Deep Latent Factor Models for Collaborative Filtering

Semih Akbayrak

May 2017

1 Introduction

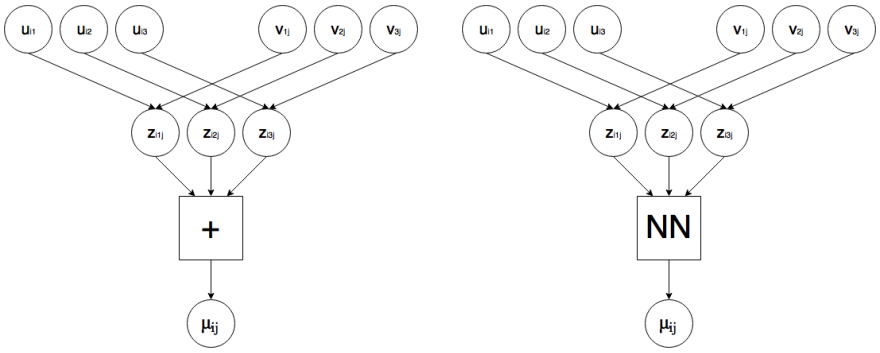
Deep Learning models produced state of the art results in Computer Vision, Speech Recognition, and NLP tasks and one common attribute of these tasks is that they require advanced perception capabilities. So it has been proven that deep learning models work well as perception machines, but their capabilities of probabilistic reasoning and relational learning have not been investigated detailly, yet.

In this project, I tried to develop a general model for collaborative filtering tasks. What makes this task different than the other deep learning tasks is that there is no actual input in collaborative filtering, because we don't know attributes of items but we know their relationship with each other. I utilized Neural Networks to estimate the sufficient statistics of likelihood probability distributions. In order to understand its efficiency, I just ran an empirical study on rating data with Gaussian likelihood, but it can be generalized to other distributions and datasets easily. I worked on two models. First model is developed in more probabilistic setting, whereas second model relies on intuition mostly.

2 First Model

In general manner, there are 2 common ways to solve collaborative filtering problems: neighbourhood models and latent factor models. My interest is in there latent factor models. In these models, we assume items has some latent attributes and if latent attributes of two items are close to each other, then probability of being in interaction with each other is high.

Matrix factorization models may be the most popular models for collaborative filtering[1] [2]. In general, matrix factorization models assume that ratings or



(a) Classical Matrix Factorization (b) Deep Latent Factorization

Figure 1: Latent Factor Models

counts are produced by inner product of item and user latent representations. It is illustrated in figure1(a) and as it can be seen, inner product can be thought as addition function which accepts element-wise multiplication of item and user latent representation vectors as input.

Accepting addition as a function may limit our model. There may be a richer function in there and instead of proposing different functions and evaluating them, we can let model to find this function by itself. Therefore, I changed the addition function with Neural Network. So during learning, parameters of Neural Network namely weights should be estimated. We can accept latent representations as parameters and estimate them during learning stage, as well, but we rather to choose accepting them as random variables so an additional inference stage needed to estimate probabilities. Interpreting latent representations as random variables brings some advantages. First of all, it allows us to take uncertainty into account. Because latent representations are random variables not point estimates, during rating of unknown places in matrix, we can sample from probability distributions and we can evaluate rating samples instead of just rating. Secondly, this model is robust to overfitting thanks to prior. But the huge disadvantage of this kind of models is difficulty in inference. In my model, for inference I utilized recent advances in variational inference methods. But before diving into details of inference, generative model is described below:

$$u \sim \mathcal{N}(u; 0, I) \tag{1}$$

$$v \sim \mathcal{N}(v; 0, I) \tag{2}$$

$$r \sim \mathcal{N}(r; f(u^T v), I) \tag{3}$$

How can we make inference in such a complicated model? Our aim is to find $p(u, v|r)$ but because we can't compute it exactly, we may try to approximate

it with another distribution $q(u, v|r)$. We need an objective and this objective should be a divergence metric, so we use reverse KL divergence.

