University of Colorado **Boulder**

INFO 5871: RECOMMENDER SYSTEMS

PREPARED FOR

PROF. ROBIN BURKE

RESTRICTED BOLTZMANN MACHINES FOR COLLABORATIVE FILTERING

Abigail J. Fernandes
Department of Computer Science
University of Colorado, Boulder
abigail.fernandes@colorado.edu

James Dykes
Department of Information Science
University of Colorado, Boulder
james.dykes@colorado.edu

May 6, 2019

TEAM ELEKTRA

1 Project Aim

Lying at the intersection of Deep Learning and Recommender Systems, the *Restricted Boltzmann Machines for Collaborative Filtering* [1] paper met our goal of trying out something new while achieving the same underlying objective of recommendation. A Restricted Boltzmann Machine (RBM) is different from other Recommendation algorithms [8], for instance say Matrix Factorization methods. It relies on energy rather than similarity or correlation [7]. It uses an undirected graphical model. This model was one of the many components of the winning recommender for the Netflix prize in 2009 [10].

Our main objective was to understand the working of the RBM. We would then translate that understanding to code. This would enable us to see the RBM in action. This paper was trained on the Netflix dataset[11]. We implemented the RBM using both, Binary and Gaussian hidden units.

This report goes on to describe our implementation, results and limitations in trying to achieve these goals.

2 RBM Methodology

2.1 Overview

An RBM consists of 2 layers.

1. **Visible layer:** This units in this layer consist of movie ratings by a user. Therefore the number of visible units differs depending on the movies rated.
2. **Hidden Layer:** This layers consists of hidden variables representing features that differ for a user. The number of hidden units stays the same irrespective of the user.

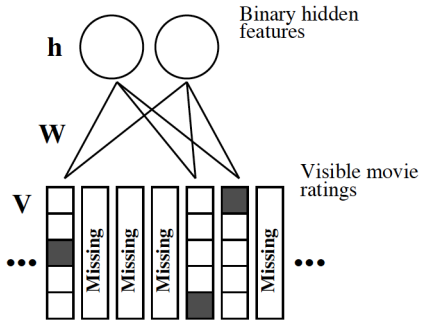
Each hidden unit is connected to all visible units. The *restricted* in RBM restricts hidden units from being connected to each other.

Ratings matrices tend to be sparse. To deal with this, we construct a different RBM for each user.

We are predicting one of K ratings. In this case, K can take on an integer value from 1-5. Figure 1 shows this configuration.

It should be noted that the weights and biases learned are shared between all the users.

Figure 1: RBM model



2.2 Dataset

In the paper, RBMs have been trained and tested on the Netflix data set, that contains over 100 million user/movie ratings.

2.3 Model

Our model is based on a single RBM for a single user.

Let V be a one-hot encoded matrix of user ratings, where $v_i^k = 1$ if movie i has rating k else 0.

$h =$ vector of hidden units

$b_i^k =$ the bias of rating k for movie i

$b_j =$ the bias of hidden unit j

We model each column of the observed visible rating matrix V using the softmax function below.

$$p(v_i^k = 1|\mathbf{h}) = \frac{exp(b_i^k + \sum_{i=1}^{F} h_j W_{ij}^k)}{\sum_{l=1}^{k} exp(b_i^l + \sum_{j=1}^{F} h_j W_{ij}^l)} \tag{1}$$

We model hidden features h using the logistic function below.

$$p(h_j = 1|\mathbf{V}) = \sigma(b_j + \sum_{i=1}^{m}\sum_{k=1}^{K} v_i^k W_{ij}^k) \tag{2}$$

Due to the interconnect structure of the RBM, these conditional probabilities are independent.

2.4 Learning

We train the RBM via a gradient ascent on the maximum likelihood:

$$\Delta W_{ij}^k = \epsilon \frac{\partial log\ p(\mathbf{V})}{\partial W_{ij}^k} = \epsilon\big(< v_i^k h_j >_{data} - < v_i^k h_j >_{model} \big) \tag{3}$$

where ϵ is the learning rate and <.> is the expectation of the model or data.

