INTERNSHIP REPORT

on

# RECOMMENDER SYSTEMS USING COLLABORATIVE FILTERING

(Neural vs. Dot Product Approach for Matrix Factorisation)


under

Dr. Kisor Kumar Sahu and Dr. Manoranjan Satpathy

Indian Institute of Technology Bhubaneswar

by

DASARI ASHISH (17CS01038)

*B. Tech, Semester VII, Computer Science and Engineering,*

*Indian Institute of Technology Bhubaneswar*

# TABLE OF CONTENTS



**Project Repository: https://github.com/ashishdasari148/Recommender-Systems-using-Collaborative-Filtering
**Google Colaboratory: https://colab.research.google.com/github/ashishdasari148/Recommender-Systems-using-Collaborative-Filtering/blob/master/Netflix.ipynb

# ABSTRACT

One of the potent personalization technologies powering the adaptive web is collaborative filtering. Collaborative filtering (CF) is the process of filtering or evaluating items through the opinions of other people. CF technology brings together the opinions of large interconnected communities on the web, supporting filtering of substantial quantities of data.

This internship report also demonstrates the core concepts of collaborative filtering, its primary uses for users of the adaptive web, the theory and practice of CF algorithms, and design decisions regarding rating systems and acquisition of ratings.

Embedding based models have been the state of the art in collaborative filtering for over a decade. Traditionally, the dot product or higher-order equivalents have been used to combine two or more embeddings, e.g., most notably in matrix factorization. In recent years, it was suggested to replace the dot product with a learned similarity, e.g., using a multi-layer perceptron (MLP). This approach is often referred to as neural collaborative filtering (NCF). In this internship report, we revisit the experiments of the NCF paper that popularized learned similarities using MLPs. First, we show that with a proper hyperparameter selection, a simple dot product substantially outperforms the proposed learned similarities. Second, while an MLP can, in theory, approximate any function, we show that it is non-trivial to learn a dot product with an MLP. Finally, we discuss practical issues that arise when applying MLP based similarities and show that MLPs are too costly to use for item recommendation in production environments while dot products allow applying very efficient retrieval algorithms. We conclude that MLPs should be used with care as embedding combiner and that dot products might be a better default choice.

# Chapter 1: INTRODUCTION

## 1.1 Intuition

Collaborative filtering is the process of filtering or evaluating items using the opinions of other people. While the term collaborative filtering (CF) has only been around for a little more than a decade, CF takes its roots from something humans have been doing for centuries - sharing opinions with others. For years, people have stood over the back fence or in the office break room and discussed books they have read, restaurants they have tried, and movies they have seen – then used these discussions to form opinions. For example, when enough of Manasa's colleagues say they liked the latest release from Hollywood, she might decide that she also should see it. Similarly, if many of them found it a disaster, she might decide to spend her money elsewhere. Better yet, Manasa might observe that Ashish recommends the types of films that she finds enjoyable, Tarun has a history of recommending movies that she despises, and Harshi seems to suggest everything. Over time, she learns whose opinions she should listen to and how these opinions can be applied to help her determine the quality of an item. Computers and the web allow us to advance beyond simple word-of-mouth. Instead of limiting ourselves to tens or hundreds of individuals, the Internet will enable us to consider the opinions of thousands. The speed of computers allows us to process these opinions in real-time and determine not only what a much larger community thinks of an item, but also develop a truly personalized view of that item using the opinions most appropriate for a given user or group of users.

## 1.2 Why Collaborative Filtering

Collaborative filtering systems produce predictions or recommendations for a given user and one or more items. Items can consist of anything for which a human can provide a rating, such as art, books, CDs, journal articles, or vacation destinations. There are also broad, abstract families of tasks that CF systems support. Ideally, the system would support all user tasks, although mapping a real application to the functionality of an actual CF system can be challenging. Here are the broad families of standard CF system functionality:

- **Recommend items**: Show a list of items to a user, in order of how useful they might be. Often this is described as predicting what the user would rate the item, then ranking the items by this predicted rating. However, some successful recommendation algorithms do not compute predicted rating values at all. For example, Amazon's recommendation algorithm aggregates items similar to a user's purchases and ratings without ever computing a predicted rating. Instead of displaying a personalized predicted rating, their user interface displays the average customer rating. As a result, the recommendation list may appear out of order with respect to the displayed average rating value. In many applications, picking the top few items well is crucial; producing predicted values is secondary.

