Restricted Boltzmann Machines for Collaborative Filtering

Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Andriy Mnih AMNIH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
University of Toronto, 6 King's College Rd., Toronto, Ontario M5S 3G4, Canada

Abstract

Most of the existing approaches to collaborative filtering cannot handle very large data sets. In this paper we show how a class of two-layer undirected graphical models, called Restricted Boltzmann Machines (RBM's), can be used to model tabular data, such as user's ratings of movies. We present efficient learning and inference procedures for this class of models and demonstrate that RBM's can be successfully applied to the Netflix data set, containing over 100 million user/movie ratings. We also show that RBM's slightly outperform carefully-tuned SVD models. When the predictions of multiple RBM models and multiple SVD models are linearly combined, we achieve an error rate that is well over 6% better than the score of Netflix's own system.

1. Introduction

A common approach to collaborative filtering is to assign a low-dimensional feature vector to each user and a low-dimensional feature vector to each movie so that the rating that each user assigns to each movie is modeled by the scalar-product of the two feature vectors. This means that the $N \times M$ matrix of ratings that N users assign to M movies is modeled by the matrix X which is the product of an $N \times C$ matrix U whose rows are the user feature vectors and a $C \times M$ matrix V' whose columns are the movie feature vectors. The rank of X is C – the number of features assigned to each user or movie.

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

Low-rank approximations based on minimizing the sum-squared distance can be found using Singular Value Decomposition (SVD). In the collaborative filtering domain, however, most of the data sets are sparse, and as shown by Srebro and Jaakkola (2003), this creates a difficult non-convex problem, so a naive solution is not going work.[1]

In this paper we describe a class of two-layer undirected graphical models that generalize Restricted Boltzmann Machines to modeling tabular or count data (Welling et al., 2005). Maximum likelihood learning is intractable in these models, but we show that learning can be performed efficiently by following an approximation to the gradient of a different objective function called "Contrastive Divergence" (Hinton, 2002).

2. Restricted Boltzmann Machines (RBM's)

Suppose we have M movies, N users, and integer rating values from 1 to K. The first problem in applying RBM's to movie ratings is how to deal efficiently with the missing ratings. If all N users rated the same set of M movies, we could treat each user as a single training case for an RBM which had M "softmax" visible units symmetrically connected to a set of binary hidden units. Each hidden unit could then learn to model a significant dependency between the ratings of different movies. When most of the ratings are missing, we use a different RBM for each user (see Fig. 1). Every RBM has the same number of hidden units, but an RBM only has visible softmax units for the movies rated by that user, so an RBM has few connections if that user rated few movies. Each RBM only has a single training case, but all of the corresponding

[1] We describe the details of the SVD training procedure in section 7.

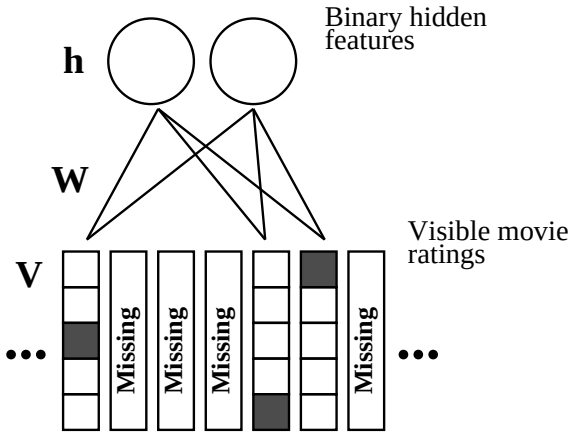

Figure 1. A restricted Boltzmann machine with binary hidden units and softmax visible units. For each user, the RBM only includes softmax units for the movies that user has rated. In addition to the symmetric weights between each hidden unit and each of the $K = 5$ values of a softmax unit, there are 5 biases for each softmax unit and one for each hidden unit. When modeling user ratings with an RBM that has Gaussian hidden units, the top layer is composed of linear units with Gaussian noise.

weights and biases are tied together, so if two users have rated the same movie, their two RBM's must use the same weights between the softmax visible unit for that movie and the hidden units. The binary states of the hidden units, however, can be quite different for different users. From now on, to simplify the notation, we will concentrate on getting the gradients for the parameters of a single user-specific RBM. The full gradients with respect to the shared weight parameters can then be obtained by averaging over all N users.

