ANALYZING DEEP LEARNING ALGORITHMS FOR RECOMMENDER SYSTEMS

An Undergraduate Research Scholars Thesis

by

TIANYU GU

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. James Caverlee

May 2022

Major: Computer Science

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Tianyu Gu, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Analyzing Deep Learning Algorithms for Recommender Systems

Tianyu Gu
Department of Computer Science and Engineering
Texas A&M University

Research Faculty Advisor: Dr. James Caverlee
Department of Computer Science and Engineering
Texas A&M University

As the volume of online information increases, recommender systems have been an effective strategy to overcome information overload by giving selective recommendations based on certain criteria such as user ratings and user interactions. Recommender systems are utilized in a variety of fields, with common examples being music recommendations and product recommendations on E-Commerce websites. These systems are usually constructed using either collaborative filtering, content-based filtering, or both. The most traditional way of developing a collaborative filtering recommender system is using matrix factorization, which works by decomposing a user-item interaction matrix into the product of two lower dimensionality rectangular matrix. However, as new technologies appear, matrix factorization is often replaced by other algorithms that could perform better than in a recommendation system.

In recent years, deep learning has garnered considerable interest in many research fields such as computer vision and natural language processing. These successes are made possible by deep learning algorithms' outstanding ability to learn feature representations non-linearly. The

influence of deep learning is also prevalent in recommender systems, as demonstrated by its effectiveness when applied to information retrieval and recommender research. This research project performs an analysis and implementation on variants of two deep learning algorithms, autoencoder and restricted Boltzmann machines, and how they perform in recommender systems compared to matrix factorization.

ACKNOWLEDGEMENTS

Contributors

I would like to thank my faculty advisor, Dr. James Caverlee, and my research mentor, Yin Zhang, for their guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

The MovieLens data analyzed/used for applying deep learning algorithms for recommender system analysis were provided by GroupLens.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

No funding was received for this research.

NOMENCLATURE

BM	Boltzmann Machine
ERBM	Explainable Restricted Boltzmann Machine
RBM	Restricted Boltzmann Machine
RMSE	Root Mean Square Error
SELU	Scaled Exponential Linear Units

1. INTRODUCTION

Broadly speaking, there are three types of recommender systems: collaborative filtering, content-based filtering, and hybrid (a mixture of the two previous methodologies) [1]. Collaborative filtering utilizes past user-item interactions to predict new recommendations and interactions. On the other hand, content-based filtering requires extra information about users and or items to generate recommendations [2]. This research project focuses on applying deep learning models to collaborative filtering-based recommender systems.

1.1 Types of Collaborative Filtering

There are two types of collaborative filtering approaches: memory-based and model based. Memory-based collaborative filtering uses the user-item interaction matrix directly and fetches similar users and/or items for predictions [3]. Model-based collaborative filtering analyzes interactions between users and items and builds a model based off these interactions that provides the recommendations. This research will focus on model-based approaches for building recommender systems.

1.2 Common Issues with Traditional Recommenders

There are several common issues that recommender systems often face. In this paper, we will look at how deep learning techniques can be incorporated into traditional recommenders to mitigate these problems.

- **Data Sparsity**: This is a problem that occurs when the users and items matrix is filled sparsely [3]. Due to the sparse relationships, recommendation accuracy and hit ratio drop.
- **Cold-Start**: This issue occurs when the recommender system is unable to recommend to new users or when it cannot recommend new items [3].

- **Inability to Analyze Multimedia**: Most of the earliest recommenders are unable to handle data that are not text or numbers. With the incorporation of more advanced algorithms such as deep learning, recommendation systems are now able to analyze heterogeneous data sources (images, audio, video etc.) Although the dataset used in this research would only include text and numbers, the models implemented can also be used to analyze multimedia data.

1.3 Deep Learning for Collaborative Filtering

1.3.1 Strengths of Deep Learning Recommendation Systems

- **Ability to Model Nonlinear Interactions [4]**. Unlike linear models, deep learning neural networks are able to model nonlinearity in data with nonlinear activation functions. This functionality allows the model to capture complex user-item interactions that may otherwise be unidentified with linear models. For example, matrix factorization, which is a linear model, would be unable to capture the complex nonlinear interactions because it linearly combines user and item latent factors. Nevertheless, user preferences are often nonlinear and require complex models to detect these underlying patterns.
- **Efficiency in Representation Learning [4]**. Deep learning neural networks are efficient in learning from underlying latent factors and representations from data. With large amounts of user-item interaction data, learning about these factors and representations would improve understanding of users and items, thus resulting in a better performing recommender.
- **High Flexibility [4]**. There is a great variety of deep learning frameworks, including Tensorflow, PyTorch, Keras etc. Most of these tools are flexible in the sense that they are developed using modular programming, which is a software design technique that

separates functionalities of a program into independent and interchangeable modules. This enables the development of various hybrid models and neural structures to be more efficient.

