CSC-2515 Final Project
Tailoring Boltzmann Machines to Netflix Data

Wesley George WGEORGE@CS.TORONTO.EDU
Dustin Wehr WEHR@CS.TORONTO.EDU

1. Introduction

A boltzmann machine, introduced in [HS83], is a probabilistic model based on an undirected graph. Let $G = (V, E)$ be an undirected graph, with a set of weights,$W_E = \{w_{\{u,v\}}\}_{\{u,v\}\in E} \subset \mathbb{R}$, on the edges and a set of weights $W_V = \{w_v\}_{v\in V} \subset \mathbb{R}$ on the vertices. We define the energy of $a : V \to \{0,1\}$, an assignment of binary states to the vertices of the graph, as

$$E(a) = -\sum_{v\in V}\left(\sum_{\{u,v\}\in E} w_{\{u,v\}}a(u)a(v) + w_v a(v)\right) \tag{1}$$

If we let $p(f) \propto e^{-E(f)}$, we obtain a probability distribution over assignments of binary states to the vertices of the machine. Let $\mathcal{A}(V)$ denote the set of assigments of $\{0,1\}$ values to V, i.e. $\mathcal{A}(V) = \{a \mid a : V \to \{0,1\}\}$.

A restricted boltzmann machine(RBM), introduced in [Smo86], is a boltzmann machine where the underlying graph is a complete bipartite graph. Let $V = V_1 \cup V_2$ be the partition of the vertex set. If we fix an assignment $a_1 \in \mathcal{A}(V_1)$ (respectively $a_2 \in \mathcal{A}(V_2)$), we can compute the conditional distribution over $\mathcal{A}(V_2)$ (resp. over $\mathcal{A}(V_1)$). Thus using Gibbs sampling we can sample from the joint distribution over $\mathcal{A}(V_1) \times \mathcal{A}(V_2)$. This leads to a tractable way of training an RBM to produce a desired probability distribution, see [HO06].

Salakhutdinov and Mnih obtained impressive results in the netflix competition by training an RBM to reproduce ratings data encoded as categorical data([SMH07]). We have attempted to improve upon their results by encorporating the ordinal nature of the data, and by explicitly considering the error function used by Netflix to evaluate the performance of a prediction model.

In section 2 we discuss non-categorical representations of the ordinal ratings data. In section 3 we discuss the error function of the RBM, how it is different from the one specified by Netflix, and how we tried to overcome this. In section 4 we present the results of our experiments obtained by altering an RBM in the methods described. Finally in section 5 we offer our conclusions and future work.

2. Representing Ordinal Data

The netflix data can be thought of as a matrix where the u, jth entry is the rating that user u has assigned to movie j, or 0 if the user has not rated the movie. Ratings are integers in the set $[1,5]$. If there are M movies in total, then correct inference of new ratings is equivalent to correct inference about the probability distribution of ratings over the space $[1,5]^M$.

Since an RBM models a probability distributions of binary strings, we need a way of encoding ratings as binary strings[1]. [SMH07] treat the ratings as categorical data, representing each one with a standard 1-of-5 encoding. Since each rating is encoded over 5 binary units, these RBMs effectively model probability distributions over $[0,1]^{5M}$.

Multinomials are mapped to ratings by taking the 'inner product' $R : [0,1]^5 \to [1,5] : x \mapsto \langle [1...5], x \rangle$ of the output in question. If we let $f_5 : [5] \to [0,1]^5 : i \mapsto$ standard 1-of-5 coding of i (i.e. $[f_5(1)|...|f_5(5)]$ is the 5x5 identity matrix) then we observe that

$$\forall i \in \{1, ..., 5\} \; [R(f_5(i)) = i] \tag{2}$$

We note that this property is necessary for any reasonable encoding and that f is not unique.

The prediction in their model for movie j by user u, is $r(\mathbb{E}[v|\mathbf{V}])$ where v is the multinomial visible variable for movie j and $\mathbf{V}$ is the configuration of visible units corresponding to the ratings for u given in the training set. The underlying probability distribution for this expectation is, of course, the probability distribution specified by the RBM, conditioned on the assignment $\mathbf{V}$.