Suppose a user rated m movies. Let $\mathbf{V}$ be a $K \times m$ observed binary indicator matrix with $v_i^k = 1$ if the user rated movie i as k and 0 otherwise. We also let h_j, $j = 1, ..., F$, be the binary values of hidden (latent) variables, that can be thought of as representing stochastic binary features that have different values for different users.

2.1. The Model

We use a conditional multinomial distribution (a "softmax") for modeling each column of the observed "visible" binary rating matrix $\mathbf{V}$ and a conditional Bernoulli distribution for modeling "hidden" user features $\mathbf{h}$ (see Fig. 1):

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

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

where $\sigma(x) = 1/(1 + e^{-x})$ is the logistic function, W_{ij}^k is a symmetric interaction parameter between feature j and rating k of movie i, b_i^k is the bias of rating k for movie i, and b_j is the bias of feature j. Note that the b_i^k can be initialized to the logs of their respective base rates over all users.

The marginal distribution over the visible ratings $\mathbf{V}$ is:

$$p(\mathbf{V}) = \sum_{\mathbf{h}} \frac{\exp\left(-E(\mathbf{V}, \mathbf{h})\right)}{\sum_{\mathbf{V}', \mathbf{h}'} \exp\left(-E(\mathbf{V}', \mathbf{h}')\right)} \quad (3)$$

with an "energy" term given by:

$$\begin{aligned} E(\mathbf{V}, \mathbf{h}) = &-\sum_{i=1}^{m}\sum_{j=1}^{F}\sum_{k=1}^{K} W_{ij}^k h_j v_i^k + \sum_{i=1}^{m} \log Z_i \\ &-\sum_{i=1}^{m}\sum_{k=1}^{K} v_i^k b_i^k - \sum_{j=1}^{F} h_j b_j \end{aligned} \quad (4)$$

where $Z_i = \sum_{l=1}^{K} \exp\left(b_i^l + \sum_j h_j W_{ij}^l\right)$ is the normalization term that ensures that $\sum_{l=1}^{K} p(v_i^l = 1|\mathbf{h}) = 1$. The movies with missing ratings do not make any contribution to the energy function.

2.2. Learning

The parameter updates required to perform gradient ascent in the log-likelihood can be obtained from Eq. 3:

$$\begin{aligned} \Delta W_{ij}^k &= \epsilon \frac{\partial \log p(\mathbf{V})}{\partial W_{ij}^k} = \\ &= \epsilon \Big(<v_i^k h_j>_{data} - <v_i^k h_j>_{model} \Big) \end{aligned} \quad (5)$$

where ϵ is the learning rate. The expectation $<v_i^k h_j>_{data}$ defines the frequency with which movie i with rating k and feature j are on together when the features are being driven by the observed user-rating data from the training set using Eq. 2, and $<\cdot>_{model}$ is an expectation with respect to the distribution defined by the model. The expectation $<\cdot>_{model}$ cannot be computed analytically in less than exponential time. MCMC methods (Neal, 1993) can be employed to approximate this expectation. These methods, however, are quite slow and suffer from high variance in their estimates.

To avoid computing $<\cdot>_{model}$, we follow an approximation to the gradient of a different objective function

called "Contrastive Divergence" (CD) (Hinton, 2002):

$$\Delta W_{ij}^k = \epsilon(<v_i^k h_j>_{data} - <v_i^k h_j>_T) \quad (6)$$

The expectation $<\cdot>_T$ represents a distribution of samples from running the Gibbs sampler (Eqs. 1,2), initialized at the data, for T full steps. T is typically set to one at the beginning of learning and increased as the learning converges. By increasing T to a sufficiently large value, it is possible to approximate maximum likelihood learning arbitrarily well (Carreira-Perpinan & Hinton, 2005), but large values of T are seldom needed in practice. When running the Gibbs sampler, we only reconstruct (Eq. 1) the distribution over the non-missing ratings. The approximate gradients of CD with respect to the shared weight parameters of Eq. 6 can be then be averaged over all N users.

