POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Deep Recommendation Systems

Relatori:
prof. Silvia Chiusano (Politecnico di Torino)
prof. Laurence Likforman-Sulem (Télécom ParisTech)

Candidato:
Davide Gallitelli

In collaborazione con:
Amazon Web Services (AWS)

Anno Accademico 2018-2019

Contents

Abstract

This Master's Thesis presents the results obtained during the 6-months internship at Amazon Web Services, a world-known company which provides on-demand cloud computing platforms to individuals, companies and governments, on a paid subscription basis.

The main focus of this work is to provide a scientific analysis and a description of the implementation realised to develop a Deep Learning based general purpose recommendation system. After having analysed the state of art for traditional recommendation systems, a few Deep Learning architectures are proposed to solve the problem at hand, and the results obtained on several freely available datasets are compared. Particular attention is given to collaborative filtering methods, based on the history of interaction between user and item, and hybrid recommendation system, which promise to solve some of the shortcoming faced with collaborative filtering methods by enriching user-item interaction with other existing information, in the form of either additional features or textual or graphical input.

Finally, a summary of the work done and of the future perspectives is given, with hints regarding a possible implementation of such a system on the AWS platform.

Chapter 1

Introduction

> We have 6.2 million customers; we should have 6.2 million stores. There should be the optimum store for each and every customer.
>
> *Jeff Bezos,*
> *CEO of Amazon.com, 1999*

Recommender Systems have become increasingly relevant for companies providing any kind of service to the users. Examples include companies such as Amazon, leader in the e-commerce world capable of recommending a small number of items the user might enjoy based on current context and past behaviour across a catalogue of hundreds of millions of items, Spotify, most used music streaming service providing a Discover playlist every week that adapts with the user tastes and lets her discover new music all the time, and Netflix, top video streaming service worldwide with its ever-growing subscription rate and hours spent binge watching series and movies.

The value generated for these companies from Recommender Systems is not negligible. According to a McKinsey study, up to 35% of Amazon.com revenue, increasing their total sales by 29% to $136B in 2016 and $177.9B in 2017, and 75%

of what Netflix views came from product recommendations based on algorithms and predictive models that analyse transaction data and digital-media trends. Moreover, Netflix executives Carlos A. Gomez-Uribe and Neil Hunt state that a combination of personalization and recommendation from their own Recommender Systems saves them more than $1B each year[9], with this number expected to be grown since the publication of the paper in 2015. One factor that all of the above implementations of Recommender Systems have in common is that user preferences are often difficult to guess, especially when facing big customer basis and even bigger catalogues of products to recommend. The difficulty of this task is such that a company either needs a group of expensive data scientist and engineers in order to develop and maintain such systems, on needs to outsource its recommendation services, an example being BBC.

With the increasing diversity and volume of data available, standard approaches to Recommender Systems do not prove to be sufficient anymore, lacking the ability to scale at the same pace as the user/catalogue base, as well as the flexibility of learning from changing trends in data, without re-architecting the whole solution from scratch.

That is why most of the most recent and most efficient Recommender System are based on Machine Learning algorithms, attempting to understand the underlying data patterns, and learning the mapping of some input to some output. Any ML algorithm relies on data available for the training process, an algorithm (model), and loss functions to optimize. By pre-processing data in a format which can be appropriately learnt by the algorithm trying to minimize/maximize the objective loss function, the model encodes the beliefs about data patterns and uses them to make predictions about other data points. This process can be used to solve many problems that traditional heuristics found too complex to tackle, including classification, regression and clustering.

In the recent years, the sheer amount of data available and the complexity of the patterns became such that classical ML algorithms were not sufficient to provide useful insights into the data. First, most of the Machine Learning algorithms

need a very detailed domain expertise, capable of grasping features of the data that were hidden in the original representation, process that became known as Feature Extraction. This implies human intervention and therefore increasing costs for companies which need to include in their team both domain experts and data scientists. Furthermore, Machine Learning fails in solving complex problems such as image classification, natural language processing (NLP), and automatic speech recognition (ASR). Due to amazing performances when large amount of data is available, lack of necessity of domain understanding for feature introspection therefore less need for feature engineering, and ability to tackle problems previously thought to be uncrackable, Deep Learning models have gathered attention and research investments all over the world, both from companies and independent researchers.

The goal of this work is to investigate how Deep Learning models can be used to effectively solve the Recommendation Problem, by proposing a few different architectures and showing the results obtained on multiple datasets freely available online. Most of the tests have been done on the MovieLens dataset, in its different declinations: *ml-100k*, 100,000 ratings from 600 users on 9,000 movies, *ml-1M*, 1 million ratings from 6000 users on 4000 movies, *ml-10M*, 10 million ratings from 72,000 users on 10,000 movies, *ml-latest-full*, 27,000,000 ratings from 280,000 users on 58,000 movies. Computations and training for the first two dataset have been done on a personal *MacBook Pro 2017*, without dedicated GPU, while tests on the bigger datasets required the use of instances on the AWS platform, namely *ml.p3.16xlarge* instances available "as-a-service" thanks to *Amazon SageMaker*.

This master's thesis is structured as follows:

- an *Introduction*, where the goals of this report and some hints to the conclusions are reported;

- a *State of the art* chapter (Chapter 2), which gives a definition to the recommendation problem, and presents the possible approaches to solve said problem; a quick overview about Machine Learning and Deep Learning is also given, introducing some of the algorithms that have been historically used to

build Recommendation systems;

- a *Deep Recommender Systems* chapter (Chapter 3), which discusses the proposed implementations of deep neural networks and their results on the MovieLens *ml-latest-small*;
- a *Results* chapter (Chapter 4) reviews the proposed algorithms and shows the results obtained on multiple datasets, comparing them with a baseline obtained by the state-of-the-art implementations;
- finally, a *Conclusion and Future Work* chapter (Chapter 5) summarises the work done and describes how the system could possibly evolve, possibly obtaining better results.

Chapter 2

State of the Art

In order to properly understand how to implement a recommendation system, it is important to correctly understand the recommendation problem itself, by providing a proper mathematical definition (Section 2.1). Once the problem has been defined, it is now possible to describe how the recommender problem has been approached in the literature (Section 2.2: collaborative filtering, content-based recommendation or hybrid recommendation. In particular, the most common implementation in Machine Learning (Section 2.3) and Deep Learning (Section 2.4) are analyzed, after going through the general concepts of these technologies and the reason why these algorithms should be used to solve the recommendation problem.

2.1 The recommendation problem

A recommender system is a technology that is deployed in the environment where items (products, movies, events, articles) are to be recommended to users (customers, visitors, app users, readers) or the opposite. Typically, there are many items and many users present in the environment making the problem hard and expensive to solve. This problem, shown in figure 2.1, can be solved in three main phases: information collection, learning, and inference (prediction or recommendation).

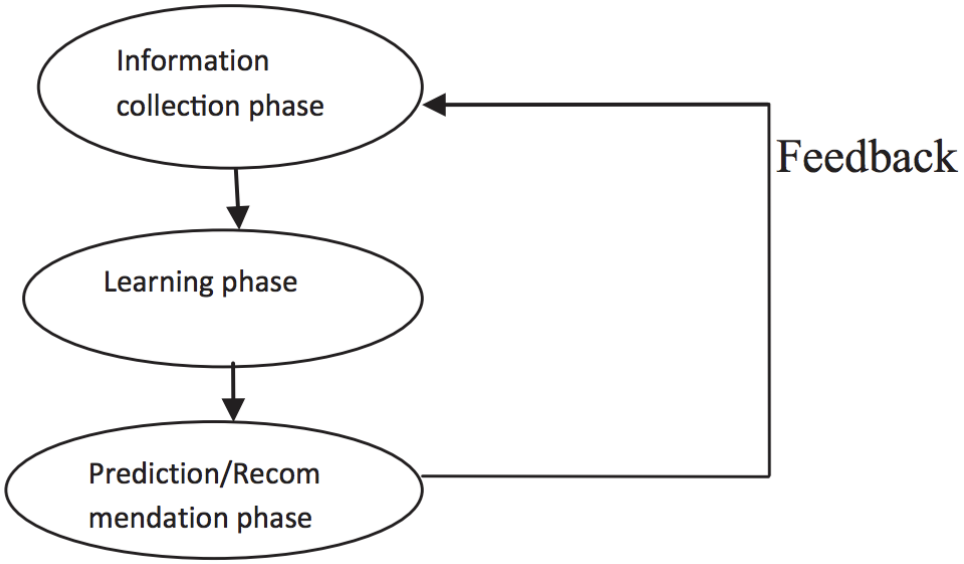

Figure 2.1: Diagram of the Recommendation problem

In the first phase, *information retrieval*, relevant information of users is collected to generate a user profile or model for the prediction tasks including user's attribute, behaviors or content of the resources the user accesses. The system needs to know as much as possible from the user in order to provide reasonable recommendation right from the onset. Recommender systems rely on different types of input, either explicit feedback, which includes explicit input by users regarding their interest in item, or implicit feedback by inferring user preferences indirectly through observing user behavior. A very common example of the first kind of feedback is the ratings given by users to movies. Hybrid feedback can also be obtained through the combination of both explicit and implicit feedback.

Once feedback is obtained and stored in the preferred way, the system applies a learning algorithm to filter and exploit the user's features from the feedback gathered in information collection phase. According to the chosen algorithm, a pre-processing phase might be needed in order to make sure that the data is in the proper format

requested by the algorithm.

With the data pattern discovered during the learning phase, the system is finally able to infer about user preferences. The task here can be either a *prediction* or a *recommendation*: the system can either predict a feedback from a user regarding a certain item, and then use this information to decide whether to suggest said item to the user, or try to rank the top N items in which the user might be interested. Most of the results presented in this work adopt the prediction approach to the problem, since it is the one that can be more easily quantified with standard metrics and the same approach can be extended to solve the recommendation problem with trivial development effort while building a complete solution.

A more mathematical definition of the recommendation problem and the recommendation problem has been by Sarwar et al. in 2001 [32].

The prediction problem can be formalised as follows: let

$$U = u_1, u_2, ..., u_m$$

be the set of all users, and let

$$I = i_1, i_2, ..., i_n$$

be the set of all possible items that can be recommended. Each user u_i has a list of items I_{u_i} . This list represents the items that the user has expressed her interests. Note that $I_{u_i} \subseteq I$, and it is possible that $I_{u_i} = \emptyset$, such as in the case of new subscription to a service. Then, the function, $P_{u_a} i_j$ is the predicted likeliness of item i_j for the active user u_a, such as $i_j \notin I_{u_i}$.

The recommendation problem can be stated as obtaining a list of N items, $I_r \subset I$, that the user will like the most (i.e the ones with higher $P_{u_a} i_j$ value). The recommended list should not contain items from the user's interests, therefore $I_r \cap I_{u_i} = \emptyset$. Both the set I of possible items and the set of users U can be very large. In most recommender systems, the prediction function $P_{u_a} i_j$ is usually represented by a rating, which is given by the user either explicitly or implicitly through some measures, e.g., by tracking if a movie is skipped in the first minutes of play. They

are represented as triplets $< h_u, i, r_i >$ where r is the rating value assigned by the user u to a particular item i. The value is usually a real number (e.g., from 0 to 1), a value in a discrete range (e.g., from 1 to 5), or a binary variable (e.g., like/dislike).

For the rest of the work, the expression "recommendation problem" will be mainly used, while still considering the "prediction problem" as the one to be solved.

2.2 Recommender Systems

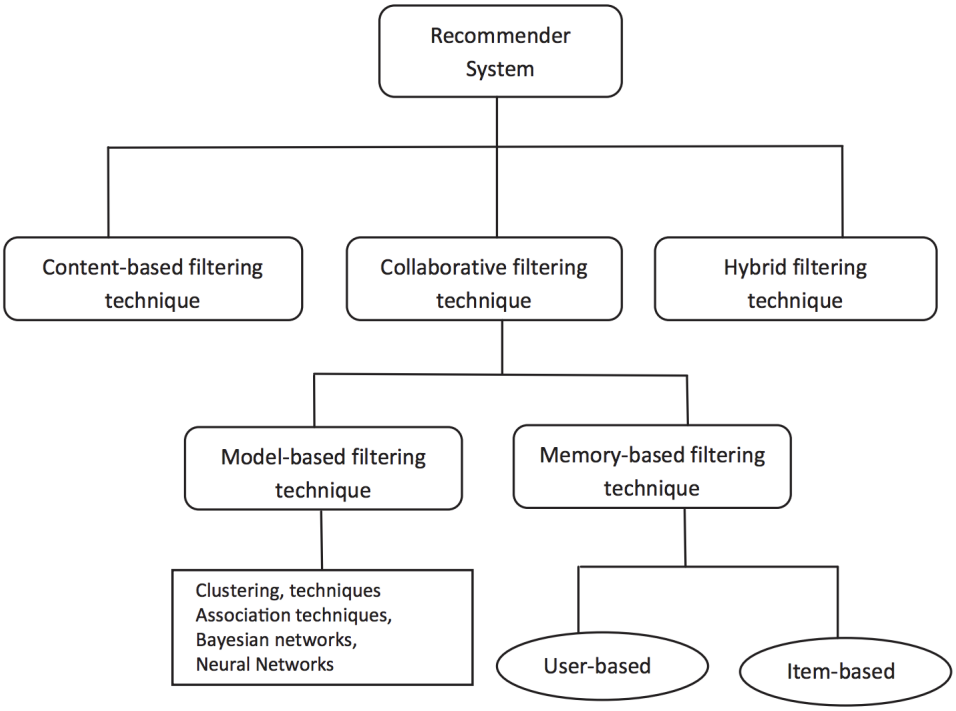

Figure 2.2: Types of Recommender System

When implementing a recommender system, there are a few different techniques to choose from with regards to exactly how the recommendations are made. Most of today's systems employ content-based filtering or collaborative filtering, or even a mixture of the two, resulting in what we call hybrid systems. As in most Data Science tasks, no single algorithm is the *silver bullet* for solving the recommendation problem every time, regardless of the data or task. It is always better to try and implement multiple solutions, and check which one of those delivers an output closer to the expected metrics or results. This work goal is to implement multiple algorithms and test the results on the *MovieLens* dataset.

2.2.1 Collaborative Filtering

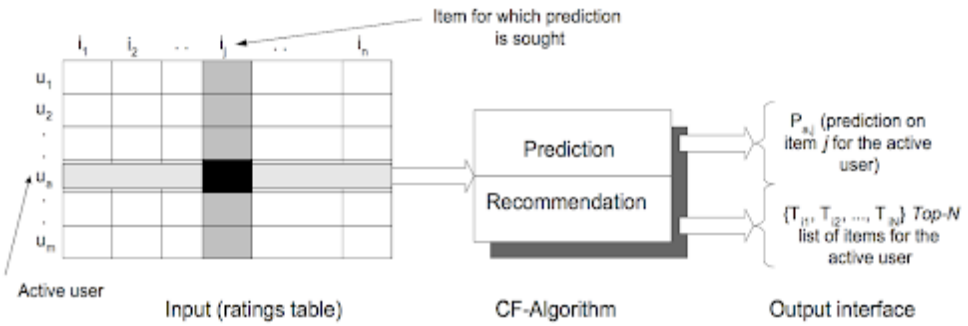

Figure 2.3: Visualization of a Collaborative Filtering algorithm

The goal of a Collaborative Filtering (CF) algorithm is to suggest new items or to predict the utility of a certain item for a particular user based on the user's previous likings and the opinions of other like-minded users[32]. CF algorithms, regardless of the problem to solve, prediction or recommendation, work with the entire mxn user-item data as a ratings matrix A, where each entry a_{ij} represents the preference score (rating) of the i-th user on the j-th item.

