Movie Recommender System using Autoencoders

Project report submitted in partial fulfillment
of the requirements for the degree of

Bachelor of Technology
in
Communication and Computer Engineering

by

Akhil Kashyap - 18UCC063
Himanshu Chahar - 18UCC101
Shivansh Jain - 18UCC169

Under Guidance of
Dr. Suvidha Tripathi

Department of Communication and Computer Engineering
The LNM Institute of Information Technology, Jaipur

April 2021

The LNM Institute of Information Technology
Jaipur, India

CERTIFICATE

This is to certify that the project entitled "Movie Recommender System using Autoencoders" , submitted by Akhil Kashyap (18UCC063), Himanshu Chahar (18UCC101) and Shivansh Jain (18UCC169) in partial fulfillment of the requirement of degree in Bachelor of Technology (B. Tech), is a bonafide record of work carried out by them at the Department of Communication and Computer Engineering, The LNM Institute of Information Technology, Jaipur, (Rajasthan) India, during the academic session 2020-2021 under my supervision and guidance and the same has not been submitted elsewhere for award of any other degree. In my/our opinion, this thesis is of standard required for the award of the degree of Bachelor of Technology (B. Tech).

Date

Adviser: Dr. Suvidha Tripathi

Dedicated to all the Doctors and Frontline workers giving their services tirelessly in the fight against the Covid-19 Pandemic

Acknowledgments

We would like to express our heartfelt gratitude towards our mentor and supervisor, **Dr. Suvidha Tripathi**, for giving us an opportunity to work on this project under her guidance. Her constant guidance, help and mentorship gave us a sense of direction and was very instrumental and helpful in putting together this project in an effective and insightful way.

The project gives us great insights about the working of Movie Recommender Systems and its implementation. It helped us in expanding our knowledge and understanding the concepts of Deep Learning and Autoencoders in the field of Recommender Systems.

Abstract

Movies, Web-series, and OTT Platforms are our go-to things nowadays. And since the pandemic, their usage and content, both have increased exponentially. Such enormous amount of Digital Data brings one major question to everyone's mind: What to Watch? Movie Recommender systems answers your question by personalising results based on your interests and tastes, thereby suggesting or recommending movies that you may like. These recommendations might be based on your watch history, Movie ratings, or based on the watch history of other users having a similar taste. In this project, we also try to implement a Movie Recommender System based on Deep Learning Tehniques, specifically Autoencoders. We read various research papers and articles based on the different types of Autoencoders used in the field of Recommender Systems, implemented them and compared them on the basis of their efficiency and accuracy.

Contents

Chapter *1*

Introduction

1.1 The Area of Work

Movie recommender Systems or Recommender systems in general are used to provide personalised results to the users based on various factors such that the user would find the results interesting and relevant according to his/her taste. Earlier, methods like nearest neighbours, clustering, matrix factorisation were popular. But they had a variety of problems like cold-start problem, or inability to capture complex user-item relations and output user-specific results.
But nowadays, Deep Learning methods are extensively used in recommender systems which not only cater to such shortcomings of the conventional systems but also improve their performance with high quality and accuracy of recommendations. Because of the representation of data in higher layers, Deep Learning based systems are able to capture the complex, non-linear user-item relations, thereby enabling more complex abstraction of the data. Deep Learning based systems capture textual, contextual, and visual information and provides increasing opportunities of improvement in terms of performance. Hence, our area of work revolves around 'Deep Learning based Recommender Systems'.

1.2 Problem Addressed

The issue of knowledge overload has grown dramatically nowadays due to the advancement of digital technologies and people's more and more reliance on the Internet. We are all drowned in a massive pool of data. As a result, there is an immense need to store, prioritize, and efficiently deliver best-suited information to the consumers. The Recommendation systems here play a vital role in solving such a problem. The main motive behind a recommender system is that it takes in vast amounts of data (be it fixed or dynamic), processes the same, and provides the users with a set of results that are actually needed and beneficial for that user. We are thus taking up this problem and trying to find the most reliable and accurate method of recommending movies. The project thus aims to implement a "Recommender Search Model using Autoencoders" that would take in MovieLens Dataset as input for the model and recommend movies to the users that remain significant to them.

1.3 Motivation & Objective

Projects are the best way to implement our knowledge and polish our skills. It gives us practical experience and also enhances our learning. Further, in our career goals, this project will reflect in our resumes and may even help us in getting jobs in companies with related fields of work. Apart from this, facing real life problems and trying to find their solution also motivates us to work on this project.
The main Objectives of this project are:

1. To learn about various architecture of Autoencoders needed to build an efficient and accurate model for recommender system
2. To choose such a configuration our model that it produces the best outcomes with the least amount of error.
3. Train the implemented model on the MovieLens Dataset
4. Test the model that is implemented and compare them on the basis of various criteria.
5. Predict movies for a user based on his preferences
6. Calculate the accuracy of the model

1.4 Existing Systems

1.4.1 Collaborative Filtering Recommender Systems

This is regarded as the most well-known approach. It assumes that "*people who agree on their taste in the past would also agree in future*".
The user's historical preferences are important in this method. The preferences of like minded people are taken into consideration for this method rather than the individual liking towards an item. The two primary areas of collaborative filtering are (i) neighborhood methods, and (ii) latent factor models.
Though, CF Recommender systems are context independent and are more accurate as compared to Content-based system, they face the problems of Sparsity, Scalability and Cold-start

1.4.2 Content-based Filtering

Recommendations given to the users by recommender systems based on Content-based filtering are based on his/her watch-history. Hence, it uses descriptive attributes of items such as genre, rating etc to produce recommendations. It can be done using the 'Tf-Idf Algorithm' (an algorithm that tells the importance of a keyword in the document by assigning hem weights based on the frequency of that word) and then we find the cosine similarity and give the respective recommendation.

1.4.3 Hybrid Model

Combining the strengths of both Collaborative and Content based filters, another form of filter known as Hybrid Model can be implemented. Hybrid recommender systems is a combination of Collaborative

and content-based filtering models to circumvent the shortcomings of each of the approaches and at the same time explores the benefits of both. Various Hybridisation techniques that have been proposed include Weighted, Switching, Mixed, Cascade, Feature combination and Feature augmentation.

