NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Deep Learning Techniques on Recommender Systems

Konstantina E. Stoikou

Supervisor: **Panagiotis Stamatopoulos,** Assistant Professor

ATHENS

SEPTEMBER 2021

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τεχνικές Βαθιάς Μάθησης σε Συστήματα Συστάσεων

Κωνσταντίνα Ε. Στόικου

Επιβλέπων: **Παναγιώτης Σταματόπουλος,** Επίκουρος Καθηγητής

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2021

BSc THESIS

Deep Learning Techniques on Recommender Systems

Konstantina Stoikou
S.N.: 1115201500151

SUPERVISOR: **Panagiotis Stamatopoulos,** Assistant Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τεχνικές Βαθιάς Μάθησης σε Συστήματα Συστάσεων

Κωνσταντίνα Στόικου
Α.Μ.: 1115201500151

ΕΠΙΒΛΕΠΩΝ: **Παναγιώτης Σταματόπουλος,** Επίκουρος Καθηγητής

ABSTRACT

A recommender system is a tool that filters information and suggests content to users which is relevant to their interests. Recommender systems have seen a rise in their use in the recent years as a result of the increasing internet use which provides researchers with huge amounts of user data. The purpose of this thesis is to study the various techniques that are applied to recommender systems as well as the deep learning models that are used to enhance those systems. Moreover, the evaluation methods of the recommender systems are described along with the challenges they face. Followingly, an implementation of a recommender system for video games which employs deep learning algorithms is provided followed by the interpretation of the results. At the end, some concerns and suggestions about the future in the field of recommendations are mentioned.

SUBJECT AREA: Recommender Systems

KEYWORDS: recommender system, recommendations, deep learning, machine learning, artificial intelligence, challenges, steam, games

ΠΕΡΙΛΗΨΗ

Ένα σύστημα συστάσεων είναι ένα εργαλείο που φιλτράρει πληροφορίες και προτείνει στους χρήστες περιεχόμενο που σχετίζεται με τα ενδιαφέροντά τους. Έχει παρατηρηθεί αύξηση στην χρήση των συστημάτων συστάσεων τα τελευταία χρόνια ως αποτέλεσμα της αυξανόμενης χρήσης του διαδικτύου η οποία παρέχει στους ερευνητές τεράστιες ποσότητες δεδομένων για τους χρήστες. Ο σκοπός αυτής της πτυχιακής εργασίας είναι να μελετήσει τις διάφορες τεχνικές που εφαρμόζονται στα συστήματα συστάσεων καθώς και τα μοντέλα βαθιάς μάθησης που χρησιμοποιούνται για την ενίσχυση αυτών των συστημάτων. Επιπλέον, οι μέθοδοι αξιολόγησης των συστημάτων συστάσεων περιγράφονται μαζί με τις προκλήσεις που αυτά αντιμετωπίζουν. Στην συνέχεια αυτής της μελέτης, περιγράφεται η υλοποίηση ενός συστήματος συστάσεων για βιντεοπαιχνίδια που χρησιμοποιεί αλγόριθμους βαθιάς μάθησης, ακολουθούμενη από την ερμηνεία των αποτελεσμάτων της. Στο τέλος παρουσιάζονται μερικά προβλήματα και προτάσεις που σχετίζονται με το μέλλον του χώρου των συστάσεων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Συστήματα Συστάσεων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: σύστημα συστάσεων, συστάσεις, βαθιά μάθηση, μηχανική μάθηση, τεχνητή νοημοσύνη, steam, παιχνίδια

ACKNOWLEDGEMENTS

I would like to thank prof. P. Stamatopoulos for supervising this thesis and its progress and contributing to its completion.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The rapid development of web services in recent years has led to an increase in the information that internet users come across everyday. In 2008, the average person was estimated to consume 34GB of information per day [1]. Therefore, it seems impossible for a user to choose the most suitable content for them by considering all the options that are available to them. Both companies and users desire to offer or get personalized suggestions respectively, in order to improve their online experience. The former want to increase sales and user engagement by displaying content to a targeted audience, while the latter want to save time and minimize their effort of searching among the vast options offered by an online service.

A recommender system (RS) is the modern solution to solving these problems. It is an intelligent system that filters the offered content or services and presents users with options that most closely match their preferences. Companies have already seen the value in the use of recommender systems. In 2006, Netflix announced the ”Netflix Prize” competition that was awarding 1,000,000 USD to the team with an algorithm that would be at least 10% more accurate than the company’s. Nowadays, recommender systems are used in e-commerce, online learning platforms, social networking websites, even in medicine to help healthcare professionals improve their decision-making process [2].

Recently, deep learning has gained attention due to the increase in the computational processing power, in the data storage capabilities and in the data volumes which usually originate from the web users. Deep learning models for RSs are able to achieve a better feature learning of the interaction between items and users and exploit contextual information from heterogeneous data sources. Researchers have already seen the potential in applying deep learning techniques to predict user preferences.

The purpose of this study is to provide and review the different RS techniques according to their recommendation approach and survey how deep learning models are applied on RSs. Also the challenges that RSs face will be mentioned along with proposed solutions. Finally, some algorithms are implemented to produce video game recommendations.

This thesis is structured in the following fashion: section 2 introduces the several categories of RSs, section 3 describes the different deep learning models that are mostly used in RSs, section 4 deals with the problems and challenges RSs face and their possible solutions, section 5 describes the experiment on the video game recommendations along with its results and section 6 includes future approaches concerning the RSs.

2. RECOMMENDER SYSTEM

As already mentioned, a recommender system is a means to assist people in their decision-making. The use of such systems attempts to solve the problem of information overload that occurs increasingly, especially on web applications. A typical RS includes some characteristics that need to be defined.

- **Usage context:** it includes the contextual factors that can affect the how and why an item is recommended or presented to the user. For example, a RS for music might suggest songs based on the popularity of the song but this cannot apply to RSs that recommend medicines to patients because some popular medicines are dangerous for people with a certain medical history.
- **Users:** the end-users to whom the system tries to provide suggestions that match their preferences.
- **Items:** the content or services that are provided to a user and will be rated or recommended by the system.

2.1 Recommendation process

There are usually three phases in the recommendation process before an item is recommended to a user [3].

2.1.1 Phase 1: Information collection

Gathering relevant user information is crucial for preference prediction tasks. The more data the system has for a user profile, the more accurate predictions it will provide. That data usually originates either from explicit feedback required from the user concerning their interest in an item or from implicit feedback by following user behavior and speculating on the user's inclination to certain items. In some systems a combination of the two is employed which results in a hybrid feedback strategy.

- **Explicit feedback:** the user is asked from the system to rate an item that they have already experienced (bought, used, watched etc.). This type of feedback, although it requires some effort from the user, it collects more reliable information since it does not infer the likings of a user.
- **Implicit feedback:** the preferences are deduced by the system from the actions of a user such as clicks, history of content/services used, duration of web page visits etc. It does not require user effort but the recommendations of the system might not be approved by the user since they are inferred rather than stated. However, there is an argument that indirect collection of data might be more accurate since there is no influence from the socially approved behaviors that a user might be biased towards.
- **Hybrid feedback:** many systems use a combination of the two aforementioned methods to leverage the advantages of both. In this way, the system can still provide suggestions even if the user has declined to provide explicit feedback or in case they do provide it, implicit feedback can be used to verify the preferences that were stated in the explicit one.

2.1.2 Phase 2: Learning

An algorithm is used to learn the user's preferences from the data that were previously collected.

2.1.3 Phase 3: Prediction/recommendation

The system predicts the items that might correspond to the user interests or it ranks them based on the relevance to those interests.

2.2 Recommendation system techniques

The recommendation problem is usually defined either as the problem of predicting whether a user will like a certain item or ranking items based on the user preferences. According to the recommendation approach and the information they use, RS have been classified into multiple categories. The three major categories are collaborative filtering, content-based and hybrid systems. Besides those there are various other techniques such as graph-based, knowledge-based, demographic-based and community-based.

2.2.1 Collaborative filtering

A collaborative filtering (CF) based system recommends items to users based on the interests of similar users. It depends on the previous user-item interactions and after finding like-minded users, it exploits their interests or ratings to recommend the same items to the target user. This technique is the most commonly used and it is usually employed in applications that might include music, movie or restaurant recommendations. CF algorithms are divided into two main subcategories:

- **Memory-based:** use nearest-neighbor algorithms to find similar entities in the user-item ratings matrix. In user-based approaches, also called *user-item filtering*, an item is recommended to a user if similar users have liked it. In item-based approaches, also called *item-item filtering*, based on a target item, the system finds users who liked that item and then it finds other items that those users also liked [4]. A list is created with the top-N of those items based on relevance.
- **Model-based:** CF models are built based on machine learning or data mining algorithms. Those algorithms include Bayesian models, clustering, decision trees, SVD, Matrix Factorization, Multi-criteria Recommender Systems etc. [5]

2.2.2 Content-based

Content-based (CB) systems take into consideration the prior preferences of a user and recommend items that are similar to the ones the user previously liked. Systems that use CB techniques exploit the item's descriptive information and that's why they are suitable for text-intensive domains where keywords are important such as e-learning [6]. CB methods offer user independence since user ratings are used for recommendations only for the same user and they are also very convenient when a new item is introduced because instead of depending on item ratings, the algorithm takes advantage of the descriptive data of the item.

2.2.3 Knowledge-based

Knowledge-based systems make use of the explicitly stated interests of the users to create recommendations. In this way, the system knows whether a specific item corresponds to a specific user's preferences because it takes into consideration and compares the user-defined needs and the item attributes. Such techniques are beneficial in domains where ratings are pretty limited such as luxurious cars, houses etc. or where it is essential for a user to specify some requirements such as allergies in specific products in a food recommendation domain. This method takes into account the changing user preferences and the system can create accurate recommendations even for new users. On the other hand, creating and updating a knowledge base requires expertise and such systems usually produce static recommendations if the users interests are constant over time.

2.2.4 Graph-based

A graph-based system constructs a graph where each node represents a user or an item and each edge is the interaction between users or between user and item. After the construction of such a graph model, graph analysis is employed to generate recommendations. The advantage is that it is easy to find similar users or items and items that have been rated or used by users. Also, adding new users and items is easy since it requires only the insertion of the corresponding nodes and edges. However, where this technique lacks is in discovering new items in the graph.

2.2.5 Demographic-based

A demographic-based recommendation system looks into the demographic attributes of a user to make recommendations such as the age, the country, the gender, the profession etc. This technique is based on stereotypes because the main idea behind it is that people who share similar demographic characteristics will have similar interests. Additionally to the negative aspects, extracting demographic details of a user might be against privacy laws. Such a technique is normally combined with other techniques such as knowledge-based ones to produce more accurate predictions.

2.2.6 Community-based

A community-based system exploits the friendship and trust between users. It bases the recommendations on the interests and rankings of users in the social circle of a certain user. This method benefits from social networks since typically there is abundant data about those relations.

2.2.7 Session-based

A session-based system depends on the user's recent information related to their behavior within the current session. This technique solves the problem when the system lacks historical data on the user's preferences. Session clicks and content information, such as item descriptions, can be integrated in the system combined with deep learning approaches to improve performance.

