using cosine similarity

Item-based collaborative filtering using cosine similarity, or ICF-s in short, using top K similar items' rating as reference of predicting rating from user to item. UCF-s use cosine as method to calculate similarity between users, the similarity equation between item u and item v is:

$$sim(x_u,\ x_v)\ = cos(x_u,\ x_v)\ =\ \frac{x_u \bullet x_v}{|x_u||x_v|}$$

The equation of how prediction works is:

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} \text{sim}(i,j) \cdot r_{uj}}{\sum_{j \in N_u^k(i)} \text{sim}(i,j)}$$

Where $\widehat{r}_{ui}$ is our predicted interaction from user u to item i, k is the number of most similar users we want to reference.

2.5 Item-based CF using pearson correlation

Item-based collaborative filtering using pearson correlation, or ICF-p in short, using top K similar users' rating as reference of predicting rating from user u to item v. UCF-s use cosine as method to calculate similarity between items, the similarity equation is:

$$sim(u,\ v)\ =\ \frac{\sum_{i \in Suv} (r_{ui} - \overline{r_u})(r_{vi} - \overline{r_v})}{\sqrt{\sum_{i \in Suv} (r_{ui} - \overline{r_u})^2}\sqrt{\sum_{i \in Suv} (r_{vi} - \overline{r_v})^2}}$$

The equation of how prediction works is:

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} \text{sim}(i,j) \cdot r_{uj}}{\sum_{j \in N_u^k(i)} \text{sim}(i,j)}$$

Where $\widehat{r}_{ui}$ is our predicted interaction from user u to item i, k is the number of most similar users we want to reference.

2.6 Matrix Factorization

Matrix factorization models map both users and items to a joint latent factor space of dimensionality f, such that user-item interactions are modeled as inner products in that space. Accordingly, each item i is associated with a vector $q_i \in R^f$, and each user u is associated with a vector $p_u \in R^f$. For a given item i, the elements of q_i measure the extent to which the item possesses those factors, positive or negative. For a given user u, the elements of p_u measure the extent of interest the user has in items that are high on the corresponding factors, again, positive or negative.

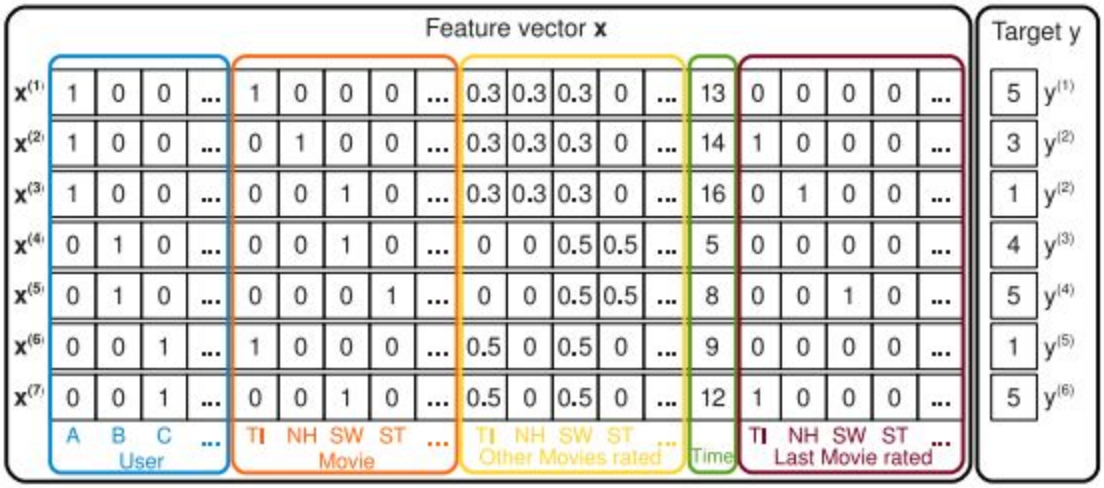
The resulting dot product, $q_i^T p_u$, captures the interaction between user u and item i —the user's overall interest in the item's characteristics. This approximates user u's rating of item i, which is denoted by rui, leading to the estimate

$$\hat{r}_{ui} = q_i^T p_u$$

2.7 Factorization Machine

Factorization Machines enables us to utilize explicit and implicit user/item features as input to predict interaction. FMs allow parameter estimation under very sparse data where SVMs fail. The model equation for a factorization machine of degree $d = 2$ is defined as :

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



The third term provides the ability of interaction between features, for example: feature i is user's country and feature j is item's manufactor. $< v_i, v_j >$ models the interaction between features.

2.8 Factorization Machine-supported Neural Network

Factorization Machine-supported Neural Network, or FNN in short, starts to add DNN layers to prediction. After we transform sparse categorical features into embedded dense feature, we use fully-connected layers to predict interactions between user and item. Unlike Factorization

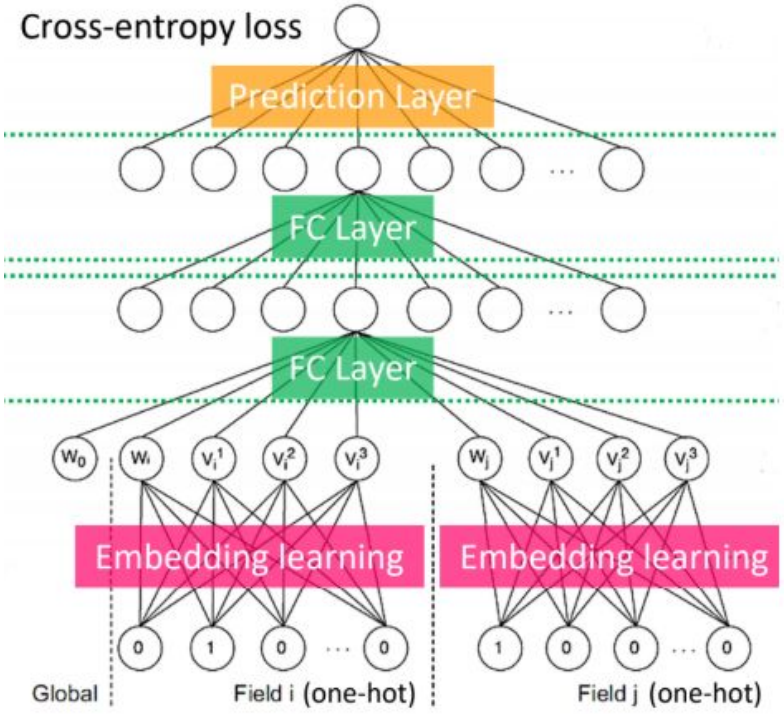
Machine, which use inner-product to handle feature interaction. The interaction of inter-features is:

$$layer1 = activation(W_1 z + b_1)$$

which z is the concatenation of all embed features:

$$z_i = (w_i, v_i^1, v_i^2, ..., v_i^K)$$

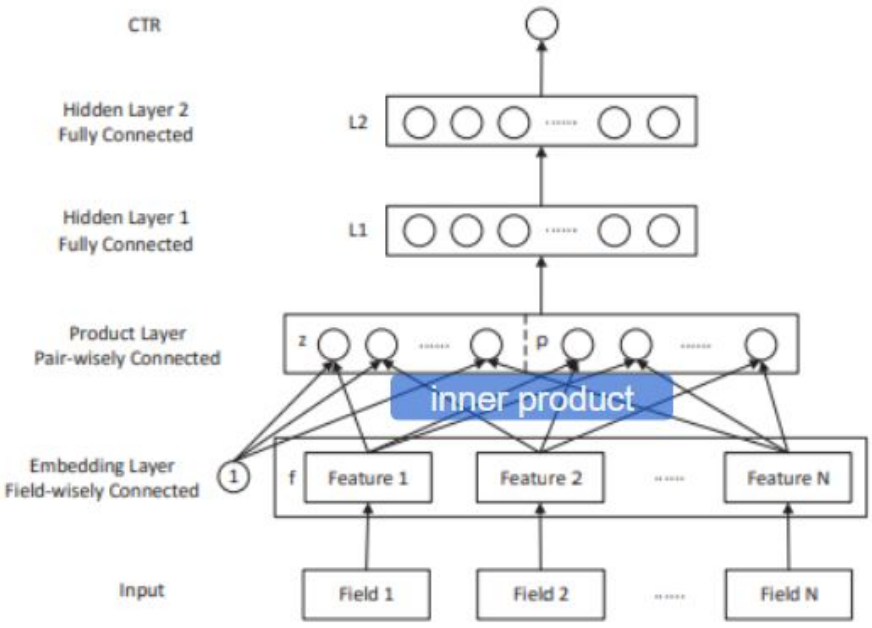
The model architecture is:



