Collaborative Filtering with Stacked Denoising AutoEncoders and Sparse Inputs

Florian Strub, Jérémie Mary, Preux Philippe

Collaborative Filtering with Stacked Denoising AutoEncoders and Sparse Inputs

Florian Strub
Univ-Lille, CRISTaL
Inria - SequeL
florian.strub@inria.fr

Jeremie Mary
Univ-Lille, CRISTaL
Inria, SequeL
jeremie.mary@inria.fr

Philippe Preux
Univ-Lille, CRISTaL
Inria, SequeL
philippe.preux@inria.fr

Abstract

Neural networks have received little attention in Collaborative Filtering. For instance, no paper using neural networks was published during the Netflix Prize apart from Salakhutdinov et al's work on Restricted Boltzmann Machine (RBM) [14]. While deep learning has tremendous success in image and speech recognition, sparse inputs received less attention and remains a challenging problem for neural networks. Nonetheless, sparse inputs are critical for collaborative filtering. In this paper, we introduce a neural network architecture which computes a non-linear matrix factorization from sparse rating inputs. We show experimentally on the movieLens and jester dataset that our method performs as well as the best collaborative filtering algorithms. We provide an implementation of the algorithm as a reusable plugin for Torch [4], a popular neural network framework.

1 Introduction

Recommendation systems try to give advice to users on which items (movies, music, products, etc.) users are more likely to be interested in. A good recommendation system may dramatically increase the amount of sales of a company and retain customers. One of the most important topics in recommendation system is collaborative filtering. It aims at predicting the preference of one user by combining his feedback on a few items and the feedback of all other users. For instance, if someone rated only a few books, can we estimate the ratings he would have given to thousands of other books by using the ratings of all the other readers? Can we turn a sparse matrix of past ratings into a dense matrix of estimate ratings?

The most successful approach to collaborative filtering is to retrieve potential latent factors from the sparse matrix of ratings. Book latent factors are likely to encapsulate the book genre (spy novel, fantasy, etc.) or some writing styles. Common latent factor techniques are to compute a low-rank rating matrix by either applying Singular Value Decomposition SVD [6] or Regularized Alternative Least Square algorithm [20]. However, it was argued that these methods are linear and cannot catch subtle factors. Newer algorithms were explored to face those constraints such as Non Linear Probabilistic Matrix Factorization (NL-PMF) [8], Local Low Rank Matrix Approximation (LLORMA) [10] or Factorization Machines (FM) [13] which are more flexible extensions of this approach.

Neural networks have already been studied in recommendation systems such as RankNet [2] to *rank* users' preferences. Yet, few works have tried to apply neural networks to collaborative filtering. It is all the more surprising considering that deep neural networks are able to discover non-linear latent variables [9, 3]. In a preliminary work, Salakhutdinov [14] tackled the Netflix challenge using Restricted Boltzmann Machines. This algorithm is well

known in the deep learning community as an important tool to initialize neural networks. Yet, the initial paper claimed that they did not manage to turn it into a neural network.

One natural approach to deal with collaborative filtering with neural networks is to use autoencoders. Those networks are trained to reconstruct their inputs through a dimension reduction. Thus, they perform a Non Linear Principal Component Analysis (NPCA) [7]. In our case, the autoencoder is fed with *sparse* rating inputs and it aims at reconstructing dense rating vectors. Very few papers exist that tackled this sparsity constraint[1, 12]. In most previous works, sparsity was avoided in neural networks. Even so, autoencoders are becoming an important topic in collaborative filtering. Hao Wang [19] used autoencoders to learn a representation of bag of words of plot to estimate the ratings of movies. AutoRec [15], which was developed independently from our work, shares similarities with our approach. Yet, they use different error constraints to train their autoencoders.

In this paper, we develop a training strategy to perform collaborative filtering using Stacked Denoising AutoEncoders neural networks (SDAE) with sparse inputs. We show that neural networks provide excellent experimental results. Benchmarks are done on RMSE metric which is commonly used to evaluate collaborative filtering algorithms. We developed several new Torch modules as the framework lacks some important tools to deal with sparse inputs. Reusable source code is provided to reproduce the results and to be used to build new networks. First, we introduce denoising autoencoders, we then explain the constraints we made to enforce sparsity in section 2. After explaining the benchmark models, we describe our experimental settings and results, respectively in section 3 and 4. Finally, we provide an underlying motivation which entails future potential works in section 5.

