SOME APPROACHES OF BUILDING RECOMMENDATION SYSTEMS

A Project Report Submitted

for the Courses

MA498 Project I

MA499 Project II

by

Raghav Somani

(Roll No. 130123029)

and

Utkarsh Gupta

(Roll No. 130123040)

to the

DEPARTMENT OF MATHEMATICS

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, INDIA

April 2017

CERTIFICATE

This is to certify that the work contained in this project report entitled "**Some approaches of building Recommendation Systems**" submitted by **Raghav Somani** (**Roll No.: 130123029**) and **Utkarsh Gupta** (**Roll No.: 130123040**) to Department of Mathematics, Indian Institute of Technology Guwahati towards the requirement of the courses **MA498 Project I** and **MA499 Project II** has been carried out by them under my supervision.

Guwahati - 781 039
April 2017

(Dr. Arabin Kumar Dey)
Project Supervisor

ABSTRACT

The project aims primarily at recommending users, items based on past rating behaviours. Given a sparse rating matrix, the goal is to come up with an ordering of items for the users. Recommendations can be generated by a wide range of algorithms. Regularized Matrix Factorization techniques are usually effective because they allow us to discover the latent features underlying the interactions between users and items. Order preservation above Matrix Factorization has been shown superior to Matrix Factorization where it is tried to fit the ratings and also preserve the relative ordering of ratings by transforming the rating matrix using learned monotonic functions σ's.

A practical drawback of these algorithms is getting good estimates of hyper-parameters involved in these algorithms. A naive approach is to validate hyper-parameters using grid searching and using the best one in the optimization framework. An alternative way is to get estimates of them using Emperical Bayes methods and using them in the optimization algorithm.

The project also discusses about a content based recommendation system which makes suggestions based on similarities between user and item feature vectors taken from the user and item factor matrices.

The project extends the notion of recommendation further to different items along with just user-movie recommendation. We use different Unsupervised Deep Learning methods to learn important high level features of item representations.

Items like Documents, Images and user-movie ratings have different representational characterstics and with the help of data, Deep Learning helps extracting relevant features that can be used to provide substantial information about a user's behaviour.

Documents or Natural Languages are known to be a discrete combinatorial system and using only language based rules do not provide scalability. Comparison in continuous space becomes less relevant because of its discrete nature, therefore learning good high level continuous features becomes important. Images on the other hand are very high dimensional representation of objects which have inherent localized structures that requires learning for relevant comparision. Images are spacially invariant to translation and small linear transformations like stretching and skewing. Along with it, they have a good amount of spatial pixel correlation which do not allow good comparison. Movie recommendation depends on the estimation of ratings provided by a user which are characterized by the "taste" of the user for different genres/kinds of movies. So the extraction of high level behavioural features is important.

We use unsupervised Deep Learning to learn such high level features in Documents, Images and User-movie ratings. These features become useful when we further use vector space models to get good recommendations.

Contents

List of Figures

List of Tables

Chapter 1

Collaborative Filtering using Matrix Factorization

Some of the most successful realizations of latent factor models are based on Matrix Factorization. In its basic form, Matrix Factorization characterizes both items and users by real vectors of factors inferred from item rating patterns represented by k dimensional vectors $\mathbf{u}_i$ and $\mathbf{v}_j$ corresponding to i^{th} user and j^{th} item respectively. High correspondence between item and user factors leads to a recommendation. The most convenient data is high-quality explicit feedback, which includes explicit input by users regarding their interest in the items. We refer the explicit user feedback as ratings. Usually, explicit feedback comprises a sparse matrix $\mathbf{M}$, since any single user is likely to have rated only a very small percentage of possible items. Characterizing the feedback linearly, it approximates the ratings m_{ij} as the dot product of $\mathbf{u}_i$ and $\mathbf{v}_j$ such that the estimate $\hat{m}_{ij} = \mathbf{u}_i^T.\mathbf{v}_j$. The major challenge is computing the mapping of each item and user to factor vectors $\mathbf{u}_i$, $\mathbf{v}_j \in \mathbb{R}^k$.

1.1 Learning latent features

Definition 1.1.1. $n :=$ Number of unique users,

$p :=$ Number of unique items,

$k :=$ Number of latent feature,

$\mathbf{M} :=$ Sparse rating matrix of dimension $(n \times p)$ where the $(i,j)^{th}$ element of the rating m_{ij} is given by user i to item j.

$\mathbf{U} :=$ The user feature matrix of dimension $(n \times k)$ where row i represents the user feature vector $\mathbf{u}_i$.

$\mathbf{V} :=$ The item feature matrix of dimension $(p \times k)$ where row j represents the item feature vector $\mathbf{v}_j$.

$\mathcal{L} :=$ The loss function which is to be minimized.

λ_1 & $\lambda_2 :=$ User and item hyper-parameters in the regularized Loss function $\mathcal{L}$.

$||\cdot||_F :=$ Frobenius norm for matrices.

$\kappa :=$ The set of user-item indices in the sparse rating matrix $\mathbf{M}$ for which the ratings are known.

$\kappa_i :=$ The set of indices of item indices for user i for which the ratings are known.

$\kappa_j :=$ The set of indices of user indices for item j who rated it.

To learn the latent feature vectors $\mathbf{u}_i$ and $\mathbf{v}_j$, the system minimizes the regularized loss on the set of known ratings.

$$\mathcal{L}(\mathbf{M}; \mathbf{U}, \mathbf{V}, \lambda_1, \lambda_2) = \frac{1}{|\kappa|} \sum_{i,j \in \kappa} (m_{ij} - \mathbf{u}_i^T . \mathbf{v}_j)^2 + \lambda_1 ||\mathbf{U}||_F^2 + \lambda_2 ||\mathbf{V}||_F^2 \quad (1.1)$$

This Loss function is a biconvex function in $\mathbf{U}$ and $\mathbf{V}$ and can be iteratively optimized by regularized least square methods keeping the hyper-

parameters fixed.

The user and item feature matrices are first heuristically initialized using normal random matrices with iid. entries such that the product $\mathbf{U}.\mathbf{V}^T$ has a mean of 3 and variance 1.

The iteration is broken into 2 steps until test loss convergence. The first step computes the regularized least squares estimate for each of the user feature vectors $\mathbf{u}_i$ from their known ratings.

The second step computes the regularized least squares estimate for each of the item feature vectors $\mathbf{v}_j$ from their known ratings.

The first step minimizes the below expression keeping $\mathbf{V}$ fixed.

$$||\mathbf{M}_{i,\kappa_i} - \mathbf{V}_{\kappa_i}.\mathbf{u}_i||^2 + \lambda_1||\mathbf{u}_i||^2 \qquad \forall i = 1, 2, \ldots, n. \tag{1.2}$$

The second step minimizes the below expression keeping $\mathbf{U}$ fixed.

$$||\mathbf{M}_{\kappa_j,j} - \mathbf{U}_{\kappa_j}.\mathbf{v}_j||^2 + \lambda_2||\mathbf{v}_j||^2 \qquad \forall j = 1, 2, \ldots, p. \tag{1.3}$$

The normal equations corresponding to the regularized least squares solution for user feature vectors are

$$(\mathbf{V}_{\kappa_i}^T.\mathbf{V}_{\kappa_i} + \lambda_1\mathbf{I}_k).\mathbf{u}_i = \mathbf{V}_{\kappa_i}^T.\mathbf{M}_{i,\kappa_i} \qquad i = 1, 2, \ldots, n \tag{1.4}$$

The normal equations corresponding to the regularized least squares solution for item feature vectors are

$$(\mathbf{U}_{\kappa_j}^T.\mathbf{U}_{\kappa_j} + \lambda_2\mathbf{I}_k).\mathbf{v}_1 = \mathbf{U}_{\kappa_j}^T.\mathbf{M}_{\kappa_j,j} \qquad i = 1, 2, \ldots, n \tag{1.5}$$

Iteratively minimizing the loss function gives us a local minima (possibly

global).

1.2 Experiments

The benchmark datasets used in the experiments were the 3 Movie Lens dataset. They can be found publically at http://grouplens.org/datasets/movielens/ The experimental results are shown in table 1.1 below.

Dataset	n	p	Ratings	Minimum RMSE
MovieLens small	751	1616	100,000	0.989
MovieLens medium	5301	3682	1,000,000	0.809
Movielens large	62007	10586	10,000,000	0.834

Table 1.1: RMSE of Matrix Factorization algorithm on Test data

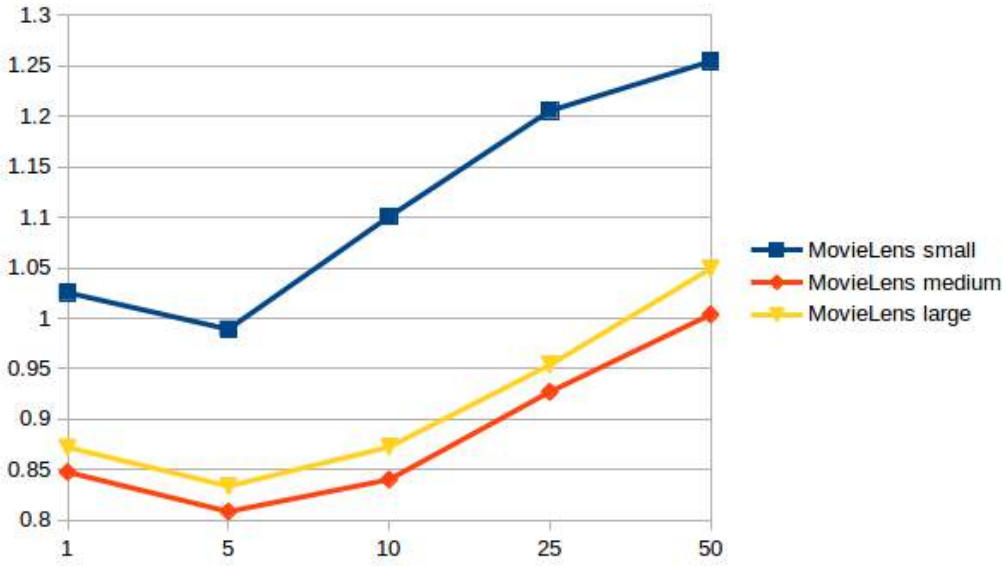

Figure 1.1: k vs RMSE of Matrix Factorization algorithm on Test data

1.3 Pros and Cons

The naive Matrix Factorization algorithm disussed in this chapter has certain pros and cons. Starting with the Pros

1. Is able to represent each user and item into a feature vectors that can be further used into several individualistic analysis.

2. Tries to approximate the ratings in a iteratve least square manner. Since the loss function is non-convex but still is bi-convex, we have an iterative algorithm that results in a good local minima.

The cons of the algorithm are

1. The value fitting aproach does not bother about maintaining the order of the ratings, which leads to instability.

2. The algorithm is sensitive with respect to hyper-parameters. So getting a good set of hyper-parameters is important and computationally very intensive.

Chapter 2

Matrix Factorization with Order preservation

Despite of acheiving good RMSE values, a simple matrix factorization misses an important criteria that is very important in providing good recommendations to users. Lets take an example.
Lets look at 2 prediction ratings to a training data [2.7, 3, 3.4].
Prediction 1: [0.10, 0.12, 42],
Prediction 2: [3.0, 3.2, 3.1].
A least square approach would choose Prediction 2. But if we desire to have a better ordering, Prediction 1 should be the correct choice. Order preserving matrix factorization uses monotonically transformed values instead of raw ratings to preserve ordering.
A non-parametric monotonic function σ is learned iteratively and is used to transform the rating space to another space that preserves ordering (monotonic property of the function). There are 3 ways in which this can be exploited :-

1. 1-σ - A universal monotonic function is used for every user.

2. N-σ - Different monotonic functions are used for every user.

3. K-σ - The users are clustered into K cluster and each cluster gets a different monotnic function.

2.1 Learning the transformation

Definition 2.1.1. $\sigma :=$ non parametric monotonic function applied to the rating matrix $\mathbf{M}$.

$\Sigma :=$ Set of all unique σ functions for every user.

$K :=$ Number of clusters of users.

$c_i :=$ The cluster id of user i.

$\hat{m}_{ij} := \mathbf{u}_i^T.\mathbf{v}_j$.

$\mathcal{R}^n \downarrow := \{\mathbf{x} \in \mathbb{R}^n : x_1 \leq x_2 \leq \dots x_n\}$. It is a convex cone and preserves isotonicity. It can be expressd as an image set of $\mathbf{x} \in \{\mathbb{R}_+^{n-1}\} \times \mathbb{R}$ under a transformation obtained by an upper triangular matrix $\mathbf{U}$ with non-negative entries.

$\tilde{\mathbf{M}} := \sigma(\mathbf{M})$ such that $\sigma_i(m_{ij}) = \tilde{m}_{ij}$ for $(i,j) \in \kappa$.