2.2.8 Hybrid

A hybrid system combines more than one recommendation approach to benefit from the advantages of them and minimize the shortcomings. To give an example, new items in the system can drive the recommender system to use a CB algorithm, while new users will trigger the CF algorithms. The several hybridization techniques that are mostly used are the following:

- **Weighted:** the output scores of the different recommendation algorithms are combined by a weighted linear formula to generate the recommendations or predictions. The advantage of this is that all the recommendation methods are being used each time.
- **Switching:** the system selects one of the recommendation methods to use depending on the situation. For example, for a new user it would swap to a collaborative filtering algorithm for suggesting items. The advantage is that the system can avoid the weaknesses of a specific recommendation method and exploit another one. The disadvantage lies in the fact that the system has to decide when to switch the method and as a result the number of parameters increases and so does the complexity.
- **Mixed:** instead of producing one recommendation per item, the results of all recommendation methods are combined and form a list. It is the simplest hybridization technique.
- **Feature combination:** the features produced by one recommendation algorithm are fed into the algorithm of the other recommendation method as supplementary feature data.
- **Feature augmentation :** the predictions or ratings of one of the recommendation algorithms are used as an input for another algorithm. This technique is order-sensitive since the input of the second algorithm depends on the output of the first.
- **Cascade:** one of the recommendation algorithms is applied first to produce a rough list of recommendations or rankings. Then a second algorithm is employed to refine that list. This technique is also order-sensitive.
- **Meta-level:** the first recommendation algorithm generates an internal model which is passed as an input to the second algorithm. Again, this technique is order-sensitive.

2.3 Social recommender systems

An interesting subcategory of recommender systems based on the domain that they are employed into are social recommender systems (SRS). In recent years, the vast use of social network applications has offered rich sources of data that were not available before. The users of such platforms disclose information about their interests, their occupation and they connect with friends or like pages with content similar to their interests, they comment and react on posts, tag their friends or rate content. All this information can be used by RSs to increase prediction accuracy. Social networks can employ SRSs that make use of the users' data in order to promote ads, recommend new friends or suggest new communities to join. SRSs can be classified based on the distinctive features that are leveraged to build the system. Those features are the following:

- **Context:** recommendations are based on the user's contextual information such as time, location, mood etc. Such recommendations might be relevant for a short time period before the user's context changes.
- **Trust:** the system detects genuine and malicious users by the level they behave in an expected way. A trust network is created from users that have formed trust relationships between them and then the SRS is generating recommendations based on the ratings and preferences of the trusted users.
- **Tag:** content recommendations are based on the user's tagging information and behavior to find similar users and similar items. Personalized tags can be also recommended to users by always taking into account the current popularity of the tags.
- **Group:** identical recommendations are given to a particular group of users with certain similarities, such as users that have all attended a specific event or who work on the same project. Sometimes, users of social networks can join communities to become part of a group without the need of an automatic grouping by an algorithm, while in other cases an algorithm is used to group users according to their data and relations. In this category also fall the SRSs which recommend groups to users.
- **Cross social media:** information about the user from different social network platforms is used in order to enhance the recommendations. After a user is identified in another platform, the knowledge about their item interactions and ratings can be transferred to another platform.
- **Temporal dynamics:** user preferences change over time as well as their relations with other users such as their friendships. The system will consider those changes and make suggestions based on the new and updated data.
- **Heterogeneous social connections:** recommendations are based on the different types of relations between users. For example, a user might trust a specific network to get similar recommendations for attending a sports event and a different network for watching movies. Also, relations with negative influence, which means non-trustworthy relations, are also weighted in the RS.
- **Semantic filtering:** actual entities are identified from textual data and they are used to make recommendations that match long-lasting preferences of users. This way insignificant information is filtered out so that the meaningful relations will be harnessed.

2.4 Evaluation metrics

It is important to define how we measure the quality of a RS. The metrics have to do with accuracy or coverage. *Accuracy* is the fraction between correct recommendations and all possible recommendations. *Coverage* is the percentage of the items and users in the search space which the system is able to recommend. Specifically the metrics are the following [7]:

2.4.1 Statistical accuracy metrics

The statistical accuracy metrics, also called *rating prediction metrics*, compare the predicted ratings with the real user ratings. These metrics evaluate the correctness of a recommendation algorithm based on its error. The metrics used are Mean Square Error (MSE), Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). Since the error formulas are calculating the difference between predicted and actual user ratings, a lower value implies better accuracy.

2.4.2 Decision support accuracy metrics

Also called *classification accuracy metrics*. These metrics measure the ability of the RS to classify items according to how they match the user preferences. Metrics that are used in this category are Reversal rate, Weighted errors, Receiver Operating Characteristics, Precision Recall Curve, Precision, Recall and F1-score. For example, precision is the percentage of items that were recommended to the user which are relevant to their interests towards the total number of recommended items.

2.4.3 Ranking metrics

These metrics evaluate the RS based on the recommendations they generate from a significantly ordered list of items. Ranking metrics for deep learning-based RS include Normalized-discounted-cumulative gains, Hit Ratio, Mean Reciprocals Ranks and Mean Average Precision. The last one, for example, is evaluating the precision of the top-K ranked recommended items.

3. DEEP LEARNING FOR RECOMMENDER SYSTEMS

Deep Learning (DL) is a subclass of machine learning and it uses artificial neural networks of three or more layers in hierarchical order to simulate the human brain behavior. The key points that have encouraged the use of deep learning architectures in many domains are the huge amounts of data, or commonly called big data and the modern infrastructure that can store this data, since deep learning algorithms learn better when there is more data fed to them. Also the recent development in the GPUs increases the computational power which is needed for the complex computations of those algorithms [8]. Deep models have been exploited to solve scalability problems because they offer dimensionality reduction and feature extraction. Deep neural networks are excellently performing supervised and unsupervised tasks. In supervised learning the algorithm learns the labeled data as it iterates through them to make predictions and it adjusts to fit the model correctly. In unsupervised learning the algorithm works on its own to discover hidden patterns and cluster unlabelled data. Some human interaction is still needed for validating the output and whether the clustering makes sense [9]. The models that are typically used by researchers in RSs will be discussed in the following paragraphs.

3.1 Autoencoders

An Autoencoder (AE) is a feedforward network which is used for unsupervised learning tasks. Feedforward means that the data moves into a forward direction from the input nodes without any cycles. An AE has three layers: the input, the hidden and the output layer, as seen in Fig. 3.1. This model is used for learning dimensionality reduction of the dataset. The purpose is to encode the input into a more compact representation and then decode it to reconstruct the input data into the output. The hidden layer which is placed between the input and output layers contains the compressed representations of the data. The compressed data is considered corrupted.

The math behind the AEs is quite easy to understand. The encoder and decoder are defined by the transitions ϕ and ψ respectively, such that:

$$\phi : X \to F$$
$$\psi : F \to X$$
$$\phi, \psi : argmin_{\phi,\psi} \|X - (\psi \circ \phi)X\|^2$$

In the simplest scenario, given one hidden layer, the above definitions mean that the encoder function (ϕ) takes the input $x \,\epsilon\, \mathbb{R}^d = X$ which represents the original data and maps it to the latent vector $h \,\epsilon\, \mathbb{R}^p = F$. The encoding network can be represented by:

$$h = \sigma(Wx + b)$$

where σ is the activation function e.g. sigmoid or Rectified Linear Unit (ReLU), W is a weight matrix and b is a bias vector. W and b are usually randomly initialized and then adjusted through backpropagation during training.

Conversely, the decoding function (ψ) maps h to x' which is the reconstruction of the input vector x. Both x and x' have the same shape. The decoding network can be represented by:

$$x' = \sigma'(W'h + b')$$

where σ', W' and b' might differ from the σ, W and b of the encoder.

The autoencoder are trained by minimizing the reconstruction error (such as squared error) which is also called the *loss* [10, 11]:

$$L(x, x') = \|x - x'\|^2 = \|x - \sigma'(W'(\sigma(Wx + b)) + b'\|^2$$

Sometimes, the AE might fail to select and extract useful features and as result the input might equal the output and overfit the training dataset [12]. To overcome this problem the Denoising Autoencoder (DAE) was introduced. A DAE uses the input data after they have been corrupted by adding noises to the input vector. The DAE is trained to produce the original uncorrupted data in the output. This way, overfitting is prevented. Multiple DAEs can be stacked on top of each other to compose a Stacked Denoising Autoencoder (SDAE) which can be useful in extracting more hidden features.

AEs are great in learning feature representations and that is a reason why they are used in RSs. They can improve the accuracy of an RS by providing a better user and item representation learning. DAEs are also useful in predicting missing values from corrupted data in an RS. SDAEs, on their part, can discover a denser form of the matrix of the input data and they can merge additional helpful information in the RS from different data sources. Stacked AEs are better than simple AEs because stacking is better at feature learning.

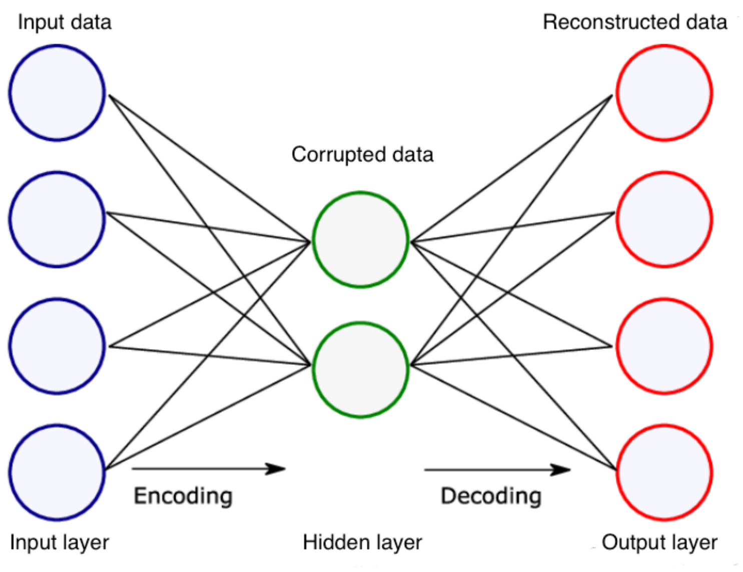


Figure 3.1: An autoencoder

3.2 Restricted Boltzmann machines

A Restricted Boltzmann Machine (RBM) has two layers of units, the first consists of the visible units and the second of the hidden units. As illustrated in Fig. 3.2, the structure resembles a bipartite graph since there are no connections between units of the same layer. For example, for images with handwritten digits, a visible unit represents one pixel and a hidden unit represents a dependency between pixels. RBMs can be used for both supervised and unsupervised learning.

The hidden and visible units of an RBM are boolean. The RBM consists of W which similarly to AEs is a weight matrix of size $m \times n$. Each connection between a visible unit v_i and a hidden unit h_j is associated with a weight element $w_{i,j}$ of W. Also, there is a bias weight a_i and b_j for v_i and h_j respectively. The *energy* of a pair of boolean vectors (also called *configuration*) v, h is the following:

$$E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j$$

The marginal probability of a visible vector v is the sum of the joint probability distribution for the visible and hidden vectors over all the possible hidden layer configurations. Specifically:

$$P(v) = \frac{1}{Z} \sum_{\{h\}} e^{-E(u,h)}$$

where Z is a partition function, defined as the sum of $e^{-E(u,h)}$ over all possible configurations, which ensures that the probabilities sum to 1.

