Improving Deep Learning Based Recommender Systems Using Dimensionality Reduction Methodologies

Authors

Mohammad Anas Jawad (154443)

Mohammed Saidul Islam (154438)

Supervisor

Dr. Abu Raihan Mostofa Kamal

Professor

Department of Computer Science and Engineering (CSE)

Islamic University of Technology (IUT)

A thesis submitted to the Department of CSE

in partial fulfillment of the requirements for the degree of

Bachelor of Science in CSE

Department of Computer Science and Engineering (CSE)

Islamic University of Technology (IUT)

Organization of the Islamic Cooperation (OIC)

Gazipur, Bangladesh

November 2019

Declaration of Authorship

This is to certify that the work presented in this thesis is the outcome of the analysis and experiments carried out by Mohammad Anas Jawad and Mohammed Saidul Islam under the supervision of Dr. Abu Raihan Mostofa Kamal, Professor, Department of Computer Science and Engineering, Islamic University of Technology (IUT), Dhaka, Bangladesh. It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Authors:

- -

Mohammad Anas Jawad
Student ID: 154443

- -

Mohammed Saidul Islam
Student ID: 154438

Supervisor:

- -

Dr. Abu Raihan Mostofa Kamal
Professor,
Department of Computer Science and Engineering
Islamic University of Technology (IUT)

Abstract

Understanding sophisticated user behavior-based feature interaction is crucial to optimizing CTR for recommender systems. Despite having significant progress, the methods that are used nowadays tend to have a strong bias towards low- or higher-order feature interactions and involve a great deal of feature engineering. However, most of the feature engineering methods are nontrivial and often requires rigorous feature engineering or exhaustive searching. DNNs can learn feature interactions automatically; but they implicitly generate all of those interactions and there is no control over the DNN about how it is generating all the cross features thus resulting in many redundant crosses. The proposed model, incorporates the strength of factorization machines for recommendation and applies PCA for learning largest data variance to feed the important features to the deep model. Compared to the Google's new Wide Deep design, we used the regular input for the factorization machines, but to get rid of the redundant cross features we feed the deep model with major features that impacts the prediction most. The results are very promising and they show a significant increase in accuracy on the predictions for CTR compared to state of the art Wide & Deep and DeepFM model.

Acknowledgement

It is an auspicious moment for us to submit our thesis work by which are eventually going to end our Bachelor of Science study. At the very beginning, we want to express our heartfelt gratitude to Almighty Allah for his blessings bestowed upon us which made it possible to complete this thesis research successfully. Without the mercy of Allah, we would not be where we are right now.

We would like to express our grateful appreciation to Dr. Abu Raihan Mostofa Kamal, Professor, Department of Computer Science and Engineering, Islamic University of Technology for being our adviser and mentor. His motivation, suggestions and insights for this thesis have been invaluable. Without his support and proper guidance, this thesis would not see the path of proper itinerary of the research world. His valuable opinion, time and input provided throughout the thesis work, from the first phase of thesis topics introduction, research area selection, proposition of algorithm, modification and implementation helped us to do our thesis work in proper way. We are grateful to him for his constant and energetic guidance and valuable advice.

We would also like to thank Mr. Hamjajul Ashmafee, Lecturer, Department of Computer Science and Engineering, Islamic University of Technology for his valuable advice and input.

We would like to extend our vote of thanks to all the respected jury members of our thesis committee for their insightful comments and constructive criticism of our research work. Surely they have helped us to improve this research work.

Last but not the least, we would like to express our sincere gratitude to all the faculty members of the Computer Science and Engineering department of Islamic University of Technology. They helped make our working environment a pleasant one by providing a helpful set of eyes and ears when problems arose.

Contents

List of Figures

Chapter 1

Introduction

We first provide an outline of our study in this section, which includes the problem statement in detail. The research challenges to be faced in the overall scenario are also discussed on the basis of the problem statement. The objectives, motivations and our contributions has been presented in different sections. At the end of this chapter there is a description of the organization of the thesis.

1.1 Overview

1.1.1 Recommender Systems

Recommender Systems, in a very general way, are algorithms designed to suggest relevant items to users (e.g., items being movies to watch, articles to read, products to read or anything else depending on the industry) [1]. In some sectors, recommendation systems are truly critical since they can generate huge profits when they are effective or can also substantially differentiate from competitors. They are widely recognized as playlist creators for video and music services such as Netflix, YouTube and Spotify, product recommendations for services such as Amazon, and user recommendations for social media platforms such as Facebook and Twitter.

1.1.2 Traditional Recommender Systems

The traditional recommender systems can be roughly categorized into three classes [2]: content-based methods, collaborative filtering (CF) based methods, and hybrid methods. Content-based methods [3] uses the user profile or product description for recommendation. They utilize a series of discrete, pre-tagged characteristics of an item in order to recommend additional items with similar properties. Collaborative filtering based methods [4] [5] make use of a user's past activities or preferences like, user ratings on items, user reviews etc without using any product information or user information. Moreover, CF based methods take into consideration whether any similar decisions made by other users or not. If so then these information are used to predict items (or ratings for items) that the user may have an interest in. The Hybrid approaches [6] [7] have the aim to merge content-based and CF-based methods to achieve the best of both worlds.

1.1.3 Recent Trends in Recommender Systems

Over the recent years, deep learning has generated a considerable interest in many fields of research, such as computer vision and natural language processing, not only because of huge performance boost, but also because of the appeal of learning features from scratch. Furthermore, the power of deep learning is widespread, showing its usefulness recently when applied to work on information retrieval and advising programs. Clearly, in the recommender model, the world of deep learning is thriving. Recommendation systems are important tools for enhancing user experience and supporting sales/services for many online websites and mobile applications [8] [9] [10]. For eg, 80% of films watched on Netflix came from recommendations [10], 60% of video clicks came from YouTube's home page recommendation [9]. Covington et al. [8] introduced a recommendation algorithm based on a deep neural network for YouTube video recommendation. Cheng et al. [11] proposed a Google Play App Recommender Framework with a wide and deep template. Shumpei et al. [12] provided Yahoo News with an RNN-based news recommendation system.All these models stood the online evaluation and demonstrated substantial improvement over traditional models. Therefore,

we can see that in industrial recommender applications, deep learning has contributed to a groundbreaking revolution. In recent years, there has been an unprecedented increase in the number of research publications on deep learning recommendation methods, providing strong evidence of the eventual presence of deep learning in recommendation system science.

1.2 Problem Statement

Recommender systems have changed the notion of our shopping behavior completely by regularly suggesting us products that we want to purchase with credible accuracy. But traditional methods sometimes provide irrelevant and inappropriate recommendation because of sparsity or unavailability of user data. Newer methods like matrix factorization, topic modeling etc are proposed to solve those problems, though they work best for small scale problems. When they are given a large set of data, like user data of a large marketplace, they tend to perform poorly because of scalability issues. Moreover, the traditional models fail to learn from sophisticated feature interactions behind user behaviors as well as from low- and higher-order feature interactions. This is where the deep learning approach shines, as they can scale large set of data as well as can learn from multi-feature combinations. This results in generalizing recommendations and thus giving us a more accurate and appropriate recommendation. Deep learning approaches can work with large scale datasets and state of the art methods can combine both linear and deep learning models to get the best out of the dataset. However, in state of the art models, e.g., Wide & Deep [11], DeepFM [13] has no control over the deep model where the higher-order feature combinations are created.
The primary goal of this thesis is "Improving the deep model's feature interactions by dimensionality reduction methods", so that only the major features that have higher impact in user choices are used in to create higher-order feature combinations.

1.3 Research Challenges

The developed methodologies should satisfy the following criteria:

(i) The model should be able to generalize recommendation for a new user, that is, the cold-start problem should not affect the model in predicting user choices.

(ii) Even if the dataset is sparse, the model should be able to generate relevant recommendation for a user.

(iii) To ensure simplicity and robustness of the recommender system, the proposed method should be able to automatically generate lower-order feature combinations.

(iv) Higher-order feature combinations should be apposite and closely related to a user.

1.4 Contributions

The presentation of this thesis is based around our goal of improving the performance of deep learning based recommender systems. The way this problem has been approached is briefly described below:

(i) From our findings, we realised that in most of the state of the art models that use deep learning approaches, the deep model fails to make efficient feature combination due to the presence of some unimportant features in the dataset. We applied feature transformation on the dataset such that only the important features are extracted.

(ii) We separated the input for the linear model and the deep model. This is because, in some of the state of the art models we studied, manual feature combination is done on the dataset and the new combined raw features are also sent as input to the linear model. So, to keep the process unhindered, we did not modify the dataset in case of the linear model.

(iii) We applied feature transformation techniques like PCA, kPCA etc. on the dataset and then used the modified features to train the deep model.

(iv) The justification behind applying said transformations has been presented.

(v) Instead of using feature selection methods, we opted for feature transformation methods because we are looking to extracting the maximum un-correlated information from features without losing much information.

(vi) Our initial experiments show promising results when compared to the standard state of the art models that we studied. More on these models has been described in the following sections.

1.5 Organization of the Thesis

The rest of this thesis is organized as follows:

Chapter 2 gives an overview of different approaches for the recommendation problem. This chapter also discusses about various methodologies of recommender systems.

Chapter 3 proposes a method that can capture the major features for generating higher-order feature combinations. It contains the framework, implementation of the proposed methodologies and also contains other evaluation methodologies that the results are tested against.

Chapter 4 presents result analysis and comparison with other implementations and studies.

Chapter 5 presents conclusions and discusses future work.

