NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

Recommender Systems Implementations with Deep Learning

Panayiotis C. Lolos

Supervisor: **Panagiotis Stamatopoulos,** Assistant Professor

ATHENS

MAY 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποιήσεις Συστημάτων Συστάσεων μέσω Βαθιάς Μάθησης

Παναγιώτης Χ. Λώλος

Επιβλέπων: **Παναγιώτης Σταματόπουλος,** Επίκουρος Καθηγητής

ΑΘΗΝΑ

ΜΑΪΟΣ 2020

BSc THESIS

Recommender Systems Implementations with Deep Learning

Panayiotis C. Lolos

S.N.: 1115201300088

SUPERVISOR: **Panagiotis Stamatopoulos,** Assistant Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποιήσεις Συστημάτων Συστάσεων μέσω Βαθιάς Μάθησης

Παναγιώτης Χ. Λώλος

Α.Μ.: 1115201300088

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Παναγιώτης Σταματόπουλος,** Επίκουρος Καθηγητής

ABSTRACT

The rise of the Internet as well as the increasingly uncontrollable flow of information in the web come with multiple advantages for today´s users, yet at the same time giving birth to an unexpected issue: the paradox of choice. To counter this problem, recommender systems have been implemented in almost everything that is part of the Internet. Large-scale data mining algorithms provide users with as many targeted choices as possible, in an effort to personalize and facilitate user experience. However, the effectiveness of such systems truly shines when combined with the still young technology of deep learning algorithms. The purpose of this work is to analyze the advantages of each system separately, in order to prove the necessity of combining them. For that reason, a review of a multitude of deep learning algorithms will be provided. Furthermore, by using a simulated example, one such indicative implementation will be created and tested to support the theory behind it. Due to lack of an actual demographic and adequate equipment, the focus of this paper will be in the procedural approach of each method, while still offering theoretical feedback and trying to predict its outcome. In the final section of this study, a few optimization problems will be addressed, as well as some possible unproven solutions.

SUBJECT AREA: Artificial Intelligence

KEYWORDS: Deep learning – Machine learning – Recommender systems – Artificial Intelligence – Optimization

ΠΕΡΙΛΗΨΗ

Η εξέλιξη του διαδικτύου και η ανεξέλεγκτη αύξηση ροής της πληροφορίας εντός του προσφέρουν πολλά πλεονεκτήματα στους σημερινούς χρήστες, γεννούν όμως ταυτόχρονα ένα απροσδόκητο πρόβλημα: το παράδοξο της επιλογής. Ως απάντηση στο εν λόγω πρόβλημα, έχουν υλοποιηθεί συστήματα προτάσεων σε σχεδόν οτιδήποτε αποτελεί μέρος του διαδικτύου. Αυτοί οι αλγόριθμοι εξόρυξης μεγάλων δεδομένων παρέχουν στους χρήστες όσο το δυνατόν περισσότερες στοχευμένες επιλογές, σε μια προσπάθεια εξατομίκευσης και διευκόλυνσης της εμπειρίας του χρήστη. Ωστόσο, η αποτελεσματικότητα τέτοιων συστημάτων εντείνεται όταν συνδυαστούν με την ακόμη νεαρή τεχνολογία αλγορίθμων βαθιάς μάθησης. Ο σκοπός της παρούσας έρευνας είναι να αναλυθούν τα θετικά στοιχεία κάθε συστήματος ξεχωριστά, ούτως ώστε να αποδειχτεί η αναγκαιότητα του συνδυασμού τους. Επιπλέον, με χρήση προσομοιωμένων παραδειγμάτων, θα δημιουργηθεί και θα δοκιμαστεί μία ενδεικτική υλοποίηση για να υποστηρίξει την θεωρία. Λόγω ελλείψεων όσον αφορά πλήθος πραγματικών χρηστών και υπάρχοντος εξοπλισμού, το έργο αυτό θα επικεντρωθεί στην διαδικαστική προσέγγιση κάθε μεθόδου, προσφέροντας ταυτόχρονα ένα θεωρητικό υπόβαθρο και προσπαθώντας να προβλέψει τα αποτελέσματα της. Στο τελευταίο τμήμα της έρευνας θα θιχτούν ορισμένα προβλήματα βελτιστοποίησης και θα προταθούν ορισμένες πιθανές μη αποδεδειγμένες λύσεις.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή Νοημοσύνη

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Βαθιά μάθηση – Μηχανική μάθηση – Συστήματα προτάσεων – Τεχνητή νοημοσύνη - Βελτιστοποίηση

AKNOWLEDGMENTS

For this BSc Thesis, I would like to thank my supervisor, ass. Prof. Mr. P. Stamatopoulos for his guidance and valuable contribution to its completion.

CONTENTS

1. INTRODUCTION 14
2. RECOMMENDER SYSTEMS 15
2.1 Analysis 15
2.2 Issues concerning Recommender Systems 16
2.2.1 Prediction Accuracy 16
2.2.2 Data Sparsity 17
2.2.3 The cold-start problem 18
2.2.4 System scalability 19
2.3 Secondary Concerns about Recommender Systems 19
3. DEEP LEARNING 21
3.1 What is Deep Learning? 21
3.2 Benefits of Deep Learning 22
3.3 Challenges of Deep Learning 23
3.4 Deep Learning Frameworks 24
4. TRANSPOSING THE RECOMMENDATION PROBLEM INTO A DEEP LEARNING PROBLEM 25
4.1 An example 25
4.2 The MovieLens Dataset 26
4.3 Enhancing the Dataset 27
5. IMPLEMENTATIONS 28
5.1 Traditional Content Based-Collaborative Filtering Hybrid (CB-CFH) 28
5.2 Linear Deep Neural Network (LDNN) 29
5.3 Convolutional Deep Neural Network (ConvDNN) 30
6. CONCLUSION AND FUTURE WORK 32
ABBREVIATIONS 33
REFERENCES 35

LIST OF TABLES

Table 1: Sample prediction matrix 17
Table 2: Movie x User matrix 25
Table 3: Movies and genres list 25
Table 4: Numerical results… 31

LIST OF FIGURES

Figure 1: Deep Neural Network schematic… 21
Figure 2: Convolutional Image Analysis… 31
Figure 3: Netflix Interface... 33

1. INTRODUCTION

Today’s era, rightfully branded the *Information Age,* is more than ever defined by an oversupply of information widely spread over the now prevalent Web. Of the amount of information provided to users on a daily basis currently 90% has been created and uploaded in the past 3 years. Combining that with the ever increasing population of users (currently at 7.7 billion worldwide) [1], it becomes obvious that the flow of data is out of control. Every day, users of various applications (social media, streaming services, search engines etc) are bombarded with an infinite amount of choices to pick from. This phenomenon has given rise to an unexpected problem concerning user experience, according to which the more choices a user is offered the more negative feelings he has [2]. This paradox of choice consists of the fact that, when offered with too many options, a user’s satisfaction from choosing one may be outweighed by his disappointment from sacrificing the rest, which in turn may lead to confusion before making the choice [3], thus the need for big data algorithms that not only personalize said options to fit a user’s needs but also limit them to an acceptable number. Such algorithms are called **recommender systems** and their main goal is to adjust search results, data recommendations, advertisements etc to each user separately according to his own needs and tastes. Still, the effectiveness of such algorithms is falling behind the rapidly growing internet industry. The purpose of this work is to propose the implementation with the still young **deep learning** technology, by offering a multitude of algorithms of different approaches and relative comparisons between them. In the following sections, we will present the two main approaches for creating a recommender system, as well as their combination, and provide a number of deep learning techniques that can be implemented within such a system to increase its overall effectiveness.