The RBMs are trained by maximizing the expected log probability of the random sample v from the training set V:

$$argmax_W \, \mathbb{E} \left[log \, P(v) \right]$$

The algorithm that is mostly used for optimizing the weight matrix W is called Contrastive Divergence (CD) which performs Gibbs sampling and uses a gradient descent procedure to update the weights similarly to how backpropagation is used in feedforward neural networks such as the AE [13].

RBMs can be exploited to improve the performance of an RS. Specifically, they can be used to extract latent features of the user's preferences or of an item's ratings. Additionally, they are useful for creating joint models of both the correlations between the items a user has rated and the correlation between the users who rated a specific item. As a result of this modeling, the accuracy of an RS can be improved. A similar modeling can be used in group recommendations by modeling collectively the features and profiles of the group. RBMs also allow the integration of additional data from various sources. RBMs are inferior in the accuracy of the user preferences compared to an AE because RBMs produce predictions using the log-likelihood maximization function while AE minimizes the squared error.

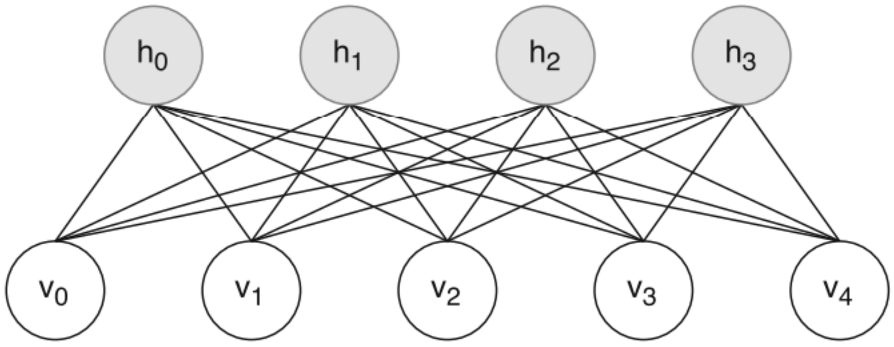


Figure 3.2: A restricted Boltzmann machine

3.3 Deep belief networks

A Deep Belief Network (DBN) is a multilayer architecture that consists of stacked RBMs. Specifically, the hidden layer of a subnetwork works as the visible layer for the next subnetwork. The original data are passed into the visible units of the bottom layer, then the parameters are adjusted and the hidden units of the first RBM are used as input for the second RBM as shown in Fig. 3.3. The learning process continues up to the top of the stacked RBMs. This process is unsupervised, so usually a new supervised learning network is added to the last layer to execute tasks such as classification or regression.

Specifically, the training process is the following. Each RBM starts training only after the previous RBM is completely trained, which means the training algorithm is greedy. The weights W learned by a RBM define both $P(v|h, W)$ and the prior distribution over the hidden vectors $P(h|W)$, where h is a sampled vector from $P(h|v, W)$ that will be used as input data for the next stacked RBM. So, the probability of generating a visible vector v is:

$$P(v) = \sum_{h} P(h|W)P(v|h, W)$$

After learning the weights W, the $P(h|W)$ is replaced by a better model of the aggregated posterior distribution over the hidden vectors. This model is learned by using the hidden vectors as training data for the next RBM [14].

In RSs, DBNs are utilized to extract high-level features on user preferences from low-level features. Besides this, DBNs can be used for classification,usually on text and audio data, for the purpose of analyzing user preferences.

3.4 Generative adversarial networks

A Generative Adversarial Network (GAN) consists of two neural networks: the generator and the discriminator. The former generates new data instances from random variables. The latter approximates the probability that a sample data comes from the real training data instead of the generator output. This is an iterative process so that both the generator and the discriminator improve on their respective jobs each time.

The loss function of the GAN is derived from the binary cross-entropy loss which can be written as:

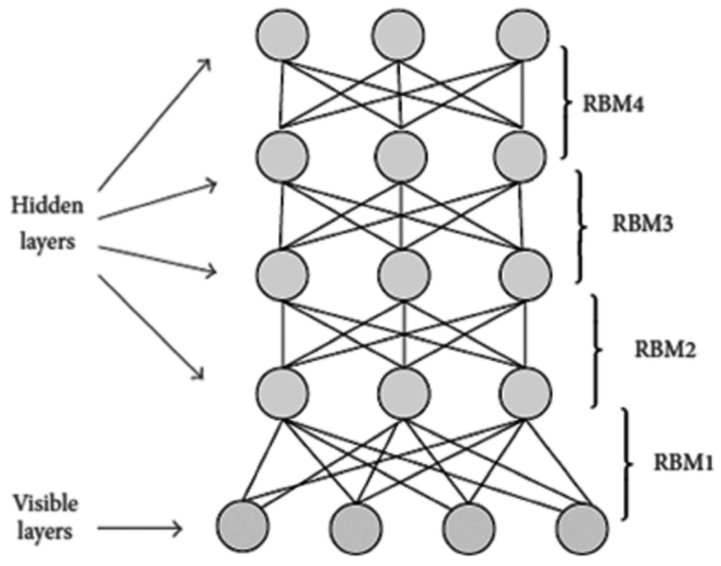


Figure 3.3: A deep belief network

$$L(\hat{y}, y) = [y \cdot log\hat{y} + (1 - y) \cdot log(1 - \hat{y})]$$

where y is the original data and $\hat{y}$ is the reconstructed data. For the training of the discriminator (D), the label of the data from the original data distribution $P_{data}(x)$ is $y = 1$ because the data are real, and $\hat{y} = D(x)$. If we substitute this in the above loss function, the result is:

$$L(D(x), 1) = log(D(x))$$

and for the generator (G), the label is $y = 0$ because the data are fake and $\hat{y} = D(G(z))$, where z is the noise vector. So:

$$L(D(G(z)), 0) = log(1 - D(G(z)))$$

As mentioned previously, the purpose of the discriminator is to classify the real and fake data correctly. So, the above loss functions should be maximized and the final loss function for the discrimination becomes:

$$L^{(D)} = max[log(D(x)) + log(1 - D(G(z)))]$$

The generator is competing against the discriminator, so it will try to minimize the above function:

$$L^{(G)} = min[log(D(x)) + log(1 - D(G(z)))]$$

If the loss functions of the discriminator and the generator are combined we get the loss function of the GAN for a single data point which is the following:

$$L = min_G\, max_D[log(D(x)) + log(1 - D(G(z)))]$$

The loss function for the entire dataset derives if we get the expectation of the above equation [15, 16]. So, in the end:

$$min_G\, max_D \mathbb{V}(D, G) = min_G\, max_D(\mathbb{E}_{x \sim P_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim P_z(z)})[log(1 - D(G(z)))])$$

RSs benefit from GANs due to the fact that there is no need for negative samples in the training dataset. Also, they can provide a distribution of ratings for all the options each user has [17].

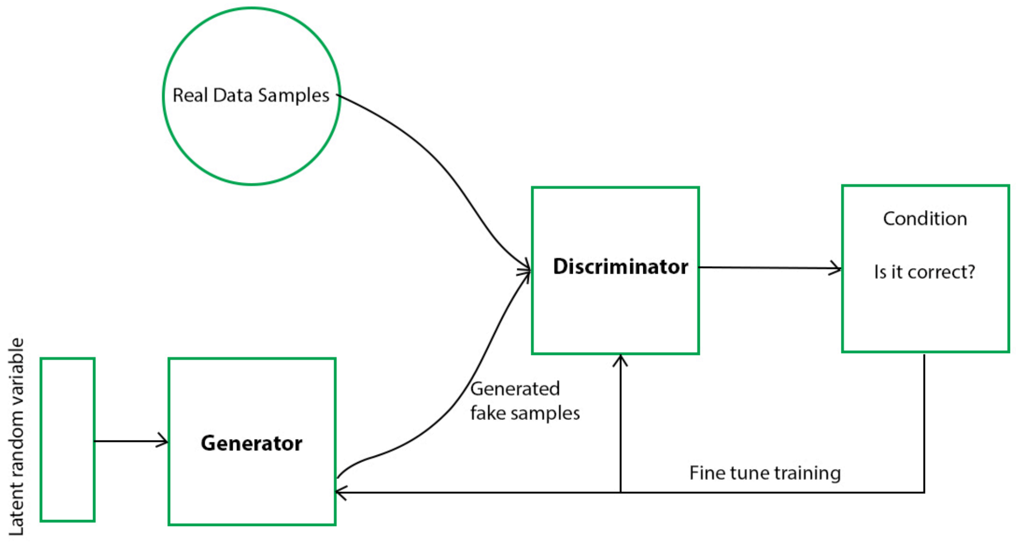


Figure 3.4: A generative adversarial network

3.5 Convolutional neural networks

A Convolutional Neural Network (CNN) is inspired by the organization of the animal visual cortex. It contains at least five layers: input, convolutional, pooling, fully connected and output layer. The convolutional layer applies convolutional operation to extract features, also called feature maps, from the input data. To introduce nonlinearity to the data, a nonlinear activation function is used, such as ReLU. The pooling operation subsamples the output of the previous convolutional layer, which means it reduces the features to improve computational time. The convolutional and pooling layers are stacked together in pairs. The fully connected layer takes the high level features of the last pooling layer and uses them for classification. Backpropagation is also employed to adjust the weights and improve accuracy.

CNNs are usually used with images as inputs. Let's see the mathematical representation of a CNN. An image can be represented as:

$$dim(image) = (n_H, n_W, n_C)$$

where n_H and n_W are the size of height and width respectively and n_C the number of channels (e.g. for RBG images: $n_C = 3$). The kernel K (also called *filter*) has the same number of channels, is squared and has an odd dimension f. So the dimensions of the filter are:

$$dim(filter) = (f, f, n_C)$$

CNNs are based on the convolution product operation which results in a two-dimensional matrix where each element is the sum of the elementwise multiplication of the filter and the subcube of an image. Specifically, for an image I and a filter K:

$$conv(I, K)_{x,y} = \sum_{i=1}^{n_H} \sum_{j=1}^{n_W} \sum_{k=1}^{n_C} K_{i,j,k} I_{x+i-1,y+j-1,k}$$

and the resulted dimensions this operation are:

$$dim(conv(I, K)) = (\lfloor \frac{n_H + 2p - f}{s} + 1 \rfloor, \lfloor \frac{n_W + 2p - f}{s} + 1 \rfloor) \ , \ s > 0$$
$$dim(conv(I, K)) = (n_H + 2p - f, n_W + 2p - f) \ , \ s = 0$$

where s is the stride and p is the padding. The pooling operation is downsampling the image's features. This operation is applied to each channel so it affects only the n_H and n_W dimensions and not the n_C. Given an image and a filter f:

$$dim(pooling(image)) = (\lfloor \frac{n_H + 2p - f}{s} + 1 \rfloor, \lfloor \frac{n_W + 2p - f}{s} + 1 \rfloor, n_C) \ , \ s > 0$$
$$dim(conv(I, K)) = (n_H + 2p - f, n_W + 2p - f, n_C) \ , \ s = 0$$

The CNN is learning by performing forward propagation and evaluating the function:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y_i}^{\theta}, y_i)$$

where m is the size of the training set, θ the model parameters and L the cost function which calculates the distance between the real and predicted values on a single data point. CNN is also performing back propagation and applies a descent method (such as gradient descent) to update the parameters θ [18].

