# NEURAL COLLABORATIVE FILTERING AND AUTOENCODER ENABLED DEEP LEARNING MODELS FOR RECOMMENDER SYSTEMS

A Thesis Submitted to the Department of Computer Science,

African University of Science and Technology

In partial fulfilment of the requirement for the award of Master of Science in Computer Science

By

Arnold Kwofie

(40852)

Abuja, Nigeria

November, 2022

## CERTIFICATION

This is to certify that, the thesis titled “**Neural Collaborative Filtering and Autoencoder Enabled Deep Learning Models for Recommender Systems**” submitted to the school of postgraduate studies, African University of Science and Technology (AUST), Abuja, Nigeria, for the award of the Master’s degree is an original work carried out by Arnold Kwofie in the Department of Computer Science.

**NEURAL COLLABORATIVE FILTERING AND AUTOENCODER ENABLED DEEP LEARNING MODELS FOR RECOMMENDER SYSTEMS**

By

Arnold Kwofie

A THESIS APPROVED BY THE COMPUTER SCIENCE DEPARTMENT

APPROVED BY:

**Supervisor**

Prof. Mohammed Hamada

Signature: [illegible]

Jan 12, 2023

**The Head of the Department**

Assoc. Prof Rajesh Prasad

Signature: [illegible] 18 Feb 2023

## ABSTRACT

Finding important and useful information is getting harder as much more information is available online. The challenge for content producers is to deliver the appropriate content to the appropriate consumers while making it challenging for users to access that content. The foundation for overcoming these difficulties is provided by recommender systems. Traditional methods like Collaborative Filtering (CF) and Content-Based Recommender Systems have historically been successful in this field of study but are now challenged by problems with data sparsity, cold start, and non-linearity interaction. Evidently, several academic areas, like image detection and natural language processing (El-Bakry, 2008), have shown great interest in deep learning due to outstanding performance and the alluring quality of learning intricate representations. The impact of deep learning is recently showing good advancement when applied to recommender systems research (He, 2008). In this research We dive deep into the Autoencoder and Neural Collaborative Filtering based deep learning models and their implementation on classical collaborative filtering. The research also evaluates the performance of both models and outlines loopholes which can further be improved in future works.

## DEDICATION

This work is foremost dedicated to God for giving me the strength and knowledge in the course of the work. Also, to my family and loved ones for the immense support. And finally, to my supervisor, Prof Mohammed Hamada for his guidance throughout the project.

## ACKNOWLEDGEMENT

My profound gratitude goes to God for the strength given to me to reach this far in my program. I would also like to thank my supervisor, Prof. Mohammed Hamada, for his guidance, advice, and support. My appreciation also goes to all my lecturers for the knowledge with they shared with me throughout this journey. I am grateful to the African University of Science Technology and its staff for giving me the opportunity to study at such a prestigious institution. Finally, I thank all and sundry who supported me in this work and my entire period with the University. I wish you success in all your endeavors.

# TABLE OF CONTENTS

## TABLE OF FIGURES

## LIST OF ABBREAVIATIONS

| | | |
|---|---|---|
| NCF | - | Neural Collaborative Filtering |
| CF | - | Collaborative Filtering |
| MF | - | Matrix Factorization |
| CB | - | Content Based |
| RSs | - | Recommender Systems |
| SVD | - | Single Value Decomposition |
| DAE | - | Denoising Autoencoders |
| NN | - | Neural Network |
| DNN | - | Deep Neural Network |
| NeuMF | - | Neural Matrix Factorization |
| GMF | - | General Matrix Factorization |
| MLP | - | Multi-Layer Perceptron |
| RMSE | - | Root Mean Squared Error |

# CHAPTER ONE

# 1. INTRODUCTION

## 1.1 Background of Study

Intelligent systems called recommender systems leverage user ratings on previously purchased goods to suggest comparable goods to other customers. These systems actively reduce the navigation options for visitors based on their preferences, which is vital for online businesses. Information overload is the main issue that recommender systems address. Modern internet technology has altered several ways to interact and share information (Hamada, 2017). For larger businesses, having a successful recommender system would increase their revenue. A recommender system's customized content would enhance the user's experience and help them save a lot of time (Sarwar, 2011). Many strategies of supervised learning methods have been used to predict user preferences for product from a large product category using datasets of numeric preferences over a period of time. close. (e.g., 1 to 10) (Hassan and Hamada, 2017). The most commonly used algorithms are, collaborative filtering, hybrid method, and content-based filtering. The hybrid and collaborative filtering systems make suggestions for users based on many criteria. For instance, CF-based methods rely on past ratings on products while hybrid methods, join two or more recommender system methods. Collaborative filtering-based methods for personalized suggestions became popular due to some security problems with Content Based methods, such as collecting user profile information. Matrix Factorization is probably the well-known method. This approach is based on user-item function, which can be modeled or depicted as the internal product of hidden vectors. Deep learning recently has championed the course when applying to Recommender System due to its ability to solve issues of sparsity, cold start and non-linearity representation of data. Deep learning Method is an advanced method over machine learning algorithms such that it is able to learn complicated data. It iterates over the data a couple of times to find better relationships that can be employed to provide better suggestions. Since it is an upcoming emerging field, this research aims at using deep learning techniques to improve recommendations. Deep learning techniques like multilayer perceptron or an autoencoder should be able to learn well and provide more accurate recommendations when applied to recommender systems. This model has been utilized for recommendation in several recent works, however these works concentrated on content descriptions, such as item content information. Even though, these models are still in use, the concentration is only on applying the ML algorithms on the hidden characteristics by users and items, ignoring the crucial user-item interaction function of collaborative filtering (Sawar, 2001). In this project, a neural network architecture takes the place of the inner item and learns a user-item relationship function from data. if the user-item relationship function exhibits any non-linearities, to handle them.

## 1.2 Problem Statement

Users now have access to enormous volumes of data and material, but because there are so many options available, exploring the data is challenging. This poses a challenge for all parties. The entities giving out the service challenged with the point of reaching right users with the right content, and most often, are forced to predict the well-known content. The content creators struggle to reach relevant users with their work and users struggle to find this content. Recommendation systems serves as the foundation of these predicaments. To make specific recommendations, the ongoing process must be studied to produce more specific recommendations (Ricci B, 2011). Machine learning approaches like Single value decomposition and matrix factorization (MF) have been used in this area of knowledge. However Deep Learning methods implement accurate recommendations as it iterates over the data many more times through multiple layers trying to find better relationships.

## 1.3 Objective of Study

Deep Learning is now a well-known approach in a wide scope of areas of research in the computing industry. Therefore, it is of interest to investigate the possibility of representing the challenges of recommendation as a classification structured problem, so that we can determine what neural networks are usable for creating a more specific recommendation. We discuss the Autoencoder and Collaborative based deep learning approaches, their accuracies, losses, loopholes and what can be done to personalize recommendations more effectively

## 1.4 Outline of the Study

The first chapter focuses on the background of the study, the problem statement and objectives. The second Chapter will look at theoretical background of recommender systems, some machine learning models in this area of research and an extensive literature review on the subject matter. The third chapter will outline in detail the methodologies and tools used in this project. In the fourth chapter, various findings and discussions will emphatically be discussed. The fifth chapter will summarize the project and suggest recommendations.

# CHAPTER TWO

# 2. LITERATURE REVIEW

This Chapter gives a comprehensive discussion on recommender systems and its improvement and flaws over the years classical machine learning models. The Chapter also contains reviews of work done by others. It focuses on the general knowledge established on this topic.

## 2.1 Recommender Systems

Recommendation systems (RS) are software methods that provide predictions on things that may be useful to a user. Recommendations involve different thinking processes, such as what to purchase or what online news to read. An example is a book recommendation platform that helps people choose what to read. The Amazon uses a recommender system to tailor its online store to each consumer. Because recommendations are frequently tailored, various individuals or user groups see a variety of recommendations. Non-personalized recommendations are another option which are perhaps easier to come by. Some examples can be the top five books. As they can be helpful in some circumstances, these methods are not really addressed by recommender system research (Ricci B, 2011). Depending on the user's likes and limitations, RSs attempt to forecast the best products or services while performing this rating. Users' preferences are gathered by RSs in order to fulfill such a complex task. These preferences can either be expressed openly, such as through product ratings, or they can be inferred through analyzing user behavior. For instance, an RS can interpret a user's movement to a specific product site as an implicit endorsement of the goods displayed there. The evolution of RS started with a fairly simple observation: people often depend on experiences provided by others to make day-to-day decisions (Sarwar, 2011).

### 2.1.1 Content Based Recommender Systems

Content-based filtering is a type of recommendation algorithm that uses the characteristics of the items being recommended to make recommendations. This approach is based on the idea that items with similar characteristics will be of interest to the same users, and that those characteristics can be used to make recommendations. For example, if a user likes action movies, a content-based algorithm may recommend other action movies to that user based on their shared characteristics. Figure 1 illustrates how content-based filtering works. In this example, a user has rated a number of movies, and the recommendation system has identified that the user likes movies in the comedy genre. The system then uses the characteristics of the movies (in this case, the genre) to recommend other movies in the comedy genre that the user may be interested in. This approach can help to make more personalized and accurate recommendations, as it takes into account the specific preferences of the user.