2.9 Inner Product Neural Network

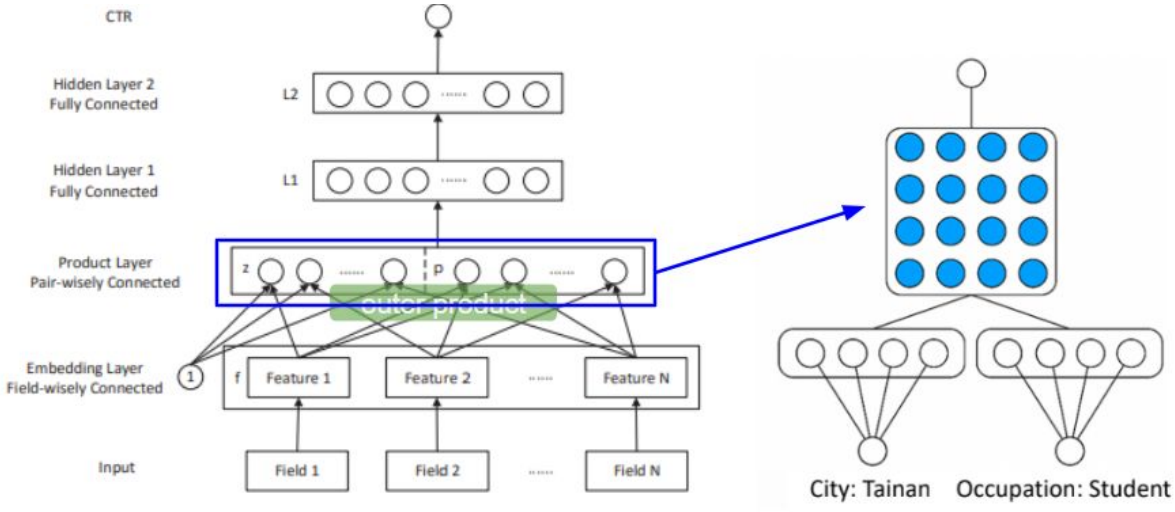
Inner Product Neural Network or IPNN for short, is similar to FNN, which take embedded vector of sparse feature as DNN's input. The only difference between PNN and FNN is that PNN uses element-wise product to implement interaction of features (embedded vectors), and FNN use fully-connected layer to aggregate feature vectors.

This is model architecture of IPNN:



2.10 Outer Product Neural Network

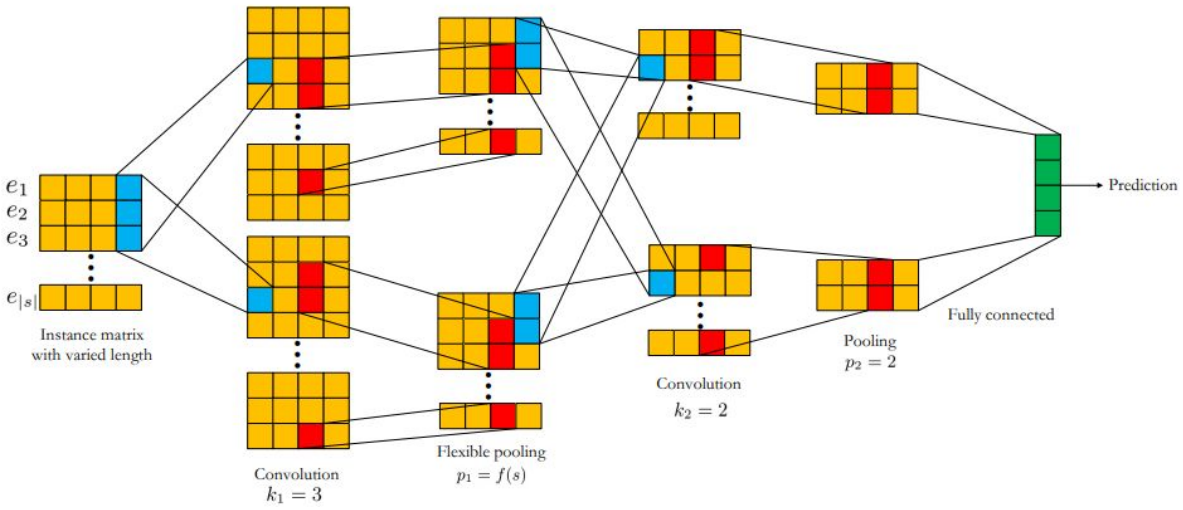
Outer Product Neural Network or OPNN for short, is very similar to IPNN except that OPNN uses outer product instead of inner product to compute element-wise feature vector interactions. Different from inner product, outer product will generate a matrix instead of a vector, so we have to further flatten the matrix into 1d vector in order to pass interacted features into DNN.



2.11 Convolutional Click Prediction Model

Convolutional Click Prediction Model (CCPM) is based on FNN, it transform sparse features into dense embedded vector in the beginning. But CCPM also adopt the concept of Convolutional Neural Network (CNN). In the model architecture, it perform convolution on embedded feature vectors and generate max-pooled feature maps, then we flatten feature maps into 1d vector as inputs of DNN.

Model architecture:

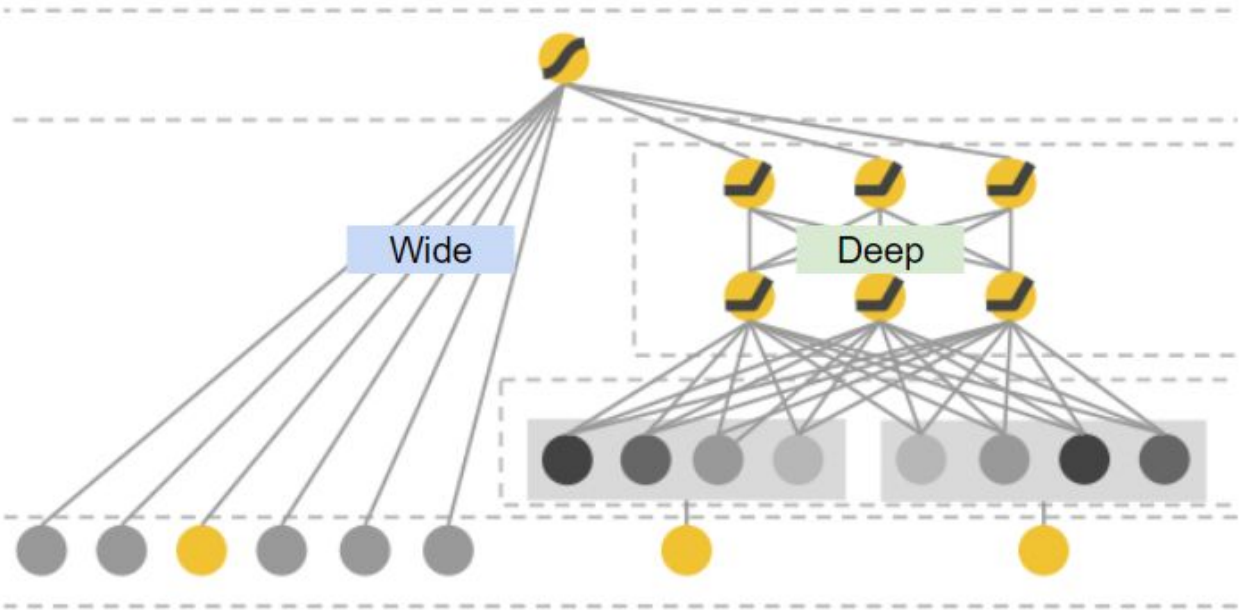


The convolutional & max-pooling layers are located between feature embedding and fully-connected network (DNN).

