Don't Stop the Music: Playlist Continuation with Autoencoders

Kevin Wu
Columbia University
New York, New York
kjw2157@columbia.edu

Vishal Anand
Columbia University
New York, New York
va2361@columbia.edu

Kirill Sydykov
Columbia University
New York, New York
ks3515@columbia.edu

ABSTRACT

The rising popularity of music streaming services has generated an ever-increasing amount of data on musical tracks and listeners' tastes. In particular, the 2018 RecSys Challenge features a newly-released dataset by Spotify containing one million playlists created by users of the online music streaming service. In this project, we explore the use of autoencoder methods for learning a low-dimensional representation of the playlist-song space, demonstrate the robustness of autoencoder methods to data sparsity, and compare the results achieved by autoencoders against a matrix factorization baseline.

1 INTRODUCTION

The use of advanced recommendation systems is a field of machine learning with wide-ranging applications for developing personalized user experiences in domains such as advertising or entertainment. Much of the research in the field of recommendation has focused on the use of unsupervised methods to learn low-dimensional representations of users' preferences over a set of items. In this paper, we build on previous work applying unsupervised methods from deep learning to recommendation, most notably the 2018 paper "Variational autoencoders for collaborative filtering" by Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara [7]. We show that even on extremely sparse data, deep autoencoder methods can provide superior performance as measured by recall and normalized discounted cumulative gain (NDCG).

2 RELATED WORK

In the framework of recommender systems, the underlying objective is to learn user preferences through *implicit feedback*, in which users reveal their preferences not explicitly but through past behavior (i.e. through clicks or views as opposed to ratings). By aggregating implicit feedback across users, one may make inferences on users' preferences on unseen items, an approach known generally as collaborative filtering. Early approaches to collaborative filtering leverage techniques such as singular value decomposition (SVD), Latent Dirichlet Allocation (LDA), and probabilistic Latent Semantic Analysis (pLSA) to learn a set of latent factors underlying the observed data [1, 3, 4].

The recent resurgence of interest in deep neural networks has led to the application of deep learning techniques to problems in the recommendation literature. In one approach, a non-linear generalization of the matrix factorization (SVD) method for collaborative filtering known as Neural Collaborative Filtering, user and item feedback vectors are mapped onto latent embedding vectors which are then passed through a multi-layer perceptron [2].

The other approach to neural-network based recommendation makes use of autoencoders from the unsupervised learning domain. The Collaborative Denoising Autoencoder method uses the denoising autoencoder method to learn latent representations, with Gaussian and logistic log-likelihood functions as the objective function for the neural network [9]. Recent work has extended the use of deep unsupervised methods to include both denoising and variational autoencoders, and demonstrated the superior performance of autoencoders trained with multinomial log-likelihood objective functions [7]. Our work closely follows that of Liang et al., extending their empirical evaluation to the newly-released Spotify playlist dataset, and examining the robustness of autoencoder-based methods to highly sparse datasets.

3 METHODS

In this section, we briefly describe the machine learning methods used in this study, including denoising and variational autoencoders as well as the matrix factorization technique used as a baseline for the recommendation task.

3.1 Autoencoders

Autoencoders are a broad class of neural networks used for unsupervised learning of non-linear latent representations. In the context of recommendation, given a set of N users and M items, the data can be described as a matrix $\mathbf{X} \in \{\mathbf{0}, \mathbf{1}\}^{\mathbf{N}\times\mathbf{M}}$, where user i's click history is described by the M-dimensional binary row vector x_i.

3.2 Denoising Autoencoders and Mult-DAE

The first type of autoencoder we will describe here is the denoising autoencoder [8]. The denoising autoencoder (DAE) takes as input a corrupted, or noisy, version of the original data. We call this modified input $\tilde{\mathbf{X}}$, with the corrupted version of user i's click history denoted as $\tilde{x}_i$.

In the first step of the DAE, the input is passed through a feed-forward neural network g with parameters ϕ (the encoder). The output of this neural network is a matrix $\mathbf{Z} \in \mathbb{R}^{\mathbf{N}\times\mathbf{K}}$, where $\mathbf{Z}$ is a low-dimensional representation of $\mathbf{X}$ with rank K.

$$z_i = g_\phi(\tilde{x}_i) \tag{1}$$

Next, the latent representation $\mathbf{Z}$ is passed through a second feed-forward network f parametrized by θ (the decoder). The output of this second network is a matrix $\mathbf{X}'$ in the original dimensions of the input.