It was shown (Hinton, 2002) that CD learning is quite efficient and greatly reduces the variance of the estimates used for learning. The learning rule for the biases is just a simplified version of Eq. 6.

2.3. Making Predictions

Given the observed ratings $\mathbf{V}$, we can predict a rating for a new query movie q in time linear in the number of hidden units:

$$p(v_q^k = 1|\mathbf{V}) \propto \sum_{h_1,\ldots,h_p} \exp(-E(v_q^k, \mathbf{V}, \mathbf{h})) \quad (7)$$

$$\propto \Gamma_q^k \prod_{j=1}^{F} \sum_{h_j \in \{0,1\}} \exp\big(\sum_{il} v_i^l h_j W_{ij}^l + v_q^k h_j W_{qj}^k + h_j b_j\big)$$

$$= \Gamma_q^k \prod_{j=1}^{F} \left(1 + \exp\big(\sum_{il} v_i^l W_{ij}^l + v_q^k W_{qj}^k + b_j\big)\right)$$

where $\Gamma_q^k = \exp(v_q^k b_q^k)$. Once we obtain unnormalized scores, we can either pick the rating with the maximum score as our prediction, or perform normalization over K values to get probabilities $p(v_q = k|\mathbf{V})$ and take the expectation $E[v_q]$ as our prediction. The latter method works better.

When asked to predict ratings for n movies q_1, q_2,..., q_n, we can also compute

$$p(v_{q_1}^{k_1} = 1, v_{q_2}^{k_2} = 1, ..., v_{q_n}^{k_n} = 1|\mathbf{V}) \quad (8)$$

This, however, requires us to make K^n evaluations for each user.

Alternatively, we can perform one iteration of the mean field updates to get the probability distribution over K ratings for a movie q:

$$\hat{p}_j = p(h_j = 1|\mathbf{V}) = \sigma(b_j + \sum_{i=1}^{m}\sum_{k=1}^{K} v_i^k W_{ij}^k) \quad (9)$$

$$p(v_q^k = 1|\hat{\mathbf{p}}) = \frac{\exp\left(b_q^k + \sum_{j=1}^{F} \hat{p}_j W_{qj}^k\right)}{\sum_{l=1}^{K} \exp\left(b_q^l + \sum_{j=1}^{F} \hat{p}_j W_{qj}^l\right)} \quad (10)$$

and take an expectation as our prediction. In our experience, Eq. 7 makes slightly more accurate predictions, although one iteration of the mean field equations is considerably faster. We use the mean field method in the experiments described below.

3. RBM's with Gaussian Hidden Units

We can also model "hidden" user features $\mathbf{h}$ as Gaussian latent variables (Welling et al., 2005). This model represents an undirected counterpart of pLSI (Hofmann, 1999):

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

$$p(h_j = h|\mathbf{V}) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{\left(h - b_j - \sigma_j \sum_{ik} v_i^k W_{ij}^k\right)^2}{2\sigma_j^2}\right)$$

where σ_j^2 is the variance of the hidden unit j.

The marginal distribution over visible units $\mathbf{V}$ is given by Eq. 3. with an energy term:

$$\begin{aligned} E(\mathbf{V}, \mathbf{h}) &= -\sum_{ijk} W_{ij}^k h_j v_i^k + \sum_i \log Z_i \\ &\quad - \sum_{ik} v_i^k b_i^k + \sum_j \frac{(h_j - b_j)^2}{2\sigma_j^2} \end{aligned} \quad (11)$$

We fix variances at $\sigma_j^2 = 1$ for all hidden units j, in which case the parameter updates are the same as defined in Eq. 6.