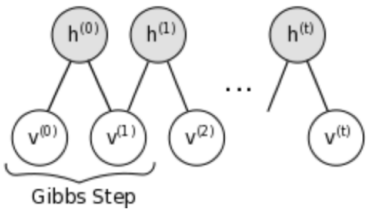
This expectation requires a lengthy calculation, so we approximate it using Contrastive Divergence as:

$$\Delta W_{ij}^k = \epsilon\big(< v_i^k h_j >_{data} - < v_i^k h_j >_T \big) \tag{4}$$

where $< . >_T$ is the distribution calculated from Gibbs Sampling on equations (1) and (2) initialized with user rating data.

2.5 Gibbs Sampling

Figure 2: Gibbs Sampling



Gibbs sampling simplifies the process of sampling from a multivariate distribution by allowing us to sample from the conditional distribution rather than the full distribution or marginal distribution. In our case, the conditional distributions of interest are independent. A single Gibbs step consists of the following:

1. Obtain the rating data for a user or group of users, **V**.
2. Calculate $\hat{p} = p(h_j = 1|\mathbf{V})$
3. Randomly sample parameters from this conditional distribution
4. Use these randomly sampled parameters to calculate $p(v_j = 1|\hat{p})$
5. Randomly sample from this conditional distribution. Use these samples to calculate V_{model}
6. Repeat steps 2-5 as many times as desired. On stopping, use the most recent samples of h and V as parameter estimates.

2.6 Prediction

We make prediction by calculating $p(v_q^k|\mathbf{V}=1)$, where $\mathbf{V}$ is the user profile of a user for whom, we wish to predict a rating for query movie q. This is computationally expensive. We approximate it with:

$$\hat{p_j} = p(h_j = 1|\mathbf{V}) = \sigma(b_j + \sum_{i=1}^{m}\sum_{k=1}^{K} v_i^k W_{ij}^k) \tag{5}$$

$$p(v_q^k = 1|\hat{p}) = \frac{exp(b_q^k + \sum_{j=1}^{F} \hat{p_j} W_{qj}^k)}{\sum_{l=1}^{K} exp(b_q^l + \sum_{j=1}^{F} \hat{p_j} W_{qj}^l)} \tag{6}$$

We use $argmax()$ on these probabilities to determine which rating is most probable.

3 Description of our methodology

We implemented RBM using the Tensorflow framework. The section below describes our methodology in doing so.

3.1 Data Processing

The first step was gathering the data. We used the MovieLens 1M dataset. In the attached code, you can find a Data Preprocessing Notebook that depicts the steps to convert .dat file to .csv which we read in, in our RBM notebook. We use Pandas for loading and processing data.

See table below for an overview of the 1M MovieLens dataset.

Table 1: Movie Lens Dataset

Movies	3883
Users	6040
Ratings	1000209

The next step would be creating the input ratings matrix for the user. The paper mentions that each user should be treated as a separate RBM and the movies rated by the user serve as the visible units. In order to create our input matrix for the RBM, we did the following:

1. We initialize our input matrix to have number of rows equal to the number of users in our dataset. The number of columns is equal to the number of movies in our dataset.
2. We first group the ratings data frame by users, so that we can get all the movies rated by a user to construct the visible units for a particular user.
3. For every user-movie pair, we add the rating as a one hot encoded vector of size 5. This gives us a matrix with shape
$$(users, movies, 5)$$
4. We reshape this matrix to be 2-D with dimensions
$$(users, movies * 5)$$
This serves as the input to the RBM. In order to ensure we use only the movies rated by a user in training, we use a mask to ignore those movies that have not been rated by a given user.

3.2 Model and Parameter Initialization

The table below describes the initialization of the parameters we are trying to learn.

Table 2: Parameter Initialization

Parameter	Description	Shape	Initialization
W	Weights connecting hidden and visible units	$(visible, hidden)$	Normal distribution with $\sigma = 0.01$ and $\mu = 0$
h_bias	Biases for hidden units	$(hidden, 1)$	0
v_bias	Biases for visible units	$(visible, 1)$	0

3.3 Learning Weights and Biases

1. Set v_0 to be the ratings for a batch of users
2. Update the state of hidden units using the logistic equation in Equation 1 to get h_0
3. We compute the positive gradient using $v_0 * h_0$
4. We reconstruct the visible units (v_1) and hidden units (h_1) using the Gibbs step mentioned in Section 2.5.
5. We compute the negative gradient using $v_1 * h_1$
6. We now update W with $(v_0 * h_0 - v_1 * h_1) * \alpha$ where $\alpha =$ learning rate. For the Gibbs sampling, we use the $ReLU$ function to randomly sample parameters from the conditional distribution.

