Friedrich-Schiller-Universität Jena
Physikalisch-Astronomische Fakultät

# Restricted Boltzmann Machines for Collaborative Filtering

REPORT

*for the lecture "Computational Physics III - Machine Learning"*

submitted by Markus Pawellek

Student Number: 144645
E-Mail Address: markuspawellek@gmail.com

Jena, February 11, 2019

# RESTRICTED BOLTZMANN MACHINES FOR COLLABORATIVE FILTERING

Markus Pawellek
markuspawellek@gmail.com

**Abstract**

*The restricted Boltzmann machine (RBM) is a network of stochastic units separated into two subsets, the visible and hidden units. It only allows undirected interactions between those two subsets. These basic requirements result in powerful properties of its mathematical structure. This leads to an incredible variety of use cases for RBMs in machine learning, mathematics and physics. Especially, for the topic of collaborative filtering it could be shown that RBMs are a state-of-the-art tool. In this report I give a short introduction to binary RBMs and their application to collaborative filtering problems using the example of predicting movie ratings made by users. Additionally, I write about some hints on how to implement such a system efficiently in C++. At the end there are suggestions for further development and investigation.*

## 1 Introduction

On October 2, 2006 Netflix announced the start of the so-called "Netflix Prize" competition. It was an open competition aimed to find the best algorithm for collaborative filtering to predict user ratings for movies based on previous ratings without any other information about the users or the movies. Netflix's current algorithm "Cinematch" introduced the threshold that had to be bested. For this Netflix provided a training dataset with over 100,000,000 ratings that more than 480,000 users gave about 17,000 movies. The complete competition lasted over three years and included two "Progress Prizes" in the years 2007 and 2008. Finally on September 18, 2009 Netflix announced the winner-team of the $\$\,1,000,000$ "Grand Prize" with its last submission 24 minutes before the conclusion of the contest. The solution of the winner-team "BellKor's Pragmatic Chaos" was based on the work of [14] which used restricted Boltzmann machines (RBMs) to efficiently predict the user ratings. With this algorithm the winner-team was able to achieve an improvement over Netflix's current algorithm by $10.05\,\%$. [11, 12, 14]

In [14] different RBMs were applied to the task of collaborative filtering for the first time. Even with the basic ideas presented in this paper one could achieve an error rate that was well over $6\,\%$ better than the score of Netflix's own system. Additionally, in comparison to other proposed solutions RBMs were able to deal much more efficiently with big datasets, like the ones given by Netflix for the competition. [14]

RBMs have found their application in other machine learning topics as well. In [10] an RBM is introduced in topic modelling as a more precise alternative to the common latent Dirichlet allocation. In [8] discriminative RBMs are used for classification in a self-contained framework and in [6] they are even used as basic building blocks in much bigger deep neural networks (DNNs) to efficiently reduce the dimensionality of some given data. According to [9] RBMs play an important role in the mathematical theory of machine learning and are not fully investigated, yet.

RBMs in general consists of a basic and simple structure with good mathematical properties for which one can find efficient learning algorithms. The successful application of RBMs in these different topics of machine learning make them an interesting subject to study and understand. In the next sections we will talk about the details of RBMs used for the topic of collaborative filtering and especially about how to predict user ratings for movies as a direct application to a real world problem.

## 2 The Problem

Collaborative filtering as seen in a modern narrow sense basically can be described as a technique to make automatic predictions about interests of users by collecting

Table 1: The table shows examples of binary ratings for movies made by some imaginary users. Every row represents a user and every column a movie. The number 0 is used to point out that the user does not like the movie and 1 for the opposite. The symbol $\times$ is used if there was no rating for the movie by this user.