2. RECOMMENDER SYSTEMS

As already stated, a recommender system is a big data algorithm, using live or saved information in order to provide the user with specific results that would more likely suit his tastes and needs. There are two main approaches when creating a recommender system, each with its advantages and disadvantages, which will be presented below. To make this analysis clearer to the reader, we will also examine Netflix's recommendation algorithm, which combines both implementations, while yet being a somewhat ineffective system.

2.1 Analysis

A precise description of recommender systems would be that by [4], where they are defined as "software tools and techniques providing suggestions for items to be of use to a user". In other words, recommender systems (from here on **RS)** optimize any amount of data, ranking it from most important or useful to least important or useless and offering it to the user as a list of items suited to his profile. The goal of said process is as accurate predictions as possible in pairing a user's needs with some data traits, both of which require a fair amount of information to be considered resourceful. There are a number of ways with which RSs procure the data required for improvements, some of which include:

- **Content-based (CB):** A history of choices is saved for each user, creating a profile and determining the traits that would make an item more attractive or useful to him. Non-viewed items are filtered by simple comparisons with viewed items (or a mean trait score of viewed content) and given a rank accordingly. This is the most basic RS technique and one of the two main approaches [5].
- **Collaborative filtering (CF):** This approach treats users as parts of bigger groups that share common traits like location, language, preferences, past choices etc. Each group is given an overall or mean profile by using a content-based technique, then a user will be recommended items that match his group's tastes. This is the second most popular approach in RS so far, as it offers immense versatility and can be combined with most other approaches effectively [6].
- **Demographic:** This technique could be treated as a more specialized and simplified collaborative filtering, with the main difference lying in the group traits. Each group is strictly divided by specific traits, including countries, language, religion etc and offered item lists that best match the group's mean score. The simplicity of said technique allows many websites to use demographic RSs for targeted advertising [7].
- **Community-based:** Yet another type of collaborative filtering. This approach groups users according to actual relations between them, rather than common traits. A user is recommended items that would best match his friends' preferences or their mean score. These algorithms are best suited for social media advertising [4].
- **Knowledge-based:** This approach concerns specified domain knowledge about an item and is often used to provide the user with items whose features would best suit his needs rather than his tastes. This is not as much a prediction making technique as it is a problem solving one and fairly static. Such systems are mostly case-based and tend to work better

than the other techniques during early deployment, but without some self-learning components they can be easily be surpassed [8].

- **Contextual Collaborative Filtering:** Quite similar to classic collaborative filtering, this method is focused on a user's preferences outside the scope of the recommended data, such as website visits, other services, social media activity etc. The purpose of this wide approach is to predict a user's preferences at a specific time, rather than according to his overall history, which results in a more adaptable and responsive system. This has become increasingly effective with the use of internet cookies and is mainly used for advertising. [9]
- **Hybrid RS:** Any combination of the above. A notable hybrid is the Netflix recommendation algorithm, which uses both content-based and collaborative filtering along with minor demographic implementations [9].

Each of the approaches presented comes with its own set of advantages and drawbacks, most of which can be mitigated or completely solved with the implementation of a self-learning algorithm within the RS [10]. As previously stated, RSs rely heavily on the amount of available information both on the user and on the data required, thus offering fairly poor results at their deployment, with the exception of knowledge-based RSs. By implementing deep learning in an RS, a developer can massively increase its effectiveness during the beginning of its deployment. Such RSs are already being applied to a variety of domains, such as entertainment (movies, music, games etc), e-commerce (consumer recommendations), services (travel agencies, consultants, real estate, matchmaking services), or any kind of content from e-mail filters to personalized newspapers [11].

2.2 Issues concerning Recommender Systems

The nature of recommending algorithms dictates several notable issues, all of which will be briefly presented in the following sections. As prediction systems, these challenges include:

1. **Prediction Accuracy**
2. **Data sparsity**
3. **The cold-start problem**
4. **System scalability**

There have been various solutions proposed for these issues, most of which revolve around deep learning technology. [10]

2.2.1 Prediction Accuracy

Arguably the issue that is getting the most attention concerning RSs is about the accuracy of their predictions. Recommender systems are, in essence, prediction algorithms, therefore, being able to quantify the scale of their predictions is one of the most essential tools current developers have in improving them. There are a few metrics used to measure accuracy, each with its own focus and limitations [4].

The metric used in this work is the most discussed one, **Root Mean Squared Error (RMSE)** [12]**.** This metric takes into account a test set *T* and its predicted results ($\hat{r}$) along with its true results (*r*) concerning the set's user-item pairings (*u,i*). The RMSE for prediction accuracy is therefore calculated as shown:

$$\mathrm{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|}\sum_{(u,i)\in\mathcal{T}}(\hat{r}_{ui} - r_{ui})^2}$$

Clearly, RMSE quantifies the ability of an RS to actively pair a user´s needs or preferences with specific item features. It should be noted that RMSE is focused on the accuracy of individual predictions over the mean accuracy of all predictions, meaning that an algorithm that hits all predictions within the error margin will be evaluated higher than an algorithm with one or two predictions way over the error margin, even if the last one had more accurate predictions overall. To be precise, when evaluating an RS using only accuracy as criteria, one can choose one of two options for improvements:

- By focusing on individual highest accuracy.
- By eliminating low accuracy below a certain threshold.

For the purposes of this work, the latter was chosen, however, the first one could be simply implemented by using the **Mean Absolute Error (MAE)** metric, which is a popular alternative to RMSE, as it offers greater variety of results within a set error margin [12].

Also, RMSE is used by Netflix to measure prediction accuracy on user ratings on movies before them being recommended, adding another layer to the factors for each recommendation [9].

2.2.2 Data Sparsity

To make this issue clearer to the reader, it is important to first understand the way most RSs work. Assuming there are X users, each with N preferences, and Y items, each owning M features, an RS creates a user-item matrix using the data at hand as well as a specific function depending on the algorithm's focus (Content-based, Collaborative – filtering, etc). The resulting matrix will be an X*Y matrix with empty cells at the beginning. For every item *y* rated by a user *x*, the cell [*x,y*] will be filled with a value processed depending on the function used by the algorithm. Since not every user has viewed and rated every item, the end result will appear as something like this:

Table 1: Sample prediction matrix

	ITEM#1	ITEM#2	ITEM#3	ITEM#4	ITEM#5	ITEM#6	ITEM#7	…	ITEM#Y
USER#1	1	0.8		3.8			4.2		
USER#2			2.2		3.1		1		
USER#3		0.5	2.7		3		5		1.4
…									
USER#X	5	1.1		2.6		1			4.9

The prediction matrix used by RSs at first only contains values in the cells that represent actual ratings. For example, User 1 has rated Item 7 for 4.2, so cell [1,7] has a value of 4.2. The empty cells must be filled by the RS with estimated values.

The purpose of every recommender system is to practically fill in the blanks with a guessed (predicted) value which will eventually be replaced by the user's real rating. If the guessed value of an empty cell is beyond a certain threshold or higher when compared to all other empty cell values, then the item will be recommended, otherwise it will be left out. It should be noted here that in dynamic systems, the increase of a user or item just by one will automatically increase the number of empty cells by Y or X respectively.