For the recommendation setting, Liang et al. [7] introduce an additional softmax operation over the rows of $\mathbf{X}'$, and the final output of their generative model is a multinomial distribution π_i for each user i from which the user's clicks are drawn.

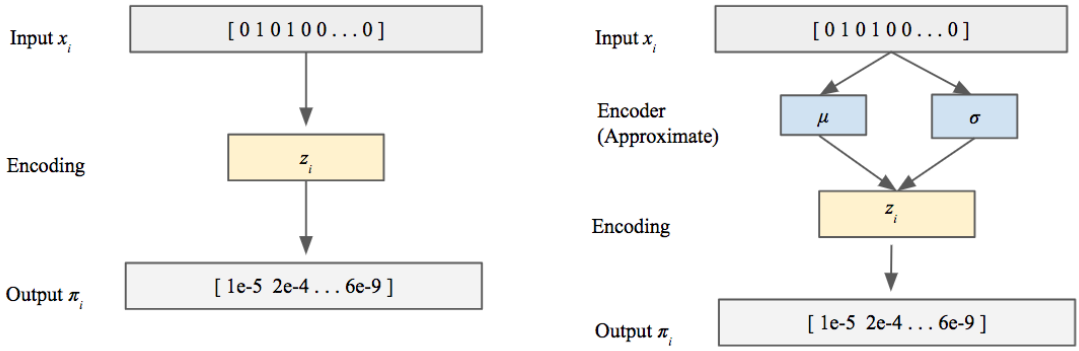

Figure 1: High-level overview of information flows through denoising (left) and variational autoencoders (right). Both Mult-DAE and Mult-VAE, pictured above, take as input a vector of click histories and output a probability distribution over items.

$$\pi_{i,j} \propto \exp(f_\theta(z_i)_j) \tag{2}$$

The encoder and decoder networks are trained in a single forward-backward pass, using as the objective function the negative multinomial log-likelihood for user i:

$$J(\phi, \theta) = -\sum_{i \in N} \sum_{j \in M} x_{i,j} \log \pi_{i,j} \tag{3}$$

Liang et al. call such a denoising autoencoder trained using a multinomial log-likelihood as Mult-DAE.

3.3 Variational Autoencoders and Mult-VAE

Since their original publication, variational autoencoders (VAEs) have quickly gained popularity as a way of learning a low-dimensional and probabilistic latent space over data [6]. The key difference between the VAE setting and vanilla and denoising autoencoders is that in the VAE setting, the latent representations z_i are sampled from some prior distribution, $p(z_i)$. With this notation, the encoder can be described as the posterior probability of z_i given an observation x_i, $p(z_i|x_i)$, while the decoder can be described as $p(x_i|z_i)$. According to Bayes' Rule, we can rewrite the posterior as

$$p(z_i|x_i) = \frac{p(x_i|z_i)p(z_i)}{\int p(x_i|z_i)p(z_i)dz_i} \tag{4}$$

The integral, however, includes the decoder neural network $p(x_i|z_i)$ and is intractable.

To get around this, we approximate the posterior using $q_\phi(z_i|x_i)$, a distribution whose integral is tractable and whose parameters are learned in the VAE.

With this new learned posterior q_ϕ, we can show that the likelihood of the data over the original posterior p_θ is lower bounded by:

$$\mathbb{E}_{q_\phi(z|x_i)}[\log p_\theta(x_i|z_i)] - KL(q_\phi(z|x_i) \,||\, p(z_i))] \tag{5}$$

Given this approximate posterior, the new objective for user i is to maximize the lower bound of the log likelihood for x_i, as measured by the KL-divergence between $q_\phi(z_i|x_i)$ and $p(z_i)$.

$$J(\phi, \theta) = -\sum_{i \in N} \sum_{j \in M} \log p_\theta(x_i|z_i)\, d\phi - \beta KL(q_\phi(z|x_i) \,||\, p(z_i))] \tag{6}$$

Just as in the DAE case, Liang et al. substitute multinomial log-loss for the standard binomial or Gaussian log-likelihood function in the last layer, which they refer to as Mult-VAE.