| | Star Trek | The Matrix | Van Helsing | Harry Potter | The Hobbit |
|---|---|---|---|---|---|
| James T. Kirk | 1 | 1 | $\times$ | 0 | $\times$ |
| Trinity | $\times$ | 1 | 0 | 1 | 1 |
| Anna Valerious | $\times$ | $\times$ | 1 | $\times$ | 0 |
| Severus Snape | 0 | 1 | 0 | 1 | 0 |
| Thorin Oakenshield | 1 | 1 | 1 | $\times$ | 0 |

the preferences or tastes of many users. One often refers to predicting as "filtering" and to collecting as "collaborating". The underlying assumption of the collaborative filtering approach is that if two persons have the same opinion on one issue then it is likely that they will also have the same opinion on another issue. [16]

For convenience table 1 shows examples for movie ratings made by some imaginary users. The entries with 0 and 1 determine if a user likes a movie or not and are already known to us. They shall be used to predict the unknown values marked with $\times$. In the example the users "James T. Kirk" and "Thorin Oakenshield" both like the movies "Star Trek" and "The Matrix". So one could assume that both users have similar preferences and therefore the user "James T. Kirk" would also like the movie "Van Helsing" because the same is already true for the user "Thorin Oakenshield". Vice versa one may say that the user "Thorin Oakenshield" does not like the movie "Harry Potter" since "James T. Kirk" provided a negative rating for that movie.

Here the collaborative filtering problem of predicting user ratings for movies shall be solved by using RBMs as it was done in [14]. For that we have to achieve three main goals that together are forming a basic outline of understanding RBMs and their application to collaborative filtering.

**Model:** Approximately represent probability distributions over different user-movie-ratings.

**Learn:** Learn an optimal probability distribution in this representation based on some given samples.

**Infer:** Make predictions for unrated movies.

# 3 The Model

## 3.1 Basic Idea

To understand the model of an RBM let us first consider the basic idea by looking at the left part of figure 1. It shows a simple schematic example of an RBM. At first sight there seems to be no real difference to a standard feed-forward neural network (FFNN) with two layers. But the subtle difference lies in the fact that every edge connecting different units in an RBM has to be undirected and not directed as in a typical FFNN. Therefore the influence of one unit to another cannot be computed directly through a feed-forward pass. [10]

Apart from this there are two obvious properties which are defining the RBM. First, the units in the RBM are separated into two subsets, the hidden units and the visible units. Second, connections between units are only allowed between those two subsets. This makes it possible to equivalently define an RBM to be an undirected bipartite graph. [9, 10]

These properties give us no real obvious interpretation for the values of these units. But we have to remember that we want to model a probability distribution over ratings. Based on this mathematically we will try to interpret every unit as a binary random variable. Later, this will enable us to sample from the probability distribution represented by the RBM and predicting the outcome. Interpreting user ratings to be realizations of the visible random variables we can then see that every visible value in an RBM stands for a movie rating made by some user. The realizations of the hidden random variables can be thought of as non-observable features the RBM has learned implicitly. [9, 10, 14]

Looking at the right part of figure 1 one sees an example of these ideas for an imaginary user. Every