Researchers have come up with multiple collaborative filtering algorithms, which can be mostly grouped into two categories: *Memory-based* (user-based) and *Model-based* (item-based) algorithms.

Memory-based algorithms utilize the entire ratings matrix to generate a prediction. Most of these algorithms employ techniques based on statistical approaches which find a set of users that share similar rating to the target user. This set of users is known as *neighbours* of user i. Starting from the neighbourhood of user i, it is possible to combine predictions of neighbours to produce a prediction or a *top-N* recommendation for the active user. Several types of similarity measures[19] are used to compute similarity between item/user. The two most popular similarity measures are correlation-based, such as the *Pearson Correlation Coefficient* (PCC) which measures the extent to which two variables linearly are related to each other,

and cosine-based, which measures the similarity between two n-dimensional vectors based on the angle between them. Traditional Machine Learning approaches to solve this problem include algorithms such as *K-Nearest Neighbours*.

Model-based algorithms exploit a probabilistic approach in order to provide the expected value of a user prediction by first developing a model of user ratings, meaning given her ratings on other items. Traditional Machine Learning algorithms to obtain the model include *Bayesian Networks*, *Clustering*, *Rule Mining* approaches, *Matrix Factorization* and *Single Value Decomposition*. Further analysis, in particular for *Matrix Factorization* algorithms, is provided later in this work.

Although collaborative filtering is currently one of the most used methods of social–based recommender systems, the approach presents some drawbacks:

- **data sparsity** and **high dimensionality**[17], due to the nature of the dataset itself being very large while the coverage of users' ratings among the items;
- **grey sheep**[7] or **outliers**, referring to the problem of some users having atypical tastes, meaning largely different from the norm, and therefore leading to poor recommendations;
- **cold-start**, caused by the lack of ratings for new users
- **early-rater**, caused by the lack of rating for new items[3]
- **popularity bias**, caused by popular items being similar to lots of items and therefore being more probable to be recommended[38]

2.2.2 Content-Based Filtering

In the Content–Based Filtering (CBF) approach, the recommender collects information describing the items and then, based on the user's preferences, it predicts which items the user could like. It is therefore a technique which is domain-dependent and emphasizes more on the analysis of the attributes of items in order to generate predictions. The process of characterising the item data set can be automatic, for

example extracting features by analysing the content, based on manual annotations by the domain experts, or even using the tags from the community of users [5][6].

Content-based filtering uses different algorithms to find similarity between items in order to generate useful recommendations by exploiting more significative representation of the data. Therefore this technique is not a social-based recommender system, since it does not need other users ratings to infer a recommendation. Content-based filtering technique can handle deviation in the tastes of the user, by adjusting the recommendations within a very short period of time. The greatest advantage of CBF is the ability to solve the previously described early-rater and cold-start problem. By extracting a significative representation of the new item, it is possible to recommend it to the appropriate users; the same is valid for new users, which can be recommended other items seen by similar users.

A recommender system implementing the above described content-based filtering technique need a well-defined architecture supporting multiple components with well-defined tasks. Lops et Al. [22] proposed a high level architecture (Figure 2.4) made of three main components:

- **Content analyzer**, which is the component responsible for "pre-processing" the data, meaning to prepare item's relevant information in a format suitable to the rest of the recommendation process;

- **Profile learner**, which is the component responsible for modeling the user profile from user preferences and interests thanks to statistical approaches or Machine Learning algorithms [27];

- **Filtering component**, component responsible for the real recommendation of new items starting from user information or provided queries.

Regarding Content-based filtering techniques, the main focus of this work will be on the first two component, coherently with the definition of recommendation problem given in Section 2.1 of this work.

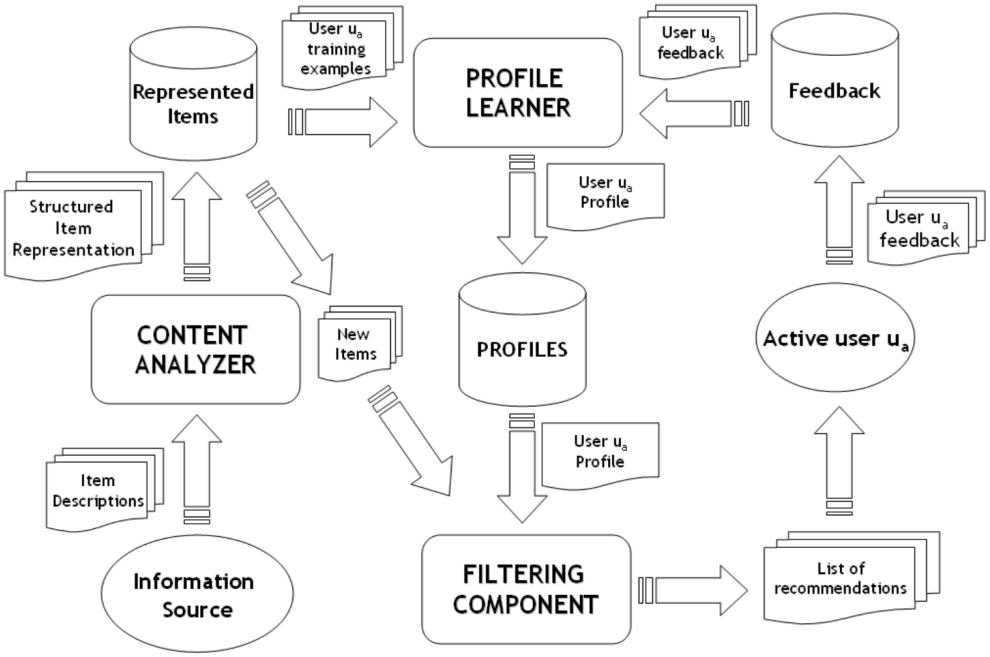

Figure 2.4: Content-Based Recommender High Level Architecture [22]

As described before, Content-based filtering techniques are dependent on items' metadata. The adoption of the content-based recommendation paradigm has several advantages when compared to collaborative filtering:

- **User independence**, meaning that extracting information and predicting ratings for the current active user only require information regarding the item itself and the user profile;
- **Transparency**, since content features and descriptions can be explicitly listed to explain why an item occurs in a list of recommendation
- **Solution to cold-start**, giving the capability to recommend new products never before rated, or existing items to new users.

However, CBF methods are not exempt from shortcomings:

- **Limited Content Analysis**[1], also known as the necessity of rich description of items and very well organized user profile before any recommendation can be made to users - a process which requires a lot of domain knowledge, given by a team of domain experts which should assist Data Scientists that build the Recommendation System.
- **Content overspecialization**[44] or **serendipity**, meaning the tendency of the content-based systems to produce recommendations with a limited degree of novelty, therefore users are restricted to getting recommendations similar to items already defined in their profiles.

Implementation examples of Content-Based Filtering include Vector Space Models such as *Term Frequency Inverse Document Frequency* (TF/IDF) or Probabilistic models such as *Naïve Bayes Classifier*, *Decision Trees* or *Neural Networks*. More details are provided in section 2.3 of this work.

2.3 Machine Learning

Producing recommendations involves applying statistical and knowledge discovery techniques in a database of user data[32]. Machine Learning models have been thoroughly researched in the last decade, with many among them already available in most of production environments dealing with the recommendation problem. Depending on the learning type of the applied algorithm, it can be classified into one of four main categories[29]: supervised, unsupervised, semi-supervised, and reinforcement learning.

Supervised learning is applied when algorithms are provided a training set with input values that are mapped to "correct" answers, the labels; therefore training set and labels are used to create a function that can map new input values to any of the existing labels. Unsupervised learning algorithms need to learn from the data set without actual labelling being provided. Semi-supervised algorithms use incomplete training sets from which predictions are made. Reinforcement learning is when learning algorithms predicts the label or generate predictions being driven by an external feedback, either positive or negative feedback according to its behaviour, and adjusts the parameters of the model accordingly.

In the following sections, some algorithms that have proved to be effective and have been adopted in production by companies are discussed and described in more detail. The chosen algorithms are: *K-Nearest Neighbours* (KNN), *Decision Trees*, and *Matrix Factorization* (or *Singular Value Decomposition*). A brief excursus on *clustering* methods applied to recommender systems is also provided.

2.3.1 Decision Trees

Decision tree learning[37] is among the most widely used and practical algorithm for inductive inference[27]. The goal of the algorithm is to approximate a discrete-value target function, where a decision tree represents the learned function. On a human perspective, decision trees can also be seen as a set of if-then rules. By traversing the tree from the root to a leaf node, it is possible to classify the instances: each node of

the tree tests the attribute specified by the node and then moves down the until a leaf node is reached, which represents the class of the instance. In general, decision trees can be seen as a disjunction of conjunction of attribute values of instances, where each path from the root is a conjunction of attribute tests and the tree itself is a disjunction of these conjunctions[4]. Since no real implementation of Decision Trees for Recommender Systems has been done in this work and a lot of research and books is available online, no more technical details on the algorithm itself will be provided.

Often, a decision tree is not enough to have a good accuracy. Ensemble methods are used to build more than one tree and join the results. The bagging/bootstrap technique build a set of M base model, by picking random samples with replacement from a dataset of instances. In a nutshell, it generates M datasets in order to obtain M different classifier models. This allows to build an ensemble of decision trees, called random forest. After a phase of bagging, it builds random trees which only use a random subset of the attributes. This combination of methods allows to generate a very accurate model, which accuracy generally stands out with respect to other more traditional ML algorithms.

A state-of-the-art implementation of the Random Forest algorithm for solving the recommendation problem on the MovieLens 1M dataset is proposed in [2]. The authors claim to have achieved an average MAE score of 0.81 and an average RMSE score of around 1.02 for forests of T trees, where $T = 10, 20, 50, 100$.

2.3.2 K-Nearest Neighbours

The k-Nearest-Neighbours[36] (kNN) is a non-parametric classification method, which is simple but effective in many cases[11]. For a data record t to be classified, its k nearest neighbours are retrieved, and this forms a neighbourhood of t. Majority voting among the data records in the neighbourhood is usually used to decide the classification for t with or without consideration of distance-based weighting. A

state-of-the-art implementation of the KNN algorithm for solving the recommendation problem on the MovieLens 1M dataset is proposed in [2]. The authors claim to have achieved an average MAE score of 0.875 and an average RMSE score of around 1.07 for k neighbours, where $k = 10, 30, 50, 100$.

2.3.3 Matrix Factorization

Matrix factorization[23] (MF) is a technique which computes a latent factor model of a system based on user-item interaction. MF creates a user-defined number of features N by breaking the original data features R (ratings), of dimension $M \times N$, into the product of two matrices U (User Matrix of 2.5), of dimension $M \times D$, and V (Item Matrix of 2.5), of dimension $D \times N$. MF uses an iterative approach to modify the initial values of U and V so that the product approaches R. When the approximation error converges or the user-defined number of iterations is reached, MF terminates. This allows MF to obtain D features which goal is to try and describe better the relationship between the original data: in the case of the MovieLens dataset, this can be seen as a way of combining a vector describing the user and a vector describing the movies seen. From this then, it is possible to predict, via simple regression, the ratings for unseen movies or cluster groups of similar users.

A state-of-the-art implementation of the Matrix Factorization algorithm for solving the recommendation problem on the MovieLens 10M dataset is proposed by Ivarsson and Lingren[18]. The authors claim to have achieved an average RMSE score of around 0.823 with 10 features extracted.

2.3.4 Clustering Methods

Clustering[8] is the distribution of a set of instances of examples into non-known groups according to some common relations or affinities. This means that clustering algorithms allow for classification of items into groups which are not known at the beginning, cluster, according to some relations or affinities between the items themselves. Given a set of instances I, a number of cluster K, an objective function

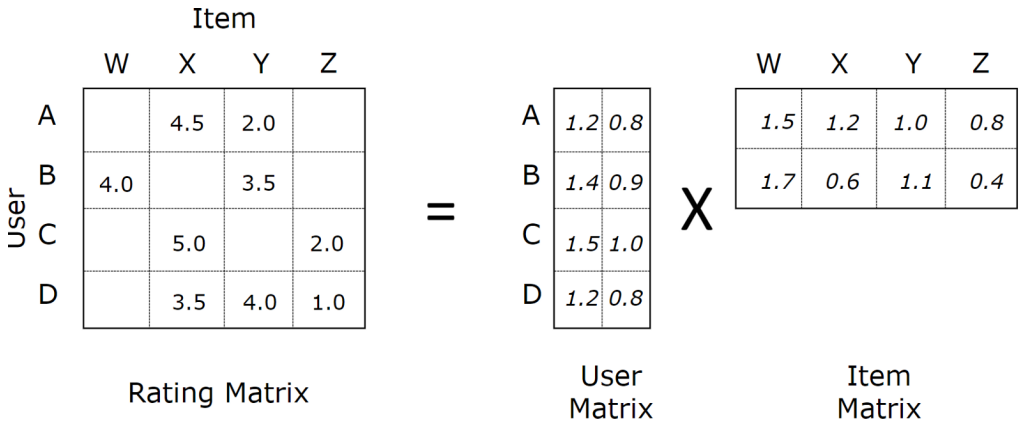

Figure 2.5: Matrix Factorization Example

$cost(C, I)$, a clustering algorithm computes a set C of instances with $|C| = K$ that minimizes the objective function $cost(C, I) = \sum_{x \in I} d^2(x, C)$ where $d(x, C)$ is the distance function between x and C, and $d^2(x, C)$ is the distance from x to the nearest point in C.

Examples of clustering applications are the market segmentation of customers, social network communities analysis, and of course product recommendation. Among the most known algorithms there are *K-Means* (or its variation *K-Means++*), *DB-Scan* (and other Density Based methods) and *BIRCH*.

2.4 Deep Learning

2.4.1 Deep Neural Networks: general concepts

An artificial Neural Network (ANN) is a nonlinear system inspired by biological neural networks, i.e. the brain. An artificial neural network allows to learn a mapping from a given input space to a desired output space. The life of a typical ANN is characterized by two phases: the training phase, which can be either supervised, unsupervised or hybrid, and the prediction phase. The structure of an ANN is usually composed of an input layer, an output layer and one or more hidden layers. Each layer of an artificial neural network is composed of several neurons. Each neuron is connected to every other neuron in the adjacent layer, that is: it takes as input the outputs of the previous layer and combines them linearly to generate the stimulus that feeds the activation function. Typical activation functions are sigmoid, hyperbolic tangent, Rectified Linear Unit (ReLU)[25]. The output of each neuron is thus a number that will be the input of the neurons at the following layer.

