Operation-aware Neural Networks for User Response Prediction

Yi Yang[a], Baile Xu[a], Furao Shen[a,*], Jian Zhao[b,*]

[a]*State Key Laboratory for Novel Software Technology,*
Department of Computer Science and Technology,
Collaborative Innovation Center of Novel Software Technology and Industrialization,
Nanjing University, China
[b]*State Key Laboratory for Novel Software Technology,*
School of Electronic Science and Engineering, Nanjing University, China

Abstract

User response prediction makes a crucial contribution to the rapid development of online advertising system and recommendation system. The importance of learning feature interactions has been emphasized by many works. Many deep models are proposed to automatically learn high-order feature interactions. Since most features in advertising system and recommendation system are high-dimensional sparse features, deep models usually learn a low-dimensional distributed representation for each feature in the bottom layer. Besides traditional fully-connected architectures, some new operations, such as convolutional operations and product operations, are proposed to learn feature interactions better. In these models, the representation is shared among different operations. However, the best representation for different operations may be different. In this paper, we propose a new neural model named *Operation-aware Neural Networks* (ONN) which learns different representations for different operations. Our experimental results on two large-scale real-world ad click/conversion datasets demonstrate that ONN consistently outperforms the state-of-the-art models in both offline-training environment and online-training environment.

Keywords: Neural Networks, Click-Through Rate Prediction, Factorization Machines

[*]Corresponding authors

Email addresses: `frshen@nju.edu.cn` (Furao Shen), `jianzhao@nju.edu.cn` (Jian Zhao)

1. Introduction

In recent years, online advertising develops rapidly among social medias such as Facebook and Wechat. As a critical role of online advertising, user response prediction makes a crucial contribution, where the task is to estimate the probability of a user will click on an ad (click-through rate, CTR) or take a desired action after clicking the ad (conversion rate, CVR).

In CTR/CVR prediction tasks, the importance of learning feature interactions has been emphasized by many works in related literature [1, 2]. For instance, users of different ages have different preferences for different types of ads, which suggests that the interaction between *User Age* and *Ad Type* is a strong signal for CTR/CVR prediction. Traditional linear models with manual feature engineering have shown decent results, but manual feature engineering is very labor-intensive and time-consuming. At the same time, it is very difficult for human experts to discover high-order interactions between features.

Deep neural networks (DNN) show very promising results at automatically learning feature representations and dependencies. Driven by the success of DNN, several neural architectures are proposed for CTR/CVR prediction in recent years. Most of these architectures can be concluded into 3 steps:

i) An embedding layer is used to map high-dimensional sparse features into low-dimensional distributed representations. The output of the step is $e = [V^0 x_0, V^1 x_1, \cdots, V^m x_m]$, where V^i is the embedding matrix of the ith feature and x_i is the one-hot representation of the ith feature.

ii) step2) Several operations are applied on the embedding vectors to get the medial features. The output of this step is $f = [o_1(e), o_2(e), \cdots, o_l(e)]$ where o_i is the ith operation. In most architectures, the operation is just a copy of the embedding vectors.

iii) A multi-layer perceptron (MLP) is applied on f to learn nonlinear relations among features. The output of this step is $\widehat{y} = \sigma(\Phi(f))$ where σ is the *sigmoid* function and Φ is the multi-layer non-linear transformation. Some architectures also have some other components. In these models, the output is $\widehat{y} = \sigma(\Phi(f) + \Delta)$ where Δ denotes the specific components.

Some recent works focus on introducing new operations to learn the feature interactions better, such as convolutional operations [3] and "inner-product/outer-product" operations [4]. Different operations play different roles in deep models. For instance, each "copy" operation reserves the original embedded representation for one feature. Each “inner-product” operation or “outer-product” operation learns a local dependency between two features. The operations on embedding vectors can be regarded as an incipient feature engineering before applying MLP.

On the other hand, the feature representation is very important to the model because a better representation makes learning feature interactions easier. However, few works focus on improving the feature representation learned by the embedding layer. Existing models usually share the same feature representation among different operations. However, the best feature representations for different operations are not always the same. It has been experimentally proven in previous works [5, 2] that structures explicitly using different embeddings among different operations perform better than structures sharing one embedding among all operations in many tasks.

Inspired by this idea, we propose a new embedding method named *operation-aware* embedding in this paper, which learns different representations for each feature when performing different operations. Models with operation-aware embedding layer are named as *Operation-aware Neural Networks* (ONN). The operation-aware embedding makes ONN more flexible than exsiting models. Compared with state-of-the-art models, ONN shows superior performances in many CTR/CVR tasks. Furthermore, experimental results show that ONN is especially suitable for online-training environments. We have used ONN to win the first prize of Tencent Social Advertising College Algorithm Competition among about 1000 teams[1].