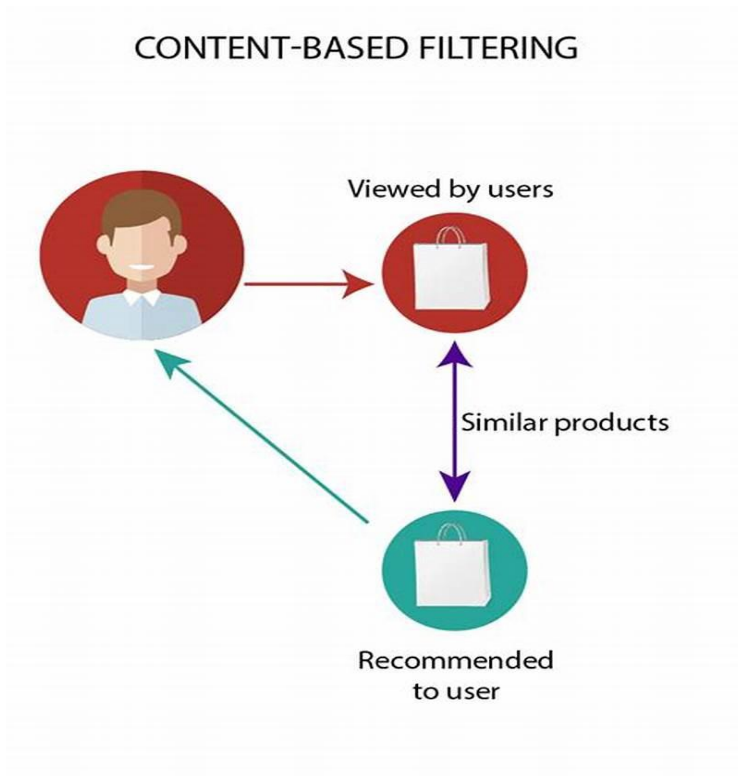


*Figure 1: Content based recommender system*

**Architecture of Content Based Recommender System**

The architecture of a content-based recommender system typically consists of three main components: the content analyzer, the profile learner, and the filtering component.

**The content analyzer** is responsible for preprocessing and extracting relevant information from the unstructured data that is used by the system. This may involve techniques such as feature extraction and natural language processing to convert the data into a form that can be used by the other components of the system.

**The profile learner** uses the data generated by the content analyzer to construct user profiles. This is typically done using machine learning algorithms, which are able to generalize the data and identify patterns in the preferences of users.

**The filtering component** uses the user profiles generated by the profile learner to identify items that are likely to be of interest to a particular user. This is done by matching the profile representation of the items to the user profile, and recommending items that are deemed to be a good match. This component is responsible for making the final recommendations to the user.

Together, these components work to analyze the content of items, learn the preferences of users, and recommend items that are likely to be of interest to those users.

**Advantages of Content Based Recommender System**

- User Independence- Content-based recommenders only use explicit ratings by same user to construct their profiles. Instead, collaborative filtering methods depends on ratings from other users to get the "nearest neighbor" of the user in question,
- New Item - Content-based recommenders can recommend items that haven't gotten any rating by other users. As a result, they are not face with the first ranker challenge.

**Disadvantages of Content Based Recommender System**

One potential limitation of content-based recommender systems is that they rely on the analysis of the content of items in order to make recommendations. This means that the system needs to have access to a sufficient amount of information about the items in order to make accurate recommendations. If the analyzed content does not contain enough information to discriminate between items that the user likes and those that the user does not like, the system may not be able to provide suitable suggestions. This can be a particular challenge in domains where the information available about the items is limited or difficult to extract, such as in the case of unstructured text data.

Another potential limitation of content-based recommender systems is that they may not perform well for new users who have not yet generated a sufficient number of ratings. Since these systems rely on the analysis of user preferences in order to make recommendations, they need a sufficient amount of data to learn the preferences of users before they can provide accurate suggestions. This can be a particular challenge when dealing with users who have not yet rated many items, as the system may not have enough data to accurately predict their preferences.

### 2.1.2 Collaborative Filtering

Recently, there has been a lot of progress and interest in the collaborative filtering (CF) approach to recommenders. Its popularity has been boosted by the fact that it was a major player in the Netflix competition. Collaborative filtering (CF) approaches provide specific suggestions of things subject to tends of ratings (e.g., purchases). buying history, browser history, items searched, and sometimes mouse motions are examples of implicit feedback, while ratings of products from 1 to 5 are examples of explicit feedback (Ricci B, 2011). There are two main methods used in collaborative filtering. Neighborhood-based methods and latent factor methods are two common approaches used in collaborative filtering-based recommendation systems. Neighborhood-based methods use the interactions between items or users to make recommendations. For example, an item-item method might construct the preferences of a user for a particular item based on the preferences of that user for similar items. This approach relies on the idea that users who have similar preferences will tend to rate items similarly, and that those ratings can be used to make recommendations to other users. Latent factor methods, such as matrix factorization, use a different approach. These methods represent both items and users in a hidden factor space, where the ratings given by users to items can be explained by the features of the items and the users on the inferred factors. This approach is based on the idea that the preferences of users can be represented by a small number of latent factors, and that those factors can be used to make predictions about the preferences of users. Both of these approaches have their own strengths and weaknesses, and the appropriate approach will depend on the specific requirements of the recommendation system.

#### 2.1.2.1 Neighborhood Approach

A neighborhood-based recommender system is a type of collaborative filtering algorithm that uses the ratings and preferences of users to make recommendations. It does this by identifying users who have similar preferences and then making recommendations based on the preferences of those users. For example, if two users have both rated a particular movie highly, the recommendation system may recommend the movie to a third user who has not yet seen it. This approach can help to make more personalized and accurate recommendations, as it takes into account the preferences of users who are similar to the user being recommended to.. We then use accuracy to evaluate the performance of the recommendation system. In so doing, the ratings R is separated into train set and test set to evaluate the prediction accuracy. Popular measures of accuracy that can be used are:

$$\text{MAE}(f) = \frac{1}{|\mathcal{R}_{\text{test}}|} \sum_{r_{ui} \in \mathcal{R}_{\text{test}}} |f(u,i) - r_{ui}|,$$

*Root Mean Squared Error* (RMSE):

$$\text{RMSE}(f) = \sqrt{\frac{1}{|\mathcal{R}_{\text{test}}|} \sum_{r_{ui} \in \mathcal{R}_{\text{test}}} (f(u,i) - r_{ui})^2}.$$

There are two ways to use the neighborhood model, known as user-based or item-based recommendations. The evaluations of this item by nearby users, also known as users with similar rating patterns, are used by user-based systems to assess a user's interest in it. The users who have ratings that are most closely associated to user u's have traditionally been referred to as user v's neighbors. On the other hand, item-based techniques forecast the likes of a user for an item based on u's ratings for products identical

**Advantages of Neighborhood Approach**

Collaborative filtering-based recommendation methods can be used to address some of the challenges associated with content-based methods. For example, collaborative methods can be used to make recommendations for items that do not have any associated content. This is because collaborative methods rely on the ratings and preferences of similar users to make recommendations, rather than on the characteristics of the items themselves. This means that even items that do not have any content can still be recommended if they have been rated by similar users. Another advantage of collaborative methods is that they are based on the evaluations of peers, rather than on the content of the items being recommended. This means that the recommendations made by a collaborative system are more likely to be accurate and relevant, as they are based on the ratings of other users who have similar preferences.

Finally, collaborative methods are able to recommend items with different content if other users have already rated those items. This is because collaborative methods do not rely on the characteristics of the items to make recommendations, but rather on the preferences of similar users. This can help to make more diverse and interesting recommendations, and can also help to overcome the limitations of content-based methods in cases where the content of the items is not a good indicator of their quality or relevance.

**User-based VS Item-based Recommendation**

User-based methods depend on the preferences of similar decisions of users to predict an item, whereas item-based methods try to use ratings giving to similar items (Mehta, 2009)

### 2.1.2.2 Latent Factor Approach

Matrix factorization, an example of latent factor method uses a different method by putting both items and users' vectors into the same hidden factor space. The hidden space then transforms and explains using the characteristics of items and users factor gotten from user ratings being it implicit or explicit

**Matrix Factorization**

Matrix Factorization (MF) is a common CF technique implemented by most industries for recommendation. Each user item relationship is related to a vector of hidden features. For instance, $p_u$ and $q_i$ are the latent vectors for user and item, respectively. it calculates ($y_{ui}$) as the multiplication of $p_u$ and $q_i$ shown below.

$$\hat{y}_{ui} = f(u, i|\mathbf{p}_u, \mathbf{q}_i) = \mathbf{p}_u^T \mathbf{q}_i = \sum_{k=1}^{K} p_{uk} q_{ki}$$

K denote hidden space. The two-way interaction of potential users and product factors using single direction of the hidden space does not relate to themselves and hence added linearly to the same load.

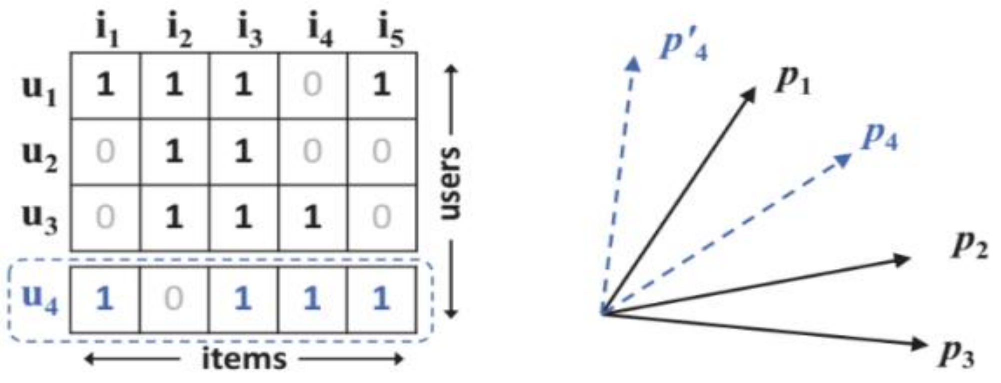


*Figure 2: Matrix factorization*

Figure 2 explains Matrix Factorization's challenges from user-item matrix, $u_4$ and $u_1$ are most related, next is $u_3$, then $u_2$. but, for user latent space, putting $p_4$ closer to $p_1$ will mean that $p_4$ will come close to $p_2$ than $p_3$, hence higher-ranking loss. Deducing from this example, it shows the negative impact generated by inner product on the performance of the model. We solve this issue by learning user item characteristic interactions with neural networks covered later in this work.

