DEEP LEARNING BASED RECOMMENDATION SYSTEM DESIGN

by

Seçil Şen

B.S., Computer Engineering, Galatasaray University, 2013

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2021

ACKNOWLEDGEMENTS

First of all, I would like to express my special appreciation to my thesis advisor Professor S. Fikret Gürgen for always supporting me on my every high and low and always believing, encouraging, and tolerating me, and lots of thanks for introducing me the wonders of natural language processing and deep learning, which became the field that I want to specialize in on my career path. I would also like to thank my committee members Prof. Tunga Güngör and Prof. Tevfik Aytekin for kindly accepting to be in my thesis committee.

I would love to dedicate this thesis to the loves of my life, my daughter and my son, Elif Feryal Şen and Emre Ekin Şen, who gave me the reason not to give up, who I want to show that hard work pays off and dedication is the only thing that a person need for success. I find myself beyond lucky that I have them, I am so proud that I will always be a successful mom in their eyes.

I am beyond grateful to my mother, my father and my sister, Şafak Özkanoğlu, Aytekin Özkanoğlu and Gözde Yenidoğan who always believed me and supported me in every sense. Especially my mother, she is not only my mother but also my best friend, thank you for being a great mother, your dedication to everything that you do inspires me, I am who I am now thanks to you. I am so thankful to have my best friend, Mislina Gök, for always supporting me and believing me during my thesis study. Your goodwill and joy always inspires me and I thank God that our paths have crossed. Thanks to you, everyday you remind me the reasons to hold on to this study.

Last but not the least, Words cannot express how grateful I am to my beloved husband Zafer Şen who has always been supporting me, stood by me and been present for me whenever I needed him. I am so glad that we share our lives and we are raising our children together. I love you to the moon and back.

ABSTRACT

DEEP LEARNING BASED RECOMMENDATION SYSTEM DESIGN

Recommendation systems are software tools that seek to suggest relevant items regarding user preferences. Preference might be any; text to read, product to buy, or anything depending on the industry. Since it's vastly preferred by companies nowadays, it has been studied extensively. However, there are still open research areas based on performance improvements. In this thesis, a movie recommendation system is built using deep learning methods in order to see the effect of deep learning on the performance of recommendation systems. Moreover, this study incorporates natural language processing methods since movie description texts are the main metadata used by the system. To increase the quality of recommendation, a two-layered system is built and each layer is a deep neural network-based. The first layer acts as a complex embedding layer built on convolutional neural network architecture which takes movie descriptions as input, gives the predicted genres as output, and uses network parameters as embedding vectors of the second layer's input. This way, each movie has more complex representation on the second layer, making system's recommendations elaborated. The second layer is a recurrent neural network architecture, which takes embedding vectors of user's pre-interacted movies and user ratings to that movie as sequential input and outputs the next items to be recommended. Hence, the system can promote any movie even though it has never been interacted with a user before.

After having completed the implementation and testing the system's recommendation performance, we have shown that such architecture which fuses two DNNs for both feature extraction and recommendation purposes produces better results than the baseline method.

ÖZET

DERİN ÖĞRENME TABANLI ÖNERİ SİSTEMİ TASARIMI

Öneri sistemleri, kullanıcı tercihlerini baz alarak kullanıcının ilgilisini çekebilecek öneriler yapmayı amaçlayan yazılım araçlarıdır. Önerilecek öğe okuyabileceği bir makale olabilir, satın alabileceği bir ürün olabilir ya da izleyebileceği bir film olabilir; kullandığı platforma göre çeşitlilik gösterebilir. Günümüzde şirketler tarafından büyük ölçüde tercih edildiğinden, akademik olarak çeşitli çalışmalar yürütülmüş ve iyi sonuçlar alınmıştır. Derin öğrenme ağlarının öneri sistemlerinin performansını arttırdığı birçok çalışmada kanıtlanmıştır. Buna rağmen, hala performans geliştirmeleri devam etmekte; bu amaç için özellikle derin öğrenme sistemleri tercih edilmektedir. Bu çalışmada, film öneri sistemlerindeki performans sorunları, derin öğrenme yöntemleri kullanılarak ele alınmaktadır. Film açıklama metinleri bu sistemin kullandığı ana içerik olduğu için bu çalışma doğal dil işleme yöntemlerini de içermektedir. Öneri kalitesini arttırmak amacıyla iki katmanlı derin sinir ağı tabanlı bir sistem kurulmuştur. İlk katman evrişimsel sinir ağları tabanlıdır, amaç film açıklamalarından filme ait matematiksel vektörler çıkarmaktır. İkinci katman tekrarlayan yapay sinir ağı tabanlıdır; ilk katmanda çıkarılmış olan vektörleri kullanarak kullanıcıya film önermeyi amaçlamaktadır. Sistem geliştirmesi tamamlandıktan ve gerekli testler yapıldıktan sonra iki katmanlı derin sinir ağı tabanlı sistemlerin performansları makine öğrenmesi yöntemlerinden çok daha iyi sonuçlar verdiği görülmüştür. Dahası, bu çalışmanın sonuçlarına göre, öneri sistemlerinde örtük öznitelik çıkarımının öğe soğuk başlatma sorununun üstesinden gelebileceği çıkarılabilir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . iv
ÖZET . . . v
LIST OF FIGURES . . . viii
LIST OF TABLES . . . ix
LIST OF SYMBOLS . . . x
LIST OF ACRONYMS/ABBREVIATIONS . . . xi
1. INTRODUCTION . . . 1
2. BACKGROUND . . . 4
2.1. Recommendation Systems . . . 4
2.1.1. Collaborative filtering methods . . . 5
2.1.2. Content-based methods . . . 5
2.1.3. Hybrid Methods . . . 6
2.2. Deep Learning . . . 6
2.2.1. Deep Feedforward Neural Networks . . . 6
2.2.2. Convolutional Neural Networks . . . 7
2.2.3. Recurrent Neural Networks - Long Short Term Memory . . . 9
2.3. Word Embeddings . . . 11
2.3.1. Word2Vec . . . 11
2.3.2. FastText . . . 12
3. RELATED WORK . . . 13
3.1. Deep Learning Based Multi-label Text Classification . . . 13
3.2. Deep Learning Based Approaches for Building Recommendation Systems 16
4. MODEL AND IMPLEMENTATION . . . 23
4.1. Movie Embedding Extraction Module . . . 24
4.1.1. Binary Cross-Entropy Loss . . . 25
4.1.2. Cosine Similarity Loss . . . 25
4.1.3. Focal Loss . . . 26

4.2. Next Movie Recommendation Module 28
5. EXPERIMENTS AND RESULTS 31
5.1. Dataset 31
5.2. Evaluation Metrics 31
5.2.1. Accuracy 32
5.2.2. Precision 32
5.2.3. Recall 33
5.2.4. F-score 33
5.2.5. Hit Ratio (HR@k) 33
5.3. Baseline Method - NeuMF 33
5.4. Results 34
5.4.1. Movie Embedding Extraction Module - CNN 34
5.4.1.1. Loss Function Comparison 35
5.4.1.2. Word2Vec vs. FastText 36
5.4.2. Next Movie Recommendation Module - RNN 38
6. CONCLUSION AND FUTURE WORK 41
REFERENCES 42

LIST OF FIGURES

Figure 2.1. Summary of Recommendation Systems. 4

Figure 2.2. Two Examples of Deep Feedforward Networks. 7

Figure 2.3. CNN architecture on sentence classification task. 9

Figure 2.4. LSTM and RNN units. 10

Figure 4.1. Our architecture. 23

Figure 4.2. Our CNN architecture. 27

Figure 4.3. Our RNN architecture. 28

Figure 4.4. Input and target list of RNN. 29

Figure 5.1. NeuMF Architecture. 34

Figure 5.2. Precision changes of loss functions per epoch. 37

Figure 5.3. Recall changes of loss functions per epoch. 37

Figure 5.4. F1 changes of loss functions per epoch. 38

Figure 5.5. Comparison of HR@10 changes per epoch. 39

Figure 5.6. Comparison of HR@k where K ranges from 1 to 10. 39

LIST OF TABLES

Table 5.1. Table of movie numbers for each genre in the dataset(by 2019). . . 32

Table 5.2. Parameter search space for CNN. 35

Table 5.3. Best parameter settings for loss functions. 35

Table 5.4. Comparison of loss functions. 35

LIST OF SYMBOLS

e	Exponential
α	Blending parameter *or* scale
$\beta_t(i)$	Backward variable
Θ	Parameter set
σ	Sigmoid activation function
θ	Tanh activation function

LIST OF ACRONYMS/ABBREVIATIONS

1D	One Dimensional
2D	Two Dimensional
CNN	Convolutional Neural Network
IMDb	International Movie Database
LSTM	Long Short Term Memory
NLP	Natural Language Processing
RNN	Recurrent Neural Network
ReLU	Rectified Linear Unit

1. INTRODUCTION

The information technology era forces companies to be available on the internet for income-related purposes. Moreover, if a company desires to not only exist but also be favored in the market, it has to exploit its benefits at its best. Since most companies are aware of that fact, the internet is currently financed by various forms of online advertising. Online shopping is a growing trend as well because times change, time is valuable and online purchasing is the fastest and most effective way. For those reasons, the major part of the economy nowadays relies on online shopping. Companies want to guide their customers to buy more and more. That desire emerges recommendation systems. Recommendation systems, as the name implies, recommend relevant items to users. Item might be any: product to sale, posts to display on a social network news feed, movies to watch, articles to read. During the process of finding the relevant item, a recommendation system relies on the association between a user and an item by predicting the probability of some action like user buying a product, clicking an article. The early work of finding the relevant match covers traditional machine learning methods like finding the similarity between the pattern of values of the target variable for different users or different items. Those methods require user and/or item information. The information type might change, let's say user's rating history which explicitly reflects user's preference or user profile information which implicitly reflects user's preference. The first information type causes limitations: when a new item or a new user is introduced, lack of rating history means that there is no way to calculate the degree of association or similarity between the new user and the existing item or the new item and the existing user. This is referred to as cold-start recommendation problem. Latter information type might alleviate this issue since it includes hints of preference of users and items. To make the content information understandable for recommendation systems, we need embeddings which are the numerical representation of content information. Moreover, the combination of both implicit and explicit features ends up with more robust systems.