1.3.2 Drawbacks to Deep Learning in Recommender Systems

- **Less Interpretable than Traditional Methods [4]**. Deep Learning algorithms are generally less interpretable because they tend to behave like black boxes. Unlike traditional methods (matrix factorization etc.) whose algorithms and matrix vectors are clearly explainable, the hidden layers and weight vectors for deep learning models are difficult to explain. However, with recent advanced research on neural architectures, neural models have become more interpretable than the past, which makes machine learning models even more desirable for recommender systems.
- **Massive Data Requirement [4]**. In order for a deep learning neural network to be well-trained, large amounts of data need to be fed into the model. However, this problem is not significant in recommender systems research because data are mostly readily available.
- **Hyperparameter Tuning [4]**. The performance of a deep learning or machine learning model is highly dependent on its parameters. This process is highly tedious and model performance can be highly depreciated if hyperparameters are not tuned properly.

2. MODELS

Various types of collaborative filtering models have been proposed to make recommendations more personalized. This section outlines the general concept behind each type of algorithm and model. The specific models and details would be explained further in the methods section.

2.1 Matrix Factorization

Matrix factorization is one of the traditional collaborative filtering methods for creating recommender systems. It is a way to generate latent features by decomposing a user-item interaction matrix into two lower dimensional matrices inferred from user-item rating patterns. These two lower dimensioned matrices can then be used to estimate user-item interaction values that were initially null.

For example, with the input of users' ratings on different movies, we can predict how the users would rate new movies based on their past ratings. Based on these estimated ratings, the recommender can then make predictions on what the users may like and recommend users based on these predictions.

A vanilla matrix factorization model would involve two vectors. Each user u is mapped to a vector v_u while each item i is mapped to vector v_i. For each user u, the elements in vector v_u represents the correlation the user has to specific latent factors. Similarly, for each item i, the elements in v_i represents the correlation the item has to corresponding latent factors. Once the two vectors are multipled together using dot product, the resulting dot product is able to capture the interaction matrix between each user and item pair [5]. This logic is represented in Equation 2.1.

$$R_{ui} = v_i \cdot v_u \tag{2.1}$$

To obtain the mapping of each item and user to vectors v_i and v_u, we have to complete an optimization problem, which minimizes the regularized squared error on the set of available ratings [5]. This equation to minimize is shown in Equation 2.2.

$$\sum(R_{ui} - v_i \cdot v_u)^2 + Regularize(v_i + v_u) \tag{2.2}$$

This model learns by generalizing known ratings to predict unknown ratings, which may also be from new users and items. However, overfitting on the training data is a potential problem in this case. Therefore, variants of the vanilla matrix factorization model would have other parameters that are added to minimize overfitting.

2.2 Autoencoder

An autoencoder is a type of directed neural network that has both encoding and decoding layers [6]. As this network is best suited for unsupervised learning, it has been shown to be exceptional in learning underlying feature representations in fields such as computer vision and natural language processing. By learning the latent set of features, the input data can be compressed in the mid layers [6]. The reconstruction process happens in the decoding layer, where inference for prediction happens.

There are typically three components in an autoencoder: visible input layer, hidden layers, and visible output layer. Figure 2.1 provides a general structure of a vanilla autoencoder. There have been many variations on how autoencoders can be applied in recommender systems, two of which will be discussed in this paper.