Chapter *2*

Literature Review and Comparison

2.1 Collaborative Variational Autoencoder for Recommender Systems

$\Longrightarrow$ **by Xiaopeng Li (The Hong Kong University of Science and Technology) and James She (The Hong Kong University of Science and Technology)** [8]

Collaborative Variational Autoencoder (CVAE) is a Bayesian Generative Model. It is generally used for recommendation in multimedia scenarios. In comparison to the generic recommendation methods, CVAE outplays generic methods and shows better performance. Inclusion of content and rating for recommendation is one of the primary factors for the same.

In CVAE, a generative latent variable model, latent content variables are used for generation of content and latent item variables are used for generation of item ratings. Both content and collaborative information from Latent content and collaborative variables respectively are embedded into these latent item variables. This results in hybridisation of information alongside deep architecture.

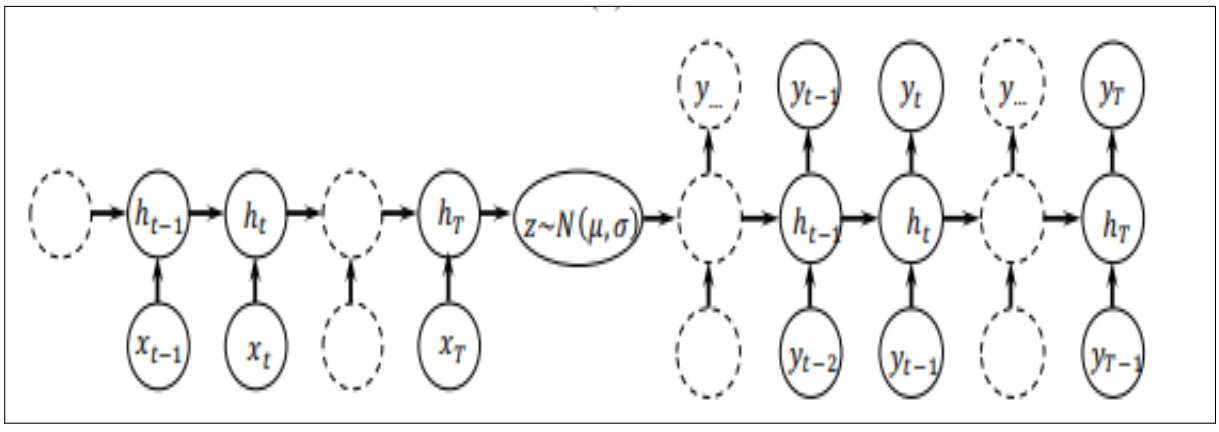


Figure 2.1 Extensions of inference and generation networks under the CVAE framework

CVAE models can also be interpreted with reference to denoising. The production of posterior variational distribution for content by the inference network also causes the latent content vector created to be corrupted with Gaussian noise. There is a need to manually tune other models, but in the case of CVAE the noise level is learned automatically by the inference network. This feature contributes to more stable and systematic learning of the model regardless what the dataset is.

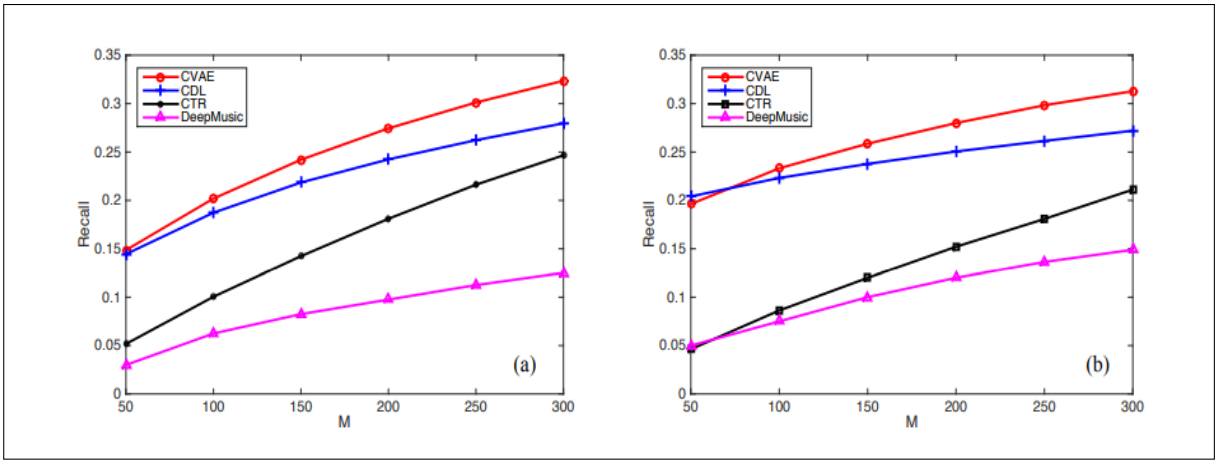


Figure 2.2 comparison of performance of CVAE, CDL, CTR and DeepMusic based on recall in the sparse setting on the two datasets: (a) citeulike-a and (b) citeulike-t

Fig shows the result comparison between CTR, DeepMusic, CDL and CVAE for the two given datasets used to train the models. It can be observed that CVAE produces better recommendations, specifically when M is large, despite the fact that CDL and CVAE are using deep learning models.

CVAE connects the different types of multimedia for recommender systems because of its stochastic distribution in latent space rather than observation space. Further, in comparison to the state-of-art recommendation methods, CVAE outplays state-of-art and shows better and robust performance.

2.2 Collaborative Denoising Auto-Encoders for Top-N Recommender Systems

$\Longrightarrow$ **by Yao Wu (Simon Fraser University, Burnaby, BC, Canada), Christopher DuBois (Dato Inc, Seattle, WA, USA), Alice X. Zheng (Dato Inc., Seattle, WA, USA) and Martin Ester (Simon Fraser University, Burnaby, BC, Canada)** [9]

The top-N results that are being recommended to the users are used to measure the performance of most real-world recommender services. The developments in top-N recommender systems have far-reaching practical implications. In this paper, we are introduced to a new approach called Collaborative Denoising Auto-Encoder (CDAE). The Encoder is based on the concept of Denoising Auto-Encoders. The experimental result is presented for various public datasets to show that CDAE outperforms the recommender system on different evaluation metrics.

CDAE is just like the traditional Denoising Auto-Encoder. We can define it as a neural network which has one hidden layer. But the major difference is that the feedback often encodes a latent vector for the recipient. This latent vector is what makes CDAE to be a much better model among other recommender models.