$$\min_{\phi} KL(q_\phi(u, v|r)||p(u, v|r)) \tag{4}$$

Here ϕ is variational parameters

$$\min_{\phi} KL(q_\phi(u, v|r)||p(u, v|r)) = \int q_\phi(u, v|r) \log \frac{q_\phi(u, v|r)}{p(u, v|r)} dudv \tag{5}$$

$$= \int q_\phi(u, v|r) \log \frac{q_\phi(u, v|r)}{p(u, v, r)} dudv + \int q_\phi(u, v|r) \log p(r) dudv \tag{6}$$

$$= \int q_\phi(u, v|r) \log \frac{q_\phi(u, v|r)}{p(u, v, r)} dudv + \log p(r) \tag{7}$$

So,

$$\log p(r) = L(\phi) + KL(q_\phi(u, v|r)||p(u, v|r)) \tag{8}$$

where

$$L(\phi) = -\int q_\phi(u, v|r) \log \frac{q_\phi(u, v|r)}{p(u, v, r)} dudv \tag{9}$$

$\log p(r)$ is constant w.r.t. ϕ so $\min_{\phi} KL(q_\phi(u, v|r)||p(u, v|r)) = \max_{\phi} L(\phi)$ Now we can focus on our new objective.

$$L(\phi) = -\int q_\phi(u, v|r) \log \frac{q_\phi(u, v|r)}{p(u, v, r)} dudv \tag{10}$$

$$= E_q[-\log q_\phi(u, v|r) + \log p(r|u, v) + \log p(u, v)] \tag{11}$$

$$= E_q[\log p(r|u, v) + \log \frac{p(u, v)}{q_\phi(u, v|r)}] \tag{12}$$

$$= E_q[\log p(r|u, v)] + \int q_\phi(u, v|r) \log \frac{p(u, v)}{q_\phi(u, v|r)} dudv \tag{13}$$

We go further and factorize $q_\phi(u, v|r)$ as $q_\phi(u|r)q_\phi(v|r)$ like mean field approximation for easier computation. Then,

$$= E_q[\log p(r|u, v)] + \int q_\phi(u|r)q_\phi(v|r) \log \frac{p(u)p(v)}{q_\phi(u|r)q_\phi(v|r)} dudv \tag{14}$$

$$= E_q[\log p(r|u, v)] + \int q_\phi(u|r)q_\phi(v|r) \log \frac{p(u)}{q_\phi(u|r)} dudv + \int q_\phi(u|r)q_\phi(v|r) \log \frac{p(v)}{q_\phi(v|r)} dudv \tag{15}$$

$$= E_q[\log p(r|u,v)] + \int q_\phi(u|r) \log \frac{p(u)}{q_\phi(u|r)} du \int q_\phi(v|r) dv + \int q_\phi(v|r) \log \frac{p(v)}{q_\phi(v|r)} dv \int q_\phi(u|r) du \tag{16}$$

$$= E_q[\log p(r|u,v)] + \int q_\phi(u|r) \log \frac{p(u)}{q_\phi(u|r)} du + \int q_\phi(v|r) \log \frac{p(v)}{q_\phi(v|r)} dv \tag{17}$$

$$= E_q[\log p(r|u,v)] - KL(q_{\phi_u}(u|r)||p(u)) - KL(q_{\phi_v}(v|r)||p(v)) \tag{18}$$

Until now, we only talked about optimization w.r.t ϕ, but we also have model parameters θ which is nothing but weights of Neural Network, so at the same time we should make optimization w.r.t. θ in order to increase log likelihood $\log p_\theta(r)$. Overall our objective is

$$maxL(\theta, \phi) = E_q[\log p_\theta(r|u,v)] - KL(q_{\phi_u}(u|r)||p(u)) - KL(q_{\phi_v}(v|r)||p(v)) \tag{19}$$

or equivalently

$$minLoss = KL(q_{\phi_u}(u|r)||p(u)) + KL(q_{\phi_v}(v|r)||p(v)) - E_q[\log p_\theta(r|u,v)] \tag{20}$$

Before making comments about this lost function, let's remember our purpose. We want to approximate $p(u, v|r)$ with another distribution q. A probability distribution is nothing but a function and we know that Neural Networks are universal function approximators. So the question is can we use Neural Networks to approximate $p(u, v|r)$?