This process poses a challenge: the fact that users only view a small percentage of the available items and rate an even smaller fraction of that. [13] This means that the number of empty cells in the matrix is vastly larger than that of the filled cells, thus making it quite difficult for an algorithm to make valid or even reliable predictions. This is called *the data sparsity issu*e and concerns especially collaborative-filtering RSs, where an increase in the number of users instantly and exponentially increases the prediction difficulty, whereas the users themselves require much longer to actively rate items – and that's assuming the items do not increase either [14].

2.2.3 The cold-start problem

Data sparsity is a major issue for all RSs, especially when they are first deployed. It becomes virtually impossible for an algorithm that uses user/item history (a.k.a content based RS) to make even the slightest predictions when the data matrix is completely empty. This phenomenon is also called the cold-start problem [10] and can be observed in three situations:

- **First deployment.** When an algorithm is first deployed to operate over an existing environment of users and items, users have not yet rated anything and items have only their static features and zero ratings on themselves. This ultimately means that an RS requires a specific amount of time and actions/ratings from users before accurately making predictions for them. Various solutions have been tried to counter this issue, with a core focus in deep learning and content filtering [10].
- **New items.** When a new item is added, the prediction matrix is instantly increased by the number of users. The item, however, is unrated and only has its static features. This increases the complexity of calculations and reduces accuracy in the beginning. One way to counter this is by using a reverse collaborative-filtering algorithm (r**CF**), which consists of using the ratings of similar items by all users to determine which user group would find a new item desirable. [15]
- **New users.** This issue is the same as that of a new item with one difference: users' preferences are ever changing whereas items' features mostly remain static [10]. This fact makes the use of similar solutions to rCF tricky, as the choices of other users do not necessarily provide accurate mean scores for the new users. This also poses a major challenge for content-based algorithms, since new users have no history of preferences. One effective solution has been implemented in the Netflix algorithm, where every new user is immediately prompted to choose some items that he would enjoy out of a provided list, giving the Netflix RS a virtual user history right from the get-go. [9]

2.2.4 System scalability

One of the main purposes for the development of RSs is reducing the time a user spends making a decision, so it is required of the algorithm to calculate the recommendations rapidly and smoothly [16]. This may become a challenge when the data processed is of a large scale, especially if the algorithm is being deployed for the first time. The cold-start problem poses another issue for RSs when observed from a business perspective: when a new user tries a content creator or host service, if the recommendations he is offered are not satisfactory early on, it is highly probable that the user will seek a different service [10]. For that reason, during development and after deployment, RSs are constantly being evaluated by factors different than accuracy, such as speed and resource consumption (for example the number of recommendations provided per second in a growing dataset).

It is also important to note that although an algorithm may behave adequately when first deployed on a data set, if that set grows beyond a specific size, the overall effectiveness of the algorithm might drop. The resources required for training and testing RSs over actual live datasets are usually much more than what an average personal computer can provide, which will be proven later on. For that reason, developers stress-test their RSs for a long time and in different systems before deploying them, ignoring actual accuracy measures and focusing strictly on the computational strength of the algorithms [17].

System scalability and the complexity of calculations of an RS may also pose a challenge for deep learning implementations, as it will be discussed later.

2.3 Secondary Concerns about Recommender Systems

Apart from the main technical issues when developing an RS, there are some other things to be taken into account:

- **Synonymy:** this is an issue that plays an important role in all machine learning problems. In a data set, if two items have similar names or similar features but not identical, they are treated as completely different, even if the semantics refer to identical terms. [14] For example, a movie under the term *animation* and a movie under the term *animated film* will be treated as entirely different genres. This poses an issue of system optimization, where various categories and features could be grouped up and identified as a single entity, doubling or even tripling processing speed.
- **Gray-sheep:** this issue concerns mostly collaborative filtering RSs, yet it can be a problem for any hybrid as well. It consists of the fact that some users make inconsistent choices either with their own history or with a group's overall mean history. Most RSs have implemented ways of identifying such erratic behaviors and omitting them from the calculation process [18], even though removing such behaviors from the calculation process poses a risk of reduced accuracy when a group's choices are shifting rapidly and unpredictably.
- **Shilling attacks**: these are another type of erratic user behavior. A shilling attack happens when a multitude of actual or fake users are trying to sway an RS (especially a collaborative filtering one) to calculate specific results and ignore others. It can be achieved when a large number of users rate specific items very high and other items very low in a relatively small amount of time, in order for the former to be more easily recommended than the latter. This is a clear example of misusage of RSs, requiring specific guidelines and defenses implemented within the algorithms to ensure objectivity during calculations [19].
- **GDPR concerns:** as of May 25th, 2018 the *General Data Protection Regulation* has been enforced throughout the European Union [20], raising concerns about the use of recommender systems. It is obvious that all RS have one main

requirement to operate: user personal data. This includes the risk of compromising user data to the public, even without a user’s approval. For example, any user may be exposed when using a collaborative filtering RS, since his personal preferences will be used for recommendations to other users in his area [14]. Another great example of GDPR breach involves contextual filtering, where the user’s entire browsing history may be exposed and shared via cookies to all content providers.

3. DEEP LEARNING

3.1 What is Deep Learning?

Recommender systems are a part of the greater field of *machine learning*, a subset of artificial intelligence used for data mining. Various techniques are used in machine learning, one of them being neural networks. Also called *artificial neural networks (***ANN***)*, they are loosely based on the neurons of living organisms and consist of a huge number of processing elements, each connected and cooperating with the others to solve specific problems. [21] ANNs are widely used today in numerous applications, including voice recognition, image processing and classification, decision making and data mining. Deep learning is a technique using ANNs to train an algorithm in order to better perform a task, usually about predicting or deciding things based on a large amount of data. [11] To better understand the concept of deep learning and its utility in RSs, it is important to first examine how it works microscopically, thus to look into how ANNs work.

Deep neural networks (**DNN**) are practically ANNs with a specific layout consisting of three layers:

- **Input layer** (where data is stored)
- **Output layer** (where the decision or prediction is returned)
- **Hidden layer** (where all of the data processing takes place)

All of the above layers are interconnected and each may have from tens to thousands of neurons. Each neuron takes into account two functions: a calculation function and a weight function. To better put things into perspective, let's assume the following deep learning network as shown in figure 1:

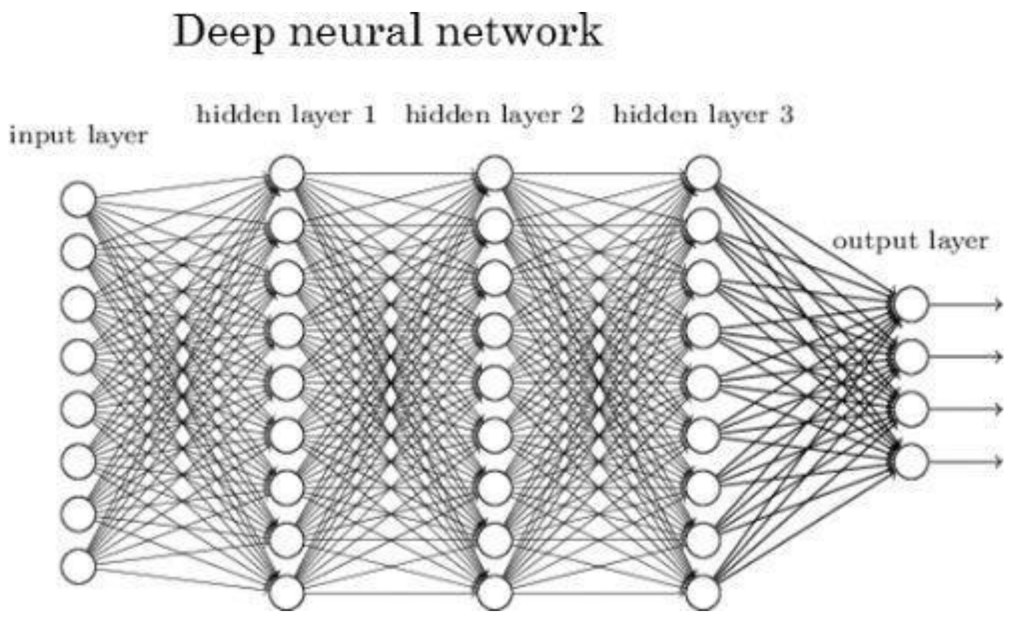

Figure 1: A typical DNN representation.