2.12 Wide & Deep

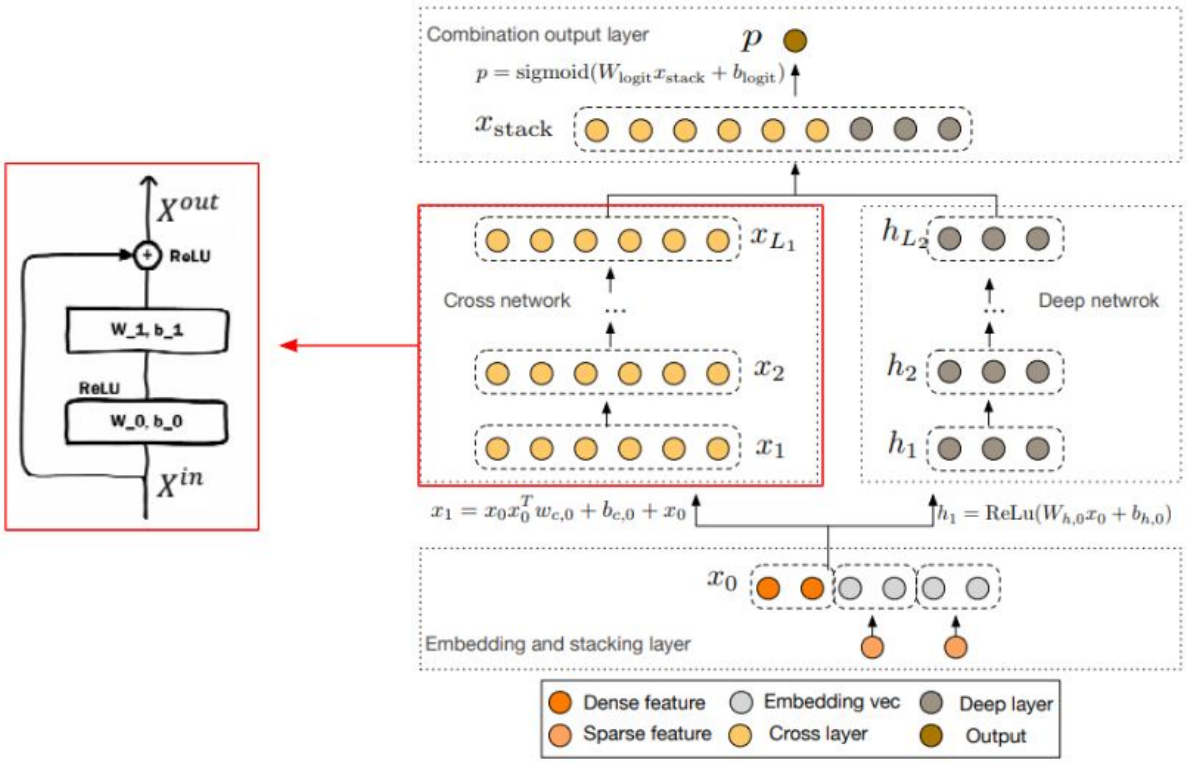
Wide & Deep Model (WD) is a combination of wide neural network and deep neural network. It has advantages from both architectures. Wide model accept many features as input but its network depth is shallower, it provides better ability of generalization because of higher variety (longer) inputs. On the other hand, Deep model accepts fewer (short) features as input but its depth of network is higher, it provides good ability to learn high order feature interactions.

The following figure is the architecture of Wide & Deep, which is simple the combination of two opposite types of network model:



2.13 Deep & Cross

Architecture of Deep & Cross Network:



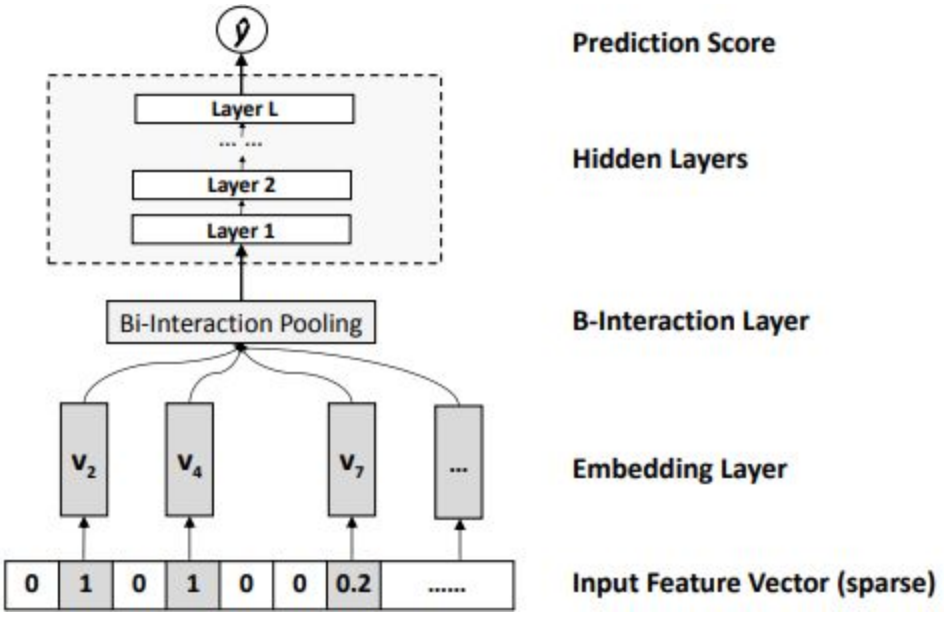
Deep & Cross Network (DCN) is also a combination of different network models, which provide different aspects of advantages. After the features being embedded, they will be pass to Cross

Network and Deep Network respectively. The cross network is a residual network which can deal with overfitting and avoid gradient vanish. The deep network provide the ability of learning high order of feature interactions.

2.14 Neural Factorization Machine

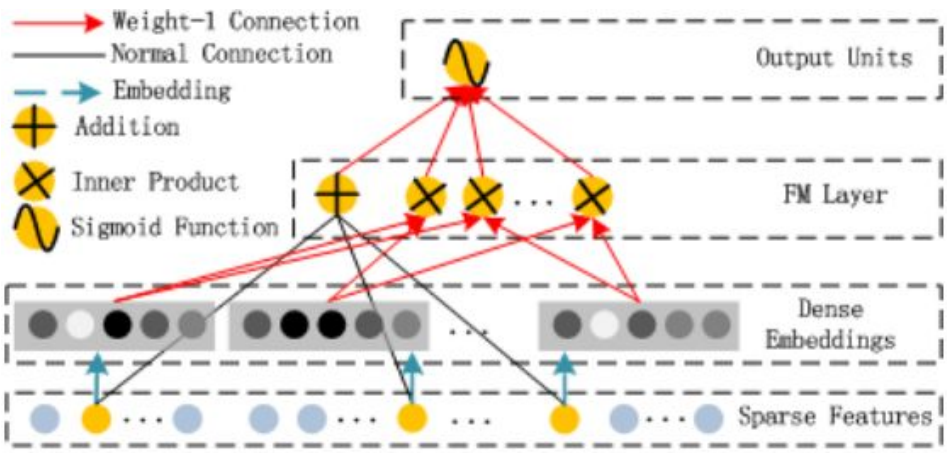
Neural Factorization Machine (NFM) is a network model based on FNN, but add a Bi-interaction layer between feature embedding and DNN. In the Bi-Interaction Layer, We then feed the embedding set V_x into the Bi-Interaction layer, which is a pooling operation that converts a set of embedding vectors to one vector:

$$f_{BI}(V_x) = \sum_{i=1} \sum_{j=i+1} x_i v_i \odot x_j v_j$$

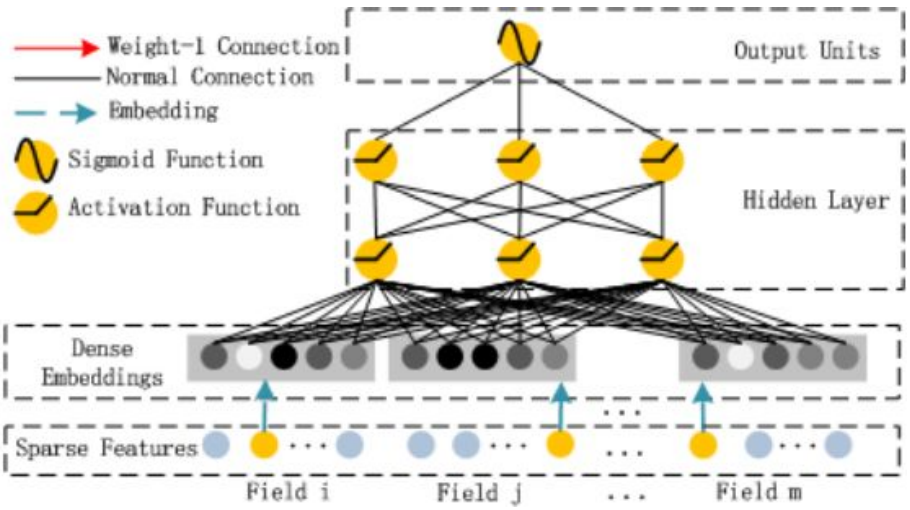


2.15 Deep Factorization Machine

Deep Factorization Machine (DeepFM) combines FM architecture and Depp Neural Network. In the FM part, it use inner product layer to simulate the feature interaction in Factorization machine,It can capture order-2 feature interactions effectively especially when the dataset is sparse. The model architecture is illustrated in the following figure:



The deep component is a feed-forward neural network, which is used to learn high-order feature interactions.