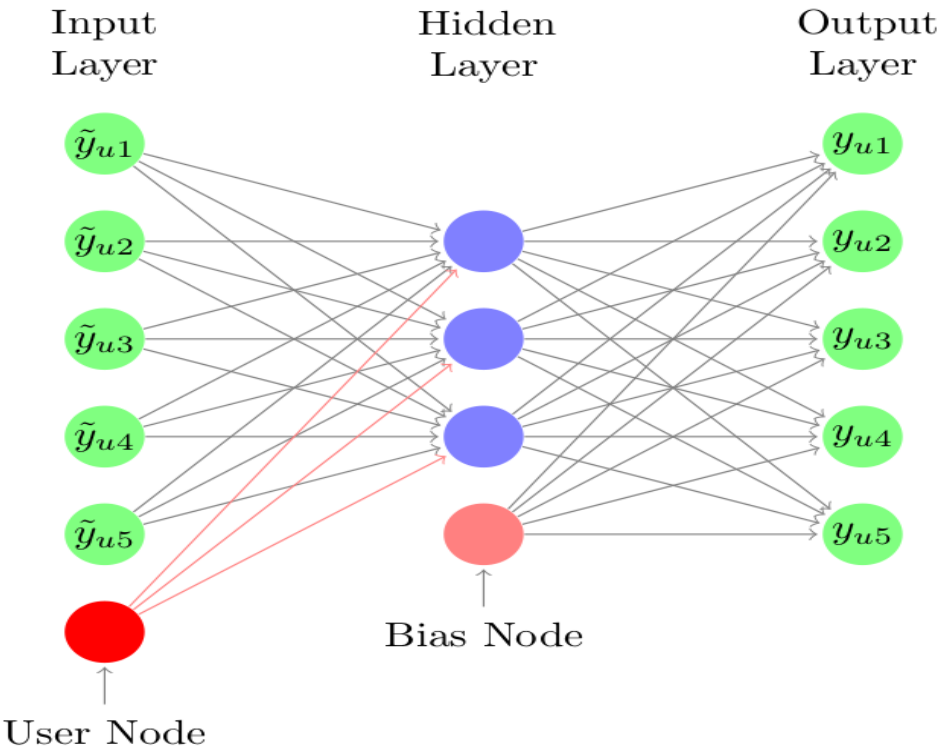


Figure 2.3 A CDAE example for an user named u. Weights are assigned to the connections between nodes. The ties in red are user-specific. Other weights are shared across all the users.

The three well-known datasets used in the experiment are : MovieLens 10M, Netflix, and Yelp Dataset. For each data set, the missed entries are discarded and the data with at least 4 ratings are used for testing and training.

The key components for the CDAE are the mapping feature forms, the corruption degree, and the failure function. Distinct combinations of the elements resulted in distinct versions of the CDAE model. The table below describes the choices of functions which we can have for each variant.

Table: Four Possible variants of the CDAE model

	Hidden Layer	Output Layer	Loss Function
M1	Identity	Identity	Square
M2	Identity	Sigmoid	Logistic
M3	Sigmoid	Identity	Square
M4	Sigmoid	Sigmoid	Logistic

Each of the four variants was then trained on various corruption levels from 0, 0.2, 0.4, 0.6, 0.8, 1.0. It was found that there is no best model for every dataset. The best CDAE model will highly depend on the dataset used. Thus, the components that are the mapping function, the objective function , and the loss function should be chosen depending on the dataset that we are using. If we consider in the results general, Model 4 produces better results relatively for all the three datasets. This proves that non-linear functions helps in increasing the model's representation ability and hence increasing the accuracy of the recommender system

2.3 A Survey of Recommender Systems Based on Deep Learning

$\Longrightarrow$ **by Ruihui Mu (College of Computer and Information, China)** [5]

This paper discusses the various topics of research in the field of deep learning based-recommender systems at length. The emergence of deep learning can be traced from artificial neural networks. The DL structure is nothing but a structure with multiple layers in which many layers are hidden.

Diving deeper into the topic, this paper then introduces and discusses in detail the Auto-encoders (AE), followed by the Restricted Boltzmann machine (RBM), Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN). Finally it describes a Deep Belief Network (DBN). These models are summarised in the table below:

TABLE 1. The summary of commonly used deep learning models.

Types of models	Deep learning models	Features
Unsupervised learning	AE	Encoder + decoder
		Dimensionality reduction
		Noise reduction
	RBM, DBN	Depth generation model (pre-training)
Supervised learning	CNN	Convolution transformation
		Pooling operation
		Processing grid data
	RNN	LSTM
		Bidirectional RNN
		Processing sequential data

Further ahead, authors introduce and discuss the features and characteristics of deep learning methods in content-based and context-aware recommender systems, deep learning-based collaborative filtering, deep learning-based hybrid recommender systems and recommender systems based on deep learning in social networks.

A brief explanation about the different models is as follows:

- **Deep learning methods in content-based recommender systems:** The main usage of such methods is in the apt capture of the non-linear and nontrivial user-item relationships and in the high layer representation of data which thereby enables the codification of more complex abstractions.
- **Deep learning methods in collaborative filtering recommender systems:** These methods make use of deep learning to train the model by making use of the explicit or implicit feedback information of the user.
- **Deep learning methods in hybrid recommender systems:** The objective of using this method is to build a unified framework that combines the user's/item's feature learning and the recommendation process, i.e., integrate the content-based methods with collaborative filtering (CF) recommendation methods.
- **Deep learning methods in social network-based recommender systems:** These systems are used mainly for recommendations based on social networks, which is used to model the impact of social relationships between users and thereby improve the quality of recommendation.
- **Deep learning methods in context-aware recommender systems:** The use of deep learning helps in the effective combining of context information into the recommendation system and its latent representation. These are also useful in complex frameworks.

2.4 Collaborative Filtering with Stacked Denoising AutoEncoders and Sparse Inputs

$\Longrightarrow$ **by Florian Strub (Univ-Lille, CRISTaL) and Jeremie Mary (Univ-Lille, CRISTaL)** [2]

The paper aimed at developing a training strategy which is used to implement collaborative filtering (CF) using Stacked Denoising AutoEncoders (SDAE). The input used is sparsh in nature. The paper also shows how the neural network provides excellent experimental results.

Use of masking noise provides us with two advantages:

- The masking of noise works as a strong regulazier for the dataset
- The AE is trained to predict missing values. Hence, to achieve this denoising aspect, the loss function of DAE is changed.

The datasets that are used in this paper to implement collaborative filtering are :-

- The Jester Joke dataset which has around 4.1mn ratings
- The MovieLens-1M dataset has around 1mn ratings (1 to 5) from 6000 users. The total number of movies is around 4000.