After applying the transformation, the new Loss function becomes

$$\mathcal{L}(\mathbf{M}; \mathbf{U}, \mathbf{V}, \Sigma, \lambda_1, \lambda_2) = \frac{1}{|\kappa|} \sum_{i,j \in \kappa} (\sigma_i(m_{ij}) - \mathbf{u}_i^T.\mathbf{v}_j)^2 + \lambda_1 ||\mathbf{U}||_F^2 + \lambda_2 ||\mathbf{V}||_F^2 \tag{2.1}$$

For 1-σ, $\sigma_i = \sigma \ \forall i = 1, 2, \dots, n$

For K-σ, $\sigma_i = \sigma_{c_i} \ \forall i = 1, 2, \dots, n$.

Remark 2.1.2. In the K-σ algorithm, clustering introduces 2 different regularization hyper-parameters, to be discussed later.

The iterative minimization is same as that for Matrix Factorization but

with an extra step in every iteration which involves finding the best monotonic function σ that further minimizes the squared loss.
This transformation should also preserve order. That is, the order of $\tilde{\mathbf{M}}_{i,\kappa_i}$ has the same order that of $\mathbf{M}_{i,\kappa_i}$ and the same time it minimizes the squared loss with $\hat{\mathbf{M}}_{i,\kappa_i}$ as target.

Theorem 2.1.3. *If* $\sum_{i=1}^{N} w_i = 1$, *and* $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} w_i y_i$, *then*

$$\sum_{i=1}^{N} w_i (z - y_i)^2 = (z - \bar{y})^2 + \sum_{i=1}^{N} w_i (y_i - \bar{y})^2 \tag{2.2}$$

Proof.

$$\begin{aligned}
\sum_{i=1}^{N} w_i (z - y_i)^2 &= \sum_{i=1}^{N} w_i (z - \bar{y} + \bar{y} - y_i)^2 \\
&= \sum_{i=1}^{N} w_i (z - \bar{y})^2 + \sum_{i=1}^{N} w_i (y_i - \bar{y})^2 + 2\sum_{i=1}^{N} w_i (z - \bar{y})(y_i - \bar{y}) \\
&= (z - \bar{y})^2 \sum_{i=1}^{N} w_i + \sum_{i=1}^{N} w_i (y_i - \bar{y})^2 + 2(z - \bar{y}) \sum_{i=1}^{N} (y_i - \bar{y}) \\
&= (z - \bar{y})^2 + \sum_{i=1}^{N} w_i (y_i - \bar{y})^2
\end{aligned}$$

□

The optimization problem for this step is defines by

$$\tilde{\mathbf{M}}_{i,\kappa_i} = \underset{\mathbf{x} \sim \mathbf{M}_{i,\kappa_i}}{\arg\min} \; ||\mathbf{x} - \hat{\mathbf{M}}_{i,\kappa_i}||_2^2 \qquad \forall i = 1, 2, \ldots, n. \tag{2.3}$$

where $\sim$ deotes "has the same order as" relation.
For a given i, let N_r = Number of ratings that user has rated as r, i.e.,

$N_r = \sum_{j \in \kappa_i} I[\mathbf{M}_{ij} = r]$. Equation (2.2) can be reordered and written as

$$\begin{aligned}
\tilde{\mathbf{M}}_{i,\kappa_i} &= \underset{\mathbf{x} \in \mathcal{R}^5 \downarrow}{\arg\min} \sum_{r=1}^{r=5} \sum_{j \ni m_{ij}=r} (x_r - \hat{m}_{ij})^2 \qquad \forall i = 1, 2, \ldots, n. \\
&= \underset{\mathbf{x} \in \mathcal{R}^5 \downarrow}{\arg\min} \sum_{r=1}^{r=5} N_r \sum_{j \ni m_{ij}=r} \frac{1}{N_r} (x_r - \hat{m}_{ij})^2 \\
&= \underset{\mathbf{x} \in \mathcal{R}^5 \downarrow}{\arg\min} \sum_{r=1}^{r=5} N_r [(x_r - \bar{\hat{m}}_{ij}^r) + \frac{1}{N_r} \sum_{j \ni m_{ij}=r} (\hat{m}_{ij} - \bar{\hat{m}}_{ij}^r)^2] \\
&= \underset{\mathbf{x} \in \mathcal{R}^5 \downarrow}{\arg\min} \sum_{r=1}^{r=5} \frac{N_r}{N} (x_r - \bar{\hat{m}}_{ij}^r) \qquad (2.4)
\end{aligned}$$

where $N = \sum_{r=1}^{5} N_r$ and $\bar{\hat{m}}_{ij}^r = \frac{1}{N_r} \sum_{j \ni m_{ij}=r}^{N_r} \hat{m}_{ij}$.

Therefore the optimization equation (2.3) is same as optimization (2.5) and is of a much lower dimension, making it suitable for faster numerical optimization.

Now we can easily see that the optimization equation (2.4) can be easily solved using weighted isotonic regression which is a semidefinite programming algorithm preserving order such that $\mathbf{x} \in \mathcal{R}^5 \downarrow$. After finding $\mathbf{x}$ we can construct $\tilde{\mathbf{M}}_{i,\kappa_i}$ such that $\tilde{\mathbf{M}}_{ij} = x_r$ if $\mathbf{M}_{ij} = r$.

The above discussed optimization strategy is when each row is uniquely solved and so it is the basis of the N-σ algorithm.

For 1-σ algorithm, we need to have a unique function σ, so the vectors $\hat{\mathbf{M}}_{i,\kappa_i}$ are concatenated for each user i and solved all together.

The K-σ algorithm, starts with the N σ's obtained after N-σ algorithm. We then cluster the functions σ_i's in K clusters such that the row loss is decreased. That is, for each user, σ_{c_i} is chosen that is capable of reducing the loss the most. Every iteration from here on performs the U step, the V

step and finding a single function σ_{c_i} for every cluster using the 1-σ logic of concatenation. The cluster center functions are re-assigned and we continue until we converge.

2.2 Experiments

The definition of test loss is changed for comparability. Since we want the deviation from $\vec{M}$, we need to bring back the transformed values to the original rating space. Test loss is defined as

$$\mathcal{L}(\mathbf{M}; \mathbf{U}, \mathbf{V}, \Sigma) = \frac{1}{|\kappa'|} \sum_{i,j \in \kappa'} (m_{ij} - \sigma_i^{-1}(\mathbf{u}_i^T . \mathbf{v}_j))^2 \tag{2.5}$$

where κ' is the set of (i, j) of known validation data.

Dataset	n	p	1-σ	N-σ	K-σ
MovieLens small	751	1616	0.908	0.948	0.911
MovieLens medium	5301	3682	0.799	0.79	0.778
MovieLens large	62007	10586	0.806	0.807	0.799

Table 2.1: RMSE of 1-σ, N-σ and K-σ algorithms on Test data

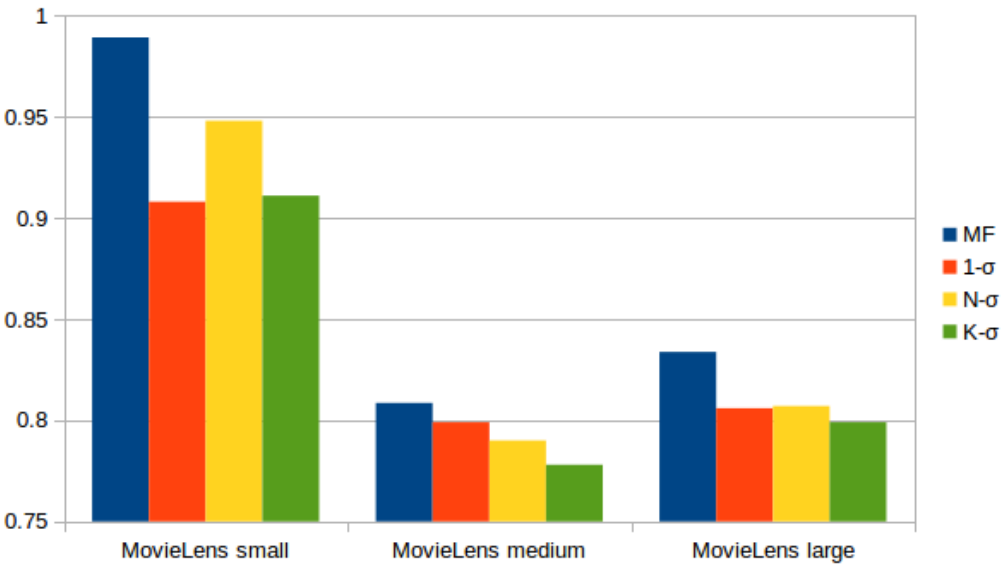

Figure 2.1: RMSE of 1-σ, N-σ and K-σ on Test data

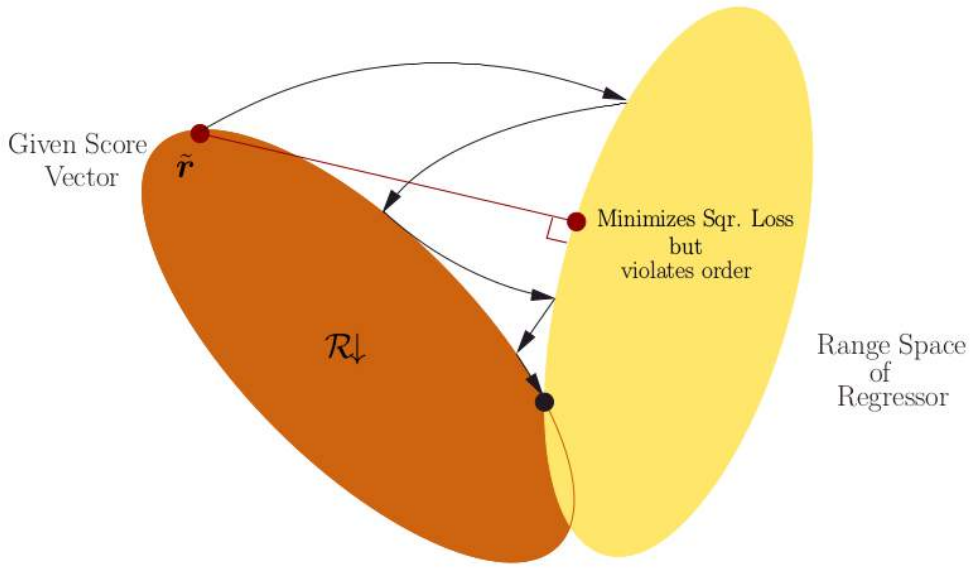

Figure 2.2: Range of Regressor - geometric illustration

2.3 Pros and Cons

The algorithms discussed in this chapter attempt to solve one of the major cons of the regularized Matrix Factorization algorithm. The pros are

1. These algorithms give prime importance to maintaining the order of the recommendation, therefore the ranks of items recommended to each user.

2. The K-σ algorithm allows us using clustering to get the benifits of both 1σ and N-σ.

3. From the nature of the functions used for transformation, we get to know about the spread of the rating behavours of the users. Example, the users who rate in extremes would majorly rate in 1 and 5. Moderately rating users would rate much in 3 and 4. Optimisic users would mostly rate in 4 and 5. So having user specific transformations allow us to take care of such rating behaviours by bringing then on a common ground.

Still we have a mojor con left unsolved.

1. Due to the complexity of the model, we have many hyper-parameters and getting the best set is computationally even more intensive. Grid searching on the MovieLens large dataset for the K-σ algorithm took 3 full days on a 4GB RAM, Intel i5 processor machine.

Chapter 3

A Bayesian framework for estimating hyper-parameters

Practical issues that usually come across in all the discussed algorithms is the choice of optimal hyperparameters. In the previous chapters, we can use Empirical Bayes inference to come up with reasonable estimates of λ_1 and λ_2 which otherwise are found using a brute-force method of grid searching.
In Empirical Bayes inference one is typically interested in sampling from the posterior distribution of a parameter with a hyper-parameter set to its maximum likelihood estimate. In the optimization setup, we are concerned about reducing the Loss function, where as in Bayesian setup, we maximize the posterior distribution corresponding to the Loss function.
The corresponding posterior distribution for the loss function in equation (1.1) is

$$\pi(\mathbf{U}, \mathbf{V} \mid \mathbf{M}, \lambda_1, \lambda_2) \propto \exp\{-\mathcal{L}(\mathbf{U}, \mathbf{V}, \mathbf{M}, \lambda_1, \lambda_2)\} \tag{3.1}$$

3.1 Empirical Bayes inference

Suppose that we observe a data $y \in Y$ generated from $f_{\theta,\lambda}(y)$. $f_{\theta,\lambda}(y)$ is the conditional distribution of y given that the parameter takes the value $(\theta, \lambda) \in \Theta \times \Lambda$. λ is treated as a hyper-parameter and assume that the conditional distribution of the parameter θ given $\lambda \in \Lambda$ is $\pi(\theta|\lambda)$. Therefore the joint distribution of y, θ given λ is thus

$$\pi(y, \theta \mid \lambda) = f_{\theta,\lambda}(y)\pi(\theta \mid \lambda) \tag{3.2}$$

The posterior distribution of θ given y,λ is then given by

$$\pi(\theta \mid y, \lambda) = \frac{\pi(y, \theta \mid \lambda)}{\pi(y \mid \lambda)} \tag{3.3}$$

where $\pi(y \mid \lambda) = \int \pi(y, \theta \mid \lambda) d\theta$.
There are 2 ways to handle hyper-parameters.