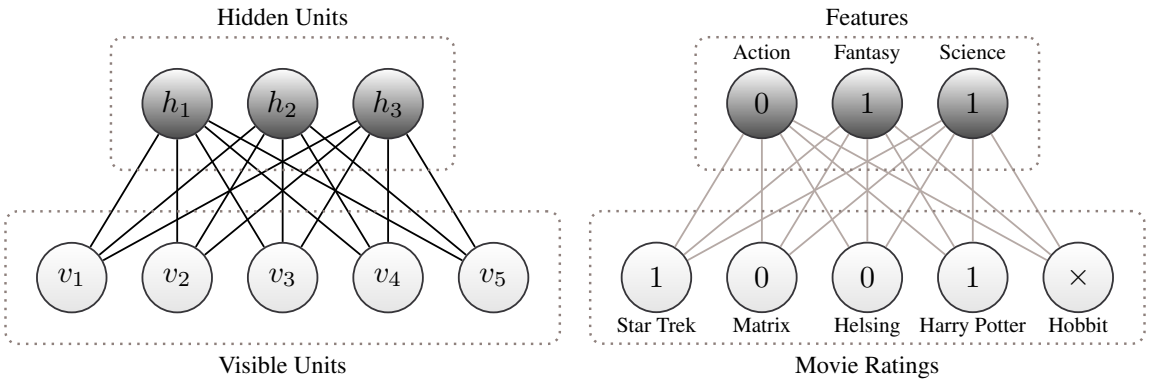


Figure 1: The left part of the figure shows a schematic example of an RBM. The nodes are separated into visible and hidden units and connections are only allowed between those two subsets. The right part shows an example application to the collaborative filtering problem of predicting movie ratings. For one user the visible values are the movie ratings and the hidden values are some features, like movie genres, implicitly learned by the RBM.

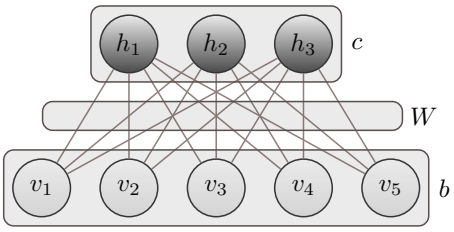


Figure 2: The figure shows the basic scheme of an RBM with the weight matrix $W$ and bias vectors $b$ and $c$ as parameters describing the probability distribution modelled by the RBM.

rating from one user can be seen as a visible value. Based on these visible values an RBM can assign some hidden values based on some learned features, like movie genres "Action" or "Fantasy". These hidden values for example would describe if the user likes the movie genre. [13]

Until now we have described how to model the collaborative filtering problem by an RBM but not how to model the RBM itself. Hence, we first have to define some parameters to be able to represent a set of probability distributions which can be learned. The procedure for an RBM is straightforward. We choose the standard approach of introducing biases for every unit and weights for every edge. Therefore we get two bias vectors for the hidden and visible units and one weight matrix for all connections. Figure 2 shows this schematically. [7, 9, 10, 14]

### 3.2 Mathematical Details

For the sake of simplicity, we will only assume so-called binary RBMs with binary hidden and visible values $\mathcal{B} := \{0, 1\}$ and will further on describe them only as RBMs. These RBMs are fundamental to every generalization and are not that much of a constraint. According to [14] and [10] binary RBMs can be easily extended to categorical RBMs or Gaussian RBMs. But it turns out that it is important for the hidden values to stay binary to introduce a sort of information bottleneck which strongly regularizes the mathematical properties of an RBM. [7, 9, 10, 14]

**DEFINITION:** (RBM)

*For $n, m \in \mathbb{N}$ an RBM $\vartheta$ is given by a tuple $(W, b, c)$ consisting of its weight matrix $W$, visible bias vector $b$ and hidden bias vector $c$.*

$$\vartheta := (W, b, c) \in \mathbb{R}^{(n \times m)+n+m}$$

*We call $n$ the number of visible units and $m$ the number of hidden units of $\vartheta$.*

As seen in figure 2 the mathematical definition of an RBM as data structure is straightforward. We want to keep such an approach for the parameterization of the probability distribution as well. Therefore we will first define the influence of every unit with respect to its connections by introducing an energy term. In this energy term every weight and bias linearly connects visible and hidden values. To model non-trivial probability distributions we need some simple non-linear function with good properties to be applied to this energy term. Afterwards, we have to make sure the resulting function is normalized such that it satisfies the definition of a probability distribution. [7, 10, 14]

**DEFINITION:** (RBM Probability Distribution)

*Let $n, m \in \mathbb{N}$ and $\vartheta$ be an RBM with $n$ visible and $m$ hidden units. The probability distribution of $\vartheta$ is then given by $p[\vartheta]$.*

$$p[\vartheta] \colon \mathcal{B}^n \times \mathcal{B}^m \to [0,1]$$

$$p[\vartheta](v,h) \coloneqq \frac{1}{Z(\vartheta)} e^{-E[\vartheta](v,h)}$$

*Here, the energy term $E[\vartheta](v,h)$ is given by the following expression.*

$$E[\vartheta](v,h) \coloneqq -v^{\mathrm{T}} W h - v^{\mathrm{T}} b - h^{\mathrm{T}} c$$