3.4 Collaborative Filtering (Baseline)

The baseline matrix factorization approach we use to generate recommendations is as follows. Given a set of N users and M items, and a binary click matrix $R \in \{0, 1\}^{NxM}$, we approximate R as the product of separate low-rank user and item matrices $V \in \mathbb{R}^{NxK}$ and $W \in \mathbb{R}^{MxK}$.

Given Ω, a set of (i, j) pairs indicating a "click" for user i and item j, we minimize the reconstruction error between the true click history and the reconstructed click history over Ω from VW^T. Adding a regularization parameter on the user and item matrices, we obtain the following loss function for matrix factorization:

$$\min_{V,\, W} \left\{ \sum_{(i,\, j) \in \Omega} [R_{i,\, j} - (VW^T)_{i,\, j}]^2 + \lambda(||V||_2 + ||W||_2) \right\} \tag{7}$$

This objective function is non-convex in V and W and it's minimum can be approximated using variants of stochastic gradient descent.

3.4.1 Alternating Least Squares. The Alternating Least Squares (ALS) technique is built on the observation that holding either V or W constant converts the above minimization problem into a Ridge Regression. By alternating between solving for W and V holding the other matrix constant, at each iteration we can calculate the following closed-form solutions for v_i and w_j for all i, j as the following:

$$v_i = (W^T W + \lambda I)^{-1} W^T R_i \quad \forall i \tag{8}$$

$$w_j = (V^T V + \lambda I)^{-1} V^T R_j \quad \forall j \tag{9}$$

The algorithm terminates upon the convergence of V and W between iterations within some small ϵ.

4 EXPERIMENT: MILLION PLAYLIST DATASET

4.1 Data

Released in January 2018, Spotify's Million Playlist Dataset contains one million playlists created by users of Spotify's music streaming service.[1] For each playlist, the dataset contains the title as well as a list of tracks and their order of appearance in the playlist. Additionally, each track contains information such as song title, artist name, album name, and duration.

```
{
  "pos": 5,
  "artist_name": "Guns N' Roses",
  "track_uri": "spotify:track:7
      o2CTH4ctstm8TNelqjb51",
  "artist_uri": "spotify:artist:3
      qm84nBOXUEQ2vnTfUTTFC",
  "track_name": "Sweet Child O' Mine",
  "album_uri": "spotify:album:3
      I9Z1nDCL4E0cP62flcbI5",
  "duration_ms": 356080,
  "album_name": "Appetite For Destruction"
}
```

4.2 Preprocessing

The original dataset contains one million unique playlists spanning 2,262,292 unique tracks. Given the relatively low barrier to entry for an artist to have their music uploaded to the platform, we wanted to include only tracks in our recommendation system that are popular to at least a certain degree. We also wanted to improve the quality of our recommendations by reducing the sparsity of the dataset.

To accomplish this, we passed the dataset through a two-stage preprocessing pipeline in which we first filtered out tracks appearing in fewer than $p_{\min}$ playlists, and then filtered out playlists with fewer than $t_{\min}$ tracks (from the remaining set of tracks).

We created two datasets in this way using different thresholds for the track and playlist filters, which we call MPD-compact and MPD-sparse. Details of the two datasets are given in Table 1, and the parameters $p_{\min}$ and $t_{\min}$ used for them are given in Table 2.

[1] Million Playlist Dataset, official website hosted at https://recsys-challenge.spotify.com/

	MSD[2]	MPD-compact	MPD-sparse	MPD-all
# playlists	571,355	373,740	919,695	1,000,000
# tracks	41,140	69,675	190,897	2,262,292
# interactions	34M	39M	59M	66M
sparsity	0.143%	0.143%	0.034%	.003%

Table 1: Million Playlists Datasets Comparison

	MPD-compact	MPD-sparse
$p_{\min}$	100	25
$t_{\min}$	50	10

Table 2: MPD-compact and MPD-sparse construction parameters

4.3 Train, Validation, Test Split

For both MPD-compact and MPD-sparse, we reserve 10,000 and 50,000 playlists respectively as validation and test sets. For each validation and test set playlist, we then used 80% of the click history as query data and the remaining 20% as "held-out" click data against which we evaluate our predictions.