1. In fully Bayesian framework, a prior distribution $\pi(\lambda)$ is assumed for λ and we sample from the posterior

$$\pi(\theta \mid y) = \int \pi(\theta \mid y, \lambda)\omega(\lambda \mid y) d\lambda \tag{3.4}$$

   where $\omega(\lambda \mid y) \propto \pi(y \mid \lambda)\pi(\lambda)$ is the posterior distribution of λ given y. The major drawback is that the posterior $\omega(\lambda \mid y)$ can be sensitive to the choice of the prior $\pi(y)$.

2. Another method to deal with λ that has been proven to be very effective in practice is Empirical Bayes. The idea consists of first using the data y to propose and estimate for λ. This estimate is typically taken as the maximum likelihood estimate of $\hat{\lambda}$ of λ given θ.

$$\hat{\lambda} = \text{Argmax } \pi(y \mid \lambda) \tag{3.5}$$

And second, sample from the distribution $\pi(\theta \mid y, \hat{\lambda})$ using typically MCMC algorithms. Often, the marginal distribution $\pi(y \mid \lambda) = \int \pi(y, \theta \mid \lambda) d\theta$ is not available in closed form making the maximum likelihood estimation computationally challenging. In these challenging cases, EB procedures can be implemented using the EM algorithm as proposed by Casella (2001). This leads to a two-stage algorithm where in the first stage, a EM algorithm is used (each step of which typically requiring a fully converged MCMC sampling from $\pi(\theta \mid y, \lambda)$) to find $\hat{\lambda}$ and in a second stage, a MCMC sampler is run to sample from $\pi(\theta \mid y, \hat{\lambda})$.

Both these problems can be addressed simultaneously in a single simulation run, which results in a sampler that is more efficient than EM-followed by-MCMC. We use ∇_x to denote partial derivatives with respect to x. Let us define $l(\lambda \mid y) = \log \pi(y \mid \lambda)$ as the marginal log-likelihood of λ given y and define $h(\lambda \mid y) = \nabla_\lambda l(\lambda \mid y)$ its gradient. From here we have

$$\begin{aligned} h(\lambda \mid y) &= \nabla_\lambda l(\lambda \mid y) \\ &= \int \frac{\partial}{\partial \lambda} \log[f_{\theta,\lambda}(y)\pi(\theta \mid \lambda)]\pi(\theta \mid y, \lambda) d\theta \\ &= \int H(\lambda, \theta)\pi(\theta \mid y, \lambda) d\theta \end{aligned} \tag{3.6}$$

where

$$H(\lambda, \theta) := \nabla_\lambda \log(f_{\theta,\lambda}(y)\pi(\theta \mid \lambda) \tag{3.7}$$

In many cases, the likelihood does not depend on the hyper-parameters so that the function H simplifies further to

$$H(\lambda, \theta) = \nabla_\lambda \log \pi(\theta \mid \lambda) \tag{3.8}$$

We search for $\hat{\lambda} = \text{Argmax } \pi(y \mid \lambda)$ by solving the equation $h(\lambda \mid y) = 0$. If h is tractable then this equation can be easily solved analytically using iteratve methods. For example the gradient method would yield an iterative algorithm of the form

$$\lambda' = \lambda + ah(\lambda \mid y) \tag{3.9}$$

for a step size $a > 0$. If h is intractable, we naturally turn to stochastic approximation algorithms.

Suppose that we have at our disposal for each $\lambda \in \Lambda$, a transition kernel P_λ on Θ with invariant distribution $\pi(\theta \mid y, \lambda)$. We let $\{a_n, n \geq 0\}$ be a non-increasing sequence of positive numbers such that

$$\lim_{n\to\infty} a_n = 0 \qquad \sum a_n < \infty \qquad \sum a_n^2 < \infty \tag{3.10}$$

The Stochastic approximation algorithm proposed to maximize the function $l(\lambda \mid y)$ is as the following.

- Generate $\theta_{n+1} = P_{\lambda_n}(\theta, \cdot)$.
- Calculate $\lambda_{n+1} = \lambda_n + a_n H(\lambda_n, \theta_{n+1})$

This is iteratively done until the convergence of hyper-parameters. The transition kernel P_λ can be the MCMC sampler that samples from the distribution $\pi(\theta \mid y, \lambda)$. And the sequence $\{a_n, n \geq 0\}$ that works reasonably well is the a/n for some $a > 0$.

3.2 Empirical Bayes on Regularized Matrix Factorization algorithm

From Equation (1.1) and (3.1) we have

$$\pi(\mathbf{U}, \mathbf{V} \mid \mathbf{M}, \lambda_1, \lambda_2) \propto \exp\{-\frac{1}{|\kappa|}\sum_{i,j\in\kappa}(m_{ij}-\mathbf{u}_i^T.\mathbf{v}_j)^2\}\exp\{-\lambda_1||\mathbf{U}||_F^2\}\exp\{-\lambda_2||\mathbf{V}||_F^2\} \tag{3.11}$$

Comparing Equation (3.11) with Equation (3.2) it can be easily figured out that conditional distribution of $\mathbf{M}$ given $\mathbf{U}$, $\mathbf{V}$, λ_1 and λ_2 is

$$f_{\theta,\lambda} \propto \exp\{-\frac{1}{|\kappa|}\sum_{i,j\in\kappa}(m_{ij} - \mathbf{u}_i^T.\mathbf{v}_j)^2\} \tag{3.12}$$

and the priors on $\mathbf{U}$ and $\mathbf{V}$ given λ_1 and λ_2 are

$$\begin{aligned} \pi(\theta \mid \lambda) &= \pi(\theta_1 \mid \lambda_1)\pi(\theta_2 \mid \lambda_2) \\ &= \exp\{-\lambda_1||\mathbf{U}||_F^2\}\exp\{-\lambda_2||\mathbf{V}||_F^2\} \end{aligned} \tag{3.13}$$

We obtain the random samples of $\mathbf{U}$ and $\mathbf{V}$ in every iteration from the distribution $\pi(\mathbf{U}, \mathbf{V} \mid \mathbf{M}, \lambda_1, \lambda_2)$ using the Metropolis-Hastings algorithm.

Metropolis-Hasting Algorithm to generate parameters

Metropolis-Hastings Algorithm is a Markov Chain Monte Carlo method for obtaining a sequence of random samples from a probability distribution for which direct sampling is difficult.
Our target density at every iteration step i is $\pi(\mathbf{U}, \mathbf{V} \mid \mathbf{M}, \lambda_1^{(i)}, \lambda_2^{(i)})$.
Let $\mathbf{U}^{(i)}$ and $\mathbf{V}^{(i)}$ be the current iterates of the iteration sequence and $q(\mathbf{U}, \mathbf{V} \mid \mathbf{U}^{(i)}, \mathbf{V}^{(i)})$ be the proposal distribution.

The algorithmic steps to get the next iterate is

- Sample $(\mathbf{U}^*, \mathbf{V}^*) \sim q(\mathbf{U}, \mathbf{V} \mid \mathbf{U}^{(i)}, \mathbf{V}^{(i)})$.
- Calculate the acceptance probability

$$\rho((\mathbf{U}^{(i)}, \mathbf{V}^{(i)}), (\mathbf{U}, \mathbf{V})) = \min\left\{1, \frac{\pi(\mathbf{U}^*, \mathbf{V}^* \mid \mathbf{M}, \lambda_1^{(i)}, \lambda_2^{(i)}) q(\mathbf{U}^{(i)}, \mathbf{V}^{(i)} \mid \mathbf{U}^*, \mathbf{V}^*)}{\pi(\mathbf{U}^{(i)}, \mathbf{V}^{(i)} \mid \mathbf{M}, \lambda_1^{(i)}, \lambda_2^{(i)}) q(\mathbf{U}^*, \mathbf{V}^* \mid \mathbf{U}^{(i)}, \mathbf{V}^{(i)})}\right\} \tag{3.14}$$

  Since q is a symmetric proposal distribution, the acceptance probability reduces to

$$\rho((\mathbf{U}^{(i)}, \mathbf{V}^{(i)}), (\mathbf{U}, \mathbf{V})) = \min\left\{1, \frac{\pi(\mathbf{U}^*, \mathbf{V}^* \mid \mathbf{M}, \lambda_1^{(i)}, \lambda_2^{(i)})}{\pi(\mathbf{U}^{(i)}, \mathbf{V}^{(i)} \mid \mathbf{M}, \lambda_1^{(i)}, \lambda_2^{(i)})}\right\} \tag{3.15}$$

  Using Equation (3.11) the above expression for acceptance probability neatly reduces to

$$\rho((\mathbf{U}^{(i)}, \mathbf{V}^{(i)}), (\mathbf{U}, \mathbf{V})) = \min\left\{1, \frac{\exp\{-\mathcal{L}(\mathbf{U}^*, \mathbf{V}^*, \mathbf{M}, \lambda_1^{(i)}, \lambda_2^{(i)})\}}{\exp\{-\mathcal{L}(\mathbf{U}^{(i)}, \mathbf{V}^{(i)}, \mathbf{M}, \lambda_1^{(i)}, \lambda_2^{(i)})\}}\right\} \tag{3.16}$$

  and further to

$$\rho((\mathbf{U}^{(i)}, \mathbf{V}^{(i)}), (\mathbf{U}, \mathbf{V})) = \min\left\{1, e^{-\mathcal{L}(\mathbf{U}^*, \mathbf{V}^*, \mathbf{M}, \lambda_1^{(i)}, \lambda_2^{(i)}) + \mathcal{L}(\mathbf{U}^{(i)}, \mathbf{V}^{(i)}, \mathbf{M}, \lambda_1^{(i)}, \lambda_2^{(i)})}\right\} \tag{3.17}$$

- Set $\mathbf{U}^{(i+1)} = \mathbf{U}^*$ and $\mathbf{V}^{(i+1)} = \mathbf{V}^*$ with probability $\rho((\mathbf{U}^{(i)}, \mathbf{V}^{(i)}), (\mathbf{U}, \mathbf{V}))$, otherwise $\mathbf{U}^{(i+1)} = \mathbf{U}^{(i)}$ and $\mathbf{V}^{(i+1)} = \mathbf{V}^{(i)}$.

The algorithm that we apply, samples one iterate using Metropolis-Hasting algorithm and in the next step updates the hyper-parameters

$$\lambda_1^{(i+1)} = \lambda_1^{(i)} + a_n H(\lambda_1^{(i)}, (\mathbf{U}^{(i+1)}, \mathbf{V}^{(i+1)})) \tag{3.18}$$

and

$$\lambda_2^{(i+1)} = \lambda_2^{(i)} + a_n H(\lambda_2^{(i)}, (\mathbf{U}^{(i+1)}, \mathbf{V}^{(i+1)})) \tag{3.19}$$

From the definition of H, from Equation (3.8), we have

$$\begin{aligned} H(\lambda_1^{(i)}, (\mathbf{U}^{(i+1)}, \mathbf{V}^{(i+1)})) &= \nabla_{\lambda_1} \log \pi(\mathbf{U}^{(i+1)} \mid \lambda_1) \\ &= \nabla_{\lambda_1} \{-\lambda_1 ||\mathbf{U}^{(i+1)}||_F^2\} \\ &= -||\mathbf{U}^{(i+1)}||_F^2 \end{aligned} \tag{3.20}$$

similarly

$$\begin{aligned} H(\lambda_2^{(i)}, (\mathbf{U}^{(i+1)}, \mathbf{V}^{(i+1)})) &= \nabla_{\lambda_2} \log \pi(\mathbf{V}^{(i+1)} \mid \lambda_2) \\ &= \nabla_{\lambda_2} \{-\lambda_2 ||\mathbf{V}^{(i+1)}||_F^2\} \\ &= -||\mathbf{V}^{(i+1)}||_F^2 \end{aligned} \tag{3.21}$$

Also the sequence chosen is $\{a_n, n > 0\}$ is a/n for a suitable choice of a. Therefore, Equations (3.16) and (3.17) reduce to

$$\lambda_1^{(i+1)} = \lambda_1^{(i)} - \frac{a}{n} ||\mathbf{U}^{(i+1)}||_F^2 \tag{3.22}$$

and

$$\lambda_2^{(i+1)} = \lambda_2^{(i)} - \frac{a}{n} ||\mathbf{V}^{(i+1)}||_F^2 \tag{3.23}$$

The proposal distribution $q(\mathbf{U}, \mathbf{V} \mid \mathbf{U}^{(i)}, \mathbf{V}^{(i)})$ used is the Auto-Regressive process with lag 1 such that

$$u_{ik}* = \alpha u_{ik} + z_{ik}^{(1)} \qquad v_{jk}* = \alpha u_{jk} + z_{jk}^{(2)} \tag{3.24}$$

for $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, p$ and $k = 1, 2, \ldots, K$ where u_{ik} and v_{jk} are the elements of $\mathbf{U}$ and $\mathbf{V}$ respectively, $z_{ik}^{(1)}$ and $z_{jk}^{(2)}$ are iid. normal random numbers independent of $\mathbf{U}$ and $\mathbf{V}$ with mean 0 and variance σ_1^2 and σ_2^2 and $\alpha \in (-1, 1) \backslash \{0\}$.