2 Denoising autoencoders for sparse inputs

2.1 Denoising Autoencoders

Autoencoders are feed-forward neural networks popularized by Kramer [7]. They are unsupervised networks where the output of the network only needs to reconstruct the initial input. The network is constrained to use narrow hidden layers, forcing an implicit dimensionality reduction on the data. The network is trained using squared error loss on the reconstruction error and back-propagation.

Recent works in deep learning advocate to stack autoencoders to pre-train Deep Neural Networks [5]. This process enables the lowest layers of the network to find low-dimensional representations. It experimentally increases the quality of the whole network. Yet, classic autoencoders often degenerate into identity networks and they fail to learn the relationship between data. Pascal Vincent et al. [17, 18] tackle the issue by corrupting inputs, pushing the network to denoise the final outputs. Three processes are described to corrupt data:

- Gaussian Noise : Gaussian Noise is added to a subset of the input
- Masking Noise : A fraction ν of the input is randomly forced to be zero.
- Salt-and-Pepper Noise : A fraction ν of the input is randomly forced to be one of the input maximum/minimum.

Therefore, the Denoising AutoEncoder (DAE) loss function is modified to emphasize the denoising aspect of the network. It is based on two main hyperparameters α, β. They balance whether the network would focus on denoising the input (α) or reconstructing the input (β).

$$L_{2,\alpha,\beta}(\mathbf{x}, \tilde{\mathbf{x}}) = \alpha \left(\sum_{j \in \mathcal{J}(\tilde{x})} [nn(\tilde{\mathbf{x}})_j - x_j]^2 \right) + \beta \left(\sum_{j \notin \mathcal{J}(\tilde{x})} [nn(\tilde{\mathbf{x}})_j - x_j]^2 \right)$$

Where $nn(\mathbf{x})_k$ is the k^{th} output of the network, $\tilde{\mathbf{x}}$ is the corrupted input $\mathbf{x}$, $\mathcal{J}$ are the indexes of the corrupted element of $\mathbf{x}$.

2.2 Sparse Inputs

There is no standard approach for using sparse vector as input of neural networks. Most of the papers dealing with sparse inputs got around by pre-computing an estimate of the missing values [16, 1]. In our case, we want the autoencoder to handle itself this prediction issue. Such problems have already been studied in industry [12]. However, the amount of missing values was very low (less than 5%) and all the missing values were known during the training. In collaborative filtering, input vectors are very sparse and target vectors have a great number of missing values.

The following subsections provide a training framework to tackle the training of sparse autoencoders by

- Inhibiting the edges of the input layers by zeroing out values in the input
- Inhibiting the edges of the output layers by zeroing out back-propagated values
- Using a denoising loss to emphasize rating prediction over rating reconstruction

One way to inhibit the input edges is to turn missing values to zero. To keep the autoencoder from always returning zero, we use an empirical loss that disregards the loss of unknown values. No error is backpropagated for missing values. Therefore, the error back-propagation will be propagated for actual zero values while it is discarded for missing values. In other words, missing values do not bring information to the network.

Using masking noise has two great advantages in our current issue. First, it works as a strong regularizer. Second, it trains the autoencoder to predict missing values. Therefore, the loss of the denoising autoencoder (DAE) becomes a promising objective function. To handle sparse inputs, the error of unknown values is discarded.

After regularization, the final training loss is:

$$L_{2,\alpha,\beta}(\mathbf{x},\tilde{\mathbf{x}}) = \alpha\left(\sum_{j\in\mathcal{J}(\tilde{\mathbf{x}})\cap\mathcal{K}(\mathbf{x})}[nn(\tilde{\mathbf{x}})_j - x_j]^2\right) + \beta\left(\sum_{j\notin\mathcal{J}(\tilde{\mathbf{x}})\cap\mathcal{K}(\mathbf{x})}[nn(\tilde{\mathbf{x}})_j - x_j]^2\right) + \lambda\left(\sum_{j\in\mathcal{K}(\mathbf{x})}|\mathbf{w_j}|^2_{Fro}\right)$$

where $\mathcal{K}(\mathbf{x})$ are the indexes of known values of $\mathbf{x}$ and λ is the regulizer hyperparameter.