NOTE: In this paper there have defined two Encoders - Uencoder and Vencoder.

The input for the Uencoder is the sparse vector ui which, after computations, returns a dense vector uˆi as the output. On the other hand, the input for the Vencoder is the sparse vector vj which, after computations returns a dense vector vˆj as the output. While the first type of networks were used to learn the user representations, the latter were used to learn item representations.

The important inference obtained is in terms of the excellent performance of Autoencoders. But, turning RBM into an AE had some important overfitting issues.

Dataset	MovieLens-1M	Jester
SVD (gradient)	0.852 ± 0.003	4.117 ± 0.04
ALS-WR	0.850 ± 0.004	4.108 ± 0.02
NL-PMF	0.879 ± 0.008	n/a
U-autorec	0.874 ± 0.003	n/a
V-autorec	$\mathbf{0.831 \pm 0.003}$	n/a
Uencoders	0.861 ± 0.003	$\mathbf{4.107 \pm 0.03}$
Vencoders	0.840 ± 0.004	5.001 ± 0.10

Another important point to observe is the difference in the representation spaces of Uencoder and Vencoder and their results w.r.t dataset. While Vencoders performed brilliantly on the MovieLens dataset but showed a poor performance on the Jester dataset. In terms of errors, current research is insufficient to comment about.

Last but not least, we could see that the scores of 4-layer autoencoders are a cut above than the scores of 2-layer autoencoders.

2.5 Deep Learning based Recommender System: A Survey and New Perspectives

$\Longrightarrow$ **by Shuai Zhang (University of New South Wales), Lina Yao (University of New South Wales), Aixin Sun (Nanyang Technological University) and Yi Tay (Nanyang Technological University)** [7]

This survey starts with a discussion on the concepts and terminologies related to Deep learning and Recommender Systems. We then dive into the need for the usage of deep neural networks in recommender systems.

In this paper a survey has been conducted to clarify about the different architectural paradigms of deep learning e.g. Multilayer Perceptron (MLP), Autoencoders(AE), Convolutional Neural Network(CNN), Recurrent Neural Network(RNN), Restricted Boltzmann Machine(RBM), Neural Autoregressive Distribution Estimation (NADE), Adversarial Networks (AN), Deep Reinforcement Learning (DRL).

The main question that was addressed is that why we need deep learning models and the answer lies in the properties of neural architectures which includes:

- end-to-end differentiable
- produce relevant inductive biases that also cater to the type of input data. Deep Neural Networks come handy when the model can utilise the inherent framework, if present

Further, the author discusses the pros and cons of the use of deep learning techniques in recommendation purposes along with the some problems that need to be dealt with and bright future prospects.

Chapter *3*

Simulation and Results

3.1 Dataset Information

The dataset that we are working with is from MovieLens. It is regarded as one of the most popular datasets for creating a Recommender System on the internet. The dataset which we are using is **MovieLens 1M Dataset** which includes 1,000,209 anonymous reviews of approximately 3,900 movies which are submitted by 6,040 MovieLens users who entered the site in 2000.

3.1.1 Rating File

All the ratings are stored in the file **"ratings.dat"**. They are formatted as follows:
UserID :: MovieID :: Rating :: Timestamp

- The UserIDs range from 1 to 6040.
- The MovieIDs range from 1 to 3952.
- The Ratings are given on a five-star scale (whole-star ratings only)
- The timestamp is in seconds
- Each user has given at least 20 ratings.

3.1.2 User File

All the user information is stored in the file **"user.dat"**. They are formatted as follows:
UserID :: Gender :: Age :: Occupation :: Zip-code

3.1.3 Movies File

All movie informations are stored in the file **"movies.dat"** and are formatted as follows:
MovieID :: Title :: Genres

- Titles are equivalent to IMDB titles (including year of release)
- Certain MovieIDs may not correlate to a movie due to unintended duplicate entries

3.2 Methods Implemented

3.2.1 AutoEncoders

Autoencoders (AE) are an unsupervised learning technique in which we manipulate neural networks for the purpose of learning. The autoencoder is mostly used in the field of collaborative filtering. Collaborative filtering means that *"if a user A has the same taste as user B in Genre 1, then A is likely to have the same taste as B in a different Genre (say 2)"*.

Autoencoder is used to encode a set of input data which is basically used for dimensionality reduction.