When a DNN is first deployed, a large dataset is required in order for the algorithm to create and calibrate its hidden layers. This process is the most time and resource consuming part of deep learning and it is called **training.** There are three types of training:

- **Unsupervised training** is when only the input layer is provided to the algorithm. Then the data is computed and correlated by using *forward propagation* through each *hidden layer.* Propagation means each piece of information is passed through each node's calculation function (also called activation function) in conjunction with each vector's weight function and processed accordingly. When all hidden layers have been successfully parsed, data is calculated one last time, taking into account weight and value limitations, and sent to the *output layer.* This process is also called *adaptive training.*
- **Supervised training** is where both the input layer and the output layer are already provided to the algorithm. In this case, the same process as in unsupervised training is used, only this time the algorithm will compare its results with the output layer's required results and then recalibrate the hidden layers by *backwards propagating* the errors and tweaking the vector weight. This procedure happens repeatedly until the output layers that occur from forward propagation are either the same as the provided ones or within an acceptable error margin. If after a set amount of repetitions or time the DNN hasn't converged, then it is up to the developer to detect possible faults with the algorithm or the data and change them accordingly [21].
- **Reinforcement learning** is a process currently used mainly for problem solvers and other AI, especially in the gaming industry and robotics. During its training, the network is deployed as an agent within a specific responsive environment and with no further instructions, then is allowed to freely execute all available sequences of actions within that environment. Each action results in a new environment state, some of which are closer to the target state. By trial and error, the agent recalibrates its network multiple times until it fully adapts to said environment. [23]

3.2 Benefits of Deep Learning

The application of DNNs comes with a fair amount of advantages [21], some of which can solve many of the issues concerning recommender systems today. Here is a list of such benefits and the problems they remedy:

- **Pattern recognition.** DNNs can create their own correlations between data during their training, which allows them to vastly improve their accuracy without any need for human interaction. This ability resolves two secondary issues of RSs. The first is synonymy, since they can now generalize data without it being strictly dependent on semantics, and the second is any kind of shilling attack, since they may be taken into account as "excused patterns" without the algorithm having to cross out users entirely from the recommendation process.
- **Adaptive systems.** Deep learning systems are developed through self-learning, rather than actual programming, meaning they adapt on the data at hand and improve on the task at hand. This solves issues concerning the human factor when developing an RS, as DNNs are self-organized and may change data representation to suit their needs in ways conceptually difficult for a human developer, yet far more efficient.
- **Flexibility.** A direct advantage of their adaptability, DNNs are capable of adjusting to ever-changing environments without suffering in terms of efficiency.

This solves issues concerning scalability, as many conventional RS approaches face hardships when the amount of data processed reaches a certain ceiling. DNNs might be slower in learning new changes but are excellent at eventually adjusting to them.

- **Performance.** DNNs are excellent systems when it comes to modeling data and statistical problems. They prove to be faster than classical approaches and the models they produce are more accurate. This amounts to a significant improvement of the accuracy of an RS when it comes to the addition of new users or items, as well as a better approach on the cold-start problem.
- **Complexity.** The use of multiple hidden layers, along with the handling of multiple interconnecting nodes, offers an ability to create accurate models out of vastly complex data. The sparsity problem of RSs consists on the fact that conventional algorithms can only do so much at their early deployment. DNNs can create so many possible pseudo-interactions with themselves and between data that sparsity can be effectively overcome during their training and even before their deployment.
- **Live Usage.** DNNs are highly responsive and adaptable, meaning they can keep their computations up while being used on multiple fronts. Accelerated computing environments and the use of Graphics Processor Units (*GPU)* as computing processors allow for a large amount of parallel computations, although hardware-wise the capability of such systems in that capacity is still limited [22]. This type of usage could prove to be a major asset for personalized RSs in the future by using client and server devices simultaneously to offer the user with results faster than ever before.

3.3 Challenges of Deep Learning

Most issues concerning deep learning algorithms have to do with equipment and processing power. Deep learning networks are vast and require millions of micro-calculations per second, which in turn requires high computation power and a big amount of memory. The equipment used for this work was a traditional personal computer with 8GB RAM and an older generation CPU, which struggled to procure the test results even for average sized datasets, although in the case of traditional RSs the same datasets were impossible to process.

In other words, while DNNs are quite demanding in terms of equipment, they scale excellently for increases in data processed. The added benefit of passing calculations to a graphics card makes efficiency almost 10 times greater or even more than that. [29]

Another challenge posed by DNNs is what is called **overfitting.** DNNs have a tendency of adjusting to their current training datasets, sometimes too quickly, resulting in prediction accuracies of 100% for the training set. This however means a drop in accuracy for the test set, because the DNN adapts fully on the training set, without actually taking advantage of any latent information. Overfitting usually occurs when a network undergoes training too much or for too many epochs (times of supervised training). Various solutions have been proposed for overfitting, most of which concern a change in training times or having some static “filtering” layers between the input layer and the hidden layers. The most commonly used currently, which is implemented in every deep learning library, is the use of **dropouts.** [30] During a network’s training, random units of data are dropped from the calculations in each epoch. This occurrence forces the algorithm to treat the training set as a different set every time and reduces the rate of overfitting significantly without impeding convergence.

3.4 Deep Learning Frameworks

In this section, some frameworks will be introduced, which seem to be working efficiently when implemented in RSs. Each comes with its own advantages and disadvantages, some of which will be analyzed in association with the needs of and RS, rather than their overall performance and utility. All of the following frameworks can be used and implemented with the Python programming language.