### 2.1.3 Hybrid Recommender Systems

A hybrid system that combines two technologies attempts to exploit the positives of one to solve the disadvantages of the other. An example is that, the Collaborative Filtering has problem with cold start or new item in the sense that it is difficult to recommend items without history. This however does not hinder content-based methods, as parameters to predict new products is based on readily available content description.

## 2.2 Related Works

Many literature reviews have been written regarding Recommender systems and deep learning. (Guan, Qin, Ling, & Ding, 2016) are intrigued by the use of algorithms and modernizing the conventional algorithms to enhance the issues. Single value decomposition and support vector machine are some examples of machine learning techniques that has been used in this field.

(Naomie, 2021) believed that e-commerce, entertainment and social media area some areas where recommender system have been used to solve the challenge of information overload. However, despite extensive research on learning-based recommender system, few research has been done in this area. Therefore, they implemented a general view of the theoretical foundations of recommender systems using on deep learning and neural networks. (Qi Zhang et al., 2016) also proposed a joint attention neural network which contained textual and visual information that can be recommended. In the past, explicit feedback has been the main source of data for recommendation tasks (Salakhutdinov, 2007) however implicit data is steadily gaining attention. Collaborative filtering's implicit feedback is typically regarded as a recommendation problem that concentrates on giving users recommendations for a short list of items. Recent works also proposes two strategies, where all missing data are considered to be negative. Specialized models have been presented by (He, 2008) and to account for the missing data, (Hornik, 1989) and (Bayer, 2017) implemented an implicit rating method on coordinate descent for the models depending on characteristic-based factorization, achieving the most advanced performance for item recommendation. A dual-layered Boltzmann Machine is used in the work by (Salakhutdinov, 2007) to represent the users who have explicit ratings for the objects. Autoencoders are now the most popular option for developing recommendation systems. User-based AutoRec is a study of hidden features that can rebuild a user's ratings using inputs from previous ratings (Sedhain, 2015). Denoising autoencoders have been introduced in order to learn or examine from the inputs and avoid autoencoder inability to generalize the unseen or missing data. Another neural network approach for collaborative filtering (CF) has been also proposed by (Zheng, 2016). Which has given neural networks (NN) a very strong foundation to solve the collaborative filtering problem, where the explicit ratings and solely observable data are used to model the problem. While some recent research has examined recommendations made using deep learning models that analyze implicit feedback (IF), they primarily used neural networks to implement the other data like textual description of the products (Rahman, 2020), properties of sound in music, and sometime the behavior or mouse movement on multiple platforms. These characteristics when derived using deep learning are subsequently combined with matrix factorization for personalized recommendations. Recently, a deep neural network was used by google for recommendation which used multi-layer perceptron architecture, and then eventually showing promising results while making the model generic. (Hamada and Hassan, 2016) confirmed the importance of modelling a neural network by obtaining input features to predict a user's preference for a product based on several features of the product in a multi-criteria recommendation system.

## 2.3 Chapter Summary

In summary, the chapter reviewed various theoretical and application models which have been used on the subject matter. Concept like recommender system, collaborative filtering and matrix factorization were discussed in this chapter.

# CHAPTER THREE

# 3 METHODOLOGY

In this Chapter, we dive deep into autoencoders and provide an explanation to the general framework on deep learning to examine user-item interaction. Additionally, we employ a multilayer perceptron to study the non-linear relationships on users and items. We also present a Matrix Factorization function which is a combination of the general framework and the multilayer perceptron.

## 3.1 Deep Learning and Artificial Neural Network

Deep learning is a part of machine learning algorithms that learn information representations. Neurons are multi-layered non-linear process units utilized in deep learning models, which are capable of remodeling features. These tools rely mainly on cross correlation within the frequency space (El-Bakry and Hamada, 2008). The neuron, that is commonly known as a node, is the smallest procedure unit. It computes associate degree output once receiving input from alternative neurons. every input to the node contains a weight (w) that represents its position in respect to other related inputs. The node, as portrayed in Figure 3.0, applies function f to the weighted total of inputs. The non-linear function f is known as the activation function.

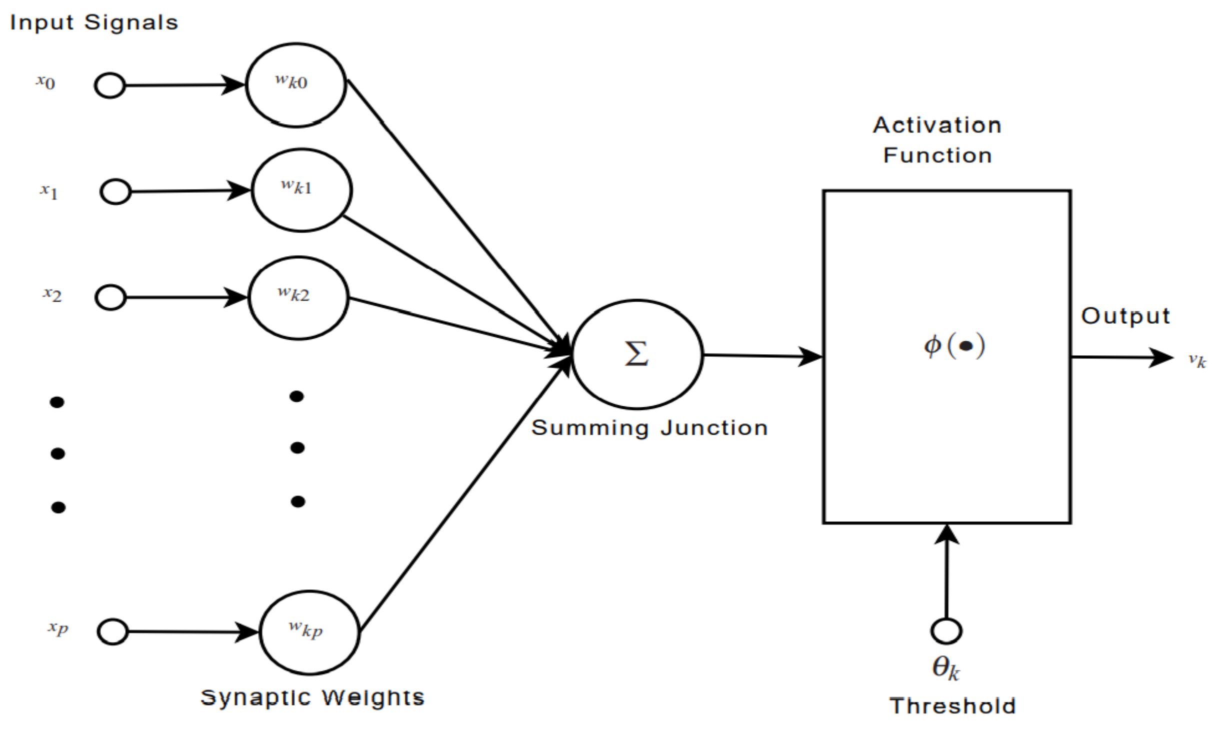


*Figure 3: Artificial Neural Network*

### 3.2 Autoencoder Based Deep Learning for Recommender System

An Autoencoder is an example of neural network that operate on two transformations namely encoder and decoder. Dimensionality reduction is the primary objective of an autoencoder in order to minimize error (Sedhain, 2015). Simply described, it is a machine learning method that employs back propagation and sets the real values to be equivalent to the input values. An autoencoder is also simple feedforward network which have an input, hidden, and output layers. In order to reconstruct its inputs, the output layer is made to have the same of neurons in terms of numbers as the input layer. With this we can describe autoencoder as an unsupervised learning algorithm, which means there are no labelled data. It is important to note that it has smaller hidden layer compared to the input layer. This method compels the model to construct a compressed representation of the data in the hidden part of the layers when learning correlations in the data. The encoding step is the operations that happens between the input and hidden layers and decoding step is the operation that happen between the hidden and output layers.

### 3.2.1 Features of Autoencoder

- Autoencoders can learn nonlinear interactions using nonlinear activation functions on multiple levels, in contrast to principal component analysis (PCA).
- When learning several layers, autoencoder are expected to be more efficient with model parameters
- Autoencoders are able translate input to output with the minimum error.

### 3.2.2 Architecture of Autoencoder

An autoencoder has several layers between its input and output layers which are smaller the input layer. It must also be noted the input and output layer's dimensionality (n) must be the same. The input is then being transformed through a layer of size P where n is greater than P. Unlabeled input are fed into an autoencoder for reconstruction. The bottleneck is the part to determine the important or necessary aspect of the observed values which should be fed forward for the next operation. It does this by following two criteria. That is the compactness of featured representation measured as the compression number of bits required to save the representation and the information the representation retains about some behavioral relevant variables. Figure 4 shows the architectural diagram of an autoencoder.