Chapter 2

Literature Review

The first section of this chapter gives a brief overview of what a recommender system is and the core idea behind its implementation. Next, we present a generic classification of different types of traditional recommender systems and how they stack against each other. In the subsequent sections, we explore a new field of recommender systems, that employ the power of deep learning combined with traditional methods and how they perform when compared to traditional models. We also discuss some of the limitations of these new recommender systems. Finally, we will discuss about the core implementation process of these models.

2.1 Background

During the last few decades, with the rise of Youtube, Amazon, Netflix and many other such web services, recommender systems have taken more and more place in our lives. From e-commerce (suggest to buyers articles that could interest them) to online advertisement (suggest to users the right contents, matching their preferences), recommender systems are today unavoidable in our daily online journeys.
Recommendation systems are, in a very general way, algorithms designed to recommend related elements to users (items like films to watch, text to read, goods to buy, etc.)
Joe Pine argues in his book "Mass Flexibility" that companies need to transition from the ancient phenomenon of mass production where "uniform goods, standardized markets, long life and development cycles were the norm" to the new world where "variety and flexibility

replace standardized products." In a vastly competitive world as ours, Pine claims that it is simply no longer sufficient to simply build a product. Companies must at least be able to produce numerous items that meet the various needs of numerous consumers. The e-commerce revolution has offered businesses more options for consumers. Nevertheless, by moving to this new customization stage, businesses are increasing the amount of data they have to process before they can choose which products they need. One fathomable approach to dealing with this copious amount of data is the use of recommender systems [1].
E-commerce websites use recommendation systems to offer goods to their customers. Such goods can be recommended on the basis of the best international sales on a web site, based on customer statistics, or on a predictive analysis of the customer's past buying activity [1]. This is the basic principle of any recommender system: Work on previous historical data to generate new data with the most relevance to a user, such that the user is inclined to purchase the article suggested by the recommender system.
Recommender systems are really critical in some industries as they can generate a huge amount of income when they are efficient or also be a way to stand out significantly from competitors. As a proof of the importance of recommender systems, we can mention that, a few years ago, Netflix organised a challenges (the "Netflix prize") where the goal was to produce a recommender system that performs better than its own algorithm with a prize of 1 million dollars to win [14].
As such, it is imperative that there is a constant influx of research geared towards the improvement of recommender systems.

2.2 Overview of Traditional Recommender Systems

In the first section we are going to overview the two major paradigms of recommender systems : collaborative and content based methods. The next two sections will then describe various methods of collaborative filtering, such and matrix factorization and such. The following section will be dedicated to content based methods and how they work.
Note: Most of the following description has been cited from `https://towardsdatascience.`

com/introduction-to-recommender-systems-6c66cf15ada. Interested readers are referred to the mentioned site for better understanding.

2.2.1 Collaborative Filtering

Collaborative methods for recommender systems are methods that are based solely on the past interactions recorded between users and items in order to produce new recommendations. These interactions are stored in the so-called "user-item interactions matrix" as shown in figure 2.1.

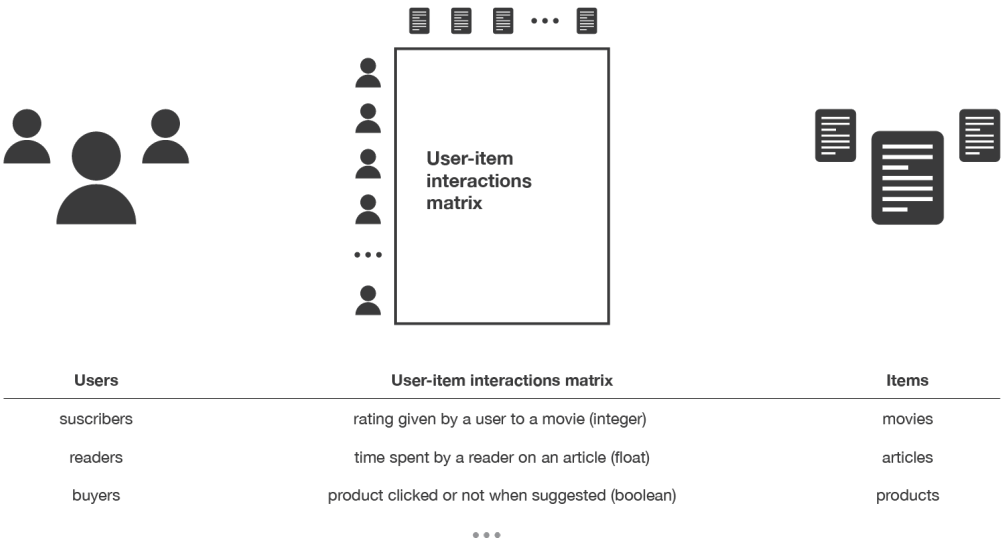


Figure 2.1: Illustration of the user-item interactions matrix

Then, the main idea that rules collaborative methods is that these past user-item interactions are sufficient to detect similar users and/or similar items and make predictions based on these estimated proximities.

The class of collaborative filtering algorithms is divided into two sub-categories that are generally called memory based and model based approaches. Memory based approaches directly work with values of recorded interactions, assuming no model, and are essentially based on nearest neighbours search (for example, find the closest users from a user of interest and suggest the most popular items among these neighbours). Model based approaches

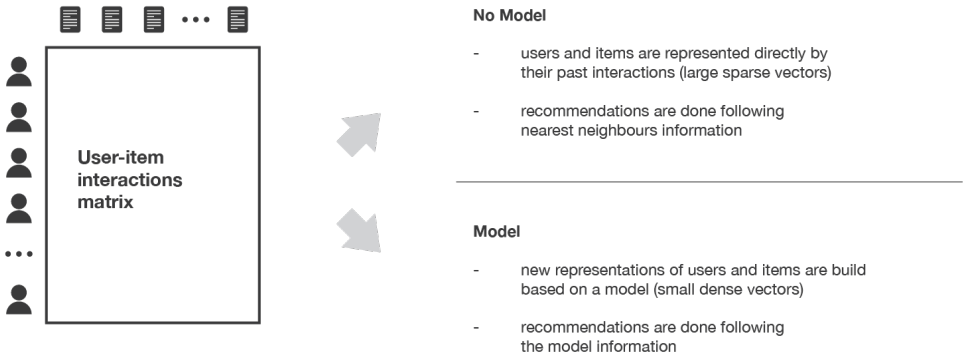


Figure 2.2: Overview of the collaborative filtering methods paradigm.

assume an underlying “generative” model that explains the user-item interactions and try to discover it in order to make new predictions. This can be seen in figure 2.2.

Collaborative filtering (CF) systems predict a user’s affinity for items or information. Unlike traditional content-based information filtering system, such as those developed using information retrieval or artificial intelligence technology, filtering decisions in CF are based on human and not machine analysis of content. Each user of an CF system rates items that they have experienced, in order to establish a profile of interests. The CF system then matches together that user with people of similar interests or tastes. Then ratings from those similar people are used to generate recommendations for the user [15]. The more users interact with items the more new recommendations become accurate: for a fixed set of users and items, new interactions recorded over time bring new information and make the system more and more effective.

However, one of the glaring faults that plagues CF systems is the infamous "Cold Start Problem", where recommendations are required for items that no one (in the data set) has yet rated. Pure collaborative filtering methods base their recommendations on community preferences (e.g., user ratings and purchase histories), ignoring user and item attributes (e.g., demographics and product descriptions). They cannot help in a cold start setting, since no user preference information is available to form any basis for recommendations. This drawback can be addressed in different way: recommending random items to new users or

new items to random users (random strategy), recommending popular items to new users or new items to most active users (maximum expectation strategy), recommending a set of various items to new users or a new item to a set of various users (exploratory strategy) or, finally, using a non collaborative method for the early life of the user or the item. Some methods include combining content and collaborative information by using expectation maximization (EM) learning to fit the model to the data [16].

2.2.2 Content-based Filtering

Unlike collaborative methods that only rely on the user-item interactions, content based approaches use additional information about users and/or items. If we consider the example of a movies recommender system, this additional information can be, for example, the age, the sex, the job or any other personal information for users as well as the category, the main actors, the duration or other characteristics for the movies (items).
Then, the idea of content based methods is to try to build a model, based on the available "features", that explain the observed user-item interactions. Still considering users and movies, we will try, for example, to model the fact that young women tend to rate better some movies, that young men tend to rate better some other movies and so on. If we manage to get such model, then, making new predictions for a user is pretty easy: we just need to look at the profile (age, sex, ...) of this user and, based on this information, to determine relevant movies to suggest.