- **TensorFlow:** Developed by Google Brain, TensorFlow is an open-source framework offering a list of machine learning libraries for C++ and Python programming languages which has gained vast popularity over the past couple of years. Its advantages rely on its ability to efficiently compute multi-dimensional arrays (tensors) by using sequential graphs, however, these graphs are mostly static with little adaptability to dynamic problems [23].
- **Microsoft Cognitive Toolkit (CNTK):** Developed by Microsoft and Facebook, CNTK is an open-source framework which, similarly to TensorFlow, offers a DNN representation of directed and forward or backward propagating graphs. Its latest release (Ver. 2.7) is highly versatile and compatible with most Windows or Linux 64-bit systems or any C++/C#/Python environment. It is one of the very few frameworks that supports the *Open Neural Network Exchange* (**ONNX**) format, which allows for easy compilation and shared optimization when combining different deep learning frameworks [24].
- **Caffe:** Developed by Berkeley AI Research (BAIR) and community contributors, Caffe is a framework with its main focus on efficiency. Its main difference with the previous two frameworks is in the representation of DNNs, where they are defined layer-by-layer, as opposed to nodes and directed graphs. This allows for much higher speeds than other frameworks, while being easy to code in Python and MatLab interfaces. Although suitable for image recognition software, Caffe lacks in versatility, since layers are predefined and static, making it quite difficult for a dynamic RS to be developed without creating custom C++ layers beforehand [25].
- **PyTorch:** A vastly improved version of the no longer in development *Torch* Framework, PyTorch is an extremely versatile and readable library for GPU-accelerated Deep Learning, developed by Facebook's AI research team. It supports ONNX for other frameworks, Windows, Linux or Mac as operating systems, as well as C++, Java and Python. A core advantage of this system is its ability for dynamic computational graphs and automatic differentiation, which allows for responsive RSs and faster live training. As of October 2019, one of its main disadvantages has been solved with the release of a mobile version for iOS and Android devices [26].
- **Keras:** possibly the most popular wrapper library, Keras was developed by Francois Chollet and supports various deep learning frameworks including TensorFlow and CNTK. Keras wraps the backend Python libraries of deep learning frameworks, making them simpler and modular while offering easy extensibility and running on both CPUs and GPUs [27].

4. TRANSPOSING THE RECOMMENDATION PROBLEM INTO A DEEP LEARNING PROBLEM

4.1 An example

Deep learning is a technique, whereas RSs are algorithms that perform specific tasks. In order to implement deep learning on RSs it is first important to identify their elements and transform them into elements that can be understood by a deep learning system. To put things into perspective, let us consider the following matrix:

Table 2: A simplistic user-movie matrix representing the moment a new user and a new movie is added to a recommender system group. The values in the cells are ratings by the users for the specified movies, each being on a scale from 1 to 5.

	Movie #1	Movie #2	Movie #3	Movie #4	Movie #5
User #1	5	3.2	1	3	
User #2		2.7		4	
User #3					

Considering the above matrix, let's assume that there is an RS in place with 2 users and 4 movies in its data. A typical hybrid recommender system (usually content based and collaborative filtering) would take into account the following:

- **Users:** for the sake of simplicity, users will be considered unrelatable entities, although most RSs account for their location, online actions and anything that can be procured by their personal browsing history.
- **Movies:** movies are the items of this example. They are static and they have specific features, according to which they can be related to each other. For this example, their features could be their respective genres, their titles, even their language so let's consider the following list about them.

Table 3: A list of the 5 movie titles and their respective genres. A movie may fall under more than just one genre, which makes an RS's computation that much harder.

Movie	Genres
Toy Story	Comedy / Family Movie / Animation
Star Wars	Sci-Fi / Drama / Action / Fantasy
The Exorcist	Horror / Thriller
Men in Black	Sci-Fi / Comedy / Action
Spirited Away	Drama / Anime / Family Movie

It must be noted that the features of movies and the characteristics of users in most cases are a lot more, especially when it comes to content filtering. The main purpose of features is for the algorithms to compute relations between items which are as detailed as possible. For instance, when movies are categorized only by their genre, the correlations are much weaker than if they are categorized depending on their genre, their director, their stars and costars, their language and their release date. The same rule applies to users, although it is debatable to what extend an algorithm should use personal details for such predictions, as it can lead to GDPR breaches. [20]

All of the methods mentioned from here on are based on **matrix factorization** (henceforth **MF)**, which has proved to be the superior when it comes to classification systems. [28] Another approach would be the K-nearest-neighbor approach in which *table 2* would be treated as a network of nodes, which would increase the complexity of the system and the overall cost. In MF two arrays are created: one for the users and one for the items. *Table 2* represents the conjunction of those two arrays but fails to show the correlations between them. In other words, *table 2* is simply a representation of actual or predicted ratings (the relation between each user and each item), even though a lot more information has been processed for the predicted numbers. In MF the value of a predicted rating that will appear in *table 2* will take into account all that “hidden” information and put it in the following equation:

$$\hat{r}_{ui} = q_i^T p_u.$$

Where $\hat{r}$ is the end matrix cell with coordinates (*u,i*) and *q* and *p* are the corresponding vectors for items *i* and users *u* respectively. These vectors are calculated by using the items’ features or the users’ preferences. In content-based RSs only the factor *p* is used, whereas in collaborative filtering, only the factor *q* is used, so it becomes evident that a hybrid model would procure much more accurate results with a minimal increase in calculations. [12]

The problem lies in what vectors *q* and *p* are going to be, what exact values they are going to have for each different pair and how they can be best calibrated live. The process in which an algorithm generates these vectors is the learning stage and traditional approaches up until recently have been limited to *stochastic gradient descent (SGD)* and *alternating least squares (ALS)* both of which will not be analyzed. The learning model used in this work is a more DNN-friendly one, called **convolutional matrix factorization** (**ConvMF).** [29]

Five different approaches were implemented for the purposes of this research, each demonstrating a specific quality for recommender systems: **simple content based filtering** *(1)*, **simple collaborative filtering** *(2)*, **enhanced content-based collaborative filtering hybrid** *(3),* **linear deep neural network** *(4)* and **convolutional deep neural network** *(5).* They were all primarily trained and tested with the MovieLens 100K dataset.

4.2 The MovieLens Dataset

Before proceeding to the deep learning techniques it should be noted that for the experimentation of the algorithms the MovieLens dataset was used. It is a directory of .csv files containing millions of ratings from random users over actual movies [30]. The ratings are not necessarily accurate or real, although the dataset itself is adequate enough for stress testing, relative accuracy testing and scalability evaluation. It is recommended that such algorithms are also tested in a live responsive environment consisting of actual users before any claims can be made about their true accuracy.

Due to the dataset’s vast size, an older, smaller version of it was used for experimenting and testing. The smaller version (**ml-latest-small)** contains over 100.000 ratings, 1296 tag applications and a combination of over 9000 movies for 671 users.

4.3 Enhancing the Dataset

For the purposes of this work, the datasets were enhanced via various methods. Several different combinational and fetching functions were developed to create other datasets using the dataset at hand as the primary source and the IMDBpy API for accessing the links in the *links.csv* file and fetching additional content. An available enhanced dataset was used from kaggle.com which includes *credits.csv* and a *movies_metadata.csv*, both adding several more features to each rated movie. Furthermore, a movie poster dataset was created, using the *links.csv* file. The dataset contains approximately 99.6% of the movies' posters, stored in a directory *movie_posters.* The 0.3-0.4% of movie posters missing is due to IMDB's server issues, broken URLs and missing material. The purpose of the last two datasets will be discussed in the **"Future work"** segment.

The above enhancement was partially done on ml-100k, as the time and equipment required for it were above expectations.

5. IMPLEMENTATIONS

5.1 Traditional Content Based-Collaborative Filtering Hybrid (CB-CFH)

The traditional model is a highly developer-dependent model. It works like a funnel with an orifice of certain diameter. The system implemented here was loosely based on the one proposed by [31] can be split into two distinct but interdependent parts:

Content-based: The algorithm takes in an item *I* along with its features, a set *S* of other items and a value *K* which signifies the amount of recommendations needed. It checks *S* and ranks each item *j* belonging to *S* depending on how similar it is with *I.* The similarity is measured using the **TF-IDF** method (a.k.a. Term Frequency – Inverse Document Frequency), which calculates a respective score for each *J* using the following formula:

$$\mathbf{tfidf}_{i,j} = \mathbf{tf}_{i,j} \times \log\left(\frac{\mathbf{N}}{\mathbf{df}_i}\right)$$

Where: *i* represents a feature word for item *I*, *j* represents the set of features for item *J, N* is the total number of items in the set *S, tf* is the number of times *i* appears inside *j* and *df* is the number of items within set *S* where *i* appears. [31] Each value is then combined into a vector with *n* dimensions, where *n* is the number of features used for comparison. When all vectors have been calculated, they are normalized with a maximum length of 1and they are compared to *i's* vector by calculating the angle between them. The *K* items with vectors that have the smallest angles are selected and returned. Cosine similarity between vectors is used as a metric to measure the above, as it increases when the angle diminishes. This method, tested strictly on the *movies.csv* set procured a low accuracy rate of 47-48%, although by extracting more information from the *movies_metadata.csv* (thus increasing processing time significantly) it was able to reach an acceptable rate of 61%.