Suppose, during the training of an RBM, a rating $r = 3$ is reconstructed as $w = [0 \; \frac{1}{2} \; 0 \; \frac{1}{2} \; 0]$ instead of the canonical form $f_5(3)$. Despite the fact that $r(w) = r(f(3)) = 3$, the RBM is penalized for this reconstruction and the weights will be adjusted so future reconstructions of r look more like $f(3)$. Now even if in adjusting the weights, future predictions of r will be some convex combination of w and $f(3)$ (i.e. $(1-\lambda)w + \lambda f_5(3)$ for some λ, and the rating is always 3 for all such representations), these weight changes are not adversely affecting the prediction, but it is unnecessarily perturbing the weights and conforming unnecessarily to the structure of training set. Our aim is to pick a representation of the ratings data that suggests to the RBM the relationships between the various categories so that the RBM is not harshly penalized if it doesn't reconstruct the exact input, as long as it gives the correct rating.

The encoding of [SMH07] is a deterministic encoding using a 5-state multinomial where the ith unit of the multinomial variable is activated for a rating of i. We propose a probabilistic generalization of this encoding by allowing a rating r as $f_5(r)$, but also occaisionally as $f_5(r-1)$ or $f_5(r+1)$. We can immediately see a problem for ratings of 1 and 5 - given a 5 state multinomial, there is no way to represent a rating of 0 or 6. If we want the (clearly necessary) property that the decoding process undoes the encoding process (or that the decoding of the expectation of the encoding is the original rating), we cannot encode a 1 as anything higher.

To overcome this difficulty, we change the representation space of the ratings presented to the RBM. Instead of using 5-state multinomials, we can use 7-state multinomials to "allow" for ratings of 0 and 6. If we say $[f_7(0)|f_7(1)|...|f_7(6)]$ is the 7x7 identity matrix, then we can define $\mathbb{P}[e(i) = f_7(i)] = p$ and $\mathbb{P}[e(i) = f_7(i-1)] = \mathbb{P}[e(i) = f_7(i+1)] = \frac{1-p}{2}$, and define our (deterministic) decoding function $d(x) = \langle [0...6], x \rangle$. In this case we have $\mathbb{E}[d(e(i))] = d(\mathbb{E}[e(i)]) = i$ for all $i \in \{1, .., 5\}$. We note also that despite the fact that we are adding units to

[1]RBM can also model distributions over the real line using normally or poisson distributed hidden or visible units, but we have ignored this units for this project

represent ratings of 0 and 6, it is very unlikely (though technically possible) that we will predict a rating lower than 1 or higher than 5, because none of the training examples we present to the RBM have this property.

Besides the attempt to make the data look more ordinal to the RBM, there are two justifications for this encoding based on the psychology of users. First, the range of ratings a user can assign is perhaps too small to capture what the user would like to express. A user may wish to assign a rating of 3.5 to a movie, but is forced to choose between 3 and 4. Second, there is certainly variance in the rating assigned by a user based on the context in which they assigned the rating - for example, depending on their current mood they may be inclined to give a higher or lower rating.

3. The Netflix error function

Given an encoding of ratings as binary strings, the distribution of vectors of ratings (i.e. representing a user's rating across all movies) induces a distribution over binary strings. Because an RBM is ultimately a representation of a probability distribution, training an RBM amounts to minimizing the Kullback Leibler divergence between the distribution of binary strings induced by the distribution of ratings and the encoding, and the distribution over binary strings as specified by the particular RBM.

The performance of a model, by Netflix standards, is the root mean square of the difference between the predictions of the model (floating point numbers in [1,5]) and the actual movie rating. What is not clear is that minimizing the KL divergence between the distribution of the RBM and the distribution of ratings actually minimizes the Netflix error function. Given our earlier comments in section 2, training by contrastive divergence is forcing the RBM to conform to the training data more than is necessary to minimize the Netflix error function. Since the ultimate goal of the Netflix competition is to minimize this specific error function, if we are not explicitly optimizing this error function (as [SMH07] is not), we may not be achieving the best possible performance by Netflix standards.

The attempts at an ordinal representation of the input data given in Section 2 do not provide an obvious solution to this issue, and without a better way of representing ordinal data with an RBM, there is no obvious way to minimize this error function by training an RBM. On the other hand, many machine learning methods rely on gradient descent of the error function, and any error function whose gradients can be derived can be used. We trained an autoencoder by first training an RBM on the data, then unrolling the RBM into a neural network and fine-tuning the weights by computing the gradient *on the actual Netflix error function.* The results of this experiment are in section 4.2.

