

Deep FM Based Neural Networks

For CTR Prediction

Zahra Dehghanian

Melika Abdollahi

Yasaman Mirmohammad

| Datamining

| December 2018

Task 1 :

**Wide & Deep Learning for Recommender Systems**

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra,

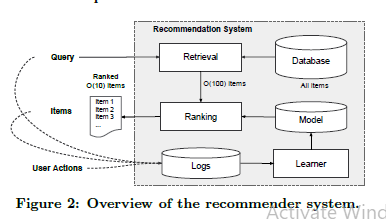
Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil,

Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, Hemal Shah

Google Inc.

# \_

Recommender system can be viewed as a search ranking



system, where the input query is a set of user and contextual

information, and the output is a ranked list of items. Given

a query, the recommendation task is to \_nd the relevant

items in a database and then rank the items based on certain

objectives, such as clicks or purchases.

One challenge in recommender systems, similar to the gen-

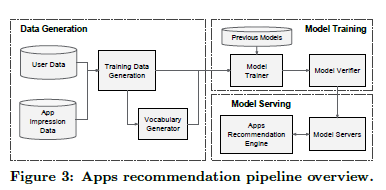
eral search ranking problem, is to achieve both memorization

and generalization. Memorization can be loosely defined as

learning the frequent co-occurrence of items or features and

exploiting the correlation available in the historical data.

Generalization, on the other hand, is based on transitivity



of correlation and explores new feature combinations that

have never or rarely occurred in the past. Recommenda-

tions based on memorization are usually more topical and

directly relevant to the items on which users have already

performed actions. Compared with memorization, general-

ization tends to improve the diversity of the recommended

items. In this paper, we focus on the apps recommendation

problem for the Google Play store, but the approach should

apply to generic recommender systems.

* How?

A query, which can include various user and

contextual features, is generated when a user visits the app

store. The recommender system returns a list of apps (also

referred to as impressions) on which users can perform cer-

tain actions such as clicks or purchases. These user actions,

along with the queries and impressions, are recorded in the

logs as the training data for the learner.

Since there are over a million apps in the database, it is

intractable to exhaustively score every app for every query

within the serving latency requirements (often O(10) mil-

liseconds). Therefore, the \_rst step upon receiving a query

is retrieval. The retrieval system returns a short list of items

that best match the query using various signals, usually a

combination of machine-learned models and human-de\_ned

rules. After reducing the candidate pool, the ranking sys-

tem ranks all items by their scores. The scores are usually

P(yjx), the probability of a user action label y given the

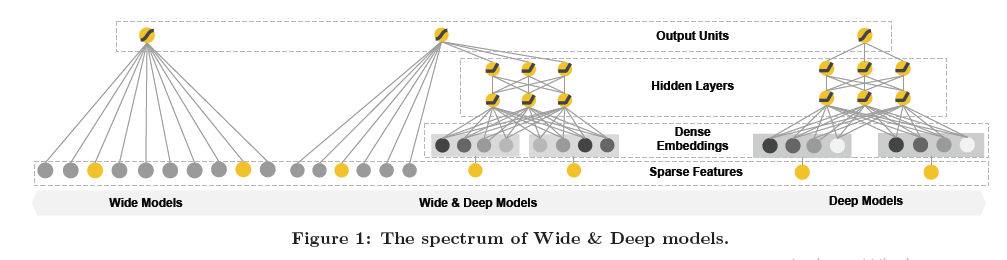
features x, including user features (e.g., country, language,

demographics), contextual features (e.g., device, hour of the

day, day of the week), and impression features (e.g., app age,

historical statistics of an app). In this paper, we focus on the

ranking model using the Wide & Deep learning framework.