Deep learning means using an artificial neural network with several layers between the input and the output. A deep architecture is applied in fields such as computer vision, speech recognition, bioinformatics, audio recognition, etc. Deep learning is a class of machine learning algorithms that uses a cascade of multiple layers of nonlinear processing units for feature extraction and transformation to solve a particular task.

When we talk about learning, what that is referring to is getting the computer to find a valid setting of weights and biases so that it will actually solve the problem at hand. To know which connection weights must be modified and by how much to perform correctly the desired classification task we must use an algorithm that efficiently modifies the different connection weights to minimize the errors at the output. Optimization algorithms used for training deep models differ from traditional optimization algorithms in several ways. Machine learning usually acts indirectly. In most machine learning scenarios, we care about some performance measure P, that is defined with respect to the test set and may also be intractable. We therefore

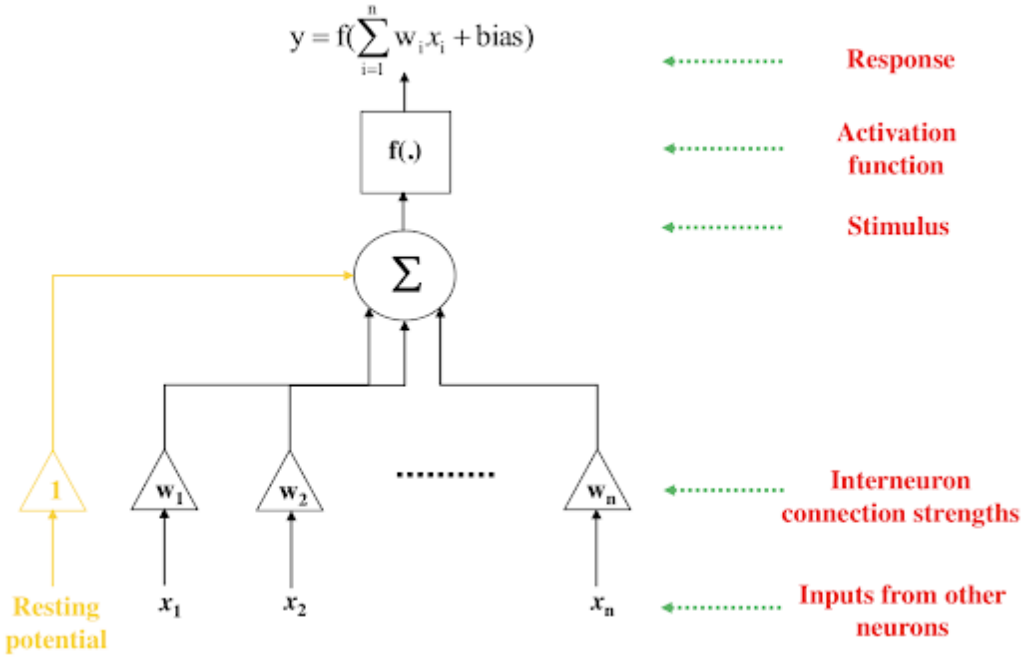

Figure 2.6: Artificial Neuron and mapping function

optimize P only indirectly. We reduce a different cost function $L(\Theta)$ in the hope that doing so will improve P. This is in contrast to pure optimization, where minimizing $L(\Theta)$ is a goal in and of itself. Typically, the cost function can be written as an average over the training set.

One algorithm that has hugely contributed to neural network fame is the iterative backpropagation algorithm. The backpropagation algorithm is a gradient-based search method which allows finding a (local) minimum of the loss (or cost) function that one wants to minimize. It works in three phases. In the forward propagation phase, the input propagates through the network to produce an output that differs from the actual target value by a quantity defined by the loss function. This error is back-propagated through the network to the first hidden layer and a weight updating

phase is now performed, aiming at minimizing the cost function. The backpropagation algorithm is iterated up to convergence. The convergence velocity depends on the (1) the complexity of the considered classification problem (I/O relationship), (2) complexity of the approximating function (i.e. number of weights and biases, number of hidden layers) and (3) value of the learning rate, a parameter that can be fixed or adaptively changed and indicates at which pace weights are updated.

Three different weight updating (training) strategies could be adopted: (1) training by pattern: for every observation, estimate the errors and update weights (one example at a time is used) (2) batch training: optimization algorithms that use the entire training set are called batch or deterministic methods because they process all the training samples (3) stochastic (mini-batch) training: in Stochastic Gradient Descent (SDG), each parameter update is computed with respect to few training samples (the choice of the training samples introduces randomness), by subtracting the gradient of the loss with respect to the parameter, scaled by the learning rate. Stochastic optimization is preferred since (1) the algorithm converges much faster in terms of total computation if they are allowed to rapidly compute approximate estimates of the gradient rather than slowly computing the exact gradient (2) there might be redundancy in the training set and we may find large number of examples that all make very similar contributions to the gradient.

However, SDG can be sometimes slow and SDG variants have been used to achieve state of the art performance. One of the most popular backpropagation algorithms is Adam. The name Adam derives from the phrase “adaptive moments”. It is an adaptive learning rate optimization algorithm and it is generally regarded as being fairly robust to the choice of hyperparameters.

A deep learning models is usually trained with many epochs. One epoch is when an entire dataset, divided into several smaller batches, is passed forward and backward through the neural network once. The number of iterations is defined as the number of batches needed to complete one epoch. More epochs are necessary because the optimization methods are all iterative processes and updating the parameters with one single pass is not enough and it could lead to underfitting. However, it is

important not to tune excessively the network on the training samples so as to leave room for a correct classification of novel patterns.

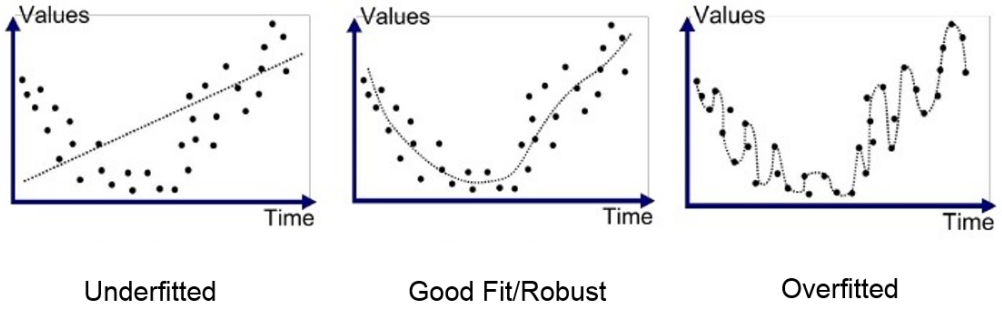

Figure 2.7: Visualization of underfitting, optimal fitting and overfitting

2.4.2 Deep Learning for Recommender Systems

Deep Learning has become a buzzword after a 2006 paper by Hinton[16], which described how to train deep neural networks using unsupervised layer-wise training (Deep Belief Networks). Since then, there has been an immense amount of research in the area with impressive results in areas such as computer vision, speech recognition and natural language processing. Some researchers, and more generally Deep Learning enthusiasts, believe that Deep Learning is the technology that will bring us closer to the definition of General AI[28]. While waiting for General AI to become a reality, a lot of investments has been put in place to industrialize Deep Learning technology in other areas, including solving the Recommendation Problem. However, is there a need for DL-based Recommender Systems?

From the previous definition of Deep Learning, it makes sense to adopt DL-based algorithms when the problem is complex enough not to be easily solved by statistical or ML-based approaches, or when the data available for learning is big enough to allow exploitation via complex algorithms.

When considering standard Collaborative Filtering Recommender Systems, one might think that ML models are developed enough to be able to face the Recommendation Problem without any particular issues - and one would be definitely correct in its statement. However, along with the enormous production of data in the most recent years, the variety of possible combinations in a recommendation problem has increased exponentially, making the problem way more complex than it was before, where virtually unlimited catalogs and ever-growing user bases were not a reality that every company had to face. Machine Learning algorithms that could easily generalize the problem at hand in reasonable computational time, could not handle the vastity and diversity of databases of giants like Netflix or Amazon.com or Spotify.

Contrary to traditional Machine Learning approaches, deep neural networks are capable of modelling the non-linearity in data with nonlinear activations such as *relu*, *sigmoid*, *tanh*, etc. This property makes it possible to capture the complex and intricate user item interaction patterns, that were previously missed or needed a manual feature extraction by a domain expert.

Deep neural networks proves to be even more useful in representation learning in recommendation models. Even though abundant data is available in real-world applications in the form of descriptive information about items and users, it is very difficult to be exploited by machines, due to its nature of being expressed in a way which is closer to our natural world. Making use of this information provides a way to enhance our understanding of items and users, therefore resulting in a better recommender. Similar results could be obtained in a traditional ML approach, but would require once again more manual and complex feature engineering, provided by a domain expert and that needs to be included in a pre-processing pipeline.

Over the last years, there have been multiple publications in the area of deep learning for recommender systems. These works can be grouped into the following categories:

- **Feed-forward Networks and Autoencoders**[33] [42], methods used mainly

for Collaborative Filtering, since it allows the implementation of a Deep Learning equivalent of the Matrix Factorization previously described. One of such algorithms is Deep Factorization Machines[10].

- **Embedding methods**, which tries to learn different representations called embeddings of users, items[40] or both, in order to use them directly to provide recommendations or to use as input to other supervised learning methods that provide recommendations. While Matrix Factorization or Singular Value Decomposition (SVD) can be seen as an Embedding Method, 2Vec-type embedding techniques[26] can prove more effective and flexible.

- **Deep Feature extraction**, methods that focus on using deep networks to perform feature extraction on the item features. These features are then either used in a hybrid collaborative filtering recommender system[13] or often the feature extractor is part of a larger deep architecture that also models other aspects of the data.

- **Session-based Recommendation with Recurrent Neural Networks**, methods leveraging the concept of "sessions" of user interactions, such as music listened consecutively, binge watching seasons of a series, shopping cart filling. The best performing models for sequential data are Recurrent Neural Networks, more in specific GRUs are used in these models as they seem to perform equivalently to LSTMs with a slightly less memory usage[14][15].

In this work, a few of these solutions to the Recommendation Problem have been re-implemented in a popular Deep Learning framework, *Keras*, comparing the results obtained on the different datasets. Due to the number of different algorithms tested during the redaction of this work, description of the Neural Network architectures as well as other technical details have been collected and included in the following section.

2.4.3 Deep Learning for Embedding Learning

Embedding Learning has been introduced in section 2.4.2 of this work as a possible application of Deep Learning to understand the representation of complex structures in a more computable way. However, it would be best to dive further into the topic, by providing technical details and possibly an example.

Let's consider the case of a document. Analysing a document can be very challenging, for both size and structure. To do so, two different levels of representations are being used: document-based representation, in which the whole document is represented as a vector, and word-based representation, in which a word is represented as a vector.

A document can be considered as a bag of words[45]: the document can be represented by a vector made of the number of occurrences of the word w in document d, removing stopwords and applying other filters where necessary. The algorithm is pretty straightforward, consisting of loops over the different documents, and building a map of accepted words and their occurrences in the corpus. The corpus then becomes a N x M term-document matrix, where N is the size of the vocabulary and M is the size of the corpus; each cell represents the frequency of a word (column) in a document (row). The term-document matrix can be very big for bigger datasets with multiple and big documents, and requires to loop over the entire document set while looking for every word in the map; moreover, it does not capture the order of the terms in the document.

Another way to represent a document is *TF-IDF*[30] (term frequency - Inverse Document Frequency). TF-IDF is a statistical measure used to evaluate the representativeness of a word for a particular document in a collection of documents: TF-IDF of a word gives a product of how frequent this word is in the document multiplied by how unique the word is with respect to the entire corpus of documents. Words in the document with a high TF-IDF score occur frequently in the document but not in all of the documents, therefore providing the most information about that specific document. This value grows proportionally to the occurrences of the word in the document (TF) but its effect is countered by the occurrences of

the word in every other document (IDF). TF-IDF is computed as:

$$TF - IDF(w, d) = TF_{w,d} * log_2(M/DF_w)$$

where M is the number of documents, TF is the Term Frequency, either number of occurrences of w in d or boolean value of presence, and DF is Document Frequency, number of documents with the word w.

Word can be represented as vectors as well. The idea behind all of the word embeddings is to capture with them as much of the semantical, morphological, context, hierarchical, etc. information as possible. Word Embedding[26] is, in other words, the vector used to represent the word, which embeds the semantic link between words and is computed from the word context. This is done by associating a numeric vector to every word in a dictionary, such that the distance (e.g. L2 distance or more commonly cosine distance) between any two vectors would capture part of the semantic relationship between the two associated words. The geometric space formed by these vectors is called an embedding space. This can be done by training a Neural Network on a big dataset, generating a dense vector v for each word w from the vocabulary V of the dataset.

For example, two words occurring in a same context have high semantic proximity, therefore their vector will be similar. This vectorial representation proves to be very useful for similarity computation, just by computing simple vectorial operations in the embedding space:

Two possible Neural Network architectures have been proposed for word embeddings computation:

- **CBOW (Continuous Bag Of Words)**: learns how to predict a word according to its context, so it tries to assign weights so that given the context as input, it predicts the target word; CBOW is better for smaller datasets, as it is more resilient to distributional information by treating an entire context as one observation;

- **Skip-Gram**: it's the opposite of CBOW, meaning that it learns how to predict a context given a word, so it tries to assign weights so that given the word as

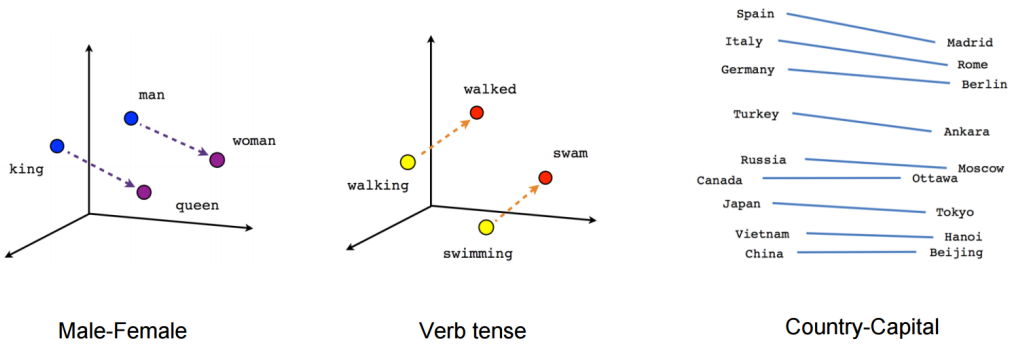

Figure 2.8: Distance between word vectors

input, it predicts the context; by treating each target-context pair as a new observation, Skip-Gram is better for larger datasets.