4.4 Implementation

Neural networks were implemented in Keras (for Mult-DAE) and Tensorflow (for Mult-VAE), following the example in [7]. For Mult-DAE no hidden layer was used; both encoder and decoder components were single-layer perceptrons with a tanh non-linearity at the latent encoding layer and a softmax non-linearity at the output layer. The Mult-VAE architecture included a hidden layer of size 600 in both the encoder and decoder.[3] For regularization, again following the example of [7], dropout was applied at the input layer with probability 0.5, and weight decay of 0.01 was used for Mult-DAE. In Mult-VAE, the coefficient for the KL-divergence term (β) was linearly annealed over the first 200,000 gradient updates.

For optimization, Adam [5] was used with a learning rate of 10^{-3} and a batch size of 500. Finally, models were trained on NVIDIA Tesla K80 GPUs hosted on Google Cloud Platform.

4.5 Evaluation Metrics

After performing the training, validation, and test splits described in the previous section, we evaluate the predictions from autoencoder and collaborative filtering models on the ground truth answers from the held-out clicks using the following two metrics.

4.5.1 Recall. Recall measures the fraction of relevant tracks, i.e. tracks in the original held-out part of the playlist, that are retrieved by a query to a model.

$$\text{recall} = \frac{|\{\text{relevant tracks} \cap \text{retrieved tracks}\}|}{|\{\text{relevant tracks}\}|} \tag{10}$$

Specifically, we calculate recall@k where k indicates the maximum number of top tracks considered for that playlist.

[3] For Mult-DAE, hidden layers of size 600 were introduced for the encoder and decoder, but there was no resulting improvement from a deeper model.

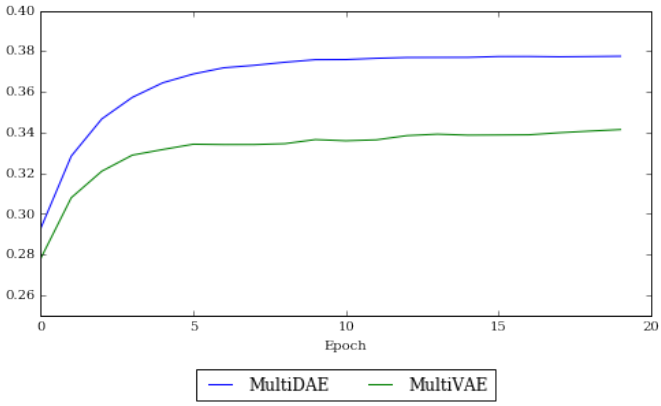

Figure 2: NDCG@100 of Mult-DAE and Mult-VAE on validation data

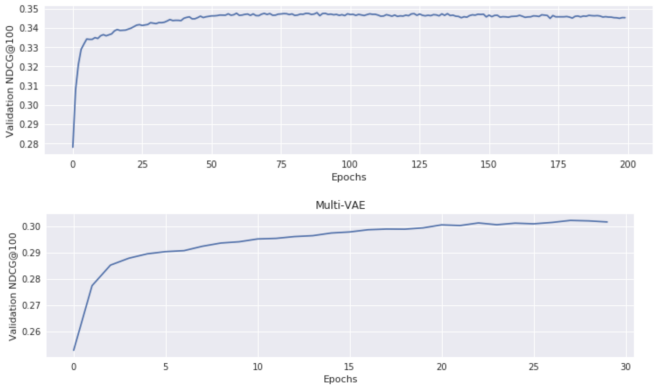

Figure 3: Mult-VAE convergence on dense (top) and sparse (bottom) datasets. Mult-VAE validation performance on the sparse dataset follows a similar trajectory as validation performance on the compact dataset.

4.5.2 NDCG. Normalized discounted cumulative gain measures the degree to which highly relevant tracks appear earlier in the playlist and less relevant — later.

$$\text{NDGC}_k = \frac{\text{DCG}_k}{\text{IDCG}_k} \tag{11}$$

where DCG for k top results (DCG_k) is

$$\text{DCG}_k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)} \tag{12}$$

with rel_i being the relevance of the track at position i (in our case rel is binary, i.e. $rel \in \{0, 1\}$) and logarithmic discount depending on the position i, and where ideal DCG_k (IDCG_k) is

$$\text{IDCG}_k = \sum_{i=1}^{|REL_k|} \frac{rel_i}{\log_2(i+1)} \tag{13}$$

with REL_k being the list of at most k relevant tracks in their original order in the playlist. Specifically, we use NDCG_{100} or NDGC@100.