* Model Training

The model structure we used in the experiment is shown in

Figure 4. During training, our input layer takes in training

data and vocabularies and generate sparse and dense fea-

tures together with a label. The wide component consists

of the cross-product transformation of user installed apps

and impression apps. For the deep part of the model, A 32-

dimensional embedding vector is learned for each categorical

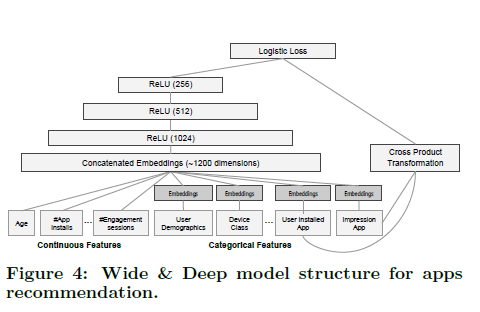
feature. We concatenate all the embeddings together with

the dense features, resulting in a dense vector of approxi-

mately 1200 dimensions. The concatenated vector is then

fed into 3 ReLU layers, and \_nally the logistic output unit.

The Wide & Deep models are trained on over 500 billion



examples. Every time a new set of training data arrives,

the model needs to be re-trained. However, retraining from

scratch every time is computationally expensive and delays

the time from data arrival to serving an updated model.

To tackle this challenge, we implemented a warm-starting

system which initializes a new model with the embeddings

and the linear model weights from the previous model.

Before loading the models into the model servers, a dry

run of the model is done to make sure that it does not cause

problems in serving live tra\_c. We empirically validate the

model quality against the previous model as a sanity check.

**Task 2 :**

**DeepFM: A Factorization-Machine based Neural Network for CTR Prediction**

Huifeng Guo⇤1 , Ruiming Tang2, Yunming Ye†1, Zhenguo Li2, Xiuqiang He2

1Shenzhen Graduate School, Harbin Institute of Technology, China

2Noah’s Ark Research Lab, Huawei, China

1huifengguo@yeah.net, yeyunming@hit.edu.cn

2{tangruiming, li.zhenguo, [hexiuqiang}@huawei.com](mailto:hexiuqiang%7d@huawei.com)

* Why?

Learning sophisticated feature interactions behind

user behaviors is critical in maximizing CTR for

recommender systems. Despite great progress, existing

methods seem to have a strong bias towards

low- or high-order interactions, or require expertise

feature engineering. In this paper, we show

that it is possible to derive an end-to-end learning

model that emphasizes both low- and highorder

feature interactions. The proposed model,

DeepFM, combines the power of factorization machines

for recommendation and deep learning for

feature learning in a new neural network architecture.

Compared to the latest Wide & Deep model

from Google, DeepFM has a shared input to its

“wide” and “deep” parts, with no need of feature

engineering besides raw features.

Comprehensiveexperiments are conducted to

demonstrate the effectivenessand efficiency of

DeepFM over the existing models for CTR prediction,

on both benchmarkdata and commercial data.

The prediction of click-through rate (CTR) is critical in recommender

system, where the task is to estimate the probability

a user will click on a recommended item. In many recommender

systems the goal is to maximize the number of clicks,

and so the items returned to a user can be ranked by estimated

CTR; while in other application scenarios such as online advertising

it is also important to improve revenue, and so the

ranking strategy can be adjusted as CTR⇥bid across all candidates,

where “bid” is the benefit the system receives if the

item is clicked by a user. In either case, it is clear that the key

is in estimating CTR correctly.

The key challenge is in effectively modeling feature interactions.

Some feature interactions can be easily understood,

thus can be designed by experts (like the instances above).

However, most other feature interactions are hidden in data

and difficult to identify a priori (for instance, the classic association

rule “diaper and beer” is mined from data, instead

of discovering by experts), which can only be captured automatically

by machine learning. Even for easy-to-understand

interactions, it seems unlikely for experts to model them exhaustively,

especially when the number of features is large.

Despite their simplicity, generalized linear models, such as

FTRL [McMahan et al., 2013], have shown decent performance

in practice. However, a linear model lacks the ability

to learn feature interactions, and a common practice is

to manually include pairwise feature interactions in its feature

vector. Such a method is hard to generalize to model

high-order feature interactions or those never or rarely appear

in the training data [Rendle, 2010].