The rest of the paper is organized as follows. Related works are introduced in section II. The model details are described in Section III. We discuss feature embedding and the relationships of ONN with related models in Section IV. Section V exhibits experiments analysis. Finally, we conclude the paper in Section VI.

2. Related Works

The CTR/CVR tasks, which can be formalized as a classic binary classification problem, have been intensely studied in the literature. Logistic Regression with FTRL optimizer [6] has been widely used in real world applications. However, the linear model needs a lot of artificial feature engineering to generate the feature interactions. In order to solve this problem, Factorization Machines (FM) [1] are proposed to learn the feature interactions automatically.

With the rapid development of deep learning, many deep methods are also proposed to solve the CTR/CVR tasks. Factorization-machine supported Neural

[1]http://algo.tpai.qq.com/home/home/index.html

Network (FNN) is the first deep model for CTR/CVR tasks, which uses a MLP to learn high-order feature dependencies on the hidden vectors of FM. Without FM initialization, FNN is equivalent to a standard MLP above an embedding layer. For the purpose of learning better feature interactions, Convolutional Click Prediction Model (CCPM) [3], DeepCross [7] and Product-based Neural Network (PNN) [4] introduce some new operations to be performed on the embedded features before applying MLP. Some researchers think that the shallow component can be a supplement to the deep models. For instance, the Wide $\&$ Deep model of Google [8] uses a linear component as the supplement to the deep component. Similarly, deepFM [9] is proposed by training a FM component and a deep component at the same time. Besides, there are also some researches which focus on improving the model performance by mining the important features. For instance, Deep Interest Network (DIN) [10] proposes using attention mechanism to adaptively learn the representation of user interests from historical behaviors. The attention mechanism is also used in [11], but it is used to learn the weights of feature interactions.

To the extent of our knowledge, few works focus on improving the embedding layer. In following parts of the paper, we demonstrate the operation-aware embedding can improve the model performance by learning better feature representations.

3. Operation-aware Neural Networks

In an advertising system, each datum consists of many features including user information (Age, Education, etc.), ad information (Publish Company, Ad Type, etc.), context information (Connection Type, Siteset ID , etc.). We only consider the case of categorical features here because most features in ad systems are either categorical or can be made categorical through discretization. Each feature is represented as a vector of one-hot encoding after data preprocessing. Suppose each datum have m features, then each training sample can be represented as (x, y) where $x = [x_1, x_2, ..., x_m]$ with x_i standing for the one-hot vector of the ith feature and $y \in \{0, 1\}$ records whether the user performed a positive action. The objective of a user response prediction system is learning a function $f : x \rightarrow \widehat{y}$ that maps the input features x to the estimated probability of positive response $\widehat{y} \in [0, 1]$.

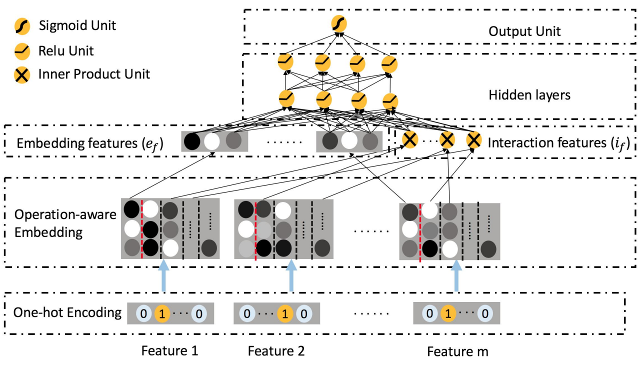

Figure 1: The architecture of the ONN model. From the bottom-up perspective, the model first uses an operation-aware embedding layer to map high-dimensional sparse features into low-dimension representations. After that, "copy" operations and "inner-product" operations are performed on corresponding features respectively. Finally, an MLP learns high-order feature interactions and produce the output.

3.1. Model Architecture Design

The architecture of the ONN model is illustrated in Figure 1. From the bottom-up perspective, the model can be divided into 3 components:

3.1.1. Operation-aware Embedding

The one-hot encoding is inefficient and can not represent correlations among features. In addition, many operations are meaningless with one-hot encoding, such as the "inner-product" operation. In order to solve this, we use an embedding layer to map the high-dimensional one-hot vectors into low-dimensional vectors. Compared with one-hot encoding, embedded representation is more efficient, informative, and learnable.