The advantage of CNNs is the reduction in the training data dimensionality. However, the need for a large amount of hyperparameter tuning is considered a disadvantage. CNNs are effective when employed in an RS because they can extract the semantic meaning of text which is valuable especially in context aware systems. CNNs are mostly leveraged for drawing out latent factors and features, usually from images or textual data.

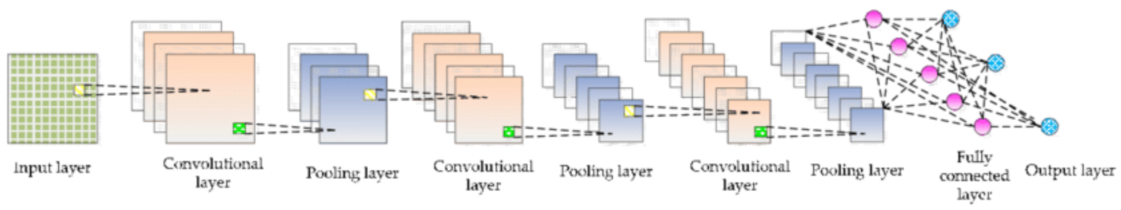


Figure 3.5: A convolutional neural network

3.6 Recurrent neural networks

A Recurrent Neural Network (RNN) is convenient for sequential data such as data from time series or sound. Unlike feedforward networks, RNNs have feedback loops and internal memory that remembers information about the previous steps. That information is actually the hidden state which specifies the previous classifications. In each next step, the previous and next hidden states are combined to get a new classification.

If we symbolize the state of the memory of the RNN at time t as h_t and the output at time t as y_t then:

$$h_t = \phi(U \cdot x_t + W \cdot h_{t-1})$$
$$y_t = \psi(V \cdot h_t)$$

where h_{-1} is randomly initialized, ϕ and ψ are non-linear functions and x_t is the input at time t. U, W and V are parameters of the linear regressions that are preceding the non-linear functions. Those parameters are the same in the whole architecture.

The training is executed by performing forward propagation which evaluates the function:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{N_y^{(i)}} L(\hat{y_t}^{\theta}, y_t^{(i)})$$

where L is the cost function which calculates the distance between the real and the predicted value on a single data point, $y^{(i)}$ is the i-th output sequence, $N_y^{(i)}$ is the length of $y^{(i)}$, $y_t^{(i)}$ is the t-th element of the output $y^{(i)}$ and θ are the model parameters. The RNN also performs backpropagation through time to update the θ [19].

The problems of RNN models are the vanishing gradient, which prevents the weights from changing their values, and the exploding gradient, which results in very large updates to the weights. These problems can be overcome by variations of RNNs such as Gated Recurrent Unit (GRU) and LSTM (Long Short Term Memory). Another shortcoming is the high amount of parameter tuning which increases the computation time.

RNNs can take into account the browsing history of a user and the order of sequence of their actions for the sake of improving RS accuracy. RNNs have also been used to combine latent factors of user preferences with representations of temporal, contextual features of user's behaviors. RNN models are capable of representing, in a non linear way, how users' and items' latent features influence each other. If the problem of the evolution of the user preferences is transformed into a sequence prediction problem, a RNN can be utilized to enhance the RS's predictions of this evolution. Because of these reasons, RNNs are very effective on short-term predictions and they can prove advantageous in session-based systems.

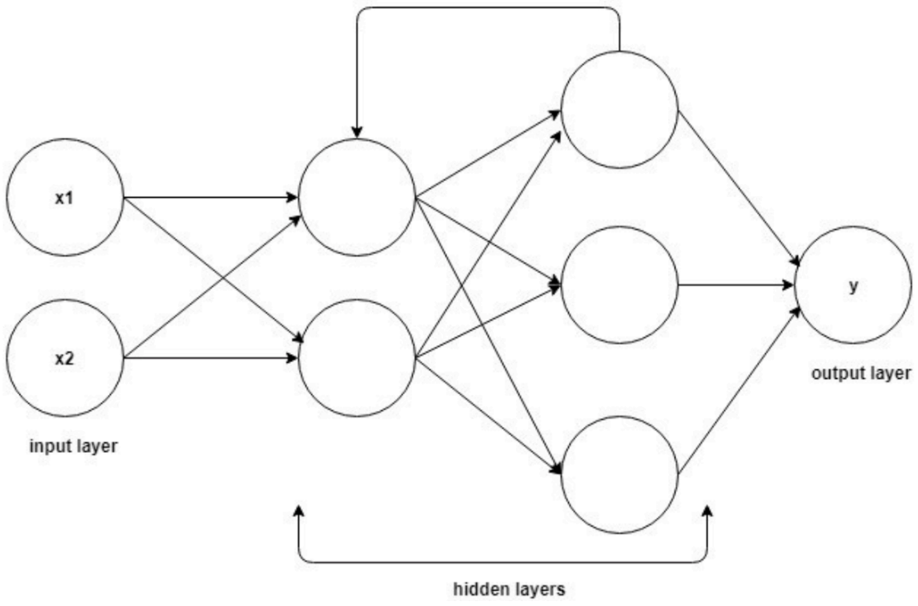


Figure 3.6: A recurrent neural network

3.7 Multilayer perceptrons

A Multilayer Perceptron (MLP) is considered the simplest deep learning architecture. It is a feedforward neural network which consists of at least three layers: the input layer, the hidden layers, which can be multiple, and the output layer. This structure can be seen in Fig. 3.7. Except for the input nodes, each node is considered a neuron, also called perceptron, that uses a non-linear activation function [20]. MLPs use supervised learning by applying backpropagation to adjust the weights and biases to minimize the error (usually the MSE).

A MLP with two hidden layers can be expressed mathematically as:

$$h_i^{(1)} = \phi^{(1)}(\sum_j w_{ij}^{(1)} x_j + b_i^{(1)})$$
$$h_i^{(2)} = \phi^{(2)}(\sum_j w_{ij}^{(2)} h_j^{(1)} + b_i^{(2)})$$
$$y_i = \phi^{(3)}(\sum_j w_{ij}^{(3)} h_j^{(2)} + b_i^{(3)})$$

where $h_i^{(l)}$ are the units in the l-th hidden layer, x_j are the activations of the input units, y is the activation of the output unit, $\phi^{(1)}$ and $\phi^{(2)}$ are the activation functions which can differ since different layers have different activation functions. Each unit has its own bias b_i and each pair of units in two consecutive layers has its own weight w_{ij} [21]. The vectorized form of the above computations is:

$$h^{(1)} = \phi^{(1)}(W^{(1)}x + b^{(1)})$$
$$h^{(2)} = \phi^{(2)}(W^{(2)}h^{(1)} + b^{(2)})$$
$$y = \phi^{(3)}(W^{(3)}h^{(2)} + b^{(3)})$$

MLPs are used to transform the linearity of RSs to non linear models that can be used in neural networks, which means MLPs can model user and item information with respect to their correlation and with close to zero amount of linearities.

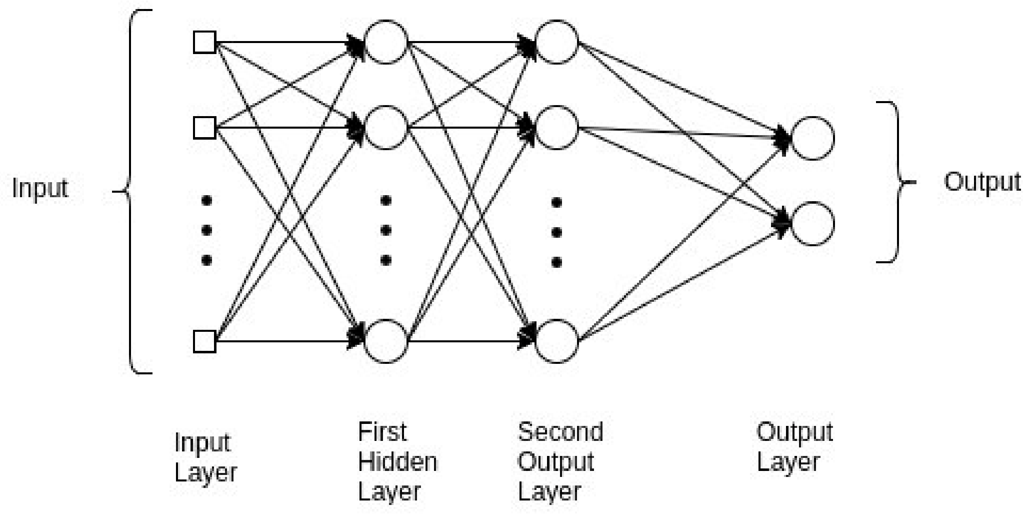


Figure 3.7: A multilayer perceptron

3.8 Attention networks

An attention network tries to imitate the human cognitive attention. It highlights the important subsets of features of the input data and de-emphasizes the rest of them. Which features require attention is learned usually by using gradient descent on the training data. By separating the important features the network will reserve more computation power for that subset instead of the whole set of features [22].

By using attention, a context vector c_i is produced based on the hidden states $s_1, ..., s_m$ that are used together along with the current hidden state h_i:

$$c_i = \sum_j a_{ij} s_j$$
$$a_i = softmax(f_{att}(h_i, s_j))$$

where f_{att} is the attention function which calculates an unnormalized alignment score between the current hidden state (h_i) and the previous hidden state (s_j). For example, f_{att} can be $v^T tanh(W[h_i; s_j])$ which represents addictive attention. v and W are learned attention parameters [23, 24].

Attention is often combined with deep learning models CNN, MLP or RNN instead of being a standalone technique. Some RSs use attention on the review level to filter the textual features which are not highly informative in an item review. For item-based CF approaches, attention is employed to define which items in the history of a user's profile are more considerable for predictions.

3.9 Hybrid architecture

Different deep learning methods can be combined in order to leverage the advantages of each method and improve the RS's performance and efficiency. An example of the above is a combination of DBN and RNN which extracts valuable comments and uses a shared layer to succeed in rating prediction [25].

4. CHALLENGES OF RECOMMENDER SYSTEMS

There are some challenges that need to be addressed when building a RS in the interest of making it as useful as possible to the users and achieving the goals of the application that uses them. The most important and frequent problems will be discussed in this section.