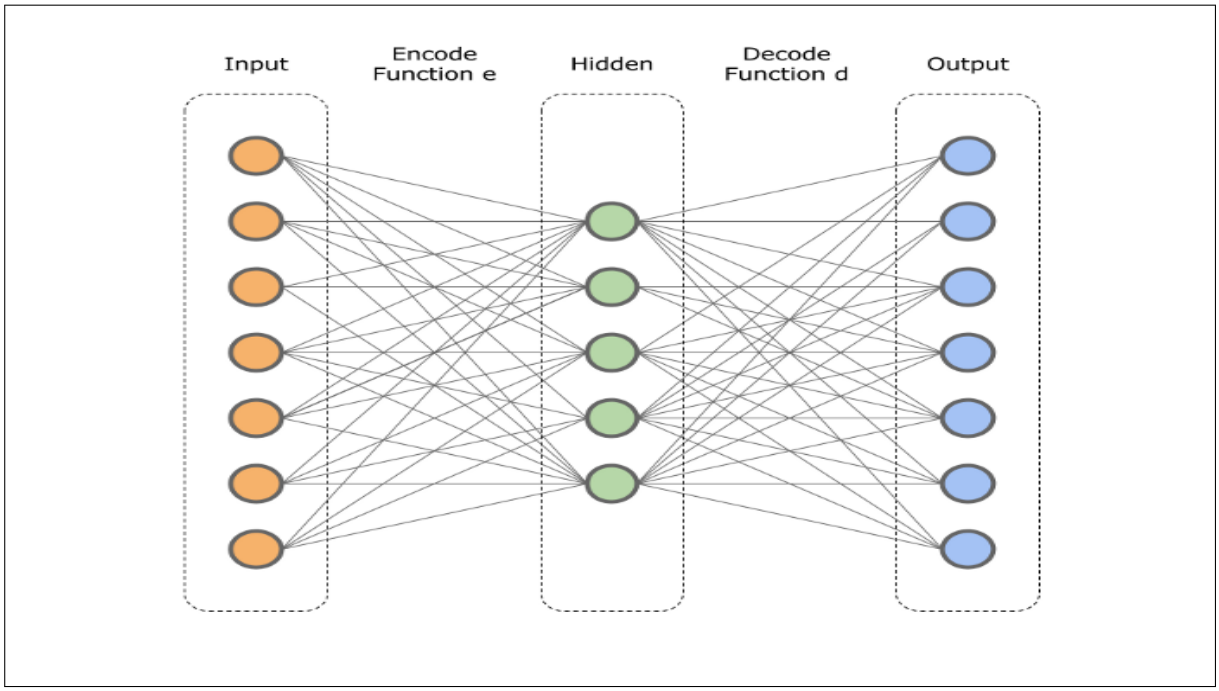


Figure 3.1 Autoencoder Architecture

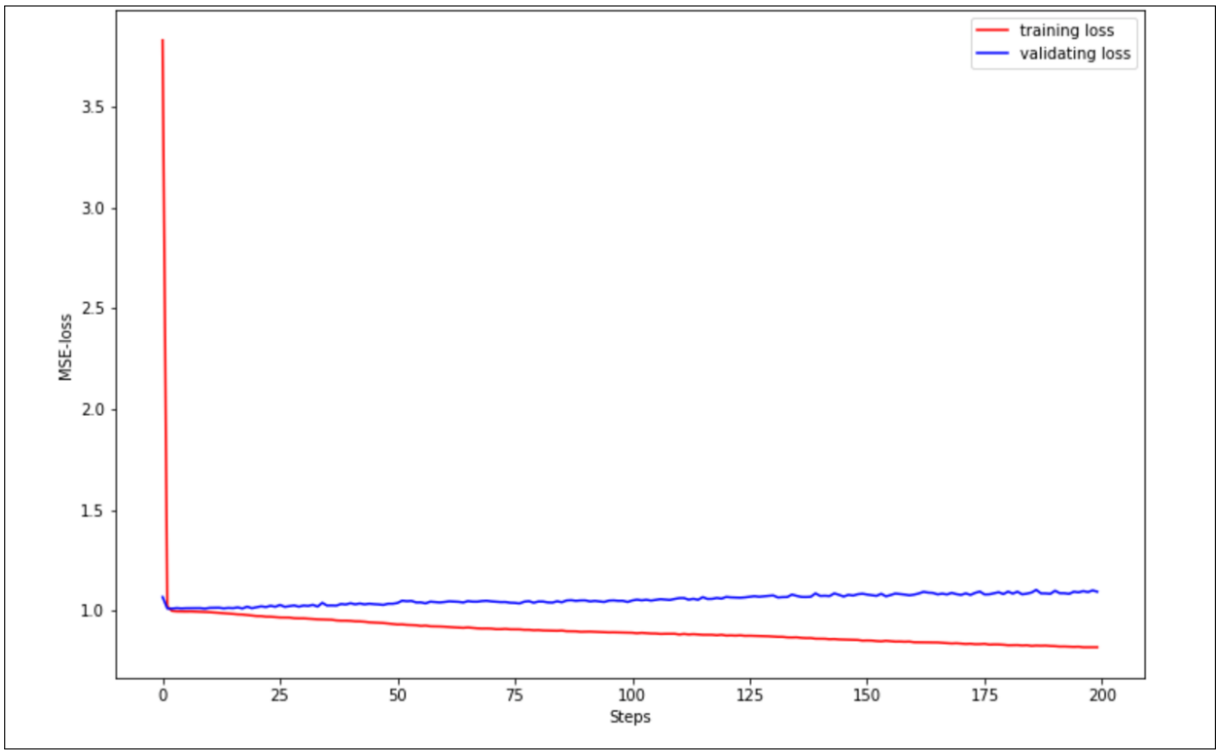
Three layers, namely the input layer, one hidden layer and the output layer(as shown in picture above) constitutes the architecture for an autoencoder model. Also AE is a form of unsupervised learning. The transition of input layer to the hidden layer is known as encoding step and while that from hidden layer to output is termed as decoding.

$$\phi : X \longrightarrow Z : x \mapsto \phi(x) = \sigma(Wx + b) := z$$
$$\varphi : Z \longrightarrow X : z \mapsto \varphi(z) = \sigma(\tilde{W}z + \tilde{b}) := x'$$

Figure 3.2 Mathematical functions of the mapping

3.2.1.1 Loss vs Epoch

Initially we can see that the loss for the model is on the higher side, as expected. It then begins to decrease considerably from the initial points itself. After a certain number of epochs the loss decreases gradually.



3.2.1.2 Test Loss

The overall average test loss for the implemented model for 200 epochs comes out to be : **1.0829**
The experimental result for the same is attached below.

```
====> Epoch: 188 Training Average loss: 0.8280, Validating Average loss: 1.0636
====> Epoch: 189 Training Average loss: 0.8256, Validating Average loss: 1.0647
====> Epoch: 190 Training Average loss: 0.8251, Validating Average loss: 1.0737
====> Epoch: 191 Training Average loss: 0.8243, Validating Average loss: 1.0573
====> Epoch: 192 Training Average loss: 0.8262, Validating Average loss: 1.0630
====> Epoch: 193 Training Average loss: 0.8249, Validating Average loss: 1.0835
====> Epoch: 194 Training Average loss: 0.8236, Validating Average loss: 1.0724
====> Epoch: 195 Training Average loss: 0.8216, Validating Average loss: 1.0608
====> Epoch: 196 Training Average loss: 0.8231, Validating Average loss: 1.0730
====> Epoch: 197 Training Average loss: 0.8221, Validating Average loss: 1.0660
====> Epoch: 198 Training Average loss: 0.8206, Validating Average loss: 1.0733
====> Epoch: 199 Training Average loss: 0.8204, Validating Average loss: 1.0814
====> Epoch: 200 Training Average loss: 0.8192, Validating Average loss: 1.0803

 ==================== Testing Auto Encoder on device: cuda:0 ====================
Average test loss: 1.0829
```

3.2.2 Stacked AutoEncoder

A stacked autoencoder (SAE) is a neural network model which is made up of many layers of sparse autoencoders. The output data from each hidden layer in turn acts as input for the next one succeeding it.