4. Conditional RBM's

Suppose that we add w to each of the K weights from the K possible ratings to each hidden feature and we subtract w from the bias of the hidden feature. So long as one of the K ratings is present, this does not have any effect on the behaviour of the hidden or visible units because the "softmax" is over-parameterized. If, however, the rating is missing, there is an effect of $-w$ on the total input to the hidden feature. So by using the over-parametrization of the softmax, the RBM can learn to use missing ratings to influence its hidden features, even though it does not try to reconstruct these

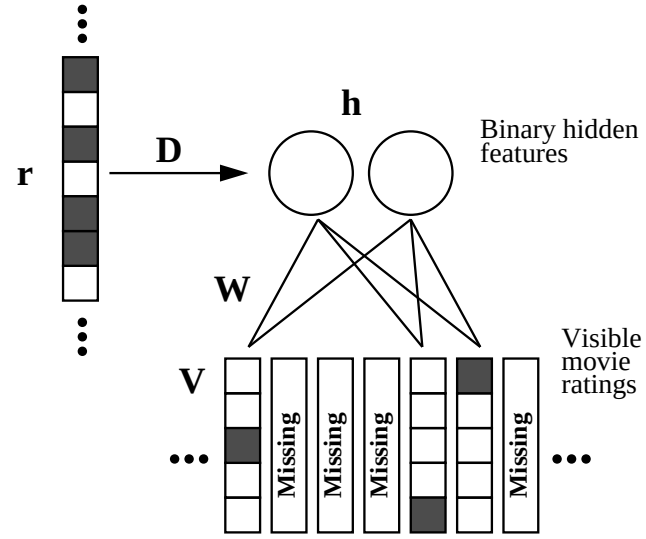

Figure 2. Conditional RBM. The binary vector $\mathbf{r}$, indicating rated/unrated movies, affects binary states of the hidden units.

missing ratings and it does not perform any computations that scale with the number of missing ratings.

There is a more subtle source of information in the Netflix database that cannot be captured by the "standard" multinomial RBM. Netflix tells us in advance which user/movie pairs occur in the test set, so we have a third category: movies that were viewed but for which the rating is unknown. This is a valuable source of information about users who occur several times in the test set, especially if they only gave a small number of ratings in the training set. If, for example, a user is known to have rated "Rocky 5", we already have a good bet about the kinds of movies he likes.

The *conditional* RBM model takes this extra information into account. Let $\mathbf{r} \in \{0,1\}^M$ be a binary vector of length M (total number of movies), indicating which movies the user rated (even if these ratings are unknown). The idea is to define a joint distribution over $(\mathbf{V}, \mathbf{h})$ conditional on $\mathbf{r}$. In the proposed conditional model, a vector $\mathbf{r}$ will affect the states of the hidden units (see Fig. 2):

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

$$p(h_j = 1|\mathbf{V}, \mathbf{r}) = \sigma\left(b_j + \sum_{i=1}^m \sum_{k=1}^K v_i^k W_{ij}^k + \sum_{i=1}^M r_i D_{ij}]\right)$$

where D_{ij} is an element of a learned matrix that models the effect of $\mathbf{r}$ on $\mathbf{h}$. Learning $\mathbf{D}$ using CD is similar to learning biases and takes the form:

$$\Delta D_{ij} = \epsilon\Big(<h_j>_{data} - <h_j>_T \Big) r_i \qquad (12)$$

We could instead define an arbitrary nonlinear function $f(\mathbf{r}|\theta)$. Provided f is differentiable with respect to θ, we could use backpropagation to learn θ:

$$\Delta\theta = \epsilon\Big(<h_j>_{data} - <h_j>_T \Big) \frac{\partial f(\mathbf{r}|\theta)}{\partial \theta} \qquad (13)$$

In particular, $f(\mathbf{r}|\theta)$ can be parameterized as a multilayer neural network.

Conditional RBM models have been successfully used for modeling temporal data, such as motion capture data (Taylor et al., 2006), or video sequences (Sutskever & Hinton, 2006). For the Netflix task, conditioning on a vector of rated/unrated movies proves to be quite helpful – it significantly improves performance.