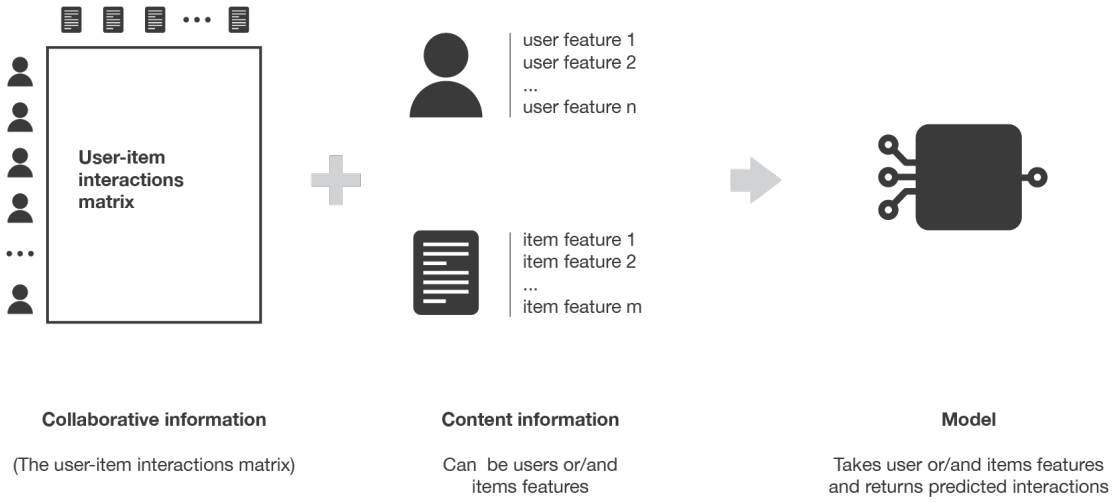


Figure 2.3: Overview of the content based methods paradigm

Formally, an item in a content based system is described as a vector $X = (x_1, x_2, ..., x_n)$ of n components. The components can have binary, nominal or numerical attributes and are derived from either the content of the items or from information about the users' preferences. The task of the learning method is to select a function based on a training set of m input vectors that can classify any item in the collection. The function $h(X)$ will either be able to classify an unseen item as positive or negative at once by returning a binary value or return a numerical value. In that case a threshold can be used to determine if the item is relevant or irrelevant to the user. A content-based filtering system selects items based on the correlation between the content of the items and the user's preferences as opposed to a collaborative filtering system that chooses items based on the correlation between people with similar preferences. PRES is a content-based filtering system. It makes recommendations by comparing a user profile with the content of each document in the collection. The content of a document can be represented with a set of terms. Terms are extracted from documents by running through a number of parsing steps [17]. In PRES the similarity between the profile vector and a document is determined by using the cosine measurement. Documents that have already been visited by the user or already appear as hyperlinks on the current page are filtered out. The remaining documents are then sorted by rank.

Content based methods suffer far less from the cold start problem than collaborative approaches: new users or items can be described by their characteristics (content) and so relevant suggestions can be done for these new entities. Only new users or items with previously unseen features will logically suffer from this drawback, but once the system becomes old enough, this has few to no chance to happen.

2.2.3 Matrix Factorization

The main assumption behind matrix factorisation is that there exists a pretty low dimensional latent space of features in which we can represent both users and items and such that the interaction between a user and an item can be obtained by computing the dot product of corresponding dense vectors in that space.
For example, consider that we have a user-movie rating matrix. In order to model the interactions between users and movies, we can assume that:

- there exists some features describing (and telling apart) pretty well movies.
- these features can also be used to describe user preferences (high values for features the user likes, low values otherwise)

However we don't want to give explicitly these features to our model (as it could be done for content based approaches that we will describe later). Instead, we prefer to let the system discover these useful features by itself and make its own representations of both users and items. As they are learned and not given, extracted features taken individually have a mathematical meaning but no intuitive interpretation (and, so, are difficult, if not impossible, to understand as human). However, it is not unusual to ends up having structures emerging from that type of algorithm being extremely close to intuitive decomposition that human could think about. Indeed, the consequence of such factorisation is that close users in terms of preferences as well as close items in terms of characteristics ends up having close representations in the latent space.

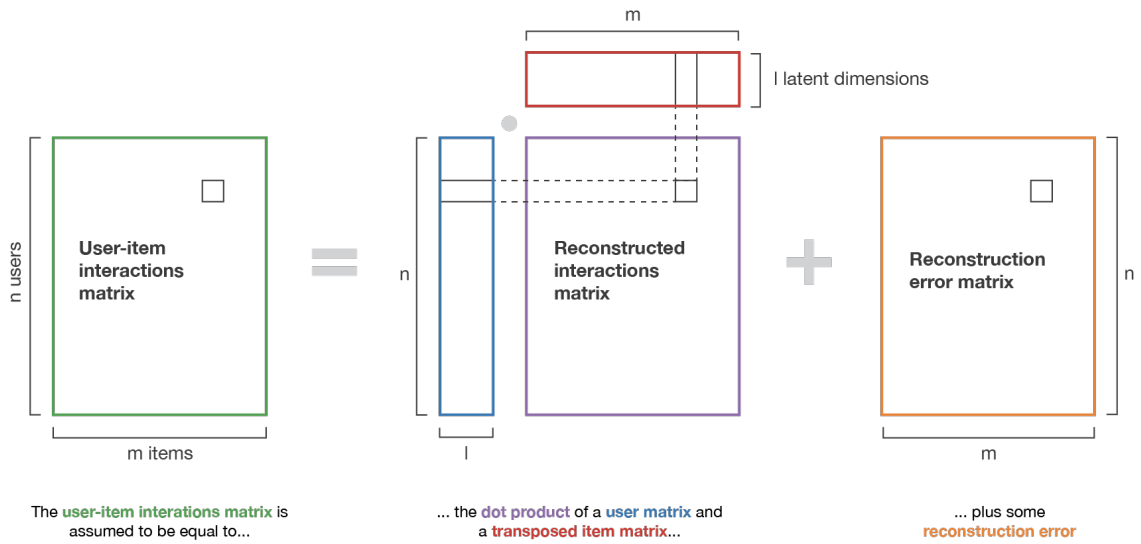


Figure 2.4: Illustration of the matrix factorization method

In this subsection, we will give a simple mathematical overview of matrix factorization. More especially, we describe a classical iterative approach based on gradient descent that makes possible to obtain factorizations for very large matrices without loading all the data at the same time in computer's memory.

Let's consider an interaction matrix M (nxm) of ratings where only some items have been rated by each user (most of the interactions are set to None to express the lack of rating). We want to factorise that matrix such that

$$M \approx X.Y^T$$

where X is the "user matrix" (nxl) whose rows represent the n users and where Y is the "item matrix" (mxl) whose rows represent the m items:

$$user_i \simeq X_i \quad \forall_i \in \{1,...,n\}$$

$$item_j \simeq Y_i \quad \forall_j \in \{1,...,m\}$$

Here l is the dimension of the latent space in which users and item will be represented. So,

we search for matrices X and Y whose dot product best approximate the existing interactions. Denoting E the ensemble of pairs (i,j) such that M_ij is set (not None), we want to find X and Y that minimise the "rating reconstruction error"

$$(X, Y) = argmin \sum_{(i,j)\in E} [(X_i)(Y_j)^T - M_{ij}]^2$$

The matrices X and Y can then be obtained following a gradient descent optimisation process for which we can notice two things. First, the gradient do not have to be computed over all the pairs in E at each step and we can consider only a subset of these pairs so that we optimise our objective function “by batch”. Second, values in X and Y do not have to be updated simultaneously and the gradient descent can be done alternatively on X and Y at each step (doing so, we consider one matrix fixed and optimise for the other before doing the opposite at the next iteration).

Once the matrix has been factorised, we have less information to manipulate in order to make a new recommendation: we can simply multiply a user vector by any item vector in order to estimate the corresponding rating. Notice that we could also use user-user and item-item methods with these new representations of users and items: (approximate) nearest neighbours searches wouldn’t be done over huge sparse vectors but over small dense ones making some approximation techniques more tractable.

2.2.4 Factorization Machines

Factorization Machines (FM) [18] are a new model class that combines the advantages of Support Vector Machines (SVM) with factorization models. Like SVMs, FMs are a general predictor working with any real valued feature vector. In contrast to SVMs, FMs model all interactions between variables using factorized parameters. Thus they are able to estimate interactions even in problems with huge sparsity (like recommender systems) where SVMs fail.

There are many different factorization models like matrix factorization, parallel factor analysis or specialized models like SVD++, PITF or FPMC. While these models require specialized

input data and are not applicable for general prediction tasks, FMs can mimic these models just by specifying the input data (i.e. the feature vectors). This makes FMs easily applicable even for users without expert knowledge in factorization models.
In sparse environments, not enough information is usually available to explicitly and independently estimate interactions between variables. Even in these environments, factorization machines can accurately estimate interactions by desecrating their independence from interaction parameters. Generally speaking, this means that information for one interaction also helps to determine the parameters for related interactions. FMs model all possible interactions between values in the feature vector x using factorized interactions instead of full parametrized ones. This has two main advantages:

(i) The interactions between values can be estimated even under high sparsity. Especially, it is possible to generalize to unobserved interactions.

(ii) The number of parameters as well as the time for prediction and learning is linear. This makes direct optimization using SGD feasible and allows optimizing against a variety of loss functions.

It has been proved in [18], that FMs have a closed model equation that can be computed in linear time. Thus, the model parameters (w_o, $\mathbf{w}$ and $\mathbf{V}$) of FMs can be learned efficiently by gradient descent methods – e.g. stochastic gradient descent (SGD) – for a variety of losses, among them are square, logit or hinge loss. The gradient of the FM model is:

$$\frac{\delta}{\delta\theta}\hat{y}(x) = \begin{cases} 1, & if\ \theta\ is\ w_o \\ x_i & if\ \theta\ is\ w_i \\ x_i \sum_{i=1}^{n} v_{j,f} x_j v_{i,f} x_i^2 & if\ \theta\ is\ v_{i,f} \end{cases}$$

The sum $\sum_{i=1}^{n} v_{j,f} x_j$ is independent of i and thus can be precomputed (e.g. when computing $\hat{y}(x)$). In general, each gradient can be computed in constant time *O(1)*. And all parameter updates for a case ($\mathbf{x}$, y) can be done in *O(kn)* – or *O(km(**x**))* under sparsity.