If we look at this lost function more closely, we can easily see that the KL terms work like encoders with regularizer that try to keep hidden variables as close as possible to priors, and the last term works like decoder which maps hidden variables to observed variables. So this is actually an autoencoder with additional regularizer term and it is called Variational Autoencoder[3] and similar equation derivations to mine can be found in [4]. According to my equation, encoder take r_{ij} as input, and I keep it like that to be in the same language with the original article [3]. However, r_{ij} as input can not give enough information to make inference so instead of it, I used entire i^{th} column and j^{th} row as input. Moreover, to prevent my model to copy from input, I deleted corresponding r_{ij} value from rows and columns. Therefore, I encouraged my model to extract information about a rating by looking at its neighbours namely ratings of user j and ratings of movie i.

In the end, our model turns out to be:

$$p(u) = N(u; 0, I) \tag{21}$$

$$p(v) = N(v; 0, I) \tag{22}$$

$$p_\theta(r|u,v) = N(r; f_\theta(u^T v), I) \tag{23}$$

$$q_{\phi_u}(u|r) = N(u; g_{mean_{\phi_u}}(r[i,:]), g_{var_{\phi_u}}(r[i,:])) \tag{24}$$

$$q_{\phi_v}(v|r) = N(v; h_{mean_{\phi_v}}(r[:,j]), h_{var_{\phi_v}}(r[:,j])) \tag{25}$$

where $f, g_{mean}, g_{var}, h_{mean}, h_{var}$ are Neural Networks, θ weights of f that is model parameters, ϕ_u, ϕ_v weights of g, h respectively that is variational parameters.

When we look at loss function again, KL terms are analytically computable, which is nothing but KL divergence of two Gaussian distributions.

$$KL(q_\phi(u|r)||p(u)) = \frac{1}{2}(tr(g_{var_{\phi_u}}) + g^T_{mean_{\phi_u}} g_{mean_{\phi_u}} - dim(u) - \log\det(g_{var_{\phi_u}})) \tag{26}$$

This is differentiable w.r.t. ϕ and does not contain θ

Last term can be computed with Monte Carlo integration

$$E_{q_\phi}[\log p_\theta(r|u,v)] \approx \frac{1}{S}\sum_{s=1}^{S} \log p_\theta(r|u^{(s)}, v^{(s)}) \tag{27}$$

for $u^{(s)} \sim q_{\phi_u}(u|r)$ and $v^{(s)} \sim q_{\phi_v}(v|r)$.

However derivative of this integration w.r.t. ϕ is 0 because it doesn't contain ϕ parameters, although u and v depends on ϕ. In order to overcome this problem, it is suggested to use reparametrization trick which is to use

$$\frac{1}{S}\sum_{s=1}^{S} \log p_\theta(x|u^{(s)}, v^{(s)}) \tag{28}$$

for

$$u^{(s)} = g_{mean_{\phi_u}} + g^{\frac{1}{2}}_{var_{\phi_u}} . \epsilon_u^{(s)} \tag{29}$$

where

$$\epsilon_u^{(s)} \sim N(0, I) \tag{30}$$

and

$$v^{(s)} = h_{mean_{\phi_v}} + h^{\frac{1}{2}}_{var_{\phi_v}} . \epsilon_v^{(s)} \tag{31}$$

where

$$\epsilon_v^{(s)} \sim N(0, I) \tag{32}$$

This is now differentiable w.r.t. ϕ. To compute gradient, we can compute it for every $u^{(s)}, v^{(s)}$ and use this values in summation.