It is possible to find pre-trained embeddings online, trained by multiple GPU-powered nodes in a cluster on huge datasets. Examples freely available online are GloVe, trained on Wikipedia texts, different crawls of the Internet and Twitter tweets, and Facebook FastText, which provides pre-trained word embeddings for 157 different languages. For the sake of experimentation, in this paper the GloVe word embeddings have been used for NLP operations.

In order to quantify similarity between vectorized items, it is possible to use functions borrowed from statistics and related fields, among which the most commonly used **cosime similarity**, calculated as the dot product between two vectors divided by their magnitudes.

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

Embedding vectors pointing in the same direction will receive high cosine similarity scores, while vectors in opposite directions will receive a very low similarity score.

Chapter 3

Deep Recommendation Systems

3.1 The Technique

3.1.1 The MovieLens Dataset

In this section of the work, the proposed algorithms are analyzed, discussed, implemented and tested on multiple variations of the **MovieLens dataset**. These datasets are a product of member activity in the MovieLens movie recommendation system, an active research platform led by GroupLens Research, a human-computer interaction research lab at the University of Minnesota, that has hosted many experiments since its launch in 1997. The MovieLens datasets provide the rating data sets collected from MovieLens website for research, counting 26,000,000 ratings and 750,000 tags applied to 45,000 movies by 270,000 users in its most complete form, the dataset *MovieLens-Latest*. The MovieLens datasets are used widely used in education, research, and industry, for running experiments on recommender systems, online communities, mobile and ubiquitious technologies, digital libraries, local geographic information systems[12].

The dataset files are written as comma-separated values files with a single header

row, encoded in UTF-8. Columns that contain commas (,) are escaped using double-quotes (").

The main file is *ratings.csv*. Each line of this file represents one rating of one movie by one user, with format $< userId, movieId, rating, timestamp >$. The lines within this file are ordered first by *userId*, then, within user, by *movieId*. Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars). Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

Movie information is contained in the file *movies.csv*. Each line of this file represents one movie, with format $< movieId, title, genres >$. The *movieId* feature is consistent with the one in *ratings.csv*, while the titles are entered manually or imported from https://www.themoviedb.org/, and include the year of release in parentheses. The *genres* feature is a categorical feature which can assume up to 5 among 19 different values (*Action, Adventure, Animation, Children, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western, (no genres listed)*).

Movie identifiers are also contained in the file *links.csv*. Each line of this file represents one movie, and contains matching movie identifiers that can be used to link to other sources of movie data on other web portals such as IMDB and TMBD, both well known data sources for movie-related information.

3.1.2 Implementation

The MovieLens dataset, in its most complete form, counts more than 20M lines. Even when taking into account just the *ratings* file, which only has three features (namely, $< userId, movieId, timestamp >$), the algorithms proved to be computationally intensive, taking longer time than needed when run on pure CPU of a Macbook Pro 2017. That's why, for most of the computations on the bigger datasets, a cloud-based instance has been used, provided by **Amazon SageMaker**.

Amazon SageMaker is a fully-managed platform enabling developers and data

scientists to quickly and easily build, train, and deploy machine learning (ML) models at any scale. Amazon SageMaker helps to speed ML adoption with modules that can be used together or separately in the build, train, and deployment processes allowing scaling resources up and down dynamically, according to user needs. Amazon SageMaker supports most major frameworks including TensorFlow, MXNet, Keras, Gluon, PyTorch, Caffe2, Chainer, Torch, and Microsoft Cognitive Toolkit (CNTK). Moreover, it is pre-configured and optimized for CUDA 9 library support for NVIDIA GPUs. For the most compute-intensive tasks, instances of the *ml.p3* family have been used, in particular *ml.p3.16xlarge* instances, with 64 vCPUs, 8 NVIDIA Tesla V100 GPUs and 128 GB of RAM. Furthermore, SageMaker exposes to the user a scientist-friendly Jupyter Notebook, a very common tool for data exploration as well as model testing.

SageMaker support, among others, **TensorFlow**. TensorFlow is an open-source software library for dataflow and differentiable programming across a range of tasks. It is currently one of the most used library for neural network development, despite the recent spike in attention towards *PyTorch*, especially for computer vision tasks. However, TensorFlow API is not what one would define as high-level, or easily readable by a human. Although it is without a doubt very powerful and versatile, the learning curve for this library is steep and discourages many deep learning beginners. In order to make this work available and understandable by the widest user base, all of the algorithms have been implemented with the **Keras** framework, with Tensorflow backend. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. Keras was developed with a focus on enabling fast experimentation since, according to its creator, "being able to go from idea to result with the least possible delay is key to doing good research". Moreover, Keras runs seamlessly on CPU and GPU, allowing a very fast prototyping on a less powerful machine such as the MacBook, as well as deployment on the *p3.16xlarge*, changing but one line of code. Keras has proved to be very versatile and a key choice in order to achieve certain levels of modularity and extensibility of some of the algorithms proposed in 3.4.

Hyperparameters of the training jobs are shared among algorithms and coded into an higher level class. They include:

- 20% of the dataset used for testing
- 25% of the training dataset used for validation
- 4096 instances per batch
- 250 epochs, with EarlyStopping set to 5 epochs
- latent features for the embeddings set to 64

All of the algorithms have been measured with **MAE (Mean Average Error)** and **RMSE (Root Mean Squared Error)** rounded to the third decimal number, via the implementation provided by the *scikit-learn* package, one of the most used libraries for Data Science on Python.

Mean absolute error (MAE) is a measure of difference between two continuous variables. MAE is the average absolute difference between the obtained output and the target output. It is computed as:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}(p_{ui} - r_{ui})$$

where p_{ui} is the predicted rating for user u and item i, r_{ui} the actual rating and N the number of predictions.

Root Mean Squared Error (RMSE) is a statistical metric that represent the standard deviation between a set of estimated values to the actual values. In recommender systems it has been used to measure how far from the true values a set of predictions was[34]. The RMSE is calculated as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_{ui} - r_{ui})^2}$$

where p_{ui} is the predicted rating for user u and item i, r_{ui} the actual rating and N the number of predictions. Both evaluation metrics have been used for the Netflix prize as well as for a lot of research on the topic of recommender systems.

3.2 Collaborative Filtering

3.2.1 Deep Matrix Factorization

The first algorithm proposed in this work and implemented with the Keras framework is the Deep Learning interpretation of the Factorization Machines proposed in [31]. In this paper, the author presents a traditional ML approach to the recommendation problem, which aims to learn feature interactions for recommendation. A factorization machine is defined as a general-purpose supervised learning algorithm that is designed to capture interactions between features within high dimensional sparse datasets.

A factorization machine model estimates a function $\hat{y}$ from a feature set x_i to a target domain, real-valued for regression and binary for classification. The factorization machine model uses a factorized parametrization to capture the pairwise feature interactions. A mathematical definition is the following:

$$y = w_0 + \sum_i w_i x_i + \sum_i \sum_{j>i} < v_i, v_j > x_i x_j$$

where the w_0 term represents the global bias, the w_i linear terms model the strength of the i-th variable, the $< v_i, v_j >$ factorization terms model the pairwise interaction between the i-th and j-th variable. The global bias and linear terms are the same as in a linear model. The pairwise feature interactions are modeled in the third term as the inner product of the corresponding factors learned for each feature.

The factorization terms are therefore real-valued vectors of variables, which pairwise interaction is used to learn global interactions in the model, by means of an

inner product operation.

Given the previous definition of the recommendation problem in Section 2.1, it is correct to assume that most of the time the source of truth will be a consumption/rating dataset formed by a collection of $< user, item, rating >$ tuples. According to the original implementation in the *Factorization Machine* paper, the training data should be structured as one-hot encoded vectors. One-hot encoding means transforming categorical features in multiple binary features. For examples, a set of users $U = [A, B, C]$ becomes:

$$\begin{bmatrix} A & B & C \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

However, embeddings are another effective way of encoding real values to a constrained space of values, as discussed in a previous section of this work. Embeddings are a fully connected layer that projects the sparse representation to a dense vector, which can be used as the latent vector for user or item. It is therefore possible to use Keras Embedding layer to reduce the dimensionality of the sparse data generated by learning a latent space representation of the user-item interaction by computing the inner product (Keras Dot Layer) of the obtained embeddings and finally a Dense layer composed of one neuron with activation function set to ReLu, since the value to be predicted is real, in the range $[1, 5]$. Keras Embedding layers works by encoding positive integers to dense vectors of fixed size; this proves extremely useful with our definition of recommendation problem, since the rating dataset is already in the form of triplets $< user, item, rating >$ where user and item are integers, which can be considered as the index in a one-hot encoded vector. The proposed NN architecture would look like in Figure 3.1.

In the DeepMF architecture proposed above, no real constraints have been given to the network regarding how to learn embeddings from the input data. However, some research[20] have shown that applying a non-negativity constraint to the matrix factorization problem, and in this case to the learning of embeddings,

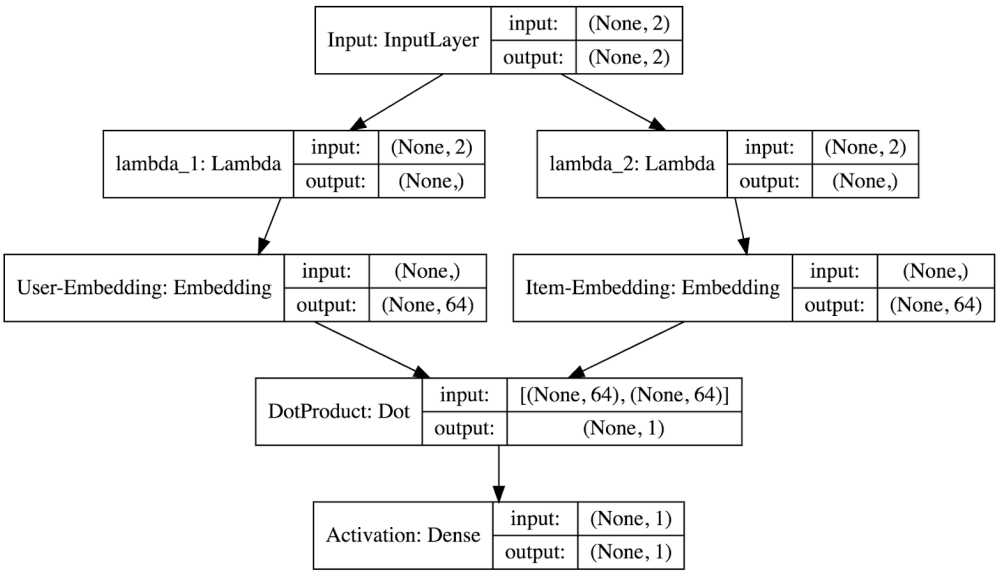

Figure 3.1: Keras DeepMF model plotted in Jupyter Notebook

proves to be extremely useful and leads to improved results. Moreover, non-negative matrix factorization popularity is also thanks to its ability to automatically extract sparse and easily interpretable factors. Successful application to non-negative matrix factorization include image processing, text mining, computational biology, clustering, music analysis, community detection, and most importantly collaborative filtering[24]. While with the traditional ML approach a completely new algorithm is needed to implement non-negativity in the factorization, for example the NMF algorithm proposed in the sklearn.decomposition library, including this constraint in Keras is as easy as changing a parameter in the Embedding layer. Therefore the architecture does not change with respect to the one proposed previously, and the results are directly comparable.

The algorithm DeepMF, without the non-negativity constraint, performed a nice MAE of 0.726 and RMSE of 0.951 (Figure 3.3). The same algorithm, with the non-negativity constraint enabled, performed improved evaluation metrics, with MAE

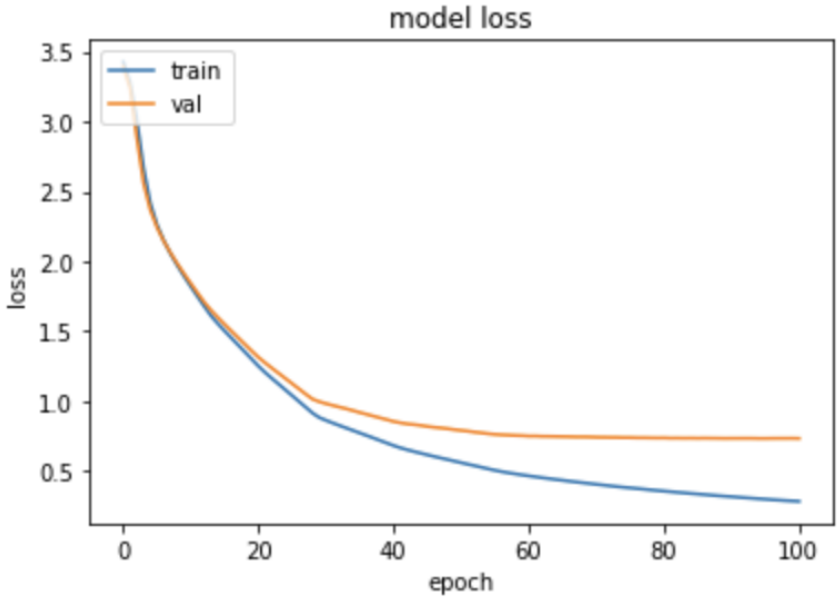

Figure 3.2: Keras DeepMF loss plotted over 100 epochs

of 0.69 and RMSE of 0.893 (Figure 3.3).

3.2.2 DeepDenseNet

Matrix Factorization combines the learnt Embeddings via the inner product (or Dot product), learning the pairwise interaction between variables. Nevertheless, the inner product is a linear operation, which might cause missing some some of the non-linear interactions between the $< user, item >$ pairs. In order to learn high-order non-linear feature interactions it is possible to use a deep feed-forward neural network (a Deep Dense Network[43]). The user embedding and item embedding are then fed into a multi-layer neural architecture to map the latent vectors to prediction scores. Each layer of the neural CF layers can be customized to discover certain latent structures of user–item interactions. The dimension of the last hidden layer X determines the model's capability. The final output layer is the predicted

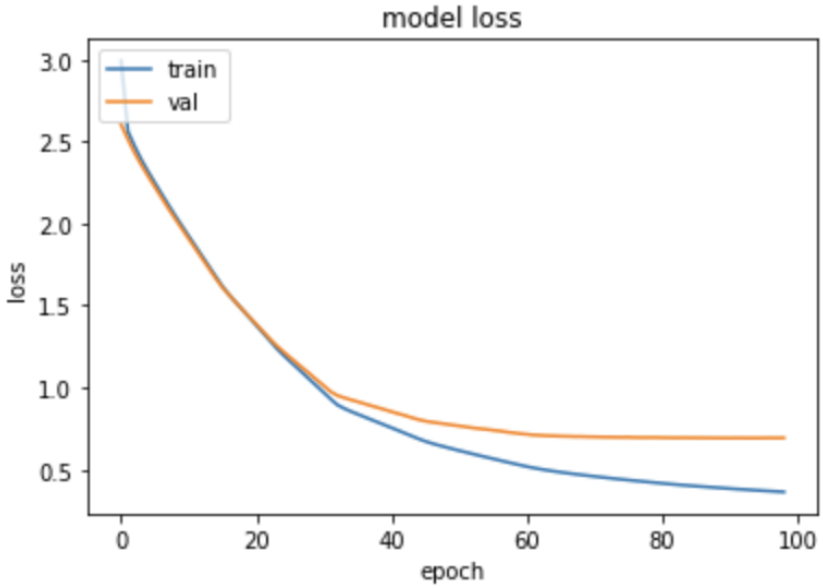

Figure 3.3: Keras Non-negative DeepMF loss plotted over 100 epochs

score $\hat{y_{ui}}$, and training is performed by minimizing the mean average error between $\hat{y_{ui}}$ and its target value y_{ui}. With three hidden layers in the dense section of the architecture, ignoring the optional dropout layers, the new architecture would look like as the one illustrated in Figure 3.4.