3.3 Experiments

For the experiments, the MovieLens small dataset has been used with 751 users and 1616 movies. The algorithm discussed above uses many hyper-hyper-parameters. They are set to

Definition 3.3.1. $\lambda_1^{(0)} = \lambda_2^{(0)} = 10$ as from the Equation (3.22) and (3.23) we see that they decrease with every iteration.
$a = 5 \times 10^{-5}$
$\mathbf{U}^{(0)}$ and $\mathbf{V}^{(0)}$ are initialized with iid normal random numbers such that the elements of $\mathbf{U}^{(0)}\mathbf{V}^{(0)T}$ have mean 3 and variance 1.
$tol = 10^{-5}$ for breaking the loop if the change in both the hyper-parameters fall below tol.
$\alpha = 0.9$ and 0.5.
$(\sigma_1, \sigma_2) = (0.5, 0.5)$ and $(1, 1)$.

When $\sigma_1 = \sigma_2 = 0.5$ and α is set to 0.9, the hyper-parameters converge to $\hat{\lambda}_1 = 8.07549$ and $\hat{\lambda}_2 = 5.83292$.

When $\sigma_1 = \sigma_2 = 1$ and α is set to 0.5, the hyper-parameters converge to $\hat{\lambda}_1 = 8.07058$ and $\hat{\lambda}_2 = 5.85553$.

Using the estimated values of hyperparameters, the optimization algorithm was run and we obtain the Validation RMSE of 1.13196 and 1.13202. Comparing this with the RMSE corresponding to the best hyper-parameter values found using grid search, the validation RMSE goes as below as 0.989.

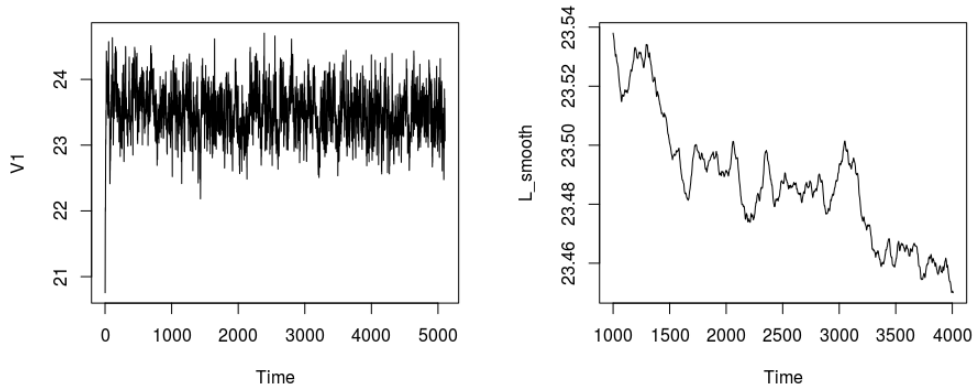

Figure 3.1: Iterations vs Loss - original and smoothened 1

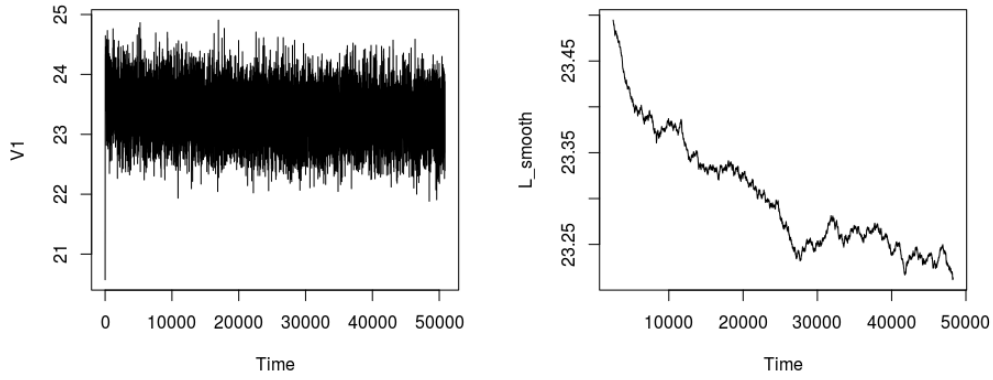

Figure 3.2: Iterations vs Loss - original and smoothened 2

3.4 Pros and Cons

This chapter tends to solve the major con of the algorithms discussed in the prebious chapters by giving a way to estimate the region of good hyper-parameters. Starting with the pros

1. The algorithm is fast and gives reasonalble estimates of the hyper-parameters. The similar idea can be used in the algorithms of chapter 2 to resolve the compute intensive hyper-parameter tuning.

2. This algorithm is also capable of giving the Bayes estimate of the parameters that are the factor matrices which can be used as an initial point for the optimization algorithm.

The cons are

1. We need to deal with some hyper-hyper parameters which makes the algorithm complex to deal with. To ensure good convergence the choice of a_n and the initial settings are crucial.

Chapter 4

Content based Recommendation System

Recommender systems are active information filtering systems which personalize the information coming to a user based on his interests, relevance of the information etc. A content based recommender works with data that the user provides, either explicitly (rating) or implicitly (clicking on a link). Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate. Each item is stored as a vector of its attributes in a k-dimensional space. The user profile vectors are also created and the similarity between an item and a user is also determined in a similar way.

4.1 Characterization of users and items

In chapter 1, we discused the regularized Matrix Factorization algorithm to best factorize the sparse rating matrix $\mathbf{M}$ into two low rank matrices $\mathbf{U}$ and

$\mathbf{V}$ so that $\mathbf{U}\mathbf{V}^T$ is able to best approximate $\mathbf{M}$ for $(i, j) \in \kappa$. The row vectors of $\mathbf{U}$ and $\mathbf{V}$, $\mathbf{u}_i$ and $\mathbf{v}_j$ represent the latent features of user i and item j and their dot product $\mathbf{u}_i^T \mathbf{v}_j$ approximates the rating m_{ij} that user i gives to item j.

The idea is that similar users rate similar items similarly. Both the feature matrices $\mathbf{U}$ and $\mathbf{V}$ are orthogonalized using PCA and 3 principal vectors explaining most of the variance are only chosen resulting in dimensionally reduced matrices $\hat{\mathbf{U}}$ and $\hat{\mathbf{V}}$.

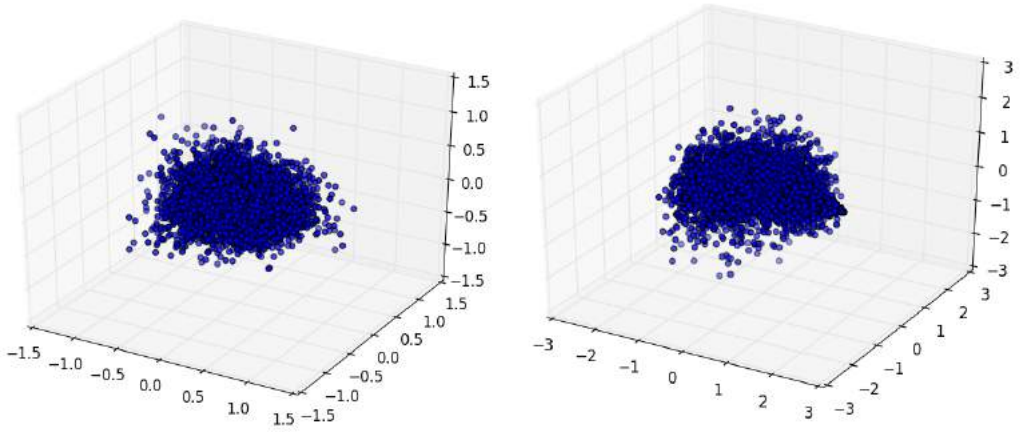

Figure 4.1: Dimensionally reduced user and item feature vectors

Both the reduced feauture matrices are clustered using the K-means algorithm.

In the K-means algorithm, our goal is to partition the data set into some number K of clusters, where we shall suppose for the moment that the value of K is given. Intuitively, we might think of a cluster as comprising a group of data points whose inter-point distances are small compared with the distances to points outside of the cluster. This optimization boils down to a minimization problem where we partition $\mathcal{S} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ into K sets

$\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_K$ which minimizes

$$\sum_{k=1}^{K} \sum_{\mathbf{x} \in \mathcal{S}_k} ||\mathbf{x} - \mu_k||^2 \tag{4.1}$$

This problem is NP-hard and so a general EM algorithm is used to iteratively minimize the objective function.
The initial centers of the K clusters $\{\mu_1, \mu_2, \ldots, \mu_K\}$ are first initialized randomly. Then the algorithm works iteratively by assigning clusters and then recomputing the centers.

- Assign each observation to the cluster whose mean yields the least within-cluster variance (WCV). Since the sum of squares is the squared Euclidean distance, this is intuitively the "nearest" mean.

$$\mathcal{S}_i^{(t)} = \{\mathbf{x}_p : ||\mathbf{x}_p - \mu_i^{(t)}|| \leq ||\mathbf{x}_p - \mu_j^{(t)}|| \quad \forall j, 1 \leq j \leq k\} \tag{4.2}$$

- Calculate the new means to be the centroids of the observations in the new clusters.

$$\mu_i^{(t+1)} = \frac{1}{|\mathcal{S}_i|} \sum_{\mathbf{x}_j \in \mathcal{S}_i} \mathbf{x}_j \tag{4.3}$$

K is chosen such that $K = \sup_k \{1 - \frac{WCV(k)}{TCV} \leq 0.8, \ k \in \mathbb{N}\}$, where $WCV(k)$ is the within cluster variance defined as

$$WCV(k) = \sum_{k=1}^{K} \sum_{\mathbf{x} \in \mathcal{S}_k} ||\mathbf{x} - \mu_k||^2 \tag{4.4}$$

and TCV is the total cluster variance defined as

$$TCV = \sum_{i=1}^{N} ||\mathbf{x}_i - \bar{\mathbf{x}}||^2 \tag{4.5}$$

where $\mathbf{x}_i$'s are feature vectors, $\mathcal{S}_k$'s are the K cluster sets. μ_k are the cluster centers and $\bar{\mathbf{x}}$ is the mean of all the feature vectors.

We pick each cluster of users with each cluster of items and assign the mean

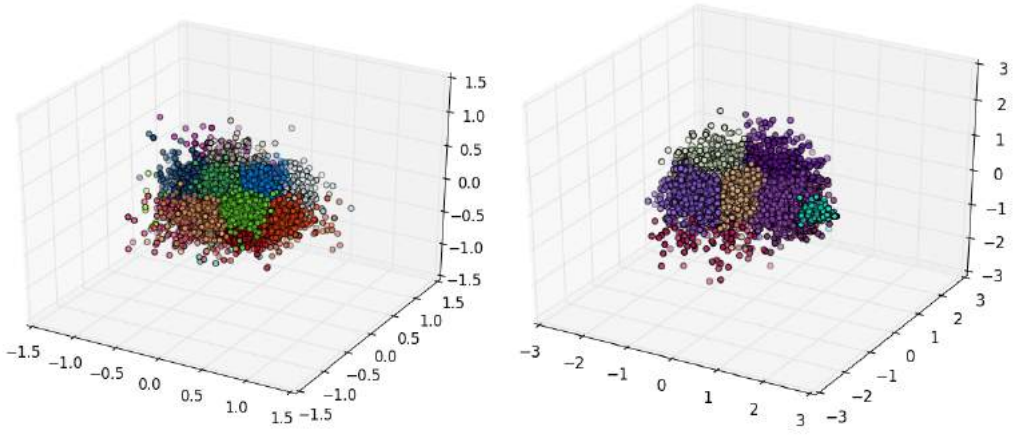

Figure 4.2: Clustering user and item feature vectors

rating to all the elements in that cluster. Mathematically, if $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{|\mathcal{P}|}$ are the clusters of users and $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_{|\mathcal{Q}|}$ are the clusters of items, then if $\mathbf{u}_i \in \mathcal{P}_{u_i}$ and $\mathbf{v}_j \in \mathcal{Q}_{v_j}$ then

$$\hat{m}_{ij} = \frac{\sum_{(i',j') \in \mathcal{P}_{u_i} \times \mathcal{Q}_{v_j}} m_{i'j'}}{|\mathcal{P}_{u_i} \times \mathcal{Q}_{v_j}|} \tag{4.6}$$

Implementing this on MovieLens medium dataset gives a validation accuracy of 0.9814 compared to 0.809 with the naive Matrix Factorization algorithm. Above this, the extra work that can be done would be to use k-Nearest Neighbours in every cluster pair $\mathcal{P}_{u_i} \times \mathcal{Q}_{v_j}$ for each item and each movie, and then take the mean rating. That is, for each $(i', j') \in \mathcal{P}_{u_i} \times \mathcal{Q}_{v_j}$ we consider only the k nearest users of user i' and k nearest neighbours of j' and assign $\hat{m}_{ij}$ with the mean of the $k \times k$ size principal sparse matrix.

