SML21 Model Description

Jiang He, Yu Shi, Hongyi Huang

2018/06/15

1 Introduction

Our model is simply an (averaging) ensemble of FFM-based Neural Networks with attention. For network structure, we mainly borrow ideas from DeepFM[1], Field-aware Factorization Machine[2] and Deep Interest Network[3]. For feature engineering, we draw on experience from 4 Idiots Approach for Click-through Rate Prediction (https://www.csie.ntu.edu.tw/ r01922136/slides/kaggle-avazu.pdf).

2 Feature Engineering

(See user_history_4.py)

In the final submission, additional to 30 raw types of features, we add four additional hand-crafted features. They are all related to the historical information of users.

Indeed, we add the positive and negative impressions on aids and advertisements' features (such as ProductId, AdvertiserId...). Because of practical issue (Mainly restrictions on gpu memory), we group those advertisements' feature values into a single field. For example, if in the dataset, there are following triplets for user u1 with feature values of advertisement shown in the brackets:

1. (u1,a1,1),a1:[1,2,3]
2. (u1,a2,-1),a2:[1,3,5]
3. (u1,a3,1),a3:[2,3,4,5]
4. (u1,a4,-1),a4:[0,1,2,5]
5. (u1,a5)(test)

Then their constructed features are shown in Table 1.

Note because we don't have real time stamps, for a user with an ad, we assume all the triplets associated with this user in the training set provides history information. But for each pair of user and ad, we have to exclude itself from history so that there is no information leak in training.

Besides, we map low-frequency features in a field to a special value.

Table 1: Hand-crafted features

triplet	pos aid	neg aid	pos ads' feats	neg ads' feats
(u1,a1,1)	a3	a2,a4	[2,3,4,5]	[0,1,2,3,5]
(u1,a2,-1)	a1,a3	a4	[1,2,3,4,5]	[0,1,2,5]
(u1,a3,1)	a1	a2,a4	[1,2,3]	[0,1,2,3,5]
(u1,a4,-1)	a1,a3	a2	[1,2,3,4,5]	[1,3,5]
(u1,a5)(test)	a1,a3	a2,a4	[1,2,3,4,5]	[0,1,2,3,5]

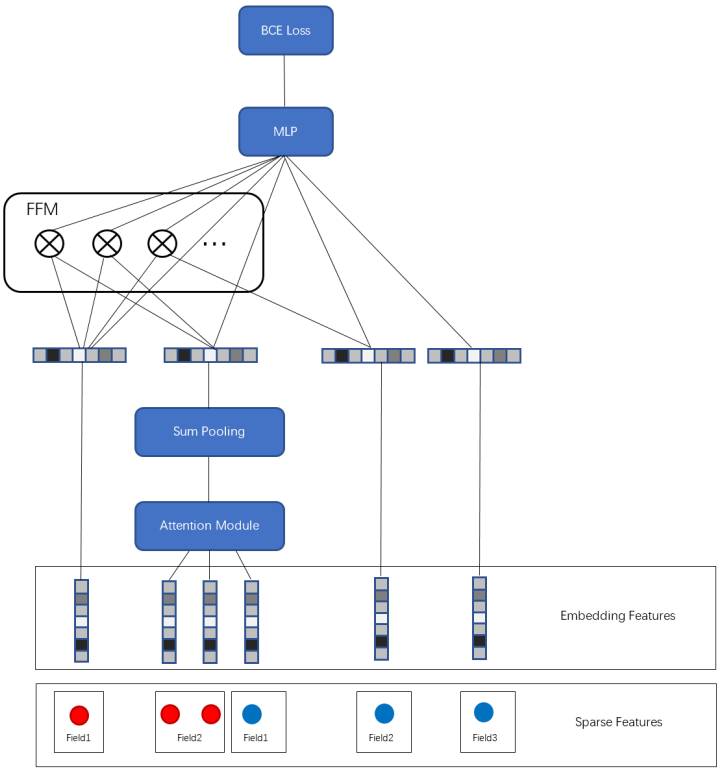

Figure 1: Main structure of network.

3 Network Structure

(See src/model/DIN_ffm_v3_r.py)

The main structure of our model is shown in Figure 1.

We describe the network from bottom to top.

3.1 Embedding

(See src/module/embedding_atten_v2.py)

First, we embedding features into a 6-dim vector space. Because we use FFM, so the actual embedding size of a single feature value is $6 * \#fields$. # is an abbreviation of 'the number of'. In our case, $\#fields = 34$. By the way, to save memory, we may use a hybrid FM and FFM in the practice, for field with large number of feature values, such as keywords, we could require feature to be not field-aware.