Instead of using a conditional RBM, we can impute the missing ratings from the ordinary RBM model. Suppose a user rated a movie t, but his/her rating is missing (i.e. it was provided as a part of the *test* set). We can initialize $\mathbf{v_t}$ to the base rate of movie t, and compute the gradient of the log-probability of the data with respect to this input (Eq. 3). The CD learning takes form:

$$\Delta v_t^k = \epsilon\Big(<\sum_j W_t^k h_j>_{data} - <\sum_j W_t^k h_j>_T \Big)$$

After updating v_t^k, for $k = 1, .., K$, v_t^k are renormalized to obtain probability distribution over K values. The imputed values $\mathbf{v_t}$ will now contribute to the energy term of Eq. 4 and will affect the states of the hidden units. Imputing missing values by following an approximate gradient of CD works quite well on a small subset of the Netflix data set, but is slow for the complete data set. Alternatively, we can use a set of mean field equations Eqs. 9, 10 to impute the missing values. The imputed values will be quite noisy, especially at the early stages of training. Nevertheless, in our experiments, the model performance was significantly improved by using imputations and was comparable to the performance of the conditional RBM.

5. Conditional Factored RBM's

One disadvantage of the RBM models we have described so far is that their current parameterization of $W \in R^{M \times K \times F}$ results in a large number of free parameters. In our current implementation, with $F = 100$

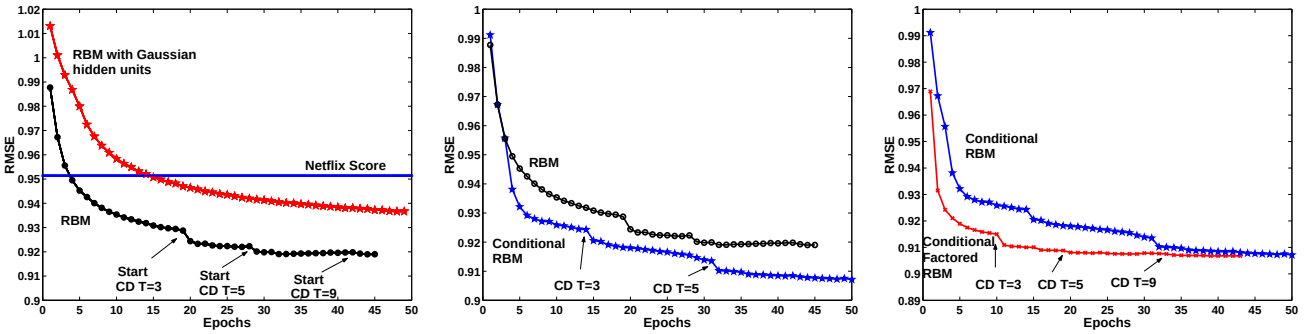

Figure 3. Performance of various models on the validation data. Left panel: RBM vs. RBM with Gaussian hidden units. Middle panel: RBM vs. conditional RBM. Right panel: conditional RBM vs. conditional factored RBM. The y-axis displays RMSE (root mean squared error), and the x-axis shows the number of epochs, or passes through the entire training dataset.

(the number of hidden units), $M = 17770$, and $K = 5$, we end up with about 9 million free parameters. By using proper weight-decay to regularize the model, we are still able to avoid serious overfitting. However, if we increase the number of hidden features or the number of movies,[2] learning this huge parameter matrix W becomes problematic. Reducing the number of free parameters by simply reducing the number of hidden units does not lead to a good model because the model cannot express enough information about each user in its hidden state.

We address this problem by factorizing the parameter matrix W into a product of two lower-rank matrices A and B. In particular:

$$W_{ij}^k = \sum_{c=1}^{C} A_{ic}^k B_{cj} \tag{14}$$

where typically $C \ll M$ and $C \ll F$. For example, setting $C = 30$, we reduce the number of free parameters by a factor of three. We call this model a factored RBM. Learning matrices A and B is quite similar to learning W of Eq. 6:

$$\Delta A_{ic}^k = \epsilon \Big(<[\sum_j B_{cj} h_j] v_i^k>_{data} - <[\sum_j B_{cj} h_j] v_i^k>_T \Big)$$

$$\Delta B_{cj} = \epsilon \Big(<[\sum_{ik} A_{ic}^k v_i^k] h_j>_{data} - <[\sum_{ik} A_{ic}^k v_i^k] h_j>_T \Big)$$

[2]Netflix's own database contains about 65000 movie titles.