With the advancement of deep learning, researchers attempt to benefit from the efficiency of deep learning architectures for alleviating the cold start problem. As stated before, content-based recommendation systems can alleviate the cold start problem, however, the process of producing embeddings for content information is challenging for traditional methods. At this point, researchers utilize deep learning architectures for embedding extraction. Content features are mostly text data, hence it requires natural language processing methods as well. With the combination of natural language processing methods and deep learning methods, state-of-the-art studies can extract rich feature vectors from content information to be further used in recommendation tasks.

After neural networks extract embeddings from content information, recommendation is performed usually by using traditional machine learning methods that are based on similarity measures. However, user's preferences might change over time and such systems are dynamic in terms of item and user circulation. Traditional machine learning methods are not sufficient for processing such sequential data for making on-point recommendations. To the best of our knowledge, recurrent neural networks are proven to be very effective for such problems since they are designed to benefit from sequential data.

In this research, we build a movie recommendation system; we exploit movie description text in order to extract movie embeddings and incorporating those embeddings and movie ratings, recommendations are produced. It might sound easy, however; the engine consists of two different architectures of neural networks which extract movie embeddings and recommend movies using complex movie representation vector regarding user's sessions, respectively. Moreover, the aim is to alleviate the item cold-start problem since movies can have representations even though they haven't been rated before. The nature of the study is highly complex and multi-disciplinary because it both involves neural networks and NLP. Furthermore, the work is conducted regarding performance issues.

This thesis has the following two main contributions:

(i) The movie embeddings are successfully extracted using CNN with the most suitable parameter setting, This is done by training the system while matching movie genres with movie descriptions.

(ii) The movie embeddings and movie ratings are concatenated in one single vector and using movie representations and user's sequential information, recommendations are produced through RNN.

2. BACKGROUND

2.1. Recommendation Systems

Recommendation systems are specially designed algorithms that seek to match relevant items with users. To this end, they aim to predict the rating that a user would give to an item or preference of the user. Figure 2.1 classify the techniques studied in the literature for this domain.

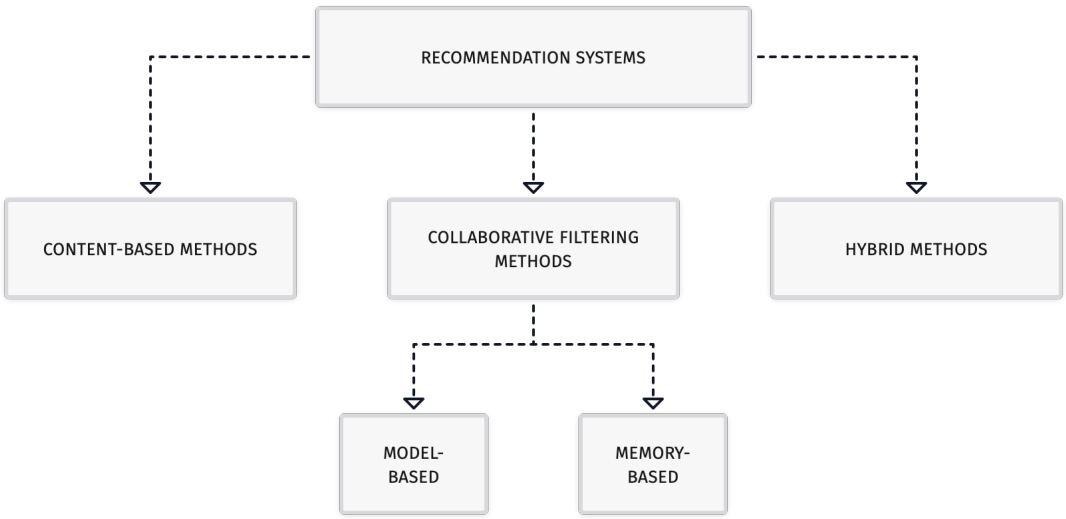

Figure 2.1. Summary of Recommendation Systems.

Each method in Figure 2.1 will be separately explained. However, we first need to have an understanding of the type of inputs that recommender systems process. Recommendation systems mainly utilize user feedback as input. Feedbacks are divided into two categories: implicit feedback, and explicit feedback. Explicit feedback is user's obvious interaction while implicit feedback is subtle as the name implies. Explicit feedback might cover user's rating, liking an item, or finishing watching a movie whereas implicit feedback might cover user's navigation history or clicking on an item. Implicit feedbacks are needed to be extracted from the data, thus requires feature engineering.

All the recommendation system design methods exploit either only one type or the union of those input types.

2.1.1. Collaborative filtering methods

Those methods solely profit from user's explicit feedback in order to produce recommendations. Feedbacks are stored in a so-called user-item feedback matrix. This method matches users with relevant interest by calculating similarity using the user-item interaction matrix and makes predictions based on the similarity of user or item profiles accordingly. Collaborative filtering methods are preferred in cases where past user-item interactions are sufficient for the recommendation and there are not enough implicit feedbacks to be extracted. This technique can be divided into two categories: memory-based and model-based. Model-based approaches employ a vector of user's previous interactions to compute a model for capturing new representations. The aim is to make recommendations immediately using the pre-computed model. Memory-based approaches are using directly the past interaction and recommendations are done using nearest neighbors information. Memory-based approaches are also categorized in themselves: user-based and item-based. User-based method bases on the user by finding similar users to make recommendations. The item-based method as well works with the same logic.

2.1.2. Content-based methods

This method benefits from both item metadata and user activity to produce recommendations; the focus is on the implicit feedback and complex representation of an item extracted from its descriptive attributes. Content-based systems attempt to extract both implicit information from users' history and dense item representation from item metadata and combine them to make the information ready for processing. However, this technique restricts systems in terms of prediction when there are a limited amount of content-based attributes. it doesn't work either for cases where a new user enters the system since he has no past activity.

2.1.3. Hybrid Methods

As its name implies, it mixes content-based and collaborative filtering approaches for the purpose of creating the most efficient recommendation system even though some information is missing for a user or an item.

2.2. Deep Learning

2.2.1. Deep Feedforward Neural Networks

Feedforward neural networks are the point of basis of all other neural network types. The intuition behind the idea is a biological neuron that takes the information from the outside world, evaluates it, and applies a filter to the result to decide whether to response.

To have a proper understanding, artificial neurons, often called nodes, should be explained since their union forms a deep neural network. A neuron receives input from other neurons or outside and computes an output. Each connection between neurons has a weight that presents the importance of its corresponding input. Weighted sum of all inputs of a neuron is calculated and then the neuron performs a calculation using a function f which is referred to as activation function. Activation functions are mostly non-linear to capture complex data representations.

As its name implies, a deep feedforward network, also referred to as a multilayer perceptron, makes the information flow through a cascade of neurons to produce the desired output. The goal is to approximate the function that calculates the output. To this end, the network defines a mapping $\mathbf{y} = f(x; \mathbf{\Theta})$ where $\mathbf{y}$ is the output, x is the input, and $\mathbf{\Theta}$ is the parameters to learn to make the best function approximation. This network does not have feedback connections that are further added to the recurrent neural networks.

Moreover, neural networks alleviate the most-encountered problem of traditional machine learning approaches: they mostly diverge when the target function is non-linear. This is a burden for the cases where the model needs to understand the interaction between any two inputs. It occurs in most real-life problems, so neural networks are always preferred since they are capable of solving non-linearities.

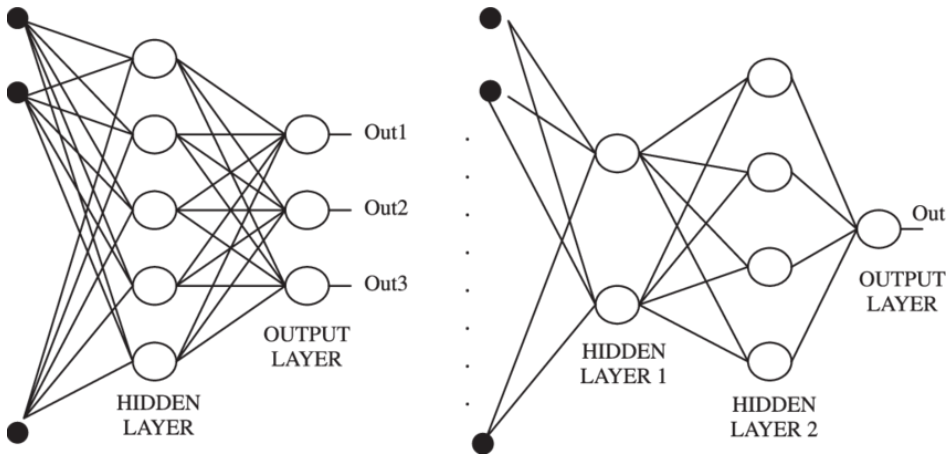

Figure 2.2. Two Examples of Deep Feedforward Networks [1].

Figure 2.2 depicts two different types of deep feedforward neural networks. In both networks, information flows through only one direction. Both networks have an input layer, hidden layer(s), and an output layer. The input layer only passes the input to the other layers, meaning that no calculation is performed. Hidden layers only take inputs from the previous layer, perform calculations and pass the information to the next layer. Hidden layers augment the network's complexity, thus increases the network's capability of capturing non-linearities. Output nodes also make calculations and pass the information to the outside world.

2.2.2. Convolutional Neural Networks

Convolutional neural network is a specialized kind of neural network that uses convolution operation in place of matrix calculation in at least one of their layers [2].

The most significant difference between traditional deep neural networks and convolutional neural networks is that every output unit interacts with every input unit in traditional approaches whereas convolutional neural networks have sparse interactions that are achieved by making the kernel smaller than the input. The most common usage of CNNs is in image recognition tasks and the idea emerges from the fact that traditional neural networks lack the capability of capturing spatial relationships of image pixels. CNN can detect meaningful features such as edges or shapes which only occupy few pixels of an image, thus shrinks the parameter size and improves the efficiency.

To comprehend the architecture, technical details should be explained. The fundamental operation of CNNs is convolution. Convolutions can be thought of as a sliding window function applied to the input matrix. CNN is composed of a bunch of convolution operations followed by a non-linear activation function application. Through convolution, the system can capture significant features. Another key aspect of CNNs is the pooling layer that is immediately applied after the convolution layer. The pooling layer is essential because it reduces the number of parameters and computational complexity by operating over the result of each input. It is considered that the pooling operation keeps the most salient information while removing the noise from the output. Moreover, it prepares the output for the classification layer by fixing the matrix size. Typically max pooling is used, either it can be applied to the complete matrix or it pools over a window. Pooling operation is followed by the fully connected layer, where the classification is performed.