Existing models usually learn one representation for each feature and use the same representation among all operations. However, the best representation may be different among different operations. Why can one feature have just one representation? If we use one representation for all operations, the representation needs to compromise among all operations. On the bright side, this compromise might have the effect of regularization sometimes, especially when the training data are not sufficient. But in CTR/CVR tasks where training data are easy to obtain, the compromise tends to limit the expression ability of the model.

To learn different representations for different operations, we propose the *operation-aware embedding* method. For each feature, an embedding vector is learned for each operation performed on it. Note that *operations of the same type*

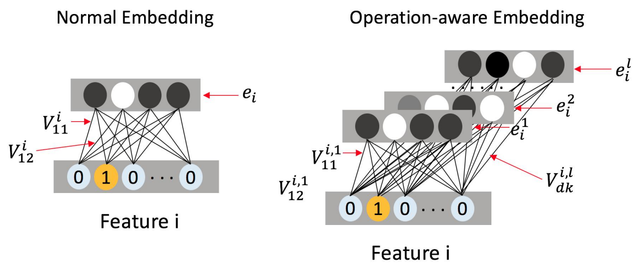

Figure 2: The structures of the normal embedding layer and the operation-aware embedding layer.

performed on different features are also regarded as different operations. For instance, for feature i, the "inner-product" with feature j and the "inner-product" with feature p are regarded as different operations. The similar idea is also used in Pairwise Interaction Tensor Factorization (PITF) [5] and Field-aware Factorization Machine (FFM) [2].

Figure 2 illustrates the difference between the normal embedding layer and the operation-aware embedding layer. In the normal embedding layer, the embedded representation for all operations of the ith feature is $e^i = V^i x_i$ where V^i is embedding matrix of the ith feature. However, in the operation-aware embedding layer, each feature has several embedded representations. Suppose there are l operations performed on the ith feature, we use $[V^{i,1}, V^{i,2}, \cdots, V^{i,l}]$ to denote the embedding matrices of the ith feature. Then the embedded representations of the ith feature are $[e^1_i, e^2_i, \cdots, e^l_i] = [V^{i,1}x_i, V^{i,2}x_i, \cdots, V^{i,l}x_i]$. When performing the kth operation on the ith feature, e^k_i is used as the representation to perform the computation.

In order to use *operation-aware* representation for each operation, we need to construct a mapping Γ which maps an operation and a feature index to the operation index of the feature. Specifically speaking, $\Gamma(o, i) = k$ means that the operation o is the kth operation to be done on the ith feature.

3.1.2. Incipient Feature Extraction Layer

Besides the embedded representations for the raw features, we also hope that the bottom layer can provide representations for the feature interactions because the feature interactions are very important in CTR/CVR tasks. Intuitively, we can use the one-hot encoding to encode the interaction of the two features and then embed it into a low-dimensional vector. However, the direct interaction of two features usually leads to a very sparse feature whose representation can not obtain adequate training. Factorization Machines (FM) solves the problem by factorizing the effect of feature interactions into a product of two latent vectors. In the PNN model, [4] solves the problem in the same way. They use the "inner-product" operation and the "outer-product" operation of the embedding vectors to represent the feature interactions. These works suggest that "product" operations are effective in learning feature interactions.

We use the same idea in the ONN model. Because the experimental results in [4] demonstrate that the "inner-product" operation performs better than the "outer-product" operation most of time, we use the "inner-product" operation as the default operation to learn the feature interactions. We only consider 2-order interactions because higher-order interactions are too complicated. We use $o(c, i)$ to denote the "copy" operation of the ith feature, and $o(p, i, j)$ to denote the "inner-product" operation between the ith feature and the jth feature. The output of this layer can be split into the embedded features e_f and the interaction features i_f. Suppose we have m features, e_f can be constructed as follow:

$$e_f = [e_1^{\Gamma(o(c,1),1)}, e_2^{\Gamma(o(c,2),2)}, \cdots, e_m^{\Gamma(o(c,m),m)}]. \tag{1}$$

recall that Γ is the mapping from operation to index. i_f can be constructed as follow:

$$i_f = [p_{1,2}, p_{1,3}, \cdots, p_{m-1,m}] \tag{2}$$

where $p_{i,j}$ is the value of the "inner-product" operation between the ith feature and the jth feature:

$$p_{i,j} =< e_i^{\Gamma(o(p,i,j),i)}, e_j^{\Gamma(o(p,i,j),j)} > . \tag{3}$$

Concatenating e_f and i_f constructs the output of this layer $f = [e_f, i_f]$.