5 RESULTS

In this section we discuss the performance of multinomial-likelihood autoencoder models on Spotify playlist data. Numerical performance on MPD-compact and MPD-sparse datasets is shown in Table 3 and Table 4, respectively. An example of an actual set of playlist recommendations (generated using Mult-DAE on heldout data from MPD-sparse) is shown in Figure 4.

5.1 Mult-DAE and Mult-VAE Performance

In our experiments on the Spotify playlist dataset, we compared the performance of both Mult-DAE and Mult-VAE against a baseline, unweighted matrix factorization model trained using Alternating Least Squares. For the matrix factorization method, we used a low-rank representation of rank 200, the same dimensionality as the latent representation learned by Mult-DAE and Mult-VAE.

We find that for both versions of the Spotify dataset, Mult-DAE outperformed matrix factorization by a large margin on nearly all metrics. Following the example of [7], we display Recall@20, Recall@50, and NDCG@100, but we note that this outperformance was consistent across recall and NDCG for all top-N metrics. In addition, we found that the performance of Mult-DAE and Mult-VAE methods on MPD-compact was similar to the performance reported by [7] on a dataset of comparable size and sparsity (the Million Song Dataset).

As seen in figure 3 Mult-VAE model performance was slow to converge on both datasets. On average, the models converged around 60 epochs. Due to resource constraints, Mult-VAE was trained for only 30 epochs on the sparser dataset (on the dense dataset, each epoch took approximately 7 minutes, while on the sparse dataset, each epoch took nearly 60 minutes). For further work, we could experiment with training Mult-VAE on MPD-sparse for longer time periods (potentially using more compute power) to compare the results with MPD-compact with similar epochs of training.

One difference to note between these results and those reported in [7] however, is the ranking of autoencoder performance; in previous experiments, Mult-VAE had either performed the same as Mult-DAE or better by a slight margin (< 1%). In our results, we found that the denoising autoencoder yielded better results on the validation and testing data. One possible reason for this result is that we did not do enough hyperparameter tuning, and that given further experiments on autoencoder depth and latent dimension size, the two models might yield similar results.

5.2 MPD-Compact vs. MPD-Sparse

While initially we made use of a fairly aggressive filtering step to generate a dataset of comparable size and sparsity as existing implicit feedback datasets (specifically, Netflix, MovieLens-20M, and MSD), the strong results on MPD-compact spurred us to try our experiments on a sparser dataset that included more of the playlists and tracks in the original Spotify dataset. We note that there are two possible effects of this expanded dataset. The first possibility is that the increased noise generated from highly infrequently occurring tracks or small playlists would make it more difficult for

	Recall@20	Recall@50	NDCG@100
Mult-DAE	**0.250**	**0.370**	**0.376**
Mult-VAE	0.232	0.327	0.346
MF	0.141	0.233	0.232

Table 3: Performance on MPD-compact

	Recall@20	Recall@50	NDCG@100
Mult-DAE	**0.376**	**0.577**	**0.406**
Mult-VAE	0.243	0.342	0.302
MF	0.202	0.347	0.235

Table 4: Performance on MPD-sparse

collaborative filtering methods to generalize the data into a lower-dimensional space. The second is that by increasing the number of users/playlists in the training data, the sparser dataset would in fact improve out-of-sample performance.

In comparing Tables 3 and 4, we can see both of these effects at work. Keeping the neural network architecture, hyperparameters, and training constant, we can see that the best-performing Mult-DAE model demonstrated significantly better results on test users from MPD-sparse; in this case, the benefits of the increase in training data seemed to outweigh the difficulties imposed by sparser data.