2.3 Deep Learning for Recommender Systems

Deep learning can be generally considered to be subfield of machine learning. The typical defining essence of deep learning is that it learns deep representations, i.e., learning multiple levels of representations and abstractions from data. For practical reasons, we consider any neural differentiable architecture as 'deep learning' as long as it optimizes a differentiable objective function using a variant of stochastic gradient descent (SGD). Neural architectures have demonstrated tremendous success in both supervised and unsupervised learning tasks [19]. Before diving into the whys and hows of deep learning in recommender systems, it is important that we are accustomed to some of the architectural paradigms concerning deep learning:

- **Multilayer Perceptron (MLP)** is a feed-forward neural network with multiple (one or more) hidden layers between the input layer and output layer. Here, the perceptron can employ arbitrary activation function and does not necessarily represent strictly binary classifier. MLPs can be intrepreted as stacked layers of nonlinear transformations, learning hierarchical feature representations. MLPs are also known to be universal approximators.

- **Autoencoder (AE)** is an unsupervised model attempting to reconstruct its input data in the output layer. In general, the bottleneck layer (the middle-most layer) is used as a salient feature representation of the input data.There are many variants of autoencoders such as denoising autoencoder, marginalized denoising autoencoder, sparse autoencoder, contractive autoencoder and variational autoencoder (VAE) [20, 21].

- **Convolutional Neural Network (CNN)** [21] is a special kind of feedforward neural network with convolution layers and pooling operations. It can capture the global and local features and significantly enhance the efficiency and accuracy. It performs well in processing data with grid-like topology.

- **Recurrent Neural Network (RNN)** [21] is suitable for modelling sequential data. Unlike feedforward neural network, there are loops and memories in RNN to remember former

computations. Variants such as Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) network are often deployed in practice to overcome the vanishing gradient problem.

- **Restricted Boltzmann Machine (RBM)** is a two layer neural network consisting of a visible layer and a hidden layer. It can be easily stacked to a deep net. Restricted here means that there are no intra-layer communications in visible layer or hidden layer.

- **Attentional Models (AM)** are differentiable neural architectures that operate based on soft content addressing over an input sequence (or image). Attention mechanism is typically ubiquitous and was incepted in Computer Vision and Natural Language Processing domains. However, it has also been an emerging trend in deep recommender system research.

It should be noted that every year, there are many advanced versions of neural networks coming out, but here we list just a few briefly. For those who want to learn more or more about advanced models, refer to [21].
The description of these paradigms has been cited and quoted from [22].
So, why exactly apply deep learning techniques to recommender systems? It is evident that numerous deep recommender systems have been proposed in a short span of several years. At this point, it would be easy to question the need for so many different architectures and/or possibly even the utility of neural networks for the problem domain. Along the same tangent, it would be apt to provide a clear rationale of why each proposed architecture and to which scenario it would be most beneficial for. All in all, this question is highly relevant to the issue of task, domains and recommender scenarios. One of the most attractive properties of neural architectures is that they are (1) end-to-end differentiable and (2) provide suitable inductive biases catered to the input data type. As such, if there is an inherent structure that the model can exploit, then deep neural networks ought to be useful.
To summarise briefly, the strengths of deep learning based recommender systems can be stated as follows [22]:

- **Nonlinear Transformation**. Contrary to linear models, deep neural networks is ca-

pable of modelling the non-linearity in data with nonlinear activations such as relu, sigmoid, tanh, etc. This property makes it possible to capture the complex and intricate user item interaction patterns. Conventional methods such as matrix factorization, factorization machine, sparse linear model are essentially linear models. For example, matrix factorization models the user-item interaction by linearly combining user and item latent factors [23]; it is well-established that neural networks are able to approximate any continuous function with an arbitrary precision by varying the activation choices and combinations [24, 25]. This property makes it possible to deal with complex interaction patterns and precisely reflect user's preference.

- **Representation Learning.** The advantages of using deep neural networks to assist representation learning are in two-folds: (1) it reduces the efforts in hand-crafted feature design. Feature engineering is a labor intensive work, deep neural networks enable automatically feature learning from raw data in unsupervised or supervised approach; (2) it enables recommendation models to include heterogeneous content information such as text, images, audio and even video.

- **Sequence Modelling.** Deep neural networks have shown promising results on a number of sequential modelling tasks such as machine translation, natural language understanding, speech recognition, chatbots, and many others. RNN and CNN play critical roles in these tasks. RNN achives this with internal memory states while CNN achieves this with filters sliding along with time. Both of them are widely applicable and flexible in mining sequential structure in data. Modelling sequential signals is an important topic for mining the temporal dynamics of user behaviour and item evolution. For example, next-item/basket prediction and session based recommendation are typical applications. As such, deep neural networks become a perfect fit for this sequential pattern mining task.

- **Flexibility.** Deep learning techniques possess high flexibility, especially with the advent of many popular deep learning frameworks such as Tensorflow, Keras, Caffe, MXnet, DeepLearning4j, PyTorch, Theano9 etc. Most of these tools are developed in a modular

way and have active community and professional support. The good modularization makes development and engineering a lot more effcient e.g. it is easy to combine different neural structures to formulate powerful hybrid models, or replace one module with others. Thus, we could easily build hybrid and composite recommendation models to simultaneously capture different characteristics and factors.

Now, we shall look at a few state of the art models that employ the powers of deep learning in recommendation systems.

2.3.1 Wide and Deep

The main inspiration behind the WDL model [26] is combining the power of memorization and generalization. Memorization can be loosely defined as learning the frequent co-occurrence of items or features and exploiting the correlation available in the historical data. Generalization, on the other hand, is based on transitivity of correlation and explores new feature combinations that have never or rarely occurred in the past. Recommendations based on memorization are good for items that the user has previously interacted with. Generalization, on the other hand, tends to improve the diversity of the recommended items.
In industry settings, for massive-scale online recommendation and ranking systems, generalized linear models such as logistic regression are widely used because they are simple, scalable and interpretable. The models are often trained on binarized sparse features with one-hot encoding. E.g., the binary feature "user_installed_app=netflix" has value 1 if the user installed Netflix. Memorization can be achieved effectively using cross-product transformations over sparse features, such as AND(user_installed_app=netflix, impression_app=pandora"), whose value is 1 if the user installed Netflix and then is later shown Pandora. This explains how the co-occurrence of a feature pair correlates with the target label. Generalization can be added by using features that are less granular, such as AND(user_installed_category=video, impression_category=music), but manual feature engineering is often required. One limitation of cross-product transformations is that they do not generalize to query-item feature pairs that have not appeared in the training data.
To combine the power of both memorization and generalization, Google introduced the Wide

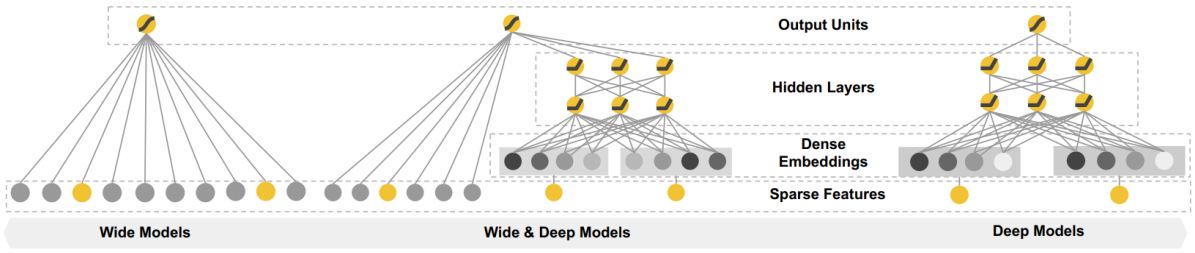


Figure 2.5: The spectrum of Wide Deep models.

and Deep Framework [26] for jointly training feed-forward neural networks with embeddings and linear model with feature transformations for generic recommender systems with sparse inputs. The basic architecture of the framework is shown in the middle of figure 2.5. The linear model captures niche features that are relevant to specific users, while the neural network creates more diversified recommendations based on abstract and high order combination of features.

Formally, the wide learning or linear part of the framework (left part of figure 2.5) is is defined as follows:

$$y = W_{wide}^{T}\{x, \phi(x)\} + b$$

where, W_{wide}^{T} and b are model parameters, $\{x, \phi(x)\}$ is the concatenated feature set consisting of raw input feature x and cross product transformed features $\phi(x)$. Cross product feature transformation is quinteseential for this model as it captures the interactions between the binary features, and adds nonlinearity to the generalized linear model.

The deep component of the model (right of figure 2.5) is a feed forward neural network of the form:

$$a_{l+1} = f(W_{deep}^{(l)} a^{(l)} + b^{(l)})$$

where l indicates the l^{th} layer, and $f(\ldots)$ is the activation function. $W_{deep}^{(l)}$ and $b^{(l)}$ are weight and bias terms.

The combined model is illustrated in figure 2.5 (center). For a logistic regression problem, the model's prediction is:

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

where Y is the binary class label, $\sigma(\dots)$ is the sigmoid function, $\phi(x)$ are the cross product transformations of the original features x, and b is the bias term. W_{wide} is the vector of all wide model weights, and W_{deep} are the weights applied on the final activations $a^{(l_f)}$.
One of the major benefits of training the model in such a joint manner is that for joint training the wide part only needs to complement the weaknesses of the deep part with a small number of cross-product feature transformations, rather than a full-size wide model.
While this model shows promising results, it has two glaring drawbacks:

(i) Manual feature engineering is required for creating cross product feature transformations which requires domain experts and is thus expensive and time consuming.

(ii) The simple structure of the deep feed forward neural network means that it works in an uncontrolled manner and may create arbitrary previously unseen feature interactions that may hinder proper recommendations.