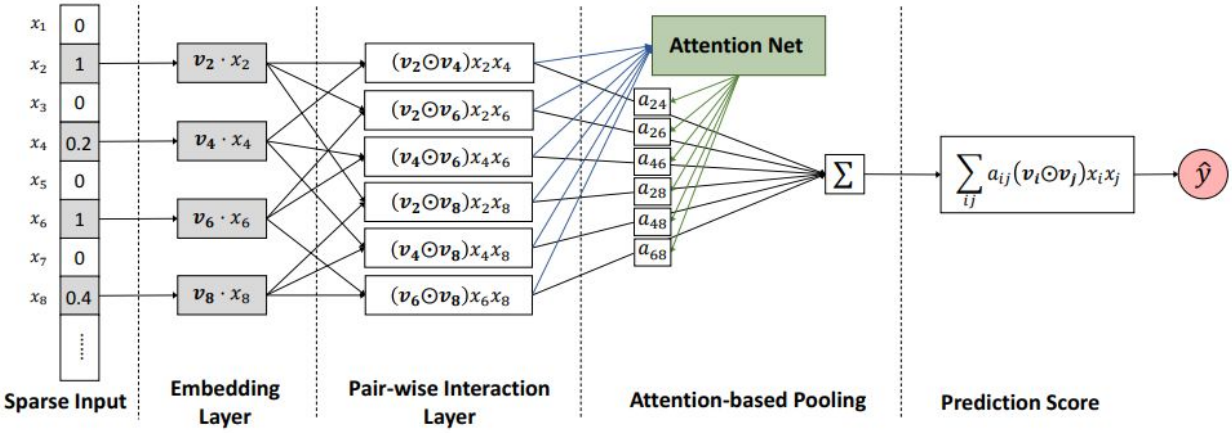
2.16 Attentional Factorization Machines

Atentional Factorization Machine (AFM) is based on Producted-based Neural Network (e.g. IPNN), AFM further incorporate attention layer on top of pair-wise interaction layer.
The idea of attention is to allow different parts contribute differently when compressing them to a single representation. Motivated by the drawback of FM, it employs the attention mechanism on feature interactions by performing a weighted sum on the interacted vectors:

$$f_{Att}(f_{PI}(\mathcal{E})) = \sum_{(i,j)\in\mathcal{R}_x} a_{ij}(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j,$$

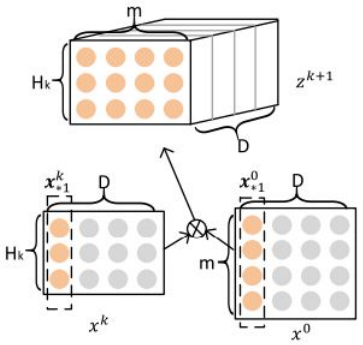
where a_{ij} is the attention score for feature interaction $\widehat{w}_{ij}$, which can be interpreted as the importance of $\widehat{w}_{ij}$ in predicting the target.

The following figure is model architecute of AFM:

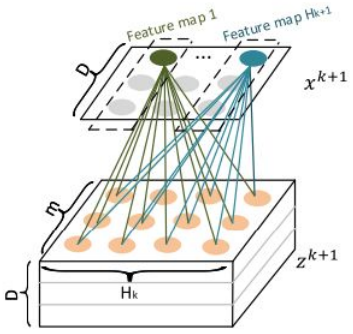


2.17 xDeepFM

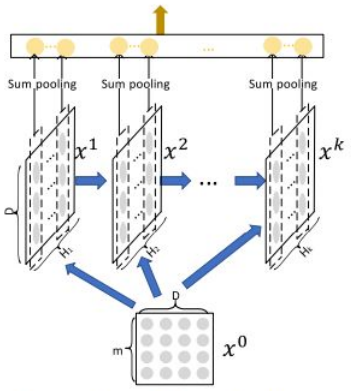
xDeepFM is an advanced model network based on DeepFM, which is also a combination of DNN part and FM part. xDeepFM introduce a new network component called Compressed Interaction Network(CIN), which aims to generate feature interactions in an explicit fashion and at the vector-wise level. Besides, CIN shares some functionalities with convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The following figure describes the function of CIN:



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

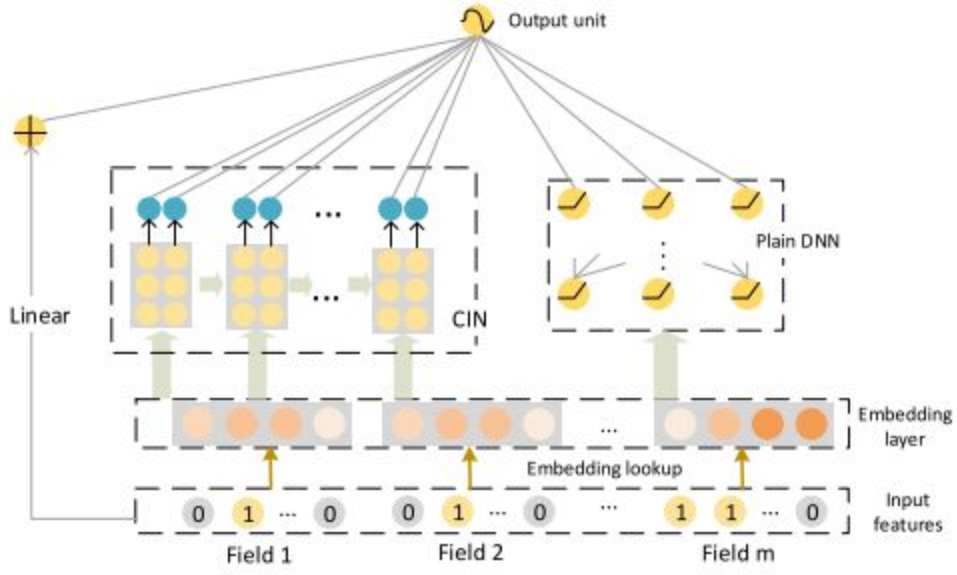


(b) The k-th layer of CIN. It compresses the intermediate tensor Z^{k+1} to H_{k+1} embedding vectors (aslo known as *feature maps*).



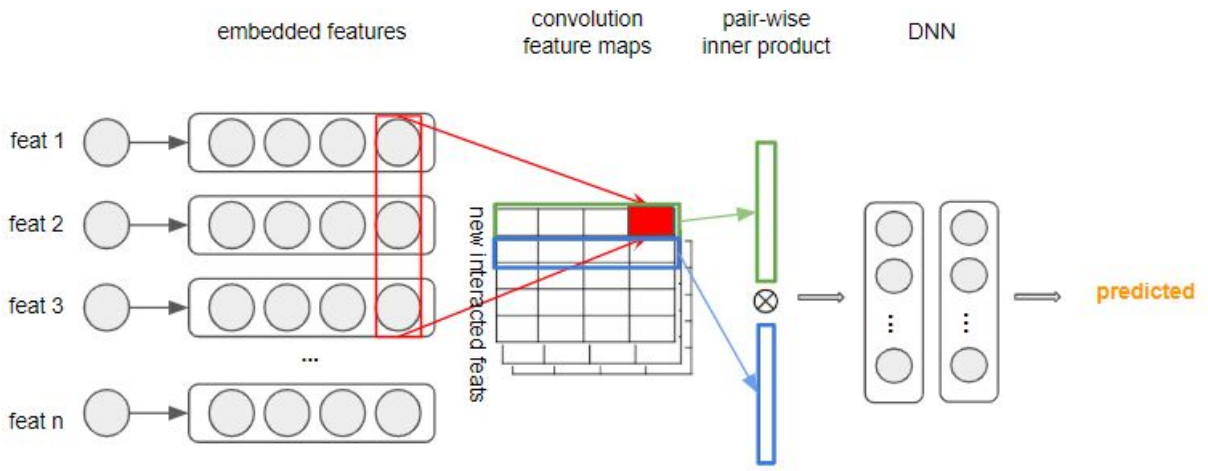
(c) An overview of the CIN architecture.