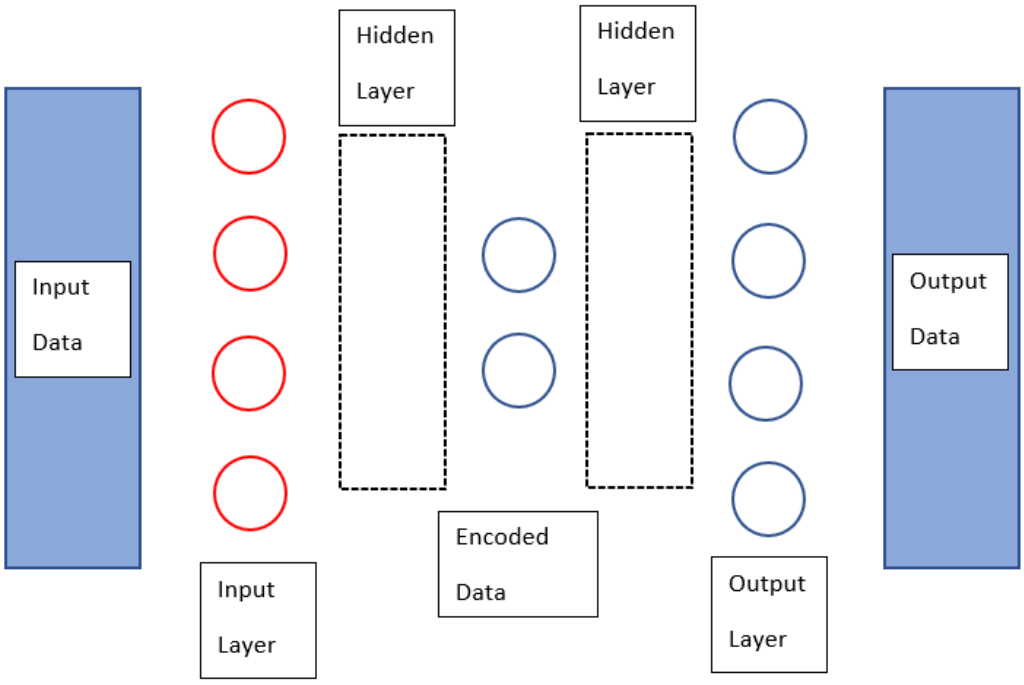

Figure 2.1: Structure of a vanilla autoencoder

In the hidden layers of the encoder, the encoder transforms the higher dimensional input data, x, into lower dimensioned encoded data, h, with a function f. This function is shown in Equation 2.3.

$$h = f(x) = S_f(Wx + b) \tag{2.3}$$

S_f : activation function (Sigmoid, TanH, ReLU etc.)

W: weight matrix

b : bias vector

On the other side, the decoder decodes the encoded data h back using a reconstruction function g. The reconstruction function is shown in Equation 2.4.

$$x' = g(h) = S_g(W'h + b') \tag{2.4}$$

S_g : activation function (Sigmoid, TanH, ReLU etc.)

W' : weight matrix for reconstruction

b' : bias vector for reconstruction

In order for the reconstruction of the original input data to be as accurate as possible, the autoencoder can be trained to minimize the reconstruction error between x and x' using the squared error (Equation 2.5) for regression problems and cross-entropy error (Equation 2.6) for classification problems.

The formula for squared error between input data, x, and reconstructed data, x' is:

$$S.E. = ||x - x'||^2 \tag{2.5}$$

The formula for cross-entropy error is:

$$C.E. = -\sum_{i=1}^{n}(x_i log x_i' + (1 - x_i)\log(1 - x_i')) \tag{2.6}$$

2.3 Restricted Boltzmann Machines (RBM)

According to the inventors of the Boltzmann Machines, "A Boltzmann Machine is a network of symmetrically connected, neuron-like units that make stochastic decisions about whether to be on or off [7]." In other words, each node in a Boltzmann Machine network has a probability associated with it that are used to make binary decisions. It can be used for two types of computational problem: the search problem and the learning problem.

- **Search Problem.** The weights of the connections connecting the nodes are fixed and used to represent the cost function of an optimization problem [7].
- **Learning Problem.** The weights of the connections are not initially given. In this problem, the Boltzmann Machine is shown a set of binary data vectors that would be used to find the weights of the connections. This is done by calculating the set of weights

that would make the binary data vectors the optimal solution for the optimization problem [7].

The Restricted Boltzmann Machine is a special type of Boltzmann Machine with two layers of nodes. The first layer contains visible nodes while the second layer contains hidden nodes. There are inter-layer connections between nodes, but no intra-layer connections. The general architecture of a RBM is shown in Figure 2.2.