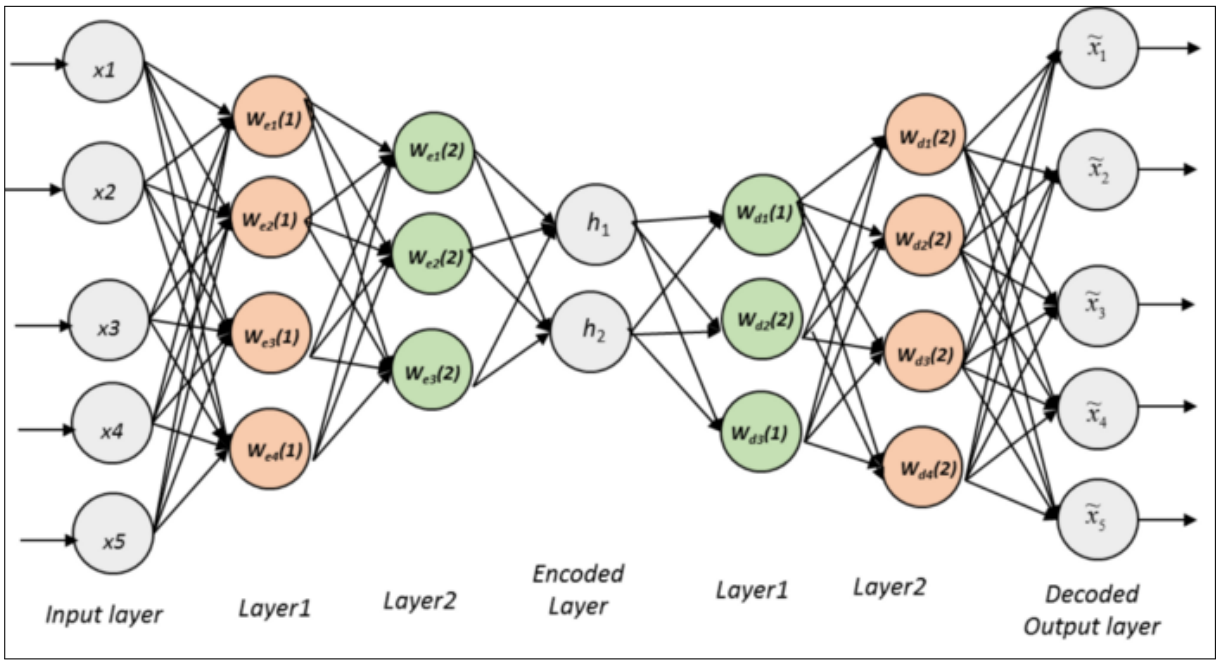


Figure 3.3 SAE Architecture

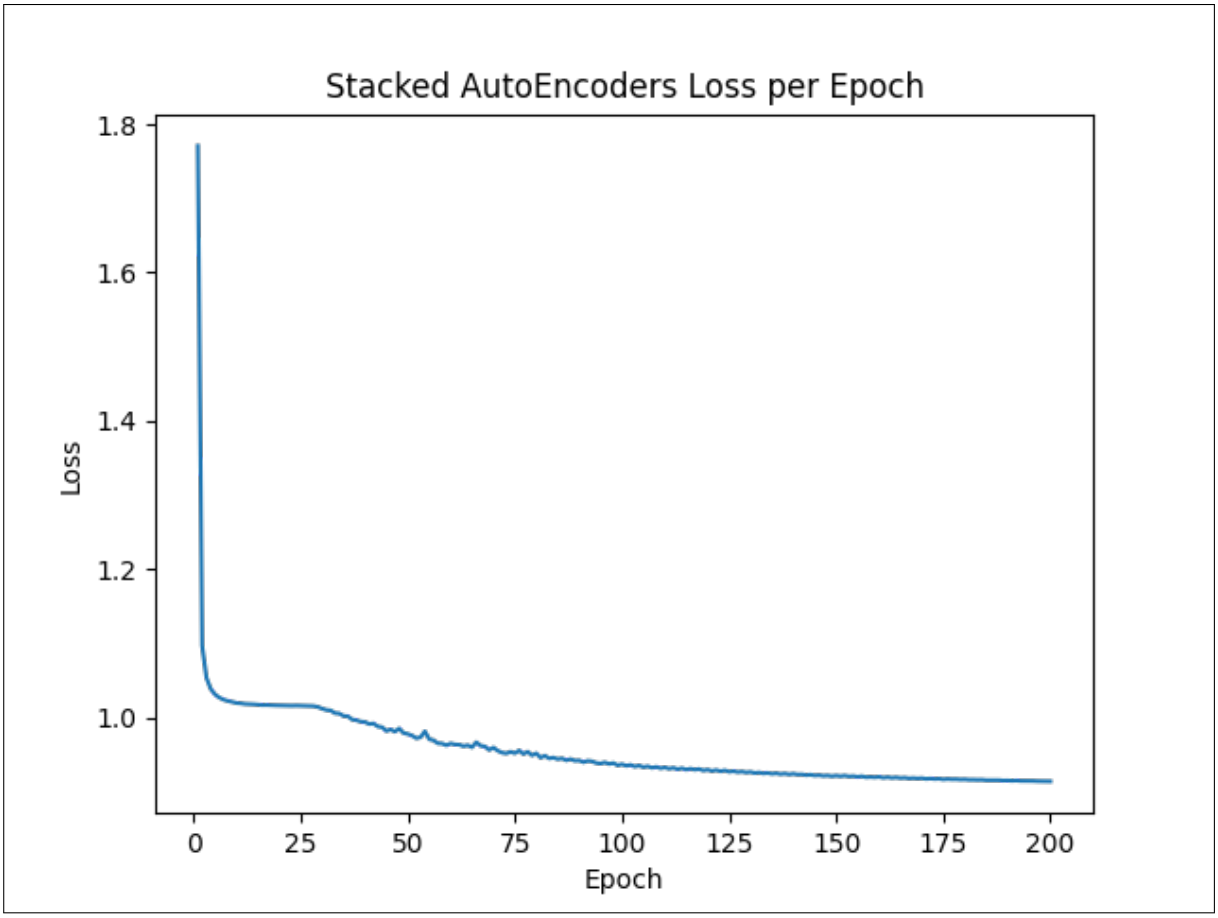
Using the unsupervised algorithm, first the hidden layers are trained, and then it is followed by tuning of layers by a supervised approach, as seen in Figure above. The stacked autoencoder is made up of three steps:

- Using input data, the autoencoder is trained and then the learned data is obtained from it.
- The learned data which we obtain as the output from the previous hidden layer, acts as input data for the next hidden layer, and the process is repeated until the training is complete for all layers
- When the training of all the hidden layers gets complete, we make use of a backpropagation algorithm which would help in minimizing the cost function, which is then followed by the updation of weights along with the training set so as to achieve fine tuning

We used this model to train on our MovieLens-1M dataset. The results which we obtained from the same are as follows.