2.3 The Model

User preferences are encoded by a sparse matrix of ratings $\mathbf{R}$. Given N users and M items, the rating r_{ij} is the rating given by the i^{th} user for the j^{th} item. A user is then described by a sparse vector u_i and an item is represented by a sparse vector v_j. Therefore, the goal is to predict an estimate $\hat{r}_{ij}$ for every missing rating. In other words, the goal is to complete the users'(or items) sparse vectors. Thus, we define two autoencoders to compute $\hat{\mathbf{R}}$:

- The *Uencoder* that takes the sparse vector u_i as input and compute a dense vector $\hat{u_i}$ as output. This network learns a user representation.
- The *Vencoder* that takes the sparse vector v_j as input and compute a dense vector $\hat{v_j}$ as output. This networks learn an item representation.

Two Mean Square Errors (MSE) co-exist for autoencoders and one must be careful to use the right estimator for benchmarking.

- Prediction Loss. This loss is actually unusual with autoencoders as it uses both the training and testing dataset. Nonetheless, this is an important metric in collaborative filtering. We use it to evaluate both the baselines and our method.

$$MSE_{pred}(\mathbf{X_{test}},\mathbf{X_{train}}) = \frac{1}{\|R_{test}\|}\sum_n\left(\sum_{k\in\mathcal{K}(\mathbf{x_{test}})}[nn(\mathbf{x_{train}})_k - x_{test,k}]^2\right)$$

- Reconstruction Loss. This is the basic autoencoder loss. This error has little interest for Collaborative filtering but it provides useful information during the training

$$MSE_{rec}(\mathbf{X_{test}}) = \frac{1}{\|R_{train}\|} \sum_{n} \left(\sum_{k \in \mathcal{K}(\mathbf{x_{test}})} [nn(\mathbf{x_{test}})_k - x_{test,k}]^2 \right)$$

Where $\|R_{test}\|$, $\|R_{train}\|$ are the number of rating in the testing dataset and training dataset.

2.4 Benchmarked Models

Latent factor matrix factorization seeks a low-rank matrix $\mathbf{R} = \mathbf{U}^T\mathbf{V}$ of rank k where $\mathbf{U} \in \mathbf{R}^{N \times k}$ is the users' representation and $\mathbf{V} \in \mathbf{R}^{M \times k}$ is the items' representation. We implemented Singular Value Decomposition (SVD) and Alternating-Least-Squares with Weighted-λ-Regularization (ALS-WR) [20] models for benchmarking. Indeed, a recent comparative study of collaborative filtering [11] pointed out that SVD technique remains the most efficient in the general case. Yet, we also reported NL-PMF [8] as it is a non-linear algorithm and AutoRec [15] as it is based on neural networks. They both use an equivalent setting in their publication.

SVD SVD looks for $\mathbf{U}$ and $\mathbf{V}$ by optimizing the following function through gradient descent:

$$L_2 = \sum_{(i,j) \in \mathcal{I} \times \mathcal{J}} (r_{ij} - \mathbf{u}_i^T \mathbf{v_j})^2 + \lambda(\|u_i\|_{Fro}^2 + \|v_j\|_{Fro}^2)$$

Where $\mathcal{I} \times \mathcal{J}$ are the set of indexes of known ratings and $\|u_i\|_{Fro}$ is the Froebenius norm. Additional parameters such as the user/item bias or handcrafted features may be added in the objective function to improve the final results. [6].

ALS-WR ALS-WR solves the low-rank matrix factorization problem by alternatively fixing $\mathbf{U}$ and $\mathbf{V}$ and solving the resulting linear problem. Tikhonov regularization is often used as it empirically provides good results [20].

$$L_2 = \sum_{(i,j) \in \mathcal{I} \times \mathcal{J}} (r_{ij} - \mathbf{u}_i^T \mathbf{v_j})^2 + \lambda(n_i\|u_i\|_{Fro}^2 + n_j\|v_j\|_{Fro}^2)$$

Where n_i is the number of rating for the i^{th} user and n_j is the number of rating for the j^{th} item.