The results were mixed for the other two models, however. Recall@20 and Recall@50 metrics showed a marginal improvement for the Mult-VAE model when training and testing on the sparser dataset, but the NDCG@100 decreased by several percentage points. Matrix factorization showed a significant improvement in the top-20 and top-50 metrics, but almost no improvement in NDCG@100. In general, it seems that most of the benefit from the larger number of playlists in MPD-sparse is distributed within the top-ranking predictions, while the additional sparsity may be generating more of an adverse effect for predictions below rank 50. Again, it is possible that the Mult-VAE model may see additional improvement on MPD-sparse with further tuning of the hyperparameters and architecture, but the effect of increased sparsity on the three evaluation metrics is fairly consistent across all three models.

6 CONCLUSION

Non-linear representations of user behavior using deep neural networks is a promising new direction in the field of collaborative filtering. In this work, we demonstrate excellent performance of autoencoder networks trained with multinomial log-likelihood on Spotify playlist data from the 2018 RecSys Challenge. We compare autoencoder methods with a matrix factorization baseline and show the effect of sparsity on the model performance. In future work, we would like to continue to experiment with different neural network architectures, hyperparameters, and optimization strategies in order to boost performance. We would also like to explore enriching the implicit feedback dataset with metadata on users and tracks, in order to tackle the cold-start problem faced in part of the 2018 RecSys Challenge test set, and to improve the quality of playlist continuation recommendations generated by autoencoder models.

```
PLAYLIST TRACKS
"Erase Your Social" - Lil Uzi Vert
"La Gozadera" - Gente De Zona
"Hasta Que Se Seque el Malecón" - Jacob Forever
"Danza Kuduro" - Don Omar
"Backseat - feat. The Cataracs & Dev" - New Boyz
"Niña Bonita" - Chino & Nacho
"Sabor a Mí" - Eydie Gormé
"Untouchable" - Tritonal
"Defying Gravity (Glee Cast Season 5 Version)" - Glee Cast
"More Than Conquerors" - Rend Collective
[....]
TOP-RANKED NON-PLAYLIST TRACKS
*"Motivos" - Grupo Mojado
*"Sometime" - DIIV
*"Appalachian Mountain Girl" - Alan Jackson
"All of Me" - The Piano Guys
"Close Your Eyes (And Count to Fuck)" - Run The Jewels
*"Are You My Woman? (Tell Me So)" - The Chi-Lites
"Bonita Applebum" - A Tribe Called Quest
*"Cosmic Love" - Florence + The Machine
"Falling (Whethan Redo)" - Opia
*"No One's Here To Sleep" - Naughty Boy
*"Esta Cabron" - Ñejo
"Hold On" - Colbie Caillat
"All I Need" - Felly
*"La Carcacha" - Selena
"A New Hope and End Credits" - John Williams
*"Sunrise" - Childish Gambino
*"Forward To Love" - Ziggy Marley
"Just Stay Here Tonight" - Augustana
"Leave (Get Out)" - JoJo
"Que linda es Lupe" - Los Tigrillos
[....]
```

Figure 4: Sample recommendations for a given playlist. Actual playlist tracks are shown above, followed by recommended tracks from a Mult-DAE model. Recommendations that match up with actual held-out tracks for the playlist are denoted by a (*).

REFERENCES

[1] David M. Blei, Andrew Y. Ng, Michael I. Jordan, and John Lafferty. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research* 3 (2003), 2003.

[2] Xiangnan He, Lizi Liao, and Hanwang Zhang. 2017. Neural Collaborative Filtering. (08 2017).

[3] Thomas Hofmann. 2004. Latent Semantic Models for Collaborative Filtering. *ACM Transactions on Information Systems* 22, 1 (January 2004), 89–115.

[4] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *In IEEE International Conference on Data Mining (ICDM 2008.* 263–272.

[5] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980

[6] Diederik P. Kingma and Max Welling. 2013. Auto-Encoding Variational Bayes. *CoRR* abs/1312.6114 (2013). arXiv:1312.6114 http://arxiv.org/abs/1312.6114

[7] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara. 2018. Variational Autoencoders for Collaborative Filtering. *ArXiv e-prints* (Feb. 2018). arXiv:stat.ML/1802.05814

[8] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research* 11 (2010), 3371–3408. http://portal.acm.org/citation.cfm?id=1953039

[9] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. 2016. Collaborative Denoising Auto-Encoders for Top-N Recommender Systems. In *WSDM*.