The algorithm DeepDenseNet behaves quite differently from the DeepMF. The deep nature of its architecture allows it to converge to a minimum (possibly the global one) in way less epochs with respect to the DeepMF algorithm. This can be caused by an excessive number of neurons per layer, causing it to overfit. The algorithm scores results comparable to the previous algorithm, with a MAE of 0.684 and RMSE of 0.91. Comparable results with a similar model loss graph were obtained with the non-negativity constraint enabled, achieving a MAE of 0.686 and RMSE of 0.915.

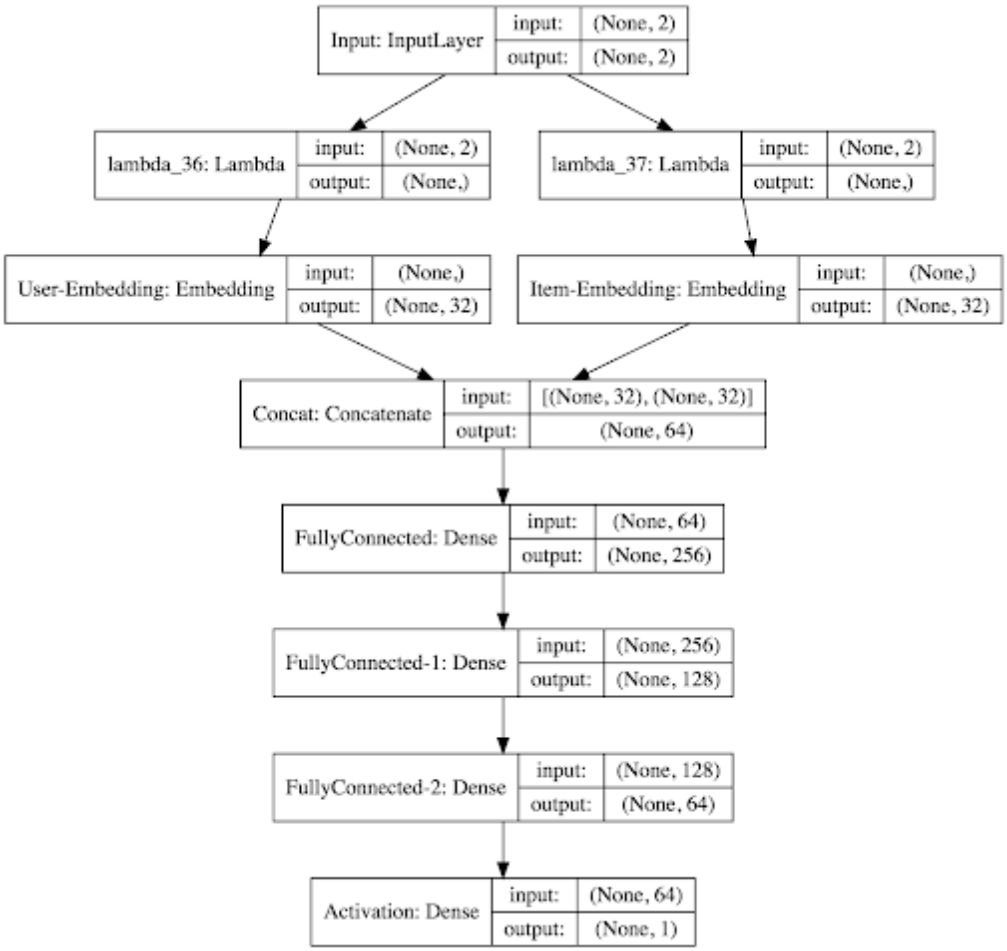

Figure 3.4: Keras DeepDenseNet model plotted in Jupyter Notebook

3.2.3 DeepWideNet

The first two proposed algorithms, DeepFM and DeepDenseNet, can be used at the same time to try and provide better insight from the data. The idea is similar to the one proposed in [10]. The goal of the presented paper is to propose a Neural Network which maximizes CTR, Click-Through Rate, for recommender systems.

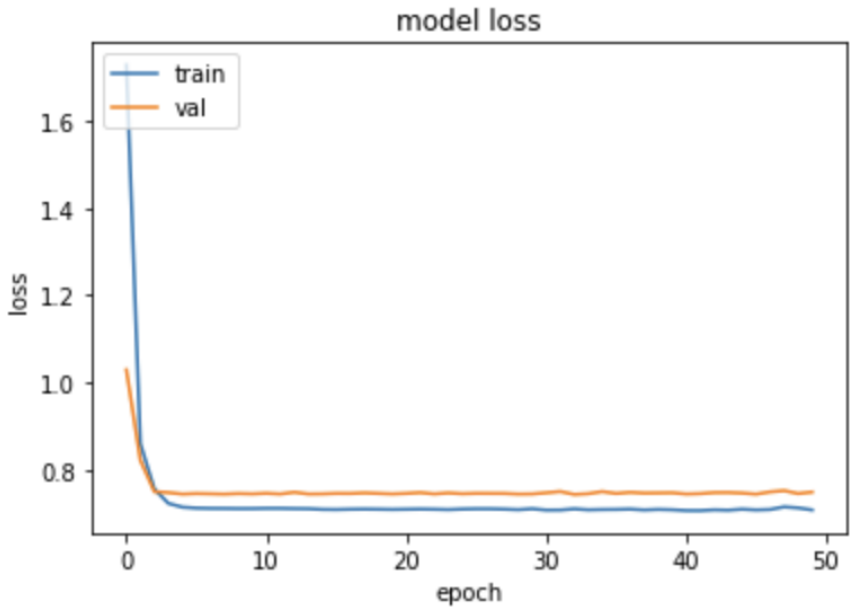

Figure 3.5: Keras DeepDenseNet loss plotted over 50 epochs

CTR is defined as a ratio of number of clicks to number of impressions of an item for a query or on a page[41].

A very similar approach can result valid in the Recommender Systems world where, instead of optimizing the probability of a click-through (computed via a final layer with a sigmoid as activation function), one can try to predict a rating as in a standard regression problem. In addition, such an architecture is easily extendable and parameterizable to include other features apart from the tuple $< userId, movieId >$ in order to extend the Collaborative filtering algorithm with content information (more on this in the Hybrid Recommender Systems section).

The proposed architecture (Figure 3.6) has two components. The "Wide" component is the **DeepMF** architecture: it computes the inner product of the the computed embedding for user/item. The "Deep" component is instead the **DeepDenseNet**: after concatenating the embeddings, it runs them through a deep dense

network. Finally, the two results are concatenated and run through a single 1-neuron layer to compute the final result of the regression. The two components belong to the same network so that the same embeddings are used during the training process.

The results obtained from this more complex network are better, as expected. The network reaches convergence much quicker than the others tested previously (figure 3.7), while achieving better metrics with respect to the single components: MAE of 0.684 and RMSE of 0.908. Similar tests have been performed by including a Dense layer right after the Dot product layer in the Wide component, however results did not improve at all and in some cases even resulted in worse performances.

A variation to the proposed DeepWideNet is the one proposed in figure 3.8. Here, instead of feeding to the Wide component the Dot product between the user and item embedding, the two embeddings are concatenated along the smaller axis and directly forwarded to the final Dense layer, which computes the regression for the rating.

The proposed network performs very similarly to the previous DeepWideNet model (figure 3.9), with MAE of 0.696 and RMSE of 0.924 .

3.3 Content-based Recommendation

As presented in Section 2.4.2 and 2.4.3 of this work, Deep Learning proves to be extremely useful for Embedding Learning, allowing to represent and exploit unstructured data as well as categorical (or, more generally, non scalar) features with a vector of scalar values data for interaction learning. While continuous values are easy enough to include in the network, due to their scalar nature, categorical or even long textual features can prove to be quite more difficult to include without the proper encoding. Scalar values can be directly digested by feed-forward neural network and treated as input by the first layer. Examples of scalar features include age, length of the movie, year of release of the movie, etc. Categorical features, such as user profession, movie genre, movie category, have to be encoded before being fed

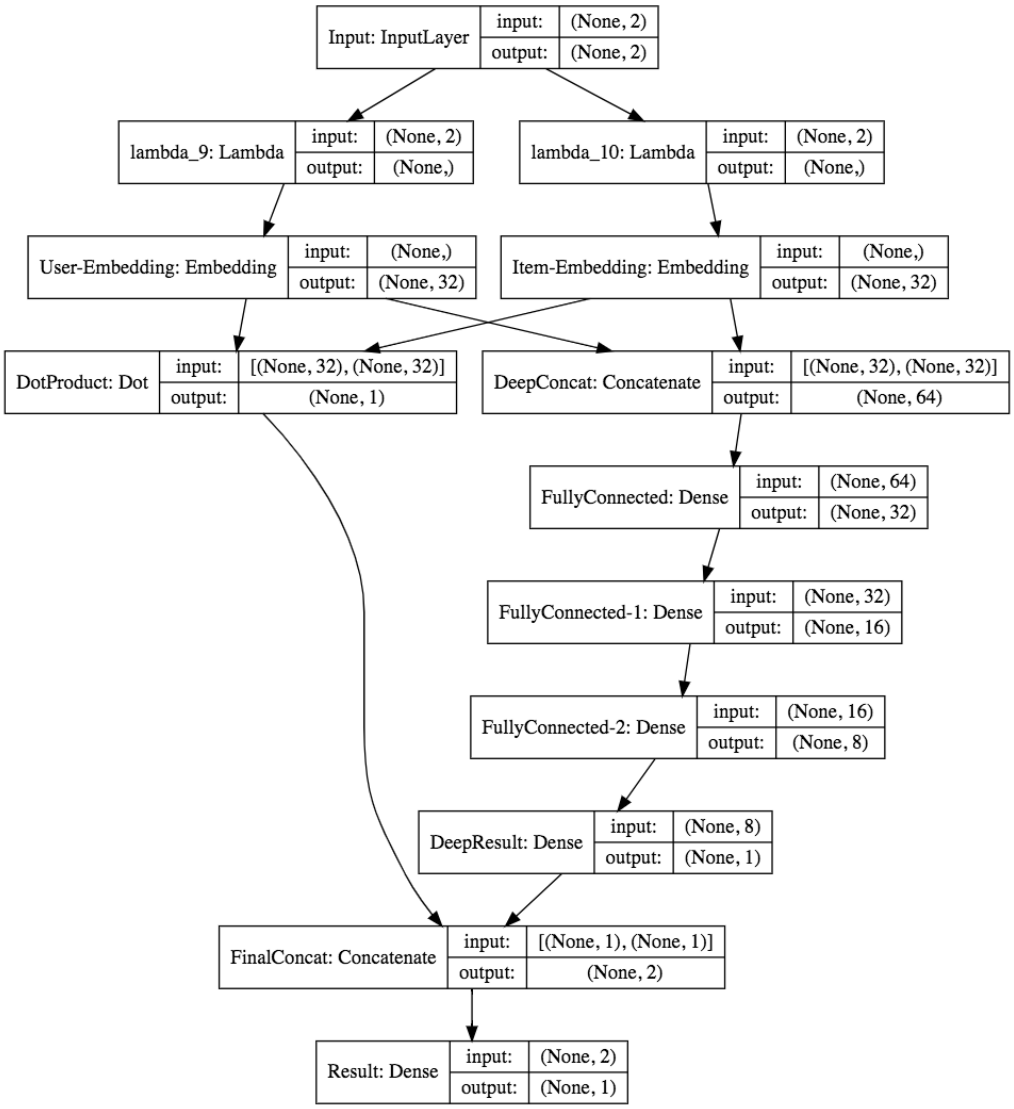

Figure 3.6: Proposed architecture for DeepWideNet

to the embedding layer. Finally, a lot of information can be extracted from unstructured data such as review text, movie summary, user description. This textual data

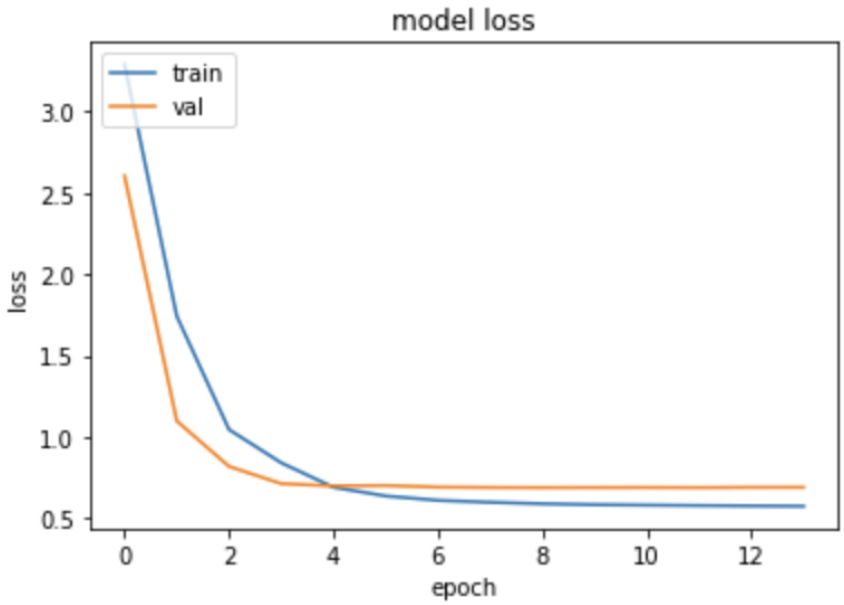

Figure 3.7: DeepWideNet Loss plot

must be processed before being fed to a deep neural network, extracting as much knowledge and information as possible.

For the following implementation, once again the tests have been done on the MovieLens dataset. Although the latest version of the MovieLens dataset does not provide any additional information regarding the user base, probably due to privacy concerns, multiple features regarding the movies are available and have been used to test the effectiveness of the implementations. In particular, the MovieLens dataset includes a mapping between *movieId*, the identifier present in the ratings file, and *imdbId*, the identifier usable in "The Open Movie Database", which provides RESTful web service to obtain movie information as well as artwork. By doing so, it is possible to obtain most of the unstructured features that will be used for demonstration in this section of the work as well as in Section 3.4 with the Hybrid Recommendation System.