*$Z(\vartheta)$ is the normalization factor of $p[\vartheta]$.*

$$Z(\vartheta) \coloneqq \sum_{v \in \mathcal{B}^n} \sum_{h \in \mathcal{B}^m} e^{-E[\vartheta](v,h)}$$

At this point one could think, that we have introduced a major design problem in our RBM model. If we cannot observe hidden values then how should we be able to use a probability distribution with hidden values as its arguments. We can omit this problem by inserting a layer of indirection. We will simply use only the probability distribution for the visible values. For convenience here we use function overloading.

$$p[\vartheta] \colon \mathcal{B}^n \to [0,1]\,, \qquad p[\vartheta](v) \coloneqq \sum_{h \in \mathcal{B}^m} p[\vartheta](v,h)$$

This finishes the complete mathematical description of the RBM model. But because of the main properties of an RBM it seems appropriate to deduce a proposition for the posterior probability to show one big advantage of an RBM. Due to its property that connections are only allowed between hidden and visible values we know that under a given visible value vector the coordinates of the hidden value vector have to be independent and vice versa. So for all $v \in \mathcal{B}^n$ and $h \in \mathcal{B}^m$ one obtains the following. [7, 10]

$$\begin{aligned} p[\vartheta](h|v) &= \prod_{j=1}^{m} p[\vartheta]\left(h_j = 1 | v\right) \\ &= \prod_{j=1}^{m} \operatorname{sigm}\left(c_j + \sum_{i=1}^{n} v_i W_{ij}\right) \\ p[\vartheta](v|h) &= \prod_{i=1}^{n} p[\vartheta]\left(v_i = 1 | h\right) \\ &= \prod_{i=1}^{n} \operatorname{sigm}\left(b_i + \sum_{j=1}^{m} W_{ij} h_j\right) \end{aligned}$$

The logistic sigmoid function $\operatorname{sigm}$ is given by the typical definition.

$$\operatorname{sigm} \colon \mathbb{R} \to (0,1)\,, \qquad \operatorname{sigm}(x) \coloneqq \frac{1}{1+e^{-x}}$$

# 4 Learning

In this section, we discuss how to estimate the parameters of an RBM, using gradient-based optimizers. Generally, RBMs have many parameters and need to be trained on very large datasets. Hence, the standard approach taken to learn an RBM is stochastic gradient ascent with the use of mini-batches. [7, 10, 14]

## 4.1 Objective Function

Learning is always about optimizing some objective function. In a typical FFNN for example one can use cross-entropy as a loss function which then has to be minimized to learn the parameters of the network. But in our current mathematical setting we want to learn a probability distribution over the visible values of an RBM. For this we have to estimate the parameters based on some given samples In statistics the standard method for estimating the parameters of a statistical model is called maximum likelihood estimation (MLE). [10]

Let $n \in \mathbb{N}$ be the number of visible and $m \in \mathbb{N}$ be the number of hidden units for all RBMs in our statistical model. Further, let $s \in \mathbb{N}$ be the sample count and define $\mathcal{S} \in \mathcal{B}^{n \times s}$ to be the vector of all samples which are distributed identically and independently. Then MLE tries to maximize the probability of the given samples. Because of their independence this results in maximizing the product of their probabilities. To simplify the analysis further one then typically takes the logarithm of this expression and constructs the log-likelihood function such that every product will be replaced by a sum.

$$\mathcal{L}[\mathcal{S}] \colon \mathbb{R}^{n \times m + n + m} \to \mathbb{R}$$

$$\mathcal{L}[\mathcal{S}](\vartheta) \coloneqq \frac{1}{s} \sum_{k=1}^{s} \ln p[\vartheta]\left(\mathcal{S}_k\right)$$

Due to the monotonic behavior of the logarithm maximizing the log-likelihood function is equivalent to maximizing the maximum-likelihood function. [10]