We do this for a number of epochs until the network converges.

3.4 Training Parameters

The training parameters are depicted in the table below. The table contrasts the values used by us with that of the paper's.

Table 3: Hyperparameters

Hyper parameter	Paper	Our implementation
Epochs	50	50
Batch size	1000	100
Learning rate	0.01/batch-size with weight decay and momentum	0.05
Gibbs step	Varying from 1- 9 across training	Fixed at 3
Number of hidden units	100	100

It should be noted that the training data used by the paper has 100M observations while our training set consists of about 1M observed ratings. This explains the need for a larger batch size.

3.5 Evaluation

The original paper uses RMSE to evaluate the performance of the RBM in predicting new ratings. We use the same metric for evaluation.

1. During the training phase, we plot the RMSE after every epoch on the train data. The plot can be seen in Figure 3.
2. We then use the predict function to calculate the RMSE on unseen test data.

In order to account for the missing ratings, we use a mask. This is done so that we do not include those movies in our RMSE calculation.

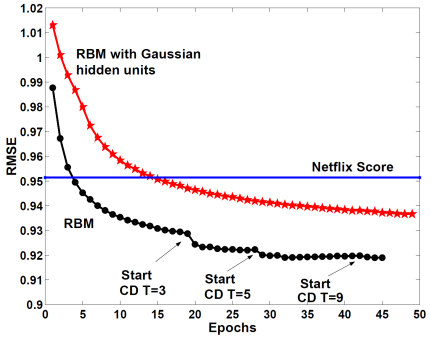
4 Results

We did a train test split on our dataset. The plots you see below depict the loss on the train set after every epoch. The training parameters we used are mentioned in Section 3.4.. The shape of our curves are similar to that of the original paper. We do see the expected pattern in training. Our network converges at a lower RMSE than the original, but as we have used a different dataset, and our RMSE calculation is not exact during training, we expect this kind of behaviour. After training, we predict the missing ratings on the test set using the learned weights and biases. The table below compares our results with the published results on the test set for both Binary as well as Gaussian Units.

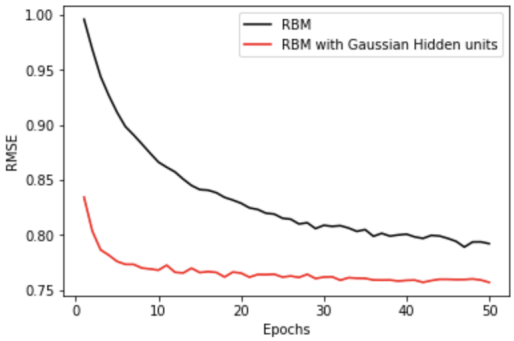
Table 4: RMSE on test predictions

Method	Published RMSE	Our RMSE
Binary Hidden Unit	0.92	1.073
Gaussian Hidden Unit	0.935	1.113

Figure 3: Plots of loss vs epochs



(a) Published training results



(b) Our training results

Our results show that, on average, our predicted ratings are off by 1. This is only slightly worse than that obtained in the paper. The paper cites the RMSE on the Netflix system (this is in 2007, so we have to be a bit cautious) as 0.9514. With additional tuning and more data, we believe we could achieve similar results.

Our predicted RMSE values show the same pattern as the published results with the binary hidden units having a lower RMSE than the Gaussian ones. We believe this is due to how we chose the parameters for the Gaussian Hidden units. The paper doesn't list any values for the Gaussian parameters. We arbitrarily chose a mean and a variance of 1. This causes the hidden unit bias to be *negative* under some circumstances, which is not desirable. We believe we need to either increase the mean to a large value of say 10 and keep the variance at 1 or leave the mean at 1 and decrease the variance to 0.1. Changing the variance to a value other than 1 requires significant changes to training and prediction calculations, therefore we have not attempted it for this project. We only recently realized that we could change the mean to a larger value and have not done so prior to writing this report.

4.1 Limitations

While we have tried our best to replicate the paper, there are some limitations that we will discuss below.