3.1.3. Multiple Nonlinear Layers

Lastly, a multi-layer perceptron (MLP) is applied on the medial features f to mining the sophisticated patterns among data and generate the model output. For each layer of the MLP, we add a batch normalization (BN)[12] layer to accelerate the training. We have conducted a large number of experiments and find that BN can not only greatly speed up the training, but also improve the prediction accuracy. For the definition of BN, please refer to [12].

We explain how to generate the output with an architecture consist of 2 hidden layers as an example. Firstly we should do batch normalization on f. Denote the result as $\widehat{f} = BN(f)$, then the output of the first hidden layer is:

$$l1 = BN(relu(W_1\widehat{f} + b_1)) \tag{4}$$

where W_1 and b_1 are the model weights and bias of the first hidden layer, $relu$ is the rectified linear unit [13], defined as $relu(x) = max(0, x)$. Hence the output of the second hidden layer is:

$$l2 = BN(relu(W_2 l_1 + b_2)). \tag{5}$$

Then the output of the model is:

$$\widehat{y} = \sigma(W_3 l_2 + b_3) \tag{6}$$

where σ is the sigmoid function, defined as $\sigma(x) = \frac{1}{1+exp(-x)}$. After that, we minimize the log loss to train the model. The loss function is defined as:

$$L(y, \widehat{y}) = -ylog(\widehat{y}) - (1 - y)log(1 - \widehat{y}). \tag{7}$$

4. Discussions

4.1. Feature Embedding

In natural language processing (NLP), a word may have various meanings in different contexts. Multi-sense embeddings [14, 15] are proposed to solve the problem of one word having multiple meanings. Similarly, each feature is faced with different contexts when performing different operations in CTR/CVR tasks. The operation-aware embedding learns representations for each feature in different contexts. The context adaptive representations are more informative then single representation, thus provide an improvement to the model performance.

Although we just use the “copy” operation and the “inner-product” operation in this paper, the ONN model can be generalized to more operations, such as the “outer-product” operation mentioned in [4]. In fact, we have tried to use a sub-network as an operation type in our experiments and have also achieved pretty good results, but the time complexity is relatively higher.

The embedding layer in the ONN model is more flexible than that in the other models [4, 9, 16], which share the same embedding among all operations and hence reduce their flexibility. For example, the embedding dimensionality of User ID should be larger than that of User Education because the possible values of User ID are far more than that of User Education. But if we share the same embedding between “copy” operations and “product” operations, the embedding dimension of User ID and User Education should be the same because “product” operations need the 2 vectors to have the same length. In contrast, we can use different embedding dimensionalities for different features in the ONN model. For each feature, when performing product operations with different features, we can also use different embedding dimensionalities. By artificially designing different embedding dimensionalities for each feature and each operation, we can minimize the computation complexity during training and testing.

4.2. Relationships with Related Models

In this subsection, we discuss the relationships of the ONN model and several related models. Firstly, we give the neural architectures of FM and FFM. If we only consider category features, the FM of degree 2 is defined as:

$$\phi_{fm}(w,x) = \Sigma_{i=1}^{m}\Sigma_{j=i+1}^{m} < v_i, v_j > \tag{8}$$

where v_i is the latent vector of the ith feature. Figure 3 illustrates the structure of FM. The bottom layer is a normal embedding layer, where each embedding vector corresponds to the latent vector of one feature in FM. The embedding vectors are connected with the inner product units. Each inner product unit corresponds to an inner product of 2 latent vectors in FM. Finally, the product units are summed up to generate the output. FFM is an optimized version of FM. But when performing the product operation with different features, FFM uses field-aware vector for each feature. The FFM model is defined as:

$$\phi_{ffm}(w,x) = \Sigma_{i=0}^{m}\Sigma_{j=i+1}^{m} < v_{i,j}, v_{j,i} > . \tag{9}$$

Actually, “field-aware” is a special case of “operation-aware” when there is only the “inner-product” operation. Figure 4 illustrates the structure of FFM. It can be seen that FM uses the normal embedding layer, but FFM uses the operation-aware embedding. Thus, we say that FFM makes FM operation-aware.

However, FM and FFM lack the ability of mining deep feature dependencies while deep models are good at that. Via comparing the architectures of FFM and ONN, we observe that the bottom layers of ONN and FFM are the same, but ONN uses several non-linear layers on top of the embedded representations and the product units to mine deep feature dependencies, while FFM directly uses the product units to generate the output. Thus we say that ONN deepens FFM. Figure 5 illustrates the architecture of PNN proposed in [4]. Via comparing the architecture of PNN and FM, it is observable that PNN also deepens FM.

Lastly, via comparing figure 1 and figure 5, we observe that the ONN uses the operation-aware embedding, but PNN uses the normal embedding. Thus ONN makes PNN operation-aware.