## 4.2 Gradient Computation

Because we want to use gradient-based optimizers we now have to compute the derivative of the log-likelihood

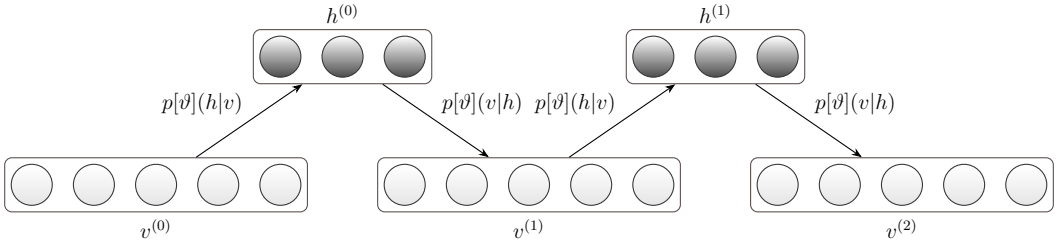


Figure 3: The figure shows the basic scheme of Gibbs sampling.

function with respect to the weight matrix and bias vectors of the RBMs in our statistical model. For the sake of simplicity we will only mention the derivative with respect to the weight matrix.

$$\nabla_W \mathcal{L}[\mathcal{S}](\vartheta) = \frac{1}{s}\sum_{k=1}^{s} \mathbb{E}_\vartheta\left[\mathcal{V}\mathcal{H}^{\mathrm{T}}\middle|\mathcal{S}_k\right] - \mathbb{E}_\vartheta\left[\mathcal{V}\mathcal{H}^{\mathrm{T}}\right]$$

At first sight this formula seems to be complicated. But the left part can be easily computed due to the good properties of the posterior probability. The right part is much more difficult. The typical method of finding such an expectation value of the model itself one has to do Gibbs sampling. Figure 3 demonstrates this method schematically. [10]

Deriving and explaining the complete procedure of Gibbs sampling would go beyond the scope of this report. But Gibbs sampling is generally used to sample from a probability distribution. Because an RBM represents such a distribution and we want to estimate a specific expectation value of the model it seems to be the natural choice. By using the posterior probability together with visible values we can sample hidden values due to the good properties of an RBM. After obtaining some sampled hidden values we do the same thing to generate new visible values. In figure 3 it becomes clear that we have to repeat this process. The main problem lies in the fact that this process takes a long time until it reaches equilibrium and therefore aborting the process too early will result in a bad estimation of the expectation value. On the other hand we cannot afford such a computational overhead in the learning process. [10]

### 4.3 Contrastive Divergence

The idea is now to make rough approximations to the expectation value and to speed up the computation. This faster method is known as contrastive divergence (CD). To make things good again this algorithm aborts the series after some given number $k \in \mathbb{N}$ and then approximates the expectation value by the following expression. Therefore the algorithm is sometimes called $\mathrm{CD}_k$. [7, 10]

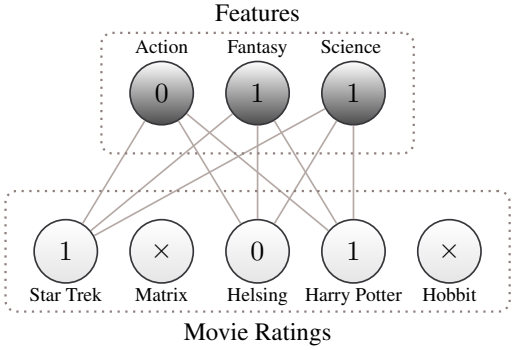


Figure 4: The figure shows the application of the learning algorithm to the movie ratings for some user.

$$\mathbb{E}_\vartheta\left[\mathcal{V}\mathcal{H}^{\mathrm{T}}\right] \approx v^{(k)} {h^{(k)}}^{\mathrm{T}}$$