4.2 Pros and Cons

The idea used in the algorithm discussed in this chapter is a bit different and it has its own pros and cons. The pros are

1. The algorithm does not deal with sensitive hyper-parameters. But still there is a scope of tuning and improving.

2. It is faster and shows comparable results if compared with the previously discussed algorithms.

3. The model is less complex and still shows to potential to come up with decent recommendations as we see from the expermental results.

The model is not complex but still suffers from a major con

1. The feature matrix used to start off with the clustering are taken from the Matrix Factorization algorithm, thus creating a dependency in the algorithmic flowchart.

Chapter 5

Feature Learning using Unsupervised Learning

In machine learning, feature learning or representation learning is a set of techniques that learn a feature: a transformation of raw data input to a representation that can be effectively exploited in machine learning tasks. Some of the most successful realizations of feature learning are based on Autoencoders, Restricted Boltzman Machine and Deep Belief Networks. We explore these constructs to learn important and relevant features in an unsupervised manner that further helps us in item and user based recommendations.

5.1 Autoencoders

Autoencoders are simple learning circuits which aim to transform inputs into outputs with the least possible amount of distortion.

To derive a fairly general framework, an n/p/n autoencoder is defined by a tuple
$n, p, m, \mathbb{F}, \mathbb{G}, \mathcal{A}, \mathcal{B}, \mathcal{X}, \Delta$ where:

1. $\mathbb{F}$ and $\mathbb{G}$ are sets.

2. n and p are positive integers. Here we consider primarily the case where $0 < p < n$.

3. $\mathcal{A}$ is a class of functions from $\mathbb{G}^p$ to $\mathbb{F}^n$.

4. $\mathcal{B}$ is a class of functions from $\mathbb{F}^n$ to $\mathbb{G}^p$.

5. $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ is a set of m training vectors in $\mathbb{F}^n$.

6. Δ is a dissimilarity of distortion function defined over $\mathbb{F}^n$.

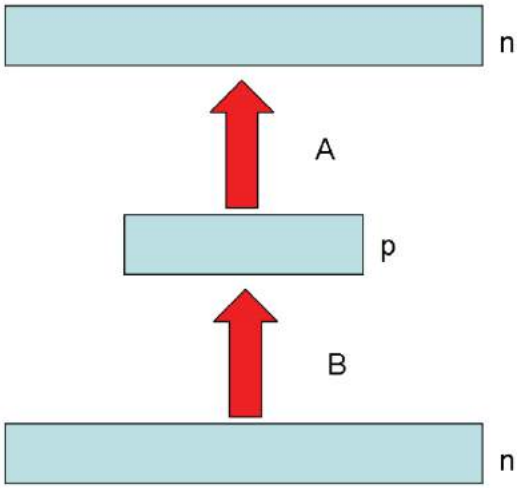

Figure 5.1: n/p/n Autoencoder (slideplayer.com)

For any $A \in \mathcal{A}$ and $B \in \mathcal{B}$, the autoencoder transforms an input vector $\mathbf{x} \in \mathbb{F}^n$ into an output vector $A \circ B(\mathbf{x}) \in \mathbb{F}^n$ (Figure 5.1). The corresponding autoencoder problem is to find $A \in \mathcal{A}$ and $B \in \mathcal{B}$ that minimizes the overall distortion function:

$$\min E(A, B) = \min_{A,B} \sum_{t=1}^{m} E(\mathbf{x}_t) = \min_{A,B} \Delta(A \circ B(\mathbf{x}_t), \mathbf{x}_t) \tag{5.1}$$

Note that $p < n$ corresponds to the regime where the autoencoder tries to implement some form of compression or feature extraction. Different kinds of autoencoders can be derived depending on the choice of sets $\mathbb{F}$ and $\mathbb{G}$, transforming classes $\mathcal{A}$ and $\mathcal{B}$, distortion function Δ as well as the presence of external constraints such as regularization. It must be noted that if $\mathbb{F} = \mathbb{G} = \mathbb{R}$ and if $\mathcal{A}$ and $\mathcal{B}$ are affine transformations then Equation 5.1 is reduced to the exact same optimization problem of Principal Component Analysis. Therefore providing a richer class of functions allows the autoencoder to model the non-linearities present in the data therefore allowing better feature extraction and dimension reduction.

We use stacked back-propagation based autoencoders, Restricted Boltzman Machine and Deep Belief Networks as the feature learning auto encoders from the high dimensional data.

5.1.1 Stacked Autoencoders

A stacked autoencoder is a neural network consisting of multiple layers of autoencoders in which the outputs of each layer is wired to the inputs of the successive layer. Stacking multiple layers and constructing deep autoencoders provides more representational power to model the high dimensional inputs over a low dimensional manifold on the same space. Back-propagation helps learning the parameters of the network therefore optimizing the loss function as described in Equation 5.1. As an example, a 5 layer autoencoder can be mathematically described as

$$\mathbf{x} = \mathbf{x}$$

$$\mathbf{h}_1 = f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h} = f(\mathbf{W}\mathbf{h}_1 + \mathbf{b})$$

$$\tilde{\mathbf{h}}_1 = \tilde{f}(\tilde{\mathbf{W}}\mathbf{h} + \tilde{\mathbf{b}})$$

$$\tilde{\mathbf{x}} = \tilde{f}_1(\tilde{\mathbf{W}}_1\tilde{\mathbf{h}}_1 + \tilde{\mathbf{b}}_1)$$

where $f_1, f, \tilde{f}, \tilde{f}_1$ are activation functions, $\mathbf{W}_1, \mathbf{W}, \tilde{\mathbf{W}}, \tilde{\mathbf{W}}_1$ and $\mathbf{b}_1, \mathbf{b}, \tilde{\mathbf{b}}, \tilde{\mathbf{b}}_1$ are weights and biases of affine transformations. The middle layer h is the encoded representation of x. Using back-propagation these parameters can be tuned to minimize the reconstruction loss. Similarly when we use convolutional operator and max-pooling instead of dot product with the different sets of parameters, known as kernels, we obtain a naive Convolutional Autoencoder.

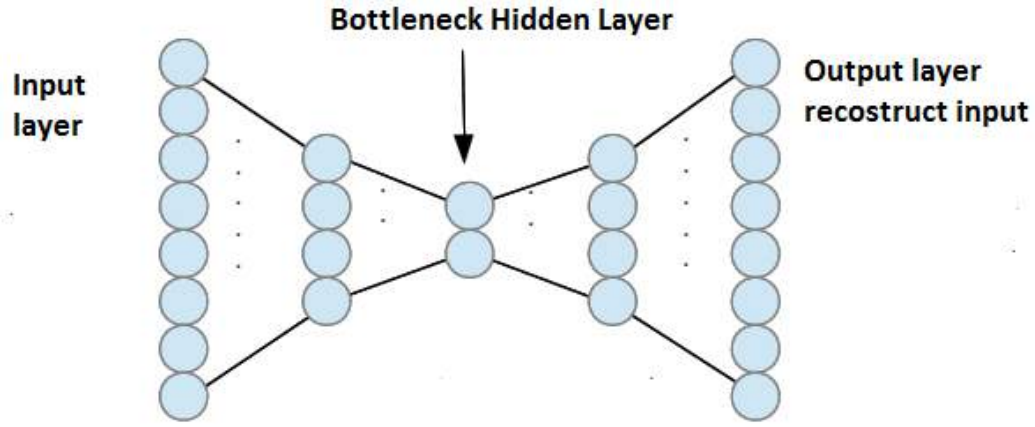

Figure 5.2: Naive Stacked Autoencoder (codingplayground.blogspot.in)

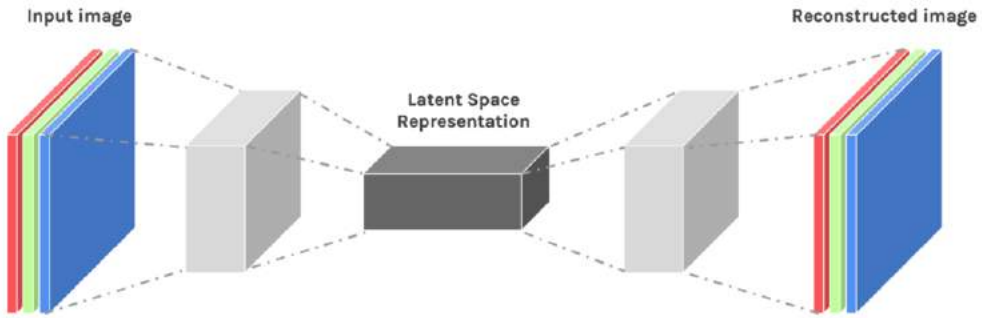

Figure 5.3: Stacked Convolutional Autoencoder (hackernoon.com)

5.1.2 Restricted Boltzman Machines

RBMs are undirected graphical models that define a distribution over the input vector $\mathbf{x} \in \{0,1\}^n$. We model this distribution using a layer of binary hidden units $\mathbf{h} \in \{0,1\}^p$. RBM is an energy based model and its energy function is defined as

$$E(\mathbf{x},\mathbf{h}) = -\mathbf{h}^T\mathbf{W}\mathbf{x} - \mathbf{c}^T\mathbf{x} - \mathbf{b}^T\mathbf{x} \tag{5.2}$$

$$= -\sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \tag{5.3}$$

Therefore its pdf becomes

$$p(\mathbf{x},\mathbf{h}) = \exp(-E(\mathbf{x},\mathbf{h}))/Z \tag{5.4}$$

where Z is the normalizing constant, also known as the partition function. $Z = \sum_{\mathbf{x},\mathbf{h}} \exp(-E(\mathbf{x},\mathbf{h}))$. Because there are no quadratic terms of $\mathbf{x}$ and $\mathbf{h}$ so the Boltzman machine is restricted enforcing no interaction within any

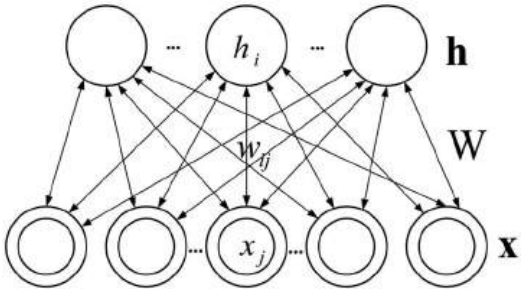

Figure 5.4: Restricted Boltzman Machine

layer. Conditional distributions are defined as

$$\begin{aligned} p(\mathbf{h} \mid \mathbf{x}) &= \prod_j p(h_j = 1 \mid \mathbf{x}) \\ p(h_j = 1 \mid \mathbf{x}) &= \frac{1}{1 + \exp(-(b_j + \mathbf{W}_{j:}\mathbf{x}))} \\ &= \sigma(b_j + \mathbf{W}_{j:}\mathbf{x}) \qquad (5.5) \\ p(\mathbf{x} \mid \mathbf{h}) &= \prod_k p(x_k = 1 \mid \mathbf{h}) \\ p(x_k = 1 \mid \mathbf{h}) &= \frac{1}{1 + \exp(-(c_k + \mathbf{h}^T\mathbf{W}_{:k}))} \\ &= \sigma(c_k + \mathbf{h}^T\mathbf{W}_{:k}) \qquad (5.6) \end{aligned}$$

We try to minimize the average Negative Log-Likelihood Loss of the input vector.

$$\frac{1}{T}\sum_t l(f(\mathbf{x}_t)) = \frac{1}{T}\sum_t -\log p(\mathbf{x}_t) \qquad (5.7)$$

The partial derivative of the loss function with respect to the parameters can be shown to be given by

$$\frac{\partial - \log p(\mathbf{x}_t)}{\partial \theta} = \mathbb{E}_{\mathbf{h}}\left[\frac{\partial E(\mathbf{x}_t, \mathbf{h})}{\partial \theta} \mid \mathbf{x}_t\right] - \mathbb{E}_{\mathbf{x},\mathbf{h}}\left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta}\right] \qquad (5.8)$$

The conditional expectation is tractable but the unconditional expectation of both $\mathbf{x}$ and $\mathbf{h}$ are not. The problem is addressed using Persistent Contrastive Divergence learning algorithm that provides an estimate of both the expectations. The derivatives of the energy function with respect to the parameters can be computed and we obtain the following update rule of Stochastic Gradient Descent as

$$\begin{aligned} \mathbf{W} &\leftarrow \mathbf{W} + \alpha(\mathbf{h}(\mathbf{x}_t)\mathbf{x}^T - \mathbf{h}(\tilde{\mathbf{x}}_t)\tilde{\mathbf{x}}^T) \\ \mathbf{b} &\leftarrow \mathbf{b} + \alpha(\mathbf{h}(\mathbf{x}_t) - \mathbf{h}(\tilde{\mathbf{x}})) \\ \mathbf{c} &\leftarrow \mathbf{c} + \alpha(\mathbf{x}_t - \tilde{\mathbf{x}}) \end{aligned} \tag{5.9}$$

where $\mathbf{h}(\mathbf{x}) = \sigma(\mathbf{b} + \mathbf{W}\mathbf{x})$ and $\tilde{\mathbf{x}}$ is the k step forward sample of $\mathbf{x}_t$ obtained from Gibb's sampling using the conditional distributions $p(\mathbf{h} \mid \mathbf{x})$ and $p(\mathbf{x} \mid \mathbf{h})$ as described in Equation 5.5 and Equation 5.6.