- **Predict for a given item:** Given a particular item, calculate its predicted rating. Note that prediction can be more demanding than the recommendation. To recommend items, a system only needs to be prepared to offer a few alternatives, but not all. Some algorithms take advantage of this to be more scalable by saving memory and computation time. To provide predictions for a particular item, a system must be prepared to say something about any requested item, even rarely rated ones. How does a system decide how a specific user would rate a requested item if very few users – let alone users similar to the particular user – have rated the item? Personalized predictions may be challenging, if not impossible.
- **Constrained recommendations**: Recommend from a set of items. Given a particular set or a constraint that gives a set of items, recommend from within that set. For example: "Consider the following scenario. Mary's 8-year-old nephew is visiting for the weekend, and she would like to take him to the movies. She would like a comedy or family movie rated no "higher" than PG-13. She would prefer that the movie contain no sex, violence, or offensive language, last less than two hours, and, if possible, show at a theatre in her neighborhood. Finally, she would like to select a movie that she might enjoy." It can be accomplished by a "meta-recommendation system" that generates recommendations from a blending of multiple recommendation sources. Users define preferences and requirements through a web form that restricts the set of potential candidate items. Recommendations are based on a ranking of how well the items within this set match the provided preferences.

## 1.3 Rating Matrix

The rating matrix consists of a table where each row represents a user, each column represents a specific movie, and the number at the intersection of a row and a column represents the user's rating value. The absence of a rating score at this intersection indicates that the user has not yet rated the item.

| | **Manasa (1)** | **Tarun (2)** | **Ashish (3)** | **Harshi (4)** |
|---|---|---|---|---|
| **Love at last (1)** | 5 | 5 | 1 | 1 |
| **Romance forever (2)** | 5 | ? | ? | 1 |
| **Cute puppies of love (3)** | ? | 4 | 1 | ? |
| **Nonstop car chases (4)** | 1 | 1 | 5 | 4 |
| **Swords vs. karate (5)** | 1 | 1 | 5 | ? |

*Table 1.3.1: Rating Matrix*

Ratings in a collaborative filtering system can take on a variety of forms.

- Scalar ratings can consist of either numerical ratings, such as the 1-5 stars provided in the Netflix dataset, or ordinal ratings such as strongly agree, agree, neutral, disagree, strongly disagree.
- Binary rating model choices between agree/disagree or good/bad.
- Unary ratings can indicate that a user has observed or purchased an item or otherwise rated the item positively. The absence of a rating means that we have no information relating the user to the item (perhaps they purchased the item somewhere else).

Ratings may be gathered through explicit means, implicit means, or both. Explicit ratings are those where a user is asked to provide an opinion on an item. Implicit ratings are those inferred from a user's actions. For example, a user who visits a product page perhaps has some interest in that product while a user who subsequently purchases the product may have a much stronger interest in that product.

# Chapter 2: Dot Product Approach

## 2.1 Intuition

Collaborative filtering uses the assumption that people with similar tastes will rate things similarly. Assume that every user is placed in a $d$-dimensional space based on his/her preferences and tastes. We have a similar notion for the movies based on the content of the film. Their feature vectors, respectively, define the positions of users and movies in $d$-dimensional space. The idea is, if a particular film ($i$) aligns with a user ($j$)'s tastes, then the feature vectors of the movie and the user are similar. In other words, the rating given by the user to a particular film will be proportional to the similarity of their feature vectors. These feature vectors are also called Embeddings.

## 2.2 Content-Based Filtering

Content-based filtering uses the assumption that items with similar objective features will be rated similarly.