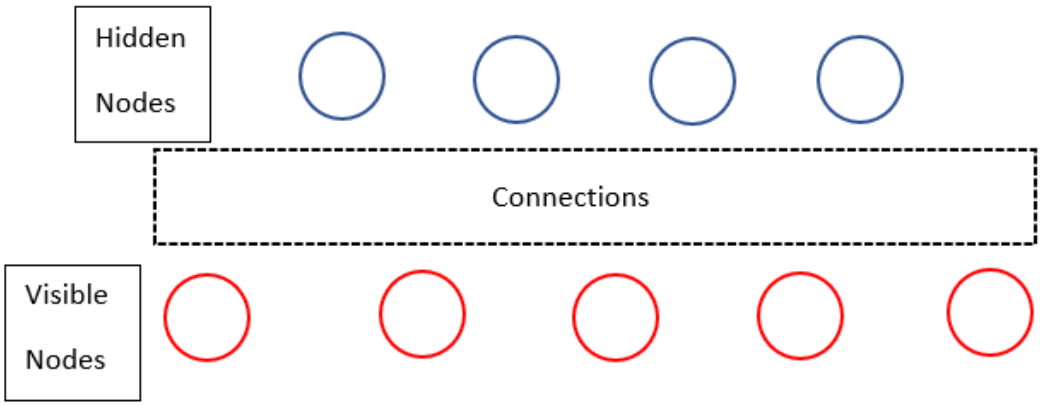

Figure 2.2: Structure of a vanilla Restricted Boltzmann Machine

3. METHODS

3.1 Data Preprocessing

The dataset used for this research project is the public MovieLens 20M dataset containing movie reviews in the schema as shown in Figure 3.1.

userID	movieID	rating	timestamp
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596

Figure 3.1: Snippet of MovieLens 20M dataset

In this dataset, users give ratings for movies within the range 1-5. In order to apply the various models, the dataset needs to be preprocessed into a matrix format to represent user-item interactions. In Python, this would be a Pandas data frame. For user-item interactions that were initially null, a value "0" would be given to replace the null value.

3.2 Training and Testing Data

To split the dataset into training and testing data, the leave-one-out methodology is used (Figure 3.2). In this strategy, the most recent review (according to the timestamp) for each user is used as testing data while the rest will be used as training data. For example, the movies reviewed by user 383 is shown below. The last movie reviewed by the user is the 2014 movie Guardians of The Galaxy. This data point will be used as the testing data for this user, and the rest of the reviewed movies will be used as training data.

This train-test split strategy is commonly used in training and evaluating recommender systems to avoid biases. If a random split is done on the dataset, a user's recent reviews could

potentially be used for training while the older reviews are used for testing. This is called the look-ahead bias, which would then decrease the performance of the trained model significantly.

Therefore, to reduce this bias, the dataset is split into a train and test set using the leave-one-out methodology.

```
ratings['rank_latest'] = ratings.groupby(['userID'])['timestamp'].rank('first', ascending=False)

train_ratings = ratings[ratings['rank_latest'] != 1]
train_ratings = ratings[ratings['rank_latest'] == 1]

train_ratings = train_ratings[['userId', 'movieId', 'rating']]
test_ratings = test_ratings[['userId', 'movieId', 'rating']]
```

Figure 3.2: Train-test split using leave-one-out methodology

3.3 Matrix Factorization

There are many different variants of matrix factorization. In this research, the model that is utilized is the vanilla matrix factorization model that involves a regularization parameter. This is a particularly important model to understand because it serves as one of the foundation methods for collaborative filtering in recommender systems. Figure 3.3 gives a visual representation of matrix factorization.