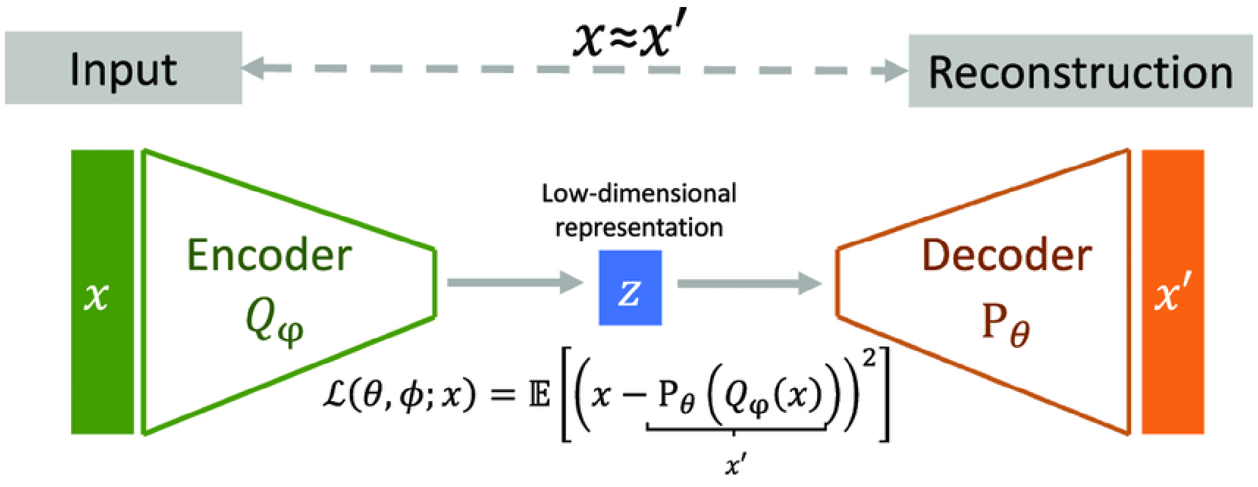

$x \approx x'$
Input
Reconstruction
$x$
Encoder
$Q_\varphi$
Low-dimensional representation
$z$
Decoder
$P_\theta$
$x'$
$\mathcal{L}(\theta, \phi; x) = \mathbb{E}\left[\left(x - \underbrace{P_\theta\left(Q_\varphi(x)\right)}_{x'}\right)^2\right]$

*Figure 4: Architecture of Autoencoder*

**Encoder**: the encoder compresses the input into a hidden space representation in a reduced dimension. The compressed data is grabbled, and does not look like the input data.

**Decoder**: This layer transforms the encoded data to the original dimension. The decoded data is lossy when reconstructed as compared to the original data.

### 3.2.3 Loss Function

The loss function determines the amount of information lost. It shows how effectively the input x and latent representation Z were reconstructed. If the data is reconstructed well, large cost can be incurred.

### 3.2.4 Flaws Associated with autoencoders

- They can only compress data which looks like what they have been trained on
- They lose data because the decompressed output is less of quality as compared to the input

### 3.2.5 Training an Autoencoder on Recommender System

**Environment**

- Hardware: Intel(R) Core (TM)i7-8550U CPU @ 1.80GHz, Installed RAM - 8.00 GB
- IDE – Visual Studio Code
- Python 3.9.12
- Libraries (Tensorflow, Pandas, Numpy)

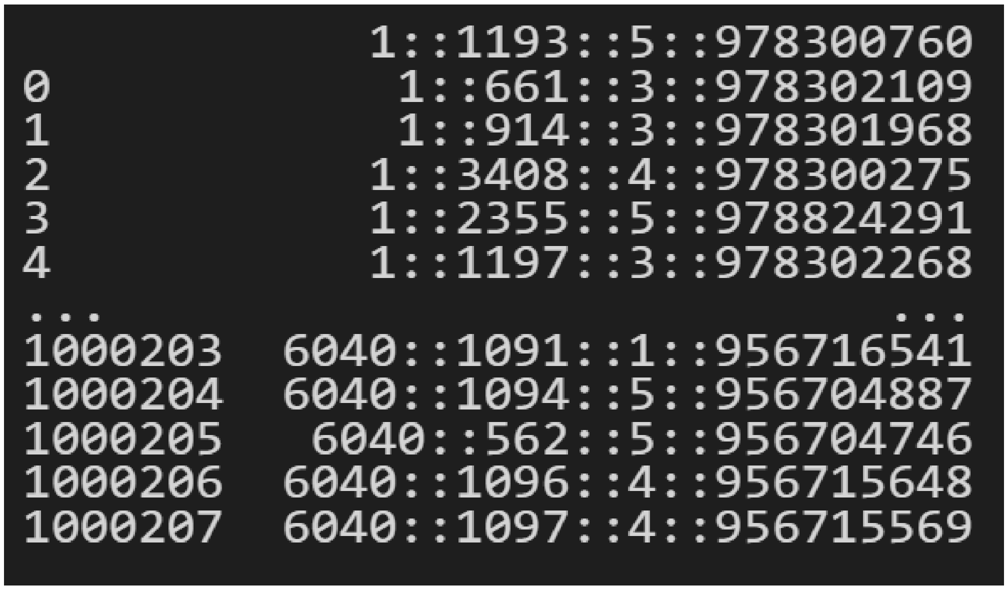
**Data**

We will use the movielens 1m data set, which contains of 1,000,209 ratings. These ratings were submitted by 6,040 members for 3,900 films. We perform data purification, and then split the data into training (75%) and testing (25%) sets, which is required for the model to be trained. We then require a user-movie matrix with a list of ratings in each row.

## Movielens Dataset

```
UserID::MovieID::Rating::Timestamp

- UserIDs range between 1 and 6040
- MovieIDs range between 1 and 3952
- Ratings are made on a 5-star scale (whole-star ratings only)
- Timestamp is represented in seconds since the epoch as returned by time(2)
- Each user has at least 20 ratings
```



```
               1::1193::5::978300760
0               1::661::3::978302109
1               1::914::3::978301968
2              1::3408::4::978300275
3              1::2355::5::978824291
4              1::1197::3::978302268
...                              ...
1000203   6040::1091::1::956716541
1000204   6040::1094::5::956704887
1000205    6040::562::5::956704746
1000206   6040::1096::4::956715648
1000207   6040::1097::4::956715569
```

*Figure 5: Sample movie lens data*

## Model Parameters

- Number of epochs = 10
- Batch Size = 16
- Learning Rate = 0.0005

- Number of hidden neurons = 128
- Number of Training Sample = 5953
- Activation Function = Sigmoid
- Optimization = Adam optimizer
- Accuracy Assessment = RMSE

**TensorFlow implementation**

The weights and biases initializers for the kernel are set in the constructor. The weights have a distribution with average and variation 0.0 and 0.02 respectively. We used three hidden layers in the network, each with 128 neurons. The number of all present movies in the dataset is shown by the input layer. Forward network output computation is done based on a sample of input data x (one row of user movie matrix). sigmoid was used as an activation function in the buried layers. Note that the last layer lacks both nonlinear and biased terms. After this step, loss and adjustment loss can be determined. The Adam optimizer minimizes the loss function. For a more accurate assessment, this method produces the root mean square error (RMSE) rather than of the original mean square error (MSE). The neural network multiplied all ratings in each user's training dataset after several stages of the training phase. By now, the model should have discovered underlying patterns in consumers' collective movie viewing preferences and data. We can now calculate the loss of root mean square error (RMSE) between the predicted and actual estimates.

## 3.3 Neural Collaborative Filtering

Neural collaborative filtering uses neural architecture instead of the internal product of the user element. In doing so, NCF uses a layered perceptron to understand user-item relation and aims to represent and generalize MF within its framework. Despite matrix factorization's usefulness for collaborative filtering, its overall performance is restrained with the aid of using easy desire of the inner product function. By including user-item bias terms in the relationship function, the performance will be enhanced (Xiangnan, 2017). This demonstrates that multiplying latent features (inner products) will not be enough to record the complexities in structure of user interaction data. This requires reconstructing a good interaction function to model the interaction of latent features between the user and the item. Neural Collaborative Filtering (NCF) aims to solve this by using the design of neural networks to model the interaction of user and item features. To learn user-item interactions, it employs the multi-layer perceptron. This is an advancement over MF since MLP is well equipped to learn user-item interaction functions due to its ability to learn any continuous function.

### 3.3.1 General Framework

We use a multi-class representation to generate the interaction between the user and the item ($y_{ui}$), where the result of a layer acts as the input of the following layer. Two input vectors vu and vi, representing user and item, are present in the initial input layer. These are hot-encoded sparse binary vectors. The integration layer which is next is fully connected and converts the sparse data to a dense space. The resulting user/item integration of the latent factor model can be regarded as a latent user/item vector. To translate latent vectors into prediction scores, these integration layers are consequently given to a multilayer neural network. To find new hidden patterns from user-item interactions, we can then alter each hidden layer. The performance of the model is determined by the size of the last hidden layer, providing the expected $y_{ui}$ score in the last layer. By reducing the loss point and its true value, Figure 6 shows a diagram of generalized neural network framework