5.1.3 Deep Belief Networks

A DBN is a generative model, which has both directed and undirected interactions that constitute the input layer and the hidden layers. The top 2 layers of the DBN form a RBM modelling $p(\mathbf{h}_{l-1}, \mathbf{h}_l)$. The other layers form a Bayesian Network with directed interactions modelling $p(\mathbf{h}_{k-1} \mid \mathbf{h}_k)$ which take the form

$$p(\mathbf{h}_{k-1} \mid \mathbf{h}_k) = \sigma(\mathbf{b}_{k-1} + \mathbf{W}_k^T \mathbf{h}_k) \tag{5.10}$$

In the above equations $\mathbf{h}_0 = \mathbf{x}$. More specifically, the joint distribution over the input layer and the hidden layers can be represented as

$$p(\mathbf{x}, \mathbf{h}_1, \ldots, \mathbf{h}_l) = p(\mathbf{h}_{l-1}, \mathbf{h}_l) \prod_{k=1}^{l-1} p(\mathbf{h}_{k-1} \mid \mathbf{h}_k) \tag{5.11}$$

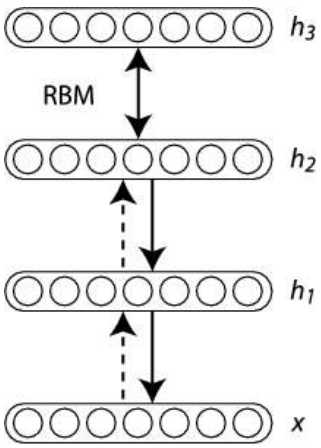

Figure 5.5: Deep Belief Networks (iro.umontreal.ca)

Training a DBN is a hard problem but it turns out that a good initialization can play a crucial role in the quality of results. Greedy layerwise pre-training procedure is used to initialize the weights of the DBN. Working around it with an example of pre-trainng of a 3 hidden layer DBN, we first start with a 1 hidden layer DBN which is an RBM. It is then used to initialize a 2 hidden layer DBN keeping the previous layer weights fixed. This process is continued untill we construct a 3 hidden layer DBN or in general, a l hidden layer DBN.

If we train a 1 hidden layer DBN that is an RBM, we model $p(\mathbf{x})$ as

$$p(x) = \sum_{\mathbf{h}_1} p(\mathbf{x}, \mathbf{h}_1)$$

Now $p(\mathbf{x}, \mathbf{h}_1) = p(\mathbf{x} \mid \mathbf{h}_1)p(\mathbf{h}_1)$ and

$$p(\mathbf{x}, \mathbf{h}_1) = p(\mathbf{x} \mid \mathbf{h}_1) \sum_{\mathbf{h}_2} p(\mathbf{h}_1, \mathbf{h}_2)$$

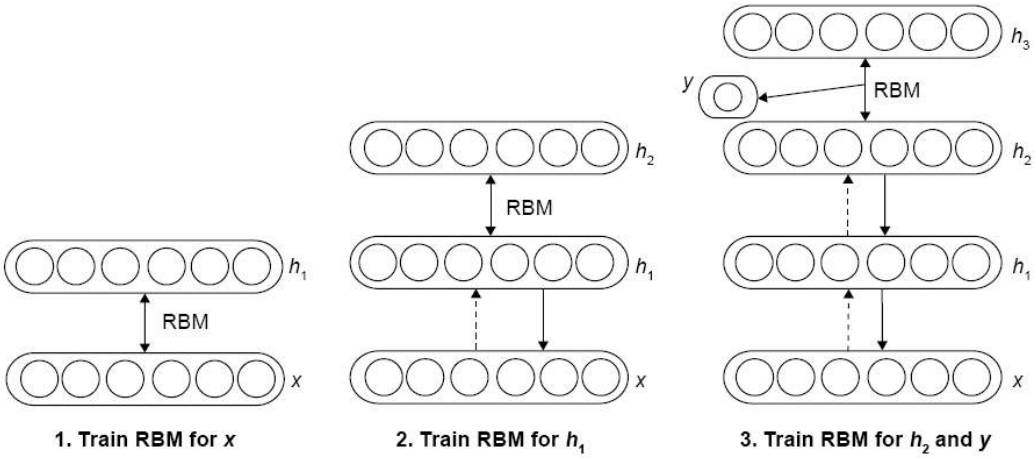

Figure 5.6: Greedy Layerwise pre-training a Deep Belief Networks (deeplearning.net)

Now $p(\mathbf{h}_1, \mathbf{h}_2) = p(\mathbf{h}_1 \mid \mathbf{h}_2)p(\mathbf{h}_2)$ and

$$p(\mathbf{h}_1, \mathbf{h}_2) = p(\mathbf{h}_1 \mid \mathbf{h}_2) \sum_{\mathbf{h}_3} p(\mathbf{h}_2, \mathbf{h}_3)$$

Continuing the same way for l hidden layers we can get a good initialization of the model parameters. After the greedy initialization, the DBN can be further tuned using the up-down algorithm to obtain the hidden features as $\mathbf{h}_l$.

5.1.4 t-SNE

t-distributed stochastic neighbor embedding (t-SNE) is a machine learning algorithm for dimensionality reduction developed by Geoffrey Hinton and Laurens van der Maaten. It is a nonlinear dimensionality reduction technique that is particularly well-suited for embedding high-dimensional data into a space of two or three dimensions, which can then be visualized in a scatter plot. Specifically, it models each high-dimensional object by a two- or three-

dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points.

The t-SNE algorithm comprises two main stages. First, t-SNE constructs a probability distribution over pairs of high-dimensional objects in such a way that similar objects have a high probability of being picked, whilst dissimilar points have an extremely small probability of being picked. Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the KullbackLeibler divergence between the two distributions with respect to the locations of the points in the map. Note that whilst the original algorithm uses the Euclidean distance between objects as the base of its similarity metric, this should be changed as appropriate.

where KL divergence is defined by

$$KL(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{5.12}$$

Chapter 6

Movie Recommendation System

We come up with the same problem of recommending movies to a user, but here we use Deep Learning to further improvise on our previous methods. In numerous recommendation technologies, the most popular and successful one is collaborative filtering (CF). However, along with the information rapidly increasing, CF is facing various challenges containing data sparsity, scalability and synonymy etc. To overcome the data sparsity problem, researchers have proposed many methods. Zhang et al. use feedforward neural network for recommendations and effectively reduce the data sparsity. Salakhutdinov et al. show us that Restricted Boltzmann Machines (RBMs) can be successfully applied to collaborative filtering. Zhao et al. show us that this idea can be extended to deep belief networks also which we try to demonstrate. We compare results of using Stacked Autoencoders, RBMs and Deep Belief Networks over the Largest public static dataset of MovieLens. The features that the user looks for before rating a movie are not directly visible if we look at his ratings for movies. The rating behaviour is hidden and needs

to be learnt. We build a hybrid recommendation model, in which the user-based CF algorithm makes predictions using the user features extracted by the DBNs. We evaluate the model on the large MovieLens dataset.

Deep learning has been proven to be very good at dimensionality reduction and feature extraction. We make a hybrid model, which improves the user-based CF algorithm by using the dimensionality reduction and feature extraction ability of DBNs. Firstly, traditional user-based CF is used to fill the missing data of rating matrix as described in Chapter 1. Secondly, we use the DBN as a feature extractor that transform the user rating data into a suitable representation of users. Finally, the user-based KNN algorithm is applied using the features extracted by DBNs according to Equation 6.1. We use l_1 norm to compute distances as because of high dimesionality, lower norms work better than higher ones (curse of dimensionality).

$$R(u,i) = \bar{R}(u) + \frac{\sum_{u_k \in N(u)} w(u,u_k)\ (R(u_k,i) - \overline{R}(u_k))}{\sum_{u_k \in N(u)} |w(u,u_k)|} \tag{6.1}$$

where $\bar{R}(u)$ and $\bar{R}(u_k)$ is the average rating for the user u and user u_k, $N(u)$ is the set of k nearest neighbors of u, and $w(u,u_k)$ denotes the similarity between the user u and user u_k.

The largest static MovieLens dataset contains 69,897 users and 10586 movies. Along with DBNs we also use RBMs and Stack Autoencoders to get 64 representative features after imputing the sparse matrix by CF. In the K-Nearest Neighbour, we compare the 3 variants and compute the test loss. Results obtained are tabulated.

We clearly see that DBN architecture outperforms the single RBM architecture and Stacked Autoencoder architecture to learn relevant features.

Feature learning model	Test RMSE
Deep Belief Networks	0.7437816
Restricted Boltzman Machine	0.767357
Stacked Autoencoders	0.752984

Table 6.1: Comparison of different Unsupervised Learning model

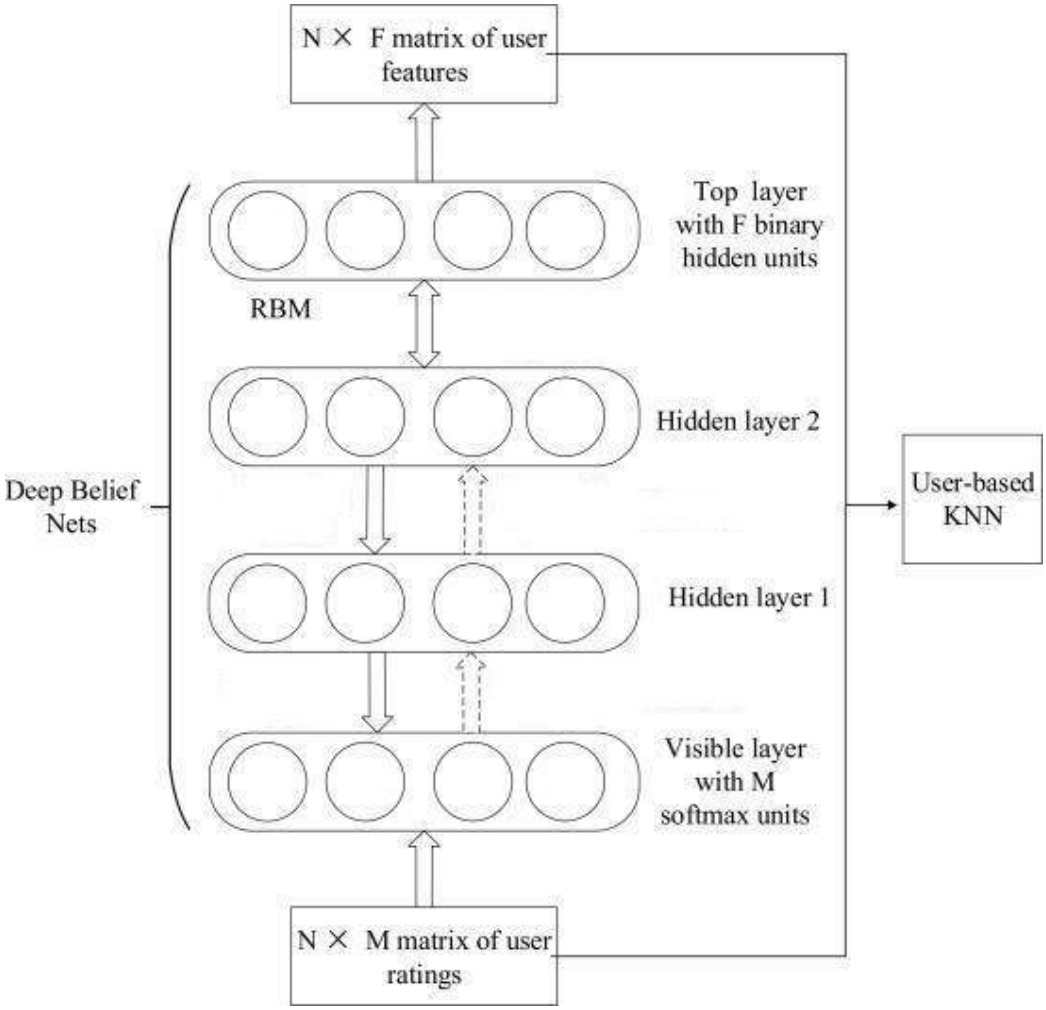

Figure 6.1: The CF-DBN-KNN architechture

The hidden features extracted can be visualized using T-SNE on a 2D plane.