This work will foucs mainly on **Recommendation via Text**, using information

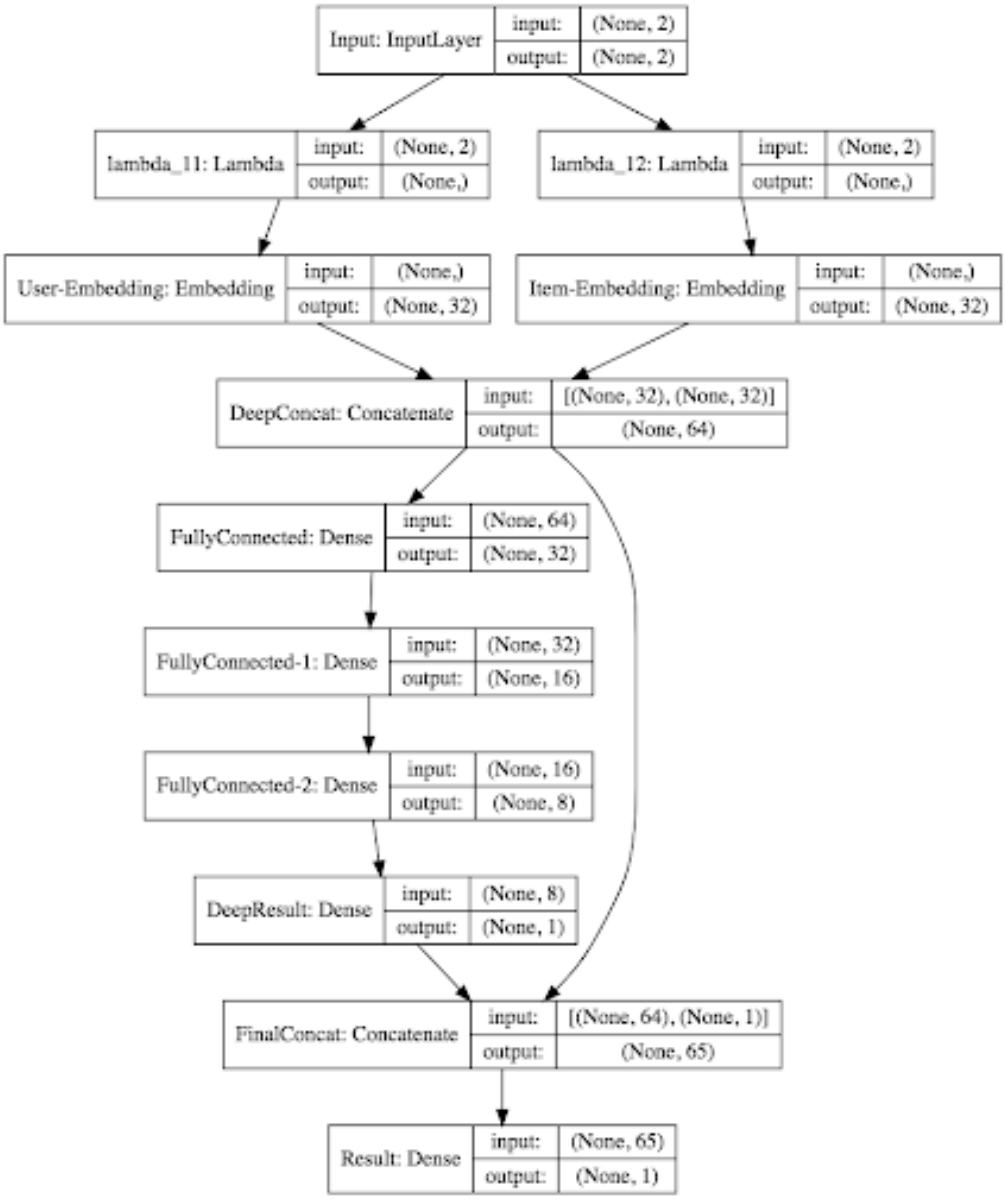

Figure 3.8: Proposed architecture for a variation of DeepWideNet

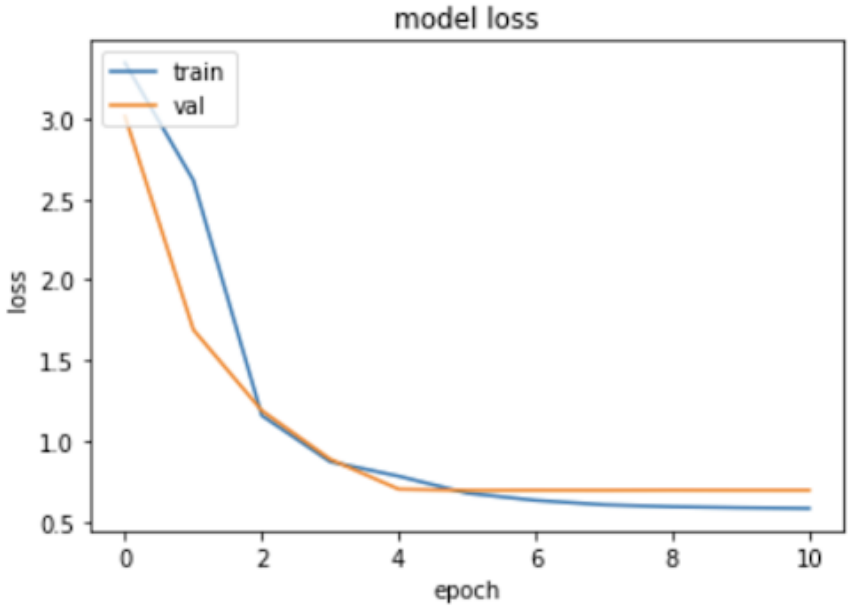

Figure 3.9: DeepWideNet variation Loss plot

such as reviews, movie summaries, description, etc. pre-processed with GloVe embeddings (freely available online); another possible approach is **Recommendation via Artwork**, using high-level features extracted from a pre-trained Deep Neural Network (VGG16). This second approach has been tested with the MovieLens Latest Small dataset, but it lead to sub-optimal results and very long runtime, especially on a MacBook Pro without dedicated GPU. Further analysis of the effect of artwork-based recommendation, especially for products to be sold (such as the products from BeerAdvocate dataset) is one of the possible Future Works presented in Chapter 5 of this work.

3.3.1 Recommendation via Text

One of the most interesting piece of information available in the Open Movie Database is the overview of the movie. It is generally a 3 to 5 lines summary of the movie,

composed of multiple sentences. It is possible to transform the overviews into a Term Frequency-Inverse Document Frequency (TF-IDF) representation. The TF-IDF vectorizer in *scikit-learn* can be used to extract the most important words from the overview, by a proper use of the *transform* and *inverse_transform* primitives of the TF-IDF vectorizer class. All of the extracted words with TF-IDF form the **vocabulary**. Once the most representative words are extracted from every overview, it is possible to obtain the embeddings from the GloVe dataset associated to the vocabulary, building an **embedding matrix**. This embedding matrix can then be used in Keras Embedding layer as weight of the layer itself, setting the parameter *Trainable* of the layer itself to False. By setting this property, the weights will not be updated during the training process. A neural network built as such will output directly the GloVe Embedding from this layer during inference, allowing them to be used in combination with other layers for more complex architectures.

CBF-Average

The first architecture proposed according to the above described approach is **CBF-A** illustrated in Figure 3.10. After having extracted with TF-IDF a set of words of length n defined at compilation time, which best represent the summary of the movie, these words are encoded and their GloVe embedding are extracted with the Keras Embedding layer. Since every word is represented by an m-dimensional array of scalar values, the Embedding layer returns a mxn matrix of scalar values; by computing the average of these values along the first axis, it is possible to obtain an n-dimensional array representing the "sense" of the overview matrix. The obtained features, along with the Embedding for the User, is finally fed to a shorter version of the DeepDenseNet proposed in Section 3.2.2, which terminates in a final layer with a ReLu activation function. The performances are definitely acceptable, although slightly worse than those obtained with Collaborative Filtering techniques: MAE of 0.719 and RMSE of 0.969.

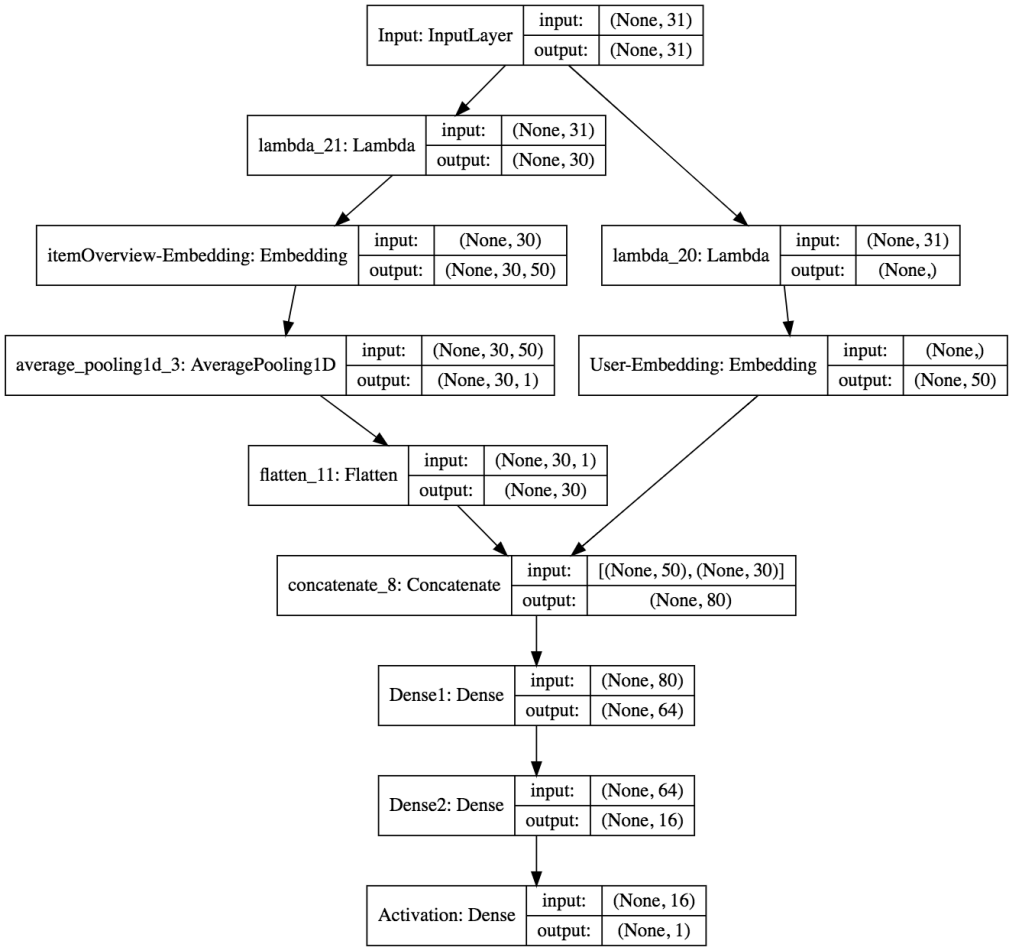

Figure 3.10: Proposed Architecture for CBF-Average

AMAR

Another proposed architecture for Content-Based Filtering is AMAR, also known as **Ask Me Any Rating**[35]. This deep neural network exploits Recurrent Neural Networks (RNNs) to jointly learn a representation for user preferences and items to be recommended. The architecture is illustrated in Figure 3.12. Embedding of the

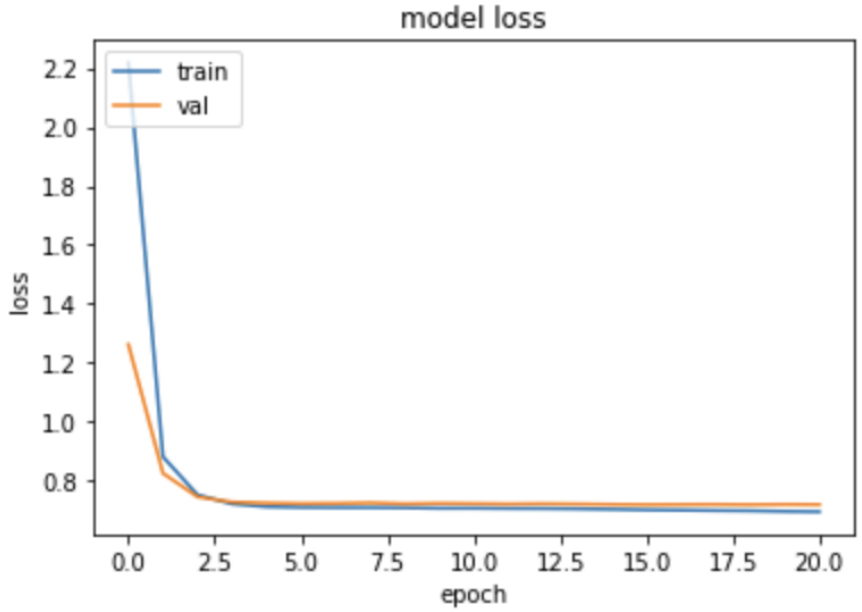

Figure 3.11: CBF-Average Loss plot

words for the item description is fed through an LSTM network which generates a user-defined latent representation for each of them. Once the word embedding is obtained by the LSTM, a mean pooling layer averages the latent representations to create an *item embedding*. The original implementation by Suglia and his team includes a Logistic Regression output layer in order to predict whether an item can be recommended to a user or not. In the Keras implementation presented in this work, this final layer has been changed to a ReLu in order to solve the recommendation problem as defined in Section 2.1. Although this Neural Network takes longer to train, around 5 minutes (25 seconds per epoch, Figure 3.13) with respect to the 2 minutes of CBF-Average (6 second per epoch, Figure 3.11) proposed in Section 3.3.1, the evaluation metrics are slightly better than those of CBF-A, scoring a MAE of 0.714 and a RMSE of 0.975.