2.5 Remarks

Neural networks actually compute a low rank approximation of $\mathbf{R}$. If we ignore the output transfer function, $\hat{\mathbf{R}}$ is iteratively built by the scalar product of the weight matrix of the last layer and the activation of the second to last layer. For instance, given a Uencoder, the $\mathbf{V}$ item representation is the uppermost weight matrix and $\mathbf{U}$ is the activation layers. The situation is reversed for Vencoders. One key point that is not clear is the link between these two representations. The user representation and the item representation do not lie in the same space. One representation belongs to the weight matrix whereas the other representation belongs to a dense reconstruction of the sparse inputs through previous layers. The task becomes more complicated if we compare the $\mathbf{U}$ representation from the activation of the Uencoders and the $\mathbf{U}$ representation from the Vencoder weights. Adding back the non-linearity makes the problem even harder. Yet, the final output of the autoencoder can still be considered as a latent non linear factor matrix factorization.

3 Experimental Setting

3.1 Training parameters

We train four-layer autoencoders. The first encoding layer is $1/10^{th}$ of the input size, the second layer is $1/12^{th}$ of the input size. The decoding layers have the same dimension in the

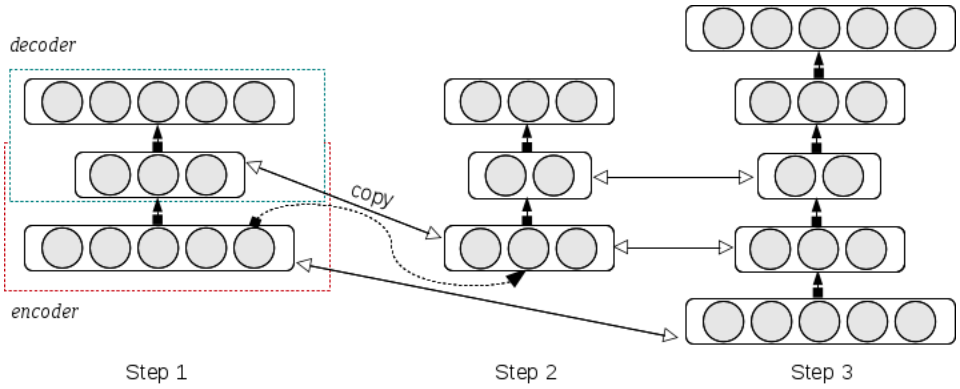

Figure 1: Training description. The first step consists in optimizing a two-layer autoencoders. A clean output is then retrieved. It is used to train another two-layer autoencoder. The final autoencoder is built by mixing the outer and inner layers. The network is finally fine-tuned by retraining the full autoencoder. This process can be recursively applied for bigger autoencoders. The third step differs from classic SDAE where the decoder is usually replaced by a classifier [18].

reverse order. Weights are initialized using the fan-in rule $\mathbf{W_{ij}} \sim \left[-\frac{1}{\sqrt{n}}, -\frac{1}{\sqrt{n}}\right]$ [9]. Transfer functions are hyperbolic tangents. The neural network is optimized with stochastic back-propagation on minibatch of size twenty. The momentum is set to 0.8. Even if DAE loss and sparsity already entail a strong regularization, a weight decay is required. The training is done using stacked autoencoders as described in the figure 1. The other hyperparameters of the training are provided in the table1 to reproduce the experiments.

hyperparameters	Outer Layers	Inner Layers	4-layers SDAE
learning rates*	0.03	1e-5	0.004
learning decay	0.1	0.1	0.2
Weight decay	0.03	0.2	0.03
α	1.	1	1.2
β	0.8	1	0.8
Gaussian noise	0	0.8 (std:0.02)	0
Masking Noise	0.15	0	0.15
Salt-and-Pepper	0.05	0.1	0.05

Table 1: Hyperparameters for training. The following parameters are set empirically to provide good results. The same parameters are used for both autoencoders for the movieLens and jester dataset. The layer ratio for the Vencoder of jester are respectively $1/1000^{th}$ and $1/1500^{th}$ of the input size.

3.2 Source code

Torch [4] is a powerful framework written in Lua to quickly prototype Deep Neural Networks. It is a widely used (Facebook, Deep Mind, Google) industry standard. However, Torch lacked some important tools to deal with sparse inputs. Thus, we developed several new modules to deal with DAE loss, sparse DAE loss and sparse inputs. They can easily be plugged into existing code. Implementing our methods can now be done in a half-day work. An out-of-the-box tutorial is also available to directly run the experiments. The code is freely available on Github and Luarocks [1].