Let's suppose that each of these movies (Table 1.3.1) has a set of features for them. In particular, let's say that each of the movies has two features denoted by $x_1$ and $x_2$ (here $x_0$ is always 1). Where $x_1$ measures the degree to which a movie is a romantic movie, and $x_2$ measures the degree to which a movie is an action movie. If we take the movie love at last, say you know it's 0.9 rating on the romance scale. This is a highly romantic movie but zero on the action scale. Similarly, let us say we have all the movies labeled for $x_1$ and $x_2$.

| | $x_1$ | $x_2$ |
|---|---|---|
| **Love at last (1)** | 0.90 | 0.00 |
| **Romance forever (2)** | 1.00 | 0.01 |
| **Cute puppies of love (3)** | 0.99 | 0.00 |
| **Nonstop car chases (4)** | 0.10 | 1.00 |
| **Swords vs. karate (5)** | 0.00 | 0.90 |

*Table 2.2.1: Movie Embedding Matrix*

If we have features like these, then each movie can be represented with a feature vector. Once we have these feature vectors, every user's preferences can be learned from the ratings he/she gives to the movies he already watched. i.e., every user also has an embedding that defines his/her preferences.

### 2.2.1 Objective

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}|\boldsymbol{X}) = \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}\left(\left(\theta^j\right)^T x^i - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{d}\left(\theta_k^{(j)}\right)^2$$

$\theta^j = feature\ vector\ of\ user\ j$ $\qquad$ $x^i = feature\ vector\ of\ movie\ i$

$r(i,j) = 1 \quad if\ user\ j\ rated\ movie\ i$ $\qquad$ $y^{(i,j)} = rating\ given\ to\ movie\ i\ by\ user\ j$

$(\theta^j)^T x^i = predicted\ rating$ $\qquad$ $n_m = number\ of\ movies$

$d = number\ of\ features/dimensions$ $\qquad$ $n_u = number\ of\ users$

The challenge is to extract the features of items that are the most predictive cleanly. Though Feature extraction/labeling of the movies manually is an exhaustive process, Content-based filtering can predict relevance for items without ratings (e.g., new items, high-turnover items like news articles, substantial item spaces like web pages); collaborative filtering needs ratings for an item to predict for it. On the other hand, content-based filtering needs content to analyze. Collaborative filtering does not require content. Therefore, Content-based filtering and collaborative filtering have long been viewed as complementary.

### 2.2.2 Drawback

A content filtering model can only be as complex as the content to which it has access. For instance, if the system only has genre metadata for movies, the model can only incorporate this one too coarse dimension. Furthermore, if there is no easy way to extract a feature automatically, then content-based filtering cannot consider that feature. Collaborative filtering allows the evaluation of such features automatically because people are doing the evaluation.

## 2.3 Why Dot Product

Traditionally, the dot product or higher-order equivalents have been used to combine two or more embeddings to give the similarity between them. For two vectors $\bar{a} = < a_1, a_2, a_3 >$ and $\bar{b} =< b_1, b_2, b_3 >$

$$dot\ product = \bar{a}.\bar{b} = a_1b_1 + a_2b_2 + a_3b_3 = |\bar{a}|\left|\bar{b}\right|\cos\theta$$

The dot product of two vectors is directly proportional to the cosine of the angle between them. $\cos\theta$ is the measure of directional similarity between two vectors in space. Therefore, the dot product or cosine similarity is considered to be an excellent choice to measure the similarity between embeddings.

## 2.4 Overcome the drawback of content-based filtering

We incorporate feature learning in collaborative filtering to overcome the possibility of underfitting. Underfitting happens in content-based filtering due to limitations in the choice of features while manually selecting features.

### 2.4.1 Feature Learning

Let's suppose that each of these users (Table 1.3.1) has a set of features for them. In particular, let's say that each of the movies has three features denoted by $\theta_0$, $\theta_1$, and $\theta_2$ that are learned from another algorithm based on the hoe the rated different movies (preferences).

| | **Manasa (1)** | **Tarun (2)** | **Ashish (3)** | **Harshi (4)** |
|---|---|---|---|---|
| $\theta_0$ | 0 | 0 | 0 | 0 |
| $\theta_1$ | 5 | 5 | 0 | 0 |
| $\theta_2$ | 0 | 0 | 5 | 5 |

*Table 2.4.1: User Embedding Matrix*

If we have features like these, then each user can be represented with a feature vector. Once we have these feature vectors, every movie's features can be learned from the ratings the users gave to the movies they already watched.