This work proposes an additional variation to AMAR as proposed in [35]. Given

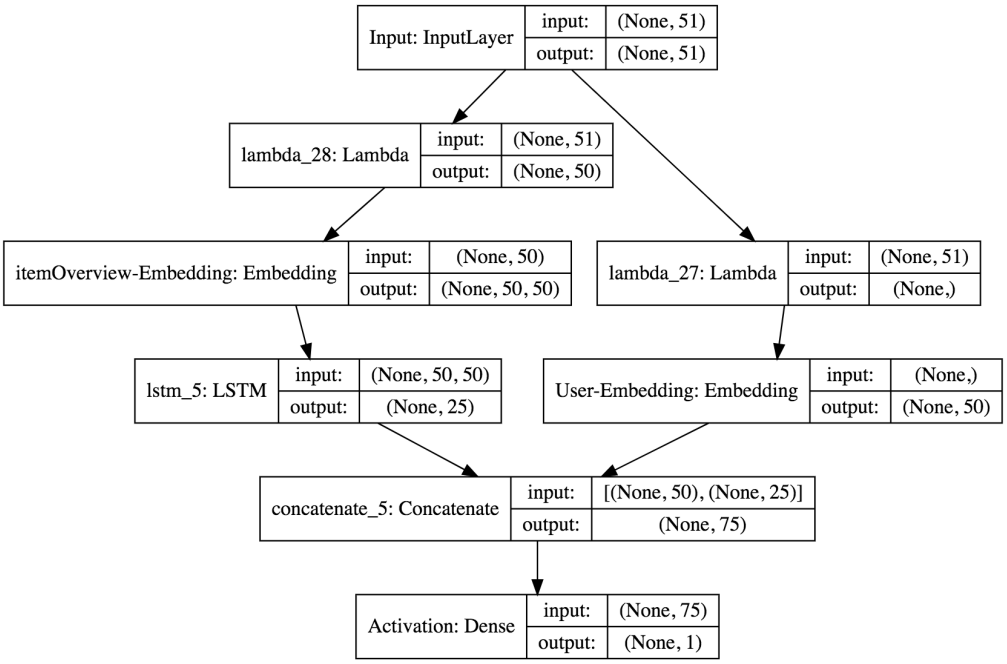

Figure 3.12: Ask Me Any Rating - Keras implementation

the recent success on certain datasets with GRU units[21][39], an adaptation of AMAR with GRU as Recurrent Neural Network has been tested. Although the GRU variant performs much better with respect to the LSTM one, with MAE of 0.683 and RMSE of 0.904, the GRU AMAR takes about 5 times as long as the LSTM AMAR, taking about 30 minutes to reach convergence in 83 epochs (22 seconds per epoch, Figure 3.14).

3.4 Hybrid Recommendation

Summarizing the previously presented results, it is clear that, for the problem at hand, Collaborative Filtering recommendation proves to be generally more effective, leading to relatively lower error. However, with new items/users (cold-start

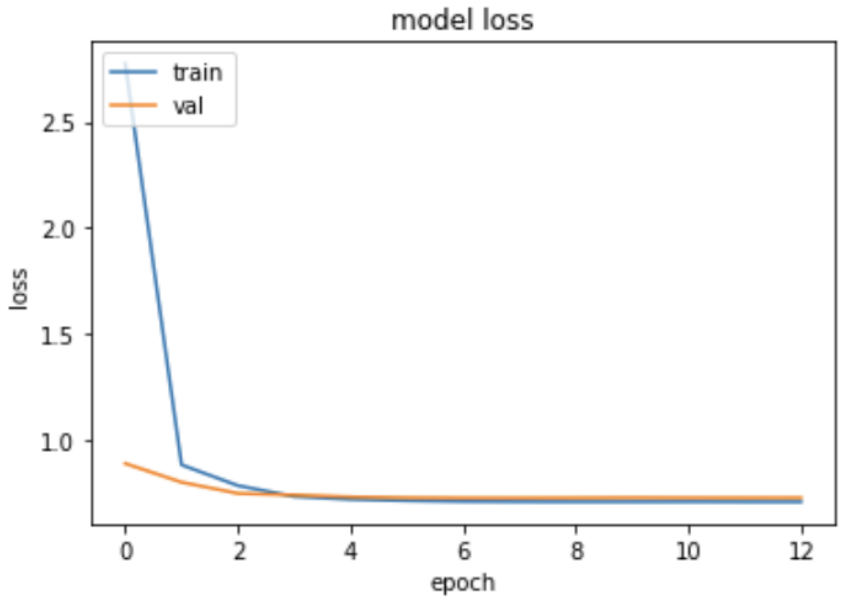

Figure 3.13: Ask Me Any Rating Loss plot

and early-rater problems), while Content-based approaches can provide some results, Collaborative Filtering algorithms fail completely, generating an Exception for Out-of-Vector embedding. Researchers have shown that **hybrid recommendation systems** can solve this problem, by combining the performances of collaborative filtering and and the adaptability of content-based recommendation[1].

Both **DeepWideNet**, presented in Section 3.2.3, and **AMAR**, presented in Section 3.3.1, are highly modular thanks to their Keras implementation, and therefore can be easily extended to accept a variable number of encoded parameters, either categorical or continuous, and use this information to enrich the prediction generated via standard collaborative-filtering methods, as well as generate one where the CF method would normally fail. **One-hot encoding** has been chosen over label encoding or others for encoding categorical features because of the *"multi-class nature"* of some of the features: movies can belong to multiple genres for examples, such as action/thriller or romantic/comedy movies. This characteristic can surely

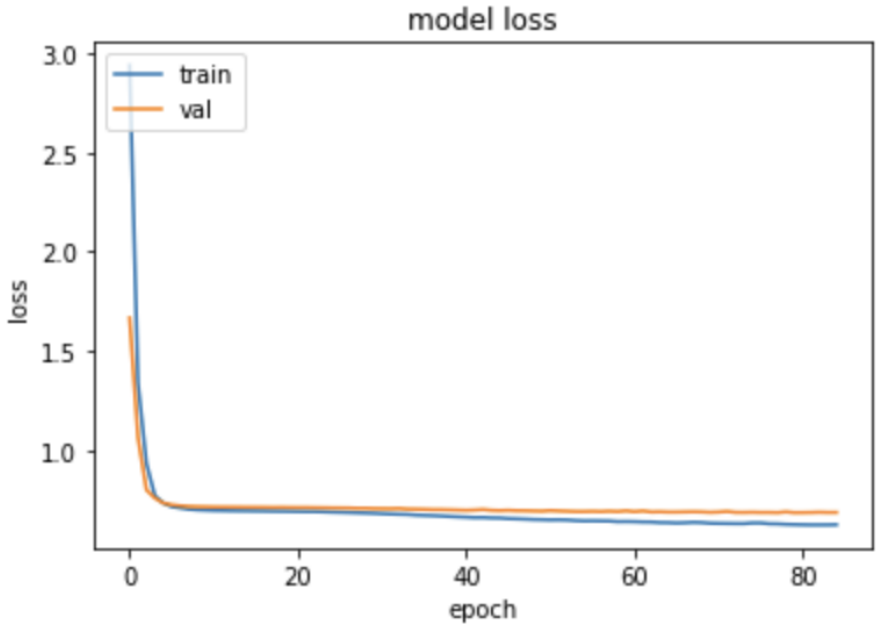

Figure 3.14: Ask Me Any Rating Loss plot - GRU Variation

be expressed with a new value in the encoding vocabulary, but that would cause losing the information about belonging to two (or more) already existing genres. On the negative side, a catalogue comprising hundreds of genres or categories would imply some sparsity in the input to the network, which is anyhow more manageable and preferable to the loss of information.

In particular, *two extensions* of the previously presented Deep neural networks have been tested: the first one is an extension of DeepWideNet which includes categorical and textual information about the movies, extracted through the GloVe embeddings freely available online (Section 3.4.1), while the other is a similar extension of AMAR, based on the LSTM variant (Section 3.4.2).

3.4.1 DeepWideNet - NLP Extension

DeepWideNet, as proposed in Section 3.2.3, is a highly modular and scalable deep neural network. It allows a variable number of input, each of fixed size, which are fed to the Embedding layer of the network. Each feature generates its own embedding, resulting in a nXm matrix of feature embedding, where m is the embedding size chosen when defining the model, and n is the actual size of the input. For exaple, with 19 different movie genres and each film belonging up to 5 genres, with embedding size set to 50, the output of the embedding layer is a matrix $[5x50]$. Then, an average along the smaller axis is computed, obtaining the "*average genre of the movie*". This feature is then fed to a concatenation layer, together with other features processed in the same way. Long textual data is processed in a similar fashion, extracting the GloVe embeddings and using them as weights in the Keras Embedding layer. Finally, the concatenated tensor is fed to the deep and wide components of the network, and a ReLu activation function is finally used to predict the rating.

This extended version of DeepWideNet is illustrated in Figure 3.15. This deep neural network generates the best results of the networks analysed up to this moment, scoring a MAE of 0.677 and a RMSE of 0.889 . The network converges slightly slower than the standard DeepWideNet, however this behaviour is accepted due to the increased number of features and parameters of the network.

3.4.2 AMAR Hybrid Recommender System

The architecture for AMAR proposed in Section 3.3.1 and implemented with the Keras framework can be easily extended to process multiple encoded features. As illustrated in Figure 3.17, the HRS of the AMAR architecture has been modularized and extended to include other features provided in the MovieLens dataset, namely the movie title, the movie genres, and the movie overview. In order to keep the information obtained with the Collaborative Filtering approach, the *movieId* feature has been also included. The same approach of averaging the result of the Embedding

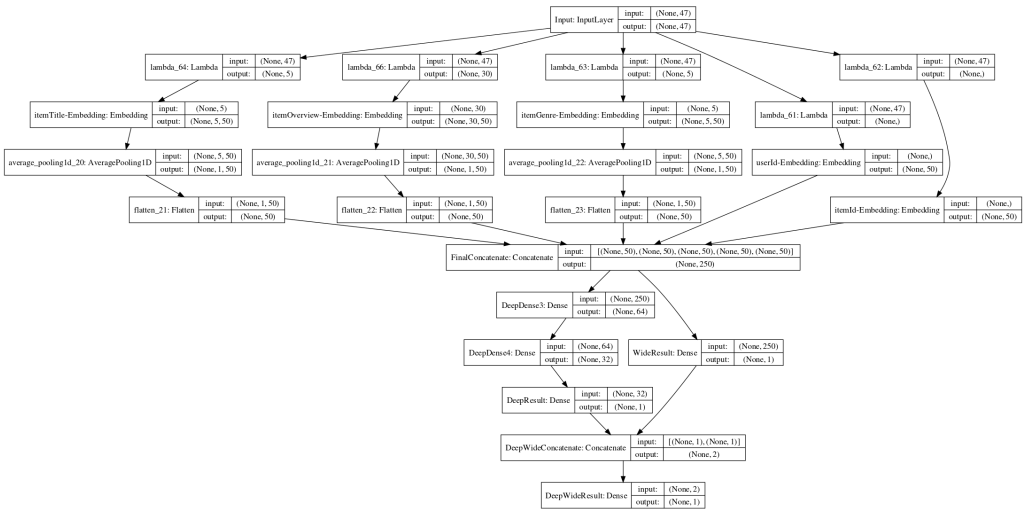

Figure 3.15: DeepWideNet - NLP Extension

layer on the same feature has been applied. LSTM is the chosen Recurrent Neural Network to process the the Embedded movie summary. This architecture of Deep Neural Network performs very similarly with respect to the extended DeepWideNet presented in the previous section, still outperforming the respective CF and CBF counterparts: AMAR HRS scores a MAE of 0.68 and a RMSE of 0.9 (Figure 3.18).

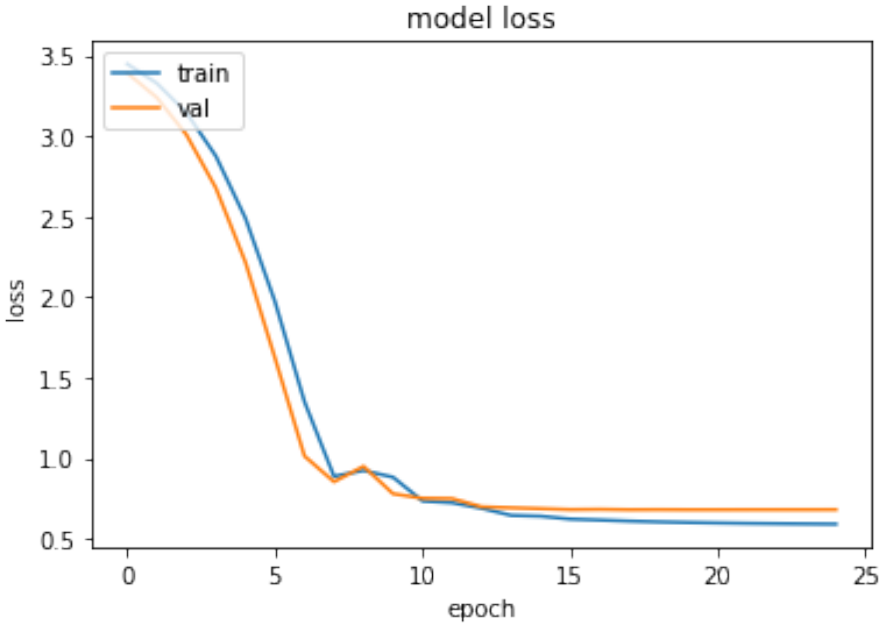

Figure 3.16: DeepWideNet - NLP Extension - Plot Loss

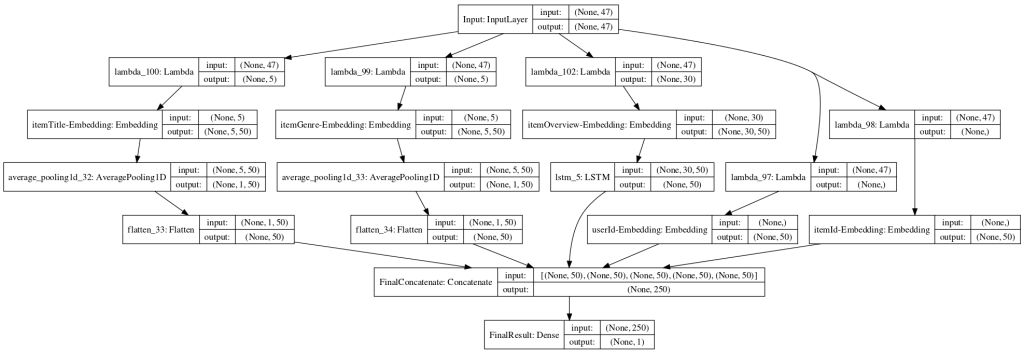

Figure 3.17: AMAR HRS

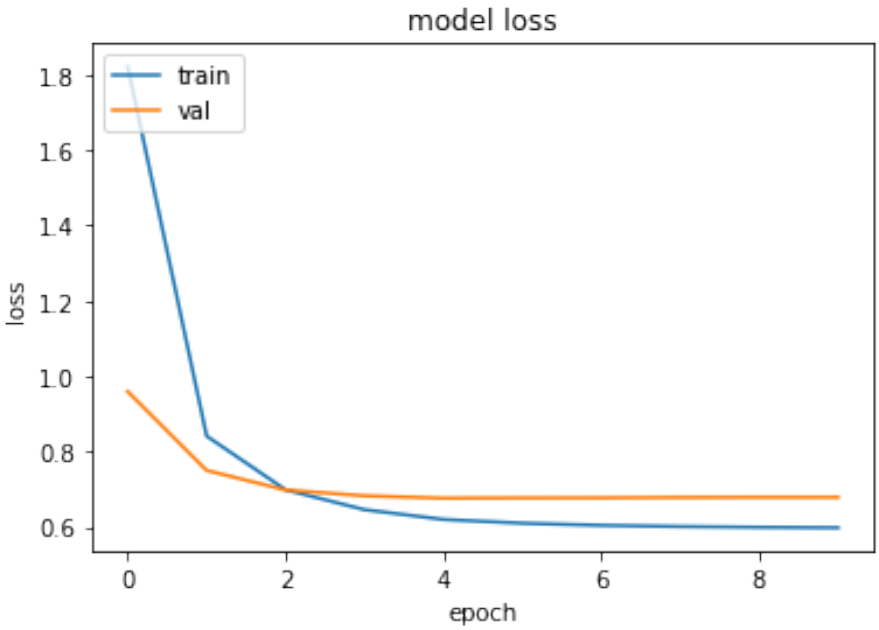

Figure 3.18: AMAR HRS - Plot Loss

Chapter 4

Results

This chapter analyzes the results obtained with the implementations discussed in Chapter 3 on multiple datasets. Along with the previously presented *MovieLens Latest Small*, several variations of the same MovieLens dataset have been used, including *ML-1M* (1 million reviews), *ML-10M* (10 million reviews), *ML-Latest* (27 million reviews), as well as another classical dataset for recommendation systems *BeerAdvocate* (approximaley 1 million reviews). Every datasset has been split randomly for *training*, *validation* and *testing*, assigning 60% of the data to the training dataset, 20% to the validation dataset, and 20% to the testing dataset. The results presented are an average on multiple runs with different random seeds. The datasets do not have correlated entries, therefore shuffling before splitting is a good practice to ensure randomicity of the data during training.