We clearly see that we are able to look at the hidden structures and patterns in the rating matrix which were not directly visible in the rating

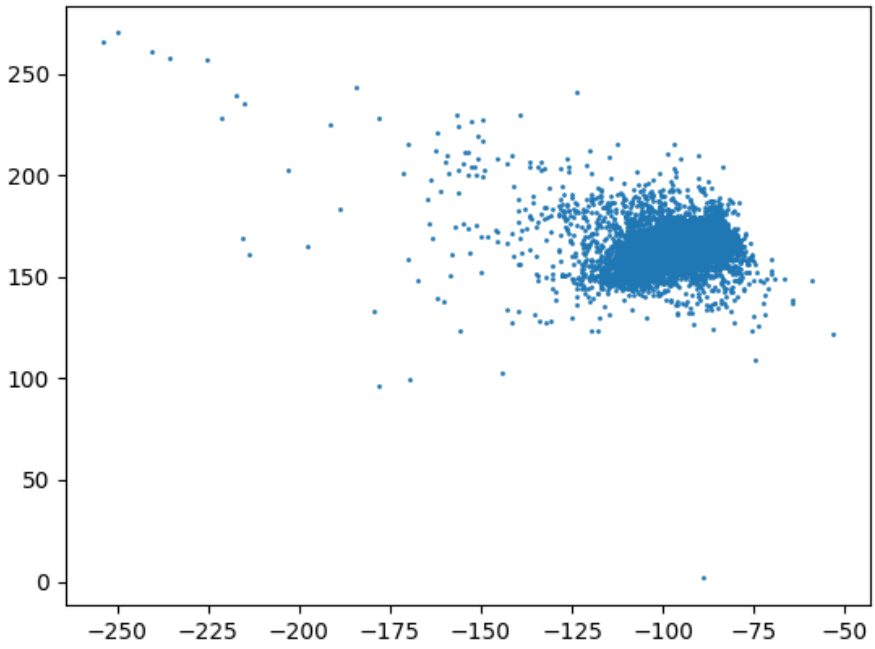

Figure 6.2: T-SNE projection of imputed rating matrix features

matrix but are visible when we learn the high level features from the same matrix using a RBM and are more prominently visible in the features learnt using DBN, justifying the results we obtain.

The architecture was trained on a 64GB RAM server machine with Nvidia Tesla K40c GPU in 10-12 hours.

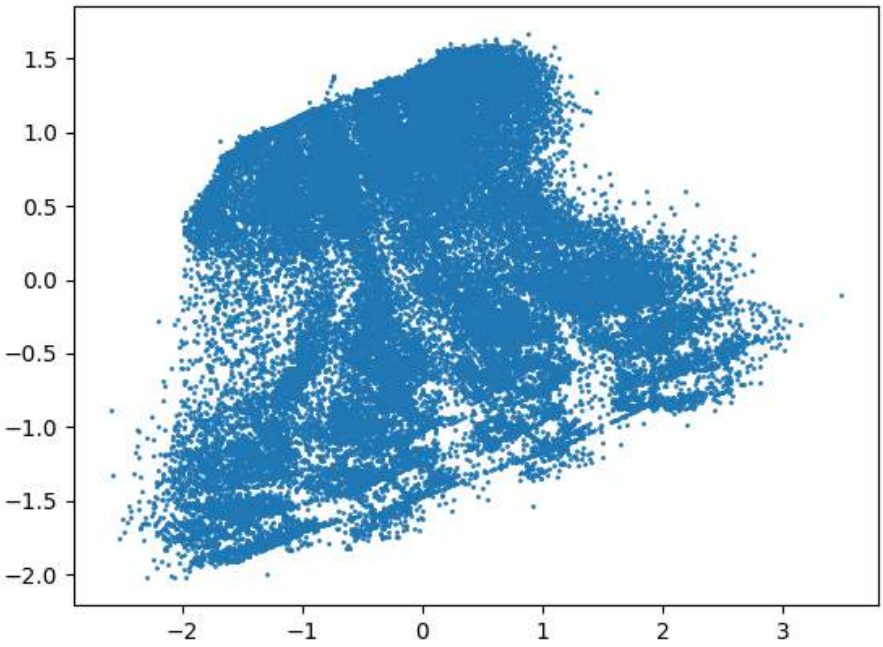

Figure 6.3: T-SNE projection of RBM features

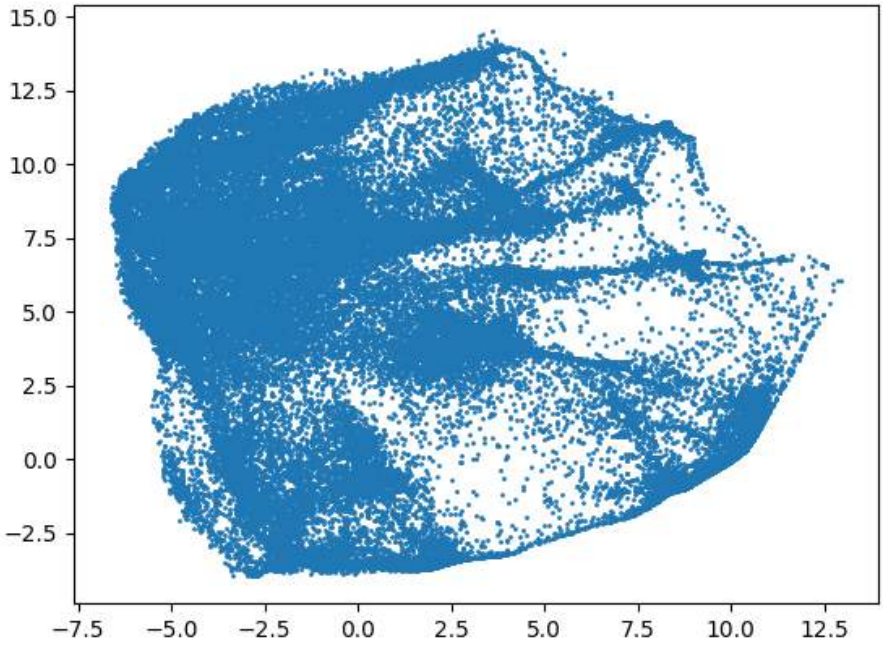

Figure 6.4: T-SNE projection of DBN features

Chapter 7

Document Recommendation System

The task of recommending documents to knowledge workers differs from the task of recommending products to consumers. Variations in search context can undermine the effectiveness of collaborative approaches, while many knowledge workers function in an environment in which the open sharing of information may be impossible or undesirable. For information seeking, what seems to be required is a document recommendation system that takes into account both the user's particular query and certain features from the overall search context.

Extracting the meaning out of discrete spaced text is indirect and requires feature learning that take care of the "meaning" of searched text and text bank. Unimportant words, synonyms, or partial sentence queries hinder the capability of a search engine to know the meaning. Therefore a word by word recomemndation would fail to provide relevant desired documents. Here comes the roll of feature learning. We use Word2vec model and Stacked autoencoders to extract high level features out of the text and compare these

features with the extracted features of the query to find nearest document using the vector space model which used cosine similarity as a similarity metric.

We use Reuters as a benchmark dataset from the python nltk module for our recommendation system. Stop words are removed and we lemmentize and stem the english words so that different english words get changed to their base form. For example, a sentence like "The cats were roaming aroud the street" gets converted to "cat roam street". This reduces noise and helps in treating words of several forms as a single unique word. Common english stop words are removed as they do not provide relevant or extra information about the text.

We first create a map from words to real number vectors that are continuous representations in a high dimensional space using Word2vec model and we represent a document by the mean of the word embeddings. These features are passed into a stacked autoencoder to further learn features in a smaller dimensional space. We use the vector space model to find similar documents to a given query.

$$\mathcal{D} \xrightarrow{\text{Word2Vec}} \mathbb{R}^{256} \xrightarrow{\text{Autoencoder}} \mathbb{R}^{16}$$

We use cosine similarity as the similarity to find nearest relevant document.

$$d(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^T \mathbf{y}}{||\mathbf{x}||_2 ||\mathbf{y}||_2} \tag{7.1}$$

The query string is also passed through the learnt transformations and is compared with all the transformed documents in $\mathbb{R}^{16}$. The results we obtained are not quantified but are analysed analytically where we found

that search results were not word based but meaning based.

Example :-

Query	Document
Economic news	...The dollar is stable again... The current level is the appropriate level...
United States	...Washington announced plans for as much as 300 million dollars...
India	...Tata Steel, a trend setter on the Bombay Stock Exchange, opened today higher at 1,040 rupees against yesterday's...

Table 7.1: Some qualitative results of Document Recommendation System

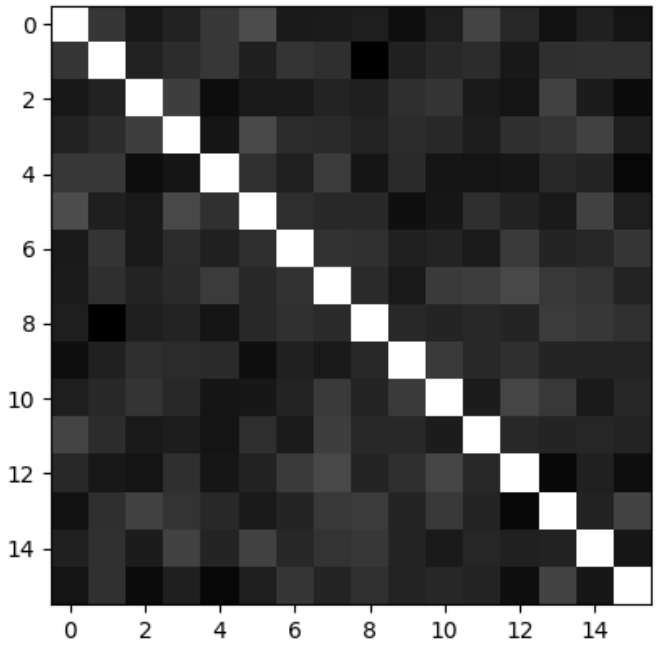

Figure 7.1: Emperical Linear Correlation matrix of learnt features

We use Dropout regularization and Batch Normalization on each of the 5 layer of the Stacked Autoencoder to obtain potentially uncorrelated features.