And the whole picture of xDeepFM network model:



2.18 Proposed Model: ConvIPNN

Convolutional Inner Product Neural Network (ConvIPNN), is a combination of product-based network and CCPM. Here is the architecture of ConvIPNN:



The sparse features are transformed into embedded vectors in the beginning of network, just like other NN-based methods. After that, we perform 1d convolution on those embedded feature matrix and obtain several feature maps. The number of feature map is determined by the number of convolution filters. After convolution, we will get new features which are the result of feature interaction (using convolution), then we concat the feature vectors of all features:

$$conv\ feat\ 1\ =\ [< conv\ feat\ 1\ in\ map 1 >, < conv\ feat\ 1\ in\ map 2 >, \ldots, < conv\ feat\ 1\ in\ map M >]$$

In the pairwise inner product layer, we conduct inner product on all pairs of concatenated convoluted features. The rest of model operation is identical to IPNN.

3 EXPERIMENTAL ANALYSIS

3.1 Hyperparameter Experiments

3.1.1 Baseline

Test on different training epochs and regularization factors:

LON-A

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
lr=5e-3, epoch=100, reg=0	0.99829	0.04770	0.99973	0.99727	0.06989	0.99989
lr=5e-3, epoch=150, reg=0	0.99837	0.04543	0.99974	0.99741	0.06559	0.99989
lr=5e-3, epoch=200, reg=0*	0.99837	0.04655	0.99975	0.99748	0.06342	0.99990
lr=5e-3, epoch=200, reg=0.02	0.99787	0.05517	0.99964	0.99714	0.07020	0.99986
lr=5e-3, epoch=200, reg=0.04	0.99784	0.06068	0.99952	0.99711	0.07267	0.99982
lr=5e-3, epoch=200, reg=0.06	0.99781	0.06643	0.99939	0.99708	0.07520	0.99977

NYC-R

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
lr=5e-3, epoch=100, reg=0	0.99868	0.13270	1.00000	0.99873	0.14736	0.99939
lr=5e-3, epoch=150, reg=0	0.99874	0.10398	0.99950	0.99893	0.11389	0.99958
lr=5e-3, epoch=200, reg=0*	0.99880	0.08598	0.99959	0.99901	0.09440	0.99966
lr=5e-3, epoch=200, reg=0.02	0.99958	0.02471	1.00000	0.99924	0.02754	0.99936
lr=5e-3, epoch=200, reg=0.04	0.99955	0.03267	1.00000	0.99922	0.03440	0.99942
lr=5e-3, epoch=200, reg=0.06	0.99952	0.04055	1.00000	0.99918	0.04218	0.99929

3.1.2 UCF-s

Test on different K (amount of similar candidates):

LON-A:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
k=3*	0.49596	2.91065	0.91598	0.50762	2.85695	0.91625
k=4	0.48061	2.96409	0.91064	0.48836	2.85471	0.90929
k=5	0.47056	2.97350	0.90730	0.47924	2.84698	0.90612
k=6	0.45843	3.00391	0.90385	0.47298	2.83998	0.90377
k=7	0.45564	2.98360	0.90260	0.46843	2.83392	0.90174
k=8	0.44970	2.98132	0.90084	0.46186	2.82954	0.89948
k=9	0.44448	2.98455	0.89912	0.45849	2.82626	0.89813
k=10	0.44197	2.97846	0.89787	0.45565	2.82376	0.89705

NYC-R:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
k=3	0.87087	0.19725	0.99563	0.95051	0.10281	0.99832
k=4	0.89208	0.19477	0.99567	0.95869	0.09371	0.99834
k=5	0.89979	0.19321	0.99570	0.96278	0.09050	0.99835
k=10	0.92028	0.19175	0.99571	0.96951	0.08984	0.99835
k=15	0.92553	0.19097	0.99572	0.97259	0.08953	0.99843
k=20	0.92832	0.19074	0.99572	0.97342	0.08694	0.99843
k=25	0.92922	0.19064	0.99572	0.97386	0.08687	0.99843
k=30*	0.93009	0.19058	0.99572	0.97404	0.08684	0.99843

3.1.3 UCF-p

Test on different K (amount of similar candidates):

LON-A:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
k=3*	0.49558	2.35525	0.93030	0.51150	2.29329	0.93262
k=4	0.48577	2.40694	0.92655	0.49634	2.38645	0.92540
k=5	0.47602	2.43115	0.92370	0.48876	2.43786	0.92147
k=6	0.46365	2.48746	0.91964	0.48159	2.47573	0.91847
k=7	0.45771	2.51404	0.91761	0.47709	2.50212	0.91615
k=8	0.45146	2.53207	0.91576	0.47356	2.50206	0.91463
k=9	0.44490	2.54086	0.91427	0.46937	2.50618	0.91340
k=10	0.44219	2.54377	0.91324	0.46620	2.51697	0.91228

NYC-R:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
k=3*	0.89886	0.16942	0.99645	0.95404	0.09448	0.99840
k=4	0.91065	0.16746	0.99648	0.96015	0.09351	0.99834
k=5	0.92134	0.16633	0.99649	0.96459	0.09288	0.99835
k=10	0.93046	0.17127	0.99631	0.97048	0.09207	0.99836
k=15	0.93871	0.17063	0.99631	0.97200	0.09178	0.99836
k=20	0.94162	0.17032	0.99631	0.97317	0.09163	0.99836
k=25	0.94363	0.17008	0.99631	0.97336	0.09161	0.99836
k=30*	0.94394	0.17005	0.99631	0.97359	0.09161	0.99836

3.1.4 ICF-s

Test on different K (amount of similar candidates):

LON-A:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
k=3*	0.49650	2.55242	0.92808	0.52433	2.12324	0.93968
k=4	0.48705	2.49657	0.92450	0.51455	2.13440	0.93399
k=5	0.47963	2.53492	0.92124	0.50672	2.17230	0.93057
k=6	0.46985	2.59390	0.91747	0.50322	2.18739	0.92855
k=7	0.46930	2.58139	0.91665	0.49993	2.20220	0.92678
k=8	0.46727	2.59588	0.91511	0.49648	2.19766	0.92571
k=9	0.46239	2.61721	0.91331	0.49478	2.19797	0.92488
k=10	0.46077	2.62906	0.91217	0.49323	2.19956	0.92423

NYC-R:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
k=3*	0.91139	0.16775	0.99766	0.95093	0.10484	0.99840
k=4	0.92084	0.17893	0.99748	0.95788	0.10601	0.99835
k=5	0.93513	0.17126	0.99769	0.96241	0.10535	0.99835
k=10	0.94623	0.17604	0.99751	0.96898	0.10650	0.99829
k=15	0.95043	0.17552	0.99751	0.97160	0.10380	0.99836
k=20	0.95278	0.17531	0.99751	0.97253	0.10369	0.99836
k=25*	0.95373	0.17515	0.99751	0.97292	0.10363	0.99837
k=30	0.95372	0.17515	0.99751	0.97329	0.10359	0.99836

3.1.5 ICF-p

Test on different K (amount of similar candidates):