Since this thesis uses CNN in the NLP task, we focus on CNN architecture applied for text data. Figure 2.3 shows convolution operations on a sentence classification model. In image classification tasks, the pixels can be fed into the CNN without any operation. However, in NLP task words themselves has no meaning for CNN since it expects numerical inputs. Hence words need a meaningful numerical representation which is further called word embeddings. We'll discuss word embeddings in Section 2.3.

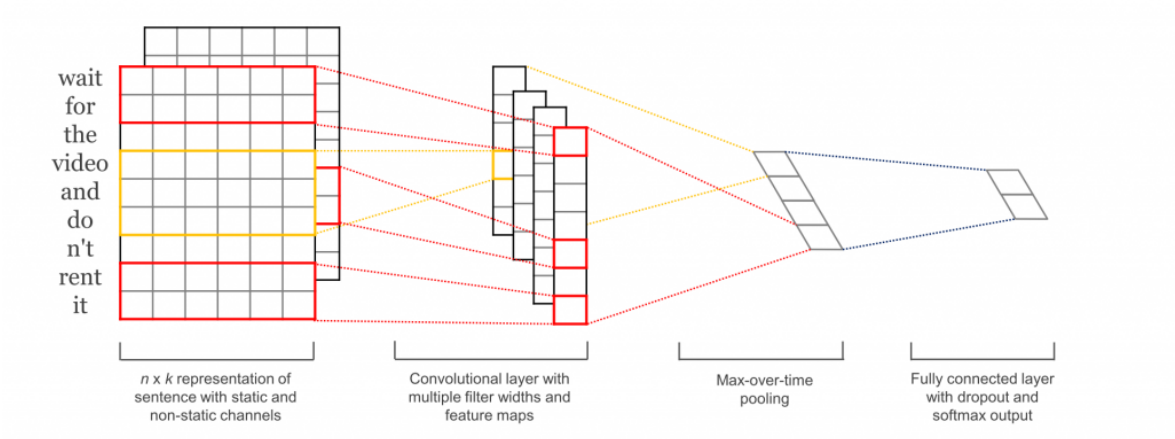

Figure 2.3. CNN architecture on sentence classification task [3].

2.2.3. Recurrent Neural Networks - Long Short Term Memory

RNNs are inspired by the human brain and the memory concept. The human brain records significant information and in some cases, when new significant information arrives, the old one might be overridden. The brain can accept inputs from our senses and our thoughts, calling it internal and external input respectively. It produces outputs in the form of actions and new thoughts, calling it internal and external outputs, respectively. RNNs work with the same logic.

As mentioned, RNNs exploits old information when producing new information, thus it makes use of sequential information. It alleviates the burden of traditional neural networks: all inputs should be independent of each other. For the tasks like predicting the next word in a sentence, the inputs must be dependent on each other. In theory, RNN can keep very old past information of long sequences but in practice, they can only look back few steps away. In RNN, the information recorded in the prior state gets embedded over and over due to the non-linearities, and that prior state is not the best usable state. The ground truth information is lost in the morphed information. LSTM alleviates this information vanishing problem of RNN.

Figure 2.4 shows both cells. We'll deep dive into LSTM which is our focus in this thesis.

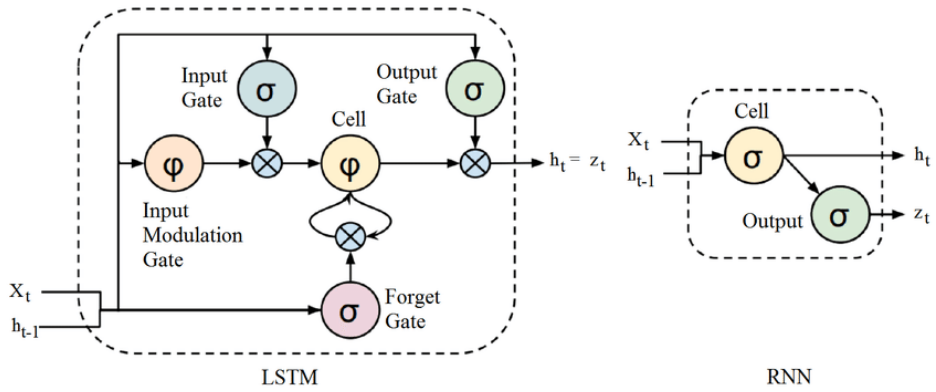

Figure 2.4. LSTM and RNN units [4].

LSTM is a derivation of a simple RNN cell; it solves the aforementioned problem by adding forget gate. Instead of updating the memory cell after each step, forget gate checks whether to keep the past information or update the memory cell with the new information of the current state. Calculations for each cell in an LSTM unit is listed below:

$$
\begin{aligned}
i_t &= \sigma(\mathrm{W}_{xi}x_t + \mathrm{W}_{hi}h_{t-1} + b_i) \\
f_t &= \sigma(\mathrm{W}_{xf}x_t + \mathrm{W}_{hf}h_{t-1} + b_f) \\
o_t &= \sigma(\mathrm{W}_{xo}x_t + \mathrm{W}_{ho}h_{t-1} + b_o) \\
g_t &= \phi(\mathrm{W}_{xc}x_t + \mathrm{W}_{hc}h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \phi(c_t)
\end{aligned}
\tag{2.1}
$$

where σ and ϕ are sigmoid and tanh functions, symbol $h_t \in R^N$ is the hidden state with N units, and $c_t \in R^N$ denotes the memory cell. $i_t \in R^N$, $f_t \in R^N$, $o_t \in R^N$, $g_t \in R^N$ stand for the input gate, forget gate, output gate and input modulation gate, respectively. $\odot$ is the element-vise multiplication [4]. Input modulation gate only ensures non-linearity. Hidden state is the long term memory of the cell and memory

cell is the short term memory of the cell which is shared with simple RNN cell.

2.3. Word Embeddings

Word embeddings stand for establishing meaning from words in a mathematical way. Since machine learning methods and neural networks expect a numerical input, words and associated words are clustered as vectors in multidimensional space for NLP tasks. Words were mapped to vectors in higher dimensional space, and the semantics of the words then learned when those words were labeled with similar meanings. In short, The objective is to make words with similar context have close spatial positions. So, for instance, when looking at movie reviews, the movies with positive sentiment have the dimensionality of their words ending up pointing a particular way, and those with negative sentiment pointing in a different direction. From this, the words in future sentences could have their direction established, and from this the sentiment inferred. It shows that both word meanings and the sequence in which they are found matter.

There are plenty of ways with proven performance to extract word embeddings. In this thesis, we will discuss Word2Vec [5] and FastText [6].

2.3.1. Word2Vec

Word2vec is the most popular embedding extraction technique which incorporates neural networks. There are two methods of Word2Vec: common bag of words (CBOW) and skip-gram. CBOW is a feedforward neural network that consists of the input layer, a hidden layer and the output layer, respectively. Hidden layer neurons don't apply an activation function; it only copies the weighted sum of inputs to the output layer. Non-linearity is ensured in the output layer by softmax calculations. The aim is to predict a target word using a single or multiple context input word(s). Assuming that network only accepts one-hot encoding word as input, it tries to predict the target word by comparing the output error with the one-hot encoding of the target word. Thus, the vector representation can be learned in the process of prediction. The network

can also take multiple words in the same context as input. Skip-gram model does the opposite; it uses the target word to predict the context. As stated, target word is fed into the neural network with one hidden layer, and the network outputs C probability distributions for N words, where C is the number of contexts to predict, and N is the total number of words in the dataset. It is said that CBOW is faster and produces better representations for frequent words whereas skip-gram works well with small datasets and is found to represent rare words better.

2.3.2. FastText

This architecture is also a feed-forward neural network with one hidden layer, but it makes use of character-level information while extracting word embeddings. Hence it achieves really good performance in the case of rare words. Each word is represented by a bag of character n-grams; meaning that the model slides over each word with a window of size n. The character n-gram method helps to preserve the meaning of shorter words which might be seen as n-grams in other words. It also allows the model to capture the meaning of suffixes and prefixes. Moreover, this model is appropriate when the words in the model don't belong to a particular language; language independence can be achieved by setting n to 0. In the training phase of FastText, the aim is to learn the weights of not only the words but also each of n-grams.

3. RELATED WORK

The thesis work covers multi-label text classification an recommendation systems that uses deep learning methods, including session-based recommendation. Thus, the related work section focuses on studies on deep learning-based multi-label text classification, CNN and MLP based approaches for building recommendation systems, and RNN-based recommenders for session-based systems. Therefore, studies are explained in this order for the sake of fine-grained understanding of the whole system.

3.1. Deep Learning Based Multi-label Text Classification

Gargiulo *et al.* [7] compare two different structures of deep neural network models in order to observe the effect of feature extraction on extreme multi-label text classification, where the number of labels are hundreds or thousands. Furthermore, they aim to analyze the relationship between hyperparameter tuning and dataset complexity. First network consists of word embeddings module and two consecutive dense layers. They use word embedding model [8] and priorly applied all NLP preprocessing steps. Second network is a classical one-dimensional CNN, also referred to as dense-CNN. As abbreviated, one-dimensional convolutional neural network is constructed with a max-pooling layer. They named the max-pooling layer as the feature extractor module. Titles and abstract parts of biomedical papers are exploited. For both networks, evaluation metrics are selected as precision, recall, and F1 scores. They prefer the balance of precision and recall so they take F1-score into account. CNN-dense outperforms the first network by means of the feature extractor module, which can capture meaning-specific features better rather than only relying on the grammar-specific features of the text. They also discover that as the label size heightens, setting the learning rate to higher values, say 0.1, increases the performance.