4.1 Accuracy

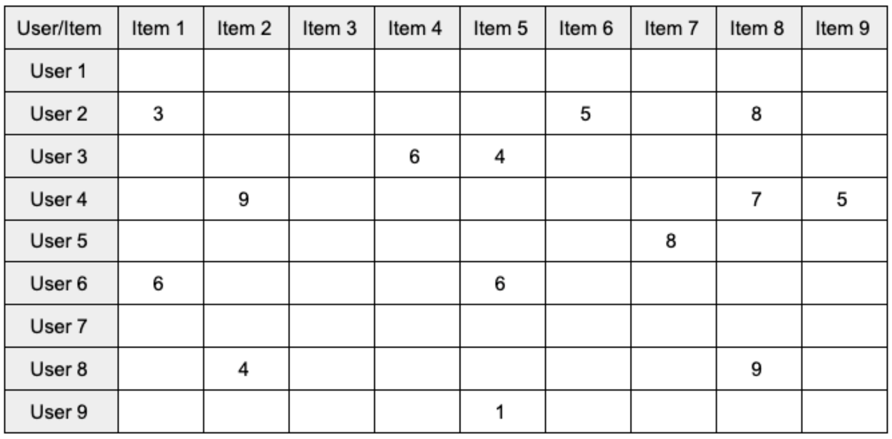
The first and foremost challenge for RSs is accuracy. As mentioned before, accuracy is used as a metric for evaluating a RS, so researchers continuously try to find ways to improve it as much as possible. The measures to evaluate the objective accuracy are usually error formulas such as MSE. Deep learning techniques are successfully achieving high accuracy results. This happens due to the ability of deep learning models to extract hidden features of users and items, to create joint models and to combine additional information from different data sources as stated in the previous chapter. However, besides objective accuracy, the user's perception of accuracy is also important. To elaborate on that, it means the extent at which the user considers that the generated recommendations meet their interests. If the user believes the recommendations are accurate, their trust in the system and consequently in the recommended items is influenced positively. Although at some level perceived accuracy is correlated with objective accuracy, displaying information about how the system makes recommendations and designing a better user interface can increase user's perceived accuracy [26].

4.2 Cold-start

Another major problem of some RSs is cold-start. It is the situation where a new user or a new item is introduced to the system and there is no adequate information to make reliable recommendations on them. When a new user is entered into the system, there is no data about their preferences and therefore they can not get proper suggestions. This type of user is called a grey sheep. When a new item is added to the system, there are no ratings provided for this item and as a result the system does not know which users should receive this item as a recommendation [27]. Cross domain recommendation can solve the problem about new users as the system can utilize knowledge about the user which was learned from different domains. Another solution are knowledge-based methods which require the new user to explicitly define their interests. CB systems can provide a solution for cold-start issues related to new items. They take advantage of the item's descriptive data instead of relying on its ratings from other users.

4.3 Data sparsity

Data sparsity is a problem closely related to cold-start. Users usually rate only a small fraction of the available items which leads to a sparse user-item rating matrix as seen in Fig. 4.1. As a result of the lack of an adequate amount of data, the system fails to recognize similar users or items. This also raises the coverage issue since a percentage of the items in the system will not be recommended to any user. Data sparsity occurs especially on CF RSs because they are based on similar users' ratings. To solve this, deep learning techniques can be used to transform the high-dimensional and sparse matrix into a low-dimensional and dense matrix. For example, autoencoders can be used to produce low-dimensional latent representations of context features in context-aware RSs. Cross domain recommendations can be used similarly to how they are used in the cold-start problem. Another solution is to handle each user-item rating as a means to predict the missing user-item ratings. The missing rating can be estimated based on the merging of different user ratings of the same item or different item ratings by the same user.



User/Item	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item 8	Item 9
User 1									
User 2	3					5		8	
User 3				6	4				
User 4		9						7	5
User 5							8		
User 6	6				6				
User 7									
User 8		4						9	
User 9					1				

Figure 4.1: A sparse user-item rating matrix

4.4 Scalability

Scalability is an issue related to the number of items and users a system was designed to cope with. A RS that is performing great with a small number of users and items might not be efficient if the items and users increase up to a million. One method to make an RS scalable is to find the nearest and furthest neighbors of a user with the purpose of reducing the dataset. Deep learning approaches can deal with scalability problems. They can map high dimensional features into lower ones, select the top-k relevant features and use k-means algorithms to cluster similar features. Parallel and distributed computing can also achieve better scalability for the system.

4.5 Synonymy

Sometimes very similar items have different titles, features or entries. It is difficult for RSs to find the similarity between those items and that's when the problem of synonymy is raised. For example, a typical RS might not find a similarity between two items titled ”pants” and ”trousers” although semantically they refer to the same thing. Solutions to this issue include the creation and use of a thesaurus by the system and Latent Semantic Indexing which finds patterns in the relationships between concepts in textual data [28].

4.6 Data noise

Noisy data are considered corrupted because the user's ratings might be biased or untruthful. Sometimes, the rankings are not representative of the user's liking towards an item because the user might not be able to transform their emotions into a number of the rating scale. Or the user's ratings may be affected by the social norms instead of reflecting the true preferences of the user. This issue can be handled by using implicit feedback from the user in order to avoid noise in the dataset. Fuzzy logic can also be used to express the level of dishonesty in user preferences. Trust-aware systems can also be employed to detect malicious users from authentic ones. Attention-based RNN models have also been used to deal with noisy input data.

4.7 Diversity

Recently, diversity in recommendations has been rendered important, even if it risks causing a decrease in the overall accuracy. Diversity can also overcome the popularity bias, which is the issue caused when the recommendation list is filled with items similar to each other that are following a popular trend. If a user doesn't like one of the trending items, they will probably not be interested in the rest of them, thus making the recommendation list useless. Content-based approaches might limit the diversity because if a user has never chosen an item with a particular keyword, then most probably they will never get recommendations of items with this keyword. Researchers are trying to find the balance between similarity and diversity. A K-Furthest Neighbors model has been used to recommend more diverse items. Another method has also been used which exploits the concept of experts. Some of the users are labelled as ”experts” for a certain preference because the system detects they are the most knowledgeable on this preference. The rest of the users are recommended items from the experts recommendation list relevant to that specific item preference [29].

4.8 Familiarity

When users receive recommendations about items that they are familiar with, their trust to the RS increases and consequently their willingness to interact with the recommendations. An example of such familiarity is when users get recommendations for clothes from a vendor they have already chosen and rated positively in the past. However, if a large percentage of the recommendations are directly related to the user's previous ratings, then users are not able to discover new products.

4.9 Novelty

Novelty is used to balance familiarity. Users should get recommendations for items that are new and surprising so that they can discover new content. Those recommendations can benefit from short-term contextual occasions such as the mood of the user or the location.

4.10 Privacy

Some recommendation approaches, such as demographic-based systems, might violate the privacy rules. Especially in the European Union those rules are stricter since companies have to comply with the General Data Protection Regulation (GDPR). Cross domain recommendation should also be applied carefully so as not to expose user's sensitive information to other domains.

4.11 User's effort

The user's effort to get reliable recommendations should be minimized. One purpose of the RSs is to help users save time instead of trying to find interesting items on their own. Asking users to state their preferences or to rate and review items must be done in moderation. Implicit feedback can be used when there is no need for the user to insert explicit feedback.

4.12 Changing preferences

Users tend to change preferences from time to time, particularly when they are exposed to new domain information. The RS should be able to monitor those changes and adjust accordingly.

5. IMPLEMENTATION OF A RECOMMENDER SYSTEM FOR VIDEO GAMES

This section will describe the implementation of a RS for video games and its results. A CB filtering algorithm is applied which uses an autoencoder to compress each game's feature vectors before finding their similarities. Additionally, a CF approach is also implemented that uses a neural network to learn the user-item interactions and make predictions on the games that the user has not interacted with. In the end, both approaches are combined in a hybrid algorithm which uses mixed hybridization to produce recommendations.

5.1 Dataset

The dataset includes data from the Steam game library [30, 31, 32]. It was provided by Professor Julian McAuley of University of California, San Diego, on his website [33]. Specifically, it includes 32,136 games with their metadata such as publisher, title, release date, genres etc. It includes another dataset with the users and the games that they own or have played (if a game is free to play). Those users are from the region of Australia and in total they are 88,310. This dataset will be called the interactions dataset since it includes user-item interactions. For each user-game interaction, the dataset contains the total playtime and the playtime over the last two weeks. Because of the large number of interactions and the lack of a professional runtime environment, this dataset was reduced to 10,000 users and their interactions with the games they have played. Furthermore, after some data exploration, most users owned the top 1000 games and for the rest of the games the interactions became very sparse. After this observation and to achieve faster computations, it was decided to reduce the games to the top 1000 most popular and keep only the interactions that included those games.

There is also another dataset with the reviews of the users on some of the games (in Steam to review a game, the user has to either own it or, if it's free, to have played it). The reviews were missing for 99% of the user-item interactions so they were not included in the final dataset. From the final dataset fields such as *metascore* and *early access* were removed since they were missing for more than 75% of the entries. *Title* was also removed since it was missing for some games and instead the *app name* was used as an identifier besides the *item id*.

In the end, after some data cleaning, the final user-item interactions matrix was composed of 9887 rows (users) and 1001 columns (items, 1 of the items is the None item when a user hasn't interacted with any games) with 90% sparsity and a total of 635,859 interactions.

In the end, the fields that were kept were the following:

- **user_id, item_id:** identifiers for users and games
- **playtime, playtime_2weeks:** the total playtime and the playtime over the last two weeks of an interaction between a user and a game
- **genres:** a list of genres that the game belongs to
- **tags:** a list of tags that are related to the game
- **specs:** options that the game offers (single player, controller support etc.)

- **sentiment:** the overall appreciation of the game from the Steam community (categorical data)
- **publisher, developer, app_name, release_date, price:** self explanatory

5.2 Libraries

Some of the worth-mentioning libraries used in this implementation are the following:

- **pandas:** offers data structures such as DataFrame and Series for data manipulation and analysis. In this project they were used for handling the various datasets.
- **scikit-learn:** includes machine learning algorithms for classification, regression and clustering, as well as functions that can be used in the process of implementing machine learning and deep learning algorithms.
- **keras:** provides implementations for basic neural networks structures such as layers, activation functions etc. Here it was used to build the autoencoder for the CB approach.
- **pytorch:** machine learning library with similar functionality as keras. It focuses mostly on tensor computing and deep neural networks.
- **pytorch lightning:** provides a high level interface for PyTorch to run deep learning experiments easier.
- **matplotlib, pyplot:** plotting library inspired by MATLAB

5.3 Runtime environment

The runtime environment that was used was the one provided by the basic plan (free of charge) of Google Colaboratory. It's specifications are the following [34]:

- 2 x CPUs Intel Xeon 2.2GHz
- 1 x GPU Nvidia Tesla K80 12GB
- 13GB RAM
- 12 hours maximum execution time (when the browser is opened)

The above specifications also depend on current availability.