LON-A

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
k=3*	0.48745	2.95258	0.91355	0.50680	2.84786	0.91566
k=4	0.47470	2.96913	0.90896	0.48928	2.86664	0.90873
k=5	0.46689	2.99067	0.90563	0.47904	2.88459	0.90487
k=6	0.45574	3.00328	0.90251	0.47158	2.88702	0.90186
k=7	0.44457	3.02319	0.89955	0.46571	2.88416	0.89949
k=8	0.44071	3.01225	0.89829	0.46105	2.87906	0.89776
k=9	0.43548	3.01783	0.89643	0.45869	2.86922	0.89669
k=10	0.43257	3.02031	0.89505	0.45468	2.87365	0.89523

NYC-R

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
k=3*	0.93551	0.09610	0.99781	0.96075	0.07931	0.99863
k=4	0.94056	0.10807	0.99764	0.96564	0.08325	0.99858
k=5	0.94329	0.10755	0.99765	0.96767	0.08523	0.99851
k=10	0.95458	0.10665	0.99766	0.97154	0.08684	0.99844
k=15	0.95794	0.10654	0.99766	0.97221	0.08907	0.99837
k=20	0.95985	0.10636	0.99766	0.97306	0.08896	0.99837
k=25	0.96112	0.10622	0.99766	0.97350	0.08891	0.99837
k=26*	0.96112	0.10622	0.99766	0.97356	0.08890	0.99837
k=30	0.96111	0.10624	0.99766	0.97479	0.08644	0.99845

3.1.6 MF

test on defferent latent vector size:

LON-A:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
factor_n=2*	0.99841	0.05098	1.00000	0.99739	0.06513	0.99985
factor_n=6	0.99820	0.05361	0.99962	0.99736	0.06490	1.00000
factor_n=10	0.99774	0.05501	0.99964	0.99746	0.06523	1.00000
factor_n=14	0.99815	0.05452	0.99965	0.99736	0.06574	0.99987
factor_n=18	0.99800	0.05710	0.99931	0.99726	0.06809	0.99987
factor_n=20	0.99809	0.05508	0.99966	0.99708	0.07104	0.99949
factor_n=40	0.99790	0.05916	0.99937	0.99737	0.06787	1.00000
factor_n=60	0.99773	0.06047	0.99939	0.99711	0.07147	0.99989
factor_n=80	0.99784	0.05947	0.99939	0.99667	0.07446	0.99989
factor_n=100	0.99803	0.05862	0.99970	0.99660	0.07951	0.99957

NYC-R:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
factor_n=2*	0.99937	0.09109	1.00000	0.99914	0.09920	0.99959
factor_n=6	0.99939	0.09199	1.00000	0.99901	0.10006	0.99980
factor_n=10	0.99925	0.09445	1.00000	0.99898	0.10414	0.99924
factor_n=14	0.99920	0.09580	1.00000	0.99904	0.10371	0.99963
factor_n=18	0.99927	0.09664	1.00000	0.99892	0.10711	0.99927

3.1.7 FM

test on different latent vector size:

LON-A:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
latent_n=1*	0.50749	2.67413	0.66084	0.49727	2.15446	0.66084
latent_n=2	0.50484	2.79746	0.66084	0.49862	2.95496	0.44685
latent_n=3	0.50840	3.45294	0.66084	0.50411	3.46178	0.27727
latent_n=4	0.49764	3.70850	0.66084	0.49985	3.64026	0.27727
latent_n=5	0.48261	3.67404	0.66084	0.48091	3.81011	0.27727
latent_n=6	0.50135	4.22715	0.66084	0.49615	4.19156	0.27727
latent_n=7	0.50788	4.42946	0.66084	0.50081	4.46515	0.27727
latent_n=8	0.49772	4.40505	0.66084	0.48850	4.60153	0.27727
latent_n=9	0.50088	4.71791	0.66084	0.49938	4.48787	0.27727
latent_n=10	0.50002	4.66355	0.66084	0.49069	4.69170	0.27727

3.1.8 BPR-FM

test on different latent vector size:

LON-A:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
latent_n=1*	0.50749	2.67413	0.66084	0.49727	2.15446	0.66084
latent_n=2	0.50484	2.79746	0.66084	0.49862	2.95496	0.44685
latent_n=3	0.50840	3.45294	0.66084	0.50411	3.46178	0.27727
latent_n=4	0.49764	3.70850	0.66084	0.49985	3.64026	0.27727
latent_n=5	0.48261	3.67404	0.66084	0.48091	3.81011	0.27727
latent_n=6	0.50135	4.22715	0.66084	0.49615	4.19156	0.27727
latent_n=7	0.50788	4.42946	0.66084	0.50081	4.46515	0.27727
latent_n=8	0.49772	4.40505	0.66084	0.48850	4.60153	0.27727
latent_n=9	0.50088	4.71791	0.66084	0.49938	4.48787	0.27727
latent_n=10	0.50002	4.66355	0.66084	0.49069	4.69170	0.27727

NYC-R:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
latent_n=1	0.47406	0.51291	1.00000	0.48215	0.50910	1.00000
latent_n=2	0.46631	0.51991	1.00000	0.45406	0.53953	1.00000
latent_n=3	0.50847	0.53254	1.00000	0.52686	0.54300	1.00000
latent_n=4	0.43753	0.53086	1.00000	0.43093	0.56579	1.00000
latent_n=5	0.50629	0.54420	1.00000	0.52933	0.55445	1.00000
latent_n=6	0.48671	0.54022	1.00000	0.49444	0.55589	1.00000
latent_n=7*	0.53060	0.53454	1.00000	0.55349	0.55589	1.00000
latent_n=8	0.51346	0.54491	1.00000	0.53972	0.56674	1.00000
latent_n=9	0.49840	0.54490	1.00000	0.51439	0.56831	1.00000
latent_n=10	0.50420	0.54695	1.00000	0.51831	0.57180	1.00000

3.1.9 FNN

test on different DNN architectures

LON-A:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
DNN architect=(32, 32)	0.99775	0.04568	1.00000	0.99647	0.05671	1.00000
DNN architect=(32, 32, 32)	0.99790	0.04657	1.00000	0.99639	0.05830	1.00000
DNN architect=(64, 64)	0.99807	0.04089	1.00000	0.99676	0.05516	1.00000
DNN architect=(64, 64, 64)	0.99813	0.04229	1.00000	0.99669	0.05547	1.00000
DNN architect=(96, 96)	0.99819	0.03794	1.00000	0.99690	0.05491	1.00000
DNN architect=(96, 96, 96)*	0.99825	0.04304	1.00000	0.99714	0.06077	1.00000
DNN architect=(128, 128)	0.99822	0.04063	1.00000	0.99704	0.05880	1.00000
DNN architect=(128, 128, 128)	0.99817	0.04251	1.00000	0.99718	0.05730	1.00000

NYC-R:

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
DNN architect=(32, 32)	0.99962	0.02681	1.00000	0.99942	0.02774	1.00000
DNN architect=(32, 32, 32)	0.99961	0.02188	1.00000	0.99949	0.02285	1.00000
DNN architect=(64, 64)	0.99968	0.02048	1.00000	0.99949	0.02152	1.00000
DNN architect=(64, 64, 64)	0.99969	0.01696	1.00000	0.99952	0.01756	1.00000
DNN architect=(96, 96)	0.99975	0.01547	1.00000	0.99946	0.01713	1.00000
DNN architect=(96, 96, 96)*	0.99980	0.01763	1.00000	0.99953	0.02030	1.00000
DNN architect=(128, 128)	0.99976	0.01351	1.00000	0.99952	0.01477	1.00000
DNN architect=(128, 128, 128)	0.99968	0.01804	1.00000	0.99946	0.02156	1.00000

3.1.10 IPNN

test on different DNN architectures

LON-A

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
DNN architect=(32, 32)	0.99753	0.05496	1.00000	0.99559	0.06264	1.00000
DNN architect=(32, 32, 32)	0.99521	0.04754	1.00000	0.99528	0.05601	1.00000
DNN architect=(64, 64)*	0.99793	0.03787	1.00000	0.99618	0.05313	1.00000
DNN architect=(64, 64, 64)	0.99506	0.05079	1.00000	0.99520	0.05524	1.00000
DNN architect=(96, 96)	0.99772	0.03786	1.00000	0.99562	0.05280	1.00000
DNN architect=(96, 96, 96)	0.99767	0.03806	1.00000	0.99565	0.05702	1.00000
DNN architect=(128, 128)	0.99758	0.03868	1.00000	0.99552	0.05811	1.00000
DNN architect=(128, 128, 128)	0.99497	0.05242	1.00000	0.99522	0.05930	1.00000

