A Survey on Deep Learning for Click-Through Rate Prediction

University of Auckland Compsci 753, Semester 2 2020

Adrian Choi | Sahil Bahl | Josh Atwal

ABSTRACT

Click-through rate prediction forms an essential part of recommender systems in industrial applications such as online advertising. Deep learning has revolutionised the fields of natural language processing, speech recognition, image analysis, and is now achieving success in recommender systems due to its ability to capture non-linear and high-order relationships in data with minimal programmer oversight. This paper focuses on recent deep learning methods for click-through rate prediction; comparing their strengths and weaknesses, and summarising the current state of research in the field.

1 INTRODUCTION

The modern pace of technology has lead to an explosion of data with “big tech" companies thriving off of their ability to collect and utilise user data. Recommender systems are then necessarily developed to provide users with relevant content even in spite of a vast number of available options. The prediction of click-through rate (CTR)–the probability that a user clicks on a recommended item–is crucial for these systems, where candidate items can be ranked by estimated CTR, or by taking a weighted value of CTR with the expected benefit to the system from clicking on a certain item. Evidently, both parties involved benefit from users receiving relevant recommendations from an effective recommender system.

Learning implicit feature interactions behind user click behaviour is important for CTR prediction [1]. For example, feature interactions could be simple: such as an increase in food-delivery apps around meal times–or far more complex. While the simple and easily understood feature interactions can be manually engineered when the number of features is low, machine learning is key to automatically capturing patterns in more complex data.

We now introduce some of the concepts and models covered in this review.

Factorisation machines (FMs) are able to map sparse features to dense low-dimensional continuous space, modelling the interactions between the features with the inner product between latent vectors. This ability to handle sparse data makes them especially useful in the field of CTR prediction, where there are many users and many items. While FMs can be used to model higher-order features, in practice usually only second order interactions are considered due to the high complexity.

Deep learning is a subset of machine learning that concerns the study of artificial neural networks. First conceptualised by Hinton et al. [2] in 2006, deep learning is able to abstract low-level features and data into high-level feature representations, achieving revolutionary performance in image, sound, and text analysis through an analogy to the function of the human brain.

Deep Neural Networks (DNNs) are networks of neurons that apply a non-linear function to a linear combination of weights and inputs. Neurons are connected such that the output of one neuron is the input of the next one, and gradients with respect to a loss function are backpropagated through the network to optimise the weights and biases associated with each neuron. The linear cumulative effect of many non-linear activation functions means that DNNs are able to efficiently and implicitly learn non-linear interactions in the data without the necessity of manual feature engineering. Specialised types of DNNs such as convolutional neural networks have been successful particularly for image-based tasks, but when applied to CTR prediction end up having a bias towards neighbouring features due to the convolution operations. Recurrent neural networks are specialised for handling sequence data, such as audio or text, which makes them more suited for sequential click data.

2 LITERATURE SURVEY

In this section we cover different methods and approaches including and related to our seed paper.

2.1 Logistic Regression

Logic regression model has been the benchmark model for CTR estimation problems.

$$CTR = \frac{1}{1+e^{-z}}, Z = \Sigma_i w_i f_i(ad)$$

Where $f_i(ad)$ represents the ith feature of a suggested item. Due to the nature of such simple model, LR is incapable of capturing non-linear relationships between input features. Another limitation of using an LR model lies in the huge cost of feature engineering. Understanding all feature pair interaction is difficult. Hence, automation of such an intensive process has been explored in later research. Since LR is very easy to implement and has a strong interpretability, it is still widely used.

2.2 Product-based Neural Networks

Data collection in information retrieval tasks is mostly in a multi-field categorical form, which is then transformed into sparse binary features through one-hot encoding. Machine learning models such as logistic regression were proposed to particularly deal with such features and great results have been seen. The main drawback of these approaches is that feature engineering is required to capture high-order latent patterns. Recent research has focused on using Deep Neural Networks (DNNs) to automatically learn feature representations and to improve the interaction between multi-categorical data. Concretely, a methodology was presented [3] based on the concatenated embedding vectors which appear during the pre-training process of a factorisation machine. However, the ‘add’ operation of the built perceptron layer is not suitable for exploration of categorical data interaction. Other common approaches such as linear logistic regression (LR) and factorisation machines (FM) are seen in industrial applications, but they cannot be further applied to mine high-order latent patterns or to better learn feature representations.