Garguilo *et al.* [9] extend the aforementioned paper, here they address showing the effect of different combinations of word embeddings on the same dataset. As the

previous study demonstrated, they prefer CNN architecture for the feature extraction module. However, the module in this work is more sophisticated; composed of 1D-CNN with a max-pooling layer followed by the other 1D-CNN with a max-pooling layer with different parameters. They cascaded two different CNNs with the purpose of capturing fine-grained features from the data. Their main contribution is to find the best combinations of existing word embedding extractors in the literature. However, the data labels are imprecise and the labels are semantically dependent, meaning that one label might be a subcategory or top category of another, which makes the label set hierarchically structured. They also indicate that the documents are labeled manually, therefore some labels might be missed. Those reasons make the dataset sparse and cause overfitting in the training phase. To solve the above-mentioned problems, they introduce a new methodology called Hierarchical Label Set Expansion, which aims to regularize the training set labels by including the complete hierarchy path for each label of the actual document. In detail, they find all the ancestors of a document's actual labels by using classic tree traversal algorithms and simply add them all to the label vector. They claim that this method overcomes the noisy training dataset problem. Henceforth, they stack five different embedding methods while training each before the latter is added. First, they only applied NLP preprocessing techniques and projects the words to the vector space using skip-gram word2vec. Then they used BioASQ [10] pre-trained word2vec embeddings. BioASQ is a large-scale biomedical semantic indexing challenge. They combine previous word embedding model with part of speech tags; which is a task of labeling each word in a sentence with its appropriate part of speech. They concatenate FastText model with the previous model, thus obtain the fourth word embedding model of the study. The final model incorporates the vectors from the dependency tree embedding model which structures sentences using a tree and utilizes it to encode them into the vector representation. Last model outperforms all on example-based precision, recall, and f1 scores. They demonstrate that regularizing the data labels avoids overfitting and different combinations of word embedding models explicitly incorporate grammatical and syntactic features and make classification more robust.

Wang *et al.* [11] develop CNN-based multi-label playlist classification using only text features of songs in a playlist. They assume that CNN can jointly learn song embedding vectors without using audio features. As a result, the system doesn't need any audio feature extraction module. They construct 2D and 1D CNNs consecutively for the purpose of information extraction of different granularity. First CNN takes the playlist which is represented by a N x D matrix, where N is the total number of the songs in the playlist and D is the embedding vector size of each song. They fetch the song dataset from a leading music service provider in Asia. Songs are labeled by language-related, genre-related, and event-related levels. They only consider 38 labels as the target. To prevent overfitting, they augmented the training dataset by subsampling playlists; they randomly take subsets of songs from playlists and create new playlists from them. The model is compared with SVM and kNN using precision, recall, and F1 scores. Data augmentation is applied for all techniques. The proposed model outperforms traditional models on all metrics. They demonstrate that applying cascaded 2D and 1D convolutions makes the classification more robust and effective.

Chen *et al.* [12] conducted this study with the motivation of achieving best results in the largest public text dataset in Chinese. To this end, they invented an architecture referred to as inception while trying several existing neural network architectures. Moreover, they make several experiments on accuracy, speed, and memory consumption of the models. The overall model has an embedding layer, feature extractor module, and classification layer. Embedding layer is a lookup table of word embeddings and classification layer is a two-layer fully connected network. Here, the feature extractor module matters since the aim is to find the best architecture for the feature extraction which leads to better classification results. They try four different architecuture named DeepText, Inception (proposed model), C-RNN, and FastText, consecutively. DeepText is CNN with more convolutional layers with fewer kernel sizes. The intuition is coming from the fact that more convolutional layers can extract n-gram features easily and can model short and long-span relations efficiently. TextInception is inspired by the GoogLeNet [13]; it has four independent convolutional layers with different kernel sizes; two layers of those four layers have two nested CNNs which brings the intuition of

the name "inception". Third architecture is called contextual recurrent neural network, abbreviated as C-RNN, uses two layers of bidirectional LSTMs. The last architecture is FastText While conducting the experiments, they exploit the top-k strategy; where k is equal to 5. In other words, they pick 5 most-likely labels ordered by the probability produced by the classifier. Precision, recall and F1 scores are chosen as comparison metrics. They also augment the data to prevent overfitting, and test each model both with solely original data and concatenation of augmented data and original data. For every case, C-RNN outperforms all models in every metric. Another interesting result that can be inferred from this study is that, data augmentation does not change classification performance of FastText since it doesn't take the order of the words or characters in a sentence into account.

3.2. Deep Learning Based Approaches for Building Recommendation Systems

Oramas *et al.* [14] seek to solve the cold start problem for music recommendation by combining text and audio features with user feedback. Their methodology is three-folded: first, they aim to learn artist embeddings from semantic text features and aggregate from user data. Then they extracted audio features and incorporate them with user feedbacks that forms track embeddings. Two different embeddings are fed into a multimodal neural network. The network is called multi-modal since its input sources have different natures. They used the million song dataset which has both precomputed audio features and song metadata. They compare two methods for extracting artist embedding: the first method is an entity linking model, which associates a textual fragment input with an entry from a given knowledge base is applied. They used Babelfly [15] for this purpose; its knowledge base has a direct mapping to DBPedia [16]. In brief, it gets biographies from Wikipedia and matches them with entities from DBPedia. Second method is CNN-based sentence classification which extracts artist embedding vectors from artist biographies. For the audio embedding extraction phase, they benefit from the relevant CNN architecture of audio feature extraction. In order to fuse two embeddings with different modalities, they normalize feature vectors

which are further fed to a simple multi-layer perceptron. They construct two different networks which differ from their normalization layer. First network connects artist embeddings and track embeddings into separate dense layers whose outputs merge in the output layer. The intuition behind this architecture is that two isolated dense layers can capture the non-linearities of each modality effectively. Second network normalized each embedding vector using L2 normalization, concatenates them into a single feature vector and finally gives one vector as input to the output layer. Experiments show that using two embeddings with different modalities improves the quality of recommendation. Furthermore, according to the comparisons with pure text or audio-based systems in terms of performance, the multi-modal approach indeed achieves better results.

Volkovs *et al.* [17] claim that applying dropout alleviates the cold start problem without the need for additional content-based objectives. Unlike most methods that incorporate different modalities to solve the cold-start problem, they focus on optimizing the architecture which can fit in every deep-learning based recommendation system model. they consider the cold start problem as the missing input problem, where the missing input is the preference information. Preference information might be user's likes/dislikes, ratings, and implicit interactions like clicks etc. That makes the system a typical collaborative filtering model which has a set of users and set of items and user-item feedback is represented by a matrix. The matrix content is user's preference information. Moreover, both users and items have content features projected to a vector space. The mentioned case targets all types for recommendation systems, thus the solution generalizes most problems in this domain. The system is built on two classical feed-forward neural network constructed correspondingly that incorporates preference and content information. Activation layer outputs of each network are then concatenated and passed through another network for obtaining the latent representation. Dropout comes into play in the training phase; they apply user dropout and input dropout consecutively to cover both item-cold start and user cold start cases for each mini sample batch. They further calculate relevance scores for the concatenated vector for each case until convergence. This way, the system doesn't need content information or preference information to calculate relevance scores. After the training phase

is completed, the model is fixed and passed through another network to extract latent representations. The system is built for some state-of-the-art network architectures for the experiments. For each system, addition of dropout technique improves the recall score for not only the cold start but also the warm start scenario.

In study [18], a very deep neural network is constructed aiming to alleviate the cold start problem. Authors claim that with collaborative filtering approaches, extracted embeddings eventually be linear, causing imprecise recommendations. With a very deep neural network, incorporating user and item side information with user feedbacks, they believe that recommendation performance increases. For those purposes, they construct an architecture that consists of three hidden layers where each layer is followed by LeakyReLU, dropout and batch normalization layers. The system takes concatenated user and item embeddings as input. Those embeddings also include user and item side information. Here, side information might be the title of the movies and tags for the movies and author name and year of publication for the books. As mentioned, they conducted their experiments on movie and book datasets. Due to the limitations of those datasets, they only tackle user cold-start problem. Moreover, they consider performance issues of such systems, and they have challenges like tuning the learning rate and controlling overfitting. Those problems can be solved by applying dropout, batch normalization, and early-stopping while searching the parameter space to find the best learning rate for this task. While simulating the user-cold start case in the training phase, users are randomly sampled into training and validation sets. Evaluation metrics are root mean squared error, mean absolute error and R-squared values. They benchmark 13 different methods including DNNRec. For both datasets, DNNRec slightly outperforms others in not only mentioned metrics but also CPU performance. Since computation is limited for most system, obtaining satisfying computational performance is valuable. The results also show that cold start case is improved with the contribution of side information of users and items. They show that a very deep neural network can capture non-linear latent features and alleviates that gap which exists in collaborative filtering systems.

Moreira *et al.* [19] instantiate a deep learning meta-architecture for news recommendation systems which is referred to as CHAMELEON. They incorporate user session data with news articles representations based on text and metadata. The architecture is composed of two base modules called the Article Content Representation and Next Article Recommendation, respectively. First module learns textual features and second module exploits those features to make session-based recommendations. Inputs of the first module are publisher information and the article itself. They used pre-trained word embeddings like Word2Vec and GloVe for constructing the embedding matrix. They construct a 1D-CNN followed by a fully connected layer whose output is further used as article representation in the next article recommendation module. For the training phase, they build one more fully connected layer to be able to make the classification. However, after the training phase is done, that layer is omitted. The second module takes the pre-trained content embedding of the last viewed article, which is extracted from the previous module, contextual properties of the article like popularity and recency, and the user context like time, location, and device. Those inputs are first concatenated in a fully connected network, then fed into the LSTM model. They seek to find the predicted next article embedding specific for the user with the proposed LSTM architecture. Only one article is recommended since news recommendation is more dynamic; thousands of articles are added and removed daily, which would require full training of the system if they constructed a network that produces more available items. That dynamism covers the cold start problem since the newly added article has no interaction with a user. Each newly added article is fed to the article content embedding layer for feature extraction purposes. Then, the next article recommendation module tries to maximize the similarity between predicted next article embedding and user personalized contextual article embedding utilizing cosine similarity and it is generalized as the relevance function. In the training phase, the model adjusts its parameters by maximizing the likelihood of the next article in the user session. For this purpose, they calculate the likelihood of the output of the softmax function. The training of the first module is performed with the same dataset since the aim is to learn the article representation. Next article recommendation evaluation uses mini-batches composed of user's sequences of clicked items. Rather than

using cross-validation, they devise a temporal offline evaluation method because cross-validation ignores temporal factors which are essential for the news recommendation. With this method, they train the model within active hours and evaluate it within the next active hour. During the benchmarks, all baseline models trained using the temporal offline evaluation method on Hit Rate and Mean Reciprocal Rank metrics. Hit Rate checks whether the clicked item is in the list of top k ranked items, where k is set by the system. Mean Reciprocal Rank rates items regarding the position of the item, meaning that it assigns higher scores at top k ranks. For both metrics, k is set to 5. The results of the benchmark show that studied method outperforms baseline methods in those metrics. Furthermore, temporal offline evaluation introduced here is promising for such recommendation systems which are extremely dynamic.