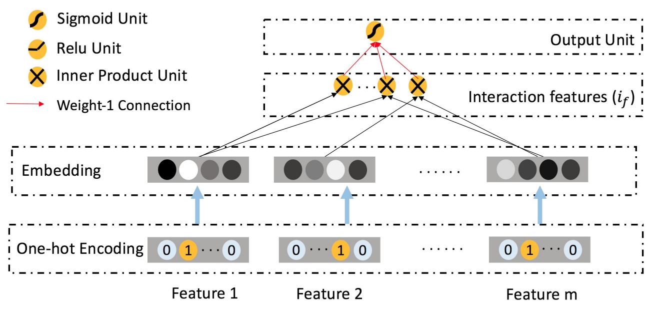

Figure 3: The architecture of the FM model. The embedding vectors correspond to the latent vectors in FM. The sum of the inner product units is fed into the output unit.

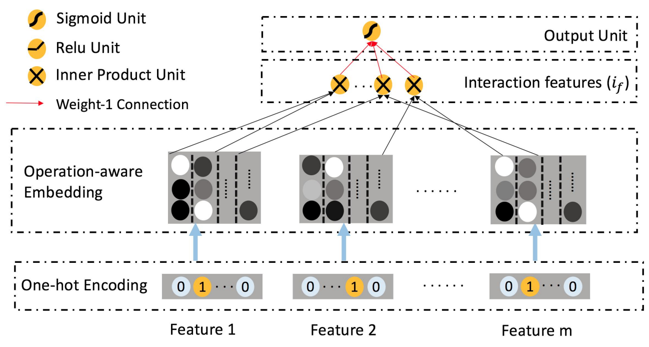

Figure 4: The architecture of the FFM model. The structure is the same as FM, except that the embedding layer is operation-aware in FFM.

Compared with shallow models, deep models can automatically learn sophisticated feature dependencies. Compared with the normal embedding layer, the operation-aware embedding layer learns better feature representations for different operations. Thus, ONN is more effective than traditional shallow models and PNN.

5. Experiments

In this section, we present our experiments in detail, including datasets, data processing, experimental setups, model comparisons in offline-training setting and online-training setting, and the analyses of different operations. In our experiments, the ONN model outperforms major state-of-the-art models in the CTR estimation task on two real-world datasets in both offline-training and online-training settings.

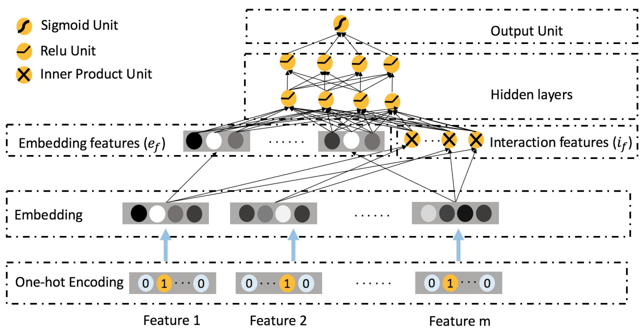

Figure 5: The architecture of the PNN model. The structure of PNN is the same as ONN, except that ONN uses operation-aware embedding layer.

5.1. Data

Criteo. Criteo dataset[17] includes 45 million users' click records. There are 13 continuous features and 26 categorical ones. We use the last 5 million records for testing and the other records for training. The continuous features are discretized by the function $discrete(x) = \lfloor 2 * log(x) \rfloor$, where $\lfloor \cdot \rfloor$ is the floor function.

Tencent Ad. The Tencent Ad dataset[18] is used in the Tencent Social Advertising College Algorithm Competition. The data contains 14 days app ad conversion data, user information, ad information and the records of installed apps. We use 39 categorical features which we used in the competition to perform the experiments. Because the data in the last 2 days is noisy, we use the first 11 days for training, and the 12th day for testing. After the split, there are about 22 million train data and 2 million test data.

5.2. Model Comparison

We compare ONN with 5 models in our experiments, which are implemented with TensorFlow and trained with the Adam optimization algorithm [19].

FM: FM learns feature interactions by factorizing it into the inner product of 2 vectors. FM has many successful applications in many CTR/CVR tasks [1].

FFM: FFM is an optimized version of FM. FFM uses field-aware vectors to perform product operation.

DNN: A simple deep model without product operations. The embedded features are directly fed into an MLP to learn high-order feature interactions and generate output.

PNN: PNN is proposed in [4]. PNN introduces the "inner-product" operation and the "outer-product" operation into deep model. In our experiments, the PNN is implemented with "inner-product" operation, since ONN uses the "inner-product" operation by default.