Thus, in order to mine the latent patterns in data and learn local dependencies more effectively than existing methods, the paper

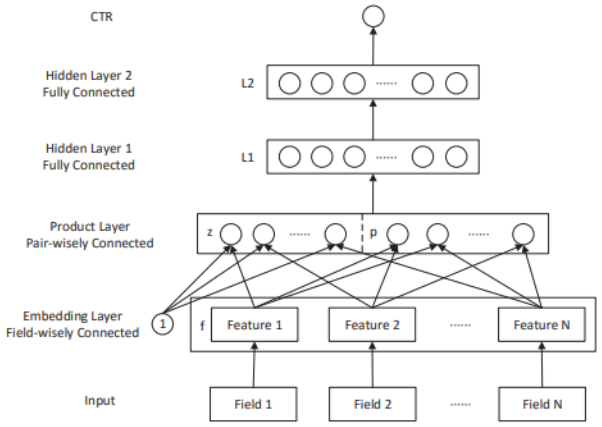

Figure 1: PNN Architecture

proposed Product-based Neural Network (PNN)[4]. This approach does not pre-train the initial embedding layer as MLPs. A product layer is built to model the inter-field feature interaction based on the embedded feature vectors. Two variants of PNNs (i) Inner Product-based NN, (ii) Outer Product-based NN, are also discussed in detail.

The architecture of the PNN model is shown in Figure 1 from a top-down perspective. Detailed implementation of PNN will be explained.

Input Layer: input data is represented as a multi-field categorical feature vector, where each field is one-hot encoded.

Embedding Layer: the embedding vector of each field i is the output of $f_i = W_0^i x[start_i : end_i]$, where x is the input vector containing multiple fields and x represents the one-hot coded vector for each field i. W are the parameters of the layer and each W_0^i is fully connected with each field i.

Product Layer: from the output of the embedding layer f_i, the product layer generates the quadratic signals p.

$$p = p_{i,j}, i = 1...N, j = 1...N$$

Note that $p_{i,j} = g(f_i, f_j)$ defines the pairwise feature interaction. Operations of g can be designed differently for different implementations. The two variants IPPN and OPPN will be discussed later. As illustrated in the Figure 1, a "1" constant signal is introduced to maintain the linear signals z.

$$z = (z_1, z_2, ..., z_N) \triangleq (f_1, f_2, ..., f_N)$$

Hidden Layer 1: this layer is fully connected with the previous product layer. Linear signals l_z and quadratic signals l_p are calculated through z and p as follows:

$$l_z^n = W_z^n \odot z$$

$$l_z^n = W_z^n \odot p$$

where the operation of tensor inner product is defined as $A \odot B \triangleq \sum_{i,j} A_{i,j} B_{i,j}$. Note that the multiplication result sums up to a scalar value. The activation function used for this hidden layer is defined as $relu(x) = max(0, x)$ due to its efficient computation and outstanding performance. The output l_1 is thus calculated as

$$l_1 = relu(l_z + l_p + b_1)$$

, where l_z, l_p and b_1 have dimension of D_1.

Hidden Layer 2: the output l_2 is computed as

$$l_2 = relu(W_2 l_1 + b_2)$$

using $l_1 \in \mathbb{R}^{D_1}$.

Output Layer: The output of PNN is real number between 0 and 1 representing the probability that a user clicks the given content (CTR estimation). The output is calculated as $\hat{y} = \sigma(W_3 l_2 + b_3)$, where $\sigma(x)$ is the sigmoid activation function: $\sigma(x) = \frac{1}{1+e^{-x}}$.

IPNN and OPNN will be discussed briefly in this section. Previously, the output of hidden layer 1 has a space complexity of $O(D_1 N(M+N)$ and time complexity of $O(N^2(D_1 + M)$. To reduce complexity, matrix factorisation is introduced. Concretely, first order decomposition on n-th single node is used reducing both space and time complexity to $O(D_1 MN)$. IPNN takes a pair of vectors as input and outputs a single scalar. On the other hand, OPNN takes a pair of vectors and outputs a matrix. The only difference between them is than in OPNN, feature interaction is defined as $g(f_i, f_j) = f_i f_j^T$, meaning in the product layer each $p, p_{i,j}$ is a square matrix with dimension $M \times M$.