In our experimental results section we show that a conditional factored RBM converges considerably faster than a conditional unfactored RBM.

6. Experimental Results

6.1. Description of the Netflix Data

According to Netflix, the data were collected between October, 1998 and December, 2005 and represent the distribution of all ratings Netflix obtained during this period. The training data set consists of 100,480,507 ratings from 480,189 randomly-chosen, anonymous users on 17,770 movie titles. As part of the training data, Netflix also provides validation data, containing 1,408,395 ratings. In addition to the training and validation data, Netflix also provides a test set containing 2,817,131 user/movie pairs with the ratings withheld. The pairs were selected from the most recent ratings from a subset of the users in the training data set, over a subset of the movies. To reduce the unintentional fine-tuning on the test set that plagues many empirical comparisons in the machine learning literature, performance is assessed by submitting predicted ratings to Netflix who then post the root mean squared error (RMSE) on an unknown half of the test set. As a baseline, Netflix provided the score of its own system trained on the same data, which is 0.9514.

6.2. Details RBM Training

We train the RBM with $F = 100$, and the conditional factored RBM with $F = 500$, and $C = 30$. To speed-up the training, we subdivided the Netflix dataset into small mini-batches, each containing 1000 cases (users), and updated the weights after each mini-batch. All models were trained for between 40 and 50 passes (epochs) through the entire training dataset.

The weights were updated using a learning rate of 0.01/batch-size, momentum of 0.9, and a weight decay of 0.001. The weights were initialized with small random values sampled from a zero-mean normal distribution with standard deviation 0.01. CD learning was started with $T = 1$ and increased in small steps during training.

6.3. Results

We compare different models based on their performance on the validation set. The error that Netflix reports on the test set is typically larger than the error we get on the validation set by about 0.0014. When the validation set is added to the training set, RMSE on the test set is typically reduced by about 0.005.

Figure 3 (left panel) shows performance of the RBM and the RBM with Gaussian hidden units. The y-axis displays RMSE, and the x-axis shows the number of epochs. Clearly, the nonlinear model substantially outperforms its linear counterpart. Figure 3 (middle panel) also reveals that conditioning on rated/unrated information significantly improves model performance. It also shows (right panel) that, when using a conditional RBM, factoring the weight matrix leads to much faster convergence.

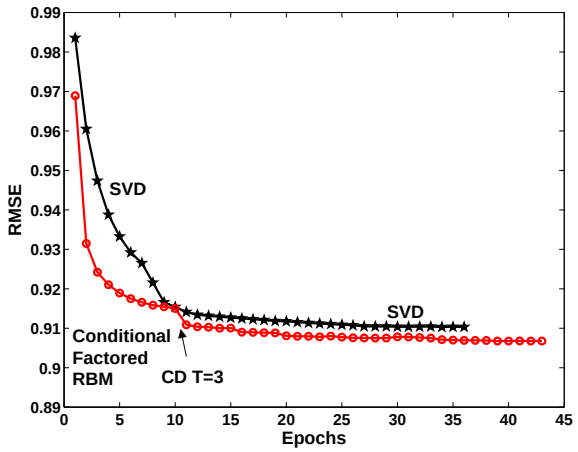

Figure 4. Performance of the conditional factored RBM vs. SVD with $C = 40$ factors. The y-axis displays RMSE (root mean squared error), and the x-axis shows the number of epochs, or passes through the entire training dataset.

7. Singular Value Decomposition (SVD)

SVD seeks a low-rank matrix $X = UV'$, where $U \in R^{N\times C}$ and $V \in R^{M\times C}$, that minimizes the sum-squared distance to the *fully observed* target matrix Y. The solution is given by the leading singular vectors of Y. In the collaborative filtering domain, most of the entries in Y will be missing, so the sum-squared distance is minimized with respect to the *partially observed* entries of the target matrix Y. Unobserved entries of Y are then predicted using the corresponding entries of X.