2.3.2 DeepFM

Currently, most CTR prediction models "are skewed to low-and high-order feature interaction," such as FNN, PNN [27] and other NN [23] models concentrate on implied high-order feature correlation, while LR, FM [18], etc. focus on explicit feature correlation, both considered by Google's 2016 Wide and Deep model, but the Wide component needs manual feature engineering participation. The paper's motive is very intuitive, both to understand the relationship between high and low-level features and to save additional feature engineering. The motivation of the paper is very intuitive, both to consider the high/low-level feature interaction and to save additional feature engineering. It is a feasible practice to use FM to replace the LR part of Wide. Of course, LR can construct higher order combination features based on a priori, while FM only considers second order.

The network architecture of DeepFM [28] consists of two parts. There is an FM(factorization machine) which is the wide part of the model and the deep part of the model is just a DNN classifier.

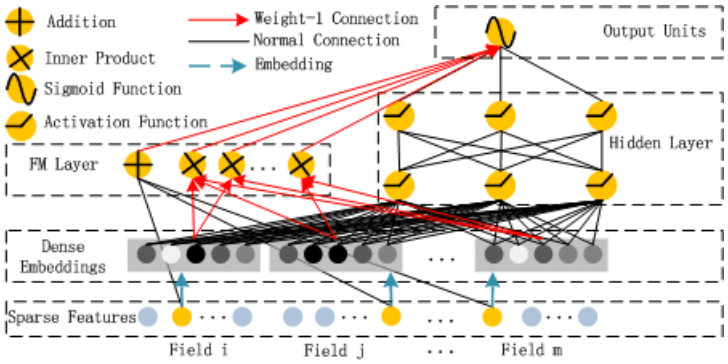


Figure 2.6: DeepFM network architecture

Looking closely at the above picture, the structure of DeepFM [28] is actually very simple: take the Wide Deep framework , the difference is to replace the LR of the Wide part with FM , thus automatically constructing the second-order feature cross multiplication instead of manually designing the cross multiplication.

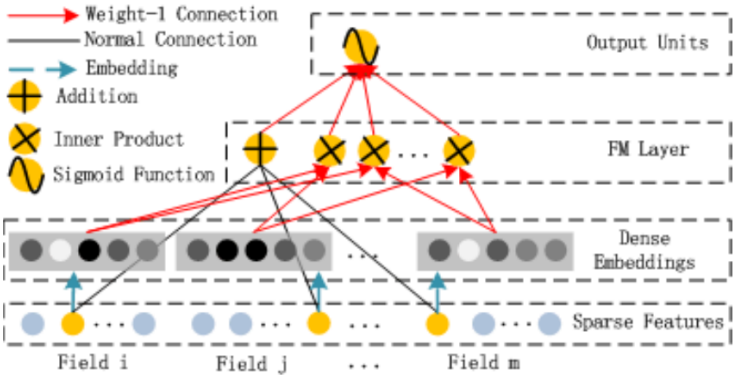


Figure 2.7: FM in the wide section

Pull out the Wide part of figure 2.7, which is the standard FM [18] structure, as shown in figure2.6 above. It is worth noting that the FM layer and the NN layer share the same feature embedding instead of embedding each learning their respective parts. The benefits of doing this are:

(i) Reducing the complexity of the model

(ii) Receiving feedback from the "low & high order interaction" part of the embedding learning to learn a better feature representation

Also, it can capture interactions of the order-2 feature much more effectively than previous approaches, especially when the dataset is sparse. In previous methods, only when feature i and feature j both appear in the same data record can the parameter of an interaction of features i and j be learned. The inner product of their latent vectors V_{i} and V_{j} is determined while in FM. FM is able to train latent vector V_{i}, V_{j} whenever i (or j) appears in a data record thanks to this dynamic model. Therefore, interactions of features that are never or rarely found in the training data are better learned by FM. The output of FM is the summation of an **Addition unit** and a number of **Inner Product units**:

$$y_{FM} = \langle w, x \rangle + \sum_{j_1=1}^{d} \sum_{j_2=j_1+1}^{d} \langle V_i, V_j \rangle x_{j1} \cdot x_{j2}$$

where $w \in R^{\mathrm{d}}$ and $V_{\mathrm{i}} \in R^{\mathrm{k}}$ (k is given). The Addition unit ($\langle w, x \rangle$) reflects the importance of order-1 features, and the Inner Product units represent the impact of order-2 feature interactions.

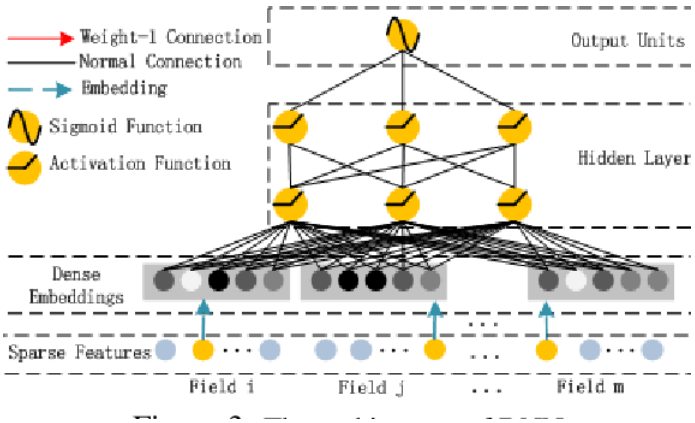


Figure 2.8: DNN in the Deep section

The deep neural network component is a feed-forward network used to learn interactions with high-order features. A data record (a vector) will be fed into the neural network as shown in figure2.8. The structure of the sub-network takes input from the embedding layer. The two interesting features of this network structure that are to be emphasized are: 1) while different input field vectors may have different lengths, their embedding is of the same size (k); 2) latent vectors (V) in the FM serves as network weights trained and used to transform input

field vectors into embedding vectors. The output of the embedding layer is denoted by:

$$a^{(0)} = [e_1, e_2, e_3, ..., e_m]$$

where e_i is the embedding of the i-th field and m is the number of fields. Then, $a^{(0)}$ is fed into the DNN and the forward process is:

$$a^{(l+1)} = \sigma(W^{(l)} a^{(l)} + b^{(l)})$$

where l is the layer depth and σ is an activation function. $a^{(l)}$, $W^{(l)}$, $b^{(l)}$ are the output, model weight, and bias of the l-th layer.
The contributions of DeepFM [28] are:

(i) Proposing a novel neural network model combining the architectures of FM and DNN

(ii) DeepFM shares the same input vector for both the FM and Deep part compared to Wide & Deep [11] model.

(iii) It models lower-order feature interactions in the FM and higher-order feature interactions in the DNN without any manual feature engineering

Despite the major contributions of the model, it still has some drawbacks:

(i) The FM can only create two-order feature combinations, not more than that.

(ii) The higher order feature combinations are created in an uncontrolled manner thus many irrelevant feature combination forms.

2.3.3 Deep and Cross Network

Cross feature transformations have proven to show significant performance gain in a model's expressiveness when it comes to recommendation. Linear models [29] are simple, interpretable and easy to scale; however, they are limited in their expressive power. Then again, cross feature transformations are expensive as they often require exhaustive manual feature

engineering which causes the model to become expensive to train and deploy. One of the ways to mitigate manual feature engineering is the use of FM networks as in [28]. However, such models suffer from the fact that the FM module is only capable of generating low order interactions, mainly interactions of order 2. Higher order interactions are computationally expensive to calculate. Deep and Cross Network (DCN) [30] proposes a novel architecture that aims to avoid task-specific feature engineering by introducing a novel neural network structure - a cross network - that explicitly applies feature crossing in an automatic fashion. DCN consists of multiple layers, and the depth of each layer represents the order of interactions for that corresponding layer. Each layer produces higher-order interactions based on existing ones, and keeps the interactions from previous layers. The reason DCN performs better than deep neural network is that DCN can explicitly capture high order cross features and requires much fewer number of parameters. Deep neural network can capture very complex interaction among features but requires an order of magnitude more model parameters. So the cross network is jointly trained with a deep neural network which greatly improves the recommendation prowess of the model.

The basic architecture of DCN is shown in figure **??**

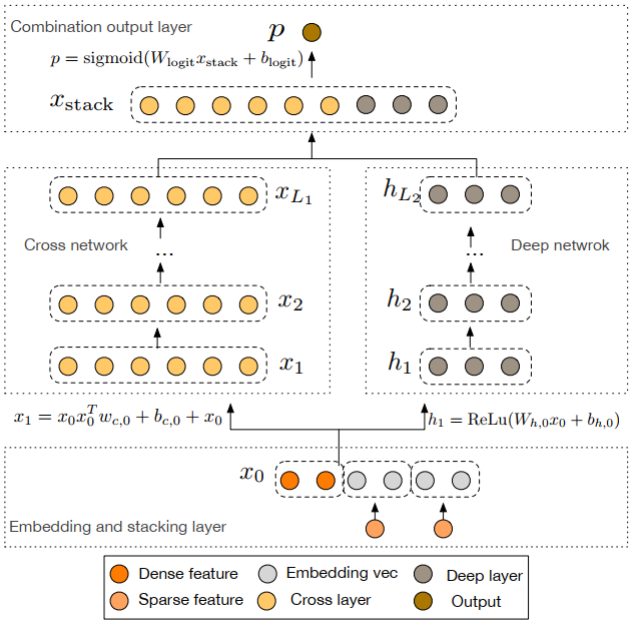


Figure 2.9: The Deep Cross Network

In CTR predictions, categorical features are often encoded into one-hot vectors e.g. [0,1,0]. But this introduces high dimensionality for large vocabularies. To reduce dimensionality, DCN employs an embedding layer and converts these binary features into dense vectors of real values i.e. embedding vectors:

$$x_{embed,i} = W_{embed,i} x_i$$

where $x_{embed,i}$ is the embedding vector, x_i is the binary input in the i^{th} category, and $W_{embed,i} x_i \in \mathbb{R}^{n_e X n_v}$ is the corresponding embedding matrix. The embedding vectors are finally stacked along with the normalized dense features x_{dense}, into one vector:

$$x_0 = [x^T_{embed,1}, \ldots, x^T_{embed,k}, x^T_{dense}]$$

Then, x_0 is fed into the network.