Removing l_p component of the product layer, PNN would be identical to FNN. IPNN is quite similar with FM as (i) there is no hidden layer, (ii) the output layer is summed up with uniform weight.

Seven models are compared (LR, FM, FNN, CCPM, IPNN, OPNN, PNN) in the experiment demonstrated in the paper implemented with TensorFlow. Overall performance on the Criteo Dataset (an industry benchmarking dataset) is illustrated in Figure 2.

Model	AUC	Log Loss	RMSE	RIG
LR	71.48%	0.1334	9.362e-4	6.680e-2
FM	72.20%	0.1324	9.284e-4	7.436e-2
FNN	75.66%	0.1283	9.030e-4	1.024e-1
CCPM	76.71%	0.1269	8.938e-4	1.124e-1
IPNN	**77.79%**	**0.1252**	**8.803e-4**	**1.243e-1**
OPNN	77.54%	0.1257	8.846e-4	1.211e-1
PNN*	77.00%	0.1270	8.988e-4	1.118e-1

Figure 2: Overall performance on the Criteo Dataset. 7 models are compared.

IPNN has the highest AUC (area under ROC curve–a widely used metric to evaluate classification problems) among seven models compared with the lowest log loss. Both IPNN and OPNN are found better converge than other network models.

The biggest advantage of PNN compared to FNN is that with the addition of the product layer PNN gains better capability in exploring feature interaction, meaning expertise feature engineering is not needed. The experiments result above show than PNN outperforms LR, FM, FNN, CCPM in four metrics (AUC, RIG, Log Loss and RMSE)on two datasets (Criteo and iPinYou). Solutions to reduce space and time complexity are also discussed previously making PNN more efficient and scalable. Additionally, the 'product' and 'add' operations in PNN provide a potential way for rule representation.

[5] discussed several practical issues encountered when implementing PNN. It was noticed that FM pre-training does not always produce better results compared to initialisation with random small

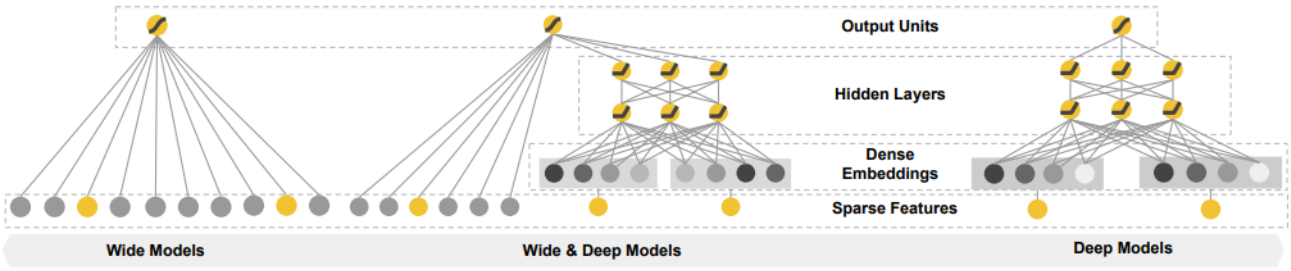

Figure 3: The spectrum of Wide & Deep models

numbers. Thus, it was believed that initialisation of embedding vectors depends on the nature of datasets, which requires further work. The original PNN has relatively inefficient space and time complexity, but this issue is addressed using IPNN. Additionally, the data in CTR prediction is often sparse and unbalanced in terms of the difference in the amount of positive values and that of negative values. Extremely small gradient might be produced when updating the parameters, hence gradient insensitivity can be a problem, (i) shallow models are found to be more sensitive to ϵ and (ii) unbalanced models are also more sensitive to ϵ and sometimes an empirical value $\epsilon = 10^{-8}$ is not a good choice. Like FNN, all PNNs ignore low-order feature interaction, which is a significant shortcoming. In the future work, PNN can be explored with more general and complex product layers. Feature vectors can also be visualised and explanations and their properties can be further investigated. Node representations can also be applied to other tasks.