As of this subsection, we'll deep dive into RNN-based recommender system starting with Hidasi *et al.* [20]. They adapt recurrent neural networks to recommendation systems by introducing ranking loss, which makes the architecture more suitable to the recommendation task. As the title implies, recommendations are performed session-based; meaning that correspondent items should be recommended regarding user interactions within a time interval that the user is online. The system takes the first item that the user clicks in his session as the initial input and further clicks produce recommendations by exploiting all previous clicks of the session. The intuition of ranking loss comes from the fact that user always expects a list of most relevant recommendations that are referred to as top-ranked items. They prefer a customized version of GRU for the implementation of RNN. The input is the current event while the output is expected to be the next item of the event in the session. The input is a 1-of-N encoded vector and normalized before being fed into the network. The normalization is essential because it strengthens the memory effect of RNN. Consecutive RNN layers are followed by a feed-forward layer; giving the list of likelihoods of being the next item for each item in the session. They use the session-parallel mini-batches method, where the first events of a batch-size number of sessions create the first batch and other batches are formed in the same manner. While sampling the outputs, they stick to the most encountered problem of recommender systems: item dynamism which makes the system

unscalable in terms of performance. Thus they use popularity-based sampling based on the fact that the likelihood of an item being in other training samples is proportional to its popularity. For calculating the loss, they devise a ranking loss function named TOP1. They calculate the mean score of the relative rank approximation of the relevant item. However, this function pushes the negative samples to have higher scores. To prevent this, they regularize the function by pushing negative sample scores to be around zero with the addition of regularization term. In the experiments, two different datasets are used; click-stream of e-commerce sites and events of watching video from a video service platform. The evaluation is performed by calculating the rank of the item of the next event. The two evaluation metrics are recall@N and MRR@N (Mean Reciprocal Rank) where N=20. N is the size of the top list to be considered in the calculation. The rank is set to 0 if the rank is above 20. They compare three baseline methods with their system, but they focus on item-kNN. GRU-based method slightly outperforms other methods, resulting in around 20-30% accuracy gain. Moreover, if the computational resources didn't restrict the augmentation of the number of hidden units, the accuracy gain would be much more.

Suglia *et al.* [21] propose an RNN-based architecture that learns user and item embeddings separately for further usage on extracting the preferences of the user and predicting the next item to be recommended. The recommendation is performed by a logistic regression layer using extracted content embeddings. They claim that they prove the impact of LSTMs on textual data in content-based recommendation systems with this architecture. They named their architecture Ask Me Any Rating (AMAR). It predicts a list of probabilities that a user can like a specific item. The approach splits the embedding module into two different submodules that jointly learn user and item embeddings. However, item embeddings are learned by the LSTM network whereas user embeddings are from a simple lookup table. LSTM network is followed by a mean pooling layer, thus obtaining the item embeddings. User embeddings and item embeddings are further concatenated via the concatenation layer and the resulting vector is fed to the logistic regression layer for calculating the relevance of an item for a certain user. They indicate that the LSTM network is chosen for this task since they

perform well when sequences of input exist. They extended their approach by adding a new feature of the input: genre. Genre information is also projected to the vector space and the lookup table is passed to the mean pooling layer before concatenation operation. This more elaborated system is referred to as AMAR Extended. They use movies and books datasets for the experiments. The baseline methods include techniques like collaborative filtering, traditional content-based recommender systems, and embedding-based recommendation systems. Metrics for comparison are chosen as F1@5 and F1@10. Amongst baseline methods, AMAR Extended gives significantly better results in both datasets.

4. MODEL AND IMPLEMENTATION

In this chapter, we explain our overall architecture. A two-layered system is built and each layer is artificial neural network-based. First layer acts as a complex embedding layer built on CNN architecture which takes movie descriptions as input, gives the predicted genres as output, and uses network parameters as embedding vectors of the second layer's input. This way, each movie has a more complex representation that can further be fed into the recommendation layer, making system's recommendations more elaborated and user experience and trust of the recommendation system is largely improved. The second layer, the recommendation layer is a recurrent neural network, which takes basically embedding vectors of user's pre-liked movies and movie IMDB ratings as sequential inputs and outputs a list of items which will further to be ranked for recommendation. The system architecture is depicted in Figure 4.1.

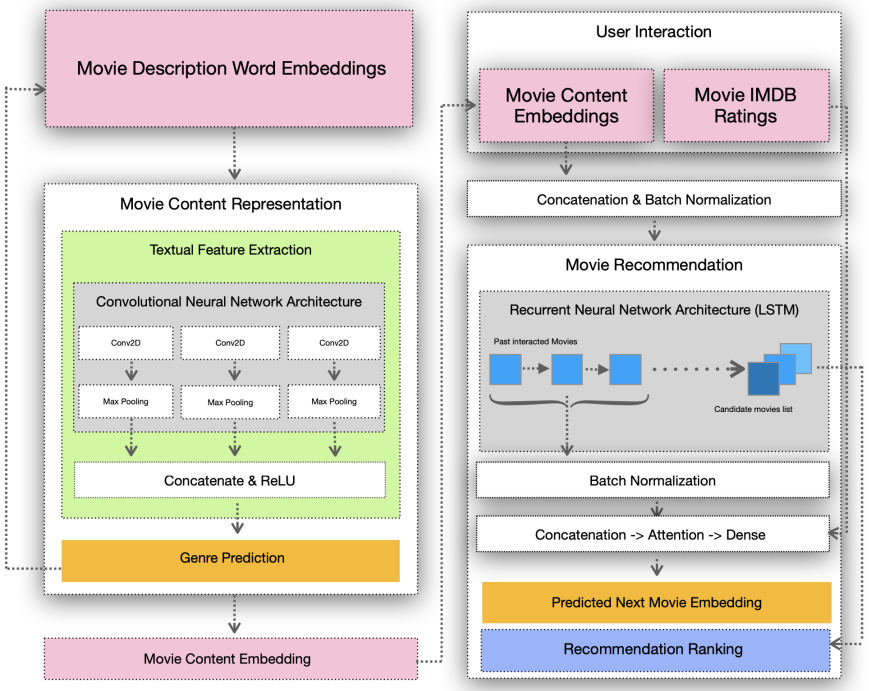

Figure 4.1. Our architecture.

The first module can be referred to as the movie embedding extraction module while the latter can be called the next movie recommendation module. We'll tackle two modules separately in the following sections.

4.1. Movie Embedding Extraction Module

The contribution here is to extract complex embeddings of movies by exploiting movie description and movie genres. This way, each movie would be projected to a multi-dimensional space dependent on the genre that they belong to. It is effective since the one-hot vector representation of a movie whose length equals the total number of genres in the dataset doesn't reflect any contextual features or dependencies. The only input of this module is a movie description text. In other words, those texts are textual movie trailers. Before being fed into the neural network, all required NLP pre-processing is applied to the movie descriptions (tokenization, removing stop words and punctuation, etc). Then each word of a movie description is represented using word embeddings. Word embeddings are extracted using FastText and Word2Vec to be further compared in terms of performance. The embedding size of each word is priorly set. Thus, each movie is represented by an N x M matrix where N is the number of words, M is the word embedding size. CNN takes a list of tokens of words in the movie description as input. Tokens are the index of a word in the lookup table. The lookup table is generated right after the word embedding extraction process. CNN is structured to have three parallel convolutional paths with different filter sizes in order to extract information of different granularity. Each convolution operation is terminated by ReLU activation function. The generated feature maps are then max-pooled for both dimension reduction purposes and hopefully promoting textual features that reflect specific genre(s). At the end of the convolutional layer, the obtained feature maps are concatenated into only one vector to be treated as one global feature vector of text sequence. In the fully connected layer, sigmoid activation is preferred since it's a multi-label classification problem where each sample can belong to more than one class. Sigmoid activation's nature lets the output be independent of other classes, so each class is evaluated individually. Sigmoid transforms the output to the probability

distribution which is stated in the equation below:

$$S(x) = \frac{1}{1 + e^x} \tag{4.1}$$

where $S(x)$ represents the sigmoid function. After the fully connected layer, the movie-genre relationship is extracted as a multi-dimensional vector. Those vectors represent movies in a more complex way; thus having more information about the movie. Genre prediction is just for the training phase, it doesn't serve the main purpose. While training, three different types of loss functions are compared based on their best parameter setting. For each loss function, the best parameter setting is searched through scanning the parameter space of CNN. Those functions are binary cross-entropy loss which is appropriate to use with sigmoid activation, cosine similarity loss, and focal loss.

4.1.1. Binary Cross-Entropy Loss

Binary cross entropy compares each predicted probability to the ground truth which can be either 0 or 1. Then it calculates the score which further penalizes the probabilities through calculating the distance from the ground truth. The binary cross-entropy loss function is given below:

$$L = -\sum_{i}^{N}\sum_{j}^{i}[y_j^i \log(o_j^i) + (1 - y_j^i)\log(1 - o_j^i)] \tag{4.2}$$

where N is the total number of training samples, t is the total number of labels, o_i is the model output, y_i is the target label for every training sample and y_j^i is 1 if training sample i has the label j, otherwise 0.

4.1.2. Cosine Similarity Loss

Cosine similarity loss is the adaptation of cosine similarity function for neural networks so that the loss function version can be minimized in gradient descent. It is also referred to as cosine proximity and corresponding calculation is given below:

$$cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \, \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{4.3}$$

where **A** and **B** are given and predicted vectors, respectively. Note that, higher similarity indicates higher accuracy, and as the similarity increases, the cosine similarity gets closer to 1. For neural network's usage as loss function, it is transformed mathematically as

$$Cosine\ similarity\ loss = -1 \text{ x } (\,Cosine\ similarity). \tag{4.4}$$

The reason behind trying cosine similarity loss is that we're trying to make our predicted genre vector as close as possible to the ground truth vector.

4.1.3. Focal Loss

Focal loss is proposed by Facebook AI research [22] for alleviating foreground-background class imbalance in object detection tasks. They claim that large class imbalances overwhelms the traditional cross-entropy loss function because easily classified negative classes comprise the majority of the loss and dominate the gradient. In detail, each input has been evaluated by the output layer with its most dominant features. Although subtle features of non-dominant labels are also detected by CNN layer, they might be ignored on the fully connected layer because of loss function's inability. As a more concrete example background objects are usually ignored and the input can be evaluated as negative example for that class and this results with class imbalance in multi-label image classification tasks. Focal loss addresses the class imbalance by reshaping the standard cross-entropy loss such that it down-weights the loss assigned to well-classified examples. To this end, a new modulating factor $(1 - p_t)^\gamma$ is added to the cross-entropy loss with tuneable focusing parameter $\gamma \geq 0$. The focal loss is defined as

$$\mathrm{FL}(p_t) = \alpha_t (1 - p_t)^\gamma \log(p_t) \tag{4.5}$$

where α is the balancing factor and γ focusing parameter. Focusing parameter adjusts the rate at which easy examples are down-weighted. Intuitively, this factor reduces the loss contribution of easily classified examples while giving the credit to the hardly classified examples (i.e. background objects in image). This way, the class imbalance claimed to be solved. The foreground-background class imbalance in object detection task is in a sense similar to the class imbalance of multi-label text classification setting; some features of the text might dominate other features resulting in preferring some classes over others.