The cross network is composed of cross layers, with each layer being calculated using the following formula:

$$x_{l+1} = x_0 X_l^T w_l + b_l + x_l = f(x_l, w_l, b_l) + x_l$$

where x_l ; $x_{l+1} \in \mathbb{R}^d$ are column vectors denoting the outputs from the l^{th} and $(l+1)^{th}$ cross layers, respectively; $w_l, b_l \in \mathbb{R}^d$ are the weight and bias parameters of the l^{th} layer. Visually, a cross layer looks like figure 2.10:

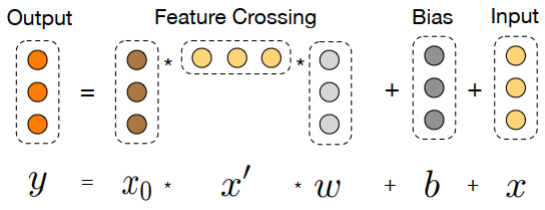


Figure 2.10: Visualization of a cross layer

The deep network is a fully-connected feed-forward neural network, with each deep layer having the ReLu function as the activation function.

The combination layer concatenates the outputs from two networks and feed the concate-

nated vector into a standard logits layer. The following is the formula for a two-class classification problem:

$$p = \sigma([x_{L_1}^T, h_{L_2}^T] w_{logits})$$

where,

$x_{L_1}^T \in \mathbb{R}^d$, $h_{L_2}^T \in \mathbb{R}^m$ = outputs from the cross network and deep network, respectively.

$w_{logits} \in \mathbb{R}^{d+m}$ = weight vector for the combination layer

$\sigma = 1/(1 + exp(x))$.

The standard logloss function is used as loss function with some regularization parameters.

Chapter 3

Proposed Architecture

This chapter presents our proposed modified recommendation models that utilizes the proposed transformed feature space.T he first section provides an overview of the proposed methodology by outlining the components of the system. In the subsequent sections, these components are described in details.

3.1 Overview

Our proposed model suggests that we use the raw features obtained from the dataset as input to the linear model. However, before feeding the features into the deep model, we do some preprocessing i.e. feature transformations that project both the categorical and real-valued features onto lower dimensional space and utilizes the high variance in data to capture most of the important information, while discarding the lesser important ones.

This results in a reduced feature set with the important information intact. This new feature set is fed into the deep model and trained jointly with the linear model. Standard logloss with backpropagation is used as loss function along with adam optimizer [31].

The workflow we have followed throughout our thesis has been shown in figure 3.1

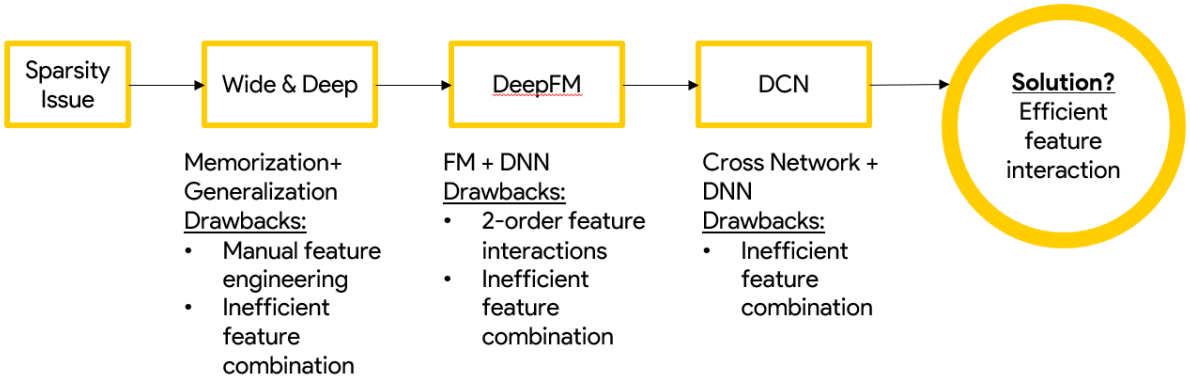


Figure 3.1: Research Workflow

3.2 Framework

Our proposed framework is shown in figure 3.2. We have highlighted the part we will be working on in yellow. As is evident in the framework, initially, the features will be normalized for both the linear and deep models. Thereafter, the sparse and dense features will be fed directly into the linear model. Note, that in case of Wide Deep model, manual feature crossings will also be included in the feature-set for linear model. As for the deep model, we perform our feature transformation on both the sparse and dense features and as output we get a reduced dataset. This reduced dataset is fed into the deep model. The output from both the linear and the deep model is passed into an activation function as the combined result is calculated.

3.3 Modules of Framework

Label Encoding and Feature Transformation

Datasets for machine learning often contain a combination of categorical features and dense features. Categorical features are non-numeric features (nominal, ordinal etc.) that cannot be directly processed by machine learning models. In such cases, we need to convert them to a format that is understandable by said models. We use the Label Encoder class to convert

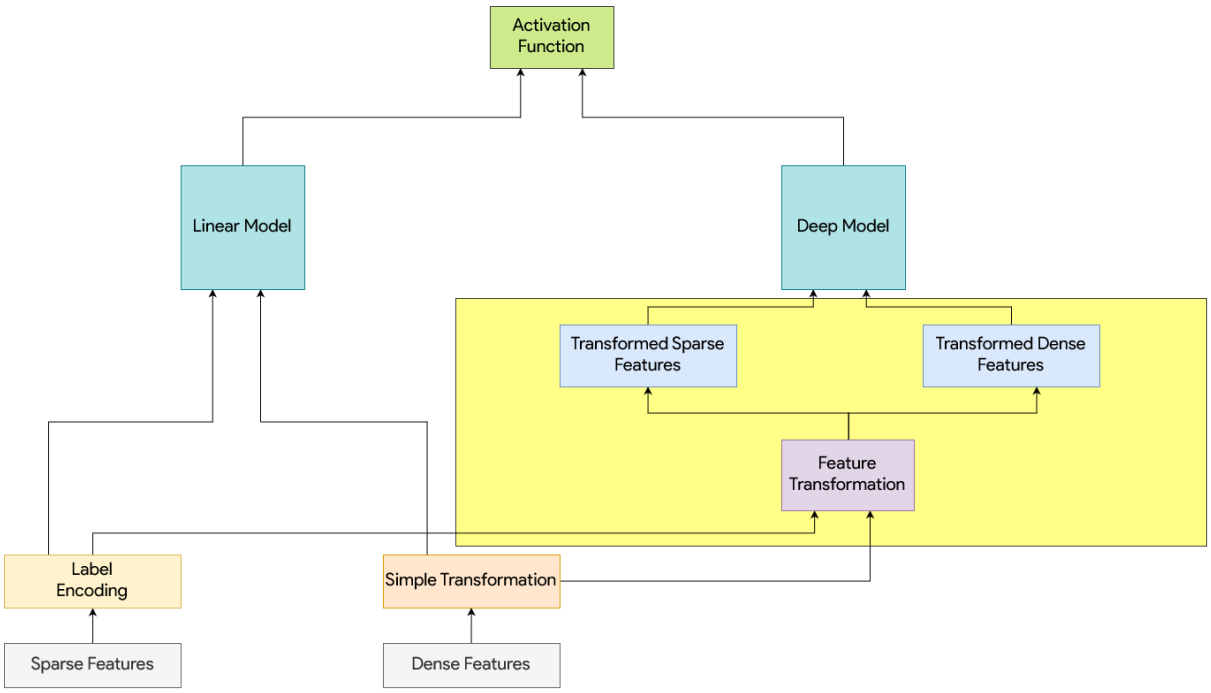


Figure 3.2: Proposed Architecture

this form of categorical text information into model-understandable numerical data. Thanks to open libraries like Scikit learn, we can do so very easily using the LabelEncoder class. All we have to do, is import the class, fit and transform the first column of the data, and then replace the existing text data with the new encoded data. The code will look something as follows:

```
18 for feat in sparse_features:
19     lbe = LabelEncoder()
20     data[feat] = lbe.fit_transform(data[feat])
```

Figure 3.3: Label Encoding

For the dense features, we need to scale or normalize them so that the model does not show any unwanted bias towards certain features. The transformation is also done using the help of Scikit learn. Sample code snippet is as follows:

```
21 mms = MinMaxScaler(feature_range=(0, 1))
22 data[dense_features] = mms.fit_transform(data[dense_features])
```

Figure 3.4: Simple Transformation

3.3.1 Feature Transformation

In this part, we perform the required feature transformation on our dataset to reduce it and project it onto a lower dimensional space. In this paper, we have applied two feature transformation techniques, Principal Component Analysis (PCA) and Kernel PCA (kPCA).