Let $X = UV'$, where $U \in R^{N\times C}$ and $V \in R^{M\times C}$ denote the low-rank approximation to the partially observed target matrix $Y \in R^{N\times M}$. Matrices U and V are initialized with small random values sampled from a zero-mean normal distribution with standard deviation 0.01. We minimize the following objective function:

$$f = \sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}\left(\mathbf{u_i}\mathbf{v_j}' - Y_{ij}\right)^2 + \lambda \sum_{ij} I_{ij}\left(\| \mathbf{u_i} \|_{Fro}^2 + \| \mathbf{v_j} \|_{Fro}^2 \right) \quad (15)$$

where $\| \cdot \|_{Fro}^2$ denotes the Frobenius norm, and I_{ij} is the indicator function, taking on value 1 if user i rated movie j, and 0 otherwise. We then perform gradient descent in U and V to minimize the objective function of Eq. 15.

To speed-up the training, we subdivided the Netflix data into mini-batches of size 100,000 (user/movie pairs), and updated the weights after each mini-batch. The weights were updated using a learning rate of 0.005, momentum of 0.9, and regularization parameter $\lambda = 0.01$. Regularization, particularly for the Netflix dataset, makes quite a significant difference in model performance. We also experimented with various values of C and report the results with $C = 40$, since it resulted in the best model performance on the validation set. Values of C in the range of $[20, 60]$ also give similar results.

We compared the conditional factored RBM with an SVD model (see Fig. 4). The conditional factored RBM slightly outperforms SVD, but not by much. Both models could potentially be improved by more careful tuning of learning rates, batch sizes, and weight-decay. More importantly, the errors made by various versions of the RBM are significantly different from the errors made by various versions of SVD, so linearly combining the predictions of several different versions of each method, using coefficients tuned on the validation data, produces an error rate that is well over 6% better than the Netflix's own baseline score.

8. Future extensions

There are several extensions to our model that we are currently pursuing.

8.1. Learning Autoencoders

An alternative way of using an RBM is to treat this learning as a *pretraining stage* that finds a good region of the parameter space (Hinton & Salakhutdinov, 2006). After pretraining, the RBM is "unrolled" as shown in figure 5 to create an autoencoder network in which the stochastic activities of the binary "hidden" features are replaced by deterministic, real-valued probabilities. Backpropagation, using the squared error objective function, is then used to fine-tune the weights for optimal reconstruction of each user's ratings. However, overfitting becomes an issue and more careful model regularization is required.

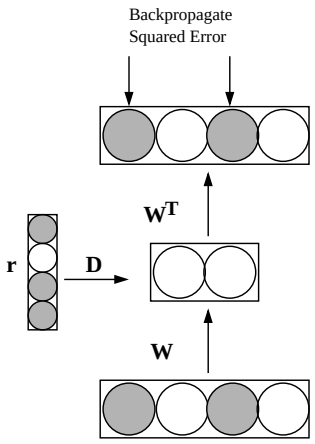

Figure 5. The "unrolled" RBM used to create an autoencoder network which is then fine-tuned using backpropagation of error derivatives.

8.2. Learning Deep Generative Models

Recently, (Hinton et al., 2006) derived a way to perform fast, greedy learning of deep belief networks one layer at a time, with the top two layers forming an undirected bipartite graph which acts as an associative memory.

The learning procedure consists of training a stack of RBM's each having only one layer of latent (hidden) feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack.

An important aspect of this layer-wise training procedure is that, provided the number of features per layer does not decrease, each extra layer increases a lower bound on the log probability of data. So layer-by-layer training can be recursively applied several times[3] to learn a deep, hierarchical model in which each layer of features captures strong high-order correlations between the activities of features in the layer below.