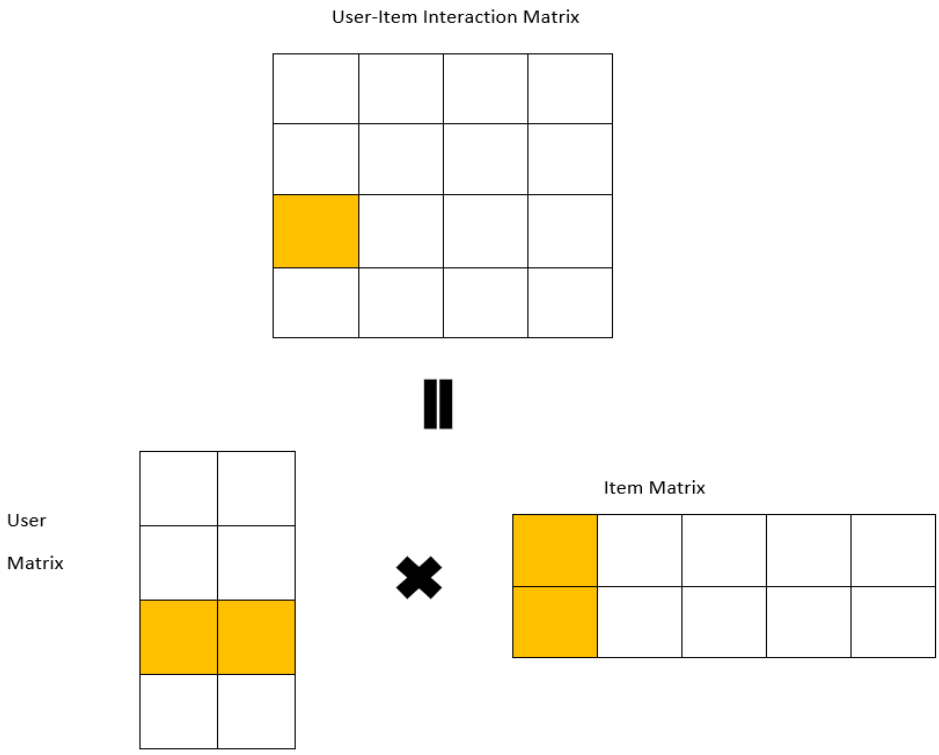

Figure 3.3: Matrix Factorization

In the implemented matrix factorization, the original user-item interaction matrix is decomposed into a user matrix and item matrix via gradient descent. Then, by estimating values in the two lower dimension matrices, the values in the user-item matrix (including values that were initially null) are predicted by multiplying the two matrices together.

The following parameters are used and tuned to perform the matrix factorization process.

- Steps: maximum number of steps to perform the optimization was set to 5000
- Alpha: the learning rate was set to 0.0002
- Beta: the regularization parameter was set to 0.02
- K: the hidden latent features was set to 8
- Iterations: the number of iterations is set to 100,000

3.4 Training AutoRec

The AutoRec is one of the earliest models that applied autoencoders onto collaborative filtering for recommender systems. This model, which has the structure of a vanilla autoencoder, serves as a solid foundation and inspiration for later models that are developed.

In this model, there are m users, n items, and a partially filled user-item interaction matrix with dimension m x n [5]. The users and items vectors are represented by r_u and r_i respectively. There are two types of AutoRecs, namely the item-based AutoRec (I-AutoRec) and the user-based AutoRec (U-AutoRec). Regardless of the model variant, the structure in Figure 2.1 is copied *n* times, one copy for each user or item. The reconstruction equation for the input for this model is shown in Equation 3.1, where *f* and *g* are both activation functions.

$$h(r;\theta) = f(W \cdot g(Vr + \mu) + b) \tag{3.1}$$

To implement this model, an Encoder class is constructed to automate the creation of neural network layers using nn.Module.List. This Encoder class is then used as a template to build autoencoder networks. Once an instance of an Autoencoder is created using the Encoder class, it can be used to train using the training set and test using the testing set obtained during the train-test split. The tuned hyperparameters used to implement this model is shown in Table 3.1.

Table 3.1: Hyperparameters for AutoRec

Number of Hidden Units	500
Activation Function	Sigmoid
Learning Rate	0.001
Batch Size	512
Learning Rate	Decay every 50 epochs
L2 Regularizer Value	1
Optimizer Method	Adam
Random Seed	20

3.5 Training DeepRec

The DeepRec is a model created by Oleissi Kuchaiev and Boris Ginsburg. This model is inspired by the AutoRec, with some significant difference and characteristics such as the following [6].

- The network is much deeper than regular autoencoder networks
- The model uses scaled exponential linear units (SELUs)
- The dropout rate is high
- Iterative output re-feeding is used during training

During forward pass and inference pass, the model takes a user represented by his vector of ratings from the training set x, which is very sparse, while the output of the decoder f(x) is dense and contains rating predictions for all items in the corpus [6]. Thus, to explicitly enforce fixed-point constraint and perform dense training updates, the model improves on every optimization iteration with an iterative dense re-feeding step. Table 3.2 shows the hyperparameters used to tune this model.

Table 3.2: Hyperparameters for DeepRec

Architecture Layers	[512, 512, 1024, 512, 512]
Activation Function	SELU
Learning Rate	0.001
Batch Size	512
L2 Regularizer Value	0.001
Optimizer Method	Stochastic Gradient Descent
Momentum	0.9
Dropout Rate	0.8

3.6 Training the Restricted Boltzmann Machine

The whole user-item interaction matrix is a collection of Vs, where each V corresponds to each user's ratings. Because each user can have different missing values, each will have a unique RBM graph. In each RBM graph, the edges connect ratings and hidden features but do not appear between items of missing ratings [7]. W is treated as a set of edge potentials that are tied across all such RBM graphs.

In the training phase, RBM characterizes the relationship between the ratings and hidden features using conditional probabilities as shown in Equation 3.2 and 3.3.

$$p\left(v_j^k = 1 \middle| h\right) = \frac{\exp\left(b_j^k + \sum_{c=1}^{F} h_a W_j^k\right)}{\sum_{k=1}^{5} \exp\left(b_j^k + \sum_{c=1}^{F} h_a W_j^k\right)} \tag{3.2}$$

$$p(h = 1 \mid V) = \sigma\left(b_a + \sum_{j=1}^{d} \sum_{k=1}^{5} v_j^k \, W_j^k\right) \tag{3.3}$$

After obtaining the probabilities, compute the distribution of each hidden feature in h based on observed ratings V and the edge potentials W.