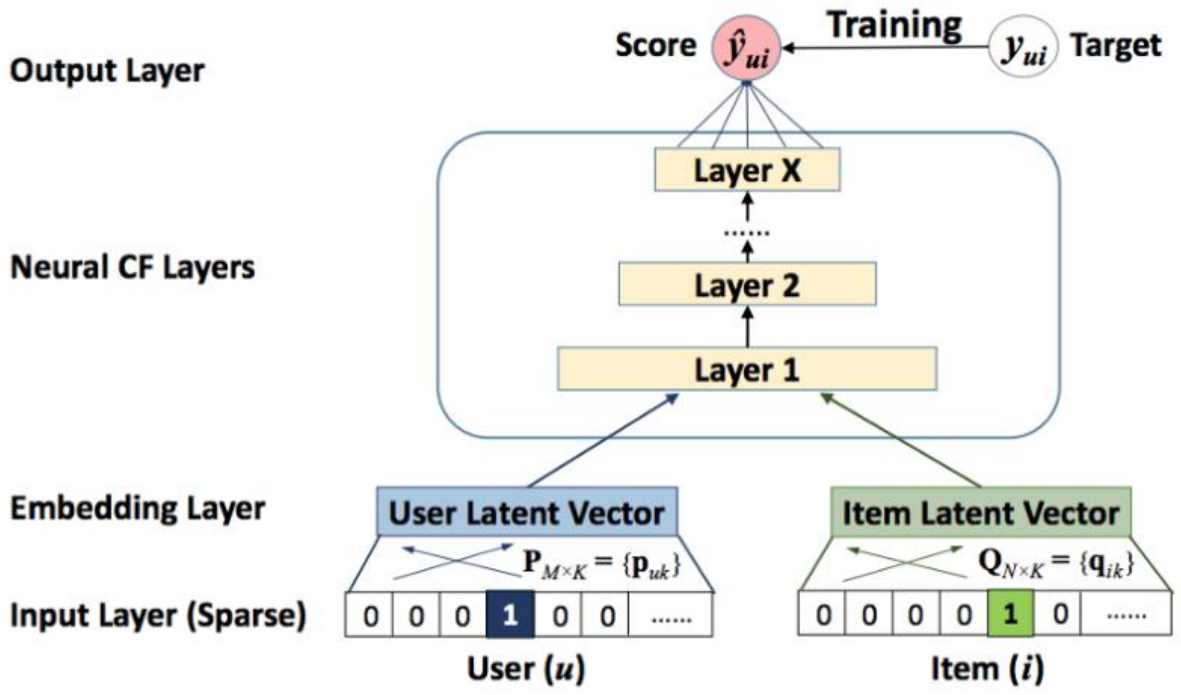


*Figure 6: Generalized Neural Network Framework*

Now we build the predictive neural network model as

$$\hat{y}_{ui} = f(\mathbf{P}^T\mathbf{v}_u^U, \mathbf{Q}^T\mathbf{v}_i^I|\mathbf{P}, \mathbf{Q}, \Theta_f)$$

where P ∈ $\Re^{MXK}$ and Q ∈ $\Re^{NXK}$, representing the matrix for users and items in latent space and $\Theta_f$ denote the interaction function’s parameters. The function $f$ which is a neural network is defined as

$$f(\mathbf{P}^T\mathbf{v}_u^U, \mathbf{Q}^T\mathbf{v}_i^I) = \phi_{out}(\phi_X(...\phi_2(\phi_1(\mathbf{P}^T\mathbf{v}_u^U, \mathbf{Q}^T\mathbf{v}_i^I))...))$$

where $\varphi_{out}$ and $\varphi_X$ denote the function for the layers in the output

**Model Parameters**

Also, this method performs specific regression duties with squared loss when learning model parameters as

$$L_{sqr} = \sum_{(u,i)\in\mathcal{Y}\cup\mathcal{Y}^-} w_{ui}(y_{ui} - \hat{y}_{ui})^2$$

where $Y$ denotes actual data in Y, and $Y-$ denote unobserved data. The squared loss fails to work well on binary data but performs better when drawn from Gaussian distribution. So, to study parameters on binary data, probabilistic function is implemented as the activation function for the layer in the output as $\varphi_{out}$. We define the function as

$$p(\mathcal{Y}, \mathcal{Y}^-|\mathbf{P}, \mathbf{Q}, \Theta_f) = \prod_{(u,i)\in\mathcal{Y}} \hat{y}_{ui} \prod_{(u,j)\in\mathcal{Y}^-} (1 - \hat{y}_{uj})$$

We then get the formula below when the function's negative logarithm is taken

$$\begin{aligned} L &= -\sum_{(u,i)\in\mathcal{Y}} \log \hat{y}_{ui} - \sum_{(u,j)\in\mathcal{Y}^-} \log(1 - \hat{y}_{uj}) \\ &= -\sum_{(u,i)\in\mathcal{Y}\cup\mathcal{Y}^-} y_{ui} \log \hat{y}_{ui} + (1 - y_{ui}) \log(1 - \hat{y}_{ui}) \end{aligned}$$

This is the cross-entropy loss of the binary term or the log loss. The optimization of this function is generalised by stochastic gradient descent.

### 3.3.2 Generalized Matrix Factorization (GMF)

One-hot encodings of user/item vectors serve as the input to the model, and subsequent embedding layers can be seen as potential user/item vectors. Let's the user and item vectors be $p_u$ and $q_i$ respectively. The function to the first layer is then defined as:

$$\varphi_{out}(p_u , q_i ) = p_u \odot q_i$$

where $\odot$ represent the multiplication of vectors. The output layer is then project by the vector as:

$$y_{ui} = a_{out}(h^{T} (p_u \odot q_i ))$$

where $a_{out}$ and $h^T$ stand for the edge weights and activation function, respectively. We used the sigmoid function as the activation function in our generalized matrix factorization implementation, which studies parameters with the objective function of the log loss.

### 3.3.3 Multi-Layer Perceptron (MLP)

Two paths are used by neural collaborative filtering method to model users and items. Concatenating these approaches makes sense in order to create a powerful deep learning-based recommender system. However, the relationships between user and item latent characteristics cannot be fully captured by a straightforward vector concatenation. To solve this problem, we concatenated the vector with hidden layers and utilized MLP to understand how the user and item vectors interacted. We state the model as follows:

$$y_{ui} = \sigma(h^{T} \varphi L(z_{L-1}))$$

### 3.3.4 Neural Matrix Factorization (Fusion of GMF and MLP)

For now, we have examined two neural network methods that learn interaction function from data: GMF employs a linear interactive method while MLP uses a non-linear interactive method. We now introduce a hybrid model that combines GMF and MLP in order to learn the intricate interactions through mutual reinforcement.

.

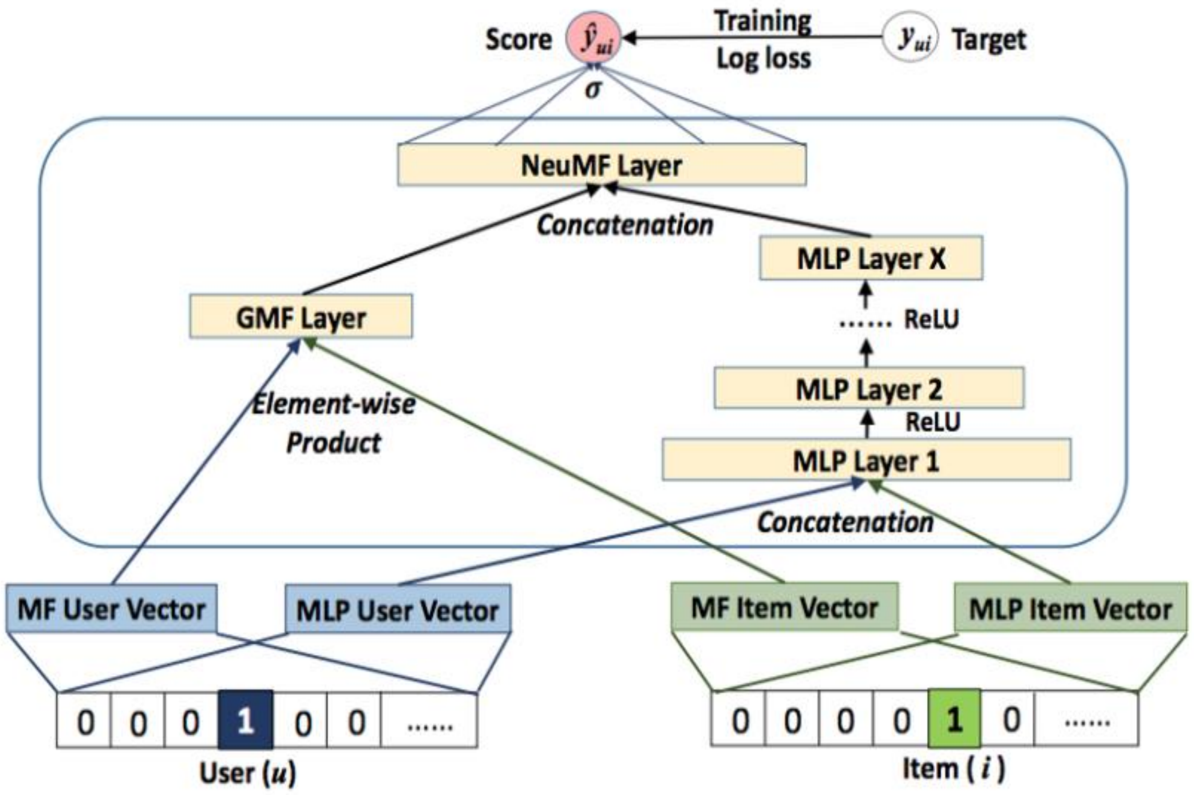


*Figure 7: Neural Matrix Factorization*

Sharing the same layer of embedding for both GMF and MLP and combining the outputs of their interaction functions is an obvious approach for fusing these models. However, the performance and flexibility of the fused model may be constrained by combined embeddings of GMF and MLP. In order to merge these models, we concatenated the final hidden layers of the GMF and MLP models, as illustrated in Figure 6. This model can be stated as follows:

$$y_{ui} = \sigma(\mathrm{h}^T (\varphi GMF_{out} . \varphi MLP_{out} ))$$

## 3.4 Chapter Summary

In Summary the Chapter out listed the tools used for the project and made a comprehensive description of the implementation of Autoencoder and Neural Collaborative Filtering Deep Learning models.