Learning multi-layer models has been successfully applied in the domain of dimensionality reduction (Hinton & Salakhutdinov, 2006), with the resulting models significantly outperforming Latent Semantic Analysis, a well-known document retrieval method based on SVD (Deerwester et al., 1990). It has also been used for modeling temporal data (Taylor et al., 2006; Sutskever & Hinton, 2006) and learning nonlinear embeddings (Salakhutdinov & Hinton, 2007). We are currently exploring this kind of learning for the Netflix data. For classification of the MNIST digits, deep networks reduce the error significantly (Hinton & Salakhutdinov, 2006) and our hope is that they will be similarly helpful for the Netflix data.

[3]In fact, one can proceed learning recursively for as many layers as desired.

9. Summary and Discussion

We introduced a class of two-layer undirected graphical models (RBM's), suitable for modeling tabular or count data, and presented efficient learning and inference procedures for this class of models. We also demonstrated that RBM's can be successfully applied to a large dataset containing over 100 million user/movie ratings.

A variety of models have recently been proposed for minimizing the loss corresponding to a specific probabilistic model (Hofmann, 1999; Canny, 2002; Marlin & Zemel, 2004). All these probabilistic models can be viewed as graphical models in which hidden factor variables have directed connections to variables that represent user ratings. Their major drawback(Welling et al., 2005) is that exact inference is intractable due to explaining away, so they have to resort to slow or inaccurate approximations to compute the posterior distribution over hidden factors.

Instead of constraining the rank or dimensionality of the factorization $X = UV'$, i.e. the number of factors, (Srebro et al., 2004) proposed constraining the norms of U and V. This problem formulation termed "Maximum Margin Matrix Factorization" could be seen as constraining the overall "strength" of factors rather than their number. However, learning MMMF requires solving a sparse semi-definite program (SDP). Generic SDP solvers run into difficulties with more than about 10,000 observations (user/movie pairs), so

direct gradient-based optimization methods have been proposed in an attempt to make MMMF scale up to larger problems. The Netflix data set, however, contains over 100 million observations and none of the above-mentioned approaches can easily deal with such large data sets.

Acknowledgments

We thank Vinod Nair, Tijmen Tieleman and Ilya Sutskever for many helpful discussions. We thank Netflix for making such nice data freely available and for providing a free and rigorous model evaluation service.

References

Canny, J. F. (2002). Collaborative filtering with privacy via factor analysis. *SIGIR* (pp. 238–245). ACM.

Carreira-Perpinan, M., & Hinton, G. (2005). On contrastive divergence learning. *10th Int. Workshop on Artificial Intelligence and Statistics (AISTATS'2005)*.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., & Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, *41*, 391–407.

Hinton, & Salakhutdinov (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, *14*, 1711–1800.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 1527–1554.

Hofmann, T. (1999). Probabilistic latent semantic analysis. *Proceedings of the 15th Conference on Uncertainty in AI* (pp. 289–296). San Fransisco, California: Morgan Kaufmann.

Marlin, B., & Zemel, R. S. (2004). The multiple multiplicative factor model for collaborative filtering. *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*. ACM.

Neal, R. M. (1993). *Probabilistic inference using Markov chain Monte Carlo methods* (Technical Report CRG-TR-93-1). Department of Computer Science, University of Toronto.

Salakhutdinov, R., & Hinton, G. E. (2007). Learning a nonlinear embedding by preserving class neighbourhood structure. *AI and Statistics*.

Srebro, N., & Jaakkola, T. (2003). Weighted low-rank approximations. *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA* (pp. 720–727). AAAI Press.

Srebro, N., Rennie, J. D. M., & Jaakkola, T. (2004). Maximum-margin matrix factorization. *Advances in Neural Information Processing Systems*.

Sutskever, I., & Hinton, G. E. (2006). *Learning multilevel distributed representations for high-dimensional sequences* (Technical Report UTML TR 2006-003). Dept. of Computer Science, University of Toronto.

Taylor, G. W., Hinton, G. E., & Roweis, S. T. (2006). Modeling human motion using binary latent variables. *Advances in Neural Information Processing Systems*. MIT Press.

Welling, M., Rosen-Zvi, M., & Hinton, G. (2005). Exponential family harmoniums with an application to information retrieval. *NIPS 17* (pp. 1481–1488). Cambridge, MA: MIT Press.