5.4 Implementation approaches

5.4.1 Content based filtering

In the content-based approach, the game descriptive data (e.g. publisher, tags) are all combined into one text field, which is called document and then the TF-IDF (Term Frequency - Inverse Document Frequency) embeddings are created for each game based on this field. TF-IDF is expressed via the following formulas [35]:

if t is a term and d is a document then:

$$TF(t,d) = \frac{count\ of\ t\ in\ d}{total\ number\ of\ words\ in\ d}$$

$$DF(t) = number\ of\ documents\ in\ which\ t\ is\ present$$

$$IDF(t) = \log \frac{total\ number\ of\ documents}{DF(t)+1}$$

$$TF\text{-}IDF = Term\ Frequency\ (TF) * Inverse\ Document\ Frequency\ (IDF)$$

The document of each game is vectorized on the vocabulary which is the list of all words in the corpus. So, in the end each game is represented by a one-dimensional vector of size equal to the vocabulary size. Those embeddings are passed into an autoencoder which compresses them. This is needed because some terms are very similar, for example *war* and *wargame* which we encounter in tags. This means, a single concept (in this case the concept of war) is dispersed over multiple dimensions in the vector. Compressing the TF-IDF vectors into a lower dimensional space would provide better results as the concepts would be fuzed into one dimension. The expectations of this method are each dimension to represent a solid, complex concept. The architecture of the autoencoder can be seen in Fig. 5.1. Then, for each game that a user has interacted with (owned/played), we find the games with the highest cosine similarity between their embedding and the embedding of the user's game. We keep only the top 20 most similar games and then we apply some evaluation metrics, that will be mentioned later, to estimate the success of the recommendations. The cosine similarity will be explained in the section of the results along with other metrics that will be used later. The initial TF-IDF embeddings (without compression from an autoencoder) have also been tested and compared with the results of the autoencoder approach. This approach was inspired by [36].

5.4.2 Collaborative filtering

In the collaborative filtering approach, a model based implementation was chosen to avoid manually computing the similarity between games or users. Specifically, the training dataset is created by keeping the positive user-game interactions (positive means that the user

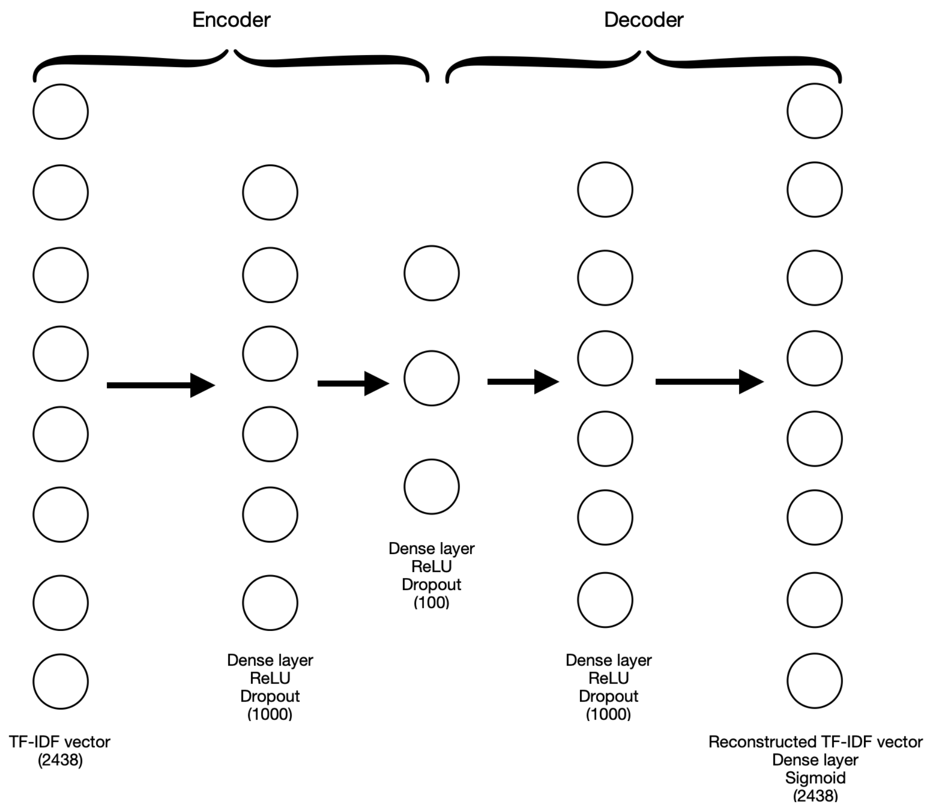


Figure 5.1: The autoencoder used in the content based implementation

has interacted with a game) and for each user we add 4 random negative interactions (games that the user hasn't interacted with). The model that was implemented was based on the Neural Collaborative Filtering (NCF) by [37, 38] and it is shown in Fig. 5.2. The input of the model is the one-hot encoded user and item vectors. Those vectors are fed into the user and item embedding layers respectively. The output of those layers are the user and item embeddings which are smaller and denser vectors. The two embeddings are concatenated into one and passed into a series of fully connected layers. In the end the model returns the probability that the user has interacted with the item. We then run the model for the test users against all games that each user has not interacted with and keep the top 10 games (sorted by the generated probability).

5.4.3 Hybrid

For the hybrid method, the mixed hybridization technique has been used. As mentioned in a previous chapter it is the simplest form of hybridization since the results of the two recommendation methods are combined and form a list. From that list we keep the top 10 most similar games. However, in the previous methods we use different metrics to pick the top 20 recommendations. In CB we use cosine similarity and in CF we use prediction probability. To be able to combine the results in a fair way, we normalize the values of the metrics.

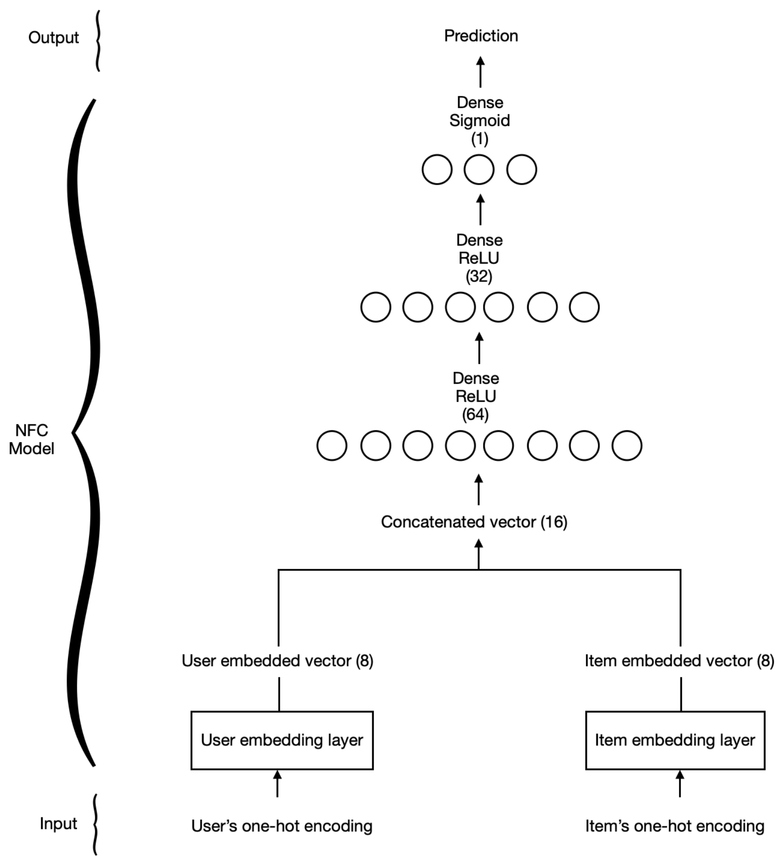


Figure 5.2: The neural model used in the CF implementation

5.5 Results and interpretation

5.5.1 Metrics

5.5.1.1 Cosine similarity

We used cosine similarity to calculate the similarity between the TF-IDF embeddings. The formula for two vectors A and B is:

$$cos(\theta) = \frac{A \cdot B}{|A| \cdot |B|}$$

Cosine similarity ranges between -1 and 1 and as seen above it is the dot product of the two vectors divided by their magnitude. Embedding vectors that point in the same direction have high similarity.

5.5.1.2 Mean average precision

We used Mean Average Precision (MAP) when testing the recommendations in a users subset. It is a common metric used in RSs. MAP calculates the mean of the average precision over all the test users. The average precision compares a list of N ranked recommendations to the set of the actually relevant recommendations [39]. Specifically:

$$precision = P(k) = \frac{\text{number of our recommendations that are relevant}}{\text{number of all recommended items}}$$

Average precision at cutoff N, where N is the number of recommended items, m is the number of all possible items and $rel(k) = 1$ if k-th recommended item is relevant, otherwise $rel(k) = 0$:

$$average\ precision = AP@N = \frac{1}{m} \cdot \sum_{k=1}^{N} P(k) \cdot rel(k)$$

Mean average precision averages the $AP@N$ over all the test users $|U|$:

$$mean\ average\ precision = MAP@N = \frac{1}{|U|} \cdot \sum_{u=1}^{|U|} (AP@N)_u$$

5.5.2 Results

To run the experiments, we have picked a subset of users and split their items into train and test. We use the same users and test items across the different methods to achieve more reliable results.

In Table 5.1 we can see the table with the different approaches and their evaluation results:

Table 5.1: Results from the experiments

Approach	Max cosine similarity	Max probability	MAP	Epochs	Batch size
CB (without AE)	0.092	-	0.05	-	-
CB (with AE)	0.099	-	0.14	20	8
CF	-	0.98	0.10	5	512
Hybrid	-	-	0.11	-	-

5.5.3 Observations

We notice that the MAP values are pretty low. This is due to a flaw in how we split the train and test sets. In the dataset we don't have any information about the date that an item was bought or first played by a user. Therefore, the splitting of the dataset is happening randomly and we might be using earlier bought items for testing and the most recently bought for training. If we could split the games by date we would have achieved better results.

We noticed that in a similar experiment [40], a limit of 5 hours of playtime was set to differentiate between preference or not of a user towards an item. In our case, this differentiation is based on whether an item was bought (or played if it is a free game). Also in [41], the achievements accomplished by a user in the game were integrated into the dataset and the authors pointed out that the achievement information may prove extra useful. In [40], the achieved accuracy of the deep algorithms was around 0.89 which is

8 times better than ours. Probably the playtime limit was important for the training process. However, in that work, a method based on matrix factorization called *Alternating Least Squares* was implemented that achieved a MAP value less than the ones we have achieved, specifically 0.107. That means our deep learning algorithms were a bit better than a traditional, non-deep method which supports the theoretical claims.

In the CB approach there is a clear advantage in the use of the autoencoder versus using the initial TF-IDF vectors. This is noticeable from the difference in the MAP values. Also, calculation of the recommendations was significantly faster when using the compressed TF-IDF embeddings. If we incorporated additional data related to the games (e.g. a description) we would have achieved even better results.

The hybrid approach has less accuracy than CB but better than CF. This might be due to the fact that the normalization of the cosine similarity and the prediction probability in the CB and CF recommendations respectively is not the best way to combine and rank the results. It would be better if both methods used the same metrics. The shortcoming of the CF algorithm might be due to the fact that the embeddings were created using only the user and item id. In other studies such as [40] that were using either pure deep neural networks or a combination of them with factorization machines, the embeddings also contained other features such as genres and as a result they provided better recommendations. However, our CF method is still providing good results compared to other works such as [42] which has a mean precision of 0.014640 for a CF model that uses a neural network that multiplies the user embeddings with the item embeddings and adds bias values before applying a sigmoid layer. A hybrid approach combines this with a CB method that uses video game vector cosine similarity and keyword similarity. The hybrid approach of [42] has a precision 0.015960 which is still lower than ours. The precision of 6 CF algorithms implemented in [43] for e-commerce ranged from 0.0041 to 0.0268 which renders the results of our RS promising.