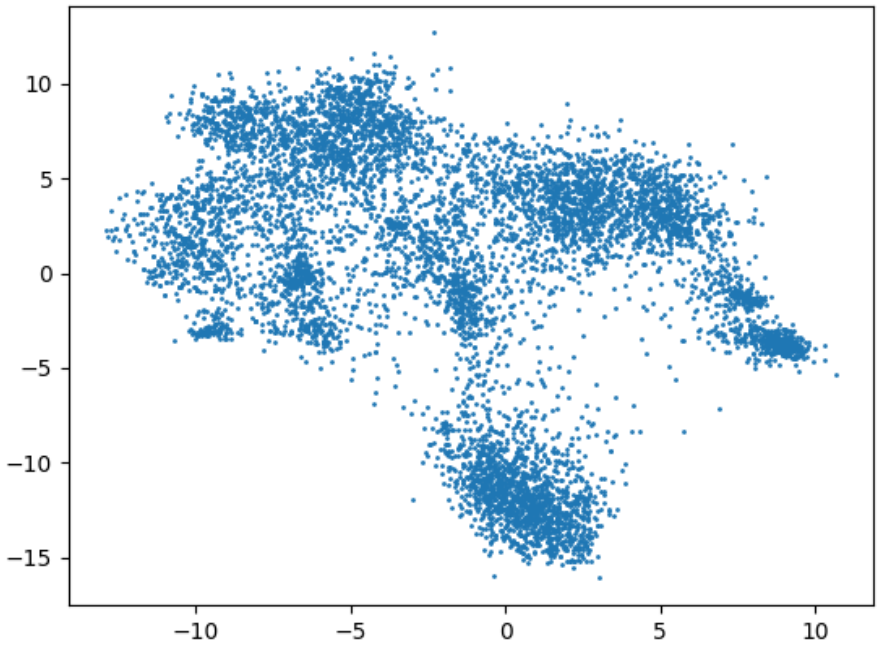

Figure 7.2: T-SNE projection of mean embeddings of documents

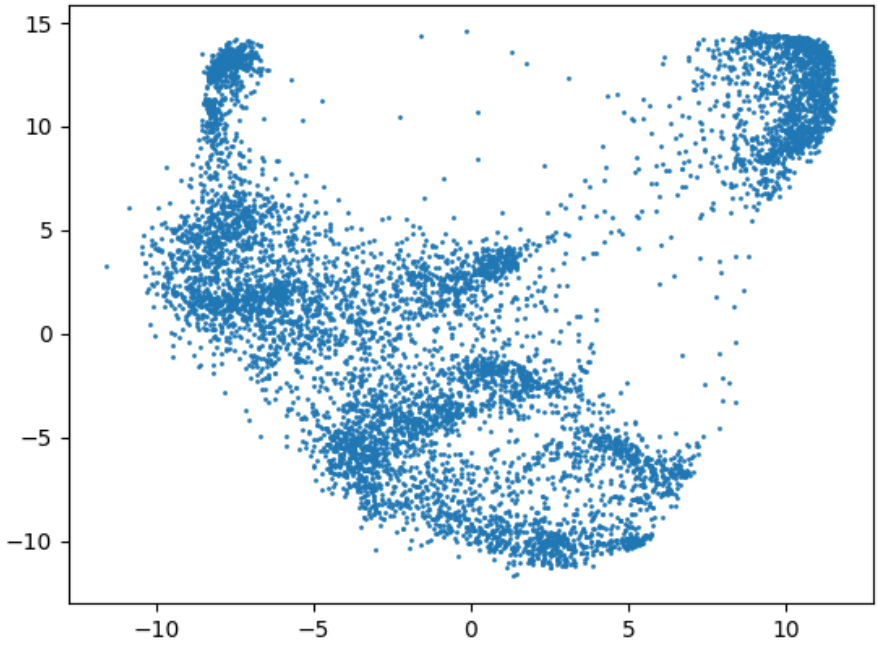

Figure 7.3: T-SNE projection of learn features of mean embeddings of documents

From the scatter plot, we clearly see cluster structures leading to good results as Reuters is a classification based dataset.

Along with the Word2vec model, we use discrete word based models.

1. Word count model where a document vector is denoted by a probability mass function vector $\in \mathbb{R}^{|\mathcal{W}|}$. Low dimensional representation of the vectors were fed into the vecor space model to retrieve the most similar document but there were hardly any relevant documents retreived.

2. Boolean term document matrix were also used as the initial data into an RBM to learn hidden features but no relevant documents were retreived.

3. We also used normalized term frequency inverse document frequency (tf-idf) matrix as initial data to a stacked autoencoder but no relevant documents were retrieved.

Reuter's dataset has 7703 train documents and 3019 test documents. We use a 64GB RAM server with Nvidia Tesla K40c GPU. Training a staked autoencoder takes a few hours whereas training a RBM takes 10-12 hours on the server machine.

Chapter 8

Image Recommendation System

Humans inevitably develop a sense of the relationships between objects, some of which are based on their appearance. Some pairs of objects might be seen as being alternatives to each other, while others may be seen as being complementary. Therefore it is interesting to uncover relationships between objects, and particularly in modelling the human notion of which objects complement each other and which might be seen as acceptable alternatives. We thus seek to model what is a fundamentally human notion of the visual relationship between a pair of objects, rather than merely modelling the visual similarity between them.

An approach to recommend similar images is to classify a labeled training set and find the most probable class (finite) to which an object might belong. However this approach required human labelled data which is difficult to prepare and not scalable. Unsupervised learning helps modelling the distribution of data and learn important features which can be used for comparision. But why learn features when we already have the image dataset?

The reason is that image data is highly locally correlated and is invariant to small affine transformations like translation, scaling and skewing, small color based changes like brightness, sharpness contrast and color. Whether a car is painted red or yellow, both are similar in the sense that it is a car. Also, an image is a 2D projection of the 3D world, it fails to capture the 3D orientation of objects. Whether we look at the side face of a human being or the front face, both represent a human being therefore learning relevant features is important that learns the human perception of recognizing similar objects. Unsupervised Learning does not require labelled data and helps in learing the important abstract features that are usually in a lower dimensional space. Therefore comparing these learnt lower dimensional features is a much better way to computationally compare images represening 2 objects.

We use a Convolutional Autoencoder to learn features out of images that are used in the vector space model to get the most similar images. The convolutional neural network contains 13 hidden layers, with 7 convolutional layers, 3 maxpool layers and 3 upsampling layers. Each convolutional layer contains a dropout layer also which helps in enforcing independence in the learn variables. The middle most 4D tensor is extracted after the training as a (n,4,4,64) 4D dimensional matrix. This is flattened to (n,1024) 2D matrix where n is the number of training images.

Cifar100 dataset from python keras module was used as a benchmark dataset which contains 50,000 training images and 10,000 test images from 100 different classes. We do not use the information of classes anywhere in the neural network while training.

The encoder of the autoencoder gives us the hidden features of an image. We then use the vector space model to find the 10 most similar images from training set for a given test set. If we pick up 10 radom images from 50,000

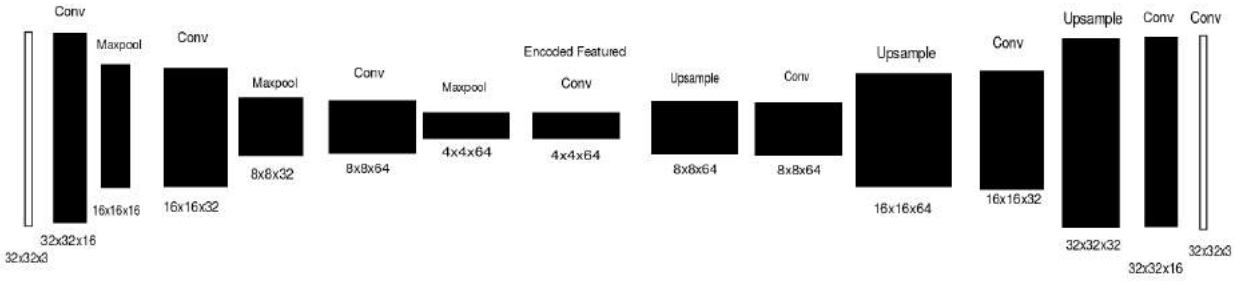

Figure 8.1: Convolutional Autoencoder used

training images then the expected number of images that would belong to the same class as that of the test image would be 0.02, but when we learn the features in an unsupervised maner and use vector space model to extract top 10 images, we obtain an average of 1.45 number of images in the same class as that of the test image. This is significant as in the learning algorithm nowhere we take the help of the label data. Therefore with large amount of data, we have better training and the algorithm is easily scalable.

Below we present some examples of images from Cifar100 as well as the reconstructed image.

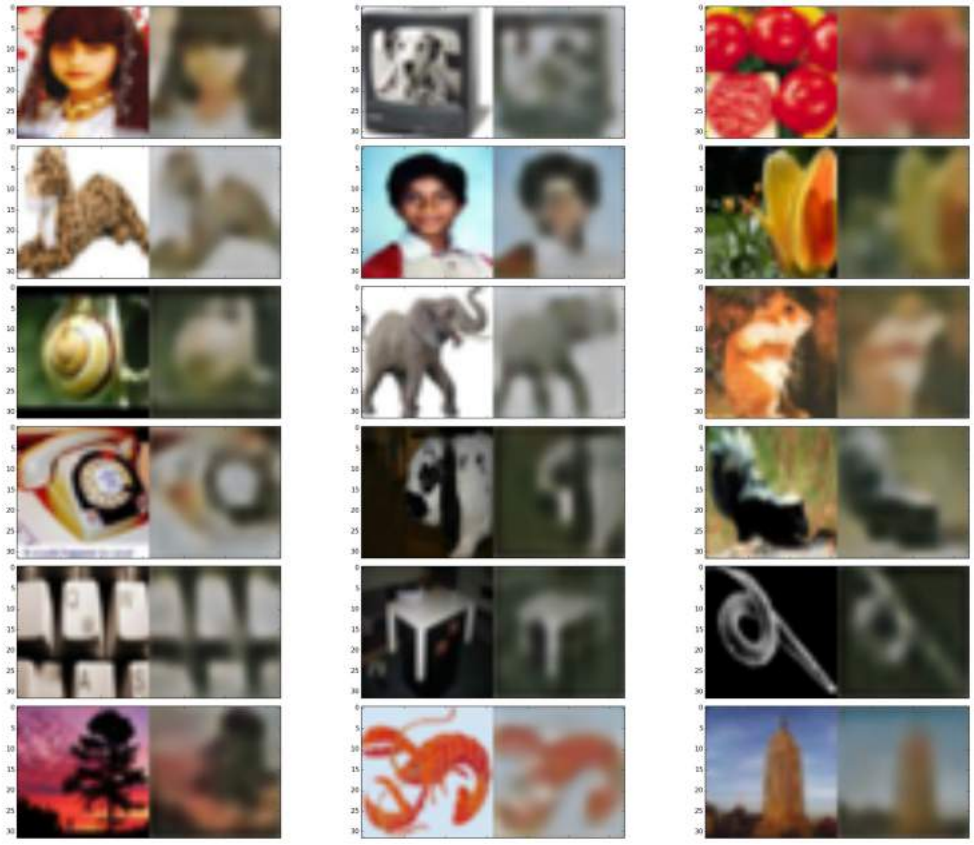

Figure 8.2: Original/Reconstructed images from the Convolutional Autoencoder

The architecture was trained on a 64GB RAM server machine with Nvidia Tesla K40c GPU in 10-12 hours.

Bibliography

[1] Fernando Ortegaa Antonio Hernando, Jess Bobadillaa. A non negative matrix factorization for collaborative filtering recommender systems based on a bayesian probabilistic model. 2016.

[2] Yves F. Atchad. *A computational framework for empirical Bayes inference*. 2010.

[3] Tao Jiang Junyao Zhao Jiehao Chen Chong Zhao, Jiyun Shi. Application of deep belief nets for collaborative filtering. 2016.

[4] K Seamons DL Lee, H Chuang. Document ranking and the vector-space model. 1997.

[5] GE Hinton. Deep belief networks. 2009.

[6] D Cirean J Schmidhuber J Masci, U Meier. Stacked convolutional auto-encoders for hierarchical feature extraction. 2011.

[7] Mingsong Mao Wei Wang Guangquan Zhang Jie Lu, Dianshuang Wu. Recommender system application developments: A survey. 2015.

[8] A Krizhevsky N Srivastava, GE Hinton. Dropout: a simple way to prevent neural networks from overfitting. 2014.

[9] IL) Koenigstein Noam (Ra'anana IL) Keren Shahar (Hemed IL) Kroskin Ayelet (Tel Aviv IL) Paquet Ulrich (Cambridge GB) Nice, Nir (Kfar Veradim. Modified matrix factorization of content-based model for recommendation system. 2016.

[10] Y Bengio P Vincent, H Larochelle. Extracting and composing robust features with denoising autoencoders. 2008.

[11] G Hinton R Salakhutdinov, A Mnih. Restricted boltzmann machines for collaborative filtering. 2007.

[12] Juan ramos. Using tf-idf to determine word relevance in document queries.

[13] X Rong. word2vec parameter learning explained. 2014.

[14] C Szegedy S Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[15] Jeawon Park Sang-Hyun and Jaehyun Choi. Personal preference based movie recommendation systems. 2015.

[16] Joydeep Ghosh Sreangsu Acharyya, Oluwasanmi Koyejo. Learning to rank with bregman divergences and monotone retargeting. 2012.

[17] Kazuyuki Motohashi Suchit Pongnumkul. Random walk-based recommendation with restart using social information and bayesian transition matrices. 2015.

[18] SKM Wong VV Raghavan. A critical analysis of vector space model for information retrieval. 1986.

[19] Min-Yen Kan Tat-Seng Chua Xiangnan He, Hanwang Zhang. Fast matrix factorization for online recommendation with implicit feedback. 2016.

[20] Massimo Quadrana Paolo Cremonesi Yashar Deldjoo, Mehdi Elahi. Toward building a content-based video recommendation system based on low-level features. 2015.

[21] Robert Bell Yehuda Koren and Chris Volinsky. Matrix factorization technique for recommendation systems. IEEE Computer Society, 2009.

[22] Hongya Wang Ching-Hsien Hsu Yukun Li Yingyuan Xiao, Pengqiang Ai. Enrs: An effective recommender system using bayesian model. 2015.

[23] Jianxun Lian Xing Xie Zhongqi Lu, Zhicheng Dou and Qiang Yang. Content-based collaborative filtering for news topic recommendation. 2015.