4. Experiments

4.1. **Representing Ordinals as Multinomials.**

4.1.1. *Neural Network with different error functions.* To investigate the benefit of using the error function specified by Netflix, we implemented a feed-forward neural network. It is the closest model to an RBM that we are familiar with, and for which we are able to optimize the error function of our choosing (assuming we can derive the expressions for the gradients needed in backpropogation, which is easy).

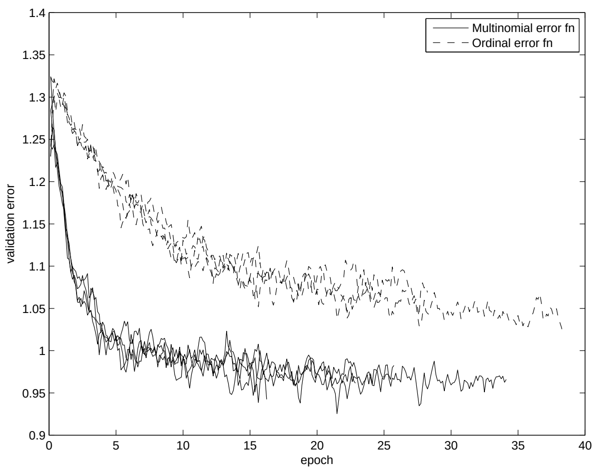


FIGURE 1. Four trials of neural net optimizing different error functions

Figure 4.1.1 depicts four trials of the following experiment. For speed, so that we could do several trials, we trained on only the first 2000 users, and used no regularization. However, the results are similiar on the whole data set. The architecture is the same as what you get if you unfold the RBM and untie the weights, so 5000 multinomial input and outputs units. We initialized two nets to the same small random values. We used a batch size of 250. Using the ordinal error function consistently produced better results. We also observed that the difference between the training and validation errors for the multinomial error function was much less than for the ordinal error function, but that is not depicted in the figure.

4.1.2. *Input encoding for a feed forward Neural Network.* Our first experiment concerning input representation was with a feed forward neural net. Our intention was to make an input representation that could not be worse than the multinomial representation, and so we simply added, for each movie, four binary units that represent the binary features “user rated 4 or 5”, “user rated 3 or 4”, etc. So all the local minima that were there before, would still be there - the net would just need to ignore the extra units. In retrospect, it *could* have been worse, because backpropogation only does local optimization. We trained two networks, Net1 with the normal representation, and Net2 with the extra units.

Figure 4.1.2 depicts the result of a single, typical trial. The weights that Net1 and Net2 had in common, were initialized to the same random values. The extra weights for Net2 (on the edges attached to the extra visible units) were initialized to zero. So the two nets started out computing exactly the same function. Also, they looked at the data in the same order. That explains why the curves are so similarly shaped.

4.1.3. *Input encodings and RBMs.* We trained an RBM with 100 hidden units, with regularization of 0.001 and learning rate (for biases and weights) of 0.1 for 10 epochs on a variety of different encodings of the netflix data. The encodings used are:

- regular - standard 1-of-5 encoding (as in [SMH07])

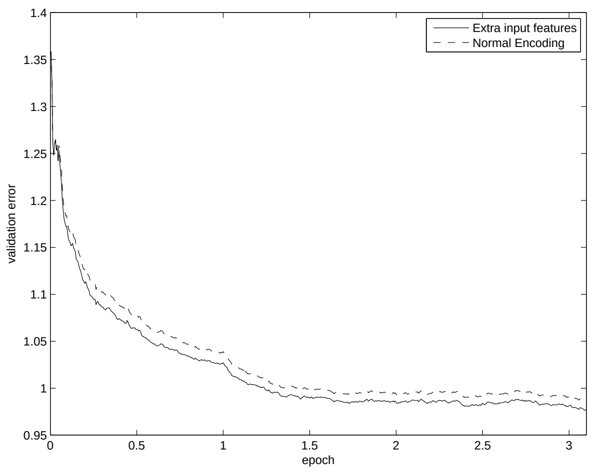


FIGURE 2. Neural net with and without extra binary input features

- 'Suzy[2]' - ratings are encoded as 7-state multinomial. The 7 states represent ratings of 0 through 6. A rating of i is encoded as ith state of the multinomial with probability $p = 0.7$ and $i-1$th or $i+1$th state of the multinomial with probability 0.15. The expectations of each of the ratings are given as the rows of the following matrix:

$$\begin{pmatrix} 0.15 & 0.7 & 0.15 & 0 & 0 & 0 & 0 \\ 0 & 0.15 & 0.7 & 0.15 & 0 & 0 & 0 \\ 0 & 0 & 0.15 & 0.7 & 0.15 & 0 & 0 \\ 0 & 0 & 0 & 0.15 & 0.7 & 0.15 & 0 \\ 0 & 0 & 0 & 0 & 0.15 & 0.7 & 0.15 \end{pmatrix}$$

  Note that a rating of 2 is always presented as (0 .15 .7 .15 0 0 0), as opposed to choosing (0 1 0 0 0 0 0) with probability .15, (0 0 1 0 0 0 0) with probability .7, etc. We did not investigate the latter encoding, though it may work well.
- 'Seth' - as suzy but with $p = 0.4$. The expectations of the encoding of each of the ratings are given as the rows of the following matrix:

$$\begin{pmatrix} 0.3 & 0.4 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0.3 & 0.4 & 0.3 & 0 & 0 & 0 \\ 0 & 0 & 0.3 & 0.4 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 0.4 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0.3 & 0.4 & 0.3 \end{pmatrix}$$

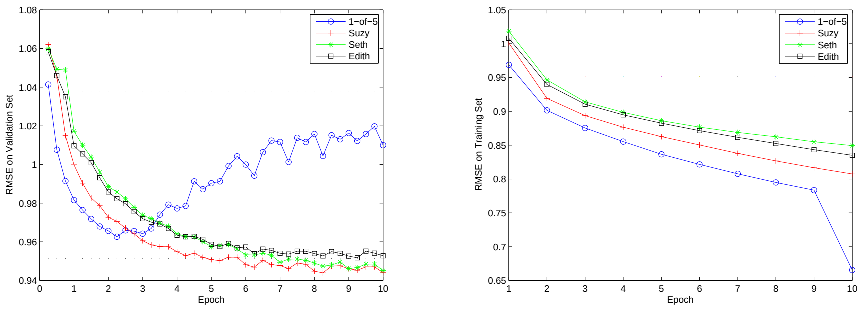
- 'Edith[3]' - ratings encoded to multinomials with 11 values, representing ratings $\{0.5, 1, 1.5, ..., 5, 5.5\}$. A rating of i is encoded as i with probability $p = 0.5$ and $i - 0.5$ or $i + 0.5$ with probability 0.25. The expectation of the encoding of each of the ratings are given as the rows of the following

[2]S for seven

[3]E for eleven

matrix:

$$\begin{pmatrix} 0.25 & 0.5 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.25 & 0.5 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0.25 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0.25 \end{pmatrix}$$



(a) Validation error (b) Training error

FIGURE 3. RBM errors for various input representations

We created a validation set from about 1% of the non-zero ratings from training data. This was selected in a way to minimize the bias towards users with a large number of ratings; first we selected a user uniformly at random, then we selected one of their ratings at random; Netflix does something similar in building their test set. The validation error for the RBM was computed by reconstructing the full vector of ratings for a user based on the training data (rather than using the free energy).

Figure 3(a) shows the validation error resulting from training RBMs on these input representations. In this figure we have also plotted the error for the movie-average predictor(1.038 on our validation set), and for the netflix baseline (0.9514 on their hidden test set). We are pleased to report that Suzy and Seth beat the netflix baseline, which is a challenging task given the small size of the dataset. The best validation error for Suzy was 0.9439, the minimum for Seth was 0.9451 and the minimum for Edith was 0.9517.