2.3 Factorisation-Machine supported Neural Networks

Zhang et al. proposed pre-training a factorisation-machine as the input to a neural network with their factorisation-machine supported neural network (FNN) [6]. The approach is fairly straightforward: the feed-forward neural network component simply consists of three fully connected layers followed by a sigmoid function, and the factorisation machine follows the typical form:

$$w_0 + \sum_{i}^{N} w_i x_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_j x_j$$

Where w_0 is a global bias term, w_i a weighting associated with the i^{th} variable, and $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ an interaction term given by the dot product of the latent vectors that provides the advantage of allowing the model to work under sparsity while greatly reducing the overall complexity of the network. Another advantage easily seen here is that number of parameters, as well as the time for training and prediction are all linear. The main drawback of this approach is that the pre-training limits the ability of the FM to learn low-order interactions. Pre-training also adds additional overhead to the model, and means that the embedding parameters may also be over-affected by this step.

2.4 Wide & Deep Learning

Wide & Deep [7] was proposed by Google to model both low-order and high-order feature interactions simultaneously. It addresses the problems of *memorisation* and *generalisation* in recommender systems. The former can be loosely defined as learning the co-occurrence of items as well as exploiting the correlation of historical data. Generalisation can help explore new combinations between features that have not previously occurred. Recommender systems based on memorisation can return more directly relevant items to which users have previously performed actions, whereas generalisation contributes to the diversity of the recommended items.

The structure of Wide Deep frameworks is illustrated in Figure 3. Memorisation and generalisation components are achieved by jointly training the 'wide' component (a generalised linear model) and the 'deep' component (a neutral network component). Note that the term 'joint training' means all parameters are optimised simultaneously by considering both components as well as the weights of their sums during training time. Concretely, the output is calculated as follows:

$$P(Y = 1|\mathbf{x}) = \sigma(\mathbf{w}_{wide}^T [\mathbf{x}, \phi(\mathbf{x})] + \mathbf{w}_{deep}^T a^{(l_f)} + b)$$

, where Y is the binary class label and $\sigma(x)$ is the sigmoid function. $\phi(x)$ is the cross-product transformations of feature x, and b corresponds to the bias term.

The implementation of wide & deep model on the Google Play Store recommendation pipeline has three stages: (i) data generation, (ii) model training, (iii) model serving. In the first stage, user impression data within some time interval was used to generate training data. Then, the model is trained over 500 billion examples. To tackle the issue that the model requires re-training every time a new set of training data arrives, a warm-starting system was built which can initialise a new one with embeddings and linear model weights from the past data. During the last step, multithreading parallelism was used to optimise the performances by running smaller sized batches in parallel.

The results were evaluated in a few aspects: (i) app acquisition, (ii) serving performance. The former was conducted in an A/B testing model for three weeks. The results show that the recommender system was able to score over 10 million apps per second. Scoring all candidates with single multithreading and spitting each batch into smaller sizes can significantly reduce the latency to 14ms. It

was showed that the Wide & Deep model could lead to significant improvement on app acquisitions.

The biggest advantage of Wide & Deep model lies in its capability to capture both low and high order features. The 'wide' component is able to memorise sparse interactions with cross-product feature transformation. The biggest limitation of Wide Deep is that it requires expertise feature engineering for the input to the 'wide' component.

2.5 Deep FM

The seed paper for this report, DeepFM [1], was introduced in 2016 as a method that was able to consider low and high-order interactions simultaneously like Wide & Deep, but did so by incorporating a factorisation machine directly into the network architecture. This is unlike the FM supported Neural Network, which pre-trains a FM and uses it as input for the DNN component in a sequential manner. Like Wide & Deep, DeepFM consists of a wide component and a deep (DNN) component, but unlike Wide & Deep, the input to both parts of the network share the same input. This makes training much more efficient, and the factorisation machine approach meant that manual feature engineering was not required.

DeepFM was a breakthrough for its time. The authors compared DeepFM to nine other models for CTR prediction: linear regression, factorisation machine, FM-supported Neural Network, three variants of Product-based Neural Networks, and Wide & Deep. DeepFM outperformed all of these models in terms of AUC and Logloss on two large datasets, while also being among the fastest to train due the shared embedding and lack of a need for pre-training.

Figure 4 shows the Deep FM architecture. The first part is the **Dense Embedding layer**. Each field (feature) is encoded as one hot encoded vector.These sparse fields are fed into the fully connected embedding layer mapping each feature from a large sparse vector into a fixed size vector.These embeddings act as latent feature vectors which are learned during network training.

The output from the dense embedding layer is fed into the **Factorisation Machine Component**.The FM layer is responsible for learning the low order feature interactions. Inside the FM layer there are addition units that are responsible for mapping order 1 feature importance, that is direct impact of field i on the output. The inner product units in the FM layer are responsible for learning the pair wise feature interactions.Output of the FM layer is the summation of the addition and inner product units and is fed into the output layer where sigmoid function is applied to convert the final output into the click through rate probability.

The output from the dense embedding layer is also fed into the **Deep Neural Network Component**. The Deep part of the architecture is responsible to learn the high order feature interactions.This component is a multi layer perceptron where non linear relationships are learnt by applying non linear activation functions on each hidden unit of the network. The output from the Deep component is fed into the output layer consisting of the output units where sigmoid function is applied.

All network parameters are trained jointly for the combined prediction model. The final result is applying the sigmoid function on the summation of the output from the FM component and the Deep Neural Network combined.

Figure 5 shows the performance of DeepFM as compared to other known models at the time of publication. We can see that DeepFM outperforms all the other previous approaches and since the network is trained end to end sharing the same input for both the deep and DM component, DeepFM is faster to train as well as compared to other networks.

2.6 xDeepFM

One of the areas the authors identified for improvement in DeepFM was modifying the network architecture to better capture high-order interactions. Lian et al. introduce an extreme Deep Factorisation-Machine (xDeepFM) [8] network that extends the idea of DeepFM by generalising the factorisation machine component with a novel "Compressed Interaction Network" (CIN). That is, setting the depth and feature maps of CIN to 1 reduces xDeepFM to DeepFM. By extending the an FM into a network, the authors generate feature interactions in a explicit manner at the vector-wise level, rather than the implicit and bit-wise interactions generated by conventional DNNs. This is done by extending the idea of a cross network proposed in [9]. The cross operation for each hidden layer is calculated as

$$x_k = x_0 x_k^\top w_k + b_k + x_{k-1}$$

Where w_k, b_k, x_k are the weights, biases, and output of the $k-th$ layer respectively. A cross network is able to learn a special kind of type of high-order feature interaction, where each hidden layer is a multiple of x_0 the input. Note that there is also a similarity here to recurrent neural networks, where the input at the next hidden layer is dependent on the last hidden layer.

xDeepFM improves on this idea of a cross network with their CIN that measures interactions at the vector level rather than the bit level, measures high-order interactions explicitly, and whose complexity does not grow exponentially. The output of the $k-th$ layer in the CIN is now a matrix, $\mathbf{X}^k \in \mathbb{R}^{H_k \times D}$ where H_k denotes the number of feature vectors in the $k-th$ layer and D is the dimension of the field embedding. The matrix output of each hidden layer now becomes:

$$\mathbf{X}_h^k = \sum_{i=1}^{H_{k-1}} \sum_{j=1}^{m} \mathbf{W}_{ij}^{k,h} (\mathbf{X}_i^{k-1} \cdot \mathbf{X}_j^0)$$

Where $1 \leq h \leq H_k$, $\mathbf{W}^{k,h} \in \mathbb{R}^{H_{k-1} \times m}$, "$\cdot$" is the Hadamard product, and m is the number of fields in the data. Because $\mathbf{X}^k$ is dependent on the interactions between $\mathbf{X}^{k-1}$ and X^0, feature interactions are measured explicitly and the degree of interactions increases throughout the network.

xDeepFM exceeds the performance of DeepFM and other popular models for CTR prediction in terms of both AUC and Logloss across three different datasets, including one of the datasets used to evaluate DeepFM. The CIN does come with an additional computational cost over the DeepFM approach, however the authors suggest that a distributed version of xDeepFM may be developed which could be trained efficiently on a GPU cluster.

2.7 Deep Interest Network

Zhou et al. [10] highlighted a major shortcoming with the deep learning models proposed for Click-Through Rate. Most of these solutions involved mapping large scale sparse input features into

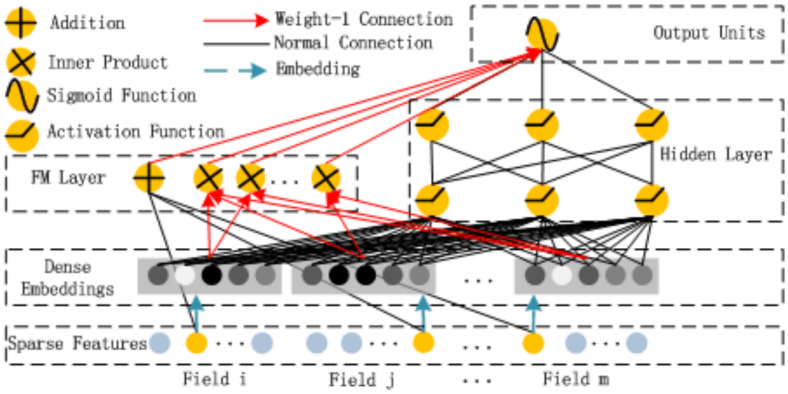

Figure 4: Deep FM Architecture: Left part of the network is the Factorisation Machine and right side of the network is the Deep Neural Network. Both receive input from the shared embedding layer

	Company∗		Criteo	
	AUC	LogLoss	AUC	LogLoss
LR	0.8640	0.02648	0.7686	0.47762
FM	0.8678	0.02633	0.7892	0.46077
FNN	0.8683	0.02629	0.7963	0.45738
IPNN	0.8664	0.02637	0.7972	0.45323
OPNN	0.8658	0.02641	0.7982	0.45256
PNN∗	0.8672	0.02636	0.7987	0.45214
LR & DNN	0.8673	0.02634	0.7981	0.46772
FM & DNN	0.8661	0.02640	0.7850	0.45382
DeepFM	**0.8715**	**0.02618**	**0.8007**	**0.45083**

Figure 5: DeepFM: Model Accuracy comparison on private company and Criteo data set

low dimensional embedding vectors and then transforming these into fixed-length vectors in a group-wise approach accross all the features. These are then concatenated together to be fed into a multilayer perceptron (MLP) to learn the nonlinear relations among features. The use of fixed length vectors limits the capability of Embedding & MLP based methods to learn user's diverse interests. Users might be interested in different kinds of goods simultaneously when visiting the e-commerce site.

An example from the paper highlighting this concept talks about how a young mother might have browsed goods including woollen coat, T-shits, earrings, tote bag, leather handbag and children's coat recently. This behaviour data gives hints about her shopping interests. When she visits the e-commerce site, the recommendation system displays a suitable ad to her, for example a new handbag. However, the displayed ad only matches or activates part of interests of this mother. In conclusion, interests of users with rich behaviours are diverse and could be locally activated given certain ads.

One way to capture this is to increase the dimension of the fixed-length vector. However this is not feasible beyond a point as increasing the vector length dramatically increases the size of learning parameters and adds the burden of computation and storage.

Zhou et al. [10] therefore proposed a novel deep interest network (DIN) model which pays attentions to the related user interests by soft-searching for relevant parts of historical behaviours and takes a weighted sum pooling to obtain the representation of user interests with respect to each candidate ad. This way DIN is able to capture the diversity characteristic of user interests under limited dimension and handles local context.

In the young mother example mentioned above, if the mother clicks on the displayed handbag suggestion, the recommendation network was able to soft-search her historical behaviour and find that she had browsed similar goods of tote bags and leather handbags recently. DIN simulates this process by paying attention to the representation of locally activated interests with respect to the given ad. This representation vector varies over different ads.

In order to understand the Deep Interest Network structure, we can first look at how most of the common Embedding & MLP based models are defined. The first layer is the **Embedding Layer**. This layer is used to transform high dimensional one hot encoded binary vectors representing different features into low dimensional dense representations.

The next layer is the **Pooling and Concat Layer**. Different users have different numbers of behaviours. As a result the number of non-zero values for multi-hot behavioural feature vector varies across instances, causing the lengths of the corresponding list of embedding vectors to be variable. Since the following fully connected MLP network will take fix length inputs, the embedding vectors are transformed into fix length vectors via a pooling layer.

Both embedding and pooling layers are applied group-wise across different features and map the original sparse features into multiple fixed length vectors. These vectors are then concatenated together to obtain the overall representation vector for the given instance. This is then followed by the Multi Layer Perceptron (MLP) layer. The fully connected layers are used to learn the combination of features.

The left part of Figure 6. shows the above architecture. The right part of Figure 6. illustrates the architecture of DIN. Compared with base model followed by the Embedding& MLP techniques, DIN structure has additional local activation units and the rest of the

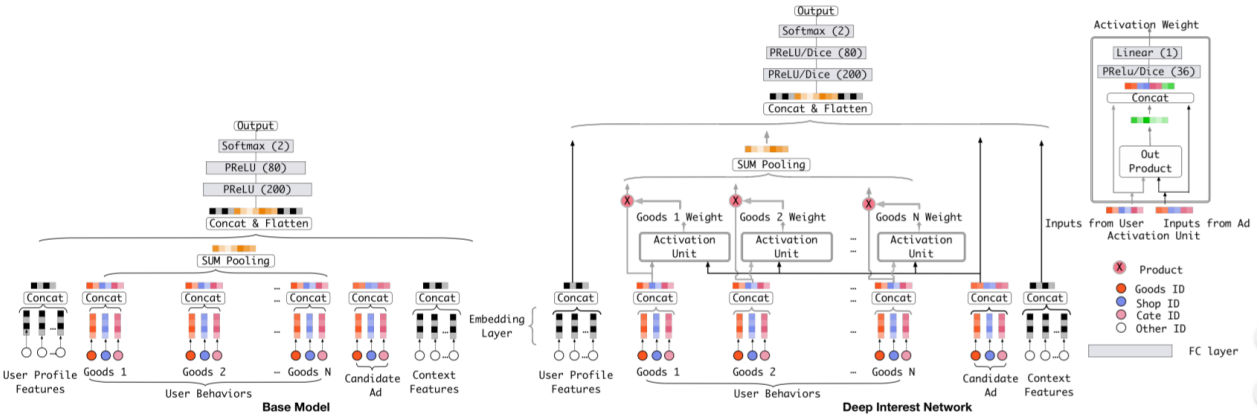

Figure 6: Deep Interest Network: Left Part denotes a typical Embedding&MLP based model architecture while right part of the figure shows the Deep Interest Network architecture

structure remains the same. The activation units are applied on the user behaviour features and they perform a weighted sum pooling to adaptively calculate user representation

$$v_U(A) = f(v_A, e_1, e_2, .., e_H) = \sum_{j=1}^{H} a(e_j, v_A)e_j = \sum_{j=1}^{H} w_j e_j$$

for a given candidate ad A, where

$$e_1, e_2, ..., e_H$$

is the list of embedding vectors of behaviours of user U with length of H.In this way, the vector varies over different ads for each user. a(·) is a feed-forward network with output as the activation weight as shown in the right part of Fig 6. Apart from the two input embedding vectors, a(·) adds the out product to feed into the subsequent network, which is an explicit knowledge to help relevance modelling.

As a result of these local activation units, DIN is able to perform better than existing Embedding & MLP models. Figure 7 shows results from the Deep Interest Network paper comparing their model's accuracy in comparison to the other models on the MovieLens and Amazon Data Set.

Model	MovieLens.		Amazon(Electro).	
	AUC	RelaImpr	AUC	RelaImpr
LR	0.7263	-1.61%	0.7742	-24.34%
BaseModel	0.7300	0.00%	0.8624	0.00%
Wide&Deep	0.7304	0.17%	0.8637	0.36%
PNN	0.7321	0.91%	0.8679	1.52%
DeepFM	0.7324	1.04%	0.8683	1.63%
DIN	**0.7337**	**1.61%**	**0.8818**	**5.35%**
DIN with Dice[a]	**0.7348**	**2.09%**	**0.8871**	**6.82%**

Figure 7: Deep Interest Networks: Model Accuracy comparison

3 DISCUSSION

We have seen a variety of techniques employed to the task of CTR prediction and how they have performed. We summarise a comparison of the models discussed in this work in terms of model capability in Table 1.

We first looked at how **Logistic Regression** is used to solve CTR prediction and it's inability of capturing high order interaction and huge reliance on feature engineering.

Factorisation Machine Based Neural Networks (FNN) and **Product Based Neural Networks (PNN)** are capable of capturing high order interactions with no feature engineering required but fail to capture low order interactions.

Wide And Deep is able to capture both high and low order interactions but relies on extensive feature engineering for the low order feature interactions and the wide and deep part have different inputs making overall training more complex.

DeepFM surpasses all the previous approaches by proposing a model architecture that combines the idea of Factorisation Machines to capture low order feature interaction and Deep Neural Networks for high order interactions. Since both of those components share the same input, DeepFM is trained end to end without any reliance on pre training or feature engineering.

xDeepFM extended the idea of DeepFM and achieved better results by generalising the idea of DeepFM; modifying the network architecture to better capture high-order interactions.

Zhou et al. stated how all these latest approaches are based on a common architecture pattern where an Embedding Layer converts highly sparse features into a fixed length vector and is followed by a Multi Layer Perceptron (Embedding & MLP paradigm). These approaches converted the diverse feature space into a fixed size vector which limits the capability of the network to capture the diverse interests of the user and local activations. This lead to the idea of **Deep Interest Networks (DIN)**. DIN introduced local activation units into the model to capture the related activation according to the candidate item. This way DIN is able to represent different user behaviour features for each candidate item and this

	No pre-training	High order features	Low order features	No feature engineering	Variable length embedding
FNN	×	✓	×	✓	×
PNN	✓	✓	×	✓	×
Wide & Deep	✓	✓	✓	×	×
DeepFM	✓	✓	✓	✓	×
xDeepFM	✓	✓	✓	✓	×
DIN	✓	✓	✓	✓	✓

Table 1: Model Comparison

adaptive representation helps to capture local context and diverse interests. DIN outperforms DeepFM and all the other previous models in terms of accuracy.

While DIN sets a new paradigm, the field of recommendation systems is always improving and new improvements have been proposed since DIN.

The authors of **Deep Interest Evolution Network (DIEN)** [11] identify that most interest models including DIN that aim to capture the diverse interest of users regard the behaviour as the interest directly. They fail to capture latent interest as it is hard to be fully reflected by explicit behaviour. The paper also highlights how user interest keeps evolving, hence capturing the dynamic of interest is important for interest representation. DIEN is able to improve the performance of CTR prediction by extracting latent temporal interests from explicit user behaviour and modelling interest evolving process.

One of the challenge to compare all these models together is that each model compares itself with others based on different data sets and different accuracy metrics. xDeepFM was evaluted on datasets from Criteo, Dianping, and Bing News using the AUC and logloss metrics, while DIN was evaluted on datasets from MovieLens, Amazon, and Alibaba only using the AUC metric. A future study could implement a robust comparison of these models, using the same evaluation metrics and datasets. This study would be able to conclude whether the DIN approach is superior over the Embedding & MLP paradigm, and hence where the direction of future research efforts should be directed. In our opinion, DIN networks do indeed seem to be a breakthrough in the field of CTR prediction.

4 CONCLUSION

CTR prediction is critical for many web applications, and comes with challenges such as modelling both high and low-order interactions, handling sparse data, and modelling diverse user interests. We have conducted a thorough survey on the state of research in the field of CTR prediction. While the Embedding & MLP paradigm has been dominant particularly since the publishing of DeepFM, the emerging DIN paradigm seems to be the way that research is heading. Unlike the fixed-length representations that came before it, DIN networks are able to capture the diverse interests of the user and local activations.

REFERENCES

[1] H. Guo, R. TANG, Y. Ye, Z. Li, and X. He, "Deepfm: A factorization-machine based neural network for ctr prediction," *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, Aug 2017. [Online]. Available: http://dx.doi.org/10.24963/ijcai.2017/239

[2] G. E. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[3] W. Zhang, T. Du, and J. Wang, "Deep learning over multi-field categorical data: A case study on user response prediction," 2016.

[4] Y. Qu, H. Cai, K. Ren, W. Zhang, Y. Y. Y. Wen, and J. Wang, "Product-based neural networks for user response prediction," 2016.

[5] Y. Qu, B. Fang, W. Zhang, R. Tang, M. Niu, H. Guo, Y. Yu, and X. He, "Product-based neural networks for user response prediction over multi-field categorical data," 2018.

[6] W. Zhang, T. Du, and J. Wang, "Deep learning over multi-field categorical data: A case study on user response prediction," 2016.

[7] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, "Wide deep learning for recommender systems," 2016.

[8] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun, "xdeepfm," *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, Jul 2018. [Online]. Available: http://dx.doi.org/10.1145/3219819.3220023

[9] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep cross network for ad click predictions," 2017.

[10] G. Zhou, C. Song, X. Zhu, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, and K. Gai, "Deep interest network for click-through rate prediction," 2018.

[11] G. Zhou, N. Mou, Y. Fan, Q. Pi, W. Bian, C. Zhou, X. Zhu, and K. Gai, "Deep interest evolution network for click-through rate prediction," 2018.