Looking at this approximation the learning procedure seems to be unstable and may even diverge in some cases. But it is a matter of fact that in reality this algorithm is working efficiently. This may be due to the stochastic gradient ascent algorithm which introduces noise in the computation. Therefore errors done by CD would average out over multiple iterations of the learning process. In [7] this is implicitly confirmed because the algorithm shows a much higher performance with a small mini-batch size. [7, 10]

### 4.4 Application to Collaborative Filtering

Now one has to talk about the application of the algorithm to collaborative filtering. The first problem that arises is that in general every user has rated different movies. Hence, we cannot use the same visible units for every user. As a solution [14] proposed to create an RBM for every user with a constant number of hidden units. Then every RBM has one trainings sample, the movie ratings of that user. To not have a set of completely independent RBMs we tie the weights and

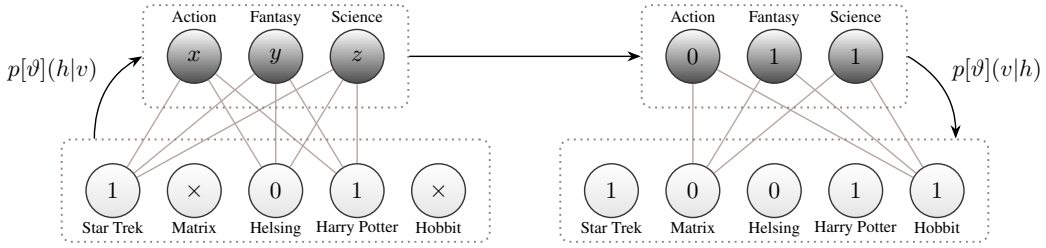


Figure 5: The figure shows a schematic example of the application of the inference process to the prediction of movie ratings for a given imaginary user. At first, only the rated movies will be used to sample hidden values. Then with respect to the sampled hidden values movie ratings for the unrated movies will be sampled and be understood as predictions.

biases of each RBM together such that if two users have rated the same movie then their two RBMs are using the same weights and biases for this visible unit. Figure 4 shows an example for the application of this algorithm for an imaginary user again.

## 5 Inference

The general inference for an RBM, that means sampling from the probability distribution, can be done by using Gibbs sampling explained in the section above. Here we will explicitly talk about the inference for the collaborative filtering problem of movie ratings from users. After talking about the model and the learning process of an RBM the inference process seems to be rather simple in some sense. Figure 5 shows an example for this procedure.

First, we get the vector of rated and unrated movies from a given user. We then have to sample the hidden values via the posterior probability by using only the values for the rated movies. After this we are now able to sample values for the unrated movies again by using the posterior probability with the previously sampled hidden values. These new sampled visible values are then taken to be the predictions of the user ratings.

## 6 Implementation

To gather some ideas about the implementation of an RBM for collaborative filtering we will first examine some real dataset for movie ratings. This will give us an impression of problems and on what to focus when implementing the code in C++. For such investigations the latest MovieLens dataset [4] seems to be appropriate. It contains about 27,000,000 ratings for nearly 58,000 movies made by around 280,000 users.

Before reading those ratings and writing them into a big matrix, like it was done in the example from table 1, we will take a look at the histograms in figure 6 I have generated from this dataset. Here it becomes clear that most of the users have given something between 10 and 100 ratings to movies. There are some outliers. But even they could not give every available movie a rating. The same is true if we look at the histogram for movies. There are a lot of movies with a really small rating count. Hence, if we would generate a big matrix of ratings as in table 1 most of the cells would contain $\times$ and would therefore be empty. This means that the matrix of ratings is really sparse without any given structure. It would be a waste of memory to use a dense matrix format to save and work with the ratings.