### 2.4.2 Objective

$$\min_{\boldsymbol{X}} J(\boldsymbol{X}|\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r(i,j)=1}\left(\left(\theta^j\right)^T x^i - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{d}\left(x_k^{(i)}\right)^2$$

$\theta^j = feature\ vector\ of\ user\ j$ $\qquad$ $x^i = feature\ vector\ of\ movie\ i$

$r(i,j) = 1 \quad if\ user\ j\ rated\ movie\ i$ $\qquad$ $y^{(i,j)} = rating\ given\ to\ movie\ i\ by\ user\ j$

$(\theta^j)^T x^i = predicted\ rating$ $\qquad$ $n_m = number\ of\ movies$

$d = number\ of\ features/dimensions$ $\qquad$ $n_u = number\ of\ users$

## 2.5 Collaborative Filtering

Randomly initialize the embeddings of movies and users and alternatingly, learn $\boldsymbol{\theta}$ and $\boldsymbol{X}$ from user ratings $y^{(i,j)}$ one step at a time with the following formulae:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}|\boldsymbol{X}) = \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}\left(\left(\theta^j\right)^T x^i - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{d}\left(\theta_k^{(j)}\right)^2$$

$$\updownarrow$$

$$\min_{\boldsymbol{X}} J(\boldsymbol{X}|\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r(i,j)=1}\left(\left(\theta^j\right)^T x^i - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{d}\left(x_k^{(i)}\right)^2$$

$$Learn\ \boldsymbol{\theta}\ \rightarrow Learn\ \boldsymbol{X}\ \rightarrow\ Learn\ \boldsymbol{\theta}\ \rightarrow\ Learn\ \boldsymbol{X}\ \rightarrow\ Learn\ \boldsymbol{\theta}\ \rightarrow\ Learn\ \boldsymbol{X}\ \rightarrow\ Learn\ \boldsymbol{\theta}$$

$$\boldsymbol{Y}(\mathbb{R}^m \times \mathbb{R}^u) = \boldsymbol{X}(\mathbb{R}^m \times \mathbb{R}^d) \times (\boldsymbol{\theta}(\mathbb{R}^u \times \mathbb{R}^d))^T$$

Here, we are deriving two embedding matrices from a single rating matrix. Therefore, we call this matrix factorization.

### 2.5.1 Optimized objective for Collaborative learning

Rather than optimizing for either $\boldsymbol{\theta}$ or $\boldsymbol{X}$ in a given step, we can optimize both $\boldsymbol{\theta}$ and $\boldsymbol{X}$ with a new cost function. Optimizing both $\boldsymbol{\theta}$ or $\boldsymbol{X}$ at a time in a single step might sometimes give smaller gradients for gradient descent. But when we consider the reduction in the amount of time spent on training the parameters cumulatively, we see a huge advantage.

$$\min_{\boldsymbol{X},\boldsymbol{\theta}} J(\boldsymbol{X}|\boldsymbol{\theta}) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left(\left(\theta^j\right)^T x^i - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{d}\left(x_k^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{d}\left(\theta_k^{(j)}\right)^2$$

### 2.5.2 Mean Normalisation

Consider a situation where there is a user who has not rated any movie yet. How will the feature vector of the user look like? We randomly initialize the feature vector. Now let us take a look at the loss component user ($j$) is responsible for in the loss function.

**Loss function component of the user (j):**

$$\min_{X,\theta} J(X|\theta) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left(\left(\theta^{j}\right)^{T}x^{i} - y^{(i,j)}\right)^{2} + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{d}\left(x_k^{(i)}\right)^{2} + \frac{\lambda}{2}\sum_{j=j}^{j}\sum_{k=1}^{d}\left(\theta_k^{(j)}\right)^{2}$$

Eventually the all the user's features will become zeroes because of the regularization term in our loss function, which happens to minimize the loss function.

If we try to recommend movies to the user with feature vector zero, the predictions to all the movies will also turn out to be zero, which is not desirable because we would wish to recommend content to new customers too. To overcome this problem, we use mean normalization.

We normalize each row in the matrix $\boldsymbol{Y}$, containing all the ratings, as a pre-processing step.

$$y^{(i,j)} = y^{(i,j)} - \mu_i \quad (where\ \mu_i\ is\ the\ mean\ all\ the\ ratings\ given\ to\ movie\ i)$$

$$new\ prediction: p(i,j) = \left(\theta^{j}\right)^{T}x^{i} + \mu_i$$

After pre-processing, the prediction $p(i,j)$ for the user ($j$) who has not rated any movies will be the mean rating of the movie ($i$), i.e., $\mu_i$, which means the user is given recommendations based on the preferences of the entire user base.