DeepFM: DeepFM trains a deep component and an FM component at the same time [9]. DeepFM can automatically learn low-order feature interactions and high-order feature interactions at the same time.

ONN: ONN is the proposed model of this paper.

For fairness, we add Batch Normalization(BN) layers to all deep models. The non-linear activation function is set to relu.

5.3. Evaluation Metrics

We use four evaluation metrics in our experiments: Area Under ROC(AUC), Cross Entropy(Logloss), Pearson's Correlation Coefficient(Pearson's R) and Root Mean Squared Error (RMSE).

5.4. Offline-Training Performance Comparison

We present the performances of models in offline-training environment. In this setting, the data can be trained several epoches. Since many hyper-parameters, such as the number of hidden layers, the hidden sizes and embedding dimensionality, have been discussed enough in [4] and [9], we just follow most parameter settings from their works. The embedding dimensionality is set to 10 for all models. Although ONN can use flexible embedding dimensionalities, we just use the same embedding dimensionality with other models for fairness. The number of non-linear hidden layers is set to 3. For the Criteo dataset, the hidden sizes are set to [400, 400, 400]. For the Tencent Ad dataset, the hidden sizes are set to [200, 200, 200]. Besides, the learning rate of Adam is set by grid search from [0.0001, 0.00025, 0.0005, 0.00075, 0.001] using cross validation. The training batch size is set to 2500.

Table 1 and tabel 2 show the overall performance in offline-training setting. The results show that ONN outperforms all the other model on all metrics. Besides, the results are consistent with the relationships among FM, FFM, PNN and ONN. We observe that PNN performs better than FM, and ONN performs better than FFM. These observations suggest that deep models are more effective than shallow models. Similarly, we observe that FFM performs better than FM, and

Table 1: Overall Performance on the Criteo dataset in Offline-Training Setting

Model	Criteo			
	Logloss	AUC	Pearson's R	RMSE
FM	0.44233	0.80464	0.48873	0.37805
FFM	0.43846	0.80920	0.49612	0.37627
DNN	0.43700	0.81059	0.49924	0.37557
PNN	0.43636	0.81134	0.50014	0.37529
DeepFM	0.43671	0.81150	0.49954	0.37541
ONN	**0.43577**	**0.8123**	**0.50139**	**0.37495**

Table 2: Overall Performance on the Tencent Ad dataset in Offline-Training Setting

Model	Tencent Ad			
	Logloss	AUC	Pearson's R	RMSE
FM	0.10684	0.82376	0.26644	0.15772
FFM	0.10639	0.82667	0.26753	0.15767
DNN	0.10581	0.82810	0.27261	0.15755
PNN	0.10550	0.82725	0.27429	**0.15734**
DeepFM	0.10595	0.82586	0.26731	0.15768
ONN	**0.10504**	**0.82993**	**0.27481**	0.15735

ONN performs better than PNN, these observations suggest that the operation-aware embedding is more effective than the normal embedding. Besides, all of PNN, DeepFM and ONN outperform DNN, suggesting that the "product" operation is useful in automatically learning feature interactions.

We find that dropout [20] does not improve the performance for all deep models after adding BN layer. In order to make a choice between the two techniques, we compare models with BN layer and models with 0.3 dropout rate on network hidden layers. Figure 6 illustrates the results. We observe that BN outperforms dropout for all deep models. The results suggest that BN is more useful than dropout in deep models for user response prediction tasks.

5.5. Online-Training Performance Comparison

In real world online advertising systems, training data are generated in a stream. Online training is more suitable for this environment than offline training. In order to validate the effectiveness of deep neural networks in online environment, we compare the performance of different models in online training settings. In

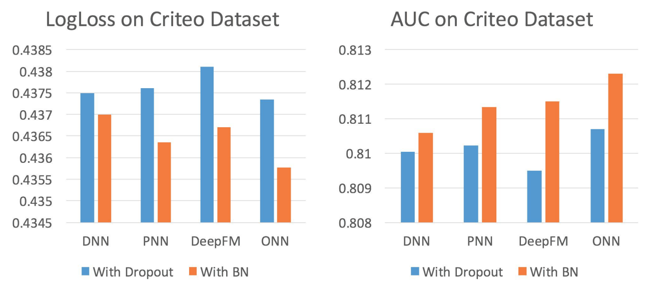

Figure 6: Performance Comparison between BN and Dropout.

this setting, the data should be trained in a stream. Mini-batch training is allowed, but the order of data sequence should be kept and each mini-batch can only be learned once in each experiment. The network parameters are consistent with the offline-training experiments. But the learning rate is searched again by cross validation.