[1]To install the module on torch, type *luarocks install nnsparse*

Dataset	MovieLens-1M	Jester
SVD (gradient)	0.852 ± 0.003	4.117 ± 0.04
ALS-WR	0.850 ± 0.004	4.108 ± 0.02
NL-PMF	0.879 ± 0.008	n/a
U-autorec	0.874 ± 0.003	n/a
V-autorec	$\mathbf{0.831 \pm 0.003}$	n/a
Uencoders	0.858 ± 0.003	$\mathbf{4.107 \pm 0.03}$
Vencoders	0.837 ± 0.004	5.001 ± 0.10

Table 2: RMSE Error with a training/testing set of 0.9/0.1 with cross-validation. The best ALS-WR results are obtained with a rank of 20 and λ_{reg} of 0.03. SVD is performed with a rank of 15, a learning rate of 0.02 and λ_{reg} of 0.02. Final MAE for Uencoders are Vencoders on the movieLens dataset are respectively 0.676 ± 0.02 and 0.656 ± 0.02.

4 Experimental results

4.1 Data

Dataset Two dataset are used to perform collaborative filtering. The Jester Joke dataset provides 4.1 million continuous ratings (-10.00 to +10.00) of 100 jokes from 73,496 users. The MovieLens-1M dataset provides 1 million discrete ratings (1 to 5) from 6 thousand users on 4 thousand movies.

Preprocessing The full dataset is considered and the ratings are normalized from -1 to 1. We split it into random 90%10% train-test dataset. Inputs are then unbiased before the training process. Given that $\bar{r}$ is the mean over the training set b_{u_i} is the mean of the i^{th} user and b_{v_i} is the mean of the v^{th} user, we apply $\hat{r}_{ij,unbiased} = \hat{r}_{ui} + \bar{r} - b_{u_i} - b_{v_j}$ for SVD/ALS algorithms, $\hat{r}_{ij,unbiased} = \hat{r}_{ui} - b_{u_i}$ for Uencoders, $\hat{r}_{ij,unbiased} = \hat{r}_{ui} - b_{v_i}$ for Vencoders.

Postprocessing The bias computed on the training set was added back while evaluating the final RMSE.

4.2 Results

The most important result is that autoencoders have excellent performance in our experiments, and are competitive compared to state-of-the-art methods. This is an improvement regarding previous work in neural networks. Indeed, Salakhutdinov et al. [14] faced important overfitting issues while turning a RBM into auto-encoders.

The second important point is that Uencoders and Vencoders may have different results regarding the dataset. Vencoders performs poorly on jester while they are excellent on movieLens. More research must be done to check whether Uencoders and Vencoders have similar errors. Indeed, their representation spaces, as described above, differs a lot.

We observe that 4-layer autoencoders return slightly better scores than 2-layer autoencoders. More importantly, they are far more robust to a change in hyperparameters. Even for a bad choice of hyperparameters, increasing the number of layers eventually provide excellent results.

As shown in figure 2, we observe that DAE loss can speed up the learning process. However, the hyperparameters α and β must be well-balanced. If the reconstruction error (β) is ignored, the prediction error cannot fully compensate the loss of information. From our experience, the ratio $\nu.\alpha/(1-\nu)\beta = 1/3$, where ν is the corruption ratio, is a good equilibrium. Furthermore, the DAE loss also improve the final RMSE. For instance, the Uencoders return better results when the inputs are corrupted.

5 Discussion

Autoencoders face some limitations that are worth to mention. Compare to ALS-WR or SVD, they are slow to train and less scalable. The training process also requires a high

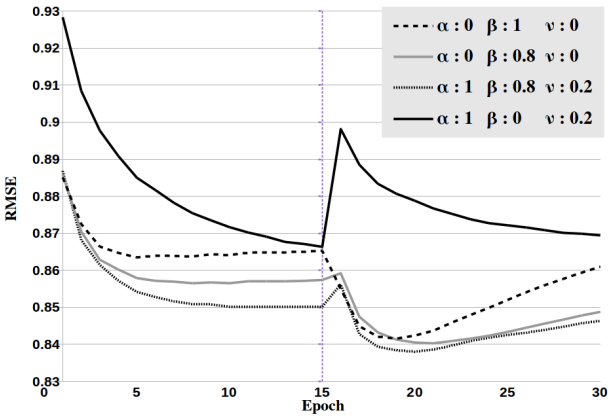

Figure 2: RMSE upon epoches. α, β are respectively the hyperparameters for the prediction and reconstruction error. ν is the corruption ratio. It is composed of 75% of masking noise and 25% of SaltAndPepper noise. The left-part of the plot is the first training step corresponding to a 2-layer autoencoders, the right-part of the plot is the third training step corresponding to the stacked 4-layer autoencoder.

number of hyperparameters which makes them difficult to tune finely. Whenever new items occurs, Vencoders need to be retrained since it changes the size of the input/ouput layer. Yet, Vencoders immediately provide additional estimates for new users. The situation is reversed for Uencoders.