# CHAPTER FOUR

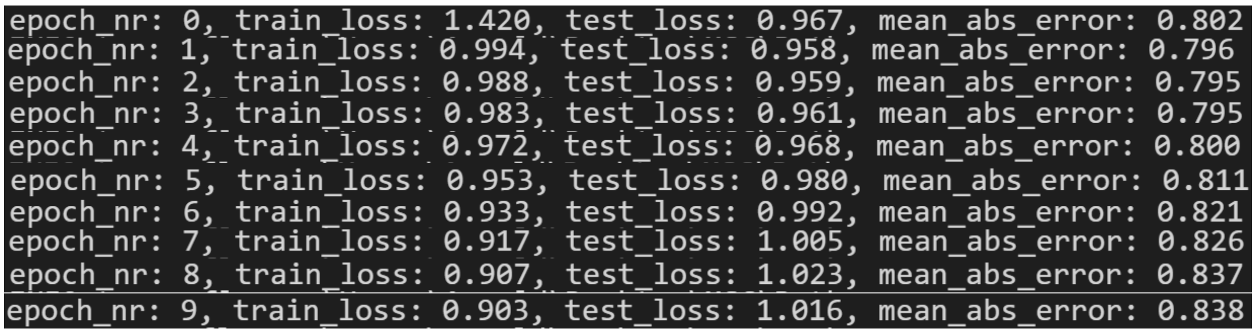
# 4 FINDINGS AND OBSERVATIONS

In this Chapter, we evaluate the result and performance of both Autoencoder and neural collaborative filtering, taking into consideration their loss functions and root mean squared errors.

## 4.1 Results

### Autoencoder

For Autoencoder, Sigmoid is used as an activation function in the buried layers. The Adam Optimizer also minimizes the loss function. For greater accuracy assessment, the method outputs a root mean squared error (RMSE) rather than a mean squared error (MSE). At various epochs, corresponding test loss, train loss and RMSE were obtained. Figure 7 shows the output of autoencoder for 10 epochs, figure 8 shows the trends in accuracy for 10 epochs and figure 9 shows the trends in train loss and deviations from actual data on tensorboard.



```
epoch_nr: 0, train_loss: 1.420, test_loss: 0.967, mean_abs_error: 0.802
epoch_nr: 1, train_loss: 0.994, test_loss: 0.958, mean_abs_error: 0.796
epoch_nr: 2, train_loss: 0.988, test_loss: 0.959, mean_abs_error: 0.795
epoch_nr: 3, train_loss: 0.983, test_loss: 0.961, mean_abs_error: 0.795
epoch_nr: 4, train_loss: 0.972, test_loss: 0.968, mean_abs_error: 0.800
epoch_nr: 5, train_loss: 0.953, test_loss: 0.980, mean_abs_error: 0.811
epoch_nr: 6, train_loss: 0.933, test_loss: 0.992, mean_abs_error: 0.821
epoch_nr: 7, train_loss: 0.917, test_loss: 1.005, mean_abs_error: 0.826
epoch_nr: 8, train_loss: 0.907, test_loss: 1.023, mean_abs_error: 0.837
epoch_nr: 9, train_loss: 0.903, test_loss: 1.016, mean_abs_error: 0.838
```

*Figure 8: Output of Autoencoder for 10 epochs*

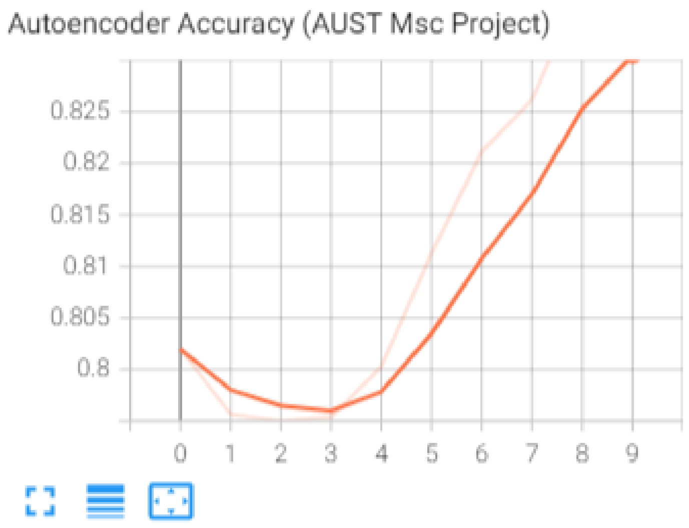


*Figure 8: Accuracy trends for 10 epochs*

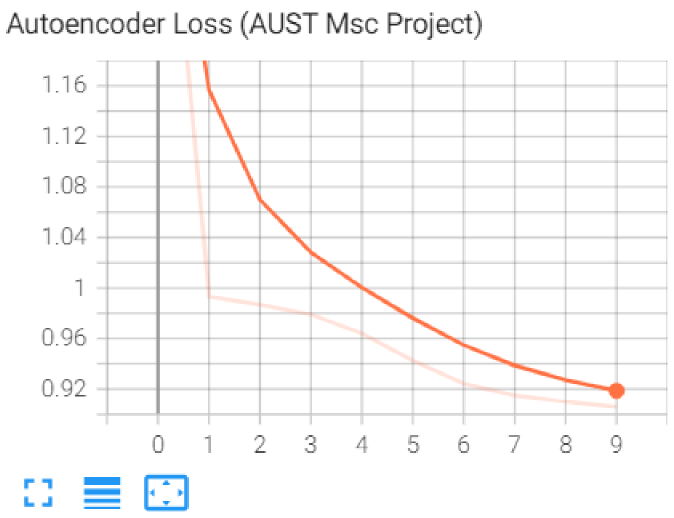
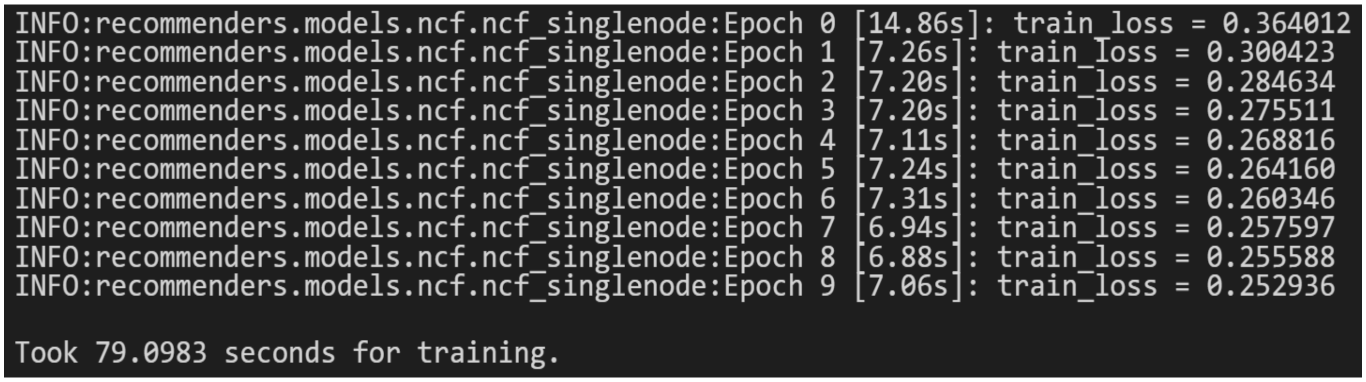


*Figure 9: Train loss trends for 10 epochs*

**Neural Collaborative Filtering**

The fusion of Linear GMF and non-linear MLP keeps NeuMF at an advantageous position to learn the more iteratively with it layers than autoencoders. Here also, at various epochs it was observed that, the train loss was lesser than the autoencoder model and traditional matrix factorization. Figure 10 shows the output of neural collaborative filtering and figure 11 shows the trend in train losses.



```
INFO:recommenders.models.ncf.ncf_singlenode:Epoch 0 [14.86s]: train_loss = 0.364012
INFO:recommenders.models.ncf.ncf_singlenode:Epoch 1 [7.26s]: train_loss = 0.300423
INFO:recommenders.models.ncf.ncf_singlenode:Epoch 2 [7.20s]: train_loss = 0.284634
INFO:recommenders.models.ncf.ncf_singlenode:Epoch 3 [7.20s]: train_loss = 0.275511
INFO:recommenders.models.ncf.ncf_singlenode:Epoch 4 [7.11s]: train_loss = 0.268816
INFO:recommenders.models.ncf.ncf_singlenode:Epoch 5 [7.24s]: train_loss = 0.264160
INFO:recommenders.models.ncf.ncf_singlenode:Epoch 6 [7.31s]: train_loss = 0.260346
INFO:recommenders.models.ncf.ncf_singlenode:Epoch 7 [6.94s]: train_loss = 0.257597
INFO:recommenders.models.ncf.ncf_singlenode:Epoch 8 [6.88s]: train_loss = 0.255588
INFO:recommenders.models.ncf.ncf_singlenode:Epoch 9 [7.06s]: train_loss = 0.252936

Took 79.0983 seconds for training.
```