Factorization Machine(FM) [Rendle, 2010] model pairwise feature interactions as

inner product of latent vectors between features and show

very promising results. While in principle FM can model

high-order feature interaction, in practice usually only order-

2 feature interactions are considered due to high complexity.

As a powerful approach to learning feature representation,

deep neural networks have the potential to learn sophisticated

feature interactions.

We propose a new neural network model DeepFM

(Figure 1) that integrates the architectures of FM and

deep neural networks (DNN). It models low-order feature

interactions like FM and models high-order feature

interactions like DNN. Unlike the wide & deep

model [Cheng et al., 2016], DeepFM canbe trained end

to-end without any feature engineering.

DeepFM can be trained efficiently because its wide part

and deep part, unlike [Cheng et al., 2016], share the

same input and also the embedding vector. In [Cheng et

al., 2016], the input vector can be of huge size as it includes

manually designed pairwise feature interactions

in the input vector of its wide part, which also greatly

increases its complexity.

**Our Approach**

Suppose the data set for training consists of n instances

(!, y), where ! is an m-fields data record usually involving

a pair of user and item, and y 2 {0, 1} is the associated label

indicating user click behaviors (y = 1 means the user

clicked the item, and y = 0 otherwise). ! may include categorical

fields (e.g., gender, location) and continuous fields

(e.g., age). Each categorical field is represented as a vector

of one-hot encoding, and each continuous field is represented

as the value itself, or a vector of one-hot encoding after

discretization. Then, each instance is converted to (x, y)

where x = [xfield1, xfield2, ...,xfiledj , ...,xfieldm] is a ddimensional

vector, with xfieldj being the vector representation

of the j-th field of !. Normally, x is high-dimensional

and extremely sparse. The task of CTR prediction is to build a

prediction model ˆy = CTR model(x) to estimate the probability

of a user clicking a specific app in a given context.

**DeepFM:**

We aim to learn both low- and high-order feature interactions.

To this end, we propose a Factorization-Machine based neural

network (DeepFM). As depicted in Figure 11, DeepFM

consists of two components, FM component and deep component,

that share the same input. For feature i, a scalar wi

is used to weigh its order-1 importance, a latent vector Vi is

used to measure its impact of interactions with other features.

Vi is fed in FM component to model order-2 feature interactions,

and fed in deep component to model high-order feature

interactions. All parameters, including wi, Vi, and the network

parameters (W(l), b(l) below) are trained jointly for the

combined prediction model:

ˆy = sigmoid(yFM + yDNN)



where ˆy 2 (0, 1) is the predicted CTR, yFM is the output of

FM component, and yDNN is the output of deep component.

**FM Component:**

The FM component is a factorization machine, which

is proposed in [Rendle, 2010] to learn feature interactions

for recommendation. Besides a linear (order-1) interactions

among features, FM models pairwise (order-2) feature interactions

as inner product of respective feature latent vectors.

It can capture order-2 feature interactions much more effectively

than previous approaches especially when the dataset is

sparse. In previous approaches, the parameter of an interaction

of features i and j can be trained only when feature i and

feature j both appear in the same data record. While in FM, it

is measured via the inner product of their latent vectors Vi and

Vj . Thanks to this flexible design, FM can train latent vector

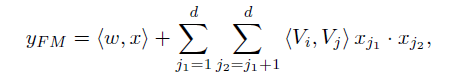
Vi (Vj ) whenever i (or j) appears in a data record. Therefore,

feature interactions, which are never or rarely appeared in the

training data, are better learnt by FM.

As Figure 2 shows, the output of FM is the summation of

an Addition unit and a number of Inner Product units:



where w 2 Rd and Vi 2 Rk (k is given)2. The Addition

unit (hw, xi) reflects the importance of order-1 features, and

the Inner Product units represent the impact of order-2 feature interactions.