Due to the lack of mapping file between $movieId$ and $imdbId$ for the datasets *ML-1M* and *ML-10M*, the results provided are obtained only by extracting information from the movie title and movie genres. Furthermore, for the *BeerAdvocate* dataset, no textual review was provided, therefore the features used are mostly scalar and categorical. All of the used datasets are freely available online.

As explained in section 3.1.2, the chosen measures as MAE and RMSE. These are industry standard metrics to measure performance of recommender systems

which solve the regression problem, with lower values representing a lower error in predicting the actual rating of the tuple $< user, movie >$. For example, a MAE of 0.5 means that, on average, the predicted score will deviate from the real score of ± 0.5.

Dataset	DeepMF	Non-Negative DeepMF	DeepDenseNet	Non-Negative DeepDenseNet
Movielens Latest Small	0.726 0.951	0.69 0.893	0.684 0.911	0.686 0.915
ML-1M	0.702 0.898	0.739 0.949	0.724 0.938	0.721 0.922
ML-10M	0.637 0.833	0.634 0.833	0.664 0.875	0.664 0.879
ML-Latest	0.626 0.832	0.626 0.835	0.658 0.883	0.658 0.887
BeerAdv	0.53 0.746	0.472 0.632	0.458 0.63	0.454 0.626

Table 4.1: MAE and RMSE Results - Part 1

As shown by the results in Tables 4.1 and 4.2, **hybrid recommender systems** prove to perform better than their collaborative filtering and content-based filtering counterparts for most of the datasets analysed in this work. This demonstrates how important it is to try and extract as much knowledge as possible from the given data, leading to better results and/or shorter convergence times.

In particular, both *DeepWideNet + NLP* and *AMAR + NLP* show comparable results across all the analysed datasets. Both algorithms have been trained and tested five times against all five datasets. Although never stable on a given value, MAE and RMSE showed a variance of less than $\pm 3\%$, with the average value shown in the Tables. Furthermore, the error values become smaller and smaller with bigger datasets. This is to be expected with a Deep Learning approach which, at the expenses of greater computational time, it is capable of better understanding the

Dataset	DeepWideNet	CBF-A	AMAR	DeepWideNet + NLP	AMAR + NLP
Movielens	0.684	0.719	0.714	**0.677**	0.68
Latest Small	0.908	0.969	0.975	**0.889**	0.9
ML-1M	0.697	0.732	0.727	**0.689**	0.693
	0.898	0.958	0.964	**0.879**	0.89
ML-10M	0.627	0.659	0.654	0.62	**0.618**
	0.83	0.886	0.891	0.812	**0.81**
ML-Latest	0.620	0.651	0.647	**0.613**	0.620
	0.832	0.887	0.893	**0.814**	0.825
BeerAdv	0.474	0.498	0.492	0.45	**0.443**
	0.633	0.675	0.679	0.622	**0.617**

Table 4.2: MAE and RMSE Results - Part 2

patterns of the dataset. The risk of incurring into *overfitting* is solved by implementing an *early stopping* policy.

Chapter 5

Conclusions and Future Work

5.1 Summary

The science of Recommender Systems is one which has been studied for many years, engaging researchers all over the world. Those systems have been adding value for many companies, allowing them to predict the future preferences of users based on their previous interactions or their tastes.

In the literature, there are many different techniques to approach this problem, such as neighbourhood based, machine-learning based and matrix-factorization based methods.

In this work, multiple deep neural network architectures have been proposed, trying to solve the recommendation problem implementing the different techniques presented in Section 2.2. All of the presented architectures have been developed with the Keras framework for Deep Learning, with a Tensorflow backend, and evaluated on well-known datasets such as **MovieLens** with **MAE** and **RMSE**, two of the most commonly used evaluation metrics for regression problems. For the **collaborative-filtering (CF)** approach, the first proposed architecture is a more traditional implementation of the matrix-factorization algorithm via Deep Learning, called **DeepMF**. DeepMF aims to learn feature interactions of higher order

by computing the inner product of the embedded $< userId, itemId >$ pair. This method has proven to be generally very effective as well as fast to predict the rating of an item, according to the history of the interactions with other users. A second approach proposed is **DeepDenseNet**, which exploits deep feed-forward neural network to learn high order non-linear interaction from the concatenation of the embedded $< userId, itemId >$ pair. With smaller datasets, DeepDenseNet achieves better results comparing to DeepMF. Finally, the last CF algorithm proposed is **DeepWideNet**, which combines the other two previously discussed algorithms, trying to use both level of extracted information from the data. As expected, the DeepWideNet implementation proves to be more effective comparing to the respective single components, achieving better results for both evaluation metrics. Another way to solve the recommendation problem is to adopt **Content-Based Filtering** algorithms. Two deep neural networks have been proposed to study this approach. The first one is **CBF-Average**, which uses a GloVe embedding based representation of information extracted from textual data such as movie summary/overview. The *Average* component of the proposed architecture is a *Mean Pooling layer* which computes the average of the embedding of the words - producing something which is comparable to an embedding of the whole given text. The second proposed architecture is **AMAR**, or *Ask Me Any Rating*, an architecture proposed by [35] but adapted to solve a regression problem (the prediction of the rating) instead of a logistic regression one. AMAR exploits RNNs to learn a hidden state representation of the information extracted from textual data, then uses these extracted features to predict the rating for a specific user. Combining the strengths of the two is techniques, as well as overcoming the shortcomings of both, is the role of **Hybrid Recommender Systems**. It is possible to extend both DeepWideNet and AMAR by including in the inputs embedded information from other categorical and textual features, embedded and averaged just like with *CBF-Average*. These two variants prove to perform better than their collaborative filtering and content-based filtering counterparts for most of the datasets analysed in this work. Moreover, this approach solves the problem of new items or users without previous interaction

(cold-start problem), while keeping the information regarding previous interactions.

5.2 Future Works

This work mostly exploited NLP to extract additional features regarding both user and item in the used datasets. However, NLP-based techniques are not the only way to enrich existing information regarding user-item interaction. An interesting possible future research can be **Recommendation via Artwork**. Both *DeepWideNet* and *AMAR* can be easily extended to include features extracted from the artwork related to the product to be recommended. These features can be obtained through the second to last layer of a *VGG-16* architecture trained on the ImageNet dataset, a state-of-the-art deep neural network used worldwide for image classification and object detection, or through a purpose-built neural network. Although feature extraction and neighbourhood definition of a movie given its artwork has been briefly tested with the MovieLens Latest Small dataset, the lack of image data for other datasets and sub-optimal results for the neighbourhood of movies in the MovieLens dataset have led to quit further investigation of this approach in this work. In particular, this approach could potentially be very successful for products to be sold (such as the products from BeerAdvocate dataset).

Another interesting approach which has not been analyzed in this work due to partial lack of dataset are **Deep Semantic Similarity Model** or *DSSM* for short. DSSM is a Deep Neural Network (DNN) used to model semantic similarity between a pair of strings, but can be extended to any number of pairs of strings. Embedding extracted with the methods described in this work can be considered as strings, and therefore used as inputs for the DSSM. DSSM networks have proven to be very effective and fast, and could be an interesting point for further research on the topic of recommender systems.

List of Figures

List of Tables

Bibliography

[1] Adomavicius, G., and Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng. 17*, 6 (June 2005), 734–749.

[2] Alshammari, G., Kapetanakis, S., Alshammari, A., Polatidis, N., and Petridis, M. A hybrid feature combination method that improves recommendations. In *Computational Collective Intelligence* (2018), N. T. Nguyen, E. Pimenidis, Z. Khan, and B. Trawiński, Eds., Springer International Publishing.

[3] Avery, C., and Zeckhauser, R. Recommender systems for evaluating computer messages. *Commun. ACM 40*, 3 (Mar. 1997), 88–89.

[4] Barros, R. C., Basgalupp, M. P., de Carvalho, A. C., and Freitas, A. A. Towards the automatic design of decision tree induction algorithms. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation* (New York, NY, USA, 2011), GECCO '11, ACM, pp. 567–574.

[5] Bobadilla, J., Ortega, F., Hernando, A., and GutiéRrez, A. Recommender systems survey. *Know.-Based Syst. 46* (July 2013), 109–132.

[6] Burke, R. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction 12*, 4 (Nov. 2002), 331–370.

[7] Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., and Sartin, M. Combining content-based and collaborative filters in an online newspaper, 1999.

[8] Ericson, K., and Pallickara, S. On the performance of distributed clustering algorithms in file and streaming processing systems. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing* (2011).

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

[10] Guo, H., Tang, R., Ye, Y., Li, Z., and He, X. Deepfm: A factorization-machine based neural network for CTR prediction. *CoRR abs/1703.04247* (2017).

[11] Hand, D. J., Smyth, P., and Mannila, H. *Principles of Data Mining*. MIT Press, Cambridge, MA, USA, 2001.

[12] Harper, F. M., and Konstan, J. A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst. 5*, 4 (Dec. 2015), 19:1–19:19.

[13] He, R., and McAuley, J. VBPR: visual bayesian personalized ranking from implicit feedback. *CoRR abs/1510.01784* (2015).

[14] Hidasi, B., and Karatzoglou, A. Recurrent neural networks with top-k gains for session-based recommendations. *CoRR abs/1706.03847* (2017).

[15] Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. Session-based recommendations with recurrent neural networks. *CoRR abs/1511.06939* (2015).

[16] Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural Comput. 18*, 7 (July 2006), 1527–1554.

[17] Huang, Z., Chen, H., and Zeng, D. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Trans. Inf. Syst. 22*, 1 (Jan. 2004), 116–142.

[18] Ivarsson, J., and Lindgren, M. *Movie recommendations using matrix factorization*. PhD thesis, KTH, 2016.

[19] Jannach, D., Zanker, M., Felfernig, A., and Friedrich, G. *Recommender Systems: an Introduction*. Cambridge University Press, 2010.

[20] Lee, D. D., and Seung, H. S. Learning the parts of objects by nonnegative

matrix factorization. *Nature 401* (1999), 788–791.

[21] Liu, D., and Singh, G. A recurrent neural network based recommendation system.

[22] Lops, P., de Gemmis, M., and Semeraro, G. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Springer, 2011, pp. 73–105.

[23] Luo, X., Zhou, M., Xia, Y., and Zhu, Q. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics 10*, 2 (May 2014), 1273–1284.

[24] Melville, P., and Sindhwani, V. Recommender systems. In *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Springer, 2010, pp. 829–838.

[25] Mhaskar, H. N., and Micchelli, C. A. How to choose an activation function. In *Proceedings of the 6th International Conference on Neural Information Processing Systems* (San Francisco, CA, USA, 1993), NIPS'93, Morgan Kaufmann Publishers Inc.

[26] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (USA, 2013), NIPS'13, Curran Associates Inc., pp. 3111–3119.

[27] Mitchell, T. M. *Machine Learning*, 1 ed. McGraw-Hill, Inc., New York, NY, USA, 1997.

[28] Özkural, E. The foundations of deep learning with a path towards general intelligence. *CoRR abs/1806.08874* (2018).

[29] Portugal, I., Alencar, P. S. C., and Cowan, D. D. The use of machine learning algorithms in recommender systems: A systematic review. *CoRR abs/1511.05263* (2015).

[30] Ramos, J. Using tf-idf to determine word relevance in document queries, 2003.

[31] Rendle, S. Factorization machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining* (Washington, DC, USA, 2010), ICDM '10, IEEE Computer Society, pp. 995–1000.

[32] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web* (New York, NY, USA, 2001), WWW '01, ACM, pp. 285–295.

[33] Sedhain, S., Menon, A. K., Sanner, S., and Xie, L. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web* (New York, NY, USA, 2015), WWW '15 Companion, ACM, pp. 111–112.

[34] Su, X., and Khoshgoftaar, T. M. A survey of collaborative filtering techniques. *Adv. in Artif. Intell. 2009* (Jan. 2009), 4:2–4:2.

[35] Suglia, A., Greco, C., Musto, C., de Gemmis, M., Lops, P., and Semeraro, G. A deep architecture for content-based recommendations exploiting recurrent neural networks. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization* (New York, NY, USA, 2017), UMAP '17, ACM, pp. 202–211.

[36] Takács, G., Pilászy, I., Németh, B., and Tikk, D. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res. 10* (June 2009), 623–656.

[37] Thiengburanathum, P., Cang, S., and Yu, H. A decision tree based recommendation system for tourists. In *2015 21st International Conference on Automation and Computing (ICAC)* (Sep. 2015), pp. 1–7.

[38] Toshihiro Kamishima, Shotaro Akaho, H. A., and Sakuma, J. Correcting popularity bias by enhancing recommendation neutrality.

[39] Tran, V., and Nguyen, L. Semantic refinement gru-based neural language generation for spoken dialogue systems. *CoRR abs/1706.00134* (2017).

[40] Vasile, F., Smirnova, E., and Conneau, A. Meta-prod2vec - product

embeddings using side-information for recommendation. *CoRR abs/1607.07326* (2016).

[41] Wang, X., Li, W., Cui, Y., and Mao, J. Click-through rate estimation for rare events in online advertising.

[42] Wu, Y., DuBois, C., Zheng, A. X., and Ester, M. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining* (New York, NY, USA, 2016), WSDM '16, ACM, pp. 153–162.

[43] Zeldes, Y., Theodorakis, S., Solodnik, E., Rotman, A., Chamiel, G., and Friedman, D. Deep density networks and uncertainty in recommender systems. *CoRR abs/1711.02487* (2017).

[44] Zhang, T., and Iyengar, V. S. Recommender systems using linear classifiers. *J. Mach. Learn. Res. 2* (Mar. 2002), 313–334.

[45] Zhang, Y., Jin, R., and Zhou, Z.-H. Understanding bag-of-words model: a statistical framework. *Int. J. Machine Learning Cybernetics 1* (2010), 43–52.