*Figure 9: Output of Neural Collaborative Filtering*

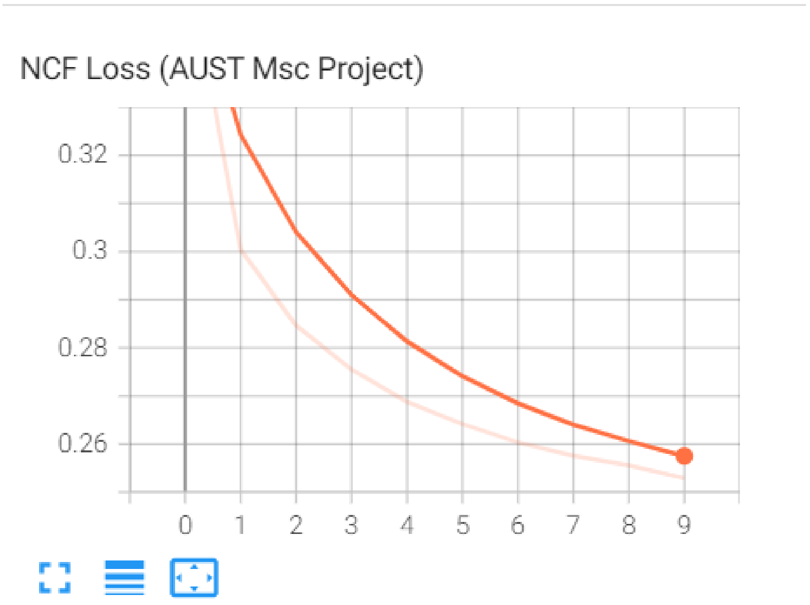
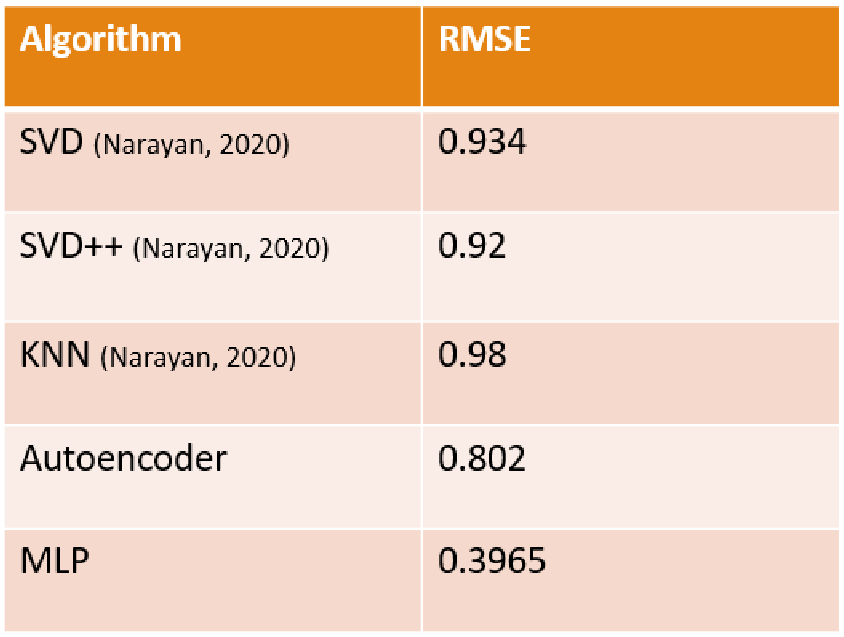


*Figure 10: Trends in train losses for NCF*

**State-of-the-Art Algorithm Comparison**



| Algorithm | RMSE |
|---|---|
| SVD (Narayan, 2020) | 0.934 |
| SVD++ (Narayan, 2020) | 0.92 |
| KNN (Narayan, 2020) | 0.98 |
| Autoencoder | 0.802 |
| MLP | 0.3965 |

*Figure 11: State-of-the-art algorithm comparison*

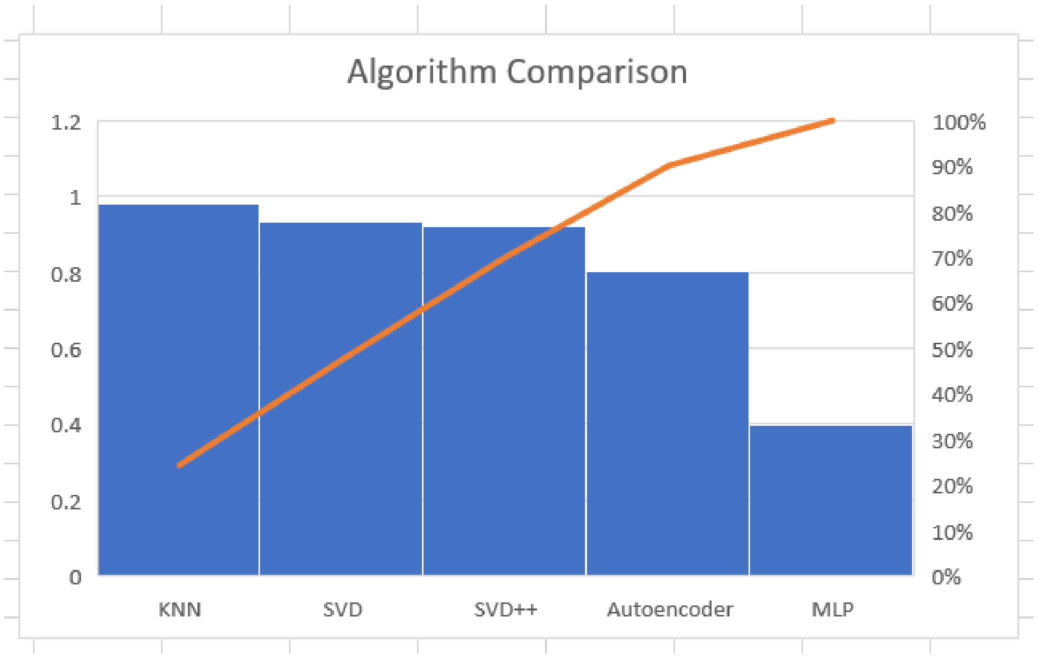


*Figure 12: Algorithm Comparison Chart*

## 4.2 Chapter Summary

To conclude, the models and findings explained in this chapter demonstrated how deep learning can be used to personalize and improve recommendation systems. The research compared the output of the Autoencoder and Neural Collaborative Filtering models on same dataset which showed a better result for NCF than Autoencoders. It is also discussed that, the compression of autoencoder degrades data which makes it lossy hence the lager values in train loss compared to NeuMF.

# CHAPTER FIVE

# 5 CONCLUSION AND FUTURE WORK

## 5.1 Conclusion

We employed various neural network topologies in this effort to get over the drawbacks of matrix factorization collaborative filtering techniques. We demonstrated that the models outperformed existing state-of-the-art models. Again, we contrasted the accuracy of the NCF deep learning model and the autoencoder model. Our research revealed that NCF performed better than autoencoders in terms of accuracy, even though both are more effective than traditional machine learning models for collaborative filtering. The models are simple and all-encompassing, and they can be used to a variety of circumstances including recommendations or improved upon. For autoencoders, the idea is to learn an predictions to the function by enforcing certain constraints, such as adding regularization, denoising, sparsity, contractive, etc. By doing so, its output is similar to input is generated, information may be effectively compressed and interesting structure about the data may be discovered.

## 5.2 Future Work

This work improves the commonly employed shallow models for collaborative filtering and opens up new avenues for deep learning-based recommendation research. Future research may examine variables and ways to reduce output losses, though. Additionally, we wish to conduct research on personalization algorithms that focus on user groups rather than specific people. These models will be useful for making recommendations to social groups.

## APPENDIX A: NUERAL COLLABORATIVE FILTERING CODE

The following codes shows the implementation of Neural Collaborative Filtering in python.

Libraries used were tensorflow and numpy on movielens one million dataset

# In the following code, we import libraries and other classes

```
import sys

import pandas as pd

import tensorflow as tf

from recommendersystem.utility.time import Timer

from recommendersystem.rs_models.ncf.ncf_single import NCF

from recommendersystem.rs_models.ncf.rs_dataset import Dataset as
NCFDataset

from recommendersystem.rs_datasets import movielens

from recommendersystem.utililty.notebook_utility import is_jupyter

from recommendersystem.rs_datasets.python_splitters import
python_chrono_split

from recommendersystem.rs_evaluation.rs_python_evaluation import
(rmse, mae, rsquared, exp_var)
```

# In the block of code below, we set our movielens data size and define the model parameters

```
MOVIELENS_DATA_SIZE = '1m'

EPOCHS_RS = 50

BATCH_SIZE_RS = 256

SEED_RS = 42
```

```
df = movielensdata.load_pandas_df(

size=MovielensData_Size,

header=["userID", "itemID", "rating", "timestamp"]

)
```

# In the block of code below, we split the data into train and test set at 75% and 25% respectively

```
trainSet, testSet = python_chrono_split(df, 0.75)

testSet =
testSet[testSet["userID"].isin(trainSet["userID"].unique())]

testSet =
testSet[testSet["itemID"].isin(trainSet["itemID"].unique())]

train_file_csv = "./train.csv"

test_file_csv = "./test.csv"

trainSet.to_csv(train_file_csv, index=False)

testSet.to_csv(test_file_csv, index=False)
```

# We then seed the training and testing data into respective csv files

```
data = NCFData(train_file_csv=train_file_csv,
test_file_csv=test_file_csv, seed=SEED)
```

# This code shows the implementation of NCF Class with couple of parameters set to it.

```
model = NCF (

num_of_users=data.num_users,

num_of_items=data.num_items,
model_type="NeuMF",

factors_used=4,

layer =[16,8,4],

num_of_epochs= EPOCHS_RS,

batch_size=BATCH_SIZE_RS,

lr=1e-3,

seed_num=SEED_RS

)
```

## APPENDIX B: AUTOENCODER CODE

# In the following code, we import libraries and other classes

```
import numpy as np

#import tensorflow as tf

#import tensorflow.compat.v1 as tf

import tensorflow._api.v2.compat.v1 as tf

import os

from data.dataset import _get_training_data, _get_test_data

from model.train_model import TrainModel

from sklearn.metrics import mean_absolute_error, mean_squared_error
```