3.7 Training the Explainable Restricted Boltzmann Machine

Explanations for recommendations can have multiple benefits.

1. Effectiveness - helping users to make the right decisions based on explanations
2. Efficiency - help users to make faster decisions based on explanations
3. Transparency – reveal the underlying reasoning behind the recommendations given

For RBM, this model assigns a low-dimensional set of features to items in a latent space, which makes it difficult to interpret these learned features. Therefore, it is necessary to choose an interpretable technique with similar prediction accuracy as RBM.

For the Explainable RBM, the researchers constructed an RBM model for a collaborative filtering recommendation system that suggests items that are explainable while maintaining accuracy [8]. The model is limited to recommendations where no additional source of data is used in explanations, and where explanations for recommended items can be generated from the ratings given to these items, by the active user's neighbors only.

The interpretability of results for this model is based on the Explainability Score for item I and for user u, which is defined in Equation 3.4 [8].

$$ExpScore(u,i) = \frac{\sum_{x \in N_k(u)} r_{x,i}}{\sum_{N(u)} R_{max}} \tag{3.4}$$

4. RESULTS

4.1 AutoRec

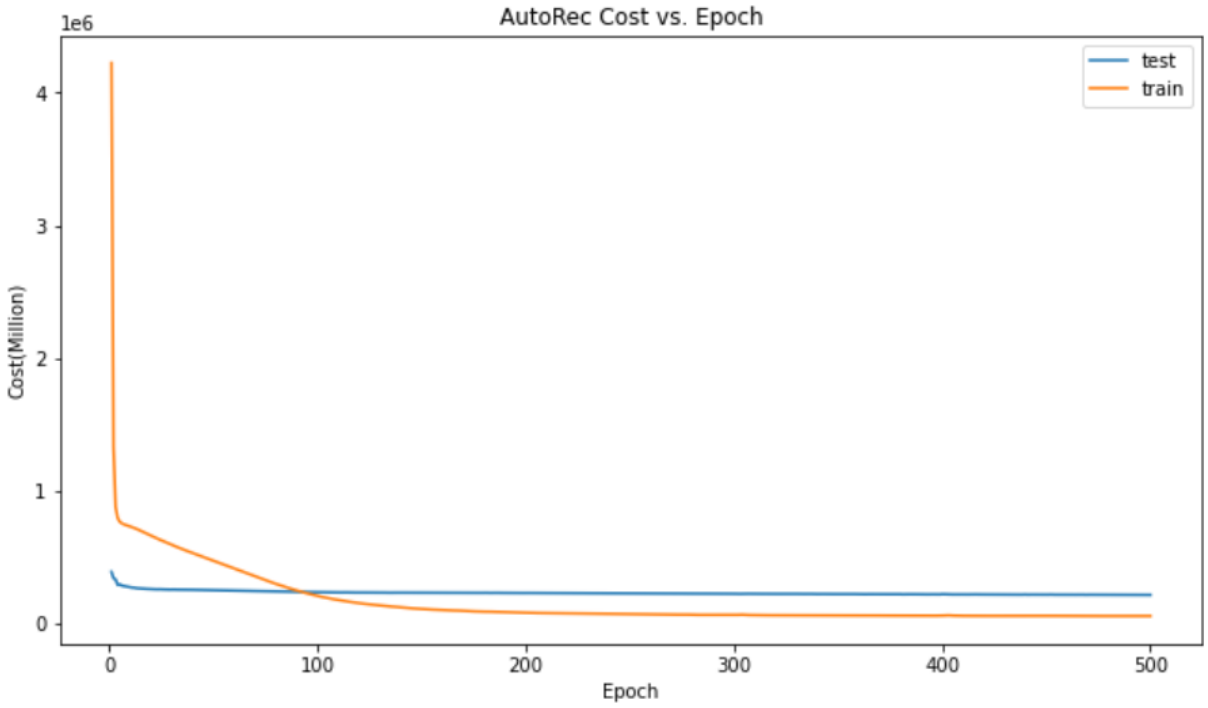

Figure 4.1 AutoRec Cost Function

One special aspect of the AutoRec model implemented is that there is a cost function associated with it. Figure 4.1 displays the exponential decrease in the cost function as the number of epochs increases. During the first epoch through the training data, the cost sits high at over 4 million. After training the model for 500 epochs, the test cost during the first few epochs are significantly lower when applied on test data. This shows that the AutoRec model was trained successfully to reduce the cost function.