Collaborative-Filtering: The algorithm takes pairs of either users or items at a time. For user pairs, if user A is the one who needs recommendations, then it returns user B's top item choices, provided that A and B have been chosen as similar. For item pairs, if item A is an item that the user liked, then a similar-item matrix is created from which the user is recommended the most similar picks. The similarity this time is decided on what other items other users who liked item A have liked and is calculated as a distance in the similar-item matrix using cosine as a metric, just like in the content-based algorithm. The preset SVD (singular value decomposition) algorithm by Surprise library was used, wielding sufficient results with an accuracy rate of 63%.

Hybrid: A combination of the above algorithms creates a traditional hybrid RS, which comes along with almost all of the problems mentioned before, even though mitigated effectively. Each of the two approaches used may to some extend solve the issues of the other, but both eventually fail in terms of scalability. The core problems that remain unsolved revolve around the absence of actual latent data, as well as the algorithms' inability to generate it. Some of the issues the hybrid would encounter when processing *table 2* would be the following:

- CB search for *Toy Story* would consider *Men in Black* and *Spirited Away* equally similar, as it would fail to recognize similarities between feature the genre *Animation* and *Anime.* Consequently, *Men in Black* would gain a value of 1 for the term *Comedy,* and *Spirited Away* would gain a value of 1 for the term *Family Movie*, whereas the term *Anime* would not be taken into account. This represents a synonymy problem and could be mitigated by CF to some extent depending on other users' ratings.
- CB search for *Star Wars* would also struggle but for a different reason. *Men in Black* would gain a value of 2 for being *Sci-Fi* and *Action* but would lose 1 since *Comedy* is opposed to *Drama* (broadening the vector angle), resulting in a final value of 1. *Spirited Away* would also gain a value of 1 for its feature *Drama,* resulting in another equalization. This represents an accuracy problem and would probably not be solved by using CF, since content similarity and preferences similarity are fundamentally different vectors.
- CF search for user 3 or movie 5 would encounter a cold-start problem, as there are no values in their respective columns and rows, and would be unable to provide any results. User 3 and movie 5 could represent a new user or movie in the system and the issue posed by them could be solved with CB on both the user and the movie, although accuracy would still suffer.
- CF search for user 1 would encounter a sparsity problem, as roughly 40% of the matrix's other values have been set, offering a very small amounts of calculation data, as opposed to the uncalculated values. In actual datasets, the resulting matrix is usually even sparser (20-25% full at most). The problem could be mitigated by CB's presence but would still suffer from synonymy and accuracy difficulties.

Consequently, the traditional hybrid model is a valid approach only if the two algorithms are allowed to work in combination. In cases where one has to take over completely, all of its issues will arise. Additionally, the combination of the two, as well as other various enhancement techniques (mainly in the CB side, including director name weights, summary and cast filtering, stemming documents etc), cause a major scalability problem, which can be eradicated by the use of any traditional DNN instead.

It should be noted that the algorithm developed within this work, achieved very promising results, while still being unable to handle large scale data, with its scalability issues still remaining the biggest hurdle.

5.2 Linear Deep Neural Network (LDNN)

The deep learning model is a machine learning model of multiple linearly interconnected layers. The complexity of said layers is a perfect asset for implementing MF. All of the issues that occurred with the traditional CB-CFH could be solved if the algorithm could take into account the latent data that can be extracted from the actual data. For example, a human who saw tables 2 and 3 would assume that User 1 would like *Spirited Away* more than *Men in Black,* when considering the similarities between *Toy Story* and *Spirited Away, Star Wars* and *Men in Black,* as well as their existing ratings. However, the hybrid would struggle to choose between them. An LDNN does not simply "read" the data provided. It calculates correlations between every piece of information and implements them as latent data. In this specific example, an LDNN would probably make the assumption that *Animation* is directly similar to *Anime* and would also give weights to how much each feature affects a user’s rating specifically, shifting the matrix vectors significantly. The existence of these weights for each feature is an excellent countermeasure to any cold-start issues, either concerning users or items. It must be noted, although, that DNNs require vast amounts of data for training, therefore the specific set is an unrealistic example for a DNN’s performance.

The LDNN used in this work is proposed and developed by [34], only uses CF and works as follows:

- Receives the incomplete (and usually sparse) *User*X*Item* matrix. **(input layer)**
- Decomposes the matrix into two one-dimensional matrices, *Users* and *Items.*
- Correlates all features and values within each matrix and recalibrates its respective vectors. During the correlations and adjustments, the data’s latent information surfaces and can be stored as weights on each connection between nodes. **(hidden layers)**
- Merges the reformed matrices into a new *User*X*Item’* matrix. **(output layer)**

The returned matrix is filled with values and estimates, which are then used for training. The activation function used for all nodes and layers was ReLU, the most commonly used function for multi-layered networks. The above model is created by using the PyTorch library for network processing, and first underwent supervised training on a percentage of data, then tested on the rest. It should be noted that the only dataset parsed through the matrix was the *ratings.csv*.

The results were higher than anticipated, although still not as good as the fully incorporated hybrid. The algorithm’s main focus in CF proved inadequate to surpass the high accuracy of incorporating content based filtering, although still much better than any traditional CF model.

5.3 Convolutional Deep Neural Network (ConvDNN)

Convolution is a method primarily used in image processing. Assuming each image is imported as a two-dimensional matrix where each cell represents a pixel (or a characteristic for that pixel), an LDNN would simply take each pixel as a separate piece of information, then correlate them all together. ConvDNN does the same and something more. It also creates sets between pixels of an image and groups them up depending on their correlations, while omitting the ones deemed “unnecessary” for prediction. The same can be done for any type of dataset, including documents, movies, music etc. [35]

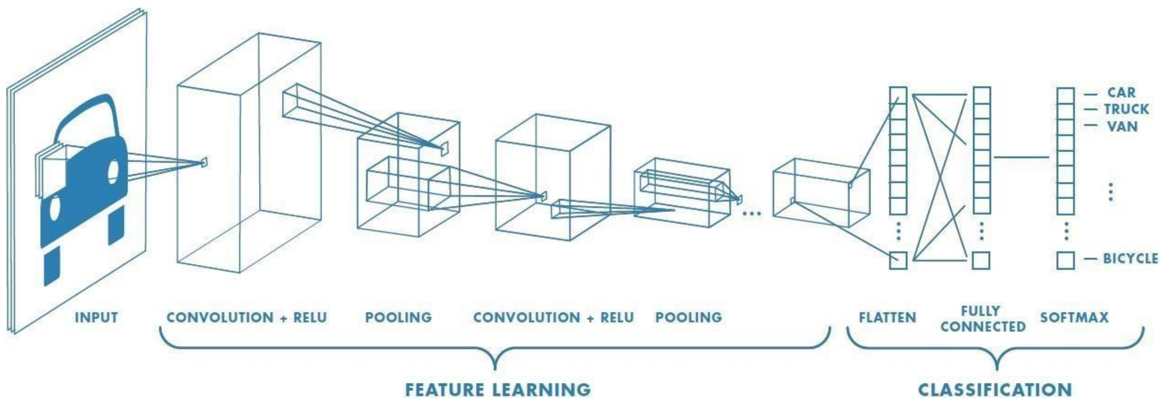

Figure 2: Representation of image analysis through a convolutional deep learning network

Implementing convolution in PyTorch over the LDNN algorithm is as simple as changing the name of one command, however it proved trickier as not all hidden layers should be convolutional. Approximately half of the previous algorithm's hidden linear layers were turned into convolutional layers.