# In the block of code below, we define the model parameters

```
tf.app.flags.DEFINE_string('tf_records_train_path',
os.path.abspath(os.path.join(os.path.dirname("__file__"), '..',
'auto/src/data/mm_records/strain/')),'Path of the training data.')

tf.app.flags.DEFINE_string('tf_records_test_path',
os.path.abspath(os.path.join(os.path.dirname("__file__"), '..',
'auto/src/data/mm_records/stest/')),'Path of the test data.')

tf.app.flags.DEFINE_string('checkpoints_path',
os.path.abspath(os.path.join(os.path.dirname( __file__ ), '..',
'checkpoints/model.ckpt')), 'Path for the test data.')

tf.app.flags.DEFINE_integer('num_epoch', 1000, 'Number of training
epochs.')

tf.app.flags.DEFINE_integer('batch_size', 16,'Size of the training
batch.')

tf.app.flags.DEFINE_float('learning_rate',0.0005, 'Learning_Rate')

tf.app.flags.DEFINE_boolean('l2_reg', False, 'L2 regularization.')

tf.app.flags.DEFINE_float('lambda_',0.01, 'Wight decay factor.')

tf.app.flags.DEFINE_integer('num_v', 3952, 'Number of visible
neurons (Number of movies the users rated.)')

tf.app.flags.DEFINE_integer('num_h', 128, 'Number of hidden
neurons.)')

tf.app.flags.DEFINE_integer('num_samples', 5953, 'Number of
training samples (Number of users, who gave a rating).')

FLAGS = tf.app.flags.FLAGS


# Building the graph, opening of a session and starting the training of the neural network.
```

```
def main(_):

    num_batches=int(FLAGS.num_samples/FLAGS.batch_size)

    with tf.Graph().as_default():

        train_data, train_data_infer=_get_training_data(FLAGS)

        test_data=_get_test_data(FLAGS)

        iter_train = train_data.make_initializable_iterator()

   iter_train_infer=train_data_infer.make_initializable_iterator()

        iter_test=test_data.make_initializable_iterator()

        x_train= iter_train.get_next()

        x_train_infer=iter_train_infer.get_next()

        x_test=iter_test.get_next()

        model=TrainModel(FLAGS, 'training')

        train_op, train_loss_op=model.train(x_train)

prediction,labels,test_loss_op,mae_ops=model._validation_loss(x_tr
ain_infer, x_test)

        saver=tf.train.Saver()

        with tf.Session() as sess:

            sess.run(tf.global_variables_initializer())

            train_loss=0

            test_loss=[]

            mae=[]

            for epoch in range(FLAGS.num_epoch)

                sess.run(iter_train.initializer)
                sess.run(iter_train_infer.initializer)

                sess.run(iter_test.initializer)

                for batch_nr in range(num_batches):

                    _, loss_=sess.run((train_op, train_loss_op))

                    train_loss+=loss_

                for i in range(FLAGS.num_samples):


                    pred,labels_,loss_,mae_=sess.run((prediction,l
                    abels, test_loss_op,mae_ops))



                    test_loss.append(loss_)

                    mae.append(mae_)

                print('epoch_nr: %i, train_loss: %.3f, test_loss:
%.3f, mean_abs_error: %.3f'


%(epoch,(train_loss/num_batches),np.mean(test_loss),
np.mean(mae)))

                if np.mean(mae)<0.9:

                    saver.save(sess, FLAGS.checkpoints_path)

                train_loss=0

                test_loss=[]

                mae=[]

if __name__ == "__main__":

    tf.app.run()
```

## REFERENCES

Aminu, D., Naomie, S. (2021). Recommendation System Based on Deep Learning Methods: A Systematic Review and New Directions. *SpringerLink. https://link.springer.com/article/10.1007%2Fs10462-019-09744-1*.

Bayer, I. H. (2017). A Generic Coordinate Descent framework for Learning from Implicit FeedBack. *WWW*.

El-Bakry, H., Hamada, M. (2008). A New Implementation for High Speed Normalized Neural Networks in Frequency Space. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5177 LNAI(PART 1), pp. 33–40.

El-Bakry, H., Hamada, M. (2008). New fast decision tree classifier for identifying protein coding regions. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5370 LNCS, pp. 489–500.

Guan, C., Qin, S., Ling, W., Ding, G. (2016). Apparel Recommendation System Evolution: An empirical review. *International Journal of Clothing Science and Technology*, 854–879.

Hamada, M., Hassan, M. (2017). An Enhanced Learning Style index: Implementation and Integration into an Intelligent and Adaptive E-Learning System. *Eurasia Journal of Mathematics, Science and Technology Education*, 13(8), pp. 4449–4470.

Hamada, M., Hassan, M. (2011). A Game-based Learning System for Theory of Computation Using Lego NXT Robot. *Procedia Computer Science*, 4, pp. 1944–1952.

Hassan, M., Hassan, M. (2016). Performance Comparison of Featured Neural Network Trained with Backpropagation and Delta Rule Techniques for Movie Rating Prediction in Multi-Criteria Recommender Systems. *Informatica (Slovenia)*, 40(4), pp. 409–414.

Hassan, M., Hamada, M. (2017). Performance Comparison of Feed-forward Neural Networks Trained with Different Learning Algorithms for Recommender Systems. *Computation*, pp 5(3), 40.

He, X. Z. (2008). Fast Matrix Factorization for Online and Implicit Feedback. *SGIR*, 549-558.

Hornik, M. S. (1989). Multilayer FeedForward Networks anre Universal Approcimators. *Neural Networks,*, Vol. 5.

Mehta, B. N. (2009). Unsupervised Strategies for Shilling Detection and Robust Collaborative Filtering. *User Modeling and User-Adaptive Interaction* , 65-97.

Oppermann, A. (2015). *https://towardsdatascience.com/deep-autoencoders-for-collaborative-filtering-6cf8d25bbf1d*.

Rahman, M. A., Hamada, M. (2020). Burrows–wheeler Tansform Based Lossless Text Compression Using Keys and Huffman Coding. *Symmetry*, 12(10), pp. 1–14, 1654.

Ricci B, R. L. (2011). Recommender System Handbook. *Library of Congress*.

Salakhutdinov, R. M. (2007). Restricted Boltzman Machines for Collaborative Filtering. *ICDM*, 791-798.

Sarwar, B. K. (2011). Item-based Collaborative Filtering. *In Proceedings of the Tenth International World Wide Web*.

Sawar, B. G. (2001). Item Based Collaborative Recommendation Algorithm. *In proceedings of the tenth International World Wide Web Conference*, 285-295.

Sedhain, S. M. (2015). Autoencoders Meet Collaborative Filtering. *WWW*, 111-11.

Xiangnan, H. L. (2017). Neural Collaborative Filtering. *In Proceedings of the 26th International Conference on World Wide Web*, 173-182.

Zheng, Y. B. (2016). A Neural Autoregressive Approach to Collaborative Filtering. *ICML*.

Sarwar, B., Karypis, G., Konstan, J., Riedl, J. (2001). "Item–based Collaborative Filtering Recommendation Algorithms". *In Proceedings of the Tenth International World Wide Web Conference* pp. 285–295

Ouyang Y, Liu W, Rong W, et al. (2014) Autoencoder Based Collaborative Filtering. *In International Conference on Neural Information Processing*., vol 8836, pp 284-291. Springer, Cham

Hornik, K., Stinchcombe, M., and White, H. (1989). "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, vol. 5.

Koren, Y. (2008) "Factorization Meets the neighborhood: A Multifaceted Collaborative Filtering Model." *in KDD*, pp. 426–434.

He, L., Liao L, Zhang, He., Nie, H., Hu, H., and Chua, T. (2017) "Neural Collaborative Filtering." In Proceedings of the 26th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee,, pp. 173–182.

Yin, Z., Cailiang, L., Bangsheng, Tang., and Hanning, Z. (2016). Neural Autoregressive Collaborative Filtering for Implicit Feedback. In Recsys.

Yin, Z., Bangsheng, T., Wenkui, D., and Hanning, Z. (2016). A Neural Autoregressive Approach to Collaborative Filtering. In ICML.

Chang, Z., Jinze, B., Junshuai S., Xiaofei L., Zhengchao, Z., Xiusi, C., and Jun Gao. (2017). ATRank: An AŠention-Based User Behavior Modeling Framework for Recommendation. *arXiv preprint arXiv:1711.06632*.

Jiang, Z., Catha, l G., and Rami, A. (2016). Applying Visual User Interest Profiles for Recommendation and Personalization

Fuzhen, Z., Zhiqiang, Z., Mingda, Q., Chuan, S., Xing, X. and Qing, H. (2017). Representation Learning via Dual-Autoencoder for Recommendation. *Neural Networks* 90 (2017), 83–89.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. *In Proceedings of the 26th International Conference on World Wide Web*, 173–182

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems. 1097 1105.

Wonsung, L., Kyungwoo, S., and Il-Chul, M. (2017). Augmented Variational Autoencoders for Collaborative Filtering with Auxiliary Information. *In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 1139–1148.

Liang, D., Krishnan, R., Hoffman, M., Tony, J. 2018. Variational Autoencoders for Collaborative Filtering. *arXiv preprint arXiv:1802.05814*

Zhang, F., Yuan, N. J., Lian, D., Xie, X. (2001). Collaborative Knowledge Based Embedding for Recommender Systems. *In KDD*, pages 353–362, 201

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. “Item-based Collaborative Filtering Recommendation Algorithms,” *in WWW*, 2015, pp. 1661– 1670.

Wang, Y., Wang, M., Xu, W. (2018) Sentiment-enhanced Hybrid Recommender System for Movie Recommendation: A Big Data Analytics Framework, *Wireless Communications and Mobile Computing*.

Sedhain, S., Menon, A., Xie, S. (2015) Autorec: Autoencoders Meet Collaborative Filtering. *In Proceedings of the 24th International Conference on World Wide Web*. ACM, 111–112.

Strub, F., Mary, J. (2015). Collaborative Filtering with Stacked Denoising AutoEncoders and Sparse Inputs. In IPS Workshop on Machine Learning for ECommerce, *https://hal.inria.fr/hal-01256422v1*

Ouyang, Y., Liu, W., Rong, W., et al. (2014) Autoencoder Based Collaborative Filtering. *International Conference on Neural Information Processing*

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization Techniques for Recommender Systems. Computer 42, 8 (2009).

Andriy, M., Ruslan, A., Salakhutdinov, R. (2008). Probabilistic Matrix Factorization. *In Advances in Neural Information Processing Systems*. 1257–1264.