Principal Component Analysis (PCA):
PCA is a way to detect patterns in information and to convey data so that similarities and variations can be recognized. PCA is a strong tool for analyzing data since data patterns are difficult to find in high dimensions where the luxury of graphical representation is unavailable. The other major benefit of PCA is that once these data patterns have been detected, data can be compressed, i.e. by reducing the number of measurements, not much loss of information occurs. PCA projects the original feature space onto a low dimensional feature space, that maximizes the variance between samples. Thus, without much loss in information, we get a representation of the same data, but by using lesser number of features. The steps required to perform PCA are as follows:

- Get the dataset
- Subtract the mean of the dataset from each sample
- Calculate covariance matrix of the dataset
- Calculate eigenvectors and eigenvalues from the covariance matrix
- Select the eigenvectors with the highest eigenvalues. These vectors are the principal components
- Project the original data into the new vector space

This whole process is simplified by using Scikit learn's open library. Following code snippet shows the process of applying PCA in our case:

```
def myPCA(dnn_feature_columns,feature_type,n):
    pca = PCA(n_components=n) #Not passing any parameter will return all the eigen vectors or principal components
    X_train = pca.fit_transform(dnn_feature_columns)
    #X_test = pca.transform(X_test)
    explained_variance = pca.explained_variance_ratio_
    print("PCA Component contribution for feature type: " + feature_type + str(explained_variance))
    return X_train
```

Figure 3.5: Applying PCA

Kernel PCA:

According to Wikipedia,

> In the field of multivariate statistics, kernel principal component analysis (kernel PCA) is an extension of principal component analysis (PCA) using techniques of kernel methods. Using a kernel, the originally linear operations of PCA are performed in a reproducing kernel Hilbert space.

PCA is a linear method. That is, it can only be applied to datasets which are linearly separable. It does an excellent job for datasets, which are linearly separable. But, if we use it to non-linear datasets, we might get a result which may not be the optimal dimensionality reduction. Kernel PCA uses a kernel function to project data in a higher dimensional space where it can be segregated linearly. The concept of support vector machines is similar. So, while PCA does an excellent job at projecting datasets which are linearly separable, kernel PCA works better in terms of non linear datasets as PCA fails to transform them accurately. Kernel PCA can be applied using Scikit learn. Sample code snippet:

```
def kPCA(dnn_feature_columns,feature_type,n):
    kPCA = KernelPCA(n_components=n) #Not passing any parameter will return all the eigen vectors or principal compo
    X_train = kPCA.fit_transform(dnn_feature_columns)
    return X_train
```

Figure 3.6: Applying kPCA

3.3.2 Linear Model

In our experiments, we have applied our proposed approach to three models: Wide Deep (WDL), DeepFM and Deep Cross Network. The linear model in case of WDL is a a generalized linear model of the form $y = w^T x + b$, where

y = prediction

$x = x_1, x_2, \ldots, x_d$ is a vector of d features

$w = w_1, w_2, \ldots, w_d$ = the model parameters b = bias. The feature set includes both raw input features and transformed features. In this case, the transformed features are cross feature combinations of the raw features. For DeepFM, the linear model is an FM or Factorization Machine [28] as shown in figure 2.7. With the aim to capture high order feature interaction, besides a linear (order-1) interactions among features, FM also models pairwise (order-2) feature interactions as inner product of respective feature latent vectors. In case of Deep and Cross network, there is no linear model. However, there is a cross network that replaces the FM module or linear model and this cross network is responsible for explicitly capturing high order feature interactions in an efficient manner.

3.3.3 Deep Model

Deep Model is where our transformed or compressed features are fed. For the WDL and DeepFM models, the deep component is a simple feed forward neural network (FFM). In our experiments, we have used 128 hidden layers with 128 units to train the model. We have used an embedding layer of 8 and ReLu as the activation function of the hidden layers. In case of DCN, both the deep component and the DNN component uses our transformed feature set as input. The configuration of the DNN is the same as WDL and DeepFM.

3.3.4 Activation Function

We have used sigmoid function $\sigma(\ldots)$ as the final activation function. The error is back propagated to both the linear models and deep models so that they can be jointly trained.

Chapter 4

Evaluation

In this chapter, we discuss about the experimental setup, comparison, dataset, result analysis based on different criteria.

4.1 Experimental Setup

4.1.1 Dataset

The effectiveness and efficiency of our proposed method has been evaluated on the following dataset. **Criteo Dataset:** The Criteo dataset includes 45 million user click records.There are 13 continuous (real-valued) features and 26 categorical features. Though the exact nature of the features is unknown to us, according to a competition admin (Olivier Chapelle), they fall in the following categories:

(i) Publisher features, such as the domain of the url where the ad was displayed;

(ii) Advertiser features (advertiser id, type of products,...)

(iii) User features, for instance browser type;

(iv) Interaction of the user with the advertiser, such as the number of the times the user visited the advertiser website.

For our experiment,

(i) We randomly sampled 10,000 samples for overall training, validation and testing process

(ii) The same samples are used across all the models

(iii) 64% of the data has been used for training, 16% for validation and 20% is used for test data

4.2 Evaluation Methodologies

These evaluation methodologies were used to compare our proposed methodologies with previous literature:

4.2.1 Log-Loss

Log Loss is the most important classification metric based on probabilities. It's hard to interpret raw log-loss values, but log-loss is still a good metric for comparing models. For any given problem, a lower log-loss value means better predictions.
The logloss function is given below:

$$H_p(q) = -1/N \sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

where y is the label of our target value and $p(y)$ is the predicted probability for all N points. The graph below shows the range of possible log loss values given a true observation.

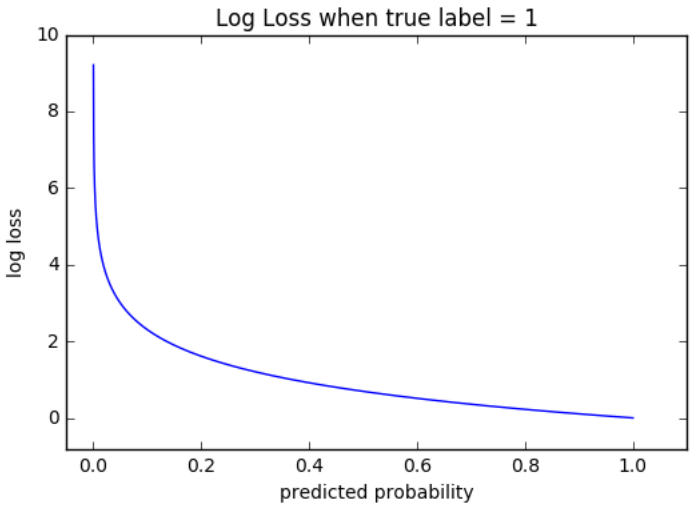


Figure 4.1: Logloss

As the predicted probability approaches 1, log loss slowly decreases. As the predicted probability decreases, however, the log loss increases rapidly. Log loss penalizes both types of errors, but especially those predictions that are confident and wrong!

4.2.2 AUC

When we need to check or visualize the performance of the multi - class classification problem, we use AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as AUROC (Area Under the Receiver Operating Characteristics). AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability.

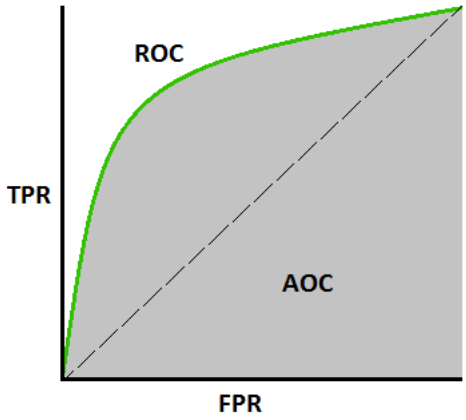
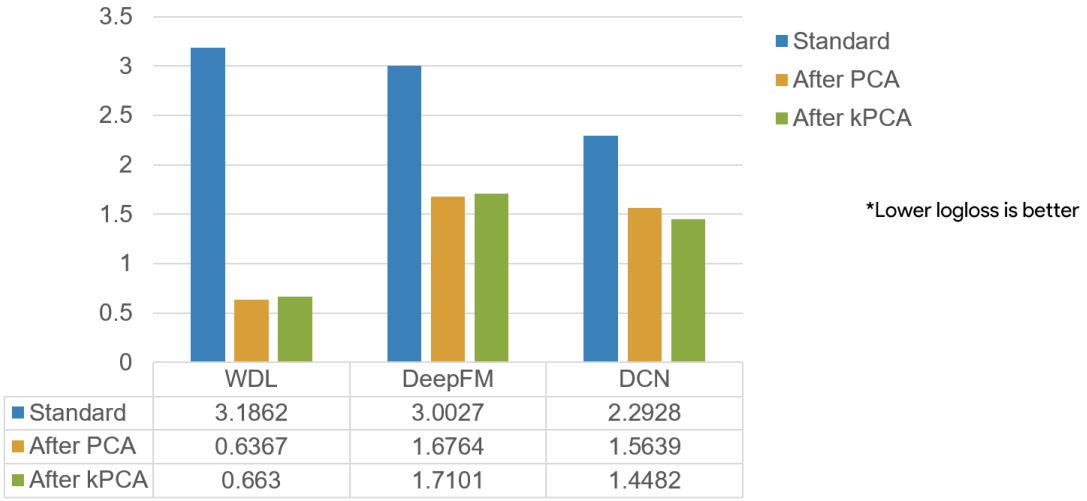


Figure 4.2: AUC and ROC

It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. By analogy, Higher the AUC, better the model is at distinguishing between patients with disease and no disease.