**Deep Component:**

The deep component is a feed-forward neural network,

which is used to learn high-order feature interactions. As

shown in Figure 3, a data record (a vector) is fed into the neural

network. Compared to neural networks with image [He

et al., 2016] or audio [Boulanger-Lewandowski et al., 2013]

data as input, which is purely continuous and dense, the input

of CTR prediction is quite different, which requires a

new network architecture design. Specifically, the raw feature

input vector for CTR prediction is usually highly sparse3,

super high-dimensional4, categorical-continuous-mixed, and

grouped in fields (e.g., gender, location, age). This suggests

an embedding layer to compress the input vector to a lowdimensional,

dense real-value vector before further feeding

into the first hidden layer, otherwise the network can be overwhelming

to train.







It is worth pointing out that FM component and deep component

share the same feature embedding, which brings two

important benefits: 1) it learns both low- and high-order feature

interactions from raw features; 2) there is no need for expertise

feature engineering of the input, as required in Wide & Deep [Cheng et al., 2016].

**Relationship with the other Neural Networks:**

Inspired by the enormous success of deep learning in various

applications, several deep models for CTR prediction

are developed recently. This section compares the proposed

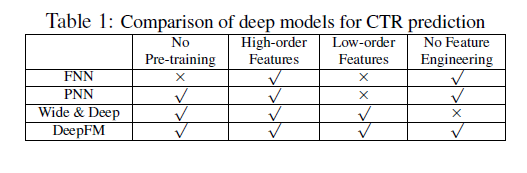
DeepFM with existing deep models for CTR prediction.



The relationship between DeepFM and the other deep models in four aspects is presented

in Table 1. As can be seen, DeepFM is the only model

that requires no pre-training and no feature engineering, and captures both low- and high-order feature interactions.



**FNN:**

As Figure shows, FNN is a FM-initialized feedforward

neural network [Zhang et al., 2016]. The FM pretraining

strategy results in two limitations: 1) the embedding

parameters might be over affected by FM; 2) the efficiency is

reduced by the overhead introduced by the pre-training stage.

In addition, FNN captures only high-order feature interactions.

In contrast, DeepFM needs no pre-training and learns

both high- and low-order feature interactions.

**PNN:**

For the purpose of capturing high-order feature interactions,

PNN imposes a product layer between the embedding

layer and the first hidden layer [Qu et al., 2016].

According to different types of product operation, there are three variants:

IPNN, OPNN, and PNN⇤, where IPNN is based on inner

product of vectors, OPNN is based on outer product, and

PNN⇤ is based on both inner and outer products.

To make the computation more efficient, the authors proposed

the approximated computations of both inner and outer

products: 1) the inner product is approximately computed by

eliminating some neurons; 2) the outer product is approximately

computed by compressing m k-dimensional feature

vectors to one k-dimensional vector. However, we find that

the outer product is less reliable than the inner product, since

the approximated computation of outer product loses much

information that makes the result unstable. Although inner

product is more reliable, it still suffers from high computational

complexity, because the output of the product layer is

connected to all neurons of the first hidden layer. Different

from PNN, the output of the product layer in DeepFM only

connects to the final output layer (one neuron). Like FNN, all

PNNs ignore low-order feature interactions.

**Wide & Deep:**

Wide & Deep (Figure 5 (right)) is proposed

by Google to model low- and high-order feature interactions

simultaneously. As shown in [Cheng et al., 2016], there is

a need for expertise feature engineering on the input to the

“wide” part (for instance, cross-product of users’ install apps

and impression apps in app recommendation). In contrast,

DeepFM needs no such expertise knowledge to handle the

input by learning directly from the input raw features.

A straightforward extension to this model is replacing LR

by FM (we also evaluate this extension in Section 3). This

extension is similar to DeepFM, but DeepFM shares the feature

embedding between the FM and deep component. The

sharing strategy of feature embedding influences (in backpropagate

manner) the feature representation by both lowand

high-order feature interactions, which models the representation

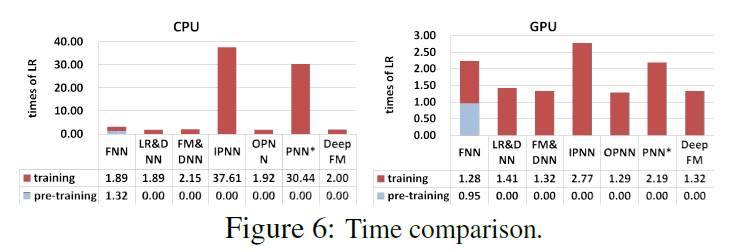
more precisely.