The results showed an accuracy increase of almost 4% over the LDNN, which was still not enough to surpass the hybrid.

Table 4 summarizes the results between each implementation while offering a better view for comparison. For CF, it was impossible to create the SVD matrix for the actual dataset, as it required immense amounts of computer memory. This applied to the hybrid for the same reasons. The DNN implementations have no response times, since the prediction matrix is calculated fully during training and no further calculations are needed afterwards. Training for DNNs obviously takes a lot longer than for any traditional hybrid, however it is possible to pass the larger dataset with a DNN by splitting it into batches of certain sizes, which reduced significantly the amount of computer memory required. Still, trying to use even larger datasets proved impossible with the current equipment.

Table 4: Numerical results of each implementation while using the MovieLens latest 100k small and regular datasets

Algorithm	Training(s) (small)	Response(s) (small)	Accuracy(%) (small)	Training(s)	Response(s)	Accuracy(%)
CF	9.771	0.007	62.899	-	-	-
CB	19.788	0.016	47.601	304.622	12.004	44.920
Hybrid	35.844	0.022	85.203	-	-	-
LDNN	49.037	-	71.035	689.101	-	70.724
ConvDNN	57.961	-	73.574	842.009	-	73.221

6. CONCLUSION AND FUTURE WORK

In this work a variety of recommender system implementations were tried and tested and, while there is a need for further experimentation with more capable equipment, there is a clear advantage of deep learning technology over traditional development. The DNN RSs used for this work were strictly focused on collaborative filtering, yielding promising results, with an increase in accuracy of over 14%. According to [35] a hybrid implementation of a deep learning model would possibly reach an astonishing accuracy rate of 92%, which is as high as one can hope. This raises a question of how much content based filtering can evolve using deep learning implementation. Therefore, further research on implementing a DNN approach of the hybrid proposed here would be in order.

During testing, another question arose concerning content based filtering. The current state of the art, while efficient, is still basic in terms of methodology: Recommender systems study data history and offer accurate predictions. However, a user's tastes are ever changing and often unpredictable, meaning that even if accuracy according to data may be high,, true accuracy will be lower. A hypothesis proved by [36] is that people's needs and wants are influenced by daily activities, current state of mind and a multitude of other stimulus enough for their choices to be inconsistent with their past. The hypothesis proves that the gray sheep problem may be much more important than it is currently considered to be. Could the next level for recommendation systems be the ability to anticipate a user's current mood? What other data features would be required for such a prediction?

A great example would be that of movie recommendations. Every movie streaming service offers the list of movies as a list of their posters. Netflix' interface is shown in *Figure 3* for example. If users choose what they would like to watch depending on their current state of mind, then it is safe to assume that they can be influenced by the movie posters they see. According to [39], it is possible to predict a movie's genres by its poster, a theory proven to a promising extent by [40], who created a ConvDNN capable of predicting a movie's genre using pattern classification. Using an improved version of that algorithm, one could add movie posters to the already dense datasets used for content filtering. Two questions need to be answered concerning this:

1. Is it possible for movies to have "hidden" genres, besides their official ones and can they be predicted accurately simply by analyzing their posters? To answer this question, the classification algorithm should focus on a lower accuracy rate which allows for more unofficial genres to be inserted as a movie's poster-genres.
2. Could a RS be implemented in such a way that its recommendations fit a user's current mood? According to [41], color plays an important role in users choices and has been proven to represent and appeal to certain emotional states.

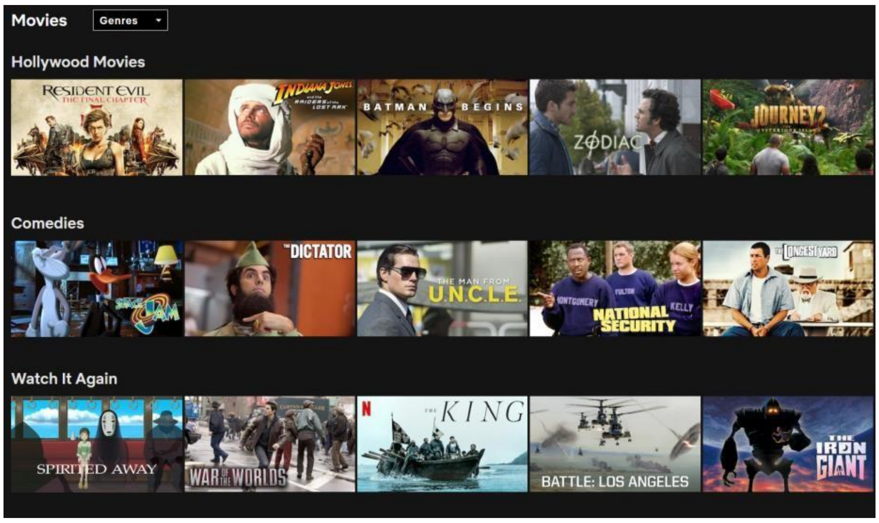

Figure 3: Netflix' interface. Screenshot taken on May 2020

Overall, recommender systems have evolved massively over the last few decades and, with the use of deep neural networks, they have already reached new heights, while still being based on the same fundamental concepts of data mining and machine learning, which revolve mainly around items. The only question that remains is how to alter those fundamentals in a way that revolves more around users and their ever changing needs.

ABBREVIATIONS

ANN	Artificial Neural Network
API	Application Programming Interface
CB	Content Based filtering
CF	Collaborative filtering
ConvDNN	Convolutional Deep Neural Network
CPU	Computer Processing Unit
DNN	Deep Neural Network
GPU	Graphics Processing Unit
LDNN	Linear Deep Neural Network
MAE	Mean Average Error
MF	Matrix Factorization
ONNX	Open Neural Network Exchange
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RS	Recommendation System
SVD	Singular Value Decomposition
TF-IDF	Term Frequency – Inverse Document Frequency

REFERENCES

[1] Miniwatts Marketing Group. (2020) Internet World Stats. [Online]. https://www.internetworldstats.com/stats.htm

[2] Rainer Bohme Stefan Korff, "Too Much Choice: End-User Privacy Decisions in the Context of Choice Proliferation," in *Symposium on Usable Privacy and Security (SOUPS)*, Menlo Park, CA, Germany, July 2014, pp. 69 - 87.

[3] Rainer Greifeneder, Peter M. Todd Benjamin Scheibehenne, "Can There Ever Be Too Many Options? A Meta-Analytic Review of Choice Overload," *Journal of Consumer Research*, vol. 37, pp. 409 - 425, Oct. 2010.