After finding the best fit loss function, trained embeddings are stored for further usage in next movie recommendation module. The overall CNN architecture is depicted in Figure 4.2. This figure is directly plotted by means of Keras helper function.

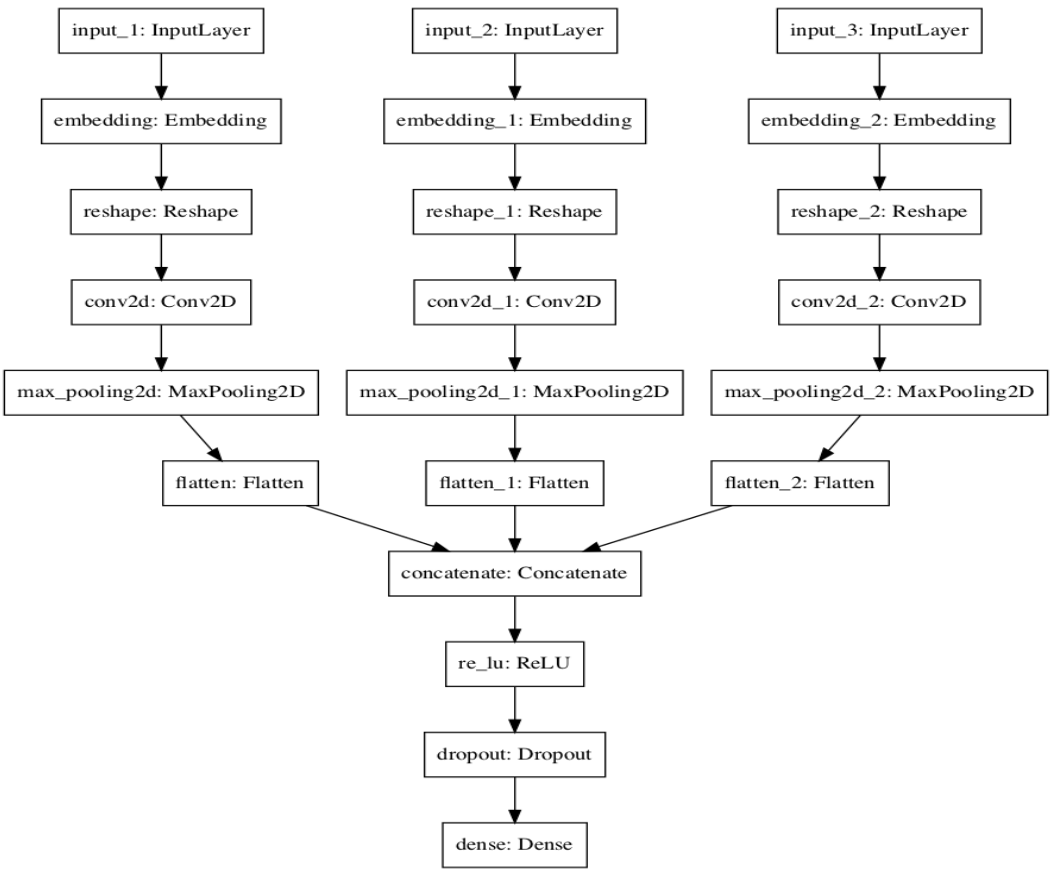

Figure 4.2. Our CNN architecture directly plotted from the source code.

4.2. Next Movie Recommendation Module

This module aims to predict, for each user, the next most likely to-be-watched movie based on the sequence of previously rated movies and those ratings. Since sequence information is crucial, the best-fit neural network architecture is LSTM. The module architecture is in Figure 4.3.

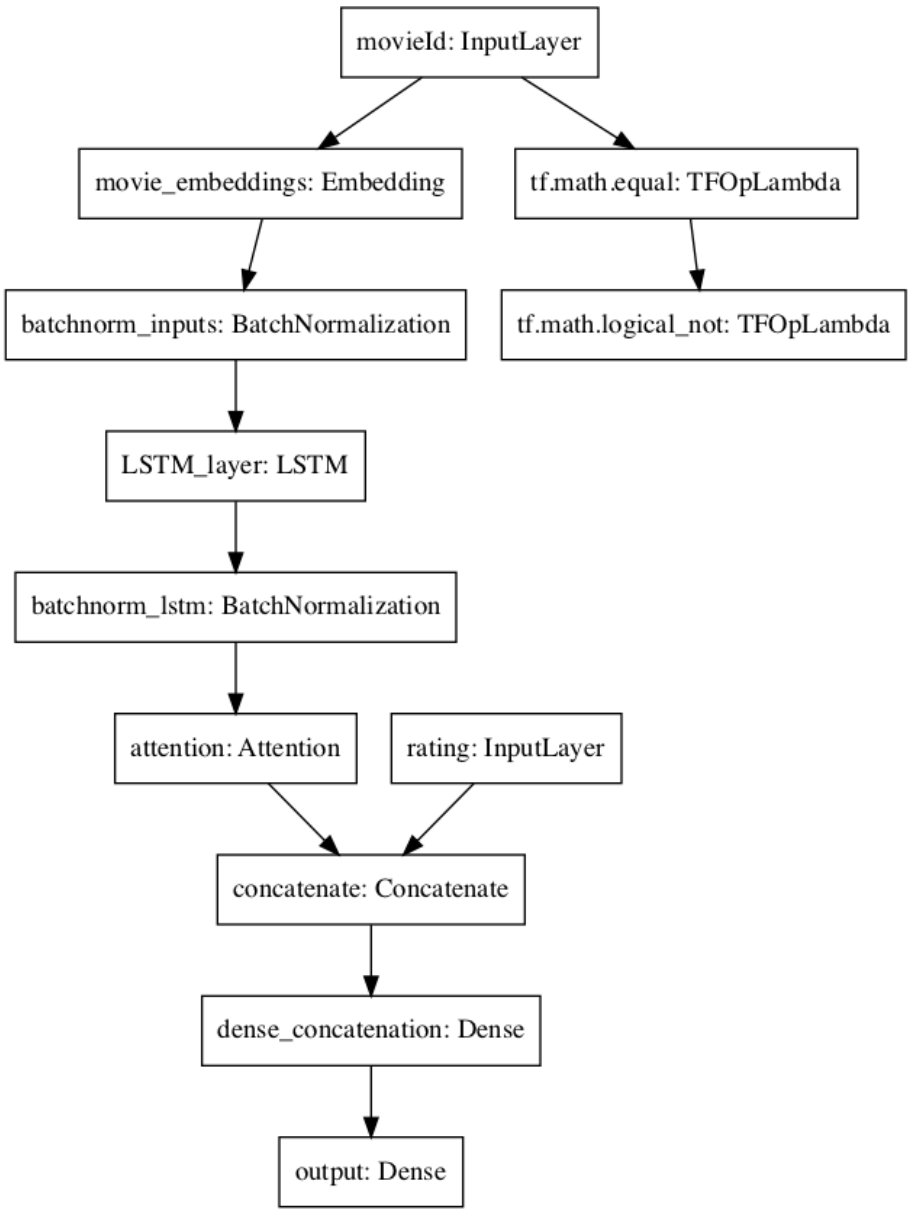

Figure 4.3. Our RNN architecture directly plotted from the source code.

The model takes movie description embeddings from the previous module regarding their corresponding movie ids. In each training batch, only one user's sequence information is taken into account. To be able to be fed into a neural network, each input should have the same length, a training sample has to be padded if its length is less than the maximum length. This length should be set manually. However, it is known that zero-padded inputs affect networks performance in terms of convergence in such sequence-to-sequence RNN architectures. Therefore we decided to use the attention layer that is introduced by Bahdanau *et al.* [23]. and which has been identified as a performance booster for sequence-to-sequence model architectures. The zero padding of each input is masked and after LSTM produces its output, it is masked in the attention layer. There is no need for masking the rating information since it is not fed into LSTM and our main contribution is to solve the item cold-start problem where the movie rating is missing. Attention layer is followed by the concatenation layer and the dense layer for learning the weights of the concatenated feature vector. At last, a list of candidate movie embedding are produced by adding one more dense layer.

The rated movies share the same sequence if they are rated by the same user within the same three-month interval. In the training phase, the target vector consists of the user's next interacted movie in the same sequence. The example input is shown in Figure 4.4.

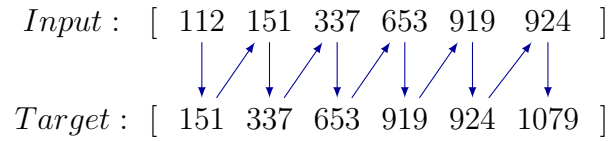

Figure 4.4. Input and target list for each sequence consist of movie ids.

Figure 4.4 shows an input example where the sequence length is equal to 6. Target list is constructed by sliding the input list forward, where each movie in target list is the next movie in the input list.

We applied categorical cross entropy as the loss function of the system. It is shown as

$$\mathbf{J(w)} = \frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{4.6}$$

where $\mathbf{w}$ refer to the weights of the neural network, y_i is the true label and $\hat{y}_i$ is the predicted label.

5. EXPERIMENTS AND RESULTS

5.1. Dataset

Two different datasets related to movies are used for the whole system. The movie embedding extraction module utilizes IMDB movie dataset [24]. However, description of movies doesn't exist in the IMDB movie dataset, thus each description is retrieved by using OMDB API [25] and IMDBPy [26]. Table 5.1 shows the number of movies of each genre in the dataset. The dataset only covers 180956 movies. It is compelling for embedding extraction phase since data is both imbalanced and insufficient for training the system. Data augmentation techniques are applied, however, it is shown in Section 5.4.1.2 that with the selected pre-embeddings, the module doesn't need extra data since it overcomes the imbalanced data issue with its powerful embedding extraction method.