NYC-R

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
DNN architect=(32, 32)	0.99968	0.04766	1.00000	0.99946	0.04628	1.00000
DNN architect=(32, 32, 32)	0.99839	0.05418	1.00000	0.99781	0.05453	1.00000
DNN architect=(64, 64)*	0.99972	0.02149	1.00000	0.99954	0.02261	1.00000
DNN architect=(64, 64, 64)	0.99869	0.01736	1.00000	0.99826	0.02169	1.00000
DNN architect=(96, 96)	0.99974	0.01450	1.00000	0.99953	0.01626	1.00000
DNN architect=(96, 96, 96)	0.99922	0.01917	1.00000	0.99867	0.02264	1.00000
DNN architect=(128, 128)	0.99967	0.01366	1.00000	0.99953	0.01447	1.00000
DNN architect=(128, 128, 128)	0.99849	0.01856	1.00000	0.99790	0.02328	1.00000

3.1.11 OPNN

test on different DNN architectures

LON-A

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
DNN architect=(32, 32)*	0.99755	0.05056	1.00000	0.99554	0.05969	1.00000
DNN architect=(32, 32, 32)	0.99518	0.04890	1.00000	0.99525	0.05595	1.00000
DNN architect=(64, 64)	0.99752	0.04496	1.00000	0.99558	0.05462	1.00000
DNN architect=(64, 64, 64)	0.99753	0.04198	1.00000	0.99556	0.05561	1.00000
DNN architect=(96, 96)	0.99751	0.04127	1.00000	0.99548	0.05869	1.00000
DNN architect=(96, 96, 96)	0.99495	0.05071	1.00000	0.99518	0.05601	1.00000
DNN architect=(128, 128)	0.99749	0.04235	1.00000	0.99530	0.05503	1.00000
DNN architect=(128, 128, 128)	0.99726	0.04747	1.00000	0.99555	0.06212	1.00000

NYC-R

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
DNN architect=(32, 32)	0.99967	0.04243	1.00000	0.99950	0.04147	1.00000
DNN architect=(32, 32, 32)	0.99956	0.03644	1.00000	0.99949	0.03650	1.00000
DNN architect=(64, 64)	0.99971	0.02154	1.00000	0.99950	0.02162	1.00000
DNN architect=(64, 64, 64)	0.99966	0.01005	1.00000	0.99950	0.01152	1.00000
DNN architect=(96, 96)	0.99961	0.01656	1.00000	0.99956	0.01736	1.00000
DNN architect=(96, 96, 96)	0.99831	0.02369	1.00000	0.99792	0.02632	1.00000
DNN architect=(128, 128)*	0.99988	0.01344	1.00000	0.99956	0.01462	1.00000
DNN architect=(128, 128, 128)	0.99844	0.02213	1.00000	0.99783	0.02513	1.00000

3.1.12 CCPM

test on different sets of filter number and kernel sizes.

LON-A

using parameter filter number=(4, 4)

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
kernel_width=(3, 2)	0.99783	0.03751	1.00000	0.99711	0.04793	1.00000
kernel_width=(4, 3)	0.99862	0.03296	1.00000	0.99787	0.04522	1.00000
kernel_width=(5, 4)	0.99882	0.03383	1.00000	0.99789	0.04574	1.00000
kernel_width=(6, 5)*	0.99889	0.03266	1.00000	0.99800	0.04496	1.00000
kernel_width=(7, 6)	0.99879	0.03561	1.00000	0.99793	0.04865	1.00000
kernel_width=(8, 7)	0.99879	0.03273	1.00000	0.99805	0.04551	1.00000

using parameter kernel_width=(6, 5)

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
filter number=(3, 3)	0.99858	0.03723	1.00000	0.99784	0.04854	1.00000
filter number=(4, 4)	0.99855	0.03565	1.00000	0.99785	0.04736	1.00000
filter number=(5, 5)*	0.99860	0.03354	1.00000	0.99801	0.04456	1.00000

NYC-R

using parameter filter number=(4, 4)

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
kernel_width=(3, 2)	0.99964	0.01277	1.00000	0.99942	0.01326	1.00000
kernel_width=(4, 3)	0.99978	0.03971	1.00000	0.99942	0.04036	1.00000
kernel_width=(5, 4)	0.99979	0.01082	1.00000	0.99952	0.01271	1.00000
kernel_width=(6, 5)	0.99969	0.01020	1.00000	0.99950	0.01032	1.00000
kernel_width=(7, 6)	0.99968	0.01248	1.00000	0.99957	0.01190	1.00000
kernel_width=(8, 7)*	0.99987	0.01024	1.00000	0.99958	0.01012	1.00000

using parameter kernel_width=(6, 5)

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
filter number=(3, 3)	0.99970	0.01155	1.00000	0.99960	0.01190	1.00000
filter number=(4, 4)*	0.99978	0.00911	1.00000	0.99953	0.00992	1.00000
filter number=(5, 5)	0.99974	0.00945	1.00000	0.99957	0.00985	1.00000

3.1.13 WD

test on different sets of DNN architectures

LON-A

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
DNN architect=(256, 128)	0.99825	0.04198	1.00000	0.99709	0.05845	1.00000
DNN architect=(256, 128, 64)	0.99830	0.04765	1.00000	0.99718	0.05913	1.00000
DNN architect=(256, 128, 64, 32)*	0.99833	0.04838	1.00000	0.99697	0.06067	1.00000
DNN architect=(128, 64)	0.99817	0.03859	1.00000	0.99694	0.05366	1.00000
DNN architect=(128, 64, 32)	0.99820	0.04680	1.00000	0.99671	0.05701	1.00000
DNN architect=(128, 64, 32, 16)	0.99815	0.05029	1.00000	0.99673	0.05715	1.00000

NYC-R

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
DNN architect=(256, 128)*	0.99980	0.01379	1.00000	0.99951	0.01376	1.00000
DNN architect=(256, 128, 64)	0.99975	0.01462	1.00000	0.99953	0.01483	1.00000
DNN architect=(256, 128, 64, 32)	0.99980	0.02749	1.00000	0.99948	0.03082	0.99964
DNN architect=(128, 64)	0.99969	0.01678	1.00000	0.99951	0.01713	1.00000
DNN architect=(128, 64, 32)	0.99974	0.02485	1.00000	0.99951	0.02525	1.00000
DNN architect=(128, 64, 32, 16)	0.99976	0.03460	1.00000	0.99950	0.03668	1.00000

3.1.14 DeepCross

test on different cross layers

LON-A

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
cross layers=1*	0.99876	0.03500	1.00000	0.99761	0.05212	1.00000
cross layers=2	0.99857	0.03885	1.00000	0.99779	0.05350	1.00000
cross layers=3	0.99859	0.04096	1.00000	0.99780	0.04875	1.00000
cross layers=4	0.99818	0.04478	1.00000	0.99763	0.05203	1.00000
cross layers=5	0.99863	0.03842	1.00000	0.99794	0.04771	1.00000

NYC-R

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
cross layers=1	0.99972	0.01144	1.00000	0.99952	0.01305	1.00000
cross layers=2*	0.99980	0.01108	1.00000	0.99950	0.01214	1.00000
cross layers=3	0.99973	0.01034	1.00000	0.99952	0.01188	1.00000
cross layers=4	0.99972	0.01118	1.00000	0.99952	0.01182	1.00000
cross layers=5	0.99977	0.00998	1.00000	0.99959	0.01225	1.00000

3.1.15 NFM

test on different bi-interaction dropout factors

LON-A

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
bi_dropout=0	0.99772	0.04934	1.00000	0.99620	0.05969	1.00000
bi_dropout=0.05	0.99765	0.05005	1.00000	0.99622	0.06218	1.00000
bi_dropout=0.1*	0.99854	0.03656	1.00000	0.99777	0.05171	1.00000
bi_dropout=0.15	0.99824	0.03783	1.00000	0.99734	0.05544	1.00000
bi_dropout=0.2	0.99777	0.04902	1.00000	0.99627	0.06209	1.00000