# Chapter 3: MLP Approach

## 3.1 Multilayer Perceptron

A multi-layer perceptron (MLP) is a class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to any feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptron (with threshold activation). Multi-layer perceptron is sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer. An MLP consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. Except for the input nodes, each node is a neuron that uses a non-linear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

## 3.2 Difference between dot product and MLP approach

Consider a function $\phi : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ that combine two d-dimensional embedding vectors $p \in \mathbb{R}^d$ and $q \in \mathbb{R}^d$ into a single score. For example, $p$ could be the embedding of a user, $q$ the embedding of an item and $\phi(p, q)$ is the affinity of this user to the item. The embeddings $p$ and $q$ can be model parameters such as in matrix factorization. Still, they can also be functions of other features; for example, the user embedding p could be the output of a deep neural network taking user features as input.

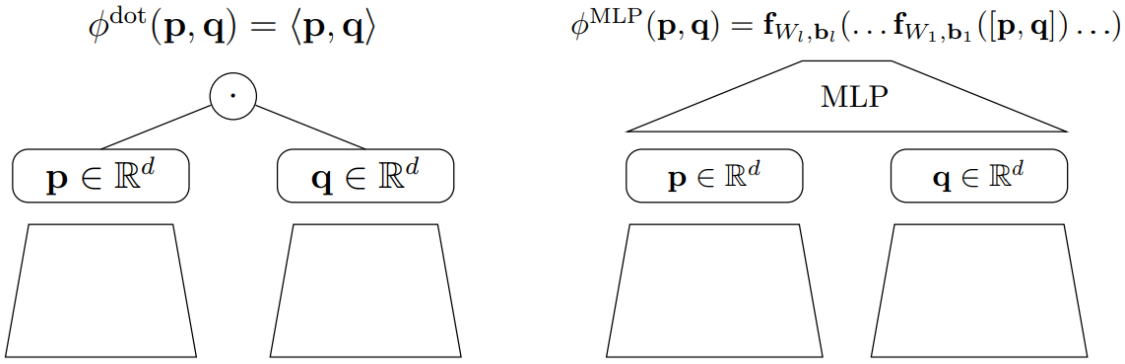


*FIGURE: 3.2.1*

The only difference between these approaches is the implementation of the similarity function. Therefore, from here on, we focus mainly on the similarity function $\phi$

**Dot Product:** The most common combination of two embeddings is the dot product.

$$\phi^{dot}(p,q) = \langle p,q \rangle = p^T q = \sum_{i=1}^{d} p_i q_i$$

If p and q are free model parameters, then this is equivalent to matrix factorization. A common trick is to add explicit biases:

$$\phi^{dot}(p,q) = b + p_1 + q_1 + \langle p_{[2,..,d]}, q_{[2,..,d]} \rangle$$

This modification did not add expressiveness but has been found to be useful because of its inductive bias.

**Learned Similarity:** Multi-layer perceptron (MLPs) are known to be universal approximators that can approximate any continuous function on a compact set as long as the MLP has enough hidden states. A layer of a multi-layer perceptron can be defined as a function $f: \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$

$$f_{W,b}(x) = \sigma(Wx + b) \qquad \sigma(z) = [\sigma(z1), \ldots, \sigma(zout)]$$

which is parameterized by $W \in \mathbb{R}^{in \times out}$, $b \in \mathbb{R}^{out}$ and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$. In a multi-layer perceptron (MLP), several layers of $f$ are stacked. The paper 'Neural collaborative filtering' proposes to replace the dot product with learned similarity functions for collaborative filtering. They suggest to concatenate the two embeddings, $p$ and $q$, and apply an MLP:

$$\phi^{MLP}(p,q) := f_{Wl,bl}(\ldots f_{W1,b1}([p,q]))$$

They further suggest a variation that combines the MLP with a weighted dot product model and named it neural matrix factorization (NeuMF):

$$\phi^{NeuMF}(p,q) := \phi^{MLP}(p[1,\ldots j], q[1\ldots j]) + \phi^{GMF}(p[j+1\ldots d], q[j+1\ldots d])$$

where GMF is a 'generalized' matrix factorization model:

$$\phi^{GMF}(p,q) := \sigma\left(\sum_{i=1}^{d} w_i p_i q_i\right)$$

## 3.3 Results

I have used the Netflix-Prize-data dataset from Kaggle for training purposes.
This dataset contains

- 96,215,056 ratings
- A range of 2,649,429 User IDs
- For 17,770 movies.