For the recommendation module, MovieLens 20M dataset [27] is used. Each user's rating history is grouped by timestamp and and two movies share sequence if they are in the same three-months interval.

5.2. Evaluation Metrics

Metrics are for quantifying the performance of a machine learning system and plays an important role for comparing performances of systems with the same aim. Metrics are classified as thresholding metrics, ranking metrics, and probability metrics. Here, four thresholding metrics and a ranking metric will be emphasized. Thresholding metrics are accuracy, precision, recall, and F1. Ranking metric is Hit Ratio.

Table 5.1. Table of movie numbers for each genre in the dataset(by 2019).

Movie Genre	**Number of Movies in the Dataset**
Action	20711
Adventure	12426
Animation	3136
Comedy	42985
Crime	16269
Documentary	37150
Drama	79689
Fantasy	6365
Film-Noir	770
Horror	14603
Musical	4046
Mystery	8159
Romance	21001
Sci-Fi	6414
Thriller	20191
War	4678
Western	4516

5.2.1. Accuracy

It is the calculation of the number of correct predictions divided by the total number of predictions.

5.2.2. Precision

It shows a classification model's ability to identify only the relevant data points. The calculation of precision for a classification task is stated below:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} . \tag{5.1}$$

5.2.3. Recall

It shows a classification model's ability to return only relevant instances. The calculation of recall for a classification task is stated below:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} . \tag{5.2}$$

5.2.4. F-score

The harmonic mean of precision and recall scores, as stated in the equation below:

$$\text{F-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} . \tag{5.3}$$

5.2.5. Hit Ratio (HR@k)

This metric checks whether the selected point is present in the top-k ranked points. Calculation is stated in the equation below:

$$\text{Hit Ratio} = \frac{\text{number of hits}}{\text{number of hits} + \text{number of misses}} \tag{5.4}$$

where hit refers to the true prediction while miss refers to the false prediction.

5.3. Baseline Method - NeuMF

He *et al.* [28] construct a recommendation system model that exploits deep neural network structures to produce recommendations using implicit data. They named their architecture neural collaborative filtering, also referred to as NeuMF. They combine multi-layer perceptron with a standard matrix factorization network in order to learn a non-linear function while increasing the effectiveness of the model. Original work

uses both Movielens and Pinterest datasets and utilizes both user and item features. While evaluating the model, they compare their method with a simple MLP-based recommendation model and a simple GMF-based recommendation model on hit ratio metric. NeuMF slightly outperforms both solo models. For comparing this method with ours, It is first converted to the Tensorflow v2.5, and adapted to our mixed movie dataset. The architecture of NeuMF is shown in Figure 5.1.

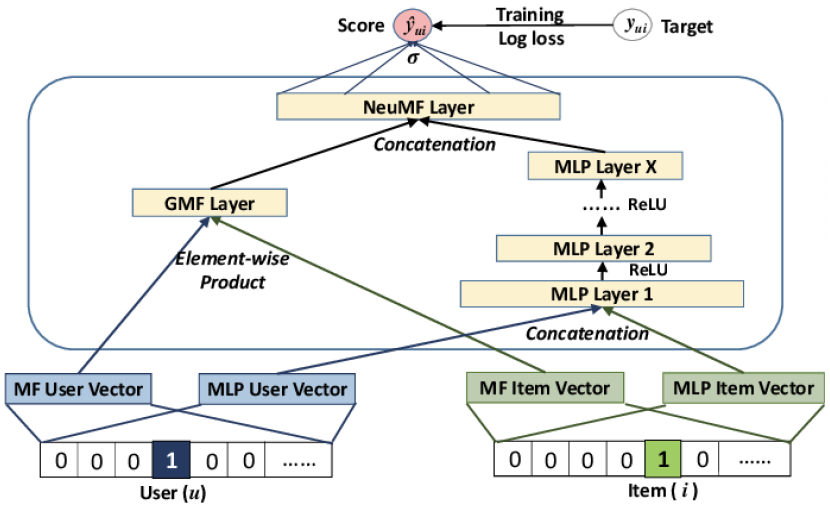

Figure 5.1. NeuMF Architecture [28].

5.4. Results

In this section, we cover the experiments module by module; both neural network architectures separately trained, evaluated, and results are elaborated.

5.4.1. Movie Embedding Extraction Module - CNN

First of all, we seek for the best loss function that fits to our CNN model. For each loss function, best parameter setting is searched and found. Parameter search space is indicated in Table 5.2. Best parameter setting for each loss function is in Table 5.3.

Table 5.2. Parameter search space for CNN.

	List of possible parameters
Number of filters of CNN	[128, 256]
Learning rate	[0.001, 0.01]
Norm ratio for L2 regularization	[1.0, 2.0, 3.0]
Decay steps	[50.000, 75.000, 100.000]

Table 5.3. Best parameter settings for loss functions.

	Cosine Loss	**Focal Loss**	**Binary Cross Ent. Loss**
N. of filters of CNN	256	256	256
Learning rate	0.01	0.01	0.01
Norm Ratio	2.0	3.0	3.0
Decay Steps	50.000	50.000	75.000

Table 5.4. Comparison of loss functions.

	Precision	**Recall**	**F1**
Binary cross entropy loss	**0.8018**	0.5074	**0.6206**
Focal loss	0.4095	0.4277	0.4184
Cosine similarity loss	0.5061	**0.5414**	0.5219

5.4.1.1. Loss Function Comparison. Table 5.4 shows the comparison of loss functions in terms of precision, recall and F1 scores. Binary cross entropy loss significantly produces better precision and F1 scores. However, cosine similarity loss outperforms other loss functions in recall score. It might be due to the fact that cosine similarity loss tries to minimize the distance between the true value and the predicted value and recall shows the ability of the model to find all the relevant cases; meaning the exact match in our case. Cosine similarity loss function implicitly tries to find the perfect match and updates the model parameters accordingly. However, we need to take precision score

into account as well since precision shows the model's ability to identify only relevant data points. While using embeddings in the recommendation module, false positives (the movies that does not belong to that genre, however it is classified into that genre by the system) cannot be ignored since it might cause the system to produce irrelevant recommendations. Thus we want a balance of both scores, and binary cross entropy loss is not only the best performer in F1 score, but also its recall score is close to the one of cosine similarity loss.

In Figure 5.2, Figure 5.3, and Figure 5.4, we can see the value changes of scores per epoch. Early stopping technique is applied for each training. The maximum epoch number is set to 20. Training with binary cross entropy loss didn't stop at epoch 20, it is forced to terminate. Training with focal loss stop at epoch 16 and training with cosine similarity loss stop at epoch 18. It can be inferred that, binary cross entropy loss could produce better scores if maximum epoch number were higher, however it is stopped due to the limitations of computational sources.

On the other hand, we expect focal loss to produce higher scores because it seems like it fits into our class imbalance problem in CNN. It might be due to the wrong parameter setting of focal loss which are balancing factor and focusing parameter. Parameter space of focal loss cannot be searched due to the both time and computational resource limitations.

5.4.1.2. Word2Vec vs. FastText. The aim is to pick the word embedding which makes the embedding module converge as fast as possible. For each word embedding, binary cross entropy loss is set with its best parameter setting. Maximum epoch number is set to 25. Embedding size is set to 256. Word2Vec hasn't converged yet when the training hits 25 epoch whereas FastText converged in 22^{nd} epoch. The reason behind significant performance of FastText is that it makes use of character level information. In movie descriptions, there is too much rare words like movie character names and abbreviations. Hence, FastText is preffered as movie embedding extraction pre-embeddings.

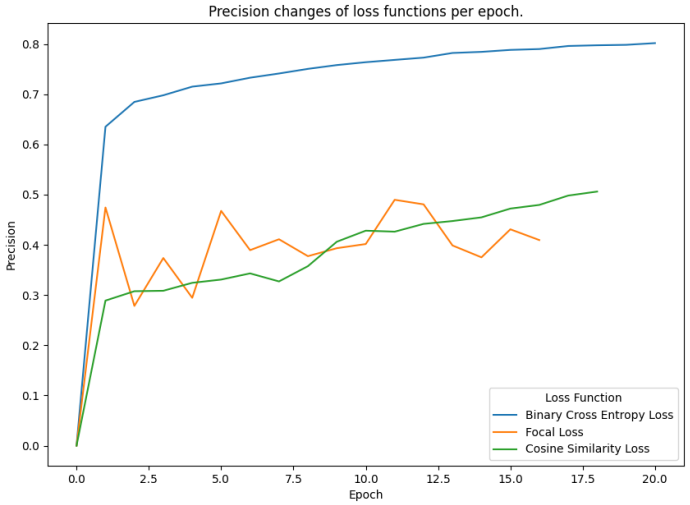

Figure 5.2. Precision changes of loss functions per epoch.

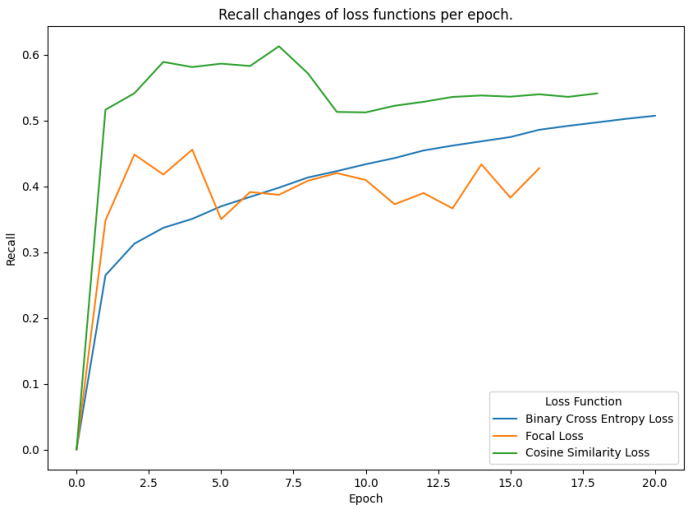

Figure 5.3. Recall changes of loss functions per epoch.