The RS is sensitive to cold-start issues related to new users. If a user has interacted with close to zero items then the CB method will have a very small set of game vectors to compare with the vectors of all the games in the dataset. The authors of [44] tried to solve the cold-start problem by asking the users through the user interface to rank at least 3 games if they have not done that already. In our case, since we do not provide any interface for the users but instead use a static dataset, we cannot generate recommendations to users that haven't interacted with any game at all. A solution would be to simply remove from the dataset the users who owned less than 5 games as in [41]. We avoided this because we wanted to experiment with the cold start issues and how they are handled by the algorithms.

It is important to mention that MAP is only one of the metrics that can be used to determine whether an RS is good or not. Measuring the diversity and the novelty also plays a significant role in the success of the recommendations as mentioned in [40].

5.6 Steam recommendation pages

After this demonstrative implementation which gives some insight on how a basic game RS works, it is worth mentioning how the Steam app displays the recommendations to the users and what does the interface reveal about the implementation of their RS.

5.6.1 New users

In Fig. 5.3 we can see a screenshot of how the cold start problem is addressed, which means what a user sees when they haven't played any games or viewed any content on the Steam website. The recommendations are guessed instead of predicted and they could be based on location information, current trends or the popularity of the games.

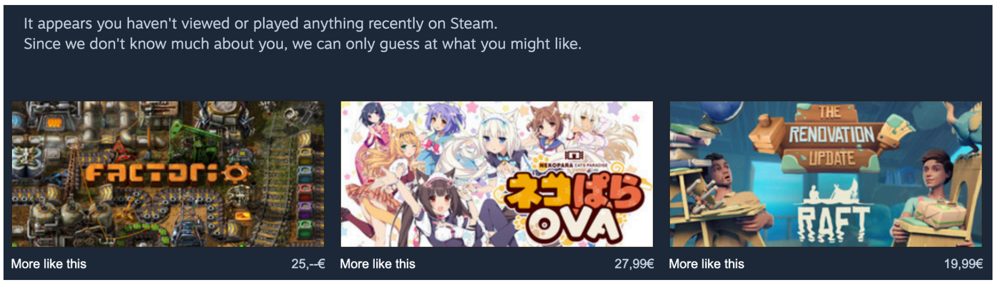

It appears you haven't viewed or played anything recently on Steam.
Since we don't know much about you, we can only guess at what you might like.

Figure 5.3: Guessed recommendations for a new user on Steam

5.6.2 Recommendations by friend activity

On Steam, users can connect with each other and become friends. As a result, they can see which games their friends are playing or have bought and which they have recommended. When a user reviews a game they can recommend it to their friends and it will appear as in Fig. 5.4. This type of recommendation could increase the familiarity level.

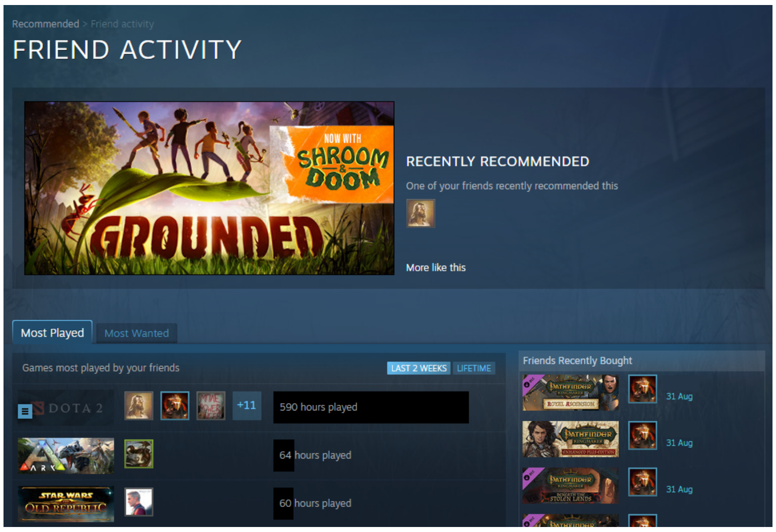
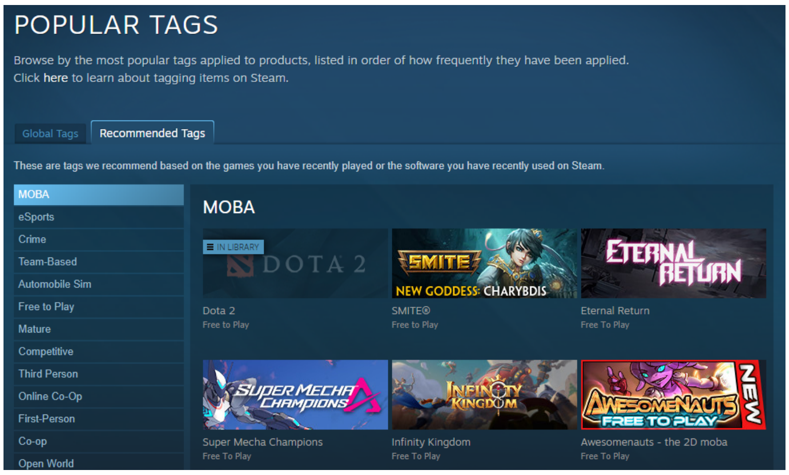

One of your friends recently recommended this

Figure 5.4: Steam recommendations by friend activity

5.6.3 Recommendations by tags

Games are also recommended based on the popularity of their tags and the relativity to the user's previous gaming history. Users can propose tags for a game title and if a tag is proposed frequently then it becomes a featured category [45]. That means a user can also get recommendations based on custom tags they have used. An example can be seen in Fig. 5.5.


POPULAR TAGS

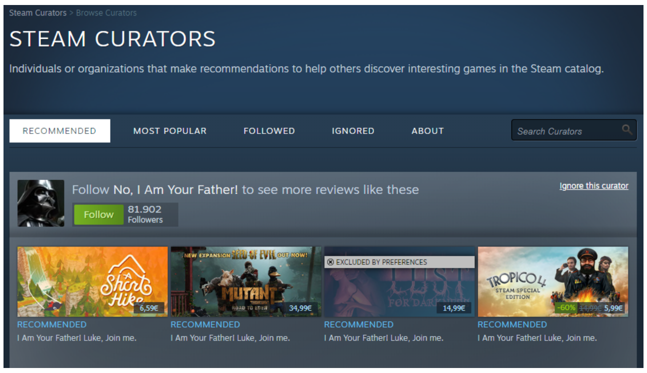
Browse by the most popular tags applied to products, listed in order of how frequently they have been applied.
Click here to learn about tagging items on Steam.

These are tags we recommend based on the games you have recently played or the software you have recently used on Steam.

Figure 5.5: Steam recommendations by popular tags

5.6.4 Recommendations by curators

Curators are either users or organizations that can recommend games. Users have the option to follow curators they like (Fig. 5.6). Steam can also recommend curators based on the relevance between the games they recommend and the games the user has been playing recently [46].


Steam Curators > Browse Curators

STEAM CURATORS

Individuals or organizations that make recommendations to help others discover interesting games in the Steam catalog.

Follow No, I Am Your Father! to see more reviews like these

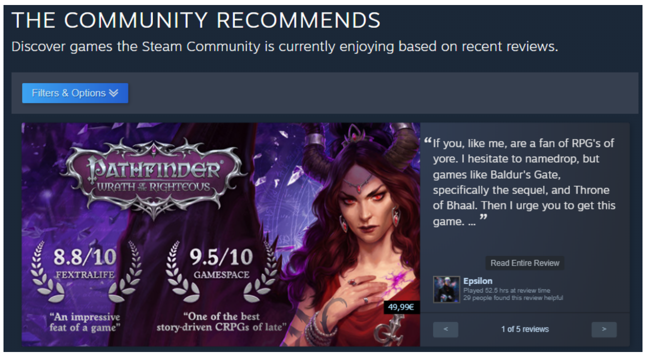
Figure 5.6: Steam recommendations by curators

5.6.5 Recommendations by the community

On this page, reviews of other users are displayed based on how many users have voted them as ”helpful”, meaning that those reviews might be not only positive but also negative. Both curator recommendations and community recommendations can provide the users with novel and surprising recommendations and help them discover new games.

5.6.6 Interactive recommender

The most interesting recommendation mechanism that Steam offers is the interactive recommender. Users can select the balance between popularity and nicheness. According to [47], the recommender is trained on the playtime histories of millions of users. Instead of being affected by the reviews and tags (content of a game), it looks into the users’ playing habits. From this information we deduce that it is a collaborative filtering model based recommender system.



THE COMMUNITY RECOMMENDS

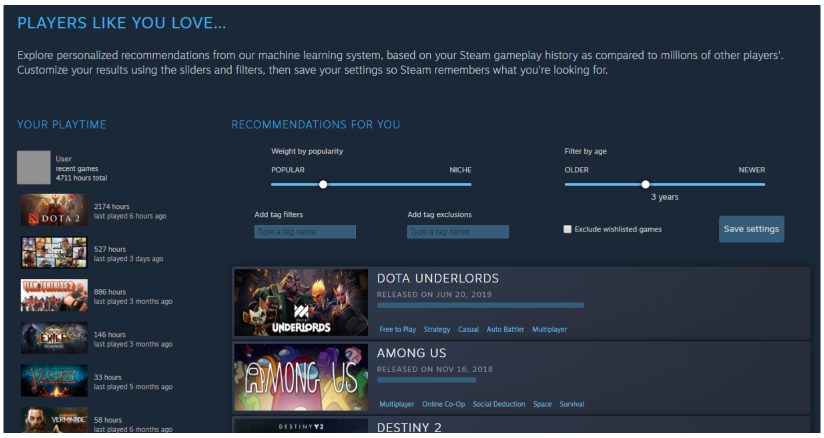
Discover games the Steam Community is currently enjoying based on recent reviews.

"If you, like me, are a fan of RPG's of yore. I hesitate to namedrop, but games like Baldur's Gate, specifically the sequel, and Throne of Bhaal. Then I urge you to get this game. ..."

Epsilon
Played 52.5 hrs at review time
29 people found this review helpful

1 of 5 reviews

Figure 5.7: Steam recommendations by the community



PLAYERS LIKE YOU LOVE...

Explore personalized recommendations from our machine learning system, based on your Steam gameplay history as compared to millions of other players'. Customize your results using the sliders and filters, then save your settings so Steam remembers what you're looking for.