Trained two models with $prediction = \phi$

$$\phi^{dot}(p,q) = b + p_1 + q_1 + \langle p_{[2,..,d]}, q_{[2,..,d]} \rangle \; for\ dot\ product\ similarity$$

$$\phi^{NeuMF}(p,q) := \phi^{MLP}(p[1,\ldots j], q[1\ldots j]) + \phi^{GMF}(p[j+1\ldots d], q[j+1\ldots d])\ for\ learned\ similarity$$

It is observed that, on training both the models, the errors in predictions made have dropped to below 0.2 relative to the interclass distance of 1 in the case of ratings. This error can not be further reduced because the target ratings are categorical, but we are trying to predict the likelihood of

the user's affinity towards a particular movie. This likelihood is a continuous variable. However, this marginal error of 0.2 can be further reduced if we snap/round the predictions to the nearest integer.

## 3.4 Computational Difference between dot product and MLP

Calculating the number of multiplications in the dot product

$$100\ per\ rating$$

Calculating the number of multiplications in NeuMF method with #hidden_units = 100 and 25

$$(100 + 1) \times 100 + (100 + 1) \times 25\ + (25 + 1) \times 1 = 12651\ per\ rating$$

Note that there is a 125 times increase in the number of multiplications needed for a single prediction of almost similar accuracy. Although one can argue that the GPU parallelizes the computation and that there will not be a 20 times difference in the actual time taken. We should observe that, if conserved, the saved computational power can be used for several other purposes concurrently in an industrial environment.

## 3.5 Learning a Dot Product with MLP is Hard

An MLP is a universal function approximator. Any continuous function on a compact set can be approximated with a large enough MLP. It is tempting to argue that this makes the MLP a more powerful embedding combiner, and it should thus perform at least as well or better than a dot product. However, such an argument neglects the difficulty of learning the target function using MLPs. The larger class of functions also implies more parameters needed for representing the function. Hence it would require more data and time to learn the function and may encounter difficulty in actually learning the desired target function. Indeed, specialized structures, e.g., convolutional, recurrent, and attention structures, are common in neural networks. There is probably no hope to replace them using an MLP though they should all be representable.

## 3.6 Conclusion and ideas for improvement

MLPs are too costly to use for item recommendation in production environments, while dot products allow applying very efficient retrieval algorithms. This is because, in production, it is essential to strike the right balance between good recommendations and the time spent on recommending. We conclude that MLPs should be used with care as embedding combiner and that dot products might be a better default choice.

It is computationally expensive to make predictions for each user movie pair. Alternatively, we can cluster movies using algorithms like k-means clustering and make a prediction on just the means of the clusters. After finding the argmax, we can recommend all the movies of the respective cluster to the user. This saves computational resources that can be spent on other tasks.

** model trained on Google Colaboratory

# Chapter 4: ANNEXURE

Implementation of the models above mentioned is done in Tensorflow, Keras.