4.2 Test RMSE

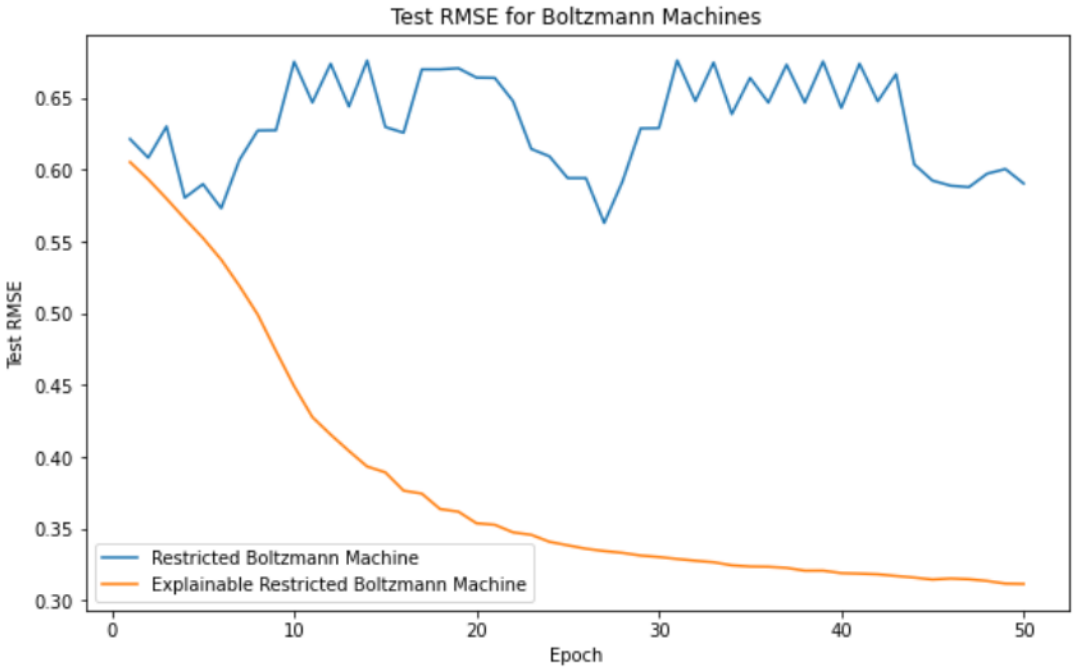

Figure 4.2 Test RMSE for Boltzmann Machines

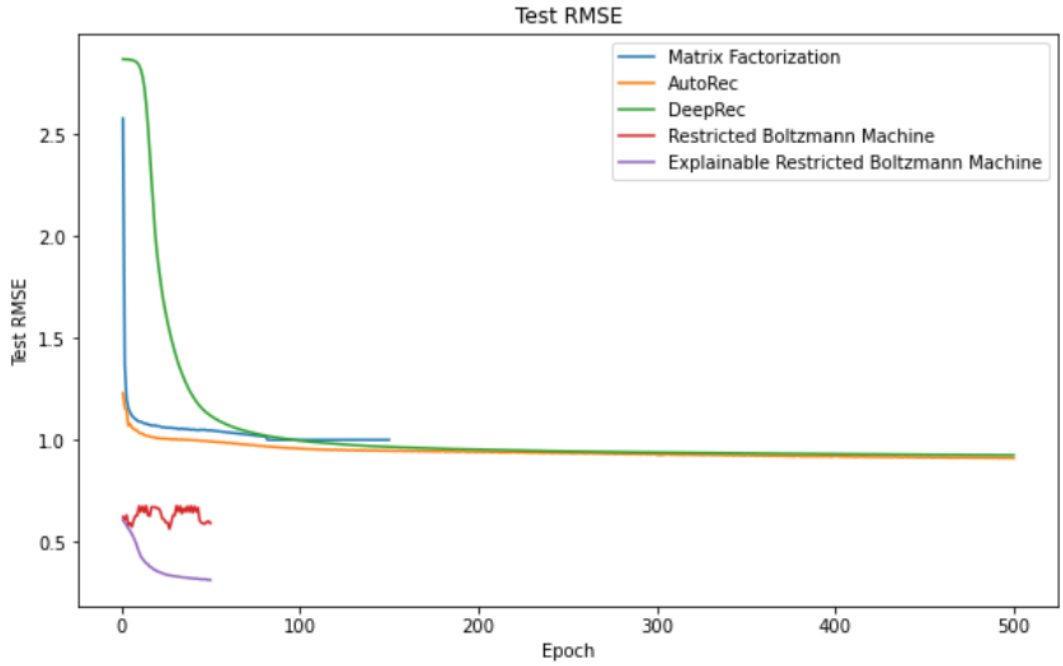

Figure 4.3 Test RMSE for all models implemented

All of the implemented models are compared using one metrics: the test RMSE. As shown in Figure 4.2, Matrix Factorization and DeepRec start off with test RMSEs that are over 2.5 before dropping down to a plateau at around 1.0. For AutoRec, its test RMSE begins at a smaller value of around 1.25 before stabilizing at a value close to the Matrix Factorization and DeepRec.