Suzy, Seth and Edith are variations on the same theme - we are probabilistically choosing which state of the multinomial to activate based on what rating the user has selected, and ratings that are close to each other will often activate the same variable. Edith is a more moderate representation - we introduced variables representing values between integer ratings, so 2 would occaisionally encoded as 1.5 or 2.5. By activating 1.5 which is also activated by the rating 1, we are suggesting an association between these variables to the RBM. Before running the experiment, we felt that Edith would be the best performer - by introducing variables that correlate neighbouring ratings, we felt we had suggested the ordinal nature of the data in a moderate way. Suzy and Seth were both much more extreme versions of this encoding, encoding neighbouring ratings in a very similar way. We were surprised to find that Seth performed almost as well as Suzy, and that Suzy was the best performer on the dataset.

We also note that RBMs trained on the standard(1-of-5) representation of the ratings data overfit much faster than RBMs trained on ratings represented as 'Suzy', 'Seth' or 'Edith'. We also tried (results not shown) training RBMs with 25 and 10 hidden units. The 25 hidden unit RBM still overfit, but a little later. The 10 hidden unit RBM did not overfit in 10 epochs, but it also did not improve on the validation error obtained by the 100 hidden unit RBM. We also point out that training time for Suzy, Seth and Edith was longer than for the regular input representation because there were more input variables (7 and 11 per movie respectively instead of 5 for the regular encoding).

4.2. **Fine tuning using backpropogation.** We hypothesized that we could improve a trained RBM by unrolling it into a feed forward neural net, and then "fine tuning" the weights with backpropogation, as in [HS06]. Further, we used the gradient computed from the ordinal error function specified by Netflix, as opposed to the multinomial error function that gets minimized by contrastive divergence (of course, decreasing the multinomial error does decrease the ordinal error). We were not successful. We observed the training of two models, one where the learning rates were initialized to zero, so that no training happened, and another where the learning rates were initialized to very small values. As in Section 4.1.2, everything else was the same between the two training sessions. We tried various input representations, including the normal one. We tried various initializations of the learning rates, including not training the first level weights and biases at all. Whenever we initialized the learning rates to large enough values that the training error would visibly decrease, the validation error would visibly increase. Because the training errror *did* decrease, we do not think there is an error in the code. We believe that the idea is flawed -in particular, the idea of using the ordinal error function; fine tuning with backpropogation has been shown to work in other situations- though we cannot give an adequate explanation. We did not attempt to do the fine tuning with the multinomial error function.

5. Conclusion and Future Work

We have shown that the 1-of-k encoding of the data is not the best, and that encodings that consider the ordinal nature of the data can obtain better results. We have also shown that by explicitly optimizing for the Netflix error function, in the feed-forward neural net model, better results can be achieved on a validation set.

There are a number of questions that this exploration raised that we did not have time to address, but that provide avenues for future work. First, it is not clear that any of our choices of ratings representation in binary stochastic units are optimal. For the input representation known as 'suzy', we are suggesting relationships between each rating and it's immediate neighbour. For example, by encoding a rating of 2 occaisionally as $f(1)$ and $f(3)$, we are suggesting associations between 2 and 1 and between 2 and 3. What we are trying to do is embed the five ratings on a line. Consider the set of points $x_1 = (0,0,0,0), x_2 = (1,0,0,0), x_3 = (1,1,0,0), x_4 = (1,1,1,0), x_5 = (1,1,1,1))$ - each one is a unit distance from it's neighbour, but distances are not additive, nor are the points arranged optimally in space.

Besides playing with various multinomial representations of the data, another choice for input representation is to represent each rating with a single binary stochastic unit, representing higher ratings by activating the unit with higher frequency. For example, we could turn a unit on with a probability of $r/6$ for a rating of r. This encoding strictly enforces the order of the ratings, it also accomodates for the fact that users are not selecting movies to watch at random - i.e. if they expect that they will hate a movie, they are extremely unlikely to view it (thus a lack of a rating does, in a certain sense, imply a dislike from the user. A further benefit of this model, is that it has a smaller number of weights than the multinomial representation. A possible drawback is that performance may be very sensitive to the particular mapping chosen.

References

[HO06] Geoffrey E. Hinton and Simon Osindero. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006.

[HS83] G. E. Hinton and T. Sejnowski. Optimal perceptual inference. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 448–453, Washington, D.C., June 1983. IEEE Computer Society Press.

[HS06] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[SMH07] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 791–798, New York, NY, USA, 2007. ACM.

[Smo86] P. Smolensky. Information processing in dynamical systems: foundations of harmony theory. pages 194–281, 1986.