NYC-R

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
bi_dropout=0	0.99966	0.01432	1.00000	0.99954	0.01589	1.00000
bi_dropout=0.05	0.99952	0.02186	1.00000	0.99958	0.02198	1.00000
bi_dropout=0.1	0.99962	0.01345	1.00000	0.99954	0.01509	1.00000
bi_dropout=0.15	0.99961	0.01438	1.00000	0.99953	0.01533	1.00000
bi_dropout=0.2*	0.99969	0.01457	1.00000	0.99949	0.01674	1.00000

3.1.17 AFM

test on different attention network architectures

LON-A

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
attention_n=2	0.99706	0.10241	1.00000	0.99557	0.12584	1.00000
attention_n=4	0.99720	0.10615	1.00000	0.99591	0.12716	1.00000
attention_n=6*	0.99746	0.07000	1.00000	0.99590	0.08987	1.00000
attention_n=8	0.99741	0.08290	1.00000	0.99597	0.10114	1.00000
attention_n=10	0.99719	0.06785	1.00000	0.99558	0.08608	1.00000

NYC-R

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
attention_n=2	0.99719	0.24524	1.00000	0.99701	0.24705	1.00000
attention_n=4	0.99837	0.18703	1.00000	0.99827	0.18979	1.00000
attention_n=6	0.99660	0.17723	1.00000	0.99599	0.18635	1.00000
attention_n=8	0.99863	0.20014	1.00000	0.99855	0.20488	1.00000
attention_n=10*	0.99868	0.16114	1.00000	0.99863	0.16333	1.00000

3.1.18 xDeepFM

test on different CIN layer sizes

LON-A

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
CIN layer size=(256, 128)	0.99787	0.04936	1.00000	0.99743	0.05961	1.00000
CIN layer size=(256, 128, 64)*	0.99854	0.04536	1.00000	0.99776	0.05771	1.00000
CIN layer size=(128, 64)	0.99816	0.04629	1.00000	0.99756	0.05748	1.00000
CIN layer size=(128, 64, 32)	0.99810	0.04706	1.00000	0.99760	0.05471	1.00000

NYC-R

settings	validation AUC	validation LogLoss	validation NDCG@5	testing AUC	testing LogLoss	testing NDCG@5
CIN layer size=(256, 128)	0.99968	0.01142	1.00000	0.99949	0.01310	0.99974
CIN layer size=(256, 128, 64)	0.99975	0.01117	1.00000	0.99957	0.01281	0.99979
CIN layer size=(128, 64)*	0.99987	0.01079	1.00000	0.99949	0.01250	1.00000
CIN layer size=(128, 64, 32)	0.99968	0.01097	1.00000	0.99955	0.01277	0.99987

3.2 Model Comparision

	LON-A			NYC-R		
	LogLoss	**AUC**	**NDCG@5**	**LogLoss**	**AUC**	**NDCG@5**
Baseline	**0.06342**	**0.99748**	**0.99990**	**0.09440**	**0.99901**	**0.99966**
UCF-s	**2.85695**	**0.50762**	**0.91625**	**0.19097**	**0.92553**	**0.99572**
UCF-p	**2.29329**	**0.51150**	**0.93262**	**0.09448**	**0.95404**	**0.99840**
ICF-s	**2.12324**	**0.52433**	**0.93968**	**0.10380**	**0.97160**	**0.99836**
ICF-p	**2.84786**	**0.50680**	**0.91566**	**0.07931**	**0.69075**	**0.99863**
MF	**0.06513**	**0.99739**	**0.99985**	**0.09920**	**0.99914**	**0.99959**
FM	**2.15446**	**0.49727**	**0.66084**	**0.55589**	**0.55349**	**1.00000**
BPR-FM	**2.15446**	**0.49727**	**0.66084**	**0.55589**	**0.55349**	**1.00000**
FNN	**0.06077**	**0.99714**	**1.00000**	**0.02030**	**0.99953**	**1.00000**
IPNN	**0.05313**	**0.99618**	**1.00000**	**0.02261**	**0.99954**	**1.00000**
OPNN	**0.05969**	**0.99554**	**1.00000**	**0.01462**	**0.99956**	**1.00000**
CCPM	**0.04456**	**0.99801**	**1.00000**	**0.00992**	**0.99958**	**1.00000**
WD	**0.06067**	**0.99697**	**1.00000**	**0.01376**	**0.99951**	**1.00000**
DeepCross	**0.05212**	**0.99761**	**1.00000**	**0.01214**	**0.99952**	**1.00000**
NFM	**0.05171**	**0.99777**	**1.00000**	**0.01674**	**0.99949**	**1.00000**
DeepFM	**0.05476**	**0.99725**	**1.00000**	**0.01248**	**0.99955**	**1.00000**
AFM	**0.08987**	**0.99590**	**1.00000**	**0.16333**	**0.99863**	**1.00000**
xDeepFM	**0.05771**	**0.99776**	**1.00000**	**0.01250**	**0.99949**	**1.00000**

ConvIPNN	**0.06929**	**0.99535**	**1.00000**	**0.01263**	**0.99957**	**1.00000**

3.2.1 Settings

All the models and methods we used to compare are contributed by open source libraries 'Surprise', 'LightFM' and 'deepctr-torch'. We adopt default values for most parameters except the parameters which are related to models' main features (e.g. convolution filter sizes and filter numbers for CCPM). In the section 3.1, the parameters we adopted for the best test results are listed on the tables, notation '*' behind the settings indicates that the setting lead to best experiment result.

3.2.2 Non-NN-Based VS. NN-Based

The traditional methods for recommendation systems are mainly consists of Collaborative Filtering (implemented by K-nearest neighbors) and matrix factorization. From the table about model comparison in section 3.2, we can learn that NN-based model generally have better performance with longer computing time as a tradeoff. Neural Network can simulate the exact the same behavior the Factorization Machine has (feature interaction), in addition, NN-based methods also introduce many features from Deep Learning including convolution, attention, combination of different network. The variety of features of Neural Network will enable the NN-based recommendation system have better generalization and accuracy.

3.2.3 Proposed Model Performance

Our proposed model, ConvIPNN, adopts features from CCPM and IPNN, has AUC score 0.99535 on LON-A dataset and AUC score 0.99957 on NYC-R dataset. The combination of convolution and inner product mechanism is proved to be useful to predict interaction between user and item. However, in the test on dataset LON-A, our proposed model's performance is slightly worse than IPNN and CCPM. The performance difference in the test on dataset NYC-R is not obvious. The possible reason besides the difference of data distribution of the two dataset might caused by the fact that the feature interaction made by convolution is local instead of global. However, the feature interaction conducted by FNN, PNN, NFM are inner product or fully-connected layer, which will consider the global features' interaction.

4 CONCLUSION

In this homework, we learned to use several methods of recommendation system include traditional collaborative filtering, matrix factorization to recent proposed NN-based methods. From results of experiment about method performance, we can learned that the evolving of Neural Network improves the ability of generalization and memorization. In addition, we also proposed a new architecture of network model using combination of IPNN and CCPM, however, from the experiment results we may conclude that convolution is not the proper way to handle the features which don't have location relationship with neighboring features.

5 CITATIONS

- Yehuda Koren, Robert Bell, Chris Volinsky, "Matrix Factorization Techniques for Recommender Systems", 2009
- Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, Guangzhong Sun. "xDeepFM: Combining Explicit and Implicit Feature Interactionsfor Recommender Systems", KDD '18
- Lipi Shah, Hetal Gaudani, Prem Balani, "Survey on Recommendation System", 2016
- Steffen Rendle, "Factorization Machines", 2010
- Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, Jun Wang, "Product-based Neural Networks for User Response Prediction", ICDM 2016
- Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, Hemal Shah, "Wide & Deep Learning for Recommender Systems", DLRS 2016
- Ruoxi Wang, Bin Fu, Gang Fu, Mingliang Wang, "Deep & Cross Network for Ad Click Predictions", ADKDD 2017
- Xiangnan He, Tat-Seng Chua, "Neural Factorization Machines for Sparse Predictive Analytics", SIGIR 2017
- Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, Xiuqiang He, "DeepFM: A Factorization-Machine based Neural Network for CTR Prediction", IJCAI 2017
- Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, Guangzhong Sun, "xDeepFM: Combining Explicit and Implicit Feature Interactionsfor Recommender Systems", KDD 2018
- Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, Tat-Seng Chua, "Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks", IJCAI 2017