[4] Lior Rokach, Bracha Shapira, Paul B. Kantor Francesco Ricci, *Recommender Systems Handbook*. London: Springer New York Dordrecht Heidelberg, 2011.

[5] Marco de Gemmis, Giovanni Semeraro Pasquale Lops, "Content-Based Recommender Systems: State of the Art and Trends," *Recommender Systems Handbook*, pp. 73 - 105, Oct. 2011.

[6] J.B., Frankowski, D., Herlocker, J., Sen, S. Schafer, "Collaborative filtering Recommender Systems," *The Adaptive Web. Lecture Notes in Computer Science*, vol. 4321, pp. 291 - 324, 2007.

[7] T., Ricci, F. Mahmood, "Towards learning user-adaptive state models in a conversational recommender system.," in *15th Workshop on Adaptivity and User Modeling in Interactive Systems*, Halle, September 2007.

[8] D., Goker, M., McGinty, L., Smyth, B. Bridge, "Case-based recommender systems," *The Knowledge Engineering Review*, pp. 315-320, May 2006.

[9] Christina Yip Chung, Long-Ji Lim Wei Guan, "Collaborative-filtering Contextual Model Based on Explicit and Implicit Ratings for Recommending Items," US 7,590,616 B2, Sep. 15, 2009.

[10] Carlos A. Gomez-Uribe, Neil Hunt, Netflix, Inc., "The Netflix Recommender System: Algorithms, Business Value and Innovation," *ACM Transactions on Management Information Systems*, vol. 6, no. 4, pp. 1 - 19, Dec. 2015.

[11] Ali Yurekli, Alper Bilge, Cihan Kaleli Zeynep Batmaz, "A Review on deep learning for recommender systems: Challenges and Remedies," *Artificial Intelligence Review*, no. 52, pp. 1 - 37, Aug. 2018.

[12] G., Dlugolinsky, S., Bobák, M. et al. Nguyen, "Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey," *Artificial Intelligence Review*, no. 52, pp. 77 - 124, Jan. 2019.

[13] Gunawardana, Guy Shani and Asela, Microsoft Research, "Evaluating Recommendation Systems," *Recommender Systems Handbook*, pp. 257-297, 2011.

[14] YiBo Chen, "Solving the Sparsity Problem in Recommender Systems Using Association Retrieval," *Journal of Computers*, vol. 6, no. 9, pp. 1896 - 1902, Sep. 2011.

[15] Taghi M. Khoshgoftaar Xiaoyuan Su, "A Survey of Collaborative Filtering Techniques," *Advances in Artificial Intelligence*, vol. 2009, pp. 1 - 19, Oct. 2009.

[16] Sungchan Park, Woosung Jung, Sang-goo Lee Youngki Park, "Reversed CF: A fast collaborative filtering algorithm using a k-nearest neighbor graph," *Expert Systems with Applications*, vol. 42, no. 8, pp. 4022 - 4028, May 2015.

[17] Gunawardana Asela Shani Guy, "Evaluating Recommendation Systems," *Recommender Systems Handbook*, pp. 257 - 297, June 2014.

[18] Srujana Merugu George Thomas, "A scalable collaborative filtering framework based on co-clustering," in *Fifth IEEE International Conference on Data Mining (ICDM'05)*, Houston, TX, 2005, pp. 4 -.

[19] Adam Prügel-Bennett Mustansar Ali Ghazanfar, "Leveraging clustering approaches to solve the gray-sheep users problem in recommender systems," *Expert Systems with Applications*, vol. 41, no. 7, pp. 3261 - 3275, June 2014.

[20] I., Kaleli, C., Bilge, A. et al. Gunes, "Shilling attacks against recommender systems: a comprehensive survey," *Artificial Intelligence Review*, no. 42, pp. 767 – 799, Nov. 2012.

[21] European Parliament and the Council of the European Union, "Regulations - General Data Protection Regulation (GDPR)," *Official Journal of the European Union* , no. 119, pp. 1 - 88, May 2016. [Online]. https://gdpr-info.eu/

[22] Priyanka Wankar Sonali. B. Maind, "Research Paper on Basic of Artificial Neural Network," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 2, no. 1, pp. 96-99, January 2014.

[23] Michael L. Littman, Andrew W. Moore Leslie Pack Kaelbling, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, no. 4, pp. 237 - 285, 1996.

[24] NVIDIA. (2020) NVIDIA High Performance Computing. [Online]. https://developer.nvidia.com/hpc

[25] Schlegel Daniel, ""Deep Machine Learning on GPUs.," in *Seminar Talk-Deep Machine Learning on GPUs*, 2015.

[26] N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R Srivastava, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929-1958, 2014.

[27] Abadi et al. (2015) Large-Scale Machine Learning on Heterogeneous Systems. [Online]. https://www.tensorflow.org/overview

[28] Microsoft. (2016, Jan.) The Microsoft Cognitive Toolkit. [Online]. https://docs.microsoft.com/en-us/cognitive-toolkit/

[29] Yangqing Jia, Berkeley AI Research. Caffe, Deep Learning Framework by BAIR. [Online]. https://caffe.berkeleyvision.org/

[30] PyTorch team, Torch Contributors. (2019) PyTorch. [Online]. https://pytorch.org/features/

[31] Francois Chollet et al, Keras. (2015) Keras Documentation. [Online]. https://keras.io/

[32] Yehuda, Robert Bell, and Chris Volinsky Koren, "Matrix Factorization Techniques for Recommender Systems," *Computer*, no. 42, pp. 30 - 37, 2009.

[33] D., Park, C., Oh, J., Lee, S., & Yu, H. Kim, "Convolutional matrix factorization for document context-aware recommendation," in *Proceedings of the 10th ACM Conference on Recommender Systems*, 2016, pp. 233 - 240.

[34] F. Maxwell Harper, Joseph A. Konstan, "The MovieLens Datasets: History and Context," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 5, no. 4, pp. 1 - 19, December 2015, https://doi.org/10.1145/2827872. [Online]. https://grouplens.org/datasets/movielens/

[35] Safa M., Fancy C., Saranya D. Geetha G., "A Hybrid Approach using Collaborative filtering and Content based," *Journal of Physics: Conference Series*, vol. 1000, pp. 1 - 7, Apr. 2018.

[36] W., Yoshida, T., & Tang, X. Zhang, "A comparative study of TF* IDF, LSI and multi-words for text classification," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2758-2765, 2011.

[37] Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, Tat-Seng Chua Xiangnan He, "Neural Collaborative Filtering," in *International World Wide Web Conference*, Perth, Australia, 2017.

[38] Changyoung Park, Jinoh Oh, Sungyoung Lee, Hwanzo Yu Donghyun Kim, "Convolutional Matrix Factorization for Document Context-Aware Recommendation," in *RecSys '16: Proceedings of the 10th ACM Conference on Recommender Systems*, 2016, pp. 233-240.

[39] Daniel Kahneman, *Thinking, Fast and Slow*.: Penguin Books, 2011.

[40] Fagerholm Cecilia, "The use of colour in movie poster design: An analysis of four genres," Metropolia University of Applied Sciences, 2009.

[41] Guo H. J. Chu W. T., "Movie Genre Classification based on Poster Images with Deep Neural Networks," in *Workshop on Multimodal Understanding of Social, Affective and Subjective Attributes*, 2017, pp. 39-45.

[42] Menezes I. L., Tagmouti Y. Ho A. T., "E-mrs: Emotion-based movie recommender system," in *IADIS e-Commerce Conference*, Washington, 2006, pp. 1-8.