As consequence a sparse matrix format should be used. A typical and efficient suggestion is the compressed sparse row format (CSR). According to [1] and [2] it seems to be one of the best suited formats for this kind of problem. Matrix-vector multiplication in CSR can be implemented in parallel on CPUs and modern GPUs using CUDA. Typically, one does not like to implement such a low-level structure by hand. As an alternative one could use the format provided by the linear algebra libary "Eigen" [15]. [1, 2]

But reading the dataset of ratings presents us now with another problem. One user can give ratings to different movies and movies will receive different ratings from different users. Talking in the senses of relational databases the rating dataset can be described as an $n$-to-$m$ connection. Users and movies are typically referenced by some magic identification number which cannot be used as index for a vector or a matrix. This means while reading the dataset we have to keep track of unique users and unique movies by their identification number. Afterwards we have to be able to map these numbers to the indices of our rating matrix in both directions. The most efficient solution is to save

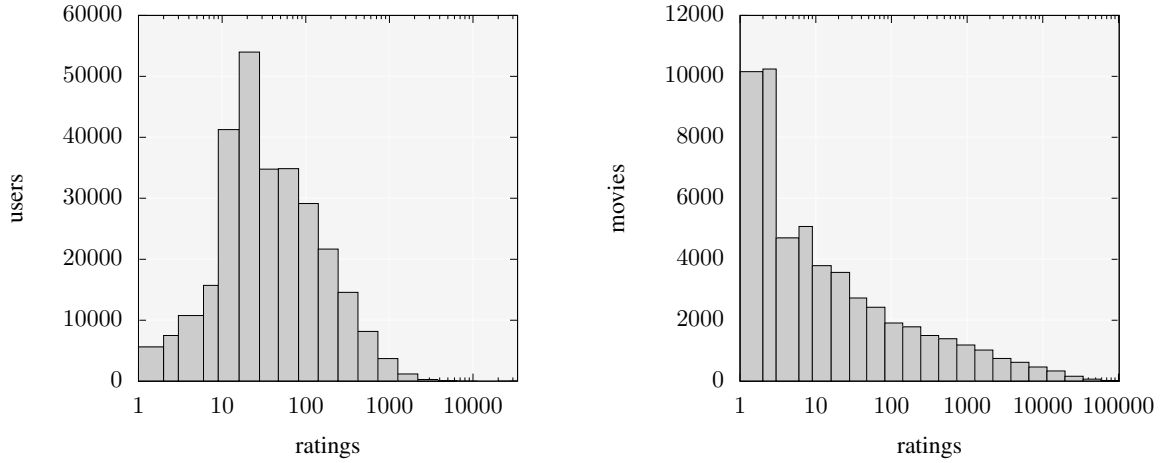


Figure 6: The figure shows two histograms for the latest MovieLens dataset [4]. The left one displays the counts of users over the number of ratings they have given to movies. The right one displays the counts of movies over the number of ratings they have received by users. It becomes clear that the matrix of ratings is a sparse matrix with no special structure.

pairs of identification numbers and indices inside a hash map. This will make sure every identification number is only inserted once and can be accessed in constant time complexity on average. Based on this hash map we can then generate a data vector of identification numbers with their position to be the index saved in the hash map.

Until now we always assumed the values of the ratings to be binary. In reality those values most of the time have a small discrete set of values like $\{1, 2, 3, 4, 5\}$. This difference can be taken into account by mapping those values to $0$ or $1$. But in reality this is a poor approximation because we loose a lot information by doing this. A much more robust method is given by the usage of categorical RBMs which keep the main structure of a binary RBM but allow the visible values to be elements of a finite set of values. [7, 10, 13, 14]

Additionally, one should think of representing rating values as floating point types in C++. This approach will of course use much more memory. But we have got back a lot of memory by using a sparse matrix format. Using this memory for the floating points types will speed up the complete computation because we do not have to cast small integers to floating point values and vice versa.

In section 4 we made clear that we will use one RBM for every user and that their weights and biases are connected. But in the end this was done from the abstract point of view of mathematical theory. Implementing one RBM for every user and connecting them somehow introduces an enormous overhead to the computation. But here the sparse matrix format comes in handy. We will only store one RBM in memory because weights and biases are connected anyway. The learning process on the other hand can be done independently for every user by using the entries in the sparse matrix of ratings directly.