Table 3: Overall Performance on the Criteo dataset in Online-Training Setting

Model	Criteo			
	Logloss	AUC	Pearson's R	RMSE
FM	0.44348	0.80346	0.48695	0.37849
FFM	0.44083	0.80643	0.49255	0.37731
DNN	0.44132	0.80579	0.49049	0.37763
PNN	0.44156	0.80584	0.49005	0.37772
DeepFM	0.44275	0.80459	0.49033	0.37768
ONN	**0.43751**	**0.81016**	**0.49813**	**0.37578**

Table 3 and table 4 show the overall performance in online-training setting. The results show that ONN outperforms all the other model on all metrics. Figure 7 illustrates the convergence curves of all models on the Criteo dataset. From the figure, we observe that the convergence curve of ONN is always at the bottom among all models, which suggests that ONN converges fastest among all models.

In online-training setting, each sample can only be trained one time. In FM, PNN and DeepFM, each feature representation needs to compromise among different operations, therefore it is hard for these models to learn a good representation within one epoch. However, in FFM and ONN which use the operation-aware embedding, each operation has its own representation, thus it is easier for these models to learn feature representations. Besides, we also observe that DNN out-

Table 4: Overall Performance on the Tencent Ad dataset in Online-Training Setting

Model	Tencent Ad			
	Logloss	AUC	Pearson's R	RMSE
FM	0.10721	0.81955	0.25699	0.15817
FFM	0.10656	0.82518	0.26572	0.15776
DNN	0.10643	0.82584	0.26744	0.15769
PNN	0.10698	0.82257	0.26563	0.15776
DeepFM	0.10653	0.82518	0.26743	0.15775
ONN	**0.10591**	**0.82812**	**0.27012**	**0.15762**

performs PNN and DeepFM in online-training environment. We think that is because there is only one "copy" operation to be done on each feature in DNN, and thus it is easier for DNN to learn feature representations compared with PNN and DeepFM. After that, ONN outperforms DNN because the "product" operations make ONN more effective than DNN to learn feature interactions. In conclusion, multiple operations are useful for deep models, and the operation-aware embedding enables models with multiple operations to learn good feature representations efficiently in online-training environment.

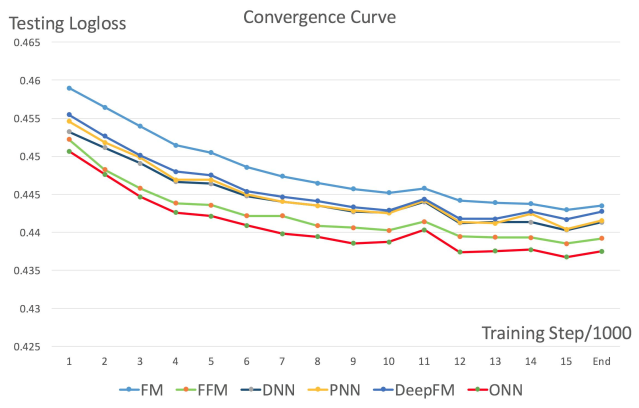

Figure 7: Model convergence curves on the Criteo dataset. The horizontal axis is the number of training steps in thousands. The vertical axis is the logloss on the testing dateset.

5.6. Analysis of Different Operations

By default, we only use the "inner-product" operation to learn the feature interactions. In fact, other operations can also be used in ONN. In this section, we compare the performance of different ONNs by replacing the default "inner-product" operation with "outer-product" and "sub-network" operations. The "sub-network" operations use a network with a hidden layer to learn the feature interactions between two features.

Besides, we also consider the combination of the "inner-product" operation and the "outer-product" operation. We do not consider the combinations of the "sub-network" operation with other operations because the space and time complexity of the "sub-network" operation is too high, which makes it unsuitable to be used in the real world systems.

Table 5: Performance comparison for different operations on the Criteo dataset

Model	Criteo			
	Logloss	AUC	Pearson's R	RMSE
sub-network	0.43597	0.81178	0.50081	0.37514
outer-product	0.43673	0.81089	0.49941	0.37547
inner-product	0.43571	0.81016	**0.50188**	**0.37482**
inner+outer-product	**0.43541**	**0.81236**	0.50165	0.37489

Table 6: Performance comparison for different operations on the Tencent Ad dataset

Model	Tencent Ad			
	Logloss	AUC	Pearson's R	RMSE
sub-network	0.10560	0.82612	0.27129	0.15753
outer-product	0.10516	0.82886	0.27328	0.15740
inner-product	0.10504	0.82993	0.27309	0.15744
inner+outer-product	**0.10502**	**0.83028**	**0.27429**	**0.15735**