4.3 Result Analysis

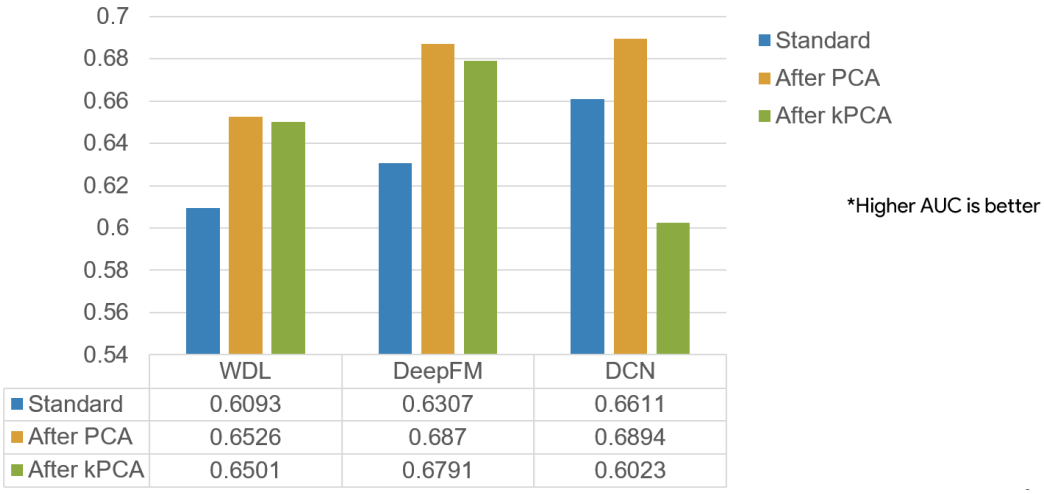
4.3.1 Log-Loss



	WDL	DeepFM	DCN
Standard	3.1862	3.0027	2.2928
After PCA	0.6367	1.6764	1.5639
After kPCA	0.663	1.7101	1.4482

Figure 4.3: Logloss Comparison

4.3.2 AUC (Area Under ROC)



	WDL	DeepFM	DCN
Standard	0.6093	0.6307	0.6611
After PCA	0.6526	0.687	0.6894
After kPCA	0.6501	0.6791	0.6023

Figure 4.4: AUC Comparison

4.4 Confusion Matrix

4.4.1 Comparison against WDL

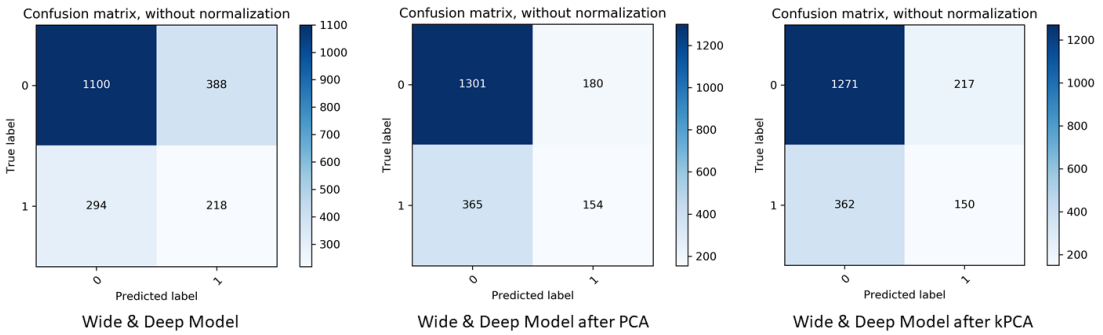


Figure 4.5: Wide & Deep Model Comparison

4.4.2 Comparison against DeepFM

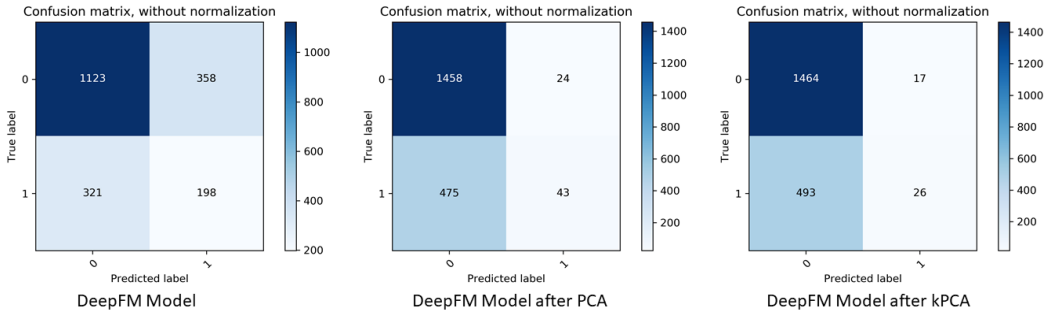


Figure 4.6: DeepFM Model Comparison

4.4.3 Comparison against DCN

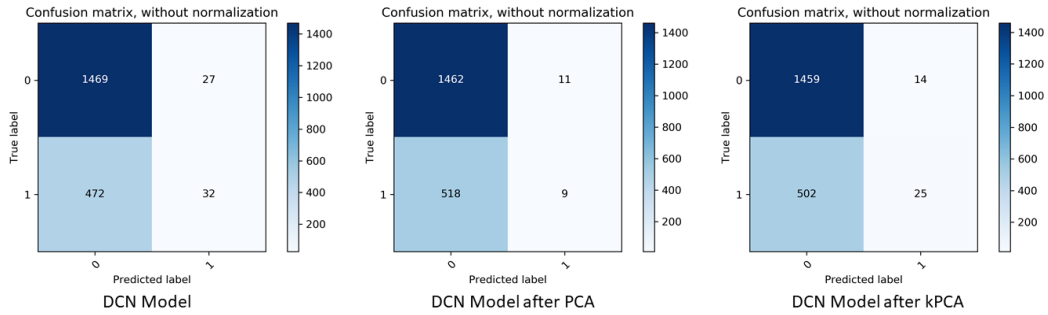


Figure 4.7: DCN Model Comparison

According to our Logloss and AUC comparison histograms, it is pretty obvious that the model performance substantially increases after applying our data transformation. Due to the data transformation, the DNN part of all the models make better predictions, thus improving the overall model performance. We can further increase our model performance by tuning the hyper parameters.

Looking at the confusion matrices, it is evident that our proposed modified model performs better when classifying the 0 label. However, a common theme that can be observed is that the modified model misclassifies the 1 label more than the original model. The reasoning behind this is that our dataset contains about 75% label 0, the remaining being label 1. Since

it's a skewed dataset, PCA and kPCA tend to favour the label 0 more as it carries the most variance. Hence, the misclassification of label 1 increases.

Chapter 5

Conclusion

5.1 Summary

The effective and efficient identification of feature interactions has been the key to success for many prediction models. Unfortunately, the process sometimes needs manual feature engineering and rigorous searching from domain experts. Use of DNNs has massively improved the learning of features from a dataset. However, most often the features learned are implicit and highly non-linear, consequently the network becomes unnecessarily large and inefficient in learning certain features. Our research provides insight into potential performance gain by using careful feature transformation techniques. The transformation or reduction of the feature space allows us to project both the categorical and real-valued features onto lower dimensional space and uses the high variance in data to capture most of the important features and discards the rest. The high priority features are then used to create feature combinations easily and efficiently which has significant impact in our prediction. This has a two-part impact: We can compress the dataset as well as discard the features that may hinder proper recommendation as they can be potential noise. Our experimental results have demonstrated improved performance over the state-of-art algorithms on both sparse and dense datasets, in terms of both model loss calculations and accuracy.

5.2 Future Work

The problems that we faced during our experiment is that when the dataset is skewed towards one class particularly, then the proposed model can not predict the other class with great accuracy. So, in our future work we would like to explore more in this area about how we can improve the prediction of both classes even if the dataset is skewed. Also, we have plans to implement different feature transformation techniques like, autoencoders [32], etc. We also want take a closer look into the deep model and improve it further for strengthening the ability to learn most useful higher order feature interactions. For that, we want try out different combinations of the nodes and layers of the network to improve overall prediction accuracy.

Bibliography

[1] J Ben Schafer, Joseph Konstan, and John Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166. ACM, 1999.

[2] Recommender systems survey. *Knowl.-Based Syst.*, 46:109–132, 2013.

[3] Ken Lang. Newsweeder: Learning to filter netnews. In *in Proceedings of the 12th International Machine Learning Conference (ML95*, 1995.

[4] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. *CoRR*, abs/1205.2618, 2012.

[5] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, pages 1257–1264, USA, 2007. Curran Associates Inc.

[6] Deepak Agarwal and Bee-Chung Chen. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 19–28, New York, NY, USA, 2009. ACM.

[7] Liang Hu, Jian Cao, Guandong Xu, Longbing Cao, Zhiping Gu, and Can Zhu. Personalized recommendation via cross-domain triadic factorization. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 595–606, New York, NY, USA, 2013. ACM.

[8] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 191–198, New York, NY, USA, 2016. ACM.

[9] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The youtube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 293–296, New York, NY, USA, 2010. ACM.

[10] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4):13:1–13:19, December 2015.

[11] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pages 7–10, New York, NY, USA, 2016. ACM.

[12] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 1933–1942, New York, NY, USA, 2017. ACM.

[13] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for CTR prediction. *CoRR*, abs/1703.04247, 2017.

[14] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA., 2007.

[15] Jonathan L Herlocker, Joseph A Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 241–250. ACM, 2000.

[16] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002.

[17] Robin Van Meteren and Maarten Van Someren. Using content-based filtering for recommendation. In *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*, pages 47–56, 2000.

[18] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.

[19] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[20] Minmin Chen, Zhixiang Xu, Kilian Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. *arXiv preprint arXiv:1206.4683*, 2012.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[22] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):5, 2019.

[23] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

[24] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[25] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[26] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10. ACM, 2016.

[27] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. Product-based neural networks for user response prediction. *CoRR*, abs/1611.00144, 2016.

[28] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[29] Olivier Chapelle, Eren Manavoglu, and Romer Rosales. Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(4):61, 2015.

[30] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, page 12. ACM, 2017.

[31] DP Kingma and JL Ba. Adam: A method for stochastic optimization. arxiv 2014. *arXiv preprint arXiv:1412.6980*, 2014.

[32] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010.