1. **Data:** The dataset we used differs from the Netflix dataset used in the paper. The sizes are different as are the way they are collected.
2. **Gaussian Hidden Units:** For Gaussian Hidden units, we have no idea about the values(mean and std) used in the original work. We need to do a more thorough analysis before drawing conclusions.
3. **Training RMSE:** In order to avoid including the unrated movies in the RMSE calculation, we add a factor $movies * users/rated\_movies$ to our error calculation. In this case we use the entire dataset. However since we calculate error on a batch, ideally we should get the values for the batch. The above factor is representative of the data in a batch and acts as a good approximation.
4. **Momentum and Weight Decay:** The paper tunes the learning rate with momentum and weight decay. We set our learning rate to be constant. However, even then, our network converges within a reasonable number of epochs.
5. **Gibbs Steps:** The paper increased the number of Gibbs steps as they progressed through the epochs, starting at 3 and ending at 9. We fixed our Gibbs steps at 3. Increasing the number of Gibbs steps should give us more accurate estimates of the parameters. This would likely make our optimum better, but requires considerably more run time for each iteration of the model.
6. **Unimplemented Extensions:** The paper mentions two other extensions, *Conditional RBM* and *Conditional Factored RBM*. Both of these require a significant amount of new work that was not possible in the stipulated time.

We have achieved a reasonable first implementation of the RBM. With more time, we could fine tune our implementation to achieve a lower RMSE.

5 Conclusion

This project has reinforced the distinction between understanding and implementing for us. We thought we understood the algorithm until we actually started implementing and testing it. Once we started implementing, we had to go back and really understand what we were doing by going through a variety of papers and other sources.

The Restricted Boltzmann Machine is a very different way to doing Collaborative Filtering. It begins with a simple idea, that relies on many approximations to make it work. These approximations such as Contrastive Divergence and Gibbs sampling are reasonably straightforward to implement on their own. When working together as they do in RBMs, it becomes a bit challenging to get them to work together in sync.

The two things that gave us the most difficulty were missing values and Gibbs sampling. We thought they worked several times, but with a lot of checking found that we were only partially right. Once completed, we were able to move forward with our work a lot more quickly.

We found the RBM to be very compelling, both in the way it works and in how fast it can run. Given the opportunity, we would both work more in this area with Abigail focusing on the Deep learning aspect and Jim focusing on Contrastive Divergence. Both areas are deeply interesting to us. We are both amazed by how much this project taught us, yet how little we know about RBMs even after a studying this particular implementation. We both want to know much more.

In our simple implementation, it already works well when compared to what Netflix was doing in 2007. With proper tuning, additional data, we believe that we could make our RBM perform much better. We believe this summarizes the project. We dug deeply, worked hard and learned a great deal, but we have only begun to learn.

References

[1] 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), Zoubin Ghahramani (Ed.). ACM, New York, NY, USA, 791-798. DOI=http://dx.doi.org/10.1145/1273496.1273596

[2] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Trans. Interact. Intell. Syst. 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

[3] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.

[4] Wikipedia contributors, 'Gibbs sampling', (2019, April 8). Retrieved, from https://en.wikipedia.org

[5] Ilker Yildirim. Bayesian Inference: Gibbs Sampling http://www.mit.edu/ ilkery/papers/GibbsSampling.pdf

[6] Geoffrey Hinton. 2010. A Practical Guide to Training Restricted Boltzmann Machines, Momentum 9 (2010): 1.

[7] Theano Development Team. DeepLearning 0.1 Documentation: Restricted Boltzmann Machines. Retrieved May 4, 2019 from http://deeplearning.net/tutorial/rbm.html

[8] Nikola Živković. 2018. Introduction to Restricted Boltzmann Machines. (October 2018). Retrieved May 4, 2019 from https://rubikscode.net/2018/10/01/introduction-to-restricted-boltzmann-machines/

[9] Oliver Woodford. Notes on Contrastive Divergence.
Retrieved May 4, 2019 from http://www.robots.ox.ac.uk/ ojw/files/NotesOnCD.pdf

[10] Koren, Yehuda. "The bellkor solution to the netflix grand prize." Netflix prize documentation 81, no. 2009 (2009): 1-10.

[11] Netflix Prize Data: https://www.kaggle.com/netflix-inc/netflix-prize-data

Appendix

1. **Original Paper:** https://www.cs.toronto.edu/ rsalakhu/papers/rbmcf.pdf
2. **Implementation:** https://github.com/AbigailFernandes/RBMRecommender
3. **Dataset:** https://grouplens.org/datasets/movielens/1m/