**Evaluation Metrics**

We use two evaluation metrics in our experiments: AUC

(Area Under ROC) and Logloss (cross entropy).

**Model Comparison**



We compare 9 models in our experiments: LR, FM, FNN,

PNN (three variants), Wide & Deep, and DeepFM. In the

Wide & Deep model, for the purpose of eliminating feature

engineering effort, we also adapt the original Wide & Deep

model by replacing LR by FM as the wide part. In order

to distinguish these two variants of Wide & Deep, we name

them LR & DNN and FM & DNN, respectively.6

**Efficiency Comparison**

The efficiency of deep learning models is important to realworld

applications. We compare the efficiency of different

models on Criteo dataset by the following formula:



including the tests on CPU (left) and GPU (right),

where we have the following observations: 1) pre-training

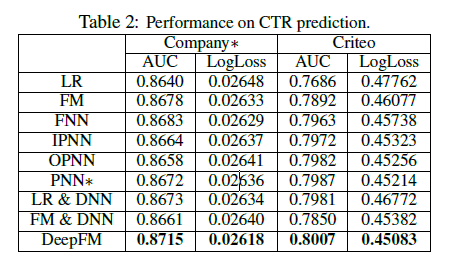
of FNN makes it less efficient; 2) Although the speed up of

IPNN and PNN⇤ on GPU is higher than the other models,

they are still computationally expensive because of the inefficient

inner product operations; 3) The DeepFM achieves

almost the most efficient in both tests.



Overall, our proposed DeepFM model beats the competitors

by more than 0.37% and 0.42% in terms of AUC and

Logloss on Company⇤ dataset, respectively. In fact, a small

improvement in offline AUC evaluation is likely to lead to a

significant increase in online CTR. As reported in [Cheng et

al., 2016], compared with LR, Wide & Deep improves AUC

by 0.275% (offline) and the improvement of online CTR is

3.9%. The daily turnover of Company⇤’s App Store is millions

of dollars, therefore even several percents lift in CTR

brings extra millions of dollars each year.

**Activation Function**

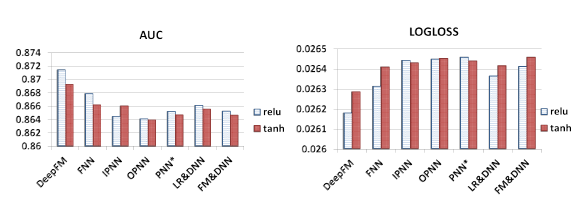
According to [Qu et al., 2016], relu and tanh are more suitable

for deep models than sigmoid. In this paper, we compare

the performance of deep models when applying relu and tanh.

As shown in Figure 7, relu is more appropriate than tanh for

all the deep models, except for IPNN. Possible reason is that relu induces sparsity.



**Dropout** [Srivastava et al., 2014] refers to the probability that

a neuron is kept in the network. Dropout is a regularization

technique to compromise the precision and the complexity of

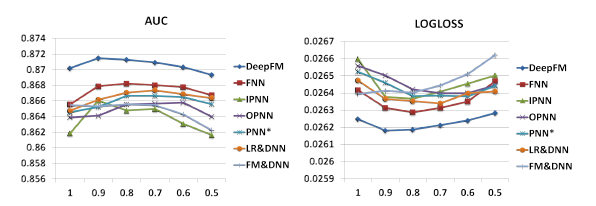
the neural network. We set the dropout to be 1.0, 0.9, 0.8, 0.7,

0.6, 0.5. As shown in Figure 8, all the models are able to reach

their own best performance when the dropout is properly set

(from 0.6 to 0.9). The result shows that adding reasonable

randomness to model can strengthen model’s robustness.