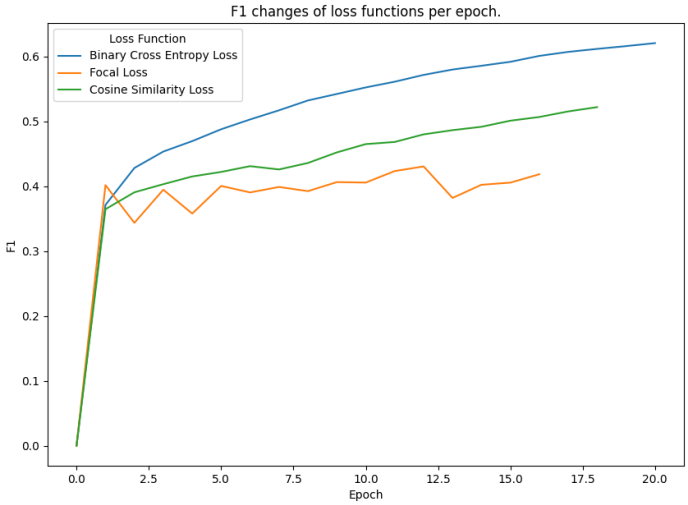

Figure 5.4. F1 changes of loss functions per epoch.

5.4.2. Next Movie Recommendation Module - RNN

As stated before, we compare our model with NeuMF model. In the original paper, MovieLens 1M is used. We adapted MovieLens 20M. For ensure the equality, we used ragged tensors for users rating history in order to involve the users that rated less than 20 movies into the training phase. In the original work, they only consider users that rates equal or more than 20 movies. While feeding into the neural network, they turned the rating information to explicit information where each entry is marked as 0 or 1 indicating whether user is rated the movie. We applied the same approach for our dataset. While evaluating their model, we set the predictive factors to 64 which they obtain the best score in the original work. We trained and evaluated both models for maximum 20 epochs. Scores are depicted in Figure 5.5 and Figure 5.6.

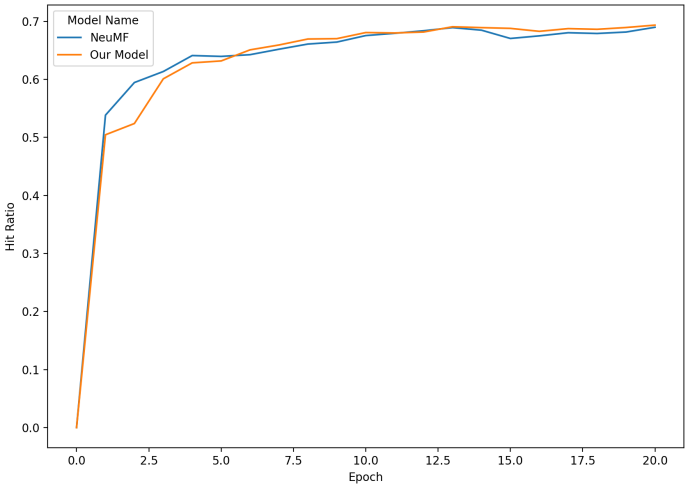

Figure 5.5. Comparison of HR@10 changes per epoch.

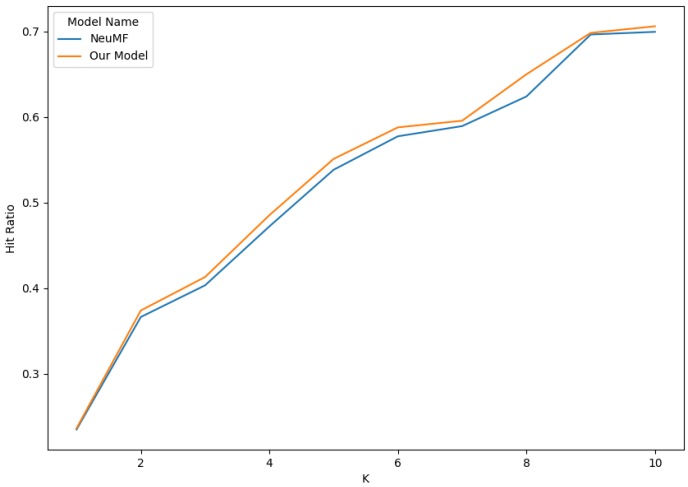

Figure 5.6. Comparison of HR@k where K ranges from 1 to 10.

Figure 5.5 shows hit ratio changes over epoch where K=10 and figure 5.6 shows hit ratio changes where K ranges from 1 to 10. Our model slightly outperforms NeuMF approach. The reason behind those results is that our model both exploits movie description text in a very efficient way and the robustness of our recommendation module

where we use attention layer in order to alleviate the zero-padding issue. However, we expected to have better results. It might be due to the fact that our recommendation model has a poor parameter setting since we couldn't search the parameter space. Moreover, we actually empowered the NeuMF architecture by adding the ragged tensors; this way model exploits sparse data and its performance is increased.

Since our dataset is a merge of MovieLens and IMDb, and IMDb movies has no user-specific ratings, we can consider IMDb dataset's discrete set as cold start items. However, we trained and evaluated our model but we couldn't test it due to the time constraints; we can only infer that our model can cover cold-start item recommendation.

6. CONCLUSION AND FUTURE WORK

Starting point of this thesis was to solve the item cold-start problem in recommendation systems, however the study become multi-domain since we both tackle multi-label text classification problem and recommendation for solving the performance issues. This work conducted an extensive study on multi-label text classification task by trying different loss functions and introducing focal loss to the text classification task and compares it with other loss functions. With the best parameter setting, movie embeddings are extracted by only using movie description texts and movie genres. Those embeddings further augment the recommendation performance. Furthermore, with our robust RNN architecture that uses attention layer, this method outperforms NeuMF architecture.

As future work, first of all we'll try to involve the ragged tensors into our architecture as well. We couldn't explicitly demonstrate that our model can alleviate the cold start problem; we only infer regarding our architecture. we can conduct experiments to see if our model covers the item cold-starts. Moreover, we can add more CNN layers to our embedding module for more fine-grained feature extraction, we can incorporate additional user features like user's reviews on movies, or user's browsing history. User cold-start case can also be tackled. Moreover, we can use our architecture for transfer learning; we can tune our network to recommend books to users.

REFERENCES

1. Huang, W., K. K. Lai, Y. Nakamori and S. Wang, "Forecasting Foreign Exchange Rates With Artificial Neural Networks: A Review", *International Journal of Information Technology and Decision Making*, pp. 145–165, 2004.

2. Goodfellow, I., Y. Bengio and A. Courville, *Deep Learning*, The MIT Press, 2016.

3. Kim, Y., "Convolutional Neural Networks for Sentence Classification", *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014.

4. Fayyaz, M., M. Hajizadeh Saffar, M. Sabokrou, M. Fathy, R. Klette and F. Huang, "STFCN: Spatio-Temporal FCN for Semantic Video Segmentation", *Computing Research Repository (CoRR)*, 2016.

5. Mikolov, T., K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space", *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, pp. 1–12, 2013.

6. Bojanowski, P., E. Grave, A. Joulin and T. Mikolov, "Enriching Word Vectors with Subword Information", *Transactions of the Association for Computational Linguistics*, Vol. 5, 2016.

7. Gargiulo, F., S. Silvestri and M. Ciampi, "Deep Convolution Neural Network for Extreme Multi-Label Text Classification", *International Workshop on Artificial Intelligence for Health*, Vol. 5, pp. 641–650, 2018.

8. Mikolov, T., K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space", *Proceedings of the International Conference on Learning Representations*, pp. 1–12, 2013.

9. Gargiulo, F., S. Silvestri, M. Ciampi and G. De Pietro, "Deep Neural Network for Hierarchical Extreme Multi-Label Text Classification", *Applied Soft Computing*, Vol. 79, pp. 125–138, 2019.

10. "A Challenge on Large-Scale Biomedical Semantic Indexing and Question Answering", `http://bioasq.org/`, 2012, accessed in June 2021.

11. Wang, G.-H., C.-H. Chung, Y. Chen and H. Chen, "Multi-Label Playlist Classification Using Convolutional Neural Network", *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pp. 1957–1962, 2018.

12. Chen, Y., B. Xiao, Z. Lin, C. Dai, Z. Li and L. Yan, "Multi-label Text Classification with Deep Neural Networks", *2018 International Conference on Network Infrastructure and Digital Content (IC-NIDC))*, pp. 409–413, 2018.

13. Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions", *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

14. Oramas, S., O. Nieto, M. Sordo and X. Serra, "A Deep Multimodal Approach for Cold-Start Music Recommendation", *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*, pp. 32–37, 2017.

15. Moro, A. and R. Navigli, "Babelfly, A Unified, Multilingual, Graph-Based Approach to Entity Linking and Word Sense Disambiguation", `http://babelfy.org/`, accessed in June 2021.

16. "DBPedia, Global and Unified Access to Knowledge Graphs", `https://www.dbpedia.org/`, accessed in June 2021.

17. Volkovs, M., G. Yu and T. Poutanen, "DropoutNet: Addressing Cold Start in Recommender Systems", *Advances in Neural Information Processing Systems*, Vol. 30,

pp. 32–37.

18. Kiran R, D., P. Kumar and B. Bhasker, "DNNRec: A Novel Deep Learning Based Hybrid Recommender System", *Expert Systems with Applications*, Vol. 144, pp. 32–37, 2019.

19. De Souza Pereira Moreira, G., F. Ferreira and A. M. Da Cunha, "News Session-Based Recommendations Using Deep Neural Networks", *Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems*, pp. 32–37, 2018.

20. Hidasi, B., A. Karatzoglou, L. Baltrunas and D. Tikk, "Session-Based Recommendations with Recurrent Neural Networks", *Computing Research Repository (CoRR)*, 2016.

21. Suglia, A., C. Greco, C. Musto, M. de Gemmis, P. Lops and G. Semeraro, "A Deep Architecture for Content-Based Recommendations Exploiting Recurrent Neural Networks", *UMAP '17: Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, pp. 202–211, 2017.

22. Lin, T.-Y., P. Goyal, R. Girshick, K. He and P. Dollár, "Focal Loss for Dense Object Detection", *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2999–3007, 2017.

23. Bahdanau, D., K. Cho and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate", *The International Conference on Learning Representations (ICRL), 2015*, 2016.

24. "IMDb Datasets", `https://www.imdb.com/interfaces/`, accessed in June 2019.

25. "The Open Movie Database API", `https://www.omdbapi.com/`, accessed in June 2019.

26. "IMDbPy, Python Package for Retrieving and Managing the Data of the IMDb

Movie Database", https://imdbpy.github.io/, accessed in June 2019.

27. "MovieLens 20 Million Dataset", https://grouplens.org/datasets/movielens/20m/, accessed in June 2019.

28. He, X., L. Liao, H. Zhang, L. Nie, X. Hu and T.-S. Chua, "Neural Collaborative Filtering", *WWW 2017 International World Wide Web Conference*, 2017.