Table 5 and table 6 show the experimental results of ONN with different operations. We can see that the "inner-product" operation performs best on the whole among the three single operations. Besides, we can see that the "sub-network" operation also provides competitive performances. For instance, the "sub-network" operation performs best on the AUC metric on the Criteo dataset. The performances of the"sub-network" operation indicate that there are more choices other than the "product" operations, which can also be used to learn feature interactions in the deep architectures, and we will explore more on that in out future works. After that, the combination of the "inner-product" operation and the "outer-product" operation achieves obvious improvement, and performs best among most metrics on the two experimental datasets.

In conclusion, we adopt the "inner-product" as the default operation because it performs very competitive and has the lowest space and time complexity. The "outer-product" operation can be used in addition if better performance needs to be achieved at the cost of storage space and computation time. In fact, the "outer-product" operation is much slower than the "inner-product" operation. For the detailed complexity analysis, please refer to [4].

6. Conclusion

In this paper, we propose a new embedding method named *operation-aware* embedding for learning feature representations in user prediction systems, and construct a new deep neural network named *Operation-aware Neural Networks* (ONN). Compared with the traditional feature embedding method which learns one representation for all operations, operation-aware embedding can learn various representations for different operations. Experimental results show that ONN outperforms several state-of-art models on 2 datasets in both offline-training and on-line training environment. Besides, ONN converges faster than other models and outperforms state-of-art models by a large margin in online-training environment, which suggests that ONN is very suitable for online system. ONN inherits the main network structure of PNN in this paper, but the operation-aware embedding layer can be applied to any neural architectures actually.

In future, we will explore the use of the operation-aware embedding layer on other applications like NLP. Besides, we are interested in how to automatically determine the embedding dimensionalities for different operations and different features. Another question is whether some operations can share one representation. We will also study on splitting the features and operations into different groups, where the embedded representations are shared in the same group, but not shared among different groups.

Acknowledgment

This work is supported in part by the National Science Foundation of China under Grant Nos. (61876076), and Jiangsu NSF grant (BK20141319).

References

[1] S. Rendle, Factorization machines, in: Data Mining (ICDM), 2010 IEEE 10th International Conference on, IEEE, pp. 995–1000.

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

[3] Q. Liu, F. Yu, S. Wu, L. Wang, A convolutional click prediction model, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, pp. 1743–1746.

[4] Y. Qu, H. Cai, K. Ren, W. Zhang, Y. Yu, Y. Wen, J. Wang, Product-based neural networks for user response prediction, in: Data Mining (ICDM), 2016 IEEE 16th International Conference on, IEEE, pp. 1149–1154.

[5] S. Rendle, L. Schmidt-Thieme, Pairwise interaction tensor factorization for personalized tag recommendation, in: Proceedings of the third ACM international conference on Web search and data mining, ACM, pp. 81–90.

[6] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al., Ad click prediction: a view from the trenches, in: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp. 1222–1230.

[7] R. Wang, B. Fu, G. Fu, M. Wang, Deep & cross network for ad click predictions (2017) 1–7.

[8] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al., Wide & deep learning for recommender systems, in: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, ACM, pp. 7–10.

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

[10] G. Zhou, X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, K. Gai, Deep interest network for click-through rate prediction, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &amp; Data Mining, ACM, pp. 1059–1068.

[11] J. Xiao, H. Ye, X. He, H. Zhang, F. Wu, T.-S. Chua, Attentional factorization machines: Learning the weight of feature interactions via attention networks, arXiv preprint arXiv:1708.04617 (2017).

[12] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: International Conference on Machine Learning, pp. 448–456.

[13] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines, in: Proceedings of the 27th international conference on machine learning (ICML-10), pp. 807–814.

[14] I. Iacobacci, M. T. Pilehvar, R. Navigli, Sensembed: Learning sense embeddings for word and relational similarity., in: ACL (1), pp. 95–105.

[15] J. Li, D. Jurafsky, Do multi-sense embeddings improve natural language understanding?, arXiv preprint arXiv:1506.01070 (2015).

[16] W. Zhang, T. Du, J. Wang, Deep learning over multi-field categorical data, in: European conference on information retrieval, Springer, pp. 45–57.

[17] Criteo, Kaggle display advertising challenge dataset, `http://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/`, 2014.

[18] Tencent, Tencent ads algorithm competition, `https://algo.qq.com/home/home/index.html`, 2018.

[19] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[20] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting., Journal of machine learning research 15 (2014) 1929–1958.