**Number of Neurons per Layer**

When other factors remain the same, increasing the number

of neurons per layer introduces complexity. As we can observe

from Figure 9, increasing the number of neurons does

not always bring benefit. For instance, DeepFM performs stably

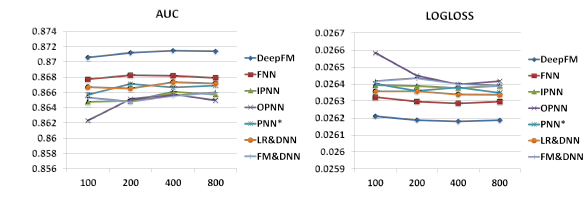
when the number of neurons per layer is increased from

400 to 800; even worse, OPNN performs worse when we increase

the number of neurons from 400 to 800. This is because

an over-complicated model is easy to overfit. In our

dataset, 200 or 400 neurons per layer is a good choice.



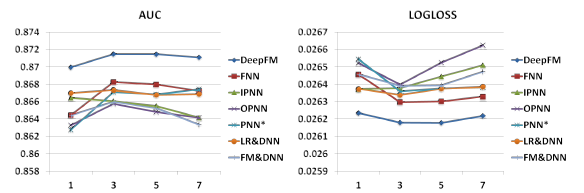
**Number of Hidden Layers**

As presented in Figure 10, increasing number of hidden layers

improves the performance of the models at the beginning,

however, their performance is degraded if the number of hidden

layers keeps increasing. This phenomenon is also because of overfitting.



**Network Shape**

We test four different network shapes: constant, increasing,

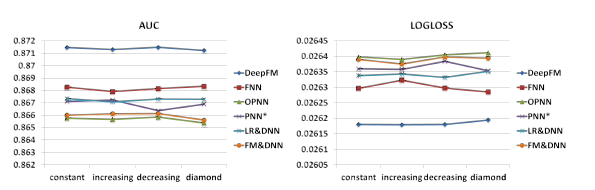
decreasing, and diamond. When we change the network

shape, we fix the number of hidden layers and the total number

of neurons. For instance, when the number of hidden layers

is 3 and the total number of neurons is 600, then four different

shapes are: constant (200-200-200), increasing (100- 200-300), decreasing (300-200-100), and diamond (150-300-150). As we can see from Figure 11, the “constant” network shape is empirically better than the other three options, which is consistent with previous studies [Larochelle et al., 2009].



**Related Work**

In this paper, a new deep neural network is proposed for CTR

prediction. The most related domains are CTR prediction and

deep learning in recommender system. In this section, we

discuss related work in these two domains.

CTR prediction plays an important role in recommender

system [Richardson et al., 2007; Juan et al., 2016; McMahan

et al., 2013]. Besides generalized linear models and

FM, a few other models are proposed for CTR prediction,

such as tree-based model [He et al., 2014], tensor based

model [Rendle and Schmidt-Thieme, 2010], support vector

machine [Chang et al., 2010], and bayesian model [Graepel

et al., 2010].

**Conclusions**

In this paper, we proposed DeepFM, a Factorization-Machine

based Neural Network for CTR prediction, to overcome the

shortcomings of the state-of-the-art models and to achieve

better performance. DeepFM trains a deep component and

an FM component jointly. It gains performance improvement

from these advantages:

1) it does not need any pre-training;

2) it learns both high- and low-order feature interactions;

3)it introduces a sharing strategy of feature embedding to avoid

feature engineering. We conducted extensive experiments on

two real-world datasets (Criteo dataset and a commercial AppStore dataset) to compare the effectiveness and efficiency of

DeepFM and the state-of-the-art models. Our experiment results

demonstrate that

1) DeepFM outperforms the state-ofthe-art models in terms of AUC and Logloss on both datasets;

2) The efficiency of DeepFM is comparable to the most efficient

deep model in the state-of-the-art.

There are two interesting directions for future study. One

is exploring some strategies (such as introducing pooling layers)

to strengthen the ability of learning most useful highorder

feature interactions. The other is to train DeepFM on a

GPU cluster for large-scale problems.