On the other end of the spectrum, the two models that have significantly lower test RMSEs from the very beginning are the two types of Boltzmann Machines: RBM and ERBM. As displayed in Figure 4.3, the test RMSEs for these two variants vary among themselves. While the test RMSE for RBM remains relatively stable with small spikes and dips, the RMSE for ERBM decreases exponentially to a low of 0.3. Table 4.1 shows the exact values obtained for each model's test RMSE at the last epoch.

Table 4.1: Test RMSE during last epoch

Model	**Number of Epochs**	**Test RMSE**
Matrix Factorization	150	0.9983
AutoRec	500	0.9101
DeepRec	500	0.9251
Restricted BM	50	0.5902
Explainable RBM	50	0.3116

The number of epochs to run for each model is determined by the trend of the test RMSE. If the test RMSE has leveled off, the process stops. Based on the test RMSE obtained as shown in Table 4.1, Restricted Boltzmann Machines have significantly lower test RMSE compared to

other models. The Autoencoder models produced RMSEs that are lower than that for Matrix Factorization model, but not as significant as the RBMs.

For datasets with only text and numbers, it appears that autoencoders do not have a significant advantage over matrix factorization. The strength of deep learning models such as autoencoders come from the fact that they are able to analyze heterogenous data that includes images, videos, and audio. With only text, the AutoRec and DeepRec are not remarkably outstanding compared to matrix factorization, especially with their more complex architecture and implementations.

Exact recommendations were not made in this research. However, if recommendations were made based these models, the Explainable RBM would be able to more accurately predict the ratings that users would give to movies and thereby recommending movies that they would more probably enjoy.

5. CONCLUSION

In this body of work, we have discussed the intuition and logic behind some of the most popular deep learning models that are used in recommender systems today. Deep learning models such as autoencoder and Boltzmann machine variants allow us to capture nonlinearity within data and are extremely efficient in learning from underlying latent features. Nevertheless, the disadvantages of using deep learning algorithms in recommender systems are also analyzed to shed light on the common problems faced by research scientists today.

Based on five models proposed by researchers, we implemented our own version of the models and hyperparameters. From the results obtained, we can observe that the two variants of Restricted Boltzmann machines performed significantly better than other models when evaluated using RMSE.

5.1 Future Work

While the results obtained show that the two variants Restricted Boltzmann Machines produced lower test RMSEs compared to the other models, this is not conclusive. The results obtained are only based on specific hyperparameters. If these hyperparameters are altered and models retrained, the results may different. Therefore, further research can be done to see how variants of these hyperparameters can affect model performance.

Additionally, the performance of the models for this study is only measured based on RMSE. However, there are many other metrics for recommender evaluation such as hit ratio, recall, precision etc. In future research endeavors, the model implementations can be evaluated using these metrics to provide a more holistic view of the models.

REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering 17, 6(2005), 734–749.

[2] Bing Bai, Yushun Fan, Wei Tan, and Jia Zhang. 2017. DLTSR: A deep learning framework for recommendation of long-tail web services. IEEE Transactions on Services Computing (2017), 1–1.

[3] Trapit Bansal, David Belanger, and Andrew McCallum. 2016. Ask the gru: Multi-task learning for deep text recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems. 107–114.

[4] Shuai Zhang, Lina Yao, and Yi Tay. 2017. Deep Learning based Recommender System: A Survey and New Perspectives. In ACM Computing Surveys

[5] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. In Computer, vol. 42, no.8, pp.30-37.

[6] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autoencoders Meet Collaborative Filtering. In Preceedings of the 24th International Conference on World Wide Web.

[7] Oleksii Kuchaiev and Boris Ginsburg. 2017. Training Deep Autoencoders for Collaborative Filtering. In arXiv.

[8] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann Machines for Collaborative Filtering. In Proceedings of the 24th international conference on Machine learning (ICML '07).

[9] Behnoush Abdollahi and Olfa Nasraoui. 2016. Explainable Restricted Boltzmann Machines for Collaborative Filtering. 2016 ICML Workshop on Human Interpretability in Machine Learning (WHI 2016).