YOUR PLAYTIME

User
recent games
4711 hours total

2174 hours
last played 6 hours ago

527 hours
last played 3 days ago

886 hours
last played 3 months ago

146 hours
last played 3 months ago

33 hours
last played 5 months ago

58 hours
last played 6 months ago

RECOMMENDATIONS FOR YOU

Weight by popularity

Filter by age: 3 years

Add tag filters

Add tag exclusions

☐ Exclude wishlisted games

DOTA UNDERLORDS
RELEASED ON JUN 20, 2019
Free to Play Strategy Casual Auto Battler Multiplayer

AMONG US
RELEASED ON NOV 16, 2018
Multiplayer Online Co-Op Social Deduction Space Survival

DESTINY 2

Figure 5.8: Steam's interactive recommender

6. CONCLUSIONS AND FUTURE WORK

In this thesis, a concise review was provided on how RSs work and what is the use and the benefits of deep learning techniques on RSs. Some basic recommendation algorithms were implemented using content based and collaborative filtering methods combined with deep learning models.

There are still improvements that can be made to those algorithms. Incorporating more information about the users and the games, even from different sources, would provide more accurate recommendations. This applies not only to game RSs but any other RS since more data leads to better prediction models. This additional information could also be used for sentiment analysis to analyze the user's current mood and predict the future one. Since a user's choices are affected by their current state of mind, this prediction would result in better suggestions. Sentiment analysis can also address the problem of the user's changing preferences as they depend on the user's changing temper.

We could go even further and apply personality detection techniques to infer the user's personality traits based on data from various sources. Those traits could be used to predict the long-term preferences of users and consequently produce more reliable recommendations.

Certainly, there is room for improvement concerning optimization. Distributed optimization algorithms can be used in the future to reduce the computational time costs.

RSs solve the problem of information overload which is very common nowadays because of the increasing amounts of data produced. Consequently, the need for recommendations will remain in the future and probably grow even more, so improvements will definitely be necessary. Although deep learning has very promising results, there are still open challenges such as achieving better results in accuracy and scalability.

ABBREVIATIONS - ACRONYMS

RS	Recommender System
CF	Collaborative Filtering
CB	Content Based
SRS	Social Recommender System
MSE	Mean Square Error
RMSE	Root Mean Square Error
MAE	Mean Absolute Error
DL	Deep Learning
AE	Autoencoder
ReLU	Rectified Linear Unit
DAE	Denoising Autoencoder
SDAE	Stacked Denoising Autoencoder
RBM	Restricted Boltzmann Machine
CD	Contrastive Divergence
DBN	Deep Belief Network
GAN	Generative Adversarial Network
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
GRU	Gated Recurrent Unit
LSTM	Long Short Term Memory
TF-IDF	Term Frequency — Inverse Document Frequency
NCF	Neural Collaborative Filtering
MAP	Mean Average Precision

REFERENCES

[1] Roger Bohn and James Short. Measuring consumer information. In *International Journal of Communication*, volume 6, pages 980–1000, 2012.

[2] Thi Ngoc Trang Tran, Alexander Felfernig, Christoph Trattner, and Andreas Holzinger. Recommender systems in the healthcare domain: state-of-the-art and research issues. In *Journal of Intelligent Information Systems*, volume 57, pages 171–201, 2021.

[3] F.O. Isinkaye, Y.O. Folajimi, and B.A. Ojokoh. Recommendation systems: Principles, methods and evaluation. In *Egyptian Informatics Journal*, volume 16, pages 261–273, 2015.

[4] Prince Grover. Various implementations of collaborative filtering. `https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0`, December 2017. Online, accessed on August 2021.

[5] Zahid Younas Khan, Zhendong Niu, Sulis Sandiwarno, and Rukundo Prince. Deep learning techniques for rating prediction: a survey of the state-of-the-art. In *Artificial Intelligence Review*, volume 54, pages 95–135, 2021.

[6] Jyoti Shokeen and Chhavi Rana. A study on features of social recommender systems. In *Artificial Intelligence Review*, volume 53, pages 965—988, 2020.

[7] Aminu Da'u and Naomie Salim. Recommendation system based on deep learning methods: a systematic review and new directions. In *Artificial Intelligence Review*, volume 53, pages 2709–2748, 2020.

[8] Zeynep Batmaz, Ali Yurekli, Alper Bilge, and Cihan Kaleli. A review on deep learning for recommender systems: challenges and remedies. In *Artificial Intelligence Review*, volume 52, pages 1–37, 2019.

[9] Julianna Delua. Supervised vs. unsupervised learning: What's the difference? `https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning`, March 2021. Online, accessed on July 2021.

[10] Wikipedia. Autoencoder. `https://en.wikipedia.org/wiki/Autoencoder`, August 2021. Online, accessed on September 2021.

[11] Matthew Stewart. Comprehensive introduction to autoencoders. `https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368`, April 2019. Online, accessed on September 2021.

[12] Dominic Monn. Denoising autoencoders explained. `https://towardsdatascience.com/denoising-autoencoders-explained-dbb82467fc2`, July 2017. Online, accessed on August 2021.

[13] Wikipedia. Restricted boltzmann machine. `https://en.wikipedia.org/wiki/Restricted_Boltzmann_machine`, August 2021. Online, accessed on September 2021.

[14] Geoffrey E. Hinton Scholarpedia. Deep belief networks. `http://www.scholarpedia.org/article/Deep_belief_networks`, October 2011. Online, accessed on September 2021.

[15] Mayank Vadsola. The math behind gans (generative adversarial networks). `https://towardsdatascience.com/the-math-behind-gans-generative-adversarial-networks-3828f3469d9c`, January 2020. Online, accessed on September 2021.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Y. Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 3, 2014.

[17] A. V. Prosvetov. Gan for recommendation system. In *Journal of Physics: Conference Series*, number 1405, 2019.

[18] Ismail Mebsout. Convolutional neural networks - part 1. `https://www.ismailmebsout.com/Convolutional%20Neural%20Network%20-%20Part%201/`, February 2020. Online, accessed on September 2021.

[19] Ismail Mebsout. Recurrent neural networks. `https://www.ismailmebsout.com/recurrent-neural-networks/`, April 2020. Online, accessed on September 2021.

[20] Wikipedia. Multilayer perceptron. `https://en.wikipedia.org/wiki/Multilayer_perceptron`, August 2021. Online, accessed on August 2021.

[21] Roger Grosse. Lecture 5: Multilayer perceptrons. https://www.cs.toronto.edu/~mren/teach/csc411_19s/lec/lec10_notes1.pdf. Online, accessed on September 2021.

[22] Wikipedia. Attention (machine learning). https://en.wikipedia.org/wiki/Attention_(machine_learning), August 2021. Online, accessed on August 2021.

[23] Sebastian Ruder. Deep learning for nlp best practices. https://ruder.io/deep-learning-nlp-best-practices/index.html#attention, July 2017. Online, accessed on September 2021.

[24] Scott Rome. Understanding attention in neural networks mathematically. https://srome.github.io/Understanding-Attention-in-Neural-Networks-Mathematically/, March 2018. Online, accessed on September 2021.

[25] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. Collaborative recurrent autoencoder: Recommend while learning to fill in the blanks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 415—423, 2016.

[26] Pearl Pu, Li Chen, and Rong Hu. Evaluating recommender systems from the user's perspective: survey of the state of the art. In *User Modeling and User-Adapted Interaction*, volume 22, pages 317–355, 2012.

[27] Erion Cano and Maurizio Morisio. Hybrid recommender systems: A systematic literature review. 21:1487–1524, 2017.

[28] Wikipedia. Latent semantic analysis. https://en.wikipedia.org/wiki/Latent_semantic_analysis, August 2021. Online, accessed on August 2021.

[29] N. Nikzad–Khasmakhi, M.A. Balafar, and M. Reza Feizi–Derakhshi. The state-of-the-art in expert recommendation systems. 82:126–147, 2019.

[30] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *International Conference in data Mining*, 2018.

[31] Mengting Wan and Julian McAuley. Item recommendation on monotonic behavior chains. In *ACM Conference on Recommender Systems*, 2018.

[32] Apurva Pathak, Kshitiz Gupta, and Julian McAuley. Generating and personalizing bundle recommendations on steam. In *Special Interest Group on Information Retrieval Conference*, 2017.

[33] Julian McAuley. Steam datasets. https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data. Online, accessed on August 2021.

[34] Amirhossein Kazemnejad. How to do deep learning research with absolutely no gpus - part 2. https://kazemnejad.com/blog/how_to_do_deep_learning_research_with_absolutely_no_gpus_part_2/, August 2019. Online, accessed on August 2021.

[35] William Scott. Tf-idf from scratch in python on real world dataset. https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089, February 2019. Online, accessed on August 2021.

[36] Adam Lineberry. Creating a hybrid content-collaborative movie recommender using deep learning. https://towardsdatascience.com/creating-a-hybrid-content-collaborative-movie-recommender-using-deep-learning-cc8b431618af, 2018. Online, accessed on August 2021.

[37] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173—182, 2017.

[38] James Loy. Deep learning based recommender systems. https://towardsdatascience.com/deep-learning-based-recommender-systems-3d120201db7e, October 2020. Online, accessed on August 2021.

[39] Sonya Sawtelle. Mean average precision (map) for recommender systems. https://sdsawtelle.github.io/blog/output/mean-average-precision-MAP-for-recommender-systems.html, October 2016. Online, accessed on September 2021.

[40] Germán Cheuque, José Guzmán, and Denis Parra. Recommender systems for online video game platforms: The case of steam. In *Companion Proceedings of The 2019 World Wide Web Conference*, page 763–771, 2019.

[41] Bram Notten. Improving performance for the steam recommender system using achievement data, 2017.

[42] Nicholas Crawford. Improving video game recommendations using a hybrid, neural network and keyword ranking approach, 2019.

[43] Zan Huang, Daniel Dajun Zeng, and Hsiu-chin Chen. A comparison of collaborative-filtering recommendation algorithms for e-commerce. *IEEE Intelligent Systems*, 22:68–78, 09 2007.

[44] Syed Muhammad Anwar, Talha Shahzad, Zunaira Sattar, Rahma Khan, and Muhammad Majid. A game recommender system using collaborative filtering (gambit). In *14th International Bhurban Conference on Applied Sciences and Technology*, pages 328–332, 2017.

[45] Steam. Introducing steam tags, a powerful new way to shop for games. `https://store.steampowered.com/tag/`. Online, accessed on August 2021.

[46] Steam. Steam curators. `https://store.steampowered.com/curators/aboutcurators/`. Online, accessed on August 2021.

[47] Steam. Introducing the steam interactive recommender. `https://store.steampowered.com/news/app/593110/view/1716373422378712840`, March 2020. Online, accessed on August 2021.

[48] Diego Monti, Giuseppe Rizzo, and Maurizio Morisio. A systematic literature review of multicriteria recommender systems. In *Artificial Intelligence Review*, volume 54, pages 427–468, 2021.

[49] Ian MacKenzie, Chris Meyer, and Steve Noble. How retailers can keep up with consumers. `https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers`, October 2013. Online, accessed on July 2021.

[50] Yash Mehta, Navonil Majumder, Alexander Gelbukh, and Erik Cambria. Recent trends in deep learning based personality detection. In *Artificial Intelligence Review*, volume 53, pages 2313–2339, 2020.

[51] Ashima Yadav and Dinesh Kumar Vishwakarma. Sentiment analysis using deep learning architectures: a review. In *Artificial Intelligence Review*, volume 53, pages 4335—4385, 2020.