While studying Restricted Boltzmann Machine [14], the authors pointed out that the errors made by RBM were significantly different from the errors made by SVD. We would like to investigate if the same behavior occurs with autoencoders. In the same way, Uencoders and Vencoders may also return different errors since they learn two different user/item representations. It may be worth finding a suitable network to mix both encoders. Autoencoders can also be used as a pre-initialization step for more complex networks.

In ALS-WR [20], Tikhonov regularization prevents the algorithm from overfitting the data. It differs from classic L_2 regularization since it takes into account the sparsity of the samples. We tried this regularization on neural networks but it did not work. However, we strongly assume that regularizing the backpropagation by using the density of the output may lead to better results.

Finally, other metrics exist to estimate the quality of collaborative filtering to fit real-world constraints. Normalized Discounted Cumulative Gain (NDCG) or Fscore are often preferred to RMSE and should be benchmarked.

6 Conclusion

We introduce a neural network architecture that performs as well as the best algorithms in collaborative filtering. We then point out the complex relationship which lies between users/items' representations that are learned. We describe the autoencoders architecture. We then explain how to add additional constraints on top of it to make the network handle sparse inputs. We detail the full training process and we provide the source code for replicating the results. We also provide Torch modules to allow other teams to build complex network using sparse inputs. Finally, we detail the main limitations of the algorithm and we suggest some leads requiring further investigations.

Acknowledgements

The authors would like to acknowledge the stimulating environment provided by SequeL research group, Inria and CRIStAL. This work was supported by French Ministry of Higher Education and Research, by CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020, and by FUI Hermès. Experiments were carried out using Grid'5000 tested, supported by Inria, CNRS, RENATER and several universities as well as other organizations.

References

[1] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[2] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. *Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005.

[3] Dumitru Erhan, Aaron Courville, and Pascal Vincent. Why Does Unsupervised Pre-training Help Deep Learning ? *Journal of Machine Learning Research*, 11:625–660, 2010.

[4] Torch framework. http://torch.ch/.

[5] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feed-forward neural networks. *Aistats*, 9:249–256, 2010.

[6] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):42–49, 2009.

[7] Mark A Kramer. Autoassociative neural networks. *Computers & Chemical Engineering*, 16(4):313–328, 1992.

[8] Neil D. Lawrence and Raquel Urtasun. Non-linear matrix factorization with gaussian processes. *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 601–608, 2009.

[9] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient BackProp. *Neural Networks: Tricks of the Trade*, 1524:9–50, 1998.

[10] Joonseok Lee, Seungyeon Kim, Guy Lebanon, and Yoram Singer. Local Low-Rank Matrix Approximation. *Proceedings of The 30th International Conference on Machine Learning*, 28:82–90, 2013.

[11] Joonseok Lee, Mingxuan Sun, and Guy Lebanon. A Comparative Study of Collaborative Filtering Algorithms. pages 1–27, 2012.

[12] Vladimiro Miranda, Jakov Krstulovic, Hrvoje Keko, Cristiano Moreira, and Jorge Pereira. Reconstructing Missing Data in State Estimation With Autoencoders. *IEEE Transactions on Power Systems*, 27(2):604–611, 2012.

[13] Steffen Rendle. Factorization machines. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 995–1000, 2010.

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

[15] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. AutoRec : Autoencoders Meet Collaborative Filtering. *Www*, pages 111–112, 2015.

[16] Volker Tresp, Subutai Ahmad, and Ralph Neuneier. Training Neural Networks with Deficient Data. *Advances in Neural Information Processing Systems 6*, pages 128–135, 1994.

[17] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[18] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, 11(3):3371–3408, 2010.

[19] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative Deep Learning for Recommender Systems. 2014.

[20] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. *Algorithmic Aspects in Information and Management*, pages 337–348, 2008.