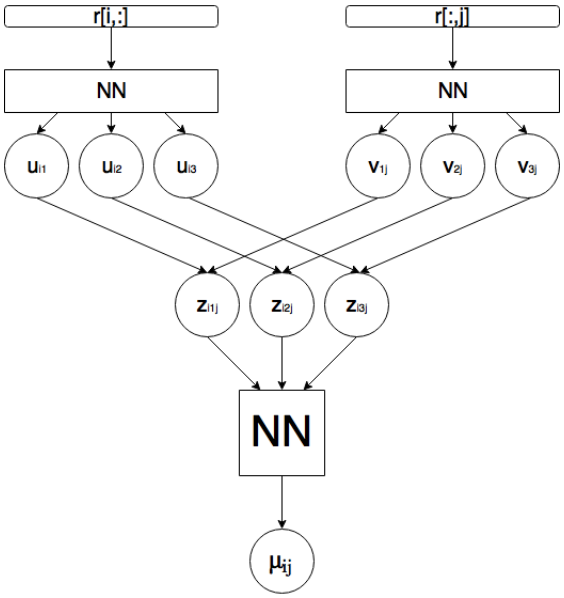


Figure 2: Graphical representation of second model

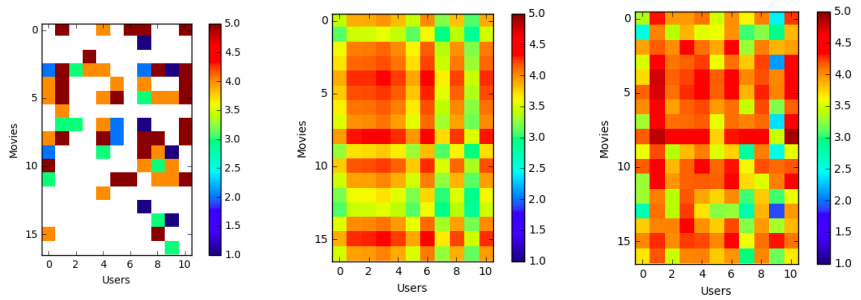
3 Second Model

In second model, I directly applied a deterministic model which has the same shape with the first model. Basically it takes i^{th} row and j^{th} columns of R matrix as input and corresponding r_{ij} values excluded from them to prevent copy. Then Neural Networks find latent representations of movies and users and their complex interaction creates the rating value r_{ij}. This model is illustrated in figure2.

4 Experiments

For experiments, Movie Lens 100k dataset was used. It contains ratings of 943 users for 1682 movies. Throughout the report, I used Gaussian link function by considering rating dataset task, so during experiments mean squared error was used which is the expectation of negative loglikelihood of Gauissian distribution. For different tasks, different link functions and their corresponding negative loglikelihoods can be used like Poisson for count data, or Bernoulli for like-dislike data, etc. However, results was not as good as I expected for even Gaussian case. Therefore I thought it is not worth it to try other tasks for different datasets.

In figure3(a), there is a slice from dataset and figere3(b) and figure3(c) are the predictions of my models for this slice. Just by looking at visualizations,



(a) Slice from dataset (b) Predictions of model1 (c) Predictions of model2

Figure 3: Visualization of predictions

Model	RMSE	MAE
KNN-Basic	.9789	.7732
KNN-With Means	.9514	.7500
KNN Baseline	.9306	.7334
SVD	.9364	.7381
SVD++	.9200	.7253
NMF	.9634	.7532
Slope One	.9454	.7430
Co clustering	.9678	.7579
PMF	.9190	-
Biased PMF	.9087	-
BMFSI	.9065	-
SCMF	.9068	-
mDA-CF	.9040	-
mSDA-CF	.9024	-
my model1	.9689	.7687
my model2	.9271	.7330

Figure 4: Results

the effect of regularizer can be seen. While second model is finding well spread results, first model is hesitating to appoint too small or too large numbers as rating predictions.

For evaluation, we used 5-fold cross validation as it is suggested in dataset instructions. Each fold consists of 80%/20% training/test data of all data. For comparison, we used several benchmarks [5],[6]. As evaluation metric, I used RMSE and MAE. As it can be seen, my first model performs poorly when compared with other models. Second model performs better but it is not as good as state of the art models.

References

[1] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Nips*, volume 1, pages 2–1, 2007.

[2] Prem Gopalan, Jake M Hofman, and David M Blei. Scalable recommendation with poisson factorization. *arXiv preprint arXiv:1311.1704*, 2013.

[3] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[4] Mart van Baalen. Deep matrix factorization for recommendation. 2016.

[5] Nicolas Hug. A python scikit for recommender systems. http://surpriselib.com.

[6] Sheng Li, Jaya Kawale, and Yun Fu. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 811–820. ACM, 2015.