Detailed implementation can be found on GitHub / Google Colaboratory

**RecommenderModelDot:**

```
class RecommenderModelDot(tf.keras.Model):
    def __init__(self, no_of_movies, no_of_users, size_of_embeddings=100):
        super(RecommenderModelDot, self).__init__()
        self.size_of_embeddings = size_of_embeddings
        self.bias = tf.random.normal([1], 0, 1, tf.float32)
        self.embeddings_for_movies = tf.keras.layers.Embedding(no_of_movies, size_of_embeddings, embeddings_regularizer=tf.keras.regularizers.L2(l2=0.00001))
        self.embeddings_for_users = tf.keras.layers.Embedding(no_of_users, size_of_embeddings,embeddings_regularizer=tf.keras.regularizers.L2(l2=0.00001))
    def call(self, inputs_movies, inputs_users):
        x = self.embeddings_for_movies(inputs_movies)
        y = self.embeddings_for_users(inputs_users)
        biasx, x = tf.split(x, [1, self.size_of_embeddings-1], 1)
        biasy, y = tf.split(y, [1, self.size_of_embeddings-1], 1)
        return tf.squeeze(tf.math.add_n([self.bias + biasx , biasy , tf.keras.backend.batch_dot(x,y)]))
model_1 = RecommenderModelDot(17770+1, 2649429+1, 100)
model_1(x,y)
model_1.summary()
```

```
Model: "recommender_model_dot"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        multiple                  1777100
_________________________________________________________________
embedding_1 (Embedding)      multiple                  264943000
=================================================================
Total params: 266,720,100
Trainable params: 266,720,100
Non-trainable params: 0
```

**RecommenderModelNeuMF:**

```
class RecommenderModelNeuMF(tf.keras.Model):
    def __init__(self, no_of_movies, no_of_users, size_of_embeddings=100):
        super(RecommenderModelNeuMF, self).__init__()
        self.size_of_embeddings = size_of_embeddings
        self.j = size_of_embeddings//2
        self.DotWeights = tf.random.normal([self.size_of_embeddings-self.j], 0, 1, tf.float32)
        self.embeddings_for_movies = tf.keras.layers.Embedding(no_of_movies, size_of_embeddings, embeddings_regularizer=tf.keras.regularizers.L2(l2=0.00001))
        self.embeddings_for_users = tf.keras.layers.Embedding(no_of_users, size_of_embeddings,embeddings_regularizer=tf.keras.regularizers.L2(l2=0.00001))
        self.hidden_layer_1 = tf.keras.layers.Dense(100, activation=tf.nn.relu, kernel_regularizer=tf.keras.regularizers.L2(l2=0.00001))
        self.hidden_layer_2 = tf.keras.layers.Dense(25, activation=tf.nn.relu, kernel_regularizer=tf.keras.regularizers.L2(l2=0.00001))
        self.output_layer = tf.keras.layers.Dense(1, activation=tf.nn.relu, kernel_regularizer=tf.keras.regularizers.L2(l2=0.00001))
    def call(self, inputs_movies, inputs_users):
        x = self.embeddings_for_movies(inputs_movies)
        y = self.embeddings_for_users(inputs_users)
        j = self.size_of_embeddings//2
        xMLP, xGMF = tf.split(x, [self.j, self.size_of_embeddings-self.j], 1)
        yMLP, yGMF = tf.split(y, [self.j, self.size_of_embeddings-self.j], 1)
        return tf.math.add(tf.math.reduce_sum(tf.math.multiply(yGMF, tf.math.multiply(self.DotWeights, xGMF)), axis=1),
                           tf.squeeze(self.output_layer(self.hidden_layer_2(self.hidden_layer_1(tf.concat([xMLP,yMLP], axis=1))))))
model_2 = RecommenderModelNeuMF(17770+1, 2649429+1, 100)
model_2(x,y)
model_2.summary()
```

```
Model: "recommender_model_neu_mf"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      multiple                  1777100
_________________________________________________________________
embedding_3 (Embedding)      multiple                  264943000
_________________________________________________________________
dense (Dense)                multiple                  10100
_________________________________________________________________
dense_1 (Dense)              multiple                  2525
_________________________________________________________________
dense_2 (Dense)              multiple                  26
=================================================================
Total params: 266,732,751
Trainable params: 266,732,751
Non-trainable params: 0
```

# Chapter 5: References and External Dataset

Schafer, Ben & J, Ben & Frankowski, Dan & Dan, & Herlocker, & Jon, & Shilad, & Sen, Shilad. (2007). Collaborative Filtering Recommender Systems.

Steffen Rendle, Walid Krichene, Li Zhang, John Anderson. (2020) Neural Collaborative Filtering vs. Matrix Factorization Revisited

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web (Republic and Canton of Geneva, Switzerland, 2017), WWW '17, International World Wide Web Conferences Steering Committee, pp. 173–182.

*Dataset is from* https://www.kaggle.com/netflix-inc/netflix-prize-data