3.2.2.1 Loss vs Epoch

Initially we can see that the loss for the model is on the higher side, as expected. It then starts decreasing considerably up until epoch 10. After a certain number of epochs the decrease in loss becomes gradual.



3.2.2.2 Test Loss

The overall test loss for the implemented model for 200 epochs comes out to be : **0.9456**
The experimental result for the same is attached below.

In [9]:
```
test_loss = 0
s = 0.
for id_user in range(nb_users):
    input = Variable(training_set[id_user]).unsqueeze(0)
    target = Variable(test_set[id_user]).unsqueeze(0)
    if torch.sum(target.data > 0) > 0:
        output = sae(input)
        target.require_grad = False
        output[target == 0] = 0
        loss = criterion(output, target)
        mean_corrector = nb_movies/float(torch.sum(target.data > 0) + 1e-10)
        test_loss += np.sqrt(loss.data*mean_corrector)
        s += 1.
print('test loss: '+str(test_loss/s))
```

```
test loss: tensor(0.9456)
```

3.2.3 Variational AutoEncoder

A variational autoencoder works on the principle where input is not mapped to a fixed vector, but is mapped to a distribution instead. In Variational AE, unlike the general autoencoder, the bottleneck vector is substituted by two separate vectors, which represents the distribution's mean and standard deviation respectively.

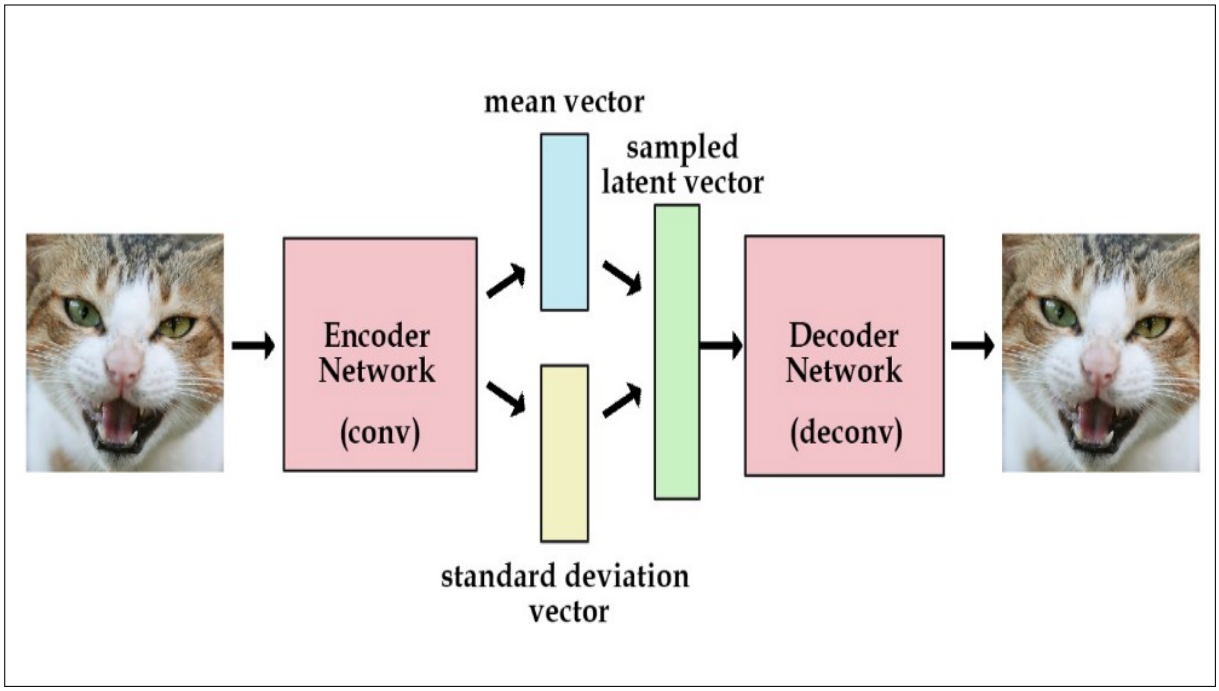
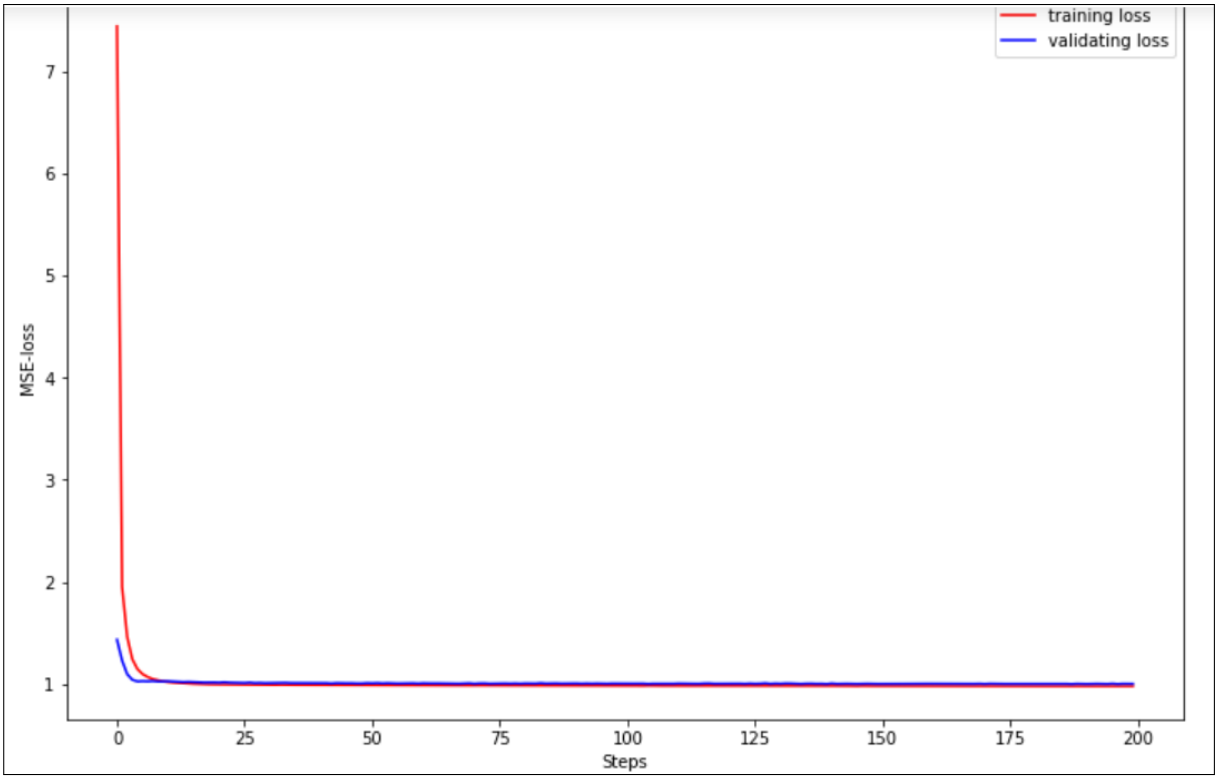