If we want to learn an RBM based on some ratings then afterwards we need to test the learned parameters by some test dataset. [13] gives us a simple and robust method to do this with the MovieLens dataset [4]. It seems to be useful to use the complete user dataset in the training and test dataset. Hence, one should only divide the ratings of users into training and test dataset. [13]

## 7 Further Development

RBMs consist of a simple mathematical structure and can be trained by using CD which makes learning and inference an efficient procedure. Additionally, CD is open to small high-level optimizations according to [7]. Adjusting the learning rate or the mini-batch size for stochastic gradient ascent is only the beginning. Typically, one introduces momentum and weight-decay to the optimization method to reduce noise and to penalize too large weights and biases. Even the initial values for the weights and biases can be tweaked by using different random distributions. Using different varieties of CD, like persistent CD, in one computation can achieve a phenomenal speed up in the learning process in com-

parison to a naive implementation of CD. Of course it is possible to change the underlying mathematical structure and to use different types of units, like softmax or multinomial units, or to change the network slightly and introduce so-called conditional factored RBMs which seem to solve the problem of collaborative filtering much more efficiently. But doing all this, one should not forget about the monitoring of the results. Especially, overfitting and the learning progress should be measured. [7, 14]

## References


[1] Bell, Nathan and Michael Garland: *Efficient sparse matrix-vector multiplication on cuda*. NVIDIA Technical Report, Dezember 2008.

[2] Bell, Nathan and Michael Garland: *Implementing sparse matrix-vector multiplication on throughput-oriented processors*. ACM, 2009.

[3] Fischer, Asja and Christian Igel: *An introduction to restricted boltzmann machines*. LNCS, 7441:14–36, 2012.

[4] GroupLens: *Movielens dataset*, 2018. https://grouplens.org/datasets/movielens/latest/, visited on 2019-01-21.

[5] Harper, F. Maxwell and Joseph A. Konstan: *The movielens datasets: History and context*. ACM Trans. Interact. Intell. Syst., 5(4):19:1–19:19, December 2015, ISSN 2160-6455. http://doi.acm.org/10.1145/2827872.

[6] Hinton, G. E. and R. R. Salakhutdinov: *Reducing the dimensionality of data with neural networks*. SCIENCE, pages 504–507, 2006.

[7] Hinton, Geoffrey: *A practical guide to training restricted boltzmann machines: Version 1*. 2010. https://www.cs.toronto.edu/~hinton/absps/guideTR.pdf.

[8] Larochelle, Hugo and Yoshua Bengio: *Classification using discriminative restricted boltzmann machines*. Proceedings of the 25th International Conference on Machine Learning, 2008.

[9] Montúfar, Guido: *Restricted boltzmann machines: Introduction and review*. CoRR, abs/1806.07066, 2018. http://arxiv.org/abs/1806.07066.

[10] Murphy, Kevin P.: *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012, ISBN 978-0-262-01802-9.

[11] Netflix: *Netflix prize*, 2009. https://www.netflixprize.com/index.html, visited on 2019-01-21.

[12] Netflix: *Netflix prize dataset*, 2009. https://archive.org/details/nf_prize_dataset.tar, visited on 2019-01-21.

[13] Oppermann, Artem: *Deep learning meets physics: Restricted boltzmann machines part i*, 2018. https://towardsdatascience.com/deep-learning-meets-physics-restricted-boltzmann-machines-part-i-6df5c4918c15, visited on 2019-01-22.

[14] Salakhutdinov, Ruslan, Andriy Mnih, and Geoffrey Hinton: *Restricted boltzmann machines for collaborative filtering*. Proceedings of the 24th international conference on Machine learning, pages 791–798, 2007.

[15] Team, Eigen: *Eigen documentation*. http://eigen.tuxfamily.org/dox/, 2018. [Online; accessed 30-August-2018].

[16] Wikipedia: *Collaborative filtering*, 2019. https://en.wikipedia.org/wiki/Collaborative_filtering, visited on 2019-02-11.