For fields that may have multiple feature feature values, before feed them into FFM and NN(Neural Network), we do sum pooling to get a fixed length of representation, namely $6 * \#fields$.

For some features of special interest, such as users' interests, keywords and topics, we add attention mechanism. Basically this attention module takes the embedding of a keyword (for example) and the embedding of aid as inputs, then a small network ending with sigmoid outputs a value between 0 and 1 to multiply with the embedding of this keyword so that the norm of embedding is controlled by aid.

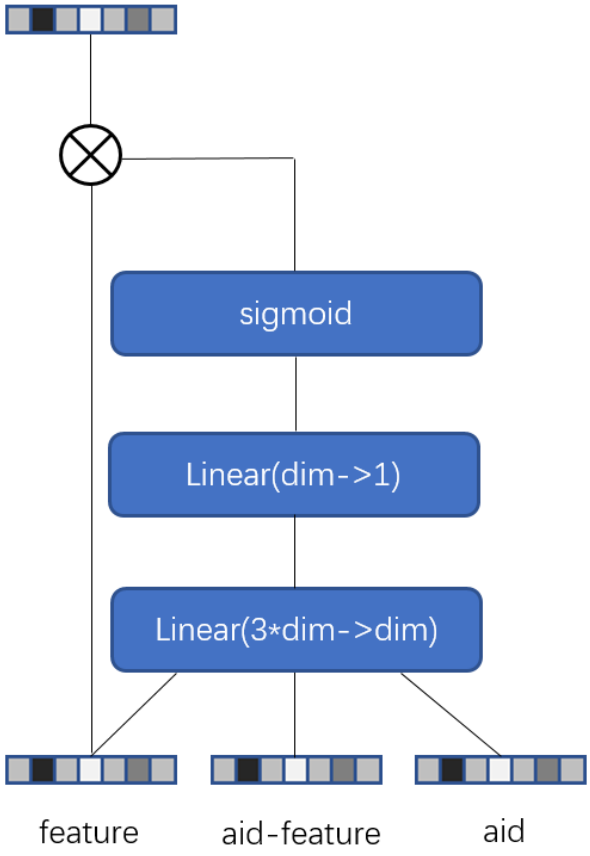

Figure 2: Attention module. $dim = 6 * \#fields$

3.2 Attention

(See src/module/embedding_atten_v2.py)

The attention module used in shown in Figure 2. The input to the network are the embedding of feature value, the embedding of aid, and the difference of them. To reduce computation, we could do pooling first, reducing the dimension of embedding from $6 * \#fields$ to 6.

3.3 FFM and NN

(See src/module/FM.py and src/model/DIN_ffm_v3_r.py)

After attention module and sum pooling, for each sample, we have 34 fields, each with $6 * 34$ vector representation, then we use traditional FFM to capture second-order interactions between features, which generates $35 * 34/2 = 595$ second-order terms. Our implementation of FFM is not efficient, we simply use indexing operation, element-wise multiplication and sum operation provided by Pytorch to implement FFM. Indeed, we should better implement it in c/cuda.

Then we concatenate second-order terms(595) with first-order term($6 * 34 * 34 = 6936$) and feed them into a 3-layer neural network. To reduce computation and save memory, we could do pooling to reduce dimension first as well.(We didn't notice this high dimensional feature problem in the competition because at the beginning, the structure is designed for FM.)

The 3-linear layers are $Linear(7513 \rightarrow 128)$, $Linear(128 \rightarrow 64)$, $Linear(64 \rightarrow 1)$. We use Dice activation function introduced by [3] between linear layers, which is similar to a combination of BatchNorm and PReLU.

Finally, we use binary cross entropy loss to train the model.

In our experiments, we find feeding concatenation of second-order term with first-order term into a multiple neural layers works better than the design of wide and deep network in [1] and earlier work. On the one hand, it makes the combination of features more complex. On the other hand, FFM is freed from directly influencing the final confidence and is only used as feature extractor. This allows us to design more complex FFM, without need to consider whether it is mathematically meaningful.

References

[1] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He. Deepfm: A factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[2] Y. Juan, Y. Zhuang, W.-S. Chin, and C.-J. Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 43–50. ACM, 2016.

[3] G. Zhou, C. Song, X. Zhu, X. Ma, Y. Yan, X. Dai, H. Zhu, J. Jin, H. Li, and K. Gai. Deep interest network for click-through rate prediction. *arXiv preprint arXiv:1706.06978*, 2017.