Figure 3.4 VAE Architecture

We can see from the above figure how the Variational encoder differs from a normal AutoEncoder. The variational AutoEncoder instead of having an 'n'-sized vector for encoding, it outputs a couple of vectors, each of size n: one for the distribution's mean, and other for standard deviation(SD). The two vectors act as parameters for a random variable vector of size "n". the i_{th} element of μ and σ denoting the mean and SD of the i_{th} random variable, X_i
We also used this model to train on our MovieLens-1M dataset. The results which we obtained from the same are as follows.

3.2.3.1 Loss vs Epoch

Initially we can see that the training loss for the model is on the higher side, as expected. The training loss then starts decreasing considerably up until epoch 5. After a certain number of epochs there is a gradual decrease in loss.



3.2.3.2 Test Loss

The overall test loss for the implemented model for 200 epochs comes out to be : **0.9927**
The experimental result for the same is attached herewith.

```
====> Epoch: 189 Training Average loss: 0.9767, Validating Average loss: 0.9857
====> Epoch: 190 Training Average loss: 0.9773, Validating Average loss: 0.9844
====> Epoch: 191 Training Average loss: 0.9768, Validating Average loss: 0.9848
====> Epoch: 192 Training Average loss: 0.9774, Validating Average loss: 0.9867
====> Epoch: 193 Training Average loss: 0.9772, Validating Average loss: 0.9872
====> Epoch: 194 Training Average loss: 0.9769, Validating Average loss: 0.9879
====> Epoch: 195 Training Average loss: 0.9774, Validating Average loss: 0.9851
====> Epoch: 196 Training Average loss: 0.9768, Validating Average loss: 0.9848
====> Epoch: 197 Training Average loss: 0.9773, Validating Average loss: 0.9865
====> Epoch: 198 Training Average loss: 0.9774, Validating Average loss: 0.9857
====> Epoch: 199 Training Average loss: 0.9770, Validating Average loss: 0.9860
====> Epoch: 200 Training Average loss: 0.9774, Validating Average loss: 0.9846

 ==================== Testing Variational Auto Encoder on device: cuda:0 =======
Average test loss: 0.9927
```

Chapter *4*

Conclusions and Future Work

4.1 Conclusion

The results of the implementation of all the three methods are summarised in the table below:

	AutoEncoder	**Stacked AutoEncoder**	**Variational AutoEncoder**
Test Loss	1.0829	0.9456	0.9927

$\Longrightarrow$ Lesser is the Average Test Loss, better is our model. From the above table we can see that **Stacked AutoEncoder performs the best out of the three models**.

Reason for Stacked AutoEncoder' better performance can be stated due to the process of 'Stacking' because stacking ensures that at every stage, our data is represented in lower/ higher dimensional space effectively. Here the input to each hidden layer is the output of the previous hidden layer.
In a nutshell, SAE trains the input data, acquires the learned data which is again given as input data for the next hidden layer. The process continues until all the hidden layers are trained. This training of hidden layers using unsupervised algorithms and then the tuning with the help of supervised algorithms helps SAE attain better accuracy.

Our first implemented model, Autoencoders perform the worst out of the three. An AutoEncoder creates an encoding of the original data from a higher to lower Dimensional Space(also known as Bottleneck) and then reconstructs it back from this bottleneck/encoded representation. The mean squared loss of the output and input thus obtained is used to train the autoencoder. Also, the validation score in this case is increasing which implies that the model is an overfitting Autoencoder. It performs bad at fitting the data that it doesn't validate.

Unlike general Autoencoders, Variational Autoencoders predict the mean and standard deviation of the hidden layers by mapping the input data to a probability distribution and not to a vector. The bottleneck vector in general AE is replaced by two distinct vectors denoting the mean and SD. The data is then sampled from the distribution by the autoencoders' decoding stage and original output is reconstructed.

Hence, for our Movie Recommender System, we will move forward with Stacked AutoEncoders.

4.2 Scope of further work

In the next semester, we will proceed with the Stacked AutoEncoder algorithm for our Movie Recommendation System. A detailed study about the particular algorithm will help us in understanding the

intricacies and learning the algorithm in depth. Post understanding and in-depth learning, we plan to bring various changes in the working of the algorithm in a way where we could improve further upon the accuracy and efficiency of the Movie Recommender System.

4.3 Code Link

Github: `https://github.com/Akhil-Kashyap/BTP-Movie-Recommender-System`

Bibliography

[1] Q. Bacuet. How variational autoencoders make classical recommender systems obsolete, April, 2019.

[2] J. M. Florian Strub. Collaborative filtering with stacked denoising autoencoders and sparse inputs. *Science*, 2016.

[3] V. K. Jonnalagadda. Sparse, stacked and variational autoencoder, December, 2018.

[4] J. Le. Recommendation system series part 6: The 6 variants of autoencoders for collaborative filtering, June, 2020.

[5] R. Mu. A survey of recommender systems based on deep learning. *Computer Science*, 2018.

[6] A. Oppermann. Deep autoencoders for collaborative filtering, April, 2018.

[7] A. S. Shuai Zhang, Lina Yao. Deep learning based recommender system: A survey and new perspectives. *Science*, 2019.

[8] J. S. Xiaopeng Li. Collaborative variational autoencoder for recommender systems. *Computer Science*, 2017.

[9] A. X. Z. M. E. Yao Wu, Christopher DuBoi. Collaborative denoising auto-encoders for top-n recommender systems. *Computer Science*, 2016.

[10] A. B. . C. K. Zeynep Batmaz, Ali Yurekli. A review on deep learning for recommender systems: challenges and remedies. *Computer Science*, 2018.