Multi Attentional Neural Factorization Machines: A State of Art Model for Click Through Rate Prediction

By

Batuhan Ozgur Basal

A Project Report

Presented to

Professor David Elizondo

MSc Intelligent Systems and Robotics Faculty of Technology

De Montfort University, Leicester

In Partial Fulfilment

Of the Requirements for the Degree

Master of Science

Abstract

Digital marketing relies heavily on click-through rate prediction, that attempts to forecast the likelihood that a customer will click on a certain product. An essential problem in CTR prediction is to extract the person's developing preferences through the user behavior pattern. Nevertheless, the majority of frameworks in use today neglect the notion that a series is made up of periods and also that user action can be segmented into several periods depending on when it occurs. CTR prediction, unfortunately, traditionally encountered number of difficulties. Having the input being high dimensional and sparse result from a large number of customers and products as well as the various depths of the vector space for various types of data. High-order feature connections also require a lot of professional expertise and take a long period of time. In this study, to obtain both low order and high order connections, we develop a brand-new model for CTR prediction named the Multi Attentional Neural Factorization Machine (MANFM). Four unique prediction concepts are combined: A new model inspired by factorization machines (FM) stacked on top of a multi-head self-attention mechanism employed to instantaneously detect meaningful and second order feature pairings, and linear model which is Logistic Regression (LR) to obtain the first order feature connections and Feed forward neural network to capture higher order feature combinations. The framework additionally includes an embedding layer to achieve integrated embedding computation of various data features, preventing feature diversity, sparsity, and excessive complexity. Furthermore, since we can perform end-to-end learning, feature engineering is not needed.

Acknowledgments

I want to express my gratitude to my supervisor, Professor David Elizondo, for his unwavering support and direction. During the study effort, David never stopped encouraging me and was always eager and willing to help in any way he could. I also want to express my gratitude to Ibrahim Ahmethan, a great friend and work colleague, for his guidance on the project's subject. Finally, a big thank you to my family who take a big part in my life and made it possible for me to finish this study.

CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction to CTR Prediction

1.1 Introduction

In 2020, the COVID-19 epidemic altered the nature of online advertising. It enabled advertisers to interact with customers in the comfort of their houses. The usage of marketing platforms including Instagram, Facebook, and Twitter surged as a result of this transition in 2021. Online marketing is anticipated to rule the most essential marketing platforms as 2022 progresses. Advertisers want to move more of their resources from standard methods to online marketing as a result. According to Statista [1], global expenditure on online marketing in 2019 hit a new record-high of $325 billion. The money spent on advertising was $332.84 billion in 2020. The statistics increased to $389.29 billion in 2021, continuing the uptrend. According to estimates, advertising spending will surpass $441 billion, breaking the previous milestone throughout all online media. Digital video sector, which is expected to rise by 76 %, will be the most popular online advertising sector. In addition, the Worldwide Digital Ad Budget 12-month Report from eMarketer [2] predicts that in 2022, online marketing spending will surpass $441.12 billion.

Click-based efficiency indices used in digital marketing, such as click-through rate (CTR), show the relevancy of adverts from the customers' point of view. Scientists and marketers alike agree that raising CTR is a practical means of ensuring the long-term growth of digital marketing networks.

How many people are likely to engage to invitation is a crucial topic for digital marketers to consider. The prediction of Click Through Rate is such subject that has recently attracted the interest of both academics and businesses. To achieve the business goals, this may result in increased sales, income, and user interaction.

The CTR prediction problem confronts two significant obstacles. High-dimensional and sparse features [3] are the first. Both the number of consumers and the products are enormous on massive e-commerce marketplaces. Precisely simulating feature interactions is the second obstacle [4]. Certain feature interactions are simple to comprehend, allowing specialists to develop them. The majority of alternative feature interactions, in opposition, are impossible to detect an inferable that only automatically caught by neural networks since they are concealed in the dataset [5]. Particularly with straightforward interactions, it appears difficult that professionals could fully model them, particularly when there are a lot of features. Additionally, the models would be simpler to overfit due to the increased feature dimensionality and sparsity brought about by the pairing of features that have high orders. The other approach would be to

reduce feature engineering, that identifies feature pairings naturally through boosting complexity of the system.

The approach we suggest in this study integrates four different models. Also, in order to incorporate sparse and high-dimensional input features an embedding layer is included in the system. We investigate several qualities individually.

First, the initial features are learned using a logistic regression algorithm. To learn relevant second-order interaction features, a multi-head self-attention system is combined with an element wise vector factorization model. In order to capture various feature connections in various vector spaces, we construct the multi-head self-attention system containing feedback connections to reflect information into vector spaces. We took an advantage of deep neural network technology for the concluding higher feature connections.

Our approach can instantly detect the connection of features and construct certain meaningful combinations of second-order features, which is critical for the model performance, as opposed to models that exclusively rely on MLP, FM, or attention mechanisms for feature interaction.

The model is also capable of handling massively sparse and high-dimensional data. This model was created in a flexible, parallel fashion, and the prediction component may be altered in accordance with particular requirements, giving the model significant versatility and adaptability. A technique for acquiring second order features relying on the multi self-attention system was presented. A feature pairing does not require feature engineering since an attention mechanism can take care of that. When contrasted to previous FM approaches, it will support the development of predictive performance by assisting in the discovery of highly beneficial pairing features on CTR prediction.

Predicting CTR is a crucial component of digital marketing and recommendation. We focus on the evaluation of models developed for CTR prediction as a result.

1.2 Problem Statement

Predicting the advertisement's click-through rate, that would be closely correlated to its effectiveness, is the crucial issue this study attempts to solve. When evaluating an advertisement's success, a number of important factors are taken into account. They include, ,the click through rate (CTR), the purchasing list, and the follow-up search activity. Since the adverts are provided to the company on a pay-per-click system, it is crucial for the editors to increase clicks, since doing so would directly assist the company in regards of the marketing they acquire.

The proportion between the number of interactions an advertising receives and the number of views it receives is known as the click through rate (CTR). The aggregate number of occasions a customer sees an advertising is referred to as an impression. This provides a prediction of the likelihood of a customer would click on the suggested advertising. For the purposes of this study, we assume that an advertising will become successful when it contains a higher proportion of clicks over views.

The different elements that affect an advertisement's CTR provide some other issue. There are specific apparent considerations, such as the advertiser's reputation, the customer's characteristics, and the search record. However, there could perhaps be a lot of undiscovered associated factors that influence the CTR. It is crucial to research and take into account these feature connections while creating models that forecast probabilities. This study focuses on feature interactions and analysis utilizing various state of art CTR models.

The dataset in this study primarily integrates the information about advertisements that publishers track with customer behavior records. For safety and privacy concerns, many of the features in the dataset are unidentified variables, and since the majority of them include categorical features, therefore datasets as a whole get extremely highly dimensional and sparse. To address the issue of estimating the click through rate, a new model has been developed. This case offers a lot of latitude for research, allowing one to overlook features that effects to the estimated likelihood are negligible.

1.3 Research Objectives

- Completing a detailed examination of the studies on CTR prediction and then reporting the results.
- Detailed examination of the data sets to be used in the experiment with an exploratory approach, then making the necessary pre-processing and more preparing for model testing.
- Testing state of art CTR prediction models with prepared datasets and recording their performances with AUC and Logloss scores. Then, performing performance analyzes and making comparisons on the models
- Installation of the model to be used in this study and providing a detailed explanation.
- Tuning the model hyperparameters to get the best out of our model performance and efficiency.

1.4 Introduction To Features of CTR

'Features' refers to existing data columns on a certain CTR example [4]. In addition to that, features can also define details regarding a given CTR instances, such as the age or height. These characteristics make up a limited, specialized collection of categorized information about products and users.

If you use the plural form of the word "features," you're referring to the collection of features given a CTR assignment or the collection of feature variables of a particular input for the CTR model. Datasets that will be used for CTR prediction has categorical features , hence the best way to express the data is as one-hot encoding.

Features contain details on the customer-product interactions that will cause a click. For example, if "banner position" is a feature of the product, in that case, assuming all other factors are identical, we could discover that an advertisement banner in a place where it will attract more attention of the customer has a higher chance to draw clicks than a poorly placed advertisement banner. Customers in a younger age group may engage with action games advertising more frequently than customers in other age categories, if customer-age and advertisement-category are two captured features.

You could notice that young customers have more attention in using food delivery applications such as Deliveroo on Saturdays than the typical customer when you take into account feature customer-age, product-category, and day. The connections between these features and their respective importance in estimating if it is clicked or not given a combination should be understood by a decent CTR predicting model.

These connections are known as "feature interactions or “feature connections." The "order" of an interaction refers to how many features are included in a certain connection. In the disciplines where they exhibit these complicated connections, the less typical higher-order feature connections we observe more than two features collaborating frequent. For instance, {a1, a2, a3, a4} are interactions of 4th order in $\log(a_1^2 + a_2 + a_3 a_4^2)$.

Finding and assessing the impact of feature interactions upon a customer-product CTR outcome is the challenge of CTR prediction. Finding the feature connections in the dataset may be helpful for a number of factors, such as:

- Knowing how the dataset's attributes relate to one another and how that affects prediction can help you evaluate models without bias by taking into account interacting impacts as well as the primary impacts.
- Employing interaction information can directly create meaningful models features of design to enhance model functionality.
- Enhancing the model performance.
- As a result, an important part of ongoing CTR development is on creating models without the requirement for explicit feature engineering as well as improving our understanding of relevant feature correlations.

1.5 Related Works

Regarding CTR prediction, Logistic Regression (LR) algorithm is employed at first [6]. Customers, products, goods and items are just a few of the many aspects that it is capable of properly utilizing. LR is a model that is straightforward, easy to implement, and parallelizable. But, on the other hand, Information loss may unavoidably result from its weak capability and inability to acquire information among features. Several researchers employ manual ways to integrate features in order to address this issue, however these approaches are incredibly ineffective and raise labor expenses.

The Degree-2 Polynomial (Poly2) [7,8] framework was suggested as a solution to this issue. It employs a basic method to mix any two properties. By going through one-hot encoding method, feature vectors happen to be extremely sparse. If two sparse features get integrated, the obtained vector could be even more sparse, making it challenging to train the system effectively upon the weights of majority feature pairings.

Rendle [9] presented Factorization Machines (FM) concept in 2010 to address POLY2 framework's problems. In comparison with POLY2 model, Factorization Machines uses the dot product two feature vectors to weight, which significantly lessens training process time and feature vector sparsity.

Even while FMs excel at handling large amounts of data sparsity, they miss the possibility of a feature may operate different while combining with different features taken out of the fields. For that reason, Juan et al. (2017) [10] devised the definition of connections and presented Field-aware Factorization Machines (FFMs). Nevertheless, it is limited to 2-order feature connections like original FM. In addition to that, the challenge of rapid growth of the complexity of a computational process will arise as the number of interactive features increases.

A huge collection of features greatly raises its modelling difficulty for FFMs in estimation of marketing CTR. In other words, the sequence of feature number × field number determines how many factors there are in FFMs. Pan et al. (2018) [8] enhanced FFMs with handling interaction factors and field combinations independently via weighting method they named "field-weighted factorization machines" for decreasing the number of factors that needed to get evaluated (FwFMs).

Later on, Field-matrixed factorization machines (FmFMs) were newly introduced by Sun et al. (2021) [11] as CTR prediction. The FmFMs describe feature correlations among field combinations like a matrix, surpass previous linear models and FM models, and perform similarly to sophisticated neural network models. A generalization of FMs and FwFMs is what FmFMs are.

Deep neural networks have lately seen a lot of achievement in several different academic domains. As a result, various deep learning based CTR prediction models have been presented by academics. Within a majority of models, a significant issue is how to accurately simulate feature interactions.

The Factorization Machine-based Neural Network (FNN) was suggested by Zhang et al. (2016) [12] The embedding layer of the system, which is modeled in a multi-layer perceptron , is pre-trained using FM. The features which have high order connections can get captured more effectively using this approach. The FM pretraining approach has two drawbacks: 1) FM may have a negative impact on the embedding variables; 2) the expense from the pre-training step, reduces the performance of the model.

In 2016, Qu et al (2016) investigated these feature connections with the introduction of a new layer and named it as the product layer. This layer is located above the embedding layer and below the fully connected layers. Thus, it provides the connection between the two layers, and named this model Product-based Neural Network (PNN) [13]. Also, this model, such as many deep networks, acquire slight low

order feature connections, that seem to be crucial for CTR prediction as well, as specified in [Cheng et al., 2016].

Cheng et al. (2016)[14] suggests an innovative dual system approach (Wide & Deep) which merges a linear ("wide") part with a deep part to explain low- and high-order feature correlations. In that paradigm, both parts demand a separate input, as well as the input of the "wide part" ultimately depends on heavy feature engineering**.** By replacing the broad section with FM and using the exact input, the DeepFM approach is able to resolve the issue [15]. In terms of CTR estimation, DeepFM method is recognized as the more sophisticated model.

The attention mechanism has been swiftly adopted in several domains with tremendous effectiveness when it was first introduced by Google in 2017 [16]. In 2018, Xiao et al. [17] introduced the Attentional Factorization Machine (AFM) concept, that is a development of the NFM model. It achieves a highly efficient outcome by connecting an attention mechanism with the feature intersection layer**.**

For the purpose of predicting CTR, Wang et al. (2022)[18] put out a brand-new model called SISA to represent user activity attention. First, they created an activity interest extractor component and specifically segment the customer's sequential activity into periods. To accurately reflect interest for every session, they use a self-attention network including bias coding.

1.6 Detailed Examinations of The Models Used In The Comparison

Later on, performance comparison of our model with some of the models mentioned below will be made. Detailed explanations of the models found in our experiment, along with the model equations, are given below.

1.6.1 Collaborative Filtering

Collaborative filtering [19], in its more recent and restricted definition, is a technique that automatically predicting (filtering) a user's preferences via gathering choices as well as liking feedback among other customers (collaborating). The collaborative filtering method's core premise is as if two people hold the identical view regarding a topic, they are higher probability to have it on a similar subject than two arbitrarily selected people.

1.6.2 Matrix Factorization

By multiplying two dissimilar types of components, matrix factorization can create hidden features. This technique is used in collaborative filtering to determine the connection among the properties of products and customers. We may want to estimate customer reviews of store products using the input data of customer reviews so that users may receive recommendations depending on the prediction. For instance, when a given film stars their favorite actor or their favorite genre, two individual users may rate it highly. In order to estimate a score with regard to the similarities in customer choices and activities, we are likely to identify these hidden features using the matrix factorization.

1.6.3 Degree-2 Polynomial & Factorization Machines

A degree-2 polynomial model may frequently successfully preserve the content in feature connectives, as demonstrated by Chang et al. [20]. Researchers also demonstrate that training as well as testing will be done significantly more quickly than with kernel approaches with utilizing a logistic regression on the essential shape of mappings for degree -2. That strategy, known as Poly2:

$$\phi_{\text{Poly2}}(\boldsymbol{w}, \boldsymbol{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} w_{h(j_1,j_2)} x_{j_1} x_{j_2}$$

where j1 and j2 are transformed into a natural number by the function h(j1, j2).

It is an improvement of a degree 2 polynomial model made to efficiently identify feature correlations in large dimensionality sparse data. For instance, the Factorization Machines approach for a click forecasting method could catch click rate trends seen whenever advertising from one class of advertisement is shown on sites from another. Applications involving large dimensional sparse databases, including click forecasting and product suggestion, benefit from factorization machines.

Order 2 FM model shown below:

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

Vi and Vj are the two feature vectors with dot production <,>.

1.6.4 Field Aware Factorization Machines

Each feature in Factorization Machines contains a single hidden vector that may be used to determine the latent relationship among some features. [21] considered a feature "A" for an illustration; wA is utilized to discover the latent effects of "B" and "C" (wA wB and wA wC). The hidden impacts of (A, B) and (B, C) might vary, though, since B and C come from other fields. Therefore, every feature in FFMs contains a number of hidden vectors. One of those is employed to complete the space of inner product based in the domain of those features.

$$\phi_{\mathrm{FFM}}(\boldsymbol{w}, \boldsymbol{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} (\boldsymbol{w}_{j_1, f_2} \cdot \boldsymbol{w}_{j_2, f_1}) x_{j_1} x_{j_2}$$

here, the fields of j1 and j2 are, correspondingly, f1 and f2.

1.6.5 Wide and Deep

By integrating the advantages of memorization and generalization in recommendation process, Cheng et al. (2016) [22] introduced Wide & Deep model—together formed "Wide" logistic regression model and "Deep" feed forward neural network.

Wide and Deep model is an innovative CTR model research that Google released in 2016. By the period of release, CTR Machine Learning algorithms concentrated on two different kinds of structures: deep structures, which employed neural networks, and shallow structures, like linear models, that did not. The Wide & Deep concept aims to combine both techniques to maximize on each one's unique advantages. Whereas deep systems excelled at generalizing feature behavior, shallow systems were effective at remembering the essential features and their connections in input features. A shallow concept and a deep concept are simultaneously calculated and combined by Wide & Deep. The ultimate estimate would then be produced a sigmoid function which was used to combine both concepts. Output of the model shown below:

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

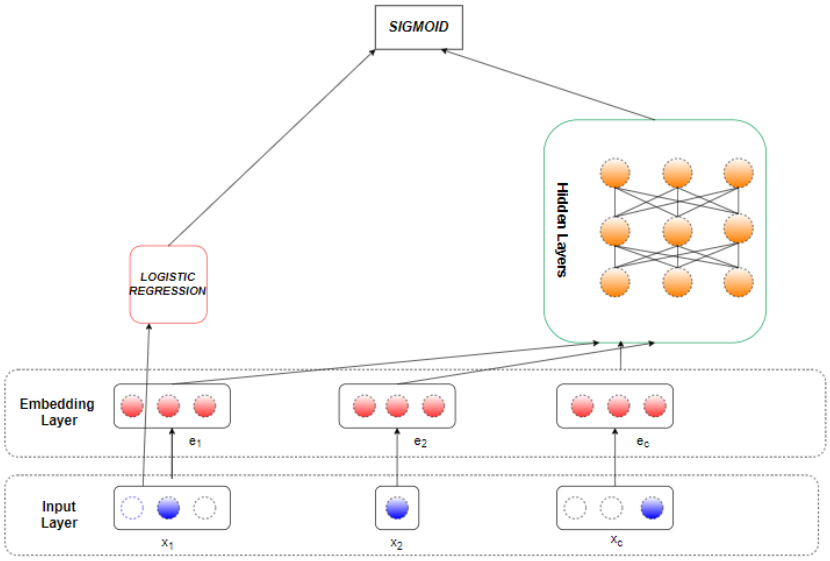


Figure 1 - The architecture of Wide and Deep Model

1.6.6 Deep and Cross Network

Cross features can be learned by Deep and Cross Network developed by Wang et al (2017) [23] more quickly and explicitly. A cross network featuring many cross layers which simulates explicit feature connections comes first, backed with an embedding layer and is combined with a deep neural network that represents hidden feature connections. At the end, the DCN is formed by combining the deep and cross network together.

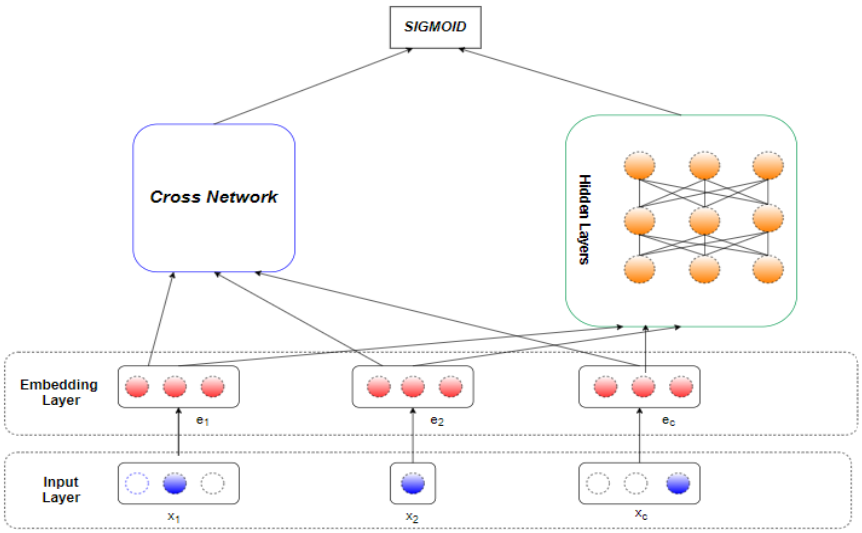


Figure 2 - The architecture of Deep and Cross Network Model

1.6.7 DeepFM

Combining low and high order feature combinations are what they hope to accomplish. In order to do that, Guo et al. (2017) [24] suggested a new state of art CTR prediction model named factorization-machine-based neural network (DeepFM). Two elements, the Factorization Machines part, and the deep part (feed forward neural network), build into DeepFM and they both use the exact input. Weighing first order feature relevance using a numerical value called wi and a hidden vector The effect of connections of various features are measured using vi. To represent second order feature connections, the FM part of Vi is supplied, while the deep part of Vi is supplied to represent high order feature connections.

$$\hat{y} = sigmoid(y_{FM} + y_{DNN})$$

yFM and yDNN are the outputs of FM and deep parts respectively.

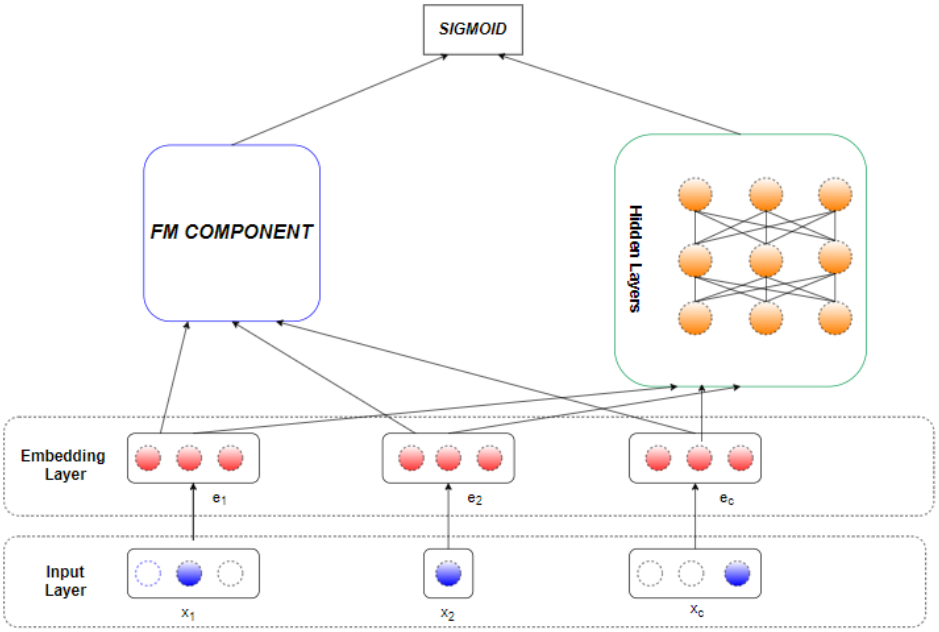


Figure 3 - The architecture of DeepFM Model

1.6.8 AutoInt

Song et al. (2018) [25] build a new model namely AutoInt which simulates high-order feature correlations of input data. Both numerical and categorical data features can be processed by this model In particular, the identical embedding layer is used to express both the numerical and category features. After that, it is suggested to explicitly describe the feature relationships inside the low-dimensional field using a multi-

head self-attentive neural network. Various degrees of feature pairings of the input may be simulated using multiple layers of the attention mechanism.

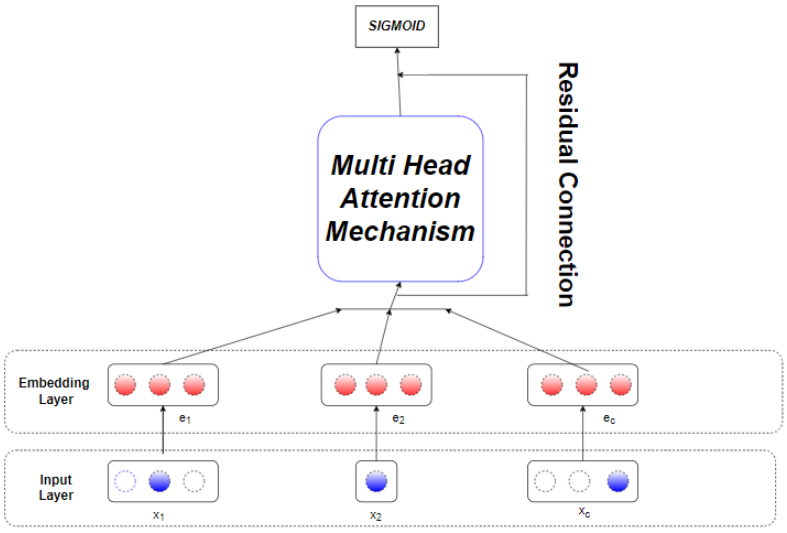


Figure 4 - The architecture of AutoInt Model

In summary, It is clear that current models either depend on feature engineering or are biased toward low- or high-order feature connection. we demonstrate in this study that it is feasible to design a learning method that can learn feature connections of all orders from beginning to finish.

CHAPTER 2

Software Framework

2.1 Introduction

The challenge of predicting click-through rate (CTR) is first explicitly defined as follows:

Consider x which is an element of R^k stand for the conjunction of the input features of users and the features of the items, wherein features that have unique values are expressed using one-hot encoding and k is the dimensionality of the combined features. The goal of the click-through rate forecasting challenge is to estimate the likelihood that user would click on item in consideration of the x which is a feature vector. Taking x as the source of the features and using linear models like Logistic Regression [6] are an easy way to forecast CTR.

The model would, nevertheless, be readily overfitted because the feature vector is extremely sparse and has large number of input features. In order to express the input information, spaces that have low dimensionality are preferred. Additionally, using higher-order multimodal features is essential to provide superior prediction efficiency, as demonstrated in the related work section.

2.2 Datasets

Criteo [26] and Avazu [27] dataset will be used for the model.

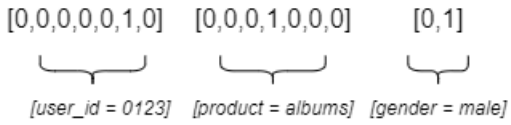
The Criteo dataset, a highly well-known digital advertising data comprising statistics about every advertisement's display including related consumer clicking response, is frequently utilized in the assessment of CTR models. Inside the Criteo database, there are 13 continuous and 26 categorical fields. It is used for ctr prediction challenge on kaggle.com [26]. All features in the dataset are anonymous due to privacy rights. Features starting with the letter “I” are numerical, while those starting with “C” are categorical values.

The dataset appears to include 21 separate properties, with the click information property—which is binary number and has values of 0 for no clicks on ads and 1 for clicks—being the target feature.
8 features are included in the dataset as unidentified features. These categorical values, which may contain specific details regarding the profiles of the customers and marketers. The CTR data in the Avazu dataset covers multiple days and is arranged sequentially. There are 40 million clicking entries inside of it. There are 24 data fields that represent the component parts of an individual ad display.

The datasets from Avazu and Criteo are massive. We mainly employ a portion of the training set for the actual database because of resource limitations in the study, specifically the first 10,000,000 rows of the training dataset. Next, each dataset is separated into a training set (80%), and a test set (20%) for the study.

2.3 Data Analysis

The input parameters for a CTR prediction are often unrelated and categorical, apart from those for voice detection or picture categorization. Raw input characteristics are often transformed through one-hot encoding into high-dimensional sparse features as representation. For instance, the customer IDs are 0123 and 0124, the products are books and albums, and the genders are male and female. Single input sample may be changed to the following via using the one-hot encoding:



Following conversion, the dimensions of the mentioned elements, particularly the user ID and products category, will grow significantly. One of the 1000 possible variables of the size of products feature will grow to 1000 after encoding, if the number of products is 1000.

2.4 Data Preprocessing

In this study, we essentially normalize the dense features between 0 and 1 values. We may also attempt log normalization or singular value decomposition as conversion techniques. Next, we create columns for features as both dense and sparse features respectively.

Then, The missing categorical data in the dataset is replaced by -1. Likewise, the missing numerical values are replaced with 0. Next, for the categorical features, we applied label encoding and one hot encoding using OneHotEnding() and get_dummies() functions provided by sklearn and pandas libraries respectively. The differences will be reported by trying two different functions on the same dataset. Finally, for the numerical values, we applied MinMaxScaler function to normalize the values between 0

and 1. Since an end to end model was built, the feature engineering method was not applied. Finally, We divided the dataset into two sets as 20% test and 80% train set.

2.5 Model Framework

In this study, Multi Attentional Neural Factorization Machines model is suggested to collect features of multiple orders. Figure 1 illustrates the model's structural layout. Such a framework has the following advantages:

The input features which are vectors, have high dimensionality and sparsity translated into a field which is less dimensional. An embedding component is in charge of reflecting the input vector through a low-dimensional field, as seen in the illustration. The result is an embedding vector, that the estimation algorithms can employ as input.

It simulates multi-order feature interactions using a variety of techniques. Among these, LR copies the input feature while keeping its memory-preserving capabilities. To capture the second order feature connections, we decide to use multi head attention mechanism, in that way, we will be able to capture meaningful dual feature connections. For the output of this component, the interaction between two features is modelled by Factorization Machines [9], since Logistic Regression [6] has already handled the first order connections, we keep the FM model's feature interaction component and remove its first order connection component, then, we connected the two model together. For the higher order connections of the feature vectors are modelled by a Feed forward neural network. Furthermore, the outputs from such three components are combined, and a sigmoid function predicts the click rates. We will then go over the specifics of our suggested approach.

2.5.1 Input

We combine all the fields to describe customer identification, product features, as well as other elements using sparse vectors.

$$X = \{X_1, X_2, X_3, X_4, X_5, X_6, X_C\}$$

Here x_i is a description for the i_{th} field and C is the total number of fields. A one-hot vector is used to describe input features when it is a categorial feature. A normalized integer value is used to express the input features if it is a numerical feature.

2.5.2 Embedding Layer

The feature collection produced by one-hot encoding will be sparse and have large density. Those indices matching to non-zero elements are often saved in the technical area thanks to space saving concerns, that makes feature designing more challenging. Throughout this study, we employ low-dimensional vectors to express all categorial variables by embedding those features to low-dimensional domain. The embedding procedure is written as follows:

$e_i = v_i \ . \ x_i$

In addition to categorized data, the input usually includes integer valued features. We further incorporate features that have numerical elements to the low-sized domain to evenly describe those features.
This procedure is denoted as follows:

$e_m = V_m \cdot x_m$

(Where, e_m = Embedding Vector, V_m = Embedding Matrix, x_m = Input Feature)

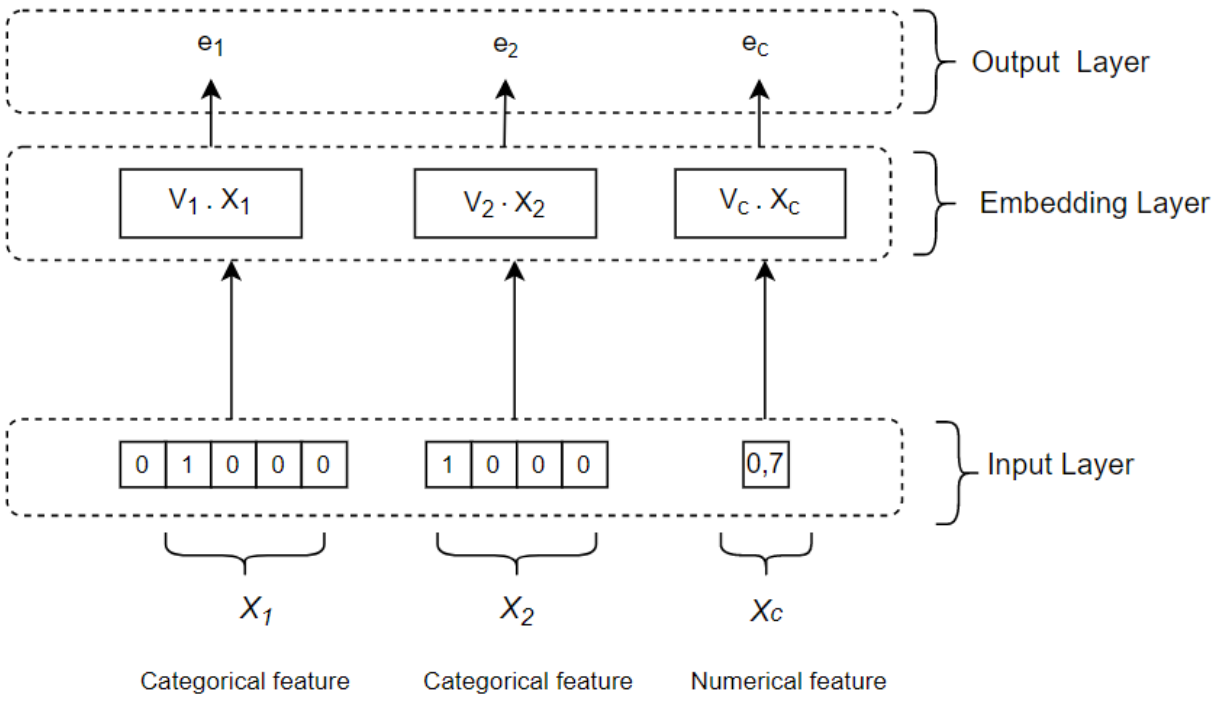


Figure 5 – The architecture of the Embedding Layer

2.5.3 Prediction Layer

2.5.3.1 Logistic Regression Component

Logistic Regression, which is the one of the simplest and most effective supervised machine learning classification algorithms, was used for the first order feature interaction. Many studies [6,14] have proven the effectiveness of this algorithm. It is a quantitative analysis approach that uses previous findings from a data set to estimate a binary result, for example True or False. Below is the equation of Logistic Regression algorithm.

$$y_1 = w_0 + \sum_{i=1}^{c}(w_i x_i)$$

Where y_1 is output of the first model, w_0 is the global bias**,** w_i is the bias for the i_{th} feature, x_i is the input feature for the i_{th} feature.

Although the logistic regression model is straightforward, understandable, and easy to deploy, however, their interpretive ability is restricted. In order to capture the feature connections in the data set, which are a little more challenging to observe, second order feature interaction has been tried which provided by using another classification algorithm, Factorization machines [9].

We must construct high-order interacting features to uncover additional valuable insights buried in the dataset. Therefore, we offer a multi-head self-attention and factorization machine based high order dynamic feature modelling approach. There are x number of steps in the computation procedure.

2.5.3.2 Multi-head Attention Mechanism Component

Bahdanau et al. (2014) [28] proposed the attention network to solve the congestion issue that occurs when using a fixed length vector for encoding considering the decoder will only has restricted accessibility to the input's data. The complexity of its depiction could be compelled to remain the identical as in shorter or more basic sequences, which is anticipated to be particularly troublesome for longer and complicated sequences.

Multi-head Attention mechanism is a component of attention networks that goes over attention numerous periods in parallel. The predicted dimension is subsequently created by linearly combining the separate attention outcomes. Multiple attention heads enable for diverse attention to be provided towards various sequence sections, instinctively.

Lately, multi-head self-attention mechanism [29] demonstrated outstanding effectiveness in modelling complex interactions. It excels at modelling random word connections in sentence translation [30] and, it has been effectively used to identify node commonalities for embedding a graph [31]. Furthermore, we expand on this most recent methodology to construct the relationships across several feature variables.

For each embedding vector we have after the embedding layer, fed into the multi-head attention network as input vectors.

$$S_i^{h:Query} = W_{Query}^{(h)} e_i$$

$$S_i^{h:Key} = W_{Key}^{(h)} e_i$$

$$S_i^{h:Value} = W_{Value}^{(h)} e_i$$

Where $W_{Query}^{(h)}, W_{Key}^{(h)}, W_{Value}^{(h)}$ are the matrices that concerts input dimension to attention mechanism.

Second step is to determine the degree to which one feature, e_i, resembles another, e_l. The selection of a function can varies depending on the specific circumstances. The dot product is chosen in this study to optimize the system and help accelerate learning.

$$\emptyset^{(h)}(e_i, e_l) = < S_{e_i}^{h:Query}, S_{e_i}^{h:Query} >$$

$\emptyset^{(h)}(,)$ is function that will be used for calculation of the similarities between two feature vectors.

Then, attention function normalized by SoftMax function.

$$a_{i,l}^{(h)} = \left(\frac{\exp\left(\emptyset^{(h)}(e_i, e_l)\right)}{\sum_{c=1}^{C} \exp\left(\emptyset^{(h)}(e_i, e_l)\right)}\right)$$

Sigma is used to generate a unique combination of feature.

$$n_i^{(h)} = \sum_{c=1}^{C} a_{i,l}^{(h)} \, S_i^{h:Value}$$

$n_i^{(h)}$ is the h^{th} head of attention network and related to feature vector e_i.

With the help of the attention network, it signifies a unique combination feature. A feature might potentially get utilized in various pairing features. Employing a multi-head attention system, which generates several subsets and learns various feature correlations, we may accomplish that.

The outcome indicates a novel feature combination discovered via this approach because it combines feature e_i and its pertinent features. A feature is therefore expected to become engaged in several combination of features, thus this can be accomplished by employing multiple heads that split out vector subspaces and individually learn various feature connections.

Considering there are "H" vector spaces, the feature "n_i" is obtained by concatenating the findings from each individual vector space. This can be described as:

$$n_i = n_i^{(1)} \oplus n_i^{(2)} \oplus n_i^{(3)} \oplus n_i^{(4)} \oplus n_i^{(5)} \oplus n_i^{(H)}$$

The easiest and most popular computing technique used in deep learning is the concatenation operator () [32] , that joins the variables collectively. This operator's benefits include being simple to use, comprehend, and compute. H is the number of the attention head.

Finally, the residual component was used across two levels of the attention model, maintaining certain original feature material enabling the following phase to keep learning, aiming to improve learning outcomes and acquire extra information.

$$n_i^{RES} = ReLU(n_i + W_{RES}\, e_i)$$

where ReLU(x) = MAX(0,x), which is an activation function commonly used in neural networks, and W_{RES} are projection matrix. In order to add these values together, a projection matrix was used to multiply with the feature vector which came from embedding layer, because the output of the attention network has different output dimension.

2.5.3.3 Elementwise Factorization Machines Component

Factorization Machines (FMs) are a family of universal classifiers that efficiently guess variables despite substantial sparsity when dealing with feature vectors. FMs anticipate the objective as follows formally:

$$y_{FM}(x) = w_0 + \sum_{i=1}^{C} w_i x_i + \sum_{i=1}^{C} \sum_{j=i+1}^{C} < v_i v_j > x_i x_j$$

$w_0 + \sum_{i=1}^{n} w_i x_i$ is the linear part of the FM model where w_0 the is global bias and w_i is the scalar weight.

$\sum_{i=1}^{C} \sum_{j=i+1}^{C} < v_i v_j > x_i x_j$ is the feature interactions part based on matrix factorizations.

$< V_I V_J >$ is the dot product between features i and j. It models how these two features interact with one another.

In order to predict every feature connection which, the i_{th} feature entails, a feature vector v_i is provided. Additionally, all predicted feature connections w_{ij} maintain a standard weight of 1. In reality, it frequently happens that some features are not important for forecasting. We can think of connections including unimportant features like noise which may not affect the forecast. Unfortunately, FM calculates every potential feature interaction using the identical weight, that could negatively affect how well it generalizes in terms of model performance. Since first order feature interactions were established using logistic regression. The linear portion was left out of the FM model, but it is relatively possible to include it. Next, the element-wise interaction layer is described in more context below.

We introduced an Element-wise Interaction Module which is influenced by FM, which employs the dot product to express the connection among each couple of features.

$$\sum_{i=1}^{C} \sum_{j=i+1}^{C} (v_i \odot v_j) x_i x_j$$

$(v_i \odot v_j)$ shows the Hadamard product (elements wise product) between vectors and creates another vector Unlike FM which creates scalar value after the dot product.

Then, new feature combinations from the attention mechanism (depending on the number of heads) are fed to the FM model.

$$y_2 = \sum_{i=1}^{C} \sum_{j=i+1}^{C} N_i^{RES} (v_i \odot v_j) x_i x_j$$

2.5.3.4 Multi-Layer Perceptron Component

An embedding vector a which is obtained by the sum of all embedding vectors from the embedding layer is fed into the feed-forward neural network that makes up the deep portion in order to acquire high-order feature connections.

$$y_{(0)} = [e_1 + e_2 + e_3 + e_4 + e_C],$$

Following layer's equation will be:

$$y_{(l+1)} = f(W_l y_l + b_l)$$

y_l , $y_{(l+1)}$ are the l-th and (l+1)th hidden layers respectively, w_l , b_l are weight and bias for l-th deep layer and f(.) is ReLU function.

$y_3 = f(W_l y_2 + b_l)$ is the output of the multi-layer perceptron.

2.5.4 Concatenating Layer

At the end, The three systems combine to create the outcome: y_1 for the Logistic Regression model, y_2 for the multi head attention mechanism with factorization machines model, and y_3 for the deep feed forward neural network. The three results are then combined to predict the Click through rate.

$$p = \sigma(y_1 \oplus y_2 \oplus y_3 + b)$$

Where, $\oplus$ is concatenation operator and b is bias and σ is sigmoid function.

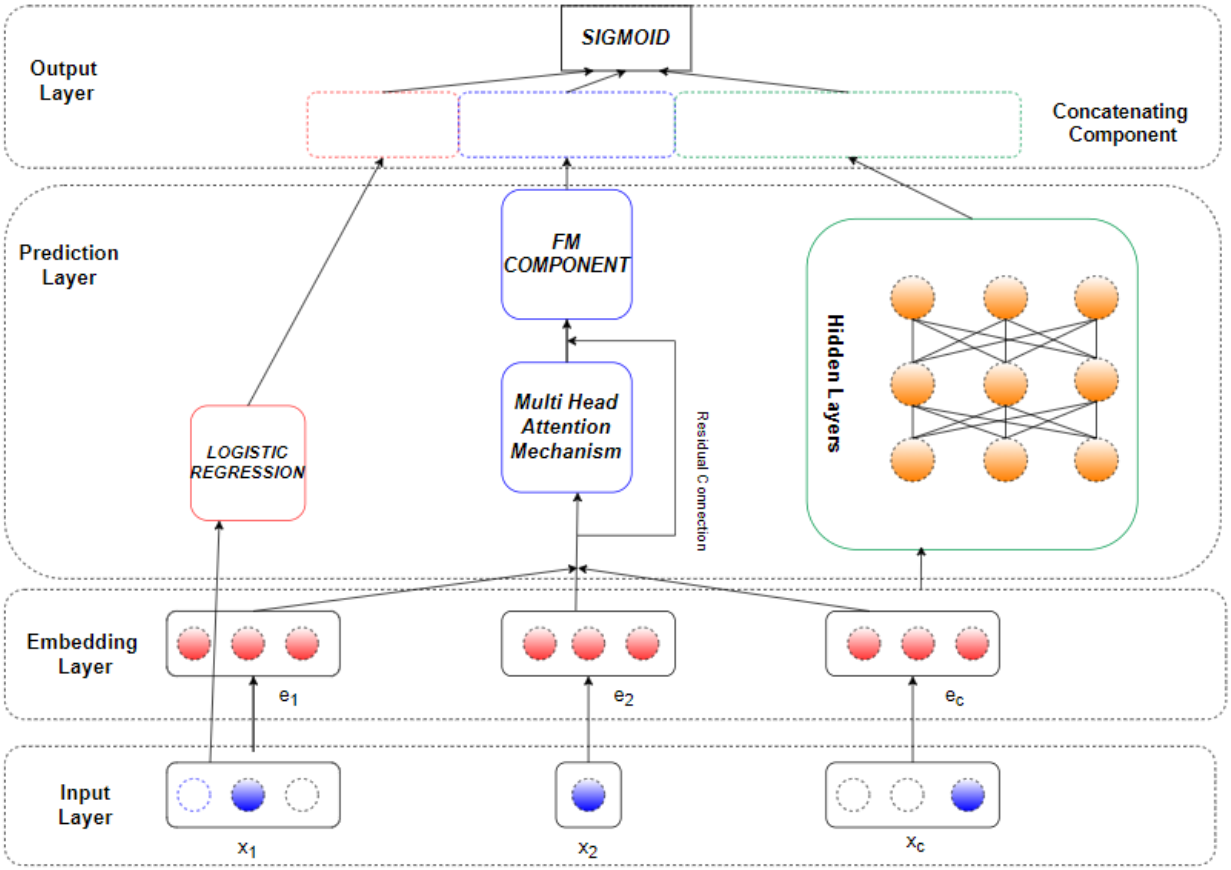


Figure 6 - The architecture of our proposed model Multi Attentional Neural Factorization Machines

CHAPTER 3

MODEL BENCHMARKING

3.1 Loss Function

As loss function of this model, Logloss function [33] decided to be used. How closely the forecast likelihood matches the associated real or actual value is shown by log-loss (under the binary classification condition it is 0 or 1). The greater the log-loss number, the greater the projected likelihood deviates off the true value.

$$loss = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1 - y_i)(\log(1 - p_i))$$

Where p_i is the probability of the CTR, y_i is {0,1} labels, N total number of inputs.

3.2 Complexity Analysis

When discussing an application's or system's overall memory usage, such as the space consumed for input data during operation, the term "space complexity" is often employed. To calculate the space complexity of a model, space that elements occupy needs to be calculated.

We will go over each space complexity corresponding to the various components. Total space used for the Embedding layer part is O(nd), where n is the dimension of the input feature and d describes the dimensionality. Total space used for Logistic Regression part is O(kd) where d is the dimension for the output value. Same for Factorization Machines O(kd). For the multi-head attention part O(L × (3dd^ + d^Hd)), where L is the number of attention layers, H is the number of attention heads d^ is the dimension of the outcome. The complexity space for the DNN model is O(d × m +m + (m^2+m) × (L_d − 1)) Where m is the layer size, L_d is the number of layers in the neural network and d is the input dimension.

3.3 Evaluation and Findings From Experiments

In this section, the effectiveness of the suggested model MANFM using two open datasets was assessed and compared its effectiveness to other traditional and cutting-edge models and look at the impact of various MANFM components. The research having the following questions was carried out.

- RQ1: How well did MANFM operate in comparison to cutting-edge models when it comes to CTR prediction? Does it work well with large datasets?
- RQ2: What effects did the components and hyperparameters effect on the performance of the MANFM?
- RQ3: In neural network-based CTR models, which feature interaction technique—Dot Product vs. Elementwise-product—is more successful?
- RQ4: Could alternative models easily exchange the components in this framework?

3.4 Evaluation Metrics

To examine all the systems for the CTR prediction challenge, one of the most widely used evaluation metrics was employed in this study. The likelihood that expected positive observations will arrive before estimated negative ones is represented by the AUC (Area Under the Curve) [34]. AUC typically ranges between 0.5 and 1, with a larger AUC indicating greater model efficiency.

The mean variance of the prediction's outcomes, or logloss, places more emphasis on the system's ranking capabilities. Greater performance is typically indicated by a lower Logloss value. The equation is displayed above.

$$AUC = \frac{\sum e_p > e_n}{N_p * N_n}$$

Where e_p and e_n are the predicted positive and negative samples and N_p and N_n are the number of positive and negative samples respectively.

3.5 Performance comparison with other models (RQ1)

Regarding the CTR prediction objective in the study, we incorporated a variety of models. For the comparison, there are 11 model examples, including deep neural networks, linear models, Factorization Machines and Attention Mechanisms. Following is a short description of them.

LR [6]. It cannot convey the connections between features; it can simply replicate the linear sum of the raw features. It is the simplest model found in this experiment and the worst model in terms of performance.

FM[9]. Since it has a second order component in addition to the linear model it contains, it has a better result than LR as it can detect the relationship between two features.

FwFM [8]. It uses a lot less memory than original FM to simulate the many feature relationships across various fields.

FNN[12]. By stacking FM on top of 2 layered deep neural network, it provided a better AUC and logloss results.

Wide and Deep[22]. There are two sections to the model (deep part is neural networks and wide part is a linear model). The deep component feeds it with sparse elements that have passed via the embedding layer, while its Wide part feeds it with explicit elements. To calculate the anticipated likelihood, the findings from the two components are combined.

DeepFM[24]. It can be said that it is the advanced version of the wide and deep model. By using FM instead of the linear model in the Wide and Deep model, more than 2 order feature interactions are provided thanks to both the first order and the second order and DNN.

NFM[39]. It has a bilinear interaction layer by stacking down of the feed forward neural network. It tried to get high order interactions via DNN.

IFM[40]. It uses FM and the modified feed forward neural network. By intentionally learning more adaptable and precise representations of features for various cases via the use of a factor estimation system, it seeks to improve conventional FMs.

AutoInt[25]. The multi-head self-attention mechanism is used instead of a Feed forward neural network to execute autonomous feature connection processing in order to create significant high-order connections and raise the precision of CTR prediction.

DCN[23]. The feed neural network is employed to acquire feature connections implicitly, while CrossNetwork ,which has cross layers, is performed to acquire both low-order and high-order connections directly. In order to achieve the anticipated likelihood, a fully connected layer receives a vector that has been linked from the outcomes of the CrossNetwork and DNN.

3.6 Implementation Details

In order to test the DAFM model, google Colab [35] notebook is used as python IDE. This makes it possible to type in and run any Python script from within a web browser, making it particularly useful for the study. It all so provides a free use of RAM and GPU for the training.

The pandas library [36] was used for analyzing the datasets to be used in this study and making the datasets ready for model training.

Documentation from DeepCTR [37] was very helpful for us to implement our model and other state of art models for comparisons. We used specific functions using DeepCTR documentation to assist in model building and hyperparameter tuning. For instance, SparseFeat, DenseFeat, get_feature_names.

For MANFM, the batch size is 256, the number of multi heads is 4, the number of attention layers is 4, the embedding size is 8. For the DNN component of our model, the number of hidden units is three (300,300,300). L2 regularization parameters is [$1e^{-5}$], dropout rates are [0,0.1,0.2…0.9]. In DCN, the number of cross layers is 2. In AutoInt model the number of heads is 2, the number of attention layers is 3 and the embedding size is 8.

For all the models that have been used in the experiment, batch size was 256, test train validation split was 0.2, number of epocs was 10, Adam optimizer used to optimize all the models.

3.7 Model Performances

Table illustrates the results of several models on two datasets, that allows us to make the relevant findings.

- Logistic Regression was the model that provided only the first order feature interaction and gave the lowest performance because it could not achieve a higher feature interaction.
- CCPM model which used conventional neural network for the ctr prediction, was the second worst performing model ,ahead of LR, in the test.

- DeepFM, NFM, Wide & Deep and DCN all used feed forward neural network as part of their model and the AUC and logloss scores showed that these models perform higher and obtain better outcomes.
- FN,FNN,IFM and all used FM as part of their model, They tried to achieve a better score by aiming to obtain second-order feature interactions, but still lagged compared other models in terms of performance. However, FwFM model emerged from among the models which used FM was able to achieve a much better score than them. FwFM could efficiently obtain the variability of field couple connections by adding and learning a field pair weight matrix and operates with less parameters than FNN.
- AutoInt model only used multi-head attention mechanism and performed better than most of these models in the experiment. That shows that the superiority of Attention mechanism for ctr prediction.
- It was also concluded that the models installed in parallel gave better performance than the models stacked on top of each other.
- As a result, since it can capture first order, second order and high order feature connections, the MANFM model outperformed all the models used in the experiment.

Table 1 – Performance comparisons of state of art CTR prediction models

	Criteo Dataset		**Avazu Dataset**	
Model	**AUC**	**Logloss**	**AUC**	**Logloss**
LR	0.7658	0.4748	0.7414	0.3954
FM	0.7698	0.4718	0.7532	0.3894
IFM	0.7778	0.4556	0.7664	0.4025
NFM	0.7891	0.4543	0.7646	0.3907
AutoInt	0.7921	0.4531	0.7743	0.3989
DeepFM	0.7926	0.4529	0.7726	0.3999
Wide & Deep	0.7928	0.4527	0.7733	0.3790
DCN	0.7929	0.4529	0.7733	0.3795
FNN	0.7934	0.4530	0.7732	0.3790
FwFM	0.7935	0.4526	0.7735	0.3793
MANFM	**0.7940**	**0.4523**	**0.7757**	**0.3776**

3.8 Impact of the MLP Hyperparameters (RQ2)

The feed forward neural network model, which is used for feature interaction with 3 orders and higher, has a very important place in the general model. The effectiveness of our feed forward neural network design is greatly influenced by the selection of suitable hyperparameters since it significantly affects the learnt paradigm. For this, parameters with different values were tried by using trial and error method and their effect on model performance was observed. Finally, the neural network was established by selecting the appropriate parameters.

3.8.1 Impact of Number of Hidden Layers

In order to observe how much the number of hidden layers used in the neural network affects the model performance, from the number of 1 to 4 hidden layers were used. According to the results obtained, a great decrease was observed in the performance of the model when only one hidden layer was used. The same situation was experienced when using two hidden layers. However, in the case of three hidden layers, there was an increase of 0.2 in the model AUC score, it would not be wrong to mention that the model performance remained constant after passing from three to four. In this case, using three hidden layers was the right choice.

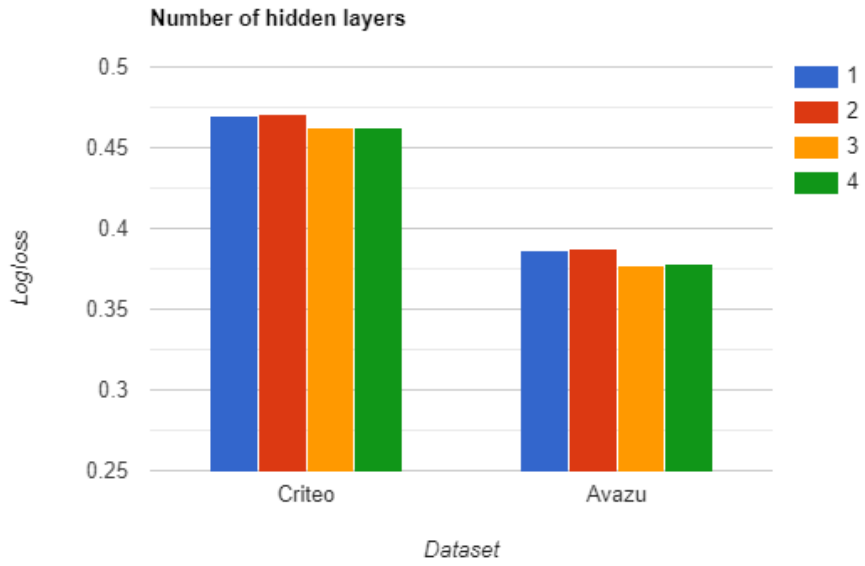


Figure 7 – Logloss Performance comparison of number of hidden layers using

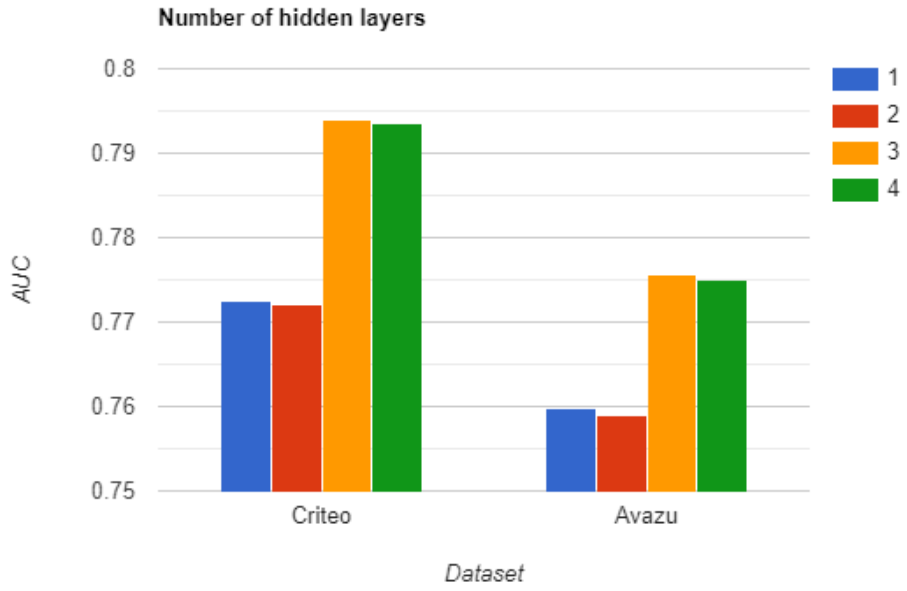


Figure 8 - AUC Performance comparison of number of hidden layers using

3.8.2 Impact of the Type of the Product Used in FM (RQ3)

A model inspired by the FM model was established to enable the interaction of the second order feature. The difference between our model and FM was that the product operator between the two vectors was different. While FM uses inner product, our model used element wise product operator. In order to determine the effect of these product operators on the model performance, the original model and the model using inner product were compared with each other. As a result, our original model using the element wise operator became the preferred product operator because of better performance and much shorter training time.

Table 2 – Performance comparison of type of product used in FM

	Criteo Dataset		**Avazu Dataset**	
Type of Product	**AUC**	**Logloss**	**AUC**	**Logloss**
Inner	0.7939	0.4524	0.7753	0.3785
Elementwise	0.7940	0.4523	0.7757	0.3776

3.8.3 Deep Neural Network Activation Function

An essential component of a neural network's architecture is its activation functions. How successfully the models can learn the data will depend on the activation function that is used for the hidden units. The kind of forecasts the model would produce is largely dependent around the activation function that is used for the output of the hidden layer. Trials were made between tanh and ReLU functions ,since sigmoid function used for output, for the activation function used in our model. After the test, ReLU was chosen because the performance of the ReLU function was superior to the tanh activation function.

Table 3 - Performance comparison of type of Activation function used in DNN

	Criteo Dataset		**Avazu Dataset**	
Activation Function	**AUC**	**Logloss**	**AUC**	**Logloss**
Tanh	0.7861	0.4603	0.7666	0.3832
ReLU	0.7940	0.4523	0.7747	0.3783

3.8.4 Impact of Using Dropout

Overfitting is a significant issue with these networks. It is challenging to prevent overfitting by merging the predictions of several big learning algorithms at testing time since large models are typically slow to utilize. Dropout is a method for solving this issue. The basic concept is to arbitrarily remove components and links from the neural net while it is being trained. Components are prevented from over co-adapting as a result. For our model, the dropout was tested from 0.6 to 0.8 and their performance was observed. While it gave the best performance at 0.6, it was observed that the performance decreased as the rate increased. As a result, it was decided not to use dropout because the obtained AUC and logloss values gave much better results when the rating was at 0.

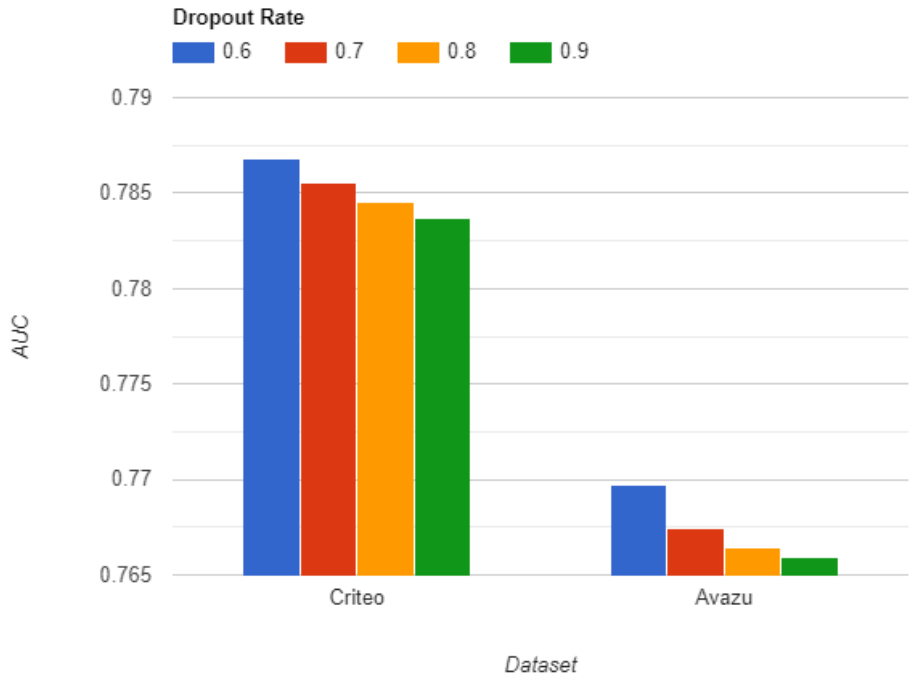


Figure 9 - AUC Performance comparison of different Dropout rates

3.8.5 Impact of Using Batch Normalization

The signal of the previous hidden layer serves as the center layer's input during learning process. As a result, changes to the hidden layer's parameters would significantly alter how its result is distributed. According to predictive analytics theory, a hidden layer's parameters must be learned again when the input distribution varies. We refer to this phenomenon as Internal Covariate Shift. Making sure that each hidden layer's input distribution remains constant throughout learning is required to solve this issue. The quickest and most straightforward way to maintain the distribution of each hidden layer would be to

normalize it. Batch Normalization technique is commonly used by researchers for normalization. We applied batch normalization to the neural network after the activation function ReLU. The tests showed that when Batch Normalization was used, it did make a very good positive impact on the model performance, therefore we decided to keep batch normalization.

Table 4 - – Performance comparison of Batch Normalization

	Criteo Dataset		Avazu Dataset	
Batch Normalization	**AUC**	**Logloss**	**AUC**	**Logloss**
True	0.7943	0.4522	0.7757	0.3776
False	0.7940	0.4523	0.7744	0.3786

3.8.6 Impact of Number of Neurons on each layer

A test experiment was conducted to find out how much the number of neurons in the hidden layers of the neural network would affect the model performance. All the parameters in the model were kept the same, and only the total number of neurons in the layers was changed. First, 100 neurons were given to each layer, then 200, 300, 400 and 500 were continued. There was also an increase in the AUC score for each layer. However, when it came to 500 neurons per layer, a decrease was observed in model success compared to 400. In this case, using 300 neurons in each layer was the right choice.

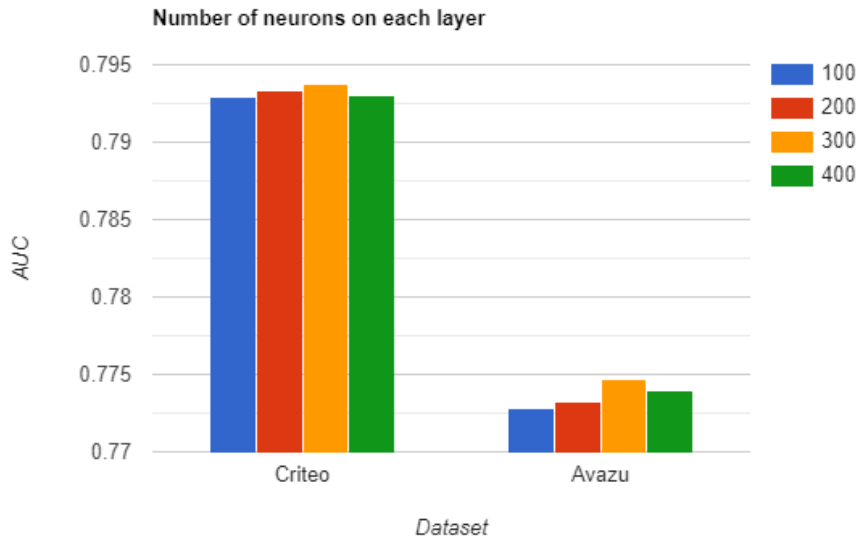


Figure 10 - AUC Performance comparison of number of neurons on each layer

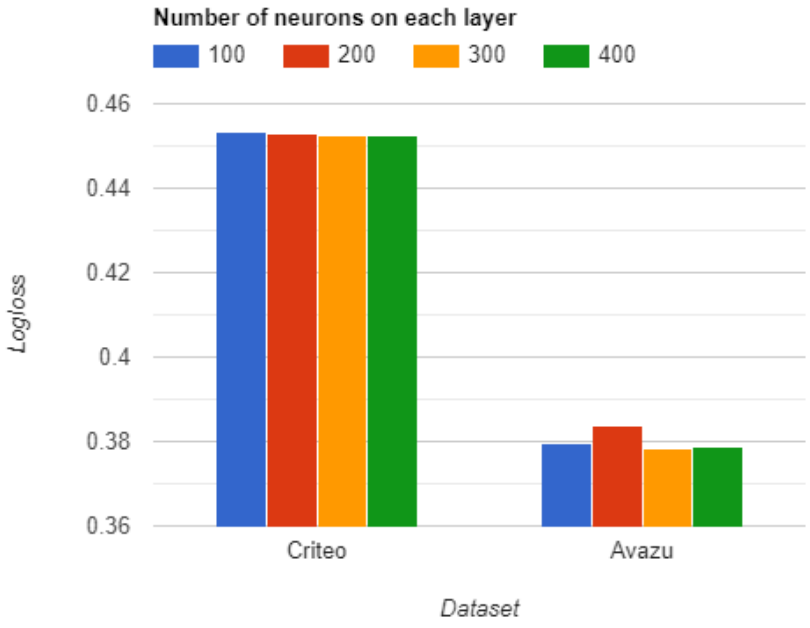


Figure 11 - Logloss Performance comparison of number of neurons on each layer

3.8.7 Neural Network Structure

Normally, the total number of neurons used in the model is 600. Without changing this number, the effect of the layer structure on the model will be examined by using a different number on each layer. 4 different shapes have been tried, these are increasing (100,300,500), decreasing (500, 300, 100) , constant (300, 300, 300) and diamond (200 500 200) order. The result of the tests showed that, Considering the arrangement of neurons in layers, the constant value order puts forward a much better performance than the others. Therefore, using three layers was the right choice.

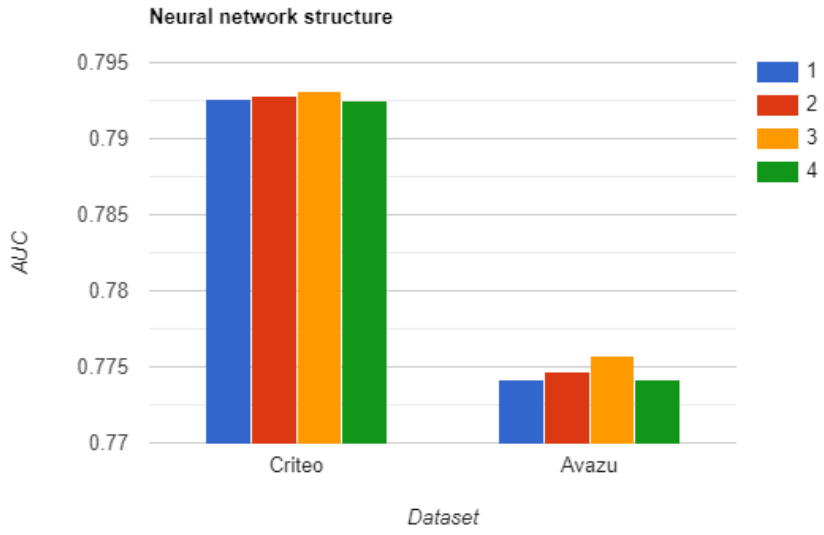


Figure 12 – AUC Performance comparison of neural network structure

3.9 Impact of Multi Attention Network Parameters

3.9.1 Impact of Residual Connection

According to related study, explicitly integrating the attention mechanism would lead to an efficiency reduction, however by integrating residual connections, the attention mechanism can function at a higher capacity. Second -order feature acquisition in MANFM is able to generate superior outcomes because to the residual connections. In order to demonstrate this, we compared the initial version of MAMFM against the version missing residual connections (named MANFM w/o), keeping all other components and settings same. The performance assessment of these two approaches is shown in Table. It is clear that MANFM outperforms MANFM w/o across both datasets. This demonstrates how crucial the residual connections are to the system's feature interactions.

Table 5 - Performance comparison of residual connection used in multi-head attention mechanism

	Criteo Dataset		**Avazu Dataset**	
Model	**AUC**	**Logloss**	**AUC**	**Logloss**
MANFM	0.7940	0.4523	0.7757	0.3776
MANFM w/o	0.7629	0.4529	0.7747	0.3782

3.9.2 Impact of Attention Heads

The ability to launch features across numerous vector spaces and hence capture various relationships in various vector spaces gives the multi-head attention mechanism an edge against the conventional attention mechanism. To contrast the single-head network with the multi-head attention network, we devised a test. The observations listed below can be seen, and inferences may be drawn accordingly.

- The multi-head attention mechanism has outperformed the single-head attention mechanism across both datasets
- Although the 3-head attention mechanism operates greater over the 2-head attention mechanism, efficiency appears to have a same result as the head number hits 3.
- It is not necessary to keep raising the number of heads until it reaches 4 or more, as doing so raises the computing cost of the system.

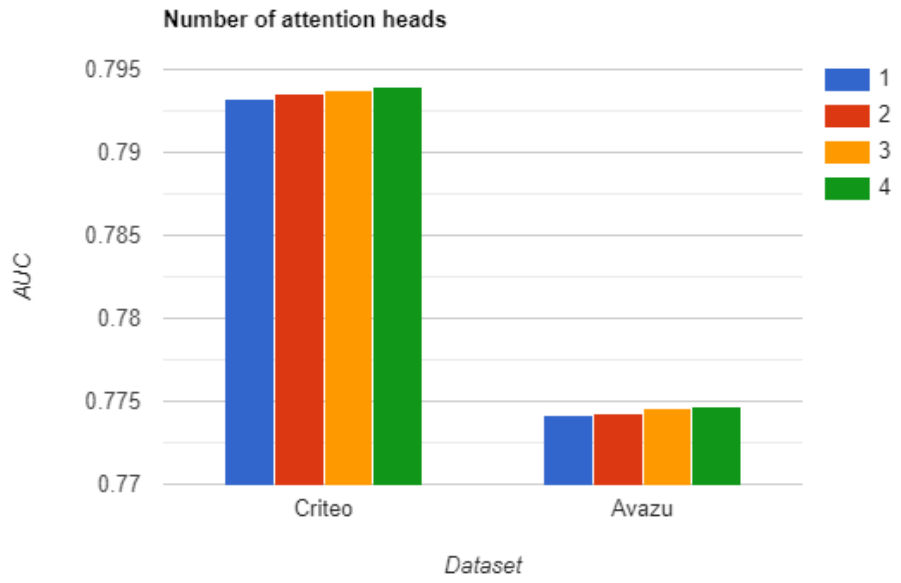


Figure 13 - AUC Performance comparison of number of attention heads in multi-head attention mechanism

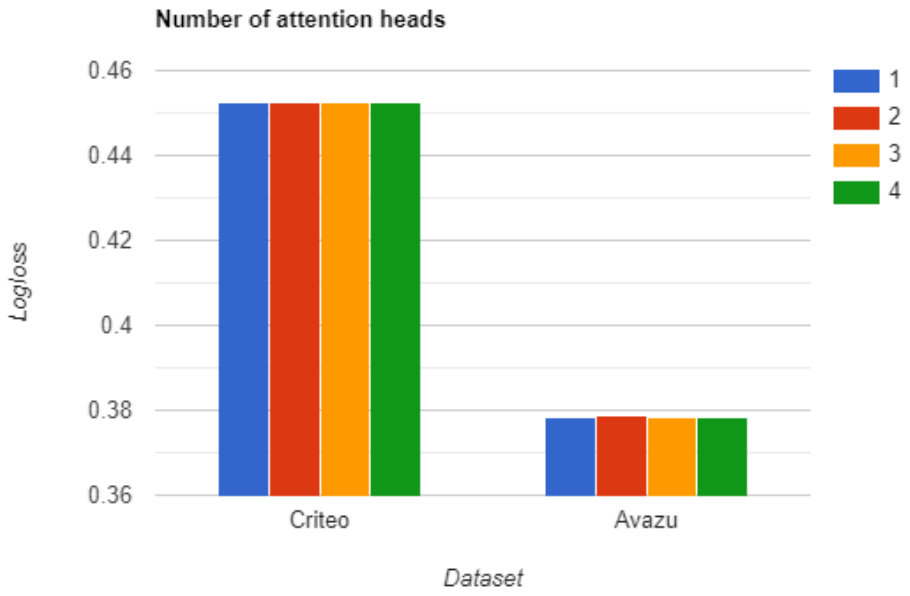


Figure 14- Logloss Performance comparison of number of attention heads in multi-head attention mechanism

3.9.3 Impact of Attention Layers

The effectiveness of the model is also impacted by modifications in network size. Several layers could be stacked to acquire high-order dynamic features. We are really interested to see whether the efficiency of the model will alter when the amount of network layers rises. A multi-head attention system is initially one layer deep, and its depth is progressively increased. After initially setting it to 1, it was increased sequentially. Detailed results are given below.

- The model's effectiveness suffers as the mechanism is expanded from one to two layers.
- The efficiency of the model improves as the mechanism is expanded from two to three layers.
- The performance of the model increases once again when the mechanism is updated from three to four layers.
- Finally, the model performance was stable when the number of layers increased from four to five.
- In summary, it was decided that the model performed highest when the number of layers was selected as 4, since having 5 layers is not contribute much to the model performance and increases the complexity.

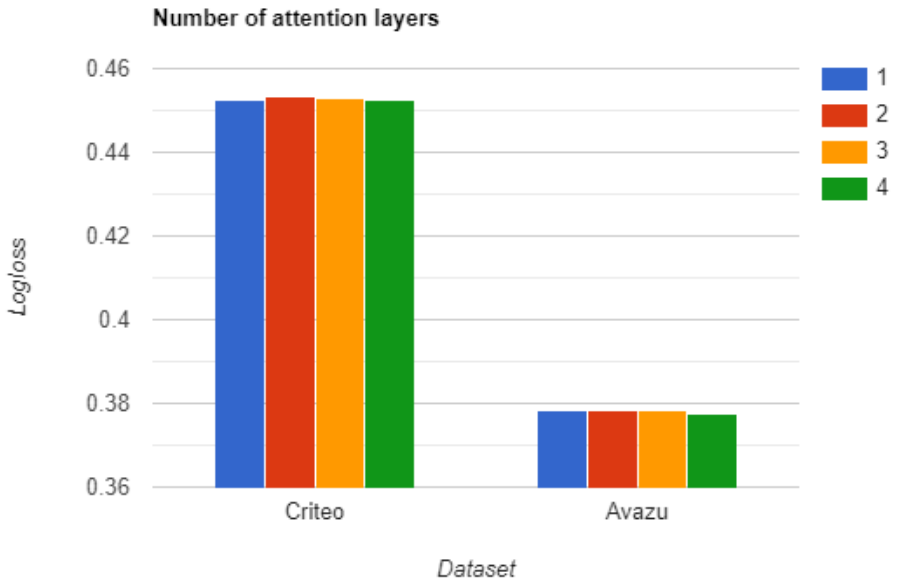


Figure 15 – Logloss Performance comparison of number of attention layers in multi-head attention mechanism

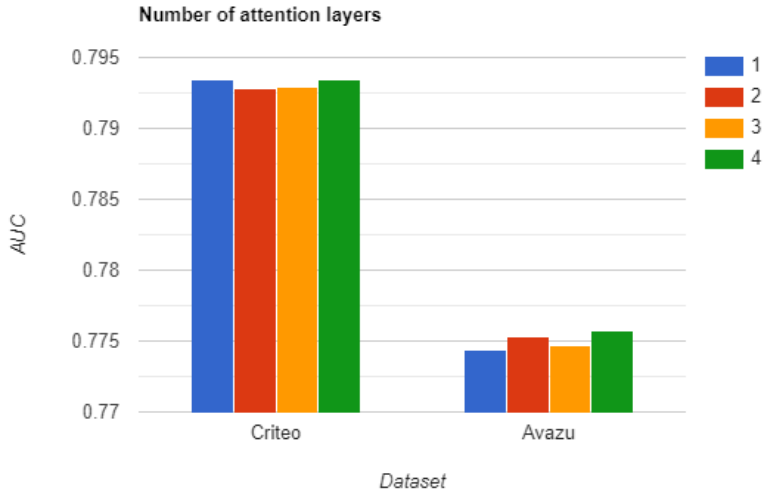


Figure 16 – AUC Performance comparison of number of attention layers in multi-head attention mechanism

3.9.4 Impact of Embedding dimensions

We're keeping an eye on how the system is affected by the embedding layer's outcome dimension. Greater size will often indicate more useful information. To retrieve additional information, we attempt to raise the dimension of the embedding layer. The test findings are depicted in Figure and the related findings may be drawn from them:

- Performance of MANFM decreases as layer dimension rises.
- When the dimension becomes 32, the model retains its performance as the dimensions are increased in both datasets.
- Ultimately, 8 is the most suitable choice because adding more dimensions would surely increase computing cost without giving any performance gain.

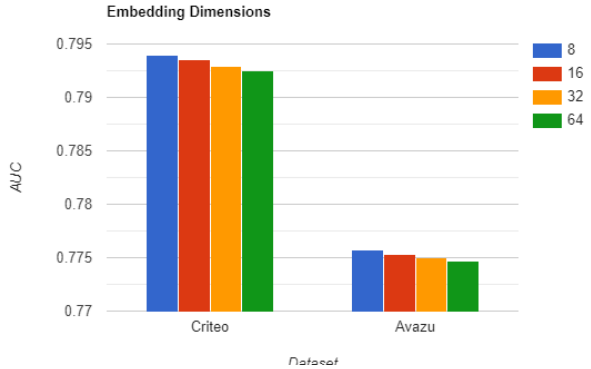


Figure 17 – AUC Performance comparison of depth of multi-head attention mechanism

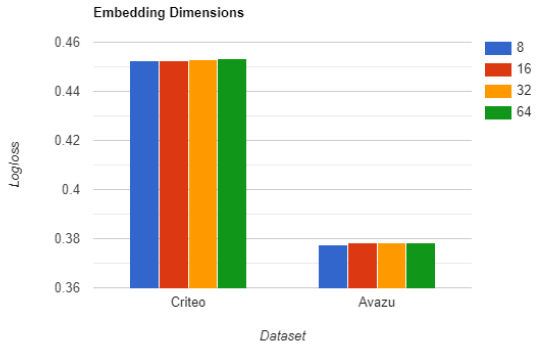


Figure 18 – Logloss Performance comparison of depth of multi-head attention mechanism

3. 10 Impact of the Model Structure (RQ4)

DCN, WDL, DeepFM, and other CTR models have all shown a preference for the parallel design. This architecture is simple and adaptable, and the many model components do not conflict with one another, this helps to some part prevent overfitting. first-order, second-order, and high-order feature connection components are present in the MANFM model. We can draw lessons from the already used, well-established models and utilize their fundamental components to swap out the relevant MANFM components.

We created the following test to investigate the cumulative impact of several components and the adaptability of MANFM. Multi-head self-attention network, Deep neural network and cross networks are options in the high-order connections components. The practical findings are shown in Table, from which we may deduce the subsequent finding.

As a result, since the model design is parallel, all kinds of components can be adapted to the model without any problems. Also, as can be seen from the AUC and Logloss scores, the DNN performance was much more successful than the other two components. This situation shows why DNN is preferred by many researchers in the industry.

Table 6 – Model structure experiment of high-order feature interactions performed

Feature Interaction Order			**Avazu Dataset**	
First Order	**Second Order**	**Higher order**	**AUC**	**Logloss**
Logistic Regression	**Multi head attention + Element wise FM**	**Deep Neural Network**	**0.7757**	**0.3776**
Logistic Regression	**Multi head attention + Element wise FM**	**Cross Network**	**0.7651**	**0.3840**
Logistic Regression	**Multi head attentio + Element wise FM**	**Multi-head Attention Mechanism**	**0.7654**	**0.3838**

CHAPTER 4

Conclusion and Future Work

4.1 Study Observations

- First of all, it was discovered that the datasets used in this study have many features. One of them is that both contain a lot of categorical features. This situation pushes us to use one hot encoding or label encoding.
- According to the results obtained after the experiments in this study for CTR estimation, the model that gave the lowest performance was the Logistic Regression model. The reasons for this are due to the fact that the Logistic model provides only the first order feature interaction. On the other hand, since the LR has a low complexity compared to the other models, the train time is very low compared to the others. This explains why the Logistic Regression model is used by many users.
- Another situation observed in CTR prediction studies is the widespread use of the FM model. The performance of the Factorization Machines model and the models using FM in its content is obviously higher than LR, because it can capture both first order and second order feature interactions. Also, its complexities are the same as LR.
- Deep Models, on the other hand, is one of the most referenced parts in model setup like FM. The reason is that it can catch order interactions higher than 2, within 1 and 2 orders, at the same time. Therefore, models using DNN in their content have been more successful.
- The Attention Mechanism can capture multiple order feature interactions such as DNN. That's why it was used in the AutoInt model. Since the performance of the AutoInt model is much better than the majority, we preferred to use the attention mechanism in our model. But he decided to combine it with FM by not using it like everyone else. This is due to the disadvantage that FM has.
- Another important issue is the model structure. Model structures are divided into two as parallel and serial, according to the results obtained, models installed in parallel achieve better performance. That's why we considered this analysis while building our model.

4.2 Conclusions

The quick expansion of social networks as a channel for digital advertising has created it essential for researchers to concentrate on reducing the rate of wrong categorization in the estimation of an ad's CTR This will assist the marketer in selecting the proper collection of characteristics to address the possible customers for their company. The precision may also contribute for determining the value of the advertisement for the marketing companies.

In this study, we present a distinctive CTR prediction framework called MANFM, which integrates four prediction components to learn various categories of features: first-, second-, and high-order features. Those three different sorts of features have separated learning procedures and a highly defined layout. A feed forward neural network is applied to extract high-order feature combinations. MANFM is successful in CTR prediction problems and achieves greater predictive performance over the state-of-the-art approaches, according to studies on three publicly available datasets. In order to create MANFM with high adaptability, durability, and compatibility with different models.

4.3 Future Work

As a further phase, the performance of our model can be increased, making it more applicable. The noise of the data may completely be eliminated if all the anonymized variables are available. By carefully adjusting the settings, we may assist address the more difficult issue of selecting the appropriate advertising for a user at the appropriate moment.

RNN or LSTM can be used instead of feed forward neural network which was used to detect high order feature connection in MANF. They model the impact previous users' sequential behaviors on current behaviors and can outperform DNN in terms of prediction accuracy.

Due to the high dimensionality of the datasets used in this study, a certain part of it could be used due to limited resources. The entire dataset can be used to unlock the maximum performance potential of this model.

Only normalization, label encoding, and one-hot encoding were done during the model preprocessing stage to make the model end to end. A better processed dataset can be achieved by manual feature engineering.

References

1- Statista. 2022. *Digital Advertising - Worldwide | Statista Market Forecast*. [online] Available at: <https://www.statista.com/outlook/dmo/digital-advertising/worldwide> [Accessed 18 July 2022].

2- Insider Intelligence. 2022. *Top 10 Countries, Ranked by Digital Ad Spending, 2022 (billions)*. [online] Available at: <https://www.insiderintelligence.com/chart/c/T11900/top-10-countries-ranked-by-digital-ad-spending-2022-billions-1> [Accessed 18 July 2022].

3- Yan, C., Chen, Y., Wan, Y. *et al.* Modeling low- and high-order feature interactions with FM and self-attention network. *Appl Intell* **51,** 3189–3201 (2021).

4- Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In Proceedings of the 26th International Joint Conference on Artificial Intelligence. AAAI Press, 1725–1731 (2017)

5- Tan, M., Zhou, J., Peng, Z., Yu, J., & Tang, F. (2020). Fine-grained image classification with factorized deep user click feature. Information Processing & Management, 57(3), 102186.

6- Kumar, Rohit & Naik, Sneha & Naik, Vani & Shiralli, Smita & V.G, Sunil & Husain, Moulahusain. (2015). Predicting Clicks: CTR Estimation of Advertisements using Logistic Regression Classifier. 10.1109/IADCC.2015.7154880.

7- Chapelle, Olivier & Manavoglu, Eren & Rosales, Rómer. (2014). Simple and Scalable Response Prediction for Display Advertising. ACM Transactions on Intelligent Systems and Technology. 5. 1-34. 10.1145/2532128.

8- Pan, Junwei, Jian Xu, Alfonso Lobos Ruiz, Wenliang Zhao, Shengjun Pan, Yu Sun and Quan Lu. "Field-weighted Factorization Machines for Click-Through Rate Prediction in Display Advertising." *Proceedings of the 2018 World Wide Web Conference* (2018): n. pag.

9- Rendle, Steffen. "Factorization Machines." *2010 IEEE International Conference on Data Mining* (2010): 995-1000.

10- Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware Factorization Machines for CTR Prediction. In Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16). Association for Computing Machinery, New York, NY, USA, 43–50. https://doi.org/10.1145/2959100.2959134

11- Sun, Yang, Junwei Pan, Alex Zhang and Aaron Flores. "FM2: Field-matrixed Factorization Machines for Recommender Systems." Proceedings of the Web Conference 2021 (2021): n. pag.

12- Zhang, W., Du, T., & Wang, J. (2016). Deep Learning over Multi-field Categorical Data - - A Case Study on User Response Prediction. ECIR.

13- Qu, Yanru, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen and Jun Wang. "Product-Based Neural Networks for User Response Prediction." *2016 IEEE 16th International Conference on Data Mining (ICDM)* (2016): 1149-1154.

14- Cheng, Heng-Tze, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishikesh B. Aradhye, Glen Anderson, Gregory S. Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu and Hemal Shah. "Wide & Deep Learning for Recommender Systems." *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems* (2016): n. pag.

15- Guo, Huifeng, Ruiming Tang, Yunming Ye, Zhenguo Li and Xiuqiang He. "DeepFM: A Factorization-Machine based Neural Network for CTR Prediction." *IJCAI* (2017).

16- Vaswani A, Shazeer N, Parmar N et al (2017) Attention is all you need. In: Proceedings of the conference on advances in neural information processing systems

17- Xiao, Jun, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu and Tat-Seng Chua. "Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks." *ArXiv* abs/1708.04617 (2017): n. pag.

18- Wang, Q., Liu, F., Zhao, X. *et al.* Session interest model for CTR prediction based on self-attention mechanism. *Sci Rep* **12,** 252 (2022). https://doi.org/10.1038/s41598-021-03871-y

19- Nilashi, Mehrbakhsh & Bagherifard, Karamollah & Ibrahim, Assoc Prof. Dr. Othman & Alizadeh, Hamid & Lasisi, Ayodele & Roozegar, Nazanin. (2013). Collaborative Filtering Recommender Systems. Research Journal of Applied Sciences, Engineering and Technology. 5. 4168-4182. 10.19026/rjaset.5.4644.

20- Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. 2010. Training and Testing Low-degree Polynomial Data Mappings via Linear SVM. J. Mach. Learn. Res. 11 (3/1/2010), 1471–1490.

21- Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware Factorization Machines for CTR Prediction. In Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16). Association for Computing Machinery, New York, NY, USA, 43–50. https://doi.org/10.1145/2959100.2959134

22- Cheng, H., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H.B., Anderson, G., Corrado, G.S., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., & Shah, H. (2016). Wide & Deep Learning for Recommender Systems. Proceedings of the 1st Workshop on Deep Learning for Recommender Systems.

23- Wang, R., Fu, B., Fu, G., & Wang, M. (2017). Deep & Cross Network for Ad Click Predictions. Proceedings of the ADKDD'17.

24- Guo, H., Tang, R., Ye, Y., Li, Z., & He, X. (2017). DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. IJCAI.

25- Song, W., Shi, C., Xiao, Z., Duan, Z., Xu, Y., Zhang, M., & Tang, J. (2019). AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. Proceedings of the 28th ACM International Conference on Information and Knowledge Management.

26- kaggle.com. (n.d.). Display Advertising Challenge. [online] Available at: https://www.kaggle.com/c/criteo-display-ad-challenge [Accessed 31 Aug. 2022].

27- kaggle.com. (n.d.). Click-Through Rate Prediction. [online] Available at: https://www.kaggle.com/c/avazu-ctr-prediction [Accessed 31 Aug. 2022].

28- Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. CoRR, abs/1409.0473.

29- Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. ArXiv, abs/1706.03762.

30- Cui, H., Iida, S., Hung, P., Utsuro, T., & Nagata, M. (2019). Mixed Multi-Head Self-Attention for Neural Machine Translation. EMNLP.

31- Lee, S. (2022). Graph-based Knowledge Distillation by Multi-head Attention Network.

32- www.ibm.com. (2021). Concatenation Operator. [online] Available at: https://www.ibm.com/docs/en/informix-servers/14.10?topic=expression-concatenation-operator.

33- Good, I. J. "Rational Decisions." Journal of the Royal Statistical Society. Series B (Methodological), vol. 14, no. 1, 1952, pp. 107–114. JSTOR, www.jstor.org/stable/2984087

34- Bradley, A.P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognit., 30, 1145-1159.

35- Google.com. (2019). Google Colaboratory. [online] Available at: https://colab.research.google.com/.

36- Pandas (2018). Python Data Analysis Library — pandas: Python Data Analysis Library. [online] Pydata.org. Available at: https://pandas.pydata.org/.

37- deepctr-doc.readthedocs.io. (n.d.). Welcome to DeepCTR's documentation! — DeepCTR 0.9.1 documentation. [online] Available at: https://deepctr-doc.readthedocs.io/en/latest/ [Accessed 31 Aug. 2022].

38- TensorFlow (2019). TensorFlow. [online] TensorFlow. Available at: https://www.tensorflow.org/.

39- He, X., & Chua, T. (2017). Neural Factorization Machines for Sparse Predictive Analytics. *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

40- *Yantao Yu, Zhen Wang, and Bo Yuan. 2019. An input-aware factorization machine for sparse prediction. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19). AAAI Press, 1466–1472.*

APPENDIX

DATA PRE PROCESSING

```
from deepctr.feature_column import SparseFeat, DenseFeat, get_feature_names

for feature in sparse_features:
    label_encoding = LabelEncoder()
    data[feature] = label_encoding.fit_transform(data[feature])
mms = MinMaxScaler(feature_range=(0, 1))
data[dense_features] = mms.fit_transform(data[dense_features])

all_feature_columns = [SparseFeat(feat, vocabulary_size=data[feature].max() + 1, embedding_dim=4) for i, feature in
enumerate(sparse_features)] \
                      + [DenseFeat(feat, 1, ) for feature in dense_features]

non_linear_columns = all_feature_columns
linear_columns = all_feature_columns

name_of_features =  get_feature_names(linear_columns + non_linear_columns)

# all_feature_columns, len(all_feature_columns)


train_data, test_data = train_test_split(data, test_size=0.2)
train_input = [train_data[name] for name in name_of_features]
test_input = [test_data[name] for name in name_of_features]
```

MAIN MODEL

```
def MANFM(linear_columns, non_linear_columns, number_attention_layer=4, attention_embedding_size=
8, attention_head_number=4,
             residual_con=True,
             dnn_hidden_units=(300, 300, 300), dnn_activation='relu', l2_reg_linear=1e-5,
             l2_reg_embedding=1e-5, l2_reg_dnn=0, dnn_use_bn=True, dnn_dropout=0, seed=1024,
             task='binary', ):


    #Returns the input features
    features = build_input_features(non_linear_columns)
    #The function returns a view object that shows a list of all the dictionary's values.
    input = list(features.values())


    # Logistic Regression Model
    linear_output = get_linear_logit(features, linear_columns, seed=seed, prefix='linear', l2_reg
=l2_reg_linear)


    sparse_embedding_vectors, dense_value_vectors = input_from_feature_columns(features, non_line
ar_columns,l2_reg_embedding, seed)


    # joins a collection of sparse vectors together and pass to att_input object.
    att_input = concat_func(sparse_embedding_vectors, axis=1)
```

```
    # For the number of attention layer in the model, do:
    for i in range(number_attention_layer):
        # the output of multi head attention fed to FM component.
        attention_input = MultiAttentionMechanism(attention_embedding_size, attention_head_number
, residual_con)(att_input)
    # The last dimension of the output tensor is of size units instead of the same form as the in
puts.
    att_output = ElementWiseFM()(attention_input)
    attention_output = Dense(1, use_bias=False)(att_output)

    # sparse and dense vectors combined together
    mlp_input = combined_dnn_input(sparse_embedding_vectors, dense_value_vectors)
    # Vectors feed to hidden layers
    mlp_output = DNN(dnn_hidden_units, dnn_activation, l2_reg_dnn, dnn_dropout, dnn_use_bn, seed=
seed)(mlp_input)
    # The last dimension of the output tensor is of size units instead of the same form as the in
puts.
    mlp_output = Dense(1, use_bias=False)(mlp_output)

    # The outcomes of the three components combined together.
    final_output = add_func([linear_output, mlp_output, attention_output])
    # Sigmoid function
    output = PredictionLayer('binary')(final_output)

    final_model = Model(inputs=input, outputs=output)

    return final_model
```

Multi Attention Mechanism

```
class MultiAttentionMechanism(Layer):

  def __init__(self, attention_embedding=8, number_of_heads=4, residual=True, use_scale =False, seed=1024, **kwargs):

    self.attention_embedding = attention_embedding
    self.number_of_heads = number_of_heads
    self.residual = residual
    self.seed = seed
    self.use_scale = use_scale

    super(MultiAttentionMechanism, self).__init__(**kwargs)


  def build(self, input_shape):

    embedding_vector = int(input_shape[-1])
    self.q = self.add_weight(name='query', shape=[embedding_vector, self.attention_embedding * self.number_of_heads],
                             dtype=tf.float32,
                             initializer=TruncatedNormal(seed=self.seed))
    self.k = self.add_weight(name='key', shape=[embedding_vector, self.attention_embedding * self.number_of_heads],
```

```
                                dtype=tf.float32,
                                initializer=TruncatedNormal(seed=self.seed + 1))
        self.v = self.add_weight(name='value', shape=[embedding_vector, self.attention_embedding * self.number_of_heads],
                                 dtype=tf.float32,
                                 initializer=TruncatedNormal(seed=self.seed + 2))
        if self.residual:
            self.r = self.add_weight(name='res', shape=[embedding_vector, self.attention_embedding * self.number_of_heads],
                                     dtype=tf.float32,
                                     initializer=TruncatedNormal(seed=self.seed))


        super(MultiAttentionMechanism, self).build(input_shape)


    def call(self, inputs, **kwargs):

        query = tf.tensordot(inputs, self.q,axes=(-1, 0))
        key = tf.tensordot(inputs, self.k, axes=(-1, 0))
        value = tf.tensordot(inputs, self.v, axes=(-1, 0))


        query = tf.stack(tf.split(query, self.number_of_heads, axis=2))
        key = tf.stack(tf.split(key, self.number_of_heads, axis=2))
        value = tf.stack(tf.split(value, self.number_of_heads, axis=2))

        dot_product = tf.matmul(query, key, transpose_b=True)
        if self.use_scale:
            dot_product /= self.attention_embedding ** 0.5

        self.attention_output = softmax(dot_product)

        output = tf.matmul(self.attention_output,value)
        output = tf.concat(tf.split(output, self.number_of_heads, ), axis=-1)
        output = tf.squeeze(output, axis=0)

        if self.residual:
            output += tf.tensordot(inputs, self.r, axes=(-1, 0))
        output = tf.nn.relu(output)

        return output

    def compute_output_shape(self, input_shape):

        return (None, input_shape[1], self.attention_embedding * self.number_of_heads)


    def get_config(self, ):
        config = {'attention_embedding': self.attention_embedding, 'number_of_heads': self.number_of_heads, 'residual': self.residual,
                  'seed': self.seed}
        base_config = super(MultiAttentionMechanism, self).get_config()
        base_config.update(config)
        return base_config
```

Elementwise Factorization Machines

```
class ElementWiseFM(Layer):
    def __init__(self, **kwargs):

        super(ElementWiseFM, self).__init__(**kwargs)

    def build(self, input_size):

        super(ElementWiseFM, self).build(
            input_size)

    def call(self, inputs_features, **kwargs):

        combined_value = inputs_features
        square_sum = tf.square(reduce_sum(
            combined_value, axis=1, keep_dims=True))
        sum_square = reduce_sum(
            combined_value * combined_value, axis=1, keep_dims=True)
        result = 0.5 * (square_sum - sum_square)

        return result

    def compute_output_shape(self, input_shape